Fusion 2.4 Documentation

2017-12-07

Table of Contents

User Manual

Chapter 1. Fusion Installation and Upgrades

1.1. Installation

You can install Fusion:

- **On a single computer** – Install Fusion on a single computer for purposes of development, evaluations, or demonstrations.

  For information, see Install Fusion on a Single Computer.

- **On multiple servers** (as a *Fusion cluster*) – For a number of reasons such as performance, scaling, and availability, you *must* install Fusion on multiple servers for a production deployment.

  Detailed instructions have been added in the documentation for Fusion 3.1.

These procedures are for *initial* installations of Fusion.

1.2. Upgrades

If you already have Fusion installed and want to upgrade to a later release, see the upgrade documentation.

1.3. Related Topics

Additional documentation that pertains to installation includes:

- System Requirements – Hardware and software needed to run Fusion in both single-node (trial) and multi-node (production) deployments
- Components – An explanation of the components that comprise Fusion
- Troubleshoot When Installing Fusion – An explanation of how to troubleshoot difficulties that occur when installing or upgrading Fusion
- Directories and Logs – What’s in the Fusion home directory (`/path/to/fusion`), including default log file locations
- Default Ports – Default Fusion server ports and how to change them
- Checking System State – How to check the default Fusion distribution
- Integrating with existing Solr instances – How to use existing Solr instances to store Fusion collections

After installation, proceed to these topics:

- Security – How to secure a Fusion deployment
- Administration – Information about configuration and monitoring

1.4. System Requirements

Requirements for Fusion installation are detailed below.

Important	Lucidworks recommends *not* virus scanning the `fusion/data` folder. Virus scanning can cause slow performance, and it can cause downtime if it quarantines an index file identified as a possible virus.

1.4.1. Supported Operating Systems

Supported for production use of Fusion services

Operating system:

- Windows Server 2012, 2012 R2, and 2016 (x64 only)

Note	Windows systems must have the Microsoft Visual C++ 2010 SP1 package installed.

- Linux 64-bit (x86_64/amd64 only) with 2.6 or later kernel

Note	RedHat based Linux, including CentOS, must be 6.6.x or later, or else the bug fix https://rhn.redhat.com/errata/RHSA-2013-1605.html must have been applied. This fix remediates a RedHat bug that causes Fusion to hang. See http://www.infoq.com/news/2015/05/redhat-futex for more information.

JVM:

- Oracle JRE or JDK 1.8, 64-bit (x64 only)
- OpenJDK JRE or JDK 1.8, 64-bit (x64/x86_64 only)

Also supported for trial and development use of Fusion services

Operating system:

- Windows 7, 8, 8.1, and 10 64-bit (x64 only)
- Mac OS X 10.8 or later

1.4.2. Hardware requirements

Fusion and Solr nodes

Here are some *minimum* recommendations for different Fusion deployments. These are for the nodes that run Fusion (including ones that *also* run Solr and/or ZooKeeper, if those cluster arrangements are used), as well as for external (non-Fusion) nodes for SolrCloud nodes:

Deployment type	**Memory**	**CPU**
Development/Testing	16 GB	4 cores
Small Production*	32 GB	8 cores
Large Production**	64+ GB	8+ cores

- Small production environments will have 2+ nodes with these specs, on which both Fusion and Solr are installed. (3+ nodes are recommended.)
    - Large production environments will have 3+ nodes with these specs. Additional nodes are needed for an external SolrCloud cluster, if that is used. An external SolrCloud cluster is only necessary for Solr collections with very large numbers of documents (for example, 100 million documents or more), though a Fusion deployment can use it regardless. On these nodes, Fusion and Solr are installed if no external SolrCloud cluster is used. If an external SolrCloud cluster is used, these nodes might or might not have Solr installed.

ZooKeeper nodes

Here are some *minimum* recommendations for the nodes that run ZooKeeper. These nodes can be more lightweight than the nodes that run Fusion and/or Solr. With the exception of small development/testing deployments, ZooKeeper nodes should only run ZooKeeper (not Fusion or Solr). Because ZooKeeper synchronizes all it's operations to disk, we recommend using a disk with high throughput and low latency for your ZooKeeper nodes.

Deployment type	**Memory**	**CPU**
Development/Testing	4 GB	4 cores
Small Production	4+ GB	4+ cores
Large Production	4+ GB	4+ cores

Fusion components

You can run Fusion components on different nodes. Different Fusion components require different amounts of resources. Below are the *minimum* recommended memory requirements; consult Lucidworks for specific recommendations tailored to your unique use case, data load, and production needs.

Fusion component	***Minimum* RAM**
Connectors	2 GB
API service	1 GB
Fusion UI	512 MB
Solr	2 GB
Spark master	512 MB
Spark worker	1 GB

1.4.3. Java

Fusion is a Java-based application, and thus requires a pre-installed JDK.

Fusion runs on JDK 1.8.

Fusion’s JavaScript pipeline stages are application-specific custom JavaScript programs. The first time a pipeline is run, the JDK compiles the JavaScript using the Nashorn JavaScript engine.

Fusion scripts execute both the `java` and `javac` commands, which are usually aliases for the current Java installation. To check which version of Java is invoked, run the following commands from a shell or terminal window:

```
java -version
javac -version
echo $JAVA_HOME     // Unix
echo %JAVA_HOME%    // Windows
```

1.4.4. Cluster Requirements

Supported Solr and SolrCloud Versions

Fusion includes an embedded instance of Solr; see the release history to find out which Solr version is included in each Fusion release.

If your search requirements are *very large* (for example, 100 million documents or more), we recommend that you use an external SolrCloud cluster. (You can use an external SolrCloud cluster regardless.)

Optionally, you can use *both* embedded Solr instances *and* an external SolrCloud cluster. In this case, only store logs in the embedded Solr instances. Store data in the external SolrCloud cluster.

Solr 4.10 and higher are supported.

If you want to know whether other versions of Solr might be compatible with Fusion, you can ask Lucidworks.

If you use an external SolrCloud cluster, it has ZooKeeper bundled with it. However, Apache recommends that you not use the bundled ZooKeeper in production. Instead, create an external ZooKeeper cluster (external to both Fusion and SolrCloud) or use the ZooKeeper embedded with Fusion, depending on the circumstance. For more information, see Supported ZooKeeper Versions and Cluster Requirements.

Note	If you decide to set up an external SolrCloud cluster, check the requirements for this as well in the Solr documentation.

We strongly recommend that you use Network Time Protocol (NTP) on a SolrCloud cluster to ensure that nodes use synchronized time. While this is not strictly required, reasoning about log contents and database entries becomes impossible without it. Information and instructions on how to install and run NTP are available at www.ntp.org.

1.4.5. Recommended HTTP Clients

The Fusion API can be accessed from any HTTP client, and allows you to build user interfaces and applications that work with any browser. However, the Fusion Administration UI, Dashboards, and built-in Search UI are supported only with:

- Chrome latest version
- Firefox latest version and latest ESR
- Internet Explorer 11

1.5. Install Fusion on a Single Node

Note	These instructions are for an initial installation of Fusion on a single node (computer). To install Fusion on multiple nodes (a *Fusion cluster*), see Install a Fusion Cluster. If you already have a version of Fusion installed and want to upgrade it, see the Fusion upgrade instructions.

Out of the box, Fusion uses the instances of Solr, ZooKeeper, and Spark that are included in the Fusion distribution. See the release history to find out which versions of Solr, Spark, and ZooKeeper are included in each Fusion release.

To use Fusion with your existing Solr installations, see Integrating with existing Solr instances.

1.5.1. Ports

To run Fusion as a single-server installation, the following ports should be available and not used by other applications or services:

- 8764
- 8765
- 8983
- 8984

See Default Ports if you need to modify the default Fusion ports before starting the application.

1.5.2. Unix installation

Fusion for Unix is distributed as a gzipped tar file.

To install Fusion on Linux or Mac

1. Download the Fusion tar/zip file for the latest version of Fusion and move it to where you would like it to reside in your filesystem (if you would like to use Upstart for process management, you must install Fusion in `/opt/lucidworks`).
2. Become the user that will run Fusion.

Important	Do not run Fusion as the root user.

3. Change your working directory to the directory in which you placed the `fusion-version.x.tar.gz` file, for example:

   ```
   $ cd /opt/lucidworks
   ```

4. Unpack the archive with `tar -xf` (or `tar -xvf`), for example:

   ```
   $ tar -xf fusion-version.x.tar.gz
   ```

   The resulting directory is named `fusion`. You can rename this if you wish. This directory is considered your Fusion home directory. See Directories and Logs for the contents of the `fusion` directory.

Starting Fusion

All Fusion start scripts must be executed by a user who has permissions to read and write to the directories where Fusion is installed. These scripts don't need to be run as root (or sudo), nor should they be. Use a suitable ID, or create a new one, and then ensure that it owns the directory where Fusion resides, (e.g. `/opt/lucidworks`).

To start all required services

`./bin/fusion start`

For information about starting individual services, see Start and Stop Fusion.

Running Fusion In The Foreground

To run Fusion or any of its services in the foreground, use the `run` command-line argument in place of `start`.

Stopping Fusion

To stop Fusion or any of its services, use the `stop` command-line argument in place of `start`.

Ubuntu Upstart Scripts

Under Ubuntu 12.04 LTS or newer we support Upstart for process management. This requires Fusion to be installed in the `/opt/lucidworks/` directory.

To configure upstart, run the following commands:

```
$ cd init/upstart
$ sudo bash install.sh
```

If this complains with `no JAVA_HOME set`, replace `sudo` with `sudo -E`. Then you can use the `service` command to control the server:

```
$ sudo service fusion-solr start
$ sudo service fusion-api start
$ sudo service fusion-connectors start
$ sudo service fusion-ui start
```

and similarly use `stop` and `status`.

Logfiles for Fusion services are found in directories under `fusion/var/log`.

Upstart log files for each service are in the `/var/log/upstart` directory.

For convenience, you can start/stop all services with Upstart using:

```
$ sudo bash start.sh
$ sudo bash stop.sh
```

1.5.3. Windows installation

Fusion for Windows is distributed as a compressed zip file. To unpack the Fusion zip file on Windows, you can use a native compression utility or the freely available 7zip file archiver. Visit the 7zip download page for the latest version.

To install Fusion on Windows

1. Download the zip file for the latest version of Fusion and move it to where you would like Fusion to reside in your filesystem. It will appear as a compressed folder.
2. Unpack the archive. In most cases, you need only right-click and choose "Extract all…". If you don't see this option, check that you have permissions to extract folders on your system.

   The resulting directory has the same name as the base name of the zip file, such as `fusion-2.4.0`, and it contains another directory named `fusion`. This directory is considered your Fusion home directory. See Directories and Logs for the contents of the Fusion home directory.

Fusion releases earlier than 3.0 do not support installation as a Windows service.

1.5.4. Starting Fusion

All Fusion start scripts must be executed by a user who has permissions to read and write to the directories where Fusion is installed. Ensure that such a user owns the directory where Fusion resides.

To start all required services

- `bin\fusion.cmd start`
- `bin\start-services.cmd` (Start all Fusion services as Windows services)

For information about starting individual services, see Start and Stop Fusion.

1.5.5. Stopping Fusion

To stop Fusion or any of its services, use the `stop` command-line argument in place of `start`.

1.5.6. Installation with an existing Solr instance or cluster

Fusion supports Solr versions 4.4 and higher. Solr 4.6.0 or 4.7.0 are not supported, as they contain severe bugs that will impact the ability of Fusion to work with your Solr system.

If installing Fusion to work with an existing Solr instance, either in SolrCloud mode or standalone, you should install Fusion as described above. You should start each of the services as described above.

Once Fusion installation is complete, you can register your existing Solr installation with Fusion to be able to use the two systems together. For details on how to do that, see the section Search Clusters.

1.5.7. Troubleshooting

For information about problems you might encounter when installing Fusion, and solutions, see Troubleshoot When Installing Fusion.

1.5.8. Directories and Logs

Directories

The directory where the Fusion files go for a specific version of Fusion is the *Fusion home directory*. The Fusion home directory is the directory `fusion`.

The directories found in the Fusion home directory `fusion` are:

Name	Description
`apps`	Fusion components 3rd-party distributions used by Fusion, including jar files and plugins
`bin`	Master script to run Fusion, and per-component run scripts
`conf`	Configuration files for Fusion and ZooKeeper that contain parameters settings tuned for common use cases
`data`	Default location of data stores used by Fusion apps
`docs`	License information
`examples`	Fusion signals example
`init`	`systemd` and `upstart` scripts and configurations for Linux
`scripts`	Developer utilities, including diagnostic scripts, for Linux and Windows. See `scripts/diag/linux/README` and `scripts/diag/win64/README.txt` for details.
`var`	Logfiles and system files created by Fusion components, as well as `.pid` files for each running process

Logfiles

Logfiles are found in directories under `fusion/var/log`. Because the Fusion components run in separate JVMs, each component has its own set of logfiles and files that monitor all garbage-collection events for that process.

Name	Description
`api`	Fusion REST API services logging and error messages. This log shows the result of service requests submitted to the REST API directly via HTTP and indirectly via the Fusion UI.
`connectors`	Fusion connector services logging and error messages. Fusion index pipeline logging stages write to this file.
`solr`	Messages from Solr
`spark-master`	Spark-master logs
`spark-worker`	Spark-worker logs
`ui`	Fusion UI messages
`zookeeper`	ZooKeeper messages

Every component logs all messages to a logfile named <component>.log. For example, the full path to the logfile for the connectors services is:

fusion/var/log/connectors/connectors.log

In addition to component logfiles, every component maintains a set of garbage-collection logfiles which are used for resource tuning. The garbage-collection logfiles are named gc_<YYYYMMDD>_<PID>.log.<CT>. In addition, the current garbage-collection logfile has suffix .current.

The Fusion REST API, UI, connectors services, and Solr all run inside a Jetty server. The Jetty server logs are also written to that component's logfile directory. The Jetty server logs are named:

- jetty-YYYY_MM_DD.request.log
- jetty-YYYY_MM_DD.stderrout.log

Fusion uses the Apache Log4j 2 logging framework with Jetty to log each of the Fusion components. Logging is configured via an xml configuration file named log4j2.xml. Log levels, frequencies, and log rotation policy can be configured by changing these configuration files:

API service	fusion/conf/api-log4j2.xml
connectors	fusion/conf/connectors-log4j2.xml
Solr	fusion/conf/solr-log4j2.xml
Spark	fusion/conf/spark-driver-log4j2.xml fusion/conf/spark-master-agent-log4j2.xml fusion/conf/spark-master-log4j2.xml fusion/conf/spark-worker-agent-log4j2.xml fusion/conf/spark-worker-log4j2.xml
UI	fusion/conf/ui-log4j2.xml
ZooKeeper	fusion/conf/zk-log4j2.xml

The Log4j2 Configuration guide provides documentation and examples of all logging configuration options.

1.5.9. Checking System State

As described in the section Default Ports, Fusion runs several components as separate JVMs running on different ports. Each of the components is capable of reporting it’s status. The proxy component reports status for all of the other components.

Full System Check

To see if each component has been started, a simple API call to the proxy (running on port 8764 by default) will return the status of each component of the system.

```
curl http://localhost:8764/api
```

The response should look similar to the following. If 'ping' is true for each service, all of the system components are running.

```
{
    "version": "0.9.0-SNAPSHOT-jenkins.build.105+git.sha.b425e2a",
    "enabledRealms": [
        "native"
    ],
    "initMeta": {
        "version": "0.9.0-SNAPSHOT-jenkins.build.105+git.sha.b425e2a",
        "initializedAt": "2014-10-06T17:43:31Z",
        "createdAt": "2014-10-06T17:43:31Z"
    },
    "startTime": "2014-10-06T18:38:08Z",
    "status": {
        "connectors": {
            "ping": true
        },
        "apollo": {
            "ping": true
        },
        "apolloZk": {
            "ping": true
        },
        "db": {
            "ping": true
        }
    }
}
```

Solr Health Check

The Fusion UI and API services are not accessible if ZooKeeper and Solr are not in healthy state. A Solr health check can be performed with a ping request.

```
curl http://localhost:8983/solr/admin/ping
```

The response will be a standard Solr XML response, similar to the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
   <lst name="responseHeader">
      <int name="status">0</int>
      <int name="QTime">6</int>
      <lst name="params">
         <str name="df">text</str>
         <str name="echoParams">all</str>
         <str name="rows">10</str>
         <str name="echoParams">all</str>
         <str name="q">solrpingquery</str>
         <str name="distrib">false</str>
      </lst>
   </lst>
   <str name="status">OK</str>
</response>
```

The 'status' should be "OK" if Solr is running properly.

REST API Services Health Check

All of the Fusion API backend services (except Connectors and the UI) are started at port 8765 when the run.sh script is executed. The Fusion UI depends on all these services.

If all the services are started without any issues, then the below ping request should return response 'ok'.

```
 curl http://localhost:8765/api/v1
```

As an additional check, you can also query the `system/status` endpoint, which should return a response 'started'.

```
curl http://localhost:8765/api/v1/system/status
```

The response would look like:

```
{
  "status" : "started"
}
```

Connectors Health Check

The Connectors health check can be performed by a ping request to port 8984. Similar to the previous ping request, the returned response is 'ok' if the service starts successfully.

```
 curl http://localhost:8984/connectors/v1
```

As an additional check, you can also query the `system/status` endpoint, which should return a response 'started'.

```
curl http://localhost:8984/connectors/v1/system/status
```

The response would look like:

```
{
  "status" : "started"
}
```

1.6. Upgrade Fusion

After you have a Fusion-based search application running, at some point it might be necessary to upgrade to a later version of Fusion. Your goal is to transfer over all of your data together with all configurations and customizations necessary to support your applications.

In this topic, we:

- Provide links to upgrade instructions for specific versions of Fusion.
- Give a general overview of the procedure for upgrading.
- Present notes about version incompatibilities.

Tip	See the release history to find out what’s new, including which versions of Solr, Spark, and ZooKeeper are bundled with each Fusion release.

1.6.1. Per-version instruction sets

To upgrade to a later version of Fusion from an existing installation requires transferring over all configurations and data from your existing Fusion installation to the new version.

Perform the steps in the appropriate section:

- Upgrade Fusion 2.1 to Fusion 2.4 – Instructions for upgrading Fusion versions 2.1.x and 2.2.1 to Fusion 2.4.
- Upgrade Fusion 1.2 to Fusion 2.4 – Instructions for upgrading Fusion versions 1.2.3 through 1.2.8 and version 1.4.0 to the latest version of Fusion 2.4.
- Upgrade Fusion 2.4 to Fusion 3.0 – Instructions for upgrading Fusion version 2.4.x to Fusion 3.0.0.
- Upgrade Fusion 3.0.0 to 3.0.1 – Instructions for upgrading Fusion version 3.0.0 to 3.0.1.
- Upgrade Fusion 3.x to a more recent release – Instructions for upgrading Fusion version 3.x to a more recent release. This procedure uses a migrator that migrates all Fusion objects in a seamless upgrade.

1.6.2. Upgrade overview: migrating data, configurations, and customizations

The Upgrade Process

The upgrade process leaves the current Fusion deployment in place while a new Fusion deployment is installed and configured. All of the upgrade operations copy information from the current Fusion over to the new Fusion. This provides a rollback option should the upgrade procedure encounter problems.

The current Fusion configurations must remain as-is during the upgrade process. In order to capture indexing job history, no indexing jobs should be running. If the new Fusion installation is being installed onto the same server that the current Fusion installation is running on, you must either run only one version at a time or else change the Fusion component server ports so that all components are using unique ports for both the current and new versions.

ZooKeeper

Migration consists of the following steps:

- Copy the ZooKeeper data nodes which contain Fusion configuration information from the FUSION-CURRENT

ZooKeeper instance to the FUSION-NEW ZooKeeper instance

- Rewrite Fusion datasource and pipeline configurations, working against the FUSION-NEW ZooKeeper instance

Important	Because some Fusion configurations have changed, ZooKeeper data must be rewritten accordingly using scripts available in the public GitHub repository: https://github.com/LucidWorks/fusion-upgrade-scripts.

Solr

Fusion-based search applications store your data in Solr. If your data is stored in an external Solr cluster, and if you aren't upgrading your Solr cluster, then you don't need to migrate your Solr data at all; you just need to configure the new Fusion deployment to use your external Solr cluster.

Fusion uses Solr as a data store for server logs, search logs, and binary components such as jar files and compiled models used by Fusion pipelines. The Fusion distribution includes a complete Solr server and is configured to use this embedded Solr instance by default. If the current Fusion deployment uses the embedded Solr for its system collections, then you must copy all of these collections over to the embedded Solr instance included with the new Fusion distribution.

Connector services data: crawldb, database drivers

The directory `fusion/data/connectors/lucid.jdbc` contains third party JDBC driver files that have been registered with Fusion in order to run the JDBC connector.

The directory `fusion/data/connectors/crawldb` is managed by Fusion's connector service. Fusion datasources that walk over websites, filesystems, or similar repository use the crawldb to store information about files visited during the crawl; this allows incremental updates and avoids data re-indexing. In current versions of Fusion, the default location is the Fusion directory `fusion/data/connectors/crawldb`.

Important	The crawldb data format was changed, therefore for upgrades from Fusion 1.2.x to the latest Fusion 2.1, these must be processed using a reformatting program that is available in the public upgrade scripts repository: https://github.com/LucidWorks/fusion-upgrade-scripts.

Pipeline services data: models used by pipeline stages

The directory `fusion/data/connectors/lucid.jdbc` contains third party JDBC driver files that have been registered with Fusion in order to run the JDBC connector.

Customized settings for Fusion run commands and configuration scripts

The scripts used to start, stop, and restart Fusion and its components are found in the top-level "bin" directory of the Fusion distribution. As of Fusion 2.0, the configuration scripts previously found in the "bin" directory were put in their own top-level directory called "conf".

Important	If you have customized the settings in these files and wish to carry these settings over to the new Fusion deployment the only way to do so is to edit the command and configuration scripts in the new Fusion deployment by hand. You cannot copy over the old configuration files because they may contain commands or settings which are no longer valid.

Custom Fusion connector plugins and pipeline stages

Custom Java components written for one version of Fusion must be re-compiled and and installed anew for the latest version of Fusion as Fusion’s Java API may have changed.

1.6.3. Version Incompatibilities

Incompatibilities between Fusion 2.4 and version 2.1

Datasource Configuration

All datasource definitions are stored inside of ZooKeeper, and in Fusion 2.4 the structure of these definitions in ZooKeeper has changed; the FUSION-UPGRADE-SCRIPTS repository contains a script which can rewrite these definitions.

Deprecated Connectors

The following connectors have been deprecated:

- DropBox
- Logstash Connector
- S3H Connector
- Slack Connector
- Twitter Search
- Twitter Stream

JavaScript Index pipeline stages

JavaScript Index pipeline stage now requires that the function argument parameters list contain the names of any variable used in the function body, excepting the global 'logger'. This includes references to:+

- object PipelineDocument 'doc'
- object PipelineContext 'ctx'
- string collection name 'collection'
- object BufferingSolrServer 'solrServer'
- object SolrClientFactory 'solrServerFactory'

Prior to 2.4, functions which referenced these objects worked even if they weren’t included in the function argument parameters list. Here are two examples of JavaScript functions which will break in Fusion 2.4 but which work in prior versions:

function returns object 'doc', empty parameter list

```
function () {
    doc.addField('example-field-100','some value');
    return doc;
}
```

function parameter list specifies 'doc' but not 'solrServerFactory'

```
function (doc) {
    var imports = new JavaImporter(
        org.apache.solr.client.solrj.SolrQuery,
        org.apache.solr.client.solrj.util.ClientUtils);
    with(imports) {
        var sku = doc.getFirstFieldValue("sku");
        if (!doc.hasField("mentions")) {
            var mentions = ""
            var productsSolr = solrServerFactory.getSolrServer("products");
            if( productsSolr != null ){
                var q = "sku:"+sku;
                var query = new SolrQuery();
                query.setRows(100);
                query.setQuery(q);
                var res = contactsClient.query(query);
                mentions = res.getResults().size();
                doc.addField("mentions",mentions);
            }
        }
    }
    return doc;
}
```

Configuration file location changes

Previously, log configuration files were located in subdirectories of their Fusion components. As of version 2.4, the Fusion distribution top-level directory 'conf' contains all log configuration files.

1.6.4. Upgrade Fusion 2.1 or 2.2 to Fusion 2.4

This instruction set is valid for all Fusion 2.1 releases as well as Fusion 2.2.0.

Note	Fusion 2.4 introduces changes to the configuration properties for some Fusion datasources. To update these configurations, we have provided a program which can be downloaded from: https://github.com/LucidWorks/fusion-upgrade-scripts. Once you have migrated all Fusion configurations from the current Fusion 2.1 ZooKeeper service to the new Fusion 2.4 ZooKeeper service, you must run this script against the new ZooKeeper service, see Migrate ZooKeeper data

The upgrade process leaves the current Fusion deployment in place while a new Fusion deployment is installed and configured. All of the upgrade operations copy information from the current Fusion over to the new Fusion. This provides a rollback option should the upgrade procedure encounter problems.

The current Fusion configurations must remain as-is during the upgrade process. In order to capture indexing job history, no indexing jobs should be running. If the new Fusion installation is being installed onto the same server that the current Fusion installation is running on, you must either run only one version at a time or else change the Fusion component server ports so that all components are using unique ports for both the current and new versions.

Terminology

These instructions use the following names to refer to the directories involved in the upgrade procedure:

- FUSION_HOME: Absolute pathname to the top-level directory of the Fusion distribution
- FUSION-CURRENT: Name of the FUSION_HOME directory for the current Fusion version, e.g. "/opt/lucidworks/fusion-2.1.2"
- FUSION-NEW: Name of the directory of the upgrade Fusion distribution during the upgrade process, e.g. "/opt/lucidworks/fusion-2.4.1"
- INSTALL-DIR: Directory where the new Fusion version will be installed, e.g. "opt/lucidworks" All scripts and commands in the upgrade instruction set are carried out from this directory.
- FUSION-UPGRADE-SCRIPTS: Full path to the directory that contains the upgrade scripts from https://github.com/LucidWorks/fusion-upgrade-scripts.

Requirements

- File-system permissions: the user running the upgrade scripts and commands must have read/write/execute (rwx) permissions on directory INSTALL-DIR.
- Download but *do not* unpack a copy of the FUSION-NEW distribution. The compressed Fusion distribution requires approximately 1.7 GB disk space. All supported version are available from Lucidworks Fusion Get Started page.
- Disk space requirements: the INSTALL-DIR must be on a disk partition which has enough free space for the complete FUSION-NEW installation, that is, there must be at least as much free space as the size of the FUSION-CURRENT directory. On a Unix system, the following commands can be used:

    - `du -sh fusion` - total size of FUSION-CURRENT.
    - `df -kH` - amount of free space on all file-systems.

- Download a copy of the Fusion upgrade scripts from the GitHub repository https://github.com/LucidWorks/fusion-upgrade-scripts. These upgrade scripts run under Python 2.7. They have been tested with version 2.7.10. If this version of Python isn't available, you should use Python's virtualenv. If you don't have permissions to install packages, you can use python to install `virtualenv` and then from your `virtualenv` python environment, you can install your own versions of theses packages.

Note	These scripts require the environment variable `FUSION_OLD_HOME` which should be set to the location of the current Fusion installation, i.e., the existing 1.2 or 2.1 install.

- Upgrades from 2.1 to 2.4 use the script `src/upgrade-ds-2.1-to-2.4.py`. This script requires the python package kazoo which is a ZooKeeper client.
- Upgrades from 1.2 to 2.4 use two scripts: `src/upgrade-ds-1.2-to-2.4.py` and `bin/download_upload_ds.py`. These scripts require python packages kazoo and requests, which is an HTTP request handler.

Procedure

Unpack FUSION-NEW

- ***Current working directory must be INSTALL-DIR***
  The commands in this section assume that your current working directory is INSTALL-DIR (e.g., "opt/lucidworks"), therefore `cd` to this directory before continuing.
- ***Avoid directory name conflicts between FUSION-CURRENT and FUSION-NEW***
  By default, the Fusion distribution unpacks into a directory named "fusion". If the INSTALL-DIR is the directory which contains the FUSION-CURRENT directory and if the FUSION-CURRENT directory is named "fusion", then you must create a new directory with a different name into which to unpack the Fusion distribution. For example, if your INSTALL-DIR is "/opt/lucidworks" and your FUSION-CURRENT directory is "/opt/lucidworks/fusion", then you should create a directory directory named "fusion-new" and unpack the contents of the distribution here:

```
> mkdir fusion-new
> tar -C fusion-new --strip-components=1 -xf fusion-2.4.1.tar.gz
```

If you are working on a Windows machine, the zipfile unzips into a folder named "fusion-2.4.1" which contains a folder named "fusion". Rename folder "fusion" to "fusion-new" and move it into folder INSTALL-DIR.

Customize FUSION-NEW configuration files and run scripts

The Fusion run scripts in the `FUSION_HOME/bin` directory start and stop Fusion and its component services. The Fusion configuration files `FUSION_HOME/conf` define environment variables used by the Fusion run scripts. The configuration and run scripts for the FUSION-NEW installation must be edited by hand, you cannot copy over existing scripts from the current installation.

The Fusion configuration scripts **might** need to be updated if you have changed default settings. These scripts **will** need to be updated for deployments that:

- Use an external ZooKeeper cluster as Fusion's ZooKeeper service

- Use an external Solr cluster to manage Fusion’s system collections
- Run on non-standard ports
- Have been configured to run over SSL

To facilitate the task of identifying changes made to the current installation, the FUSION-UPGRADE-SCRIPTS repository contains a directory "reference-files" which contains copies of the contents of these directories for all Fusion releases. To identify changes, use the Unix `diff` command with the `-r` flag; e.g., if FUSION-CURRENT is 2.1.1, then these diff commands will report the set of changed files and the changes that were made:

```
> diff -r FUSION-CURRENT/bin FUSION-UPGRADE-SCRIPTS/reference-files/bin-2.1.1
> diff -r FUSION-CURRENT/conf FUSION-UPGRADE-SCRIPTS/reference-files/conf-2.1.1
```

A copy of Fusion is installed on every node in a Fusion deployment. Depending on the role that node plays in the deployment, the configuration settings and run scripts are customized accordingly. Therefore, if you are running a multi-node Fusion deployment this configuration step will be carried out for each node in the cluster.

Copy local data stores in directory `FUSION-CURRENT/data`

The directory `FUSION_HOME/data` contains the on-disk data stores managed directly or indirectly by Fusion services.

- `FUSION_HOME/data/connectors` contains data required by Fusion connectors.
  - `FUSION_HOME/data/connectors/lucid.jdbc` contains third-party JDBC driver files. If your application uses a JDBC connector, you must copy this information over to every server on which will this connector will run.
  - `FUSION_HOME/data/connectors/crawldb` contains information on the filed visited during a crawl. (Preserving crawldb history may not be possible if there are multiple different servers running Fusion connectors services.)
- `FUSION_HOME/data/nlp` contains data used by Fusion NLP pipeline stages. If you are using Fusion’s NLP components for sentence detection, part-of-speech tagging, and named entity detection, you must copy over the model files stored under this directory.
- `FUSION_HOME/data/solr` contains the backing store for Fusion’s embedded Solr (developer deployment only).
- `FUSION_HOME/data/zookeeper` contains the backing store for Fusion’s embedded ZooKeeper (developer deployment only).

If FUSION_CURRENT and FUSION_NEW are installed on the same server, you can copy a subset of these directories using the Unix "cp" command, e.g.:

```
> cp -R FUSION-CURRENT/data/connectors/lucid.jdbc FUSION-NEW/data/connectors
> cp -R FUSION-CURRENT/data/connectors/crawldb FUSION-NEW/data/connectors
> cp -R FUSION-CURRENT/data/nlp FUSION-NEW/data/
```

If FUSION_CURRENT and FUSION_NEW are on different servers, use the Unix `rsync` utility.

Migrate ZooKeeper and Solr for single-node Fusion deployment

If you are running a single-node Fusion deployment and using both the embedded ZooKeeper and the embedded Solr that ships with this distribution, then you must copy over both the configurations and data.

To copy the ZooKeeper configuration:

```
> cp -R FUSION-CURRENT/data/zookeeper FUSION-NEW/data
```

To copy the Solr data:

```
> cp -R FUSION-CURRENT/data/solr FUSION-NEW/data
```

If the Solr collections are very large this may take a while.

Migrate Fusion configurations between ZooKeeper instances

Migration consists of the following steps:

- Copy the ZooKeeper data nodes which contain Fusion configuration information from the FUSION-CURRENT ZooKeeper instance to the FUSION-NEW ZooKeeper instance

  Fusion's utility script zkImportExport.sh is used to copy ZooKeeper data between ZooKeeper clusters. This script is included with all Fusion distributions in the top-level directory named `scripts`.

- Rewrite Fusion datasource configurations

  Fusion 2.4 changed and standardized the configuration properties used by several datasources. The public GitHub repository https://github.com/LucidWorks/fusion-upgrade-scripts contains a python script `src/upgrade-ds-2.1-to-2.4.py` which rewrites these properties.

Copying ZooKeeper data nodes

Note	This step is not necessary if you are doing an in-place upgrade of a single-node Fusion deployment; the copy command described in procedure single-node Fusion ZooKeeper data (above) is sufficient.

Fusion configurations are stored in Fusion's ZooKeeper instance under two top-level znodes:

- Node `lucid` stores all application-specific configurations, including collection, datasource, pipeline, signals, aggregations, and associated scheduling, jobs, and metrics.
- Node `lucid-apollo-admin` stores all access control information, including all users, groups, roles, and realms.

Fusion's utility script zkImportExport.sh is used to migrate ZooKeeper data between ZooKeeper clusters. Migrating configuration information from one deployment to another requires running this script twice:

- The first invocation runs the script in "export" mode, in order to get the set of configurations to be migrated as a JSON dump file.
- The second invocation runs the script in "import" or "update" mode, in order to sent this configuration set to the other Fusion deployment.

When running this script against a Fusion deployment, it is advisable to stop all Fusion services except for Fusion's ZooKeeper service.

Exporting Fusion configurations from FUSION-CURRENT ZooKeeper Service

The ZooKeeper service for FUSION-CURRENT must be running. Either stop all other Fusion services or otherwise ensure that no changes to Fusion configurations take place during this procedure. If you are upgrading from a Fusion 1.2 installation which uses Fusion's embedded Solr service and the ZooKeeper service included with that Solr installation, then starting just the Solr service will start the ZooKeeper service as well. If you are upgrading from a Fusion 2 installation, you can start just the ZooKeeper service via the script "zookeeper" in the `$FUSION_HOME/bin` directory.

The zkImportExport.sh script arguments are:

- `-cmd export` - This is the command parameter which specifies the mode in which to run this program.
- `-zkhost <FUSION_CURRENT ZK>` - The ZooKeeper connect string is the list of all servers,ports for the FUSION_CURRENT ZooKeeper cluster. For example, if running a single-node Fusion developer deployment with embedded ZooKeeper, the connect string is `localhost:9983`. If you have an external 3-node ZooKeeper cluster running on servers "zk1.acme.com", "zk2.acme.com", "zk3.acme.com", all listening on port 2181, then the connect string is `zk1.acme.com:2181,zk2.acme.com:2181,zk3.acme.com:2181`
- `-filename <path/to/JSON/dump/file>` - The name of the JSON dump file to save to.
- `-path <start znode>`
    - To migrate all ZooKeeper data, the path is "/".
    - To migrate only the Fusion services configurations, the path is "/lucid". Migrating just the "lucid" node between the ZooKeeper services used by different Fusion deployments results in deployments which contain the same applications but not the same user databases.
    - To migrate the Fusion users, groups, roles, and realms information, the path is "/lucid-apollo-admin".

Example of exporting Fusion configurations for znode "/lucid" from a local single-node ZooKeeper service:

```
> $FUSION_HOME/scripts/zkImportExport.sh -zkhost localhost:9983 -cmd export -path /lucid -filename
znode_lucid_dump.json
```

Importing ZooKeeper data into FUSION-NEW

ZooKeeper service for FUSION-NEW must be running.

To import configurations, run the zkImportExport.sh script, this time with arguments:

- command; must be `import`
- ZooKeeper connect string for the FUSION-NEW Zookeeper cluster
- Location of JSON dump file.

This command will fail if the "lucid" znode in this Fusion installation contains configuration definitions which are in conflict with the exported data.

Example of importing exported data from previous step into FUSION_NEW ZooKeeper running on test server 'test.acme.com':

```
> $FUSION_HOME/scripts/zkImportExport.sh -zkhost test.acme.com:9983 -cmd import -filename
znode_lucid_dump.json
```

Note that the above command will fail if there is conflict between existing znode structures or contents between the ZooKeeper service and the dump file.

Rewrite datasource configurations for Fusion 2.4

Once all Fusion configurations have been uploaded to the FUSION-NEW ZooKeeper service and while that service is running, you can run the Python programs upgrade-ds-2.1-to-2.4.py or upgrade-ds-1.2-to-2.4.py to update these configurations.

Note	These programs require: • The environment variable "FUSION_HOME" must be set to the FUSION-NEW directory. • The environment variable "FUSION_OLD_HOME" must be set to the FUSION-CURRENT directory. • Python version 2.7, preferably version 2.7.10. • Package: kazoo - a ZooKeeper client The Python virtualenv tool can be used to install the correct Python version and required package.

Set environment variable "FUSION_HOME" to the full path of the FUSION-NEW directory, e.g.:

```
> export FUSION_HOME=/Users/demo/test_upgrade/fusion_2_4_1
```

Run this program with arguments: "--datasources all"

If your current Fusion version is 1.2, run:

```
> python upgrade-ds-1.2-to-2.4.py --datasources all
```

If your current Fusion is version 2, run:

```
> python upgrade-ds-2.1-to-2.4.py --datasources all
```

If a datasource wouldn't have a valid implementation, the application will print a log message on console and continue with the next datasource.

Troubleshooting the upgrade

- Clear your browser cache after starting the UI in the new Fusion instance
- The Fusion 2.4 Index Pipeline Simulator can be used to verify that the existing set of datasource configurations work as expected.

1.6.5. Upgrade Fusion 1.2 to Fusion 2.4

These instructions are valid for Fusion 1.2.3 releases through Fusion 1.2.8.

Note	Several changes have been made to Fusion configurations stored in ZooKeeper: • Fusion 2.1 introduced enhanced security for Fusion datasource passwords which are stored in ZooKeeper as part of datasource and pipeline stage configuration properties. • Fusion 2.4 introduces changes to the configuration properties for some Fusion datasources. To update these configurations, we have provided two python scripts which can be downloaded from: https://github.com/LucidWorks/fusion-upgrade-scripts. Once you have migrated all Fusion configurations from the current Fusion 1.2.x ZooKeeper service to the new Fusion 2.4 ZooKeeper service, you must run both of these scripts against the new ZooKeeper service. This procedure is covered in detail in section Migrate ZooKeeper data

The upgrade process leaves the current Fusion deployment in place while a new Fusion deployment is installed and configured. All of the upgrade operations copy information from the current Fusion over to the new Fusion. This provides a rollback option should the upgrade procedure encounter problems.

The current Fusion configurations must remain as-is during the upgrade process. In order to capture indexing job history, no indexing jobs should be running. If the new Fusion installation is being installed onto the same server that the current Fusion installation is running on, you must either run only one version at a time or else change the Fusion component server ports so that all components are using unique ports for both the current and new versions.

Terminology

These instructions use the following names to refer to the directories involved in the upgrade procedure:

- FUSION_HOME: Absolute pathname to the top-level directory of the Fusion distribution
- FUSION-CURRENT: Name of the FUSION_HOME directory for the current Fusion version, e.g. "/opt/lucidworks/fusion-2.1.2"
- FUSION-NEW: Name of the directory of the upgrade Fusion distribution during the upgrade process, e.g. "/opt/lucidworks/fusion-2.4.1"
- INSTALL-DIR: Directory where the new Fusion version will be installed, e.g. "opt/lucidworks" All scripts and commands in the upgrade instruction set are carried out from this directory.
- FUSION-UPGRADE-SCRIPTS: Full path to the directory that contains the upgrade scripts from https://github.com/LucidWorks/fusion-upgrade-scripts.

Requirements

- File-system permissions: the user running the upgrade scripts and commands must have read/write/execute (rwx) permissions on directory INSTALL-DIR.
- Download but *do not* unpack a copy of the FUSION-NEW distribution. The compressed Fusion distribution requires approximately 1.7 GB disk space. All supported version are available from Lucidworks Fusion Get Started page.
- Disk space requirements: the INSTALL-DIR must be on a disk partition which has enough free space for the complete FUSION-NEW installation, that is, there must be at least as much free space as the size of the FUSION-CURRENT directory. On a Unix system, the following commands can be used:
  - `du -sh fusion` - total size of FUSION-CURRENT.
  - `df -kH` - amount of free space on all file-systems.
- Download a copy of the Fusion upgrade scripts from the GitHub repository https://github.com/LucidWorks/fusion-upgrade-scripts. These upgrade scripts run under Python 2.7. They have been tested with version 2.7.10. If this version of Python isn't available, you should use Python's virtualenv. If you don't have permissions to install packages, you can use python to install `virtualenv` and then from your `virtualenv` python environment, you can install your own versions of theses packages.

Note	These scripts require the environment variable `FUSION_OLD_HOME` which should be set to the location of the current Fusion installation, i.e., the existing 1.2 or 2.1 install.

- Upgrades from 2.1 to 2.4 use the script `src/upgrade-ds-2.1-to-2.4.py`. This script requires the python package kazoo which is a ZooKeeper client.
- Upgrades from 1.2 to 2.4 use two scripts: `src/upgrade-ds-1.2-to-2.4.py` and `bin/download_upload_ds.py`. These scripts require python packages kazoo and requests, which is an HTTP request handler.

Procedure

Unpack FUSION-NEW

- ***Current working directory must be INSTALL-DIR***
  The commands in this section assume that your current working directory is INSTALL-DIR (e.g., "opt/lucidworks"), therefore `cd` to this directory before continuing.
- ***Avoid directory name conflicts between FUSION-CURRENT and FUSION-NEW***
  By default, the Fusion distribution unpacks into a directory named "fusion". If the INSTALL-DIR is the directory which contains the FUSION-CURRENT directory and if the FUSION-CURRENT directory is named "fusion", then you must create a new directory with a different name into which to unpack the Fusion distribution. For example, if your INSTALL-DIR is "/opt/lucidworks" and your FUSION-CURRENT directory is "/opt/lucidworks/fusion", then you should create a directory directory named "fusion-new" and unpack the contents of the distribution here:

```
> mkdir fusion-new
> tar -C fusion-new --strip-components=1 -xf fusion-2.4.1.tar.gz
```

If you are working on a Windows machine, the zipfile unzips into a folder named "fusion-2.4.1" which contains a folder named "fusion". Rename folder "fusion" to "fusion-new" and move it into folder INSTALL-DIR.

Customize FUSION-NEW configuration files and run scripts

The Fusion run scripts in the `FUSION_HOME/bin` directory start and stop Fusion and its component services. The Fusion configuration files `FUSION_HOME/conf` define environment variables used by the Fusion run scripts. The configuration and run scripts for the FUSION-NEW installation must be edited by hand, you cannot copy over existing scripts from the current installation.

The Fusion configuration scripts **might** need to be updated if you have changed default settings. These scripts **will** need to be updated for deployments that:

- Use an external ZooKeeper cluster as Fusion's ZooKeeper service
- Use an external Solr cluster to manage Fusion's system collections
- Run on non-standard ports
- Have been configured to run over SSL

To facilitate the task of identifying changes made to the current installation, the FUSION-UPGRADE-SCRIPTS repository contains a directory "reference-files" which contains copies of the contents of these directories for all Fusion releases. To identify changes, use the Unix `diff` command with the `-r` flag; e.g., if FUSION-CURRENT is 2.1.1, then these diff commands will report the set of changed files and the changes that were made:

```
> diff -r FUSION-CURRENT/bin FUSION-UPGRADE-SCRIPTS/reference-files/bin-2.1.1
> diff -r FUSION-CURRENT/conf FUSION-UPGRADE-SCRIPTS/reference-files/conf-2.1.1
```

A copy of Fusion is installed on every node in a Fusion deployment. Depending on the role that node plays in the deployment, the configuration settings and run scripts are customized accordingly. Therefore, if you are running a multi-node Fusion deployment this configuration step will be carried out for each node in the cluster.

In Fusion 1.2, the `FUSION_HOME/bin` directory contains both the Fusion run scripts and the helper scripts which define common settings and environment variables. In Fusion 2.1, the configuration files `config.sh` and `config.cmd` have been moved to directory `FUSION_HOME/conf`.

Checking a 1.2 installation against the reference scripts for that release requires only a single diff command:

```
> diff -r FUSION-CURRENT/bin FUSION-UPGRADE-SCRIPTS/reference-files/bin-1.2.3
```

If either the "config.sh" or "config.cmd" files have changed, the corresponding files for the Fusion 2 release will be in directory `FUSION_HOME/conf`.

Copy local data stores in the directory FUSION-CURRENT/data

The directory `FUSION_HOME/data` contains the on-disk data stores managed directly or indirectly by Fusion services.

- `FUSION_HOME/data/connectors` contains data required by Fusion connectors.
    - `FUSION_HOME/data/connectors/lucid.jdbc` contains third-party JDBC driver files. If your application uses a JDBC connector, you must copy this information over to every server on which will this connector will run.
    - `FUSION_HOME/data/connectors/crawldb` contains information on the filed visited during a crawl. (Preserving crawldb history may not be possible if there are multiple different servers running Fusion connectors services.)
- `FUSION_HOME/data/nlp` contains data used by Fusion NLP pipeline stages. If you are using Fusion's NLP components

for sentence detection, part-of-speech tagging, and named entity detection, you must copy over the model files stored under this directory.

- `FUSION_HOME/data/solr` contains the backing store for Fusion’s embedded Solr (developer deployment only).
- `FUSION_HOME/data/zookeeper` contains the backing store for Fusion’s embedded ZooKeeper (developer deployment only).

If FUSION_CURRENT and FUSION_NEW are installed on the same server, you can copy a subset of these directories using the Unix "cp" command, e.g.:

```
> cp -R FUSION-CURRENT/data/connectors/lucid.jdbc FUSION-NEW/data/connectors
> cp -R FUSION-CURRENT/data/connectors/crawldb FUSION-NEW/data/connectors
> cp -R FUSION-CURRENT/data/nlp FUSION-NEW/data/
```

If FUSION_CURRENT and FUSION_NEW are on different servers, use the Unix `rsync` utility.

Migrate ZooKeeper and Solr for single-node Fusion deployment

If you are running a single-node Fusion deployment and using both the embedded ZooKeeper and the embedded Solr that ships with this distribution, then you must copy over both the configurations and data.

To copy the ZooKeeper configuration:

```
> mkdir -p FUSION-NEW/data/zookeeper
> cp -R FUSION-CURRENT/solr/zoo_data/* FUSION-NEW/data/zookeeper
```

To check your work: compare the directories `FUSION-CURRENT/solr/zoo_data/` and `FUSION-NEW/data/zookeeper` using the `diff` command. This command succeeds silently when the contents are the same.

```
> diff -r FUSION-CURRENT/solr/zoo_data FUSION-NEW/data/zookeeper
```

To copy the Solr data:

```
> find FUSION-CURRENT/solr -maxdepth 1 -mindepth 1 | grep -v -E "zoo*" | while read f ; do cp -R $f FUSION-NEW/data/solr/; done
```

If the Solr collections are very large this may take a while.

You can use the `diff` command to check your work. The copy command excluded ZooKeeper config data, therefore you should see the following output:

```
> diff -r FUSION-CURRENT/solr FUSION-NEW/data/solr
Only in FUSION-NEW/data/solr: configsets
Only in FUSION-CURRENT/solr: zoo.cfg
Only in FUSION-CURRENT/solr: zoo_data
```

Migrate Fusion configurations between ZooKeeper instances

Migration consists of three steps:

- Copy the ZooKeeper data nodes which contain Fusion configuration information from the FUSION-CURRENT ZooKeeper instance to the FUSION-NEW ZooKeeper instance

  Fusion's utility script zkImportExport.sh is used to copy ZooKeeper data between ZooKeeper clusters. This script is included with all Fusion distributions in the top-level directory named `scripts`.

- Rewrite Fusion datasource configurations

  Fusion 2.4 changed and standardized the configuration properties used by several datasources. The public GitHub repository https://github.com/LucidWorks/fusion-upgrade-scripts contains a python script `src/upgrade-ds-1.2-to-2.4.py` which rewrites these properties.

- Rewrite stored password information used by Fusion datasources and pipelines.

  Fusion 2 encrypts all passwords use by datasources and pipelines to access password-protected data repositories. The public GitHub repository https://github.com/LucidWorks/fusion-upgrade-scripts. contains a a python script `bin/download_upload_ds_pipelines.py` used to edit the stored password information.

Copying ZooKeeper data nodes

Note	This step is not necessary if you are doing an in-place upgrade of a single-node Fusion deployment; the copy command described in procedure single-node Fusion ZooKeeper data (above) is sufficient.

Fusion configurations are stored in Fusion's ZooKeeper instance under two top-level znodes:

- Node `lucid` stores all application-specific configurations, including collection, datasource, pipeline, signals, aggregations, and associated scheduling, jobs, and metrics.
- Node `lucid-apollo-admin` stores all access control information, including all users, groups, roles, and realms.

Fusion's utility script zkImportExport.sh is used to migrate ZooKeeper data between ZooKeeper clusters. Migrating configuration information from one deployment to another requires running this script twice:

- The first invocation runs the script in "export" mode, in order to get the set of configurations to be migrated as a JSON dump file.
- The second invocation runs the script in "import" or "update" mode, in order to sent this configuration set to the other Fusion deployment.

When running this script against a Fusion deployment, it is advisable to stop all Fusion services except for Fusion's ZooKeeper service.

Exporting Fusion configurations from FUSION-CURRENT ZooKeeper Service

The ZooKeeper service for FUSION-CURRENT must be running. Either stop all other Fusion services or otherwise ensure that no changes to Fusion configurations take place during this procedure. If you are upgrading from a Fusion 1.2 installation which uses Fusion's embedded Solr service and the ZooKeeper service included with that Solr installation, then starting just the Solr service will start the ZooKeeper service as well. If you are upgrading from a Fusion 2 installation, you can start just the ZooKeeper service via the script "zookeeper" in the `$FUSION_HOME/bin` directory.

The zkImportExport.sh script arguments are:

- `-cmd export` - This is the command parameter which specifies the mode in which to run this program.
- `-zkhost <FUSION_CURRENT ZK>` - The ZooKeeper connect string is the list of all servers,ports for the FUSION_CURRENT ZooKeeper cluster. For example, if running a single-node Fusion developer deployment with embedded ZooKeeper, the connect string is `localhost:9983`. If you have an external 3-node ZooKeeper cluster running on servers "zk1.acme.com", "zk2.acme.com", "zk3.acme.com", all listening on port 2181, then the connect string is `zk1.acme.com:2181,zk2.acme.com:2181,zk3.acme.com:2181`
- `-filename <path/to/JSON/dump/file>` - The name of the JSON dump file to save to.
- `-path <start znode>`
    - To migrate all ZooKeeper data, the path is "/".
    - To migrate only the Fusion services configurations, the path is "/lucid". Migrating just the "lucid" node between the ZooKeeper services used by different Fusion deployments results in deployments which contain the same applications but not the same user databases.
    - To migrate the Fusion users, groups, roles, and realms information, the path is "/lucid-apollo-admin".

Example of exporting Fusion configurations for znode "/lucid" from a local single-node ZooKeeper service:

```
> $FUSION_HOME/scripts/zkImportExport.sh -zkhost localhost:9983 -cmd export -path /lucid -filename
znode_lucid_dump.json
```

Importing ZooKeeper data into FUSION-NEW

ZooKeeper service for FUSION-NEW must be running.

To import configurations, run the zkImportExport.sh script, this time with arguments:

- command; must be `import`
- ZooKeeper connect string for the FUSION-NEW Zookeeper cluster
- Location of JSON dump file.

This command will fail if the "lucid" znode in this Fusion installation contains configuration definitions which are in conflict with the exported data.

Example of importing exported data from previous step into FUSION_NEW ZooKeeper running on test server 'test.acme.com':

```
> $FUSION_HOME/scripts/zkImportExport.sh -zkhost test.acme.com:9983 -cmd import -filename
znode_lucid_dump.json
```

Note that the above command will fail if there is conflict between existing znode structures or contents between the ZooKeeper service and the dump file.

Rewrite datasource configurations for Fusion 2.4

Once all Fusion configurations have been uploaded to the FUSION-NEW ZooKeeper service and while that service is running, you can run the Python programs upgrade-ds-2.1-to-2.4.py or upgrade-ds-1.2-to-2.4.py to update these configurations.

Note	These programs require: • The environment variable "FUSION_HOME" must be set to the FUSION-NEW directory. • The environment variable "FUSION_OLD_HOME" must be set to the FUSION-CURRENT directory. • Python version 2.7, preferably version 2.7.10. • Package: kazoo - a ZooKeeper client The Python virtualenv tool can be used to install the correct Python version and required package.

Set environment variable "FUSION_HOME" to the full path of the FUSION-NEW directory, e.g.:

```
> export FUSION_HOME=/Users/demo/test_upgrade/fusion_2_4_1
```

Run this program with arguments: "--datasources all"

If your current Fusion version is 1.2, run:

```
> python upgrade-ds-1.2-to-2.4.py --datasources all
```

If your current Fusion is version 2, run:

```
> python upgrade-ds-2.1-to-2.4.py --datasources all
```

If a datasource wouldn't have a valid implementation, the application will print a log message on console and continue with the next datasource.

Rewrite stored password information used by Fusion datasources and pipelines

Once you have migrated all Fusion configurations to the FUSION_NEW ZooKeeper service, you must update the migrated datasource configurations by running the script download_upload_ds_pipelines.py against the FUSION_NEW zookeeper in order to rewrite any stored datasource passwords that are specified as part of the configuration for a datasource or pipeline.

Note	The script bin/download_upload_ds_pipelines.py requires: • Python version 2.7, preferably version 2.7.10. • Package: kazoo - a ZooKeeper client • Package: requests - an HTTP request handler • Environment variable FUSION_OLD_HOME set to location of Fusion 1.2 home. The Python virtualenv tool can be used to install the correct Python version and required packages.

The rewrite process consists of a download step which exports the ZooKeeper configuration information and an upload step which rewrites the information and then imports it back into ZooKeeper.

This script uses the following arguments and values:

- "--zk-connect": the ZooKeeper server:port for FUSION-NEW
- "--action": either "download" or "upload".
- "--fusion-url": URL of Fusion API service to upload configurations to
- "--fusion-username": name of Fusion user with admin privileges; the script will prompt for username's password.

Download configurations from ZooKeeper

No services for FUSION-NEW should be running, except for ZooKeeper. If your Fusion installation uses an external ZooKeeper, then this must be running. If your Fusion installation uses an embedded ZooKeeper, then you must have copied the ZooKeeper data from FUSION-CURRENT to FUSION-NEW.

Start the ZooKeeper service:

```
> FUSION-NEW/bin/zookeeper start
```

Run the script to download the configurations.

```
> python FUSION-UPGRADE-SCRIPTS/bin/download_upload_ds_pipelines.py \
 --zk-connect localhost:9983 --action download
```

To check your work, check that directory "fusion_upgrade_2.1" was created and that is contains definitions for all datasources and pipelines. Do not remove this directory until you have successfully completed the upload step.

If you are running embedded ZooKeeper, shut it down again:

```
> FUSION-NEW/bin/zookeeper stop
```

Upload configurations to the Fusion API service

Start FUSION-NEW:

```
> FUSION-NEW/bin/fusion start
```

Once it is running, run the script in upload mode to propagate the configurations in directory "fusion_upgrade_2.1".

At this point in the migration process, the FUSION-NEW ZooKeeper information is the same as the FUSION-CURRENT Zookeeper information; therefore the password for the admin user is the same.

To upload data to the Fusion API services, you must supply the admin username and password as arguments to the script:

- "--fusion-username": name of Fusion user with admin privileges
- "--fusion-password": password for Fusion user

```
> FUSION-NEW/bin/fusion start
> python FUSION-UPGRADE-SCRIPTS/bin/download_upload_ds_pipelines.py \
 --zk-connect localhost:9983 --action upload --fusion-url http://localhost:8764/api \
 --fusion-username <admin>
```

Copy and convert the crawldb

The Fusion "crawldb" records the results of running datasource jobs. This information must be copied from FUSION-CURRENT to FUSION-NEW and the data format must be converted to the format used in Fusion 2.1 via the conversion utility com.lucidworks.fusion-crawldb-migrator-0.1.0.jar.

Copy the Fusion "crawldb" directory:

```
> cp -R FUSION-CURRENT/data/connectors/crawldb FUSION-NEW/data/connectors/
```

The crawldb data format changed in Fusion 2.1, therefore to upgrade to 2.1, the crawldb data must be converted with the the conversion utility com.lucidworks.fusion-crawldb-migrator-0.1.0.jar.

The `anda-v1-to-v2` command allows Fusion 1.2.x connector DBs to be updated to the new v2.x format. It requires:

- A Fusion pre 2.1 installation (FUSION-CURRENT)
- A Fusion 2.1 or later installation (FUSION-NEW).
  - All FUSION-CURRENT datasource configurations must have been migrated to FUSION-NEW (see Migrate ZooKeeper data)
  - All FUSION-CURRENT crawldb files must have been copied over to the FUSION-NEW deployment.

If the FUSION-NEW installation is not currently running, start it:

```
> FUSION-NEW/bin/fusion start
```

The `anda-v1-to-v2` takes the following arguments:

- path-to-FUSION-CURRENT
- path-to-FUSION-NEW

- the -z flag specifies the ZooKeeper server:port for FUSION-NEW

The command to run this utility from the INSTALL-DIR is:

```
> java -jar FUSION-UPGRADE-SCRIPTS/bin/com.lucidworks.fusion-crawldb-migrator-0.1.0.jar anda-v1-v2 fusion
fusion-new -z localhost:9983
```

Once the task successfully completes, **the last few lines of logging show the output directory of the new DB files**. The output must be copied over to FUSION-NEW. To do this, remove the existing `lucid.anda` db directories, then copy the new `lucid.anda` directories generated from this utility into that same location:

```
> rm -Rf FUSION-NEW/data/connectors/crawldb/lucid.anda/*
> mv ${path-printed-from-command-output} FUSION-NEW/data/connectors/crawldb/lucid.anda/
```

This completes the upgrade process.

Troubleshooting the upgrade

- Clear your browser cache after starting the UI in the new Fusion instance
- The Fusion 2.4 Index Pipeline Simulator can be used to verify that the existing set of datasource configurations work as expected.

1.7. Integrate Fusion with an Existing Solr Deployment

If you have already implemented Solr as a standalone instance or as a SolrCloud cluster, you can add Fusion to your existing Solr deployment and import your Solr collections into Fusion. Each Fusion collection can import one Solr collection.

- *If your existing Solr instance is running in SolrCloud mode*, you can use Fusion's UI to modify configuration files (such as schema.xml or solrconfig.xml) and create Solr collections.
- *If your existing Solr instance is running in standalone mode*, you can still connect it to Fusion. Fusion can send documents to a standalone Solr instance and query the instance. But you won't be able to use Fusion's UI to create Solr collections (Solr cores) or to modify Solr configuration files.

1.7.1. Prerequisites

- You have already installed Fusion.
- You have already installed Solr, which must meet these Solr requirements.
- You have already installed ZooKeeper, which must meet these ZooKeeper requirements.

Note	We recommend that you create an *external* ZooKeeper cluster (external to both Fusion and SolrCloud).

- Your Solr deployment must contain one or more collections (cores).
- In SolrCloud mode, Solr must be configured to use ZooKeeper.

1.7.2. Configure Fusion to use an existing Solr deployment

Use the Fusion UI or the Fusion API to integrate Fusion with an existing Solr deployment.

Use the Fusion UI

1. Create a Fusion search cluster:
    a. In the Fusion UI, navigate to **System** > **Solr Clusters** and click **New Solr Cluster**.
    b. Enter this information:
        - A cluster ID of your choice
        - Whether SolrCloud is enabled
        - The connect string (to tell Fusion how to connect to the SolrCloud cluster or Solr instance)
            - For SolrCloud, this is the ZooKeeper connect string.
            - For a standalone Solr instance, this is the URL of the Solr instance.
    c. Verify that the connection is working by clicking **Cores** in the new cluster and inspecting the contents.
2. Create a Fusion collection that points to your Solr cluster and collection:
    a. In the UI, navigate to **Collections** and click **Add a Collection**.
    b. Enter a name for the new collection.
    c. Click **Advanced**.
    d. Select your SolrCloud cluster or Solr instance from the dropdown.

e. Enter the name of the Solr collection to import.

Use the Fusion API

Use the Search Cluster API to create a Solr cluster.

Then use the Collections API to create and configure a collection.

1.7.3. Sending Documents to Solr through Fusion

You can use the Fusion connectors to crawl documents and index them to your existing Solr deployment.

1. Follow the steps above to create and configure a search cluster and a collection that points to Solr.
2. Define an index pipeline that ends with a Solr Indexer stage that sends the documents to Solr.
3. Use one of these methods to ingest your data:
    - In the collection that points to your Solr collection, define a datasource using the connector of choice.
    - Send prepared documents directly to the index pipeline for processing. See Pushing Documents to a Pipeline.
    - It's also possible to use a different indexing process besides a connector, such as a script that sends documents through the index pipeline.

When documents are sent to Solr, a buffering solrServer is used. Buffering the updates reduces the number of HTTP requests made from Fusion to Solr, which can significantly affect processing time. For example, when processing simple documents, you should always try to buffer as many documents as possible to increase throughput. When processing complex documents, you should use small batch sizes. You should only turn buffering off if you are using an older version of Solr and you want Fusion to catch and document indexing errors.

1.7.4. Querying Solr via Fusion requests

Indexed documents are stored in Solr indexes. You can query for these documents by using query pipelines. The query pipelines let you define your query parameters – such as how many records to return, the fields you'd like, how to structure facets, and so on. You also have the ability to add JavaScript to the response processing, and define landing pages or specific boost levels depending on the user's query. See Query Pipelines.

If you prefer, you can also use the Solr API and SolrAdmin API to query Solr directly.

1.8. Configuring Fusion for SSL

Fusion uses the Java Secure Socket Extension (JSSE) framework to enable SSL configuration for secure communication between the Fusion UI and any HTTP client.

To configure Fusion for SSL you must install an SSL certificate and enable SSL in the Fusion UI.

1.8.1. Installing an SSL certificate

The server has a locally-protected private key that is accessible via a JSEE keystore. The keystore maintains both the server certificate and the private key.

Important	In a production environment, SSL certificates must be signed by a trusted Certificate Authority (CA).

To store certificates, you can use the Java keytool, which is part of the JDK. When you have a signed certificate, then you create a JSSE keystore by using the keytool "import" command.

In the following example, we have a signed certificate, in pfx format, in a PKCS#12 file called `fusion.keystore.p12`. The following command creates a new JSSE keystore (JKS):

```
keytool -importkeystore -srckeystore /opt/lucidworks/fusion/apps/jetty/ui/etc/fusion.keystore.p12 \
 -srcstoretype pkcs12 -destkeystore /opt/lucidworks/fusion/jetty/ui/etc/keystore -deststoretype JKS
```

If you have the certificate and private key as separate files, then you need to use openssl to create a PKCS#12 file. For example:

```
openssl pkcs12 -export -out /home/admin/keys/keystore.pkcs12 -in /home/admin/keys/fullchain.pem -inkey
/home/admin/keys/privkey.pem
```

Note	When prompted for a password, **do not** enter a blank password.

Now use keytool to import the PKCS#12 file into Java keystore format and optionally delete the PKCS#12 file:

```
keytool -importkeystore -srckeystore /home/admin/keys/keystore.pkcs12 -srcstoretype PKCS12 -destkeystore
{fusion_path}/apps/jetty/ui/etc/keystore
```

Note	If your server certificate is signed by an intermediate CA rather than a root CA, you must add the intermediate certificate to the keystore before you add the server certificate.

1.8.2. Configuring the Fusion UI

The current version of the Fusion UI runs an embedded Jetty server. In the Fusion distribution, this server is in directory `fusion/apps/jetty/ui`. Server configuration files including the keystore file and HTTPS and SSL config files are stored in the subdirectory `fusion/apps/jetty/ui/etc`.

To configure the Fusion UI server for SSL, you must:

- configure the embedded web server to use the JSSE keystore which has your certificates
- configure the web server ports
- redirect HTTP request to HTTPS

As a training exercise, we show how to configure the Fusion UI for SSL with the default Fusion developer deployment and test the configuration via a web browser running on the same machine, i.e., the web client sends requests to server "localhost" on port 8764, the default port for the Fusion UI. In an enterprise deployment, of course, the browser client and Fusion UI will be on different machines, and the Fusion UI wouldn't be using port 8764.

1.8.3. Configure Jetty to use the JSSE keystore

The following details are taken from the eclipse.org Jetty documentation pages.

The generated SSL certificates held in the key store are configured on Jetty by injecting an instance of a SslContextFactory object and passing it to the connector's SslConnectionFactory, which is done in the Jetty distribution in file `jetty-https.xml`. The SslContextFactory object is configured in the file `jetty-ssl.xml`.

These XML files are in the `fusion/apps/jetty/home/etc` directory. The directory that contains the Jetty server instance which runs the Fusion UI is directory `fusion/apps/jetty/ui`.

This means that you must:

1. Copy the files `jetty-https.xml` and `jetty-ssl.xml` from `fusion/apps/jetty/home/etc` to `fusion/apps/jetty/ui/etc`.
2. Edit the `jetty-ssl.xml` file and update the following the sslContextFactory class properties so that they match the actual keystore location and passwords:
   - KeyStorePath
   - KeyStorePassword
   - KeyManagerPassword
   - TrustStorePath
   - TrustStorePassword

For our example, the path values are all "my.keystore.jks" and the password values are all "secret".

```
<Configure id="sslContextFactory" class="org.eclipse.jetty.util.ssl.SslContextFactory">
  <Set name="KeyStorePath"><Property name="jetty.base" default="." />/<Property name="jetty.keystore"
default="etc/my.keystore.jks"/></Set>
  <Set name="KeyStorePassword"><Property name="jetty.keystore.password" default="secret"/></Set>
  <Set name="KeyManagerPassword"><Property name="jetty.keymanager.password" default="secret"/></Set>
  <Set name="TrustStorePath"><Property name="jetty.base" default="." />/<Property name="jetty.truststore"
default="etc/my.keystore.jks"/></Set>
  <Set name="TrustStorePassword"><Property name="jetty.truststore.password" default="secret"/></Set>
```

1.8.4. Configure the Fusion UI startup script

The Fusion UI startup script, path `fusion/bin/ui`, sets a series of environment variables and then starts the Jetty server. The arguments to exec command to start the Jetty server should be changed as follows:

- change command line argument "jetty.port=$HTTP_PORT" to "https.port=$HTTP_PORT"
- add XML config files to the command line arguments

Here is the command to start Jetty in the script fusion/bin/ui, after making the above edits:

```
exec "$JAVA" \
    $JAVA_OPTIONS \
    "-Djava.io.tmpdir=$JETTY_BASE/work" \
    "-Dlog4j.configurationFile=file:$JETTY_BASE/resources/log4j2.xml" \
    "-Djava.util.logging.manager=org.apache.logging.log4j.jul.LogManager" \
    "-Dapollo.home=/path/to/fusion" \
    "-Dcom.lucidworks.apollo.admin.zk.connect=$FUSION_ZK" \
    "-XX:OnOutOfMemoryError=/path/to/fusion/bin/oom.sh ui" \
    -jar "$JETTY_HOME/start.jar" \
    "jetty.home=$JETTY_HOME" \
    "jetty.base=$JETTY_BASE" \
    "https.port=$HTTP_PORT" \
    "STOP.PORT=$STOP_PORT" \
    "STOP.KEY=$STOP_KEY" \
    "$JETTY_BASE/etc/jetty-ssl.xml" \
    "$JETTY_BASE/etc/jetty-https.xml" \
    "$JETTY_BASE/etc/jetty-logging.xml"
```

1.8.5. Disabling HTTP

Note	Only do this if blocking access to HTTP using the firewall is not feasible.

To entirely disable HTTP, remove the HTTP connector from the Jetty startup configuration:

```
rm {fusion_path}/apps/jetty/ui/start.d/http.ini
```

1.8.6. Configuring Fusion for SSL Solr/SolrCloud

To configure the Fusion HTTP client for a SSL-ed Solr or SolrCloud server, you must specify the javax.net.ssl system properties.

See the Solr wiki instructions for enabling SSL for SolrCloud.

Step 1. Edit the configuration file

In the `fusion/conf/config.sh` file, add the following properties to the options `API_JAVA_OPTIONS`, `CONNECTORS_JAVA_OPTIONS` and `UI_JAVA_OPTIONS`:

```
    -Djavax.net.ssl.keyStore=/path/to/solr-ssl.keystore.jks \
    -Djavax.net.ssl.keyStorePassword=secret \
    -Djavax.net.ssl.trustStore=/path/to/solr-ssl.keystore.jks \
    -Djavax.net.ssl.trustStorePassword=secret
```

Step 2. Register SolrCloud as a search cluster in Fusion

Send a request to the REST API 'searchCluster' endpoint:

```
curl -u admin:pass -H 'Content-type: application/json' -X POST
'http://localhost:8764/api/apollo/searchCluster' -d '
{
  "id" : "ssl",
  "connectString" : "localhost:2181",
  "cloud" : true
}
```

Step 3. Test: create collection, index data, query collection

Create collection in SolrCloud with configured SSL:

```
curl -u admin:pass -H 'Content-type: application/json' -X POST 'http://localhost:8764/api/apollo/collections'
-d '
{
  "id" : "mycollection",
  "searchClusterId" : "ssl"
}'
```

Index data using an existing pipeline:

```
curl -u admin:pass 'http://localhost:8764/api/apollo/index-pipelines/mycollection-
default/collections/mycollection/index' -XPOST -H "Content-type: application/json" -d '{
  "id": "1",
  "foo_s": "bar",
  "spam_s": 42
}'
```

Query the collection using the default query pipeline:

```
curl -u admin:pass 'http://localhost:8764/api/apollo/query-pipelines/mycollection-
default/collections/mycollection/select?q=*:*'
```

1.8.7. Generating a self-signed certificate

If don’t have signed certificates from an external CA, then you can generate a set of self-signed certificates using the Java keytool utility to generate a public/private key pair and generate a self-signed certificate.

The keytool option "-genkeypair" generates a public/private key pair. It wraps the public key into an X.509 v3 self-signed certificate, which is stored as a single-element certificate chain. This certificate chain and the private key are stored in a new keystore entry identified by alias. The full set of arguments to this command are:

```
-genkeypair
    {-alias alias}
    {-keyalg keyalg}
    {-keysize keysize}
    {-sigalg sigalg}
    [-dname dname]
    [-keypass keypass]
    {-startdate value}
    {-ext ext}*
    {-validity valDays}
    {-storetype storetype}
    {-keystore keystore}
    [-storepass storepass]
    {-providerClass provider_class_name {-providerArg provider_arg}}
    {-v}
    {-protected}
    {-Jjavaoption}
```

Arguments of interest are:

- keyalg specifies the algorithm to be used to generate the key pair. This must be RSA in order to talk to browser clients IE and Navigator.
- keysize specifies the size of each key to be generated. Depends on keyalg. For RSA, this should be 2048.
- dname specifies the X.500 Distinguished Name to be associated with the alias, and is used as the issuer and subject fields in the self-signed certificate. If no distinguished name is provided at the command line, the user will be prompted for one. An X.500 Distinguished name is a set of named fields, of these, the field named "CN", ("Common Name"), is the internet-facing fully qualified domain name of the server.
- keypass is a password used to protect the private key of the generated key pair. If no password is provided, the user is prompted for it. If you press RETURN at the prompt, the key password is set to the same password as that used for the keystore. keypass must be at least 6 characters long.
- ext is used to embed extensions into the certificate generated. For Fusion, the server certificate should include the "SAN" or SubjectAlternativeName extension which allows alternative URIs and IP addresses to be associated with this certificate.

Keystore files created by the keystore tool are in the JKS format, which is a proprietary file format capable of storing multiple key-pairs, certificates, and symmetric encryption keys, and with all entries indexed by the keypair alias.

To generate a self-signed certificate for the Fusion UI running as "localhost", we use the following arguments:

```
keytool -genkeypair \
 -alias localhost -keyalg RSA -keysize 2048 \
 -keypass secret -storepass secret \
 -validity 365 -keystore my.keystore.jks \
 -ext SAN=DNS:localhost,IP:127.0.0.1 \
 -dname "CN=localhost, OU=org unit, O=org, L=loc, ST=st, C=country"
```

Note	The value for `CN` must match the hostname you will be using to access Fusion. For example, if will use the URL `https://myfusion.com:8864` to access the UI then the CN should have the value `myfusion.com`.

This keystore file can now be imported to the Fusion UI keystore. To check the generated keystore we use the openssl tool to pretty-print the signed certificate in the file `my.keystore.jks`. This is not strictly necessary; this should always be done before sending the certificate to a CA to get it signed properly in order to verify that the certificate information is complete and correct.

To get from the keystore JKS format to a human-readable printout, it must be converted to the text-based "PEM" format, which is ASCII (Base64) armored data prefixed with a `—— BEGIN ···` line. This requires three steps.

First, we use the keytool to convert the proprietary JKS format to the PKCS #12 format:

```
keytool -importkeystore \
 -srckeystore my.keystore.jks -destkeystore my.keystore.p12 \
 -srcstoretype jks -deststoretype pkcs12
```

This command prompts for passwords - as before, the password is "secret".

Next, we use openssl to convert the PKCS format to PEM format:

```
openssl pkcs12 -in my.keystore.p12 -out my.keystore.pem
```

Finally, to pretty-print the certificate, we use the following openssl command:

```
openssl x509 -in my.keystore.pem -text -noout
```

This converts the PEM format to text format, and writes the output to the terminal. The output is:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1442779707 (0x55ff123b)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=country, ST=st, L=loc, O=org, OU=org unit, CN=localhost
        Validity
            Not Before: Apr  8 07:39:35 2015 GMT
            Not After : Apr  7 07:39:35 2016 GMT
        Subject: C=country, ST=st, L=loc, O=org, OU=org unit, CN=localhost
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:96:04:46:6c:be:7f:ec:ea:18:fa:28:11:a9:fb:
                    3d:07:c5:3c:49:39:57:11:24:1d:75:47:5d:76:26:
                    4b:73:c0:ea:44:7a:a5:59:3a:a7:4b:16:eb:1f:be:
                    05:f1:2a:be:62:72:2c:67:ec:d3:8b:ad:76:af:dc:
                    6d:14:ca:c9:75:5a:76:24:80:c6:f8:55:b0:27:6a:
                    fa:a8:1b:4b:5c:55:93:49:ff:f8:84:67:29:56:80:
                    ca:c1:d8:c4:8c:b8:57:a4:78:6a:e8:e0:2c:51:74:
                    fb:fb:52:3d:d3:e9:58:e5:11:79:5f:5a:70:ec:c3:
                    de:d7:56:36:67:fd:52:dd:73:60:f7:93:9f:00:c0:
                    36:49:65:7d:77:45:76:34:0e:81:38:96:2b:19:b0:
                    30:8b:ac:2f:ae:dd:92:41:78:da:47:32:02:f7:0d:
                    04:f5:51:85:dc:06:58:08:9b:d2:a0:69:52:ac:b2:
                    7d:c7:bd:16:1d:9e:af:e2:2b:6a:61:8e:cb:a9:ec:
                    fc:01:fe:6b:34:49:1c:d8:75:8b:ca:ec:ea:fd:93:
                    0a:8b:34:6b:77:98:ec:83:6f:d2:bc:81:ec:f5:18:
                    48:41:db:92:da:ef:19:19:27:5b:05:5f:8c:6e:1e:
                    9d:e5:90:42:6f:36:8f:11:49:05:aa:dd:a5:c9:0a:
                    fe:81
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                DNS:localhost, IP Address:127.0.0.1
            X509v3 Subject Key Identifier:
                E1:D1:66:F6:1F:5A:18:1A:82:33:A7:32:0E:0B:EA:8E:C0:D3:ED:05
    Signature Algorithm: sha256WithRSAEncryption
        56:a8:31:c0:94:8f:5a:80:27:17:36:ba:e0:ff:e6:59:13:9e:
        67:6b:1e:7d:67:04:5c:e1:88:9e:d9:89:11:ca:88:a0:21:4c:
        90:a8:8b:9d:b5:ba:0f:92:65:da:c5:a9:91:82:17:46:55:43:
        82:78:92:b1:f9:f5:2b:65:5e:b0:93:03:3b:94:83:bb:fe:9b:
        09:a9:f3:82:d2:4d:b5:72:e9:ee:75:15:31:3d:18:a7:c1:e8:
        45:44:1d:40:d2:eb:96:b7:01:41:dd:9d:1c:31:e6:45:4a:c2:
        3d:ec:22:1a:35:ec:38:62:e8:3d:b3:30:10:d3:88:09:5a:87:
        54:87:fb:92:d6:0e:74:52:3c:f2:c3:f5:70:61:ea:72:3f:cd:
        65:72:34:6f:51:94:13:e0:7a:73:bb:57:c8:ad:98:f7:3f:43:
        4d:75:96:db:cf:2e:e6:82:1c:c2:97:38:2d:37:06:c4:27:db:
        87:82:6b:c6:01:71:f7:e9:1f:69:62:0d:cc:54:e9:f4:25:86:
        6a:e2:38:72:c3:9c:53:b4:6e:4f:ae:8a:09:36:14:f0:10:57:
        9c:c9:a8:a3:a5:e6:db:d1:d4:39:95:f3:54:95:4f:2f:db:59:
        b6:bd:77:00:77:c2:9d:4f:d9:04:d5:af:33:bb:2e:0f:65:a9:
        74:ff:66:f2
```

1.8.8. References and tutorials

Transport Layer Security (wikipedia.org)

Public Key Certificate(wikipedia.org)

OpenSSL Cookbook (free ebook)

OpenSSL Command Line Utilities (openssl.org wiki)

Java Tutorials: Generating and Verifying Certificates

IBM developerWorks: What is the JSSE all about?

1.9. Web Authentication Cookie FAQs

Does the application use web authentication cookies?

Yes, we use a session cookie for maintaining authenticated user identification.

Is the cookie used for non-authentication purposes?

No, the cookie contains exactly one value, the session ID.

Is the cookie set to the narrowest/lowest path or domain needed in order to prevent inadvertent or unauthorized sharing of cookies by other web applications?

Yes, the path is set to /api only (the narrowest path). As recommended by OWASP, we do not directly set the domain attribute, so the default ends up being the origin server.

Are the cookies non-persistent?

The cookies are session based, and not persisted beyond logout or via timeout.

Is the value of the cookie not predictable and does it provide 64-bit entropy?

The cookie value is a Java UUID, which uses the SecureRandom class. A little research leads me to believe it’s a 128 bit value, with 122 bit randomness.

Are default values not used for the name of the cookie?

As recommended by OWASP, the cookie name is vague/meaningless. It is simply, "id".

Is the cookie set via SSL channel and are the 'secure' and 'HTTPOnly' attributes set?

If the web server is running under SSL, then the cookie is set to secure and HttpOnly is set to true.

Can the cookie be manually deleted through a logout button that sets the cookie value to null or the cookie value is rendered invalid on the server after a period of inactivity?

Yes, the cookie can be manually deleted. There is also a timeout mechanism:

- 8 hour absolute lifetime - it never lasts longer than 8 hours
- 1 hour soft lifetime - if the last request time was > than 1 hour, the session is destroyed. Otherwise, the lifetime is bumped up 1 more hour, until the maximum 8 hour limit is met.

Is the cookie cleared during the authentication of a user?

Yes.

1.10. System Usage Monitor

The System Usage Monitor is a voluntary program to allow users to anonymously send basic information about their system to Lucidworks. We use this information to analyze the types of systems in use by our customers and how they are used so we can improve our product.

At no point does the system collect information that could identify you, your organization or the specific documents indexed. Only minimal data is sent about the type of content indexed. Our website has more information about our privacy policy.

1.10.1. Information Collected by the Usage Monitor

The System Usage Monitor collects the following information:

- uuid - a randomly generated identifier per Fusion cluster.
- System information: the operating system, version and java version will also be reported.
- nodes - the number of Apollo nodes in use. This is calculated from Jetty processes on the same node or on a different node.
- Solr statistics:
  - Number of Solr nodes, total number of collections and number of Fusion collections.
  - Number of documents and the number that are regular documents and the number that are signals.
  - Number of total search requests and the number that are document search requests and the number that are signal search requests.
  - Total time to execute all search requests, to execute only document search requests, and to execute signal search requests.
- Aggregations:
  - Number of aggregation runs and how long they took (in ms).
  - Number of signals processed.
  - Number of aggregated signals.
- Recommendations:
  - Number of recommendation requests for each type of recommentation.
  - Total time to execute each type of recommendation request.

You can see the data that will be sent to Fusion with the Usage API and also in the UI by going to the Systems tab, then 'Heartbeat data'. The UI and the REST API will only report the data currently scheduled to be sent so is not a complete picture of all data collected.

1.10.2. How Data is Sent

Data is sent to Fusion once per week, and also whenever the system is restarted.

When Fusion is started, the System Usage Monitor will transmit data about your system to a server hosted by Lucidworks with two HTTP requests. The first request contains system-level information and if that is successful, the second request will send system-specific information.

The information is sent via an encrypted POST request to https://heartbeat.lucidworks.io. Each request includes a unique identifier, which is anonymous and can’t be used to identify the sender. The IP that sent the request is not stored with the request.

1.10.3. How to Opt-Out

By default, the usage monitor is enabled in your system. If you would like to opt-out of sending this data to Lucidworks, you can disable the usage monitor. There are two ways to enable or disable the usage monitor:

1. Go to the Heartbeat Data page in the Fusion UI (System → Heartbeat Data), and deselect the "Report Heartbeat" option.
2. Use the Configurations API and send a PUT request as follows:

```
curl -u user:pass -H 'Content-type: application/json' -X PUT -d '"false"'
http://localhost:8764/api/apollo/configurations/usageMonitor
```

Chapter 2. First Run Tutorial

This section is an introductory, step-by-step guide to using the Fusion UI for basic search and indexing. This guide helps you to:

- Verify that the installed Fusion instance components are installed and communicating properly.
- Use the Fusion UI to create and search a small dataset.
- Introduce a few underlying principles of search and indexing over structured and semi-structured texts.

Fusion supports several approaches to indexing content. In this section we use Fusion's web crawler to index a website which contains Shakespeare's sonnets. This dataset is chosen precisely because it is simple and trivial. Web data is challenging to parse because a page of HTML contains many different kinds of elements: content, meta-data, scripts, layout directives, and links, all of which can be effectively processed using Fusion pipelines. By the end of this exercise, you should understand the Fusion UI and workflow and will be ready to create custom pipelines in order to effectively process your data.

2.1. Fusion Key Concepts

Fusion uses the Solr/Lucene search engine to evaluate search requests and return results in the form of a ranked list of document ids. Fusion extends Solr/Lucene functionality via a REST API and a UI built on top of that REST API.

The Fusion UI is organized around the following key concepts:

- **Collections** store your data.
- **Documents** are the items that are returned as search results.
- **Fields** are the things that are actually stored in a collection.
- **Datasource** are the conduit between your data repository and Fusion.
- **Pipelines** encapsulate a sequence of processing steps, called **stages**.
    - Index pipelines process the raw data received from a datasource into fielded documents for indexing into a Fusion collection.
    - Query pipelines process search requests and return an ordered list of matching documents.
- **Relevancy** is the metric used to order search results. It is a non-negative real number that indicates the similarity between a search request and a document.

2.2. Download and Install Fusion

Fusion is distributed as a gzipped tar file or as a compressed zip file that can be used directly to run a single-node local Fusion instance.

On Linux and Mac OS, the Fusion distribution unpacks into a directory named `fusion`; this is the Fusion home directory.

On Windows, the archive unzips as a directory (folder) with the name of the zip file, for example, `fusion-2.1.0`. That directory contains a single directory named `fusion`. The directory `fusion` is the Fusion home directory.

Note: If you install Fusion on Windows, use the freely available 7zip file archiver to unzip the archive, because 7zip is more robust than the standard Windows compression utility. The Fusion archive contains a large number of 3rd party jar files, and the standard Windows compression utility cannot reliably deal with all of them. Visit the 7zip download page for the latest version.

Use the script fusion/bin/fusion with the command-line arguments `start` and `stop` to start and stop Fusion, respectively.

Start Fusion from a terminal window (Linux or Mac):

```
$ cd /path/to/fusion
$ ./bin/fusion start
```

Successful startup results in several lines of output to the terminal window, listing the Fusion components and the ports they are listening on:

```
2016-10-04 18:52:04Z Starting Fusion ZooKeeper on port 9983
2016-10-04 18:52:14Z Starting Fusion Solr on port 8983
2016-10-04 18:52:39Z Starting Fusion Spark Master on port 8766
2016-10-04 18:52:39Z Starting Fusion Spark Worker on port 8769
2016-10-04 18:52:39Z Starting Fusion API Services on port 8765
2016-10-04 18:52:45Z Starting Fusion Connectors on port 8984
2016-10-04 18:52:50Z Starting Fusion UI on port 8764
```

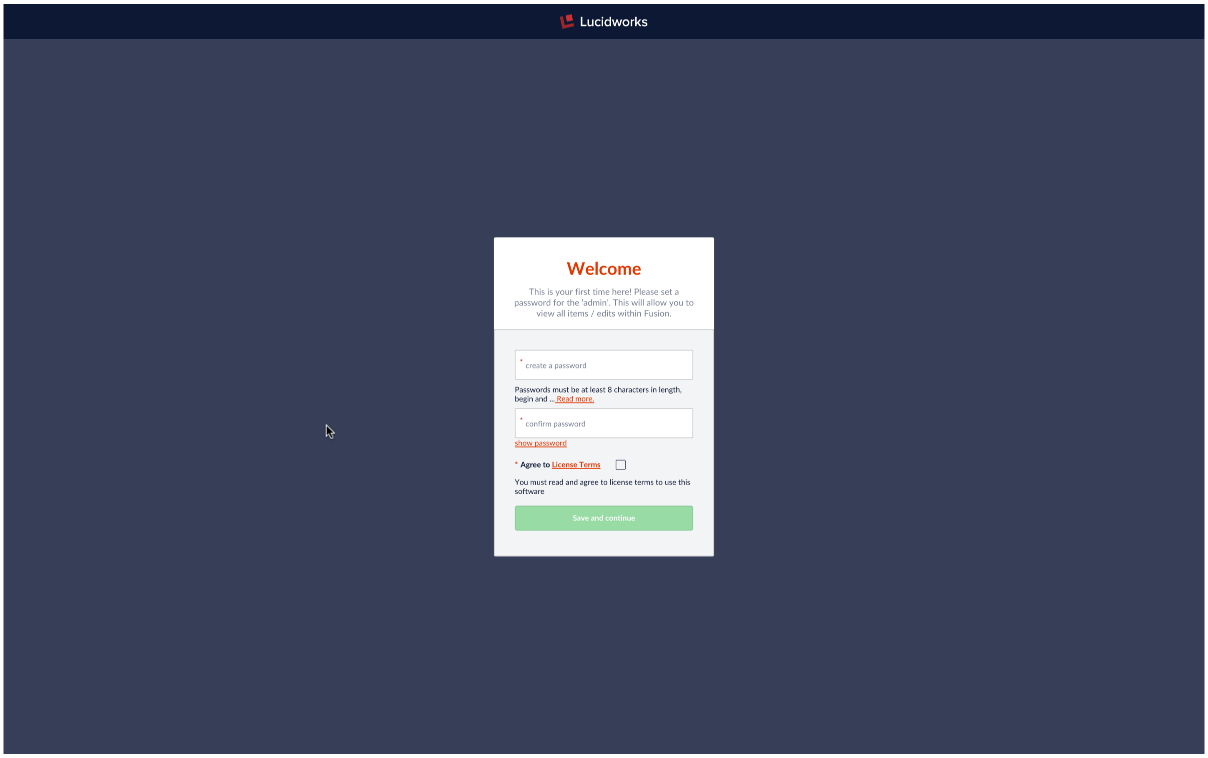
2.3. Create a Collection

The next step in getting started is to create a Fusion collection, the repository which stores your data. In this example, we create a collection called "sonnets" which will store Shakespeare’s sonnets.

Once Fusion is running, open a web browser and access the server and port that the Fusion UI is listening on. For a default single-node Fusion installation, the Fusion UI runs on port 8764 (as shown above), so the URL is: http://localhost:8764/.

Upon initial installation, when you first access the Fusion UI, it will present a sequence of panels:

- The initial login panel, URL: "\http://localhost:8764/initial-login". Set the Fusion admin password. Remember to check box "Agree to License Terms".



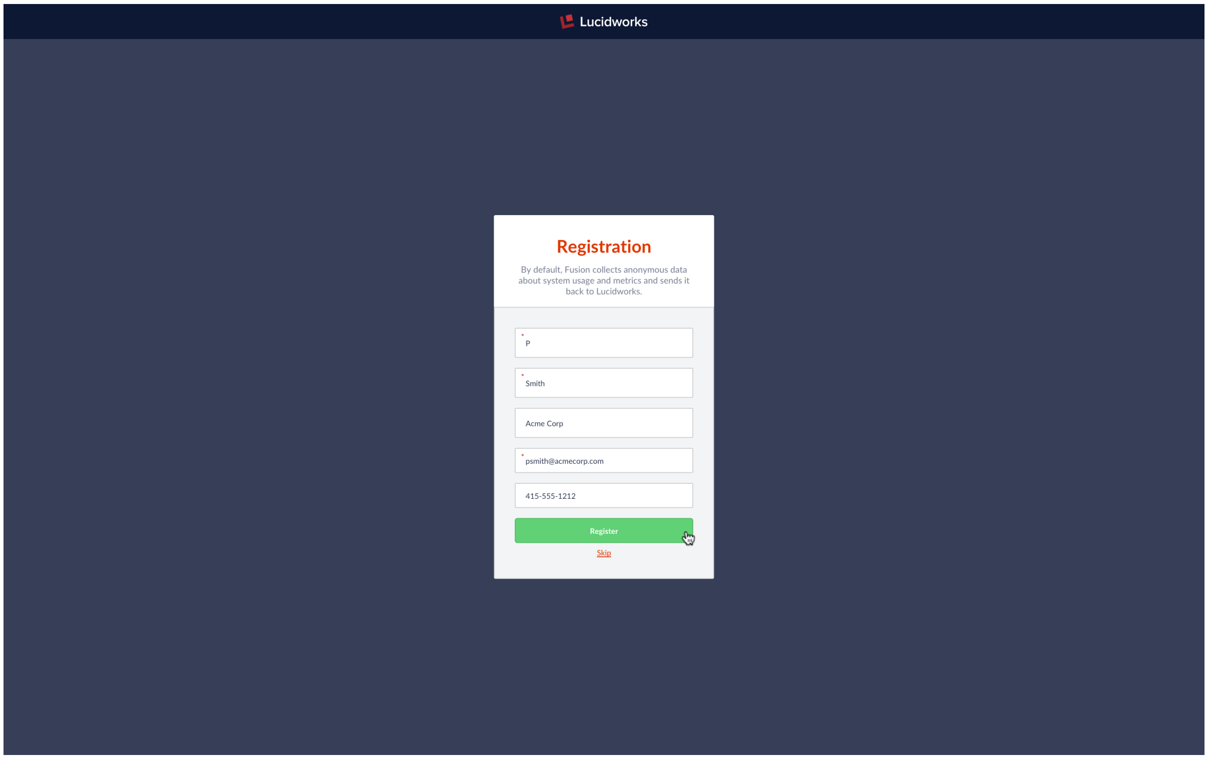
This is your first time here! Please set a password for the 'admin'. This will allow you to view all items / edits within Fusion.

Passwords must be at least 8 characters in length, begin and ... Read more.

* Agree to License Terms ☐

You must read and agree to license terms to use this software

- The registration panel, URL: "\http://localhost:8764/registration". Registration is optional. You can opt-out by clicking the "Skip" link at the bottom of this panel. Please see System Usage Monitor for information about how Fusion collects and uses this information.



By default, Fusion collects anonymous data about system usage and metrics and sends it back to Lucidworks.

* P

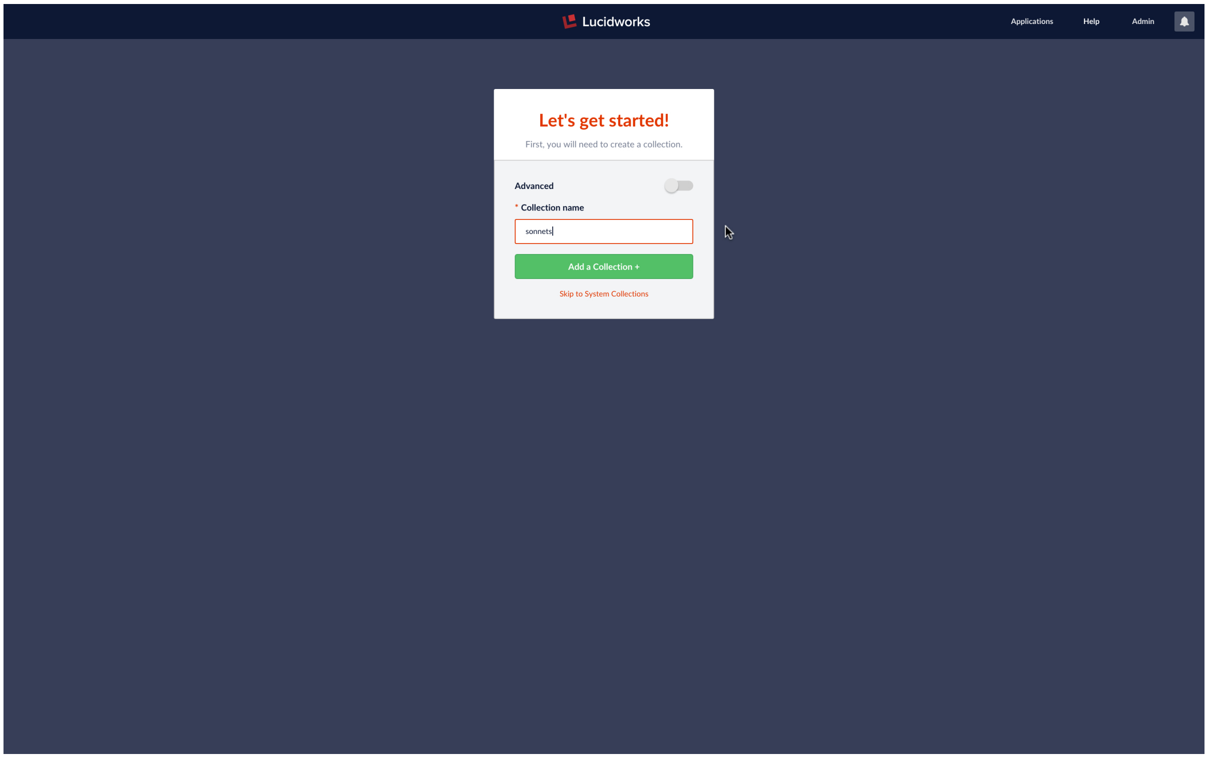
* Smith

Acme Corp

* psmith@acmecorp.com

415-555-1212

- The welcome panel, URL: "\http://localhost:8764/welcome". This panel prompts you to create a collection. When running a single-server developer instance of Fusion, the "Advanced" options don't apply, therefore, just enter the collection name. Here, we use the name "sonnets".



Let's get started!

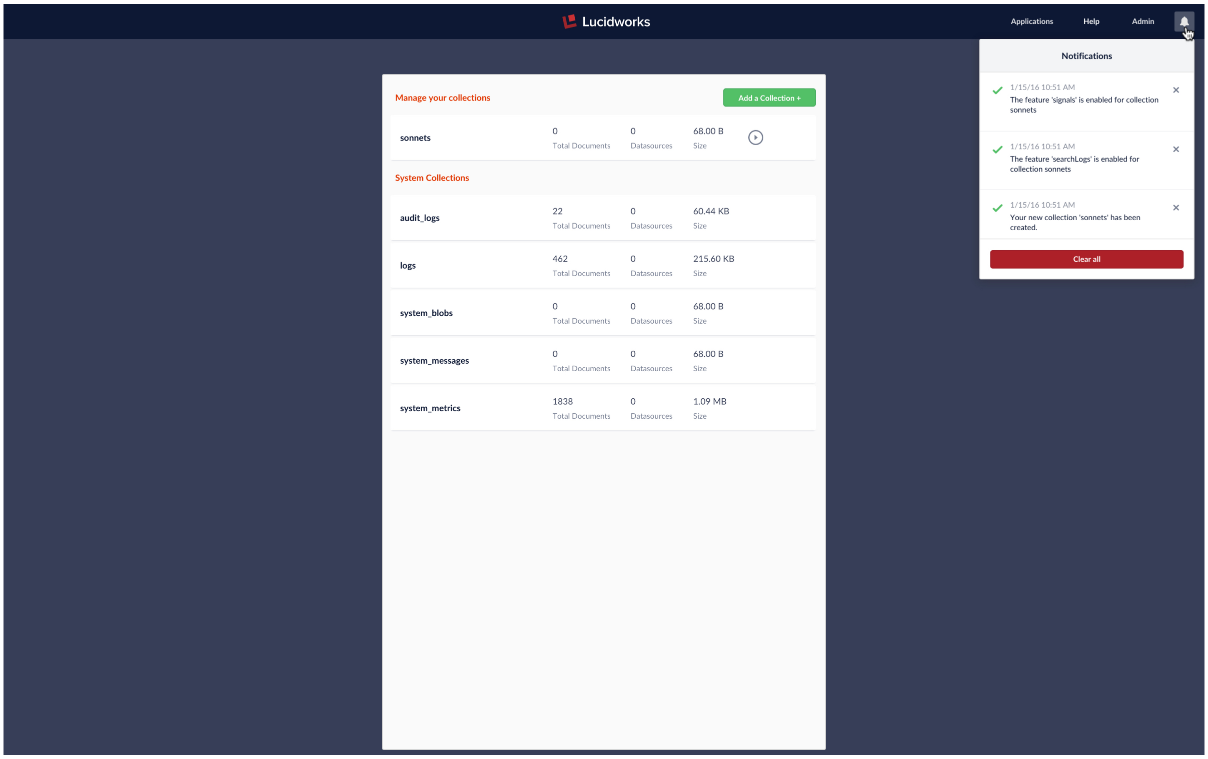
First, you will need to create a collection.

Advanced

* Collection name: sonnets

Note	Although Fusion will recognize collections named "TesT" and "tESt" as different collections, the filesystem on which the underlying Solr collection is stored may not. Therefore, avoid using letter case as the sole distinguishing feature for a collection name.

Upon successful create on collection "sonnets", Fusion displays the main collections panel.



Manage your collections

sonnets	0 Total Documents	0 Datasources	68.00 B Size

System Collections

audit_logs	22 Total Documents	0 Datasources	60.44 KB Size
logs	462 Total Documents	0 Datasources	215.60 KB Size
system_blobs	0 Total Documents	0 Datasources	68.00 B Size
system_messages	0 Total Documents	0 Datasources	68.00 B Size
system_metrics	1838 Total Documents	0 Datasources	1.09 MB Size

Notifications

1/15/16 10:51 AM
The feature 'signals' is enabled for collection sonnets

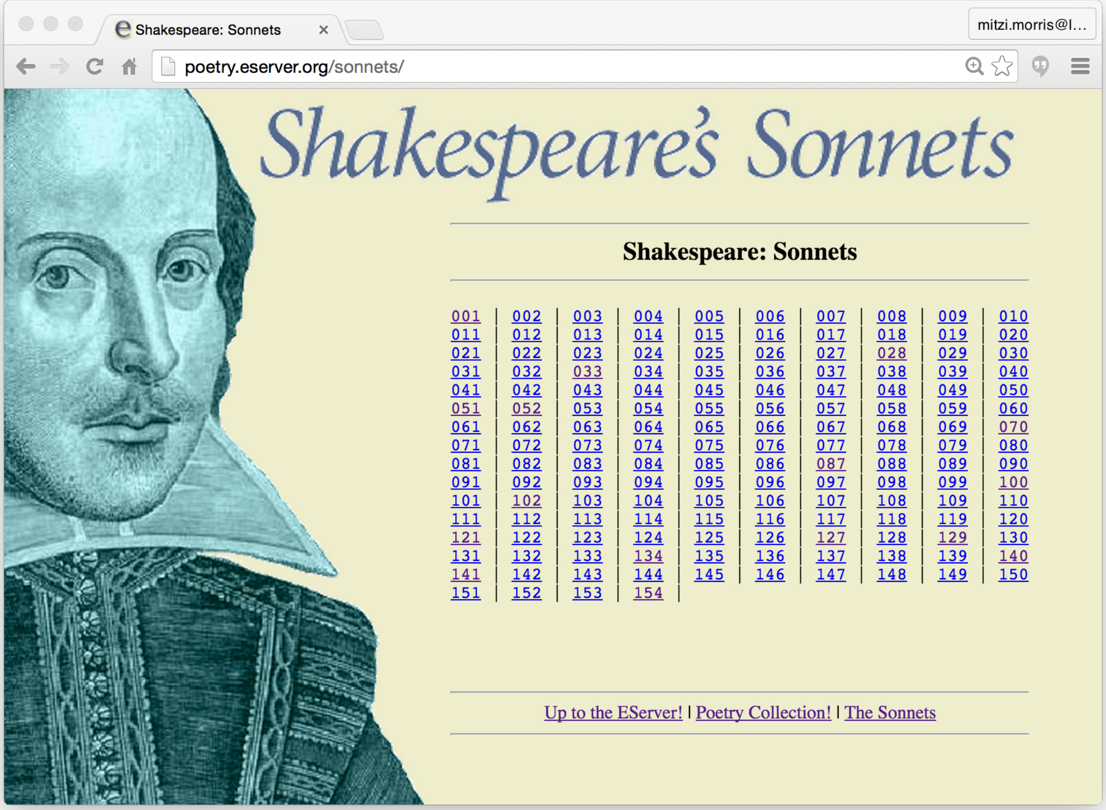
1/15/16 10:51 AM
The feature 'searchLogs' is enabled for collection sonnets

1/15/16 10:51 AM
Your new collection 'sonnets' has been created.

As the collection is created, the UI generates and stores a series of notifications. The rightmost icon on the Fusion top menu bar toggles display of these notifications.

2.4. Document Indexing with Datasources and Pipelines

The target dataset for this example is the website "Shakespeare: Sonnets", URL: "http://poetry.eserver.org/sonnets/":

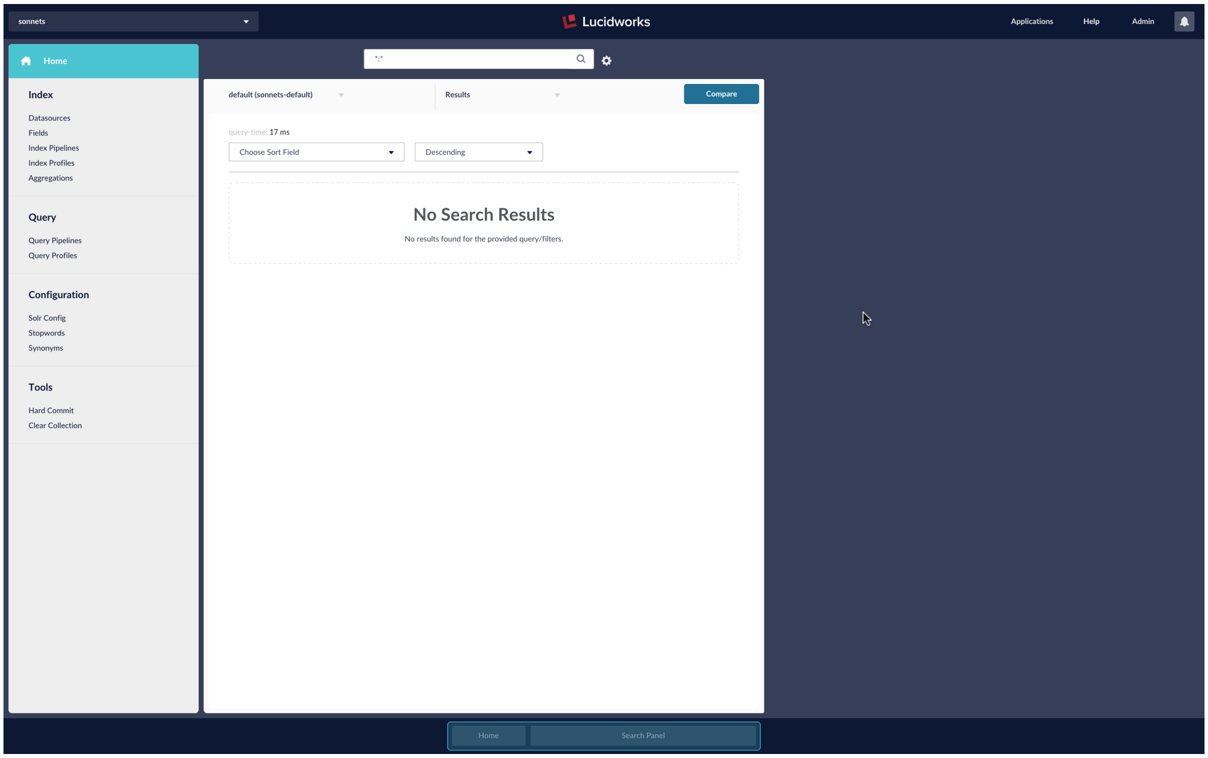


Shakespeare: Sonnets

001	002	003	004	005	006	007	008	009	010
011	012	013	014	015	016	017	018	019	020
021	022	023	024	025	026	027	028	029	030
031	032	033	034	035	036	037	038	039	040
041	042	043	044	045	046	047	048	049	050
051	052	053	054	055	056	057	058	059	060
061	062	063	064	065	066	067	068	069	070
071	072	073	074	075	076	077	078	079	080
081	082	083	084	085	086	087	088	089	090
091	092	093	094	095	096	097	098	099	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130
131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150
151	152	153	154						

Up to the EServer! | Poetry Collection! | The Sonnets

To do this, we need to configure and run a Fusion datasource over this collection. From the main collections panel (which is shown in previous screenshot), click on the name of the collection "sonnets", so that the UI now displays the sonnets collection home, URL "\http://localhost:8764/panels/sonnets". The default initial display for a collection has the collection "Home" panel on the left and the collection search panel next to it:

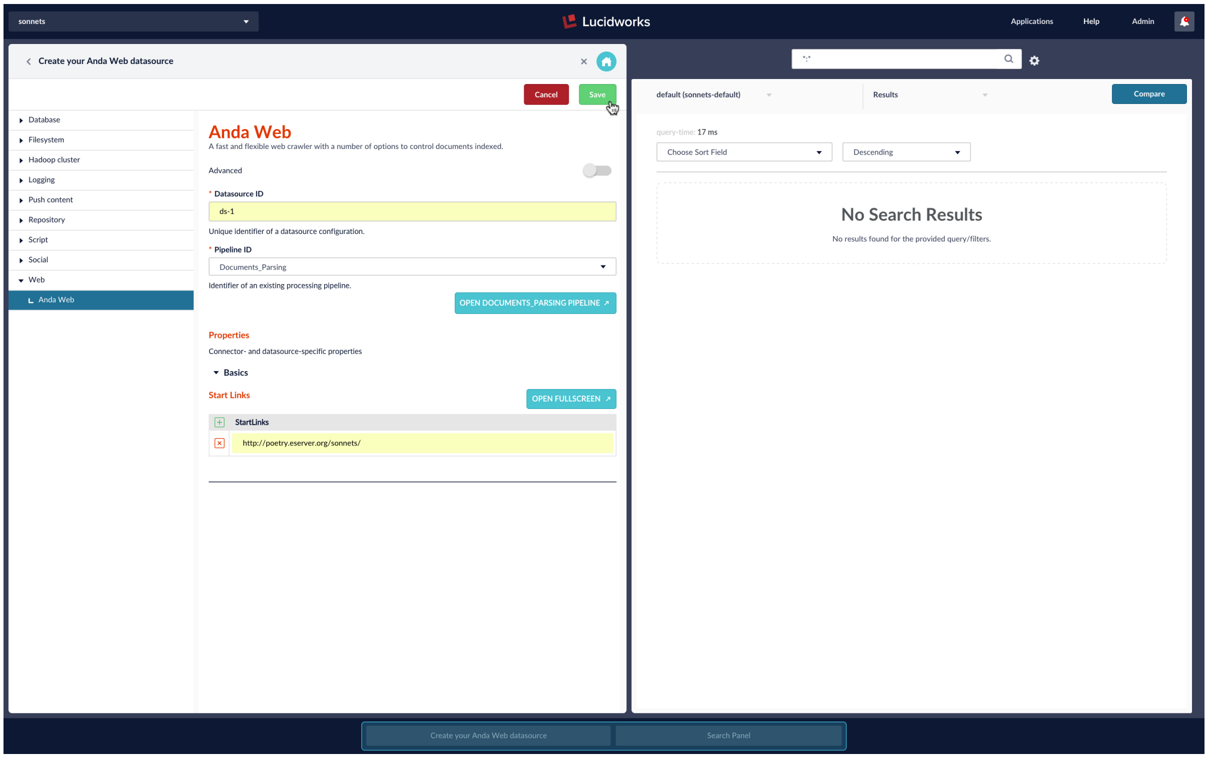


At the top of the search panel display is the search query input text box and control in the form of a gear icon which toggles display of the search results display controls. The search results display panel is underneath these elements. As collection "sonnets" doesn’t yet contain any documents, the search results panel is empty.

The narrow panel on the left is the collection "Home" panel. The Home panel contains the list of collection admin tools which are used for indexing, search, and system administration. It’s possible to have multiple home panels open at once, as needed to view configuration information. Clicking the Home icon at the top of the home panel will open a new home panel.

The collections admin tools under the "Index" heading are the controls for defining and using datasources and pipelines. Click the "Datasources" tool at the top of the Index section.

After clicking on "Datasources", the admin panel displays a choice of connector types. Because we want to get data from a website, we select **Web** > **Anda Web**. In configuring the web datasource, we only need to specify a datasource name, here "ds-1", and a starting URL, `http://poetry.eserver.org/sonnets/`:



Create your Anda Web datasource

Anda Web

A fast and flexible web crawler with a number of options to control documents indexed.

* Datasource ID: ds-1

Unique identifier of a datasource configuration.

* Pipeline ID: Documents_Parsing

Identifier of an existing processing pipeline.

Properties

Connector- and datasource-specific properties

Start Links

StartLinks: http://poetry.eserver.org/sonnets/

:

query-time: 17 ms

No Search Results

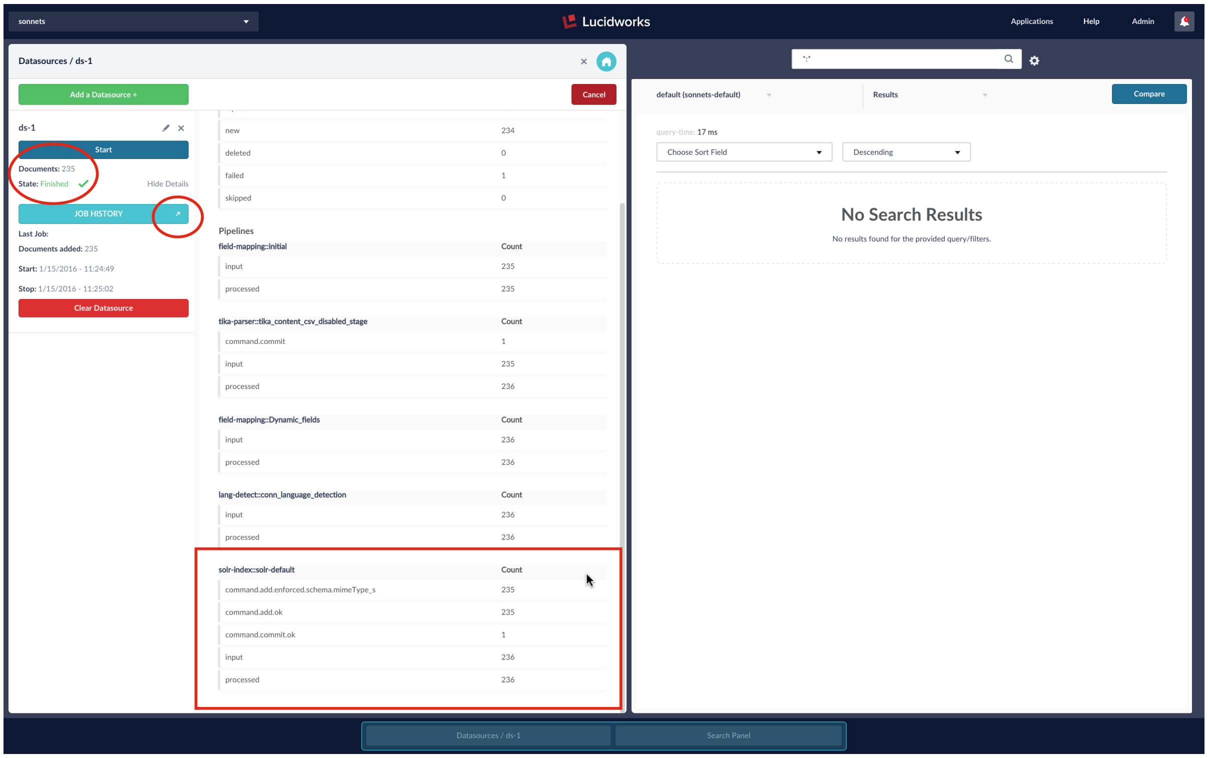
No results found for the provided query/filters.

Note	Configuration choices are only set/updated if you click on the red "Save" button at the bottom of this panel. Always scroll to the bottom of the panel and make sure that you have saved your work! Fusion displays a notification to confirm the datasource save.

Running the datasource will cause Fusion to fire up the connector, which will retrieve documents from the eserver.org website. Each webpage is treated as a separate document. The datasource hands off each document to the index pipeline. Here we ran the Anda Web datasource with index pipeline "Documents_Parsing", which is the default pipeline for this datasource. The "Documents_Parsing" pipeline consists of the following processing stages:

- Apache Tika Parser - recognizes and parses most common document formats, including HTML
- Field Mapping - transforms field names to valid Solr field names, as needed
- Detect Language - transforms text field names based on language of field contents
- Solr Indexer - transforms Fusion index pipeline document into Solr document and adds (or updates) document to collection.

To run the datasource, simply click the "Start" button on the Datasource panel. While the job is running, this control changes to "Stop / Abort", when the job is finished, the control changes back to a "Start" button and the job status is displayed directly below the "Start" button. To inspect the results, click on the control which toggles "show/hide" details. This opens an adjoining panel which lists each run of the datasource job. Each job has a small control that expands the job listing into a detailed listing the results of the pipeline processing stage:



Documents: 235
State: Finished ✓

Last Job:
Documents added: 235
Start: 1/15/2016 - 11:24:49
Stop: 1/15/2016 - 11:25:02

new	234
deleted	0
failed	1
skipped	0

Pipelines

field-mapping::initial	Count
input	235
processed	235

tika-parser::tika_content_csv_disabled_stage	Count
command.commit	1
input	235
processed	236

field-mapping::Dynamic_fields	Count
input	236
processed	236

lang-detect::conn_language_detection	Count
input	236
processed	236

solr-index::solr-default	Count
command.add.enforced.schema.mimeType_s	235
command.add.ok	235
command.commit.ok	1
input	236
processed	236

:

default (sonnets-default)

Results

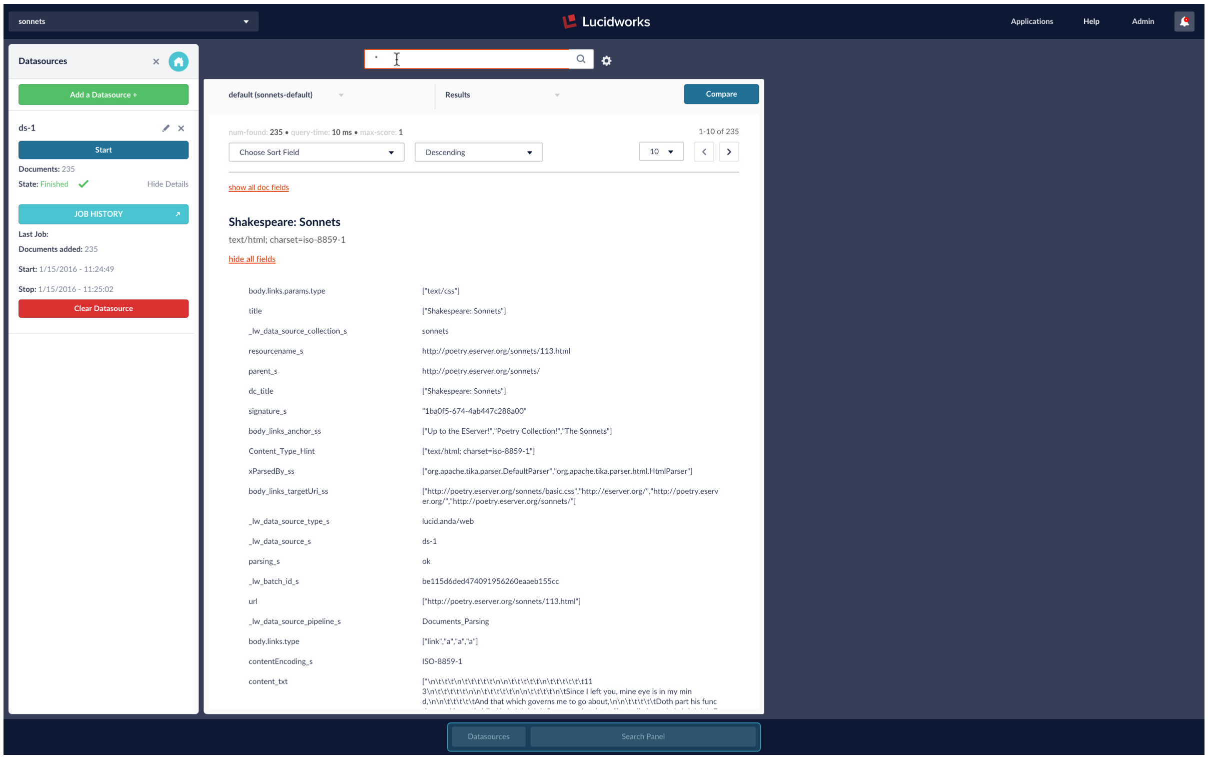
query-time: 17 ms

No Search Results

No results found for the provided query/filters.

2.5. Search

Once a datasource has been configured and the indexing job is complete, the collection can be searched using the search results tool. The wildcard query "*" matches all documents in the collection. Here is the result of running this search, showing all fields in one document:



Documents: 235
State: Finished ✓

Last Job:
Documents added: 235
Start: 1/15/2016 - 11:24:49
Stop: 1/15/2016 - 11:25:02

*

default (sonnets-default)

Results

num-found: 235 • query-time: 10 ms • max-score: 1

1-10 of 235

Shakespeare: Sonnets

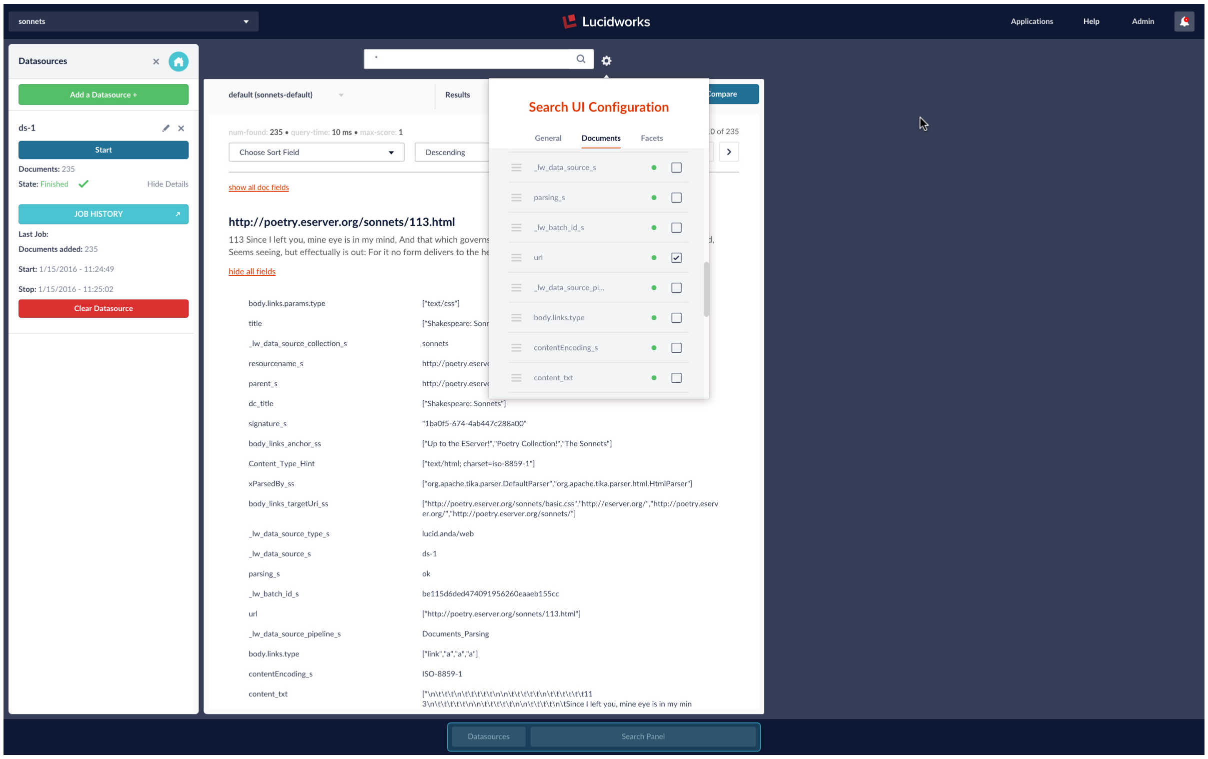
text/html; charset=iso-8859-1

body.links.params.type	["text/css"]
title	["Shakespeare: Sonnets"]
_lw_data_source_collection_s	sonnets
resourcename_s	http://poetry.eserver.org/sonnets/113.html
parent_s	http://poetry.eserver.org/sonnets/
dc_title	["Shakespeare: Sonnets"]
signature_s	"1ba0f5-674-4ab447c288a00"
body_links_anchor_ss	["Up to the EServer!","Poetry Collection!","The Sonnets"]
Content_Type_Hint	["text/html; charset=iso-8859-1"]
xParsedBy_ss	["org.apache.tika.parser.DefaultParser","org.apache.tika.parser.html.HtmlParser"]
body_links_targetUri_ss	["http://poetry.eserver.org/sonnets/basic.css","http://eserver.org/","http://poetry.eserver.org/","http://poetry.eserver.org/sonnets/"]
_lw_data_source_type_s	lucid.anda/web
_lw_data_source_s	ds-1
parsing_s	ok
_lw_batch_id_s	be115d6ded474091956260eaaeb155cc
url	["http://poetry.eserver.org/sonnets/113.html"]
_lw_data_source_pipeline_s	Documents_Parsing
body.links.type	["link","a","a","a"]
contentEncoding_s	ISO-8859-1
content_txt	["\n\t\t\t\n\t\t\t\t\t\n\n\t\t\t\t\t\n\t\t\t\t\t11 3\n\t\t\t\t\t\n\n\t\t\t\t\t\n\n\t\t\t\t\n\tSince I left you, mine eye is in my mind,\n\n\t\t\t\t\tAnd that which governs me to go about,\n\n\t\t\t\t\tDoth part his func

In order to use the search results tool to examine the documents in this collection, we need to configure the search results tool to show only those fields we care about. In this case the field "url" shows the sonnet number, and the field "content_txt" shows the raw text contents extracted from the HTML page.

Clicking on the gear icon next to the search box toggles the Search Results Configuration Tool open and close. The get a compact display of search results over this dataset, you should:

- Toggle the configuration tool open by clicking the gear icon.
- Choose the tab "Documents".
- Select the display "Primary".
- Uncheck all fields above field "URL".
- Select the display "Secondary".
- Uncheck all fields above the field "content_txt".
- Scroll down to the bottom of the control and click "Save".
- Toggle the configuration tool closed by clicking the gear icon.



ds-1

Documents: 235

State: Finished ✓ Hide Details

Last Job:

Documents added: 235

Start: 1/15/2016 - 11:24:49

Stop: 1/15/2016 - 11:25:02

default (sonnets-default)

num-found: 235 • query-time: 10 ms • max-score: 1

0 of 235

Search UI Configuration

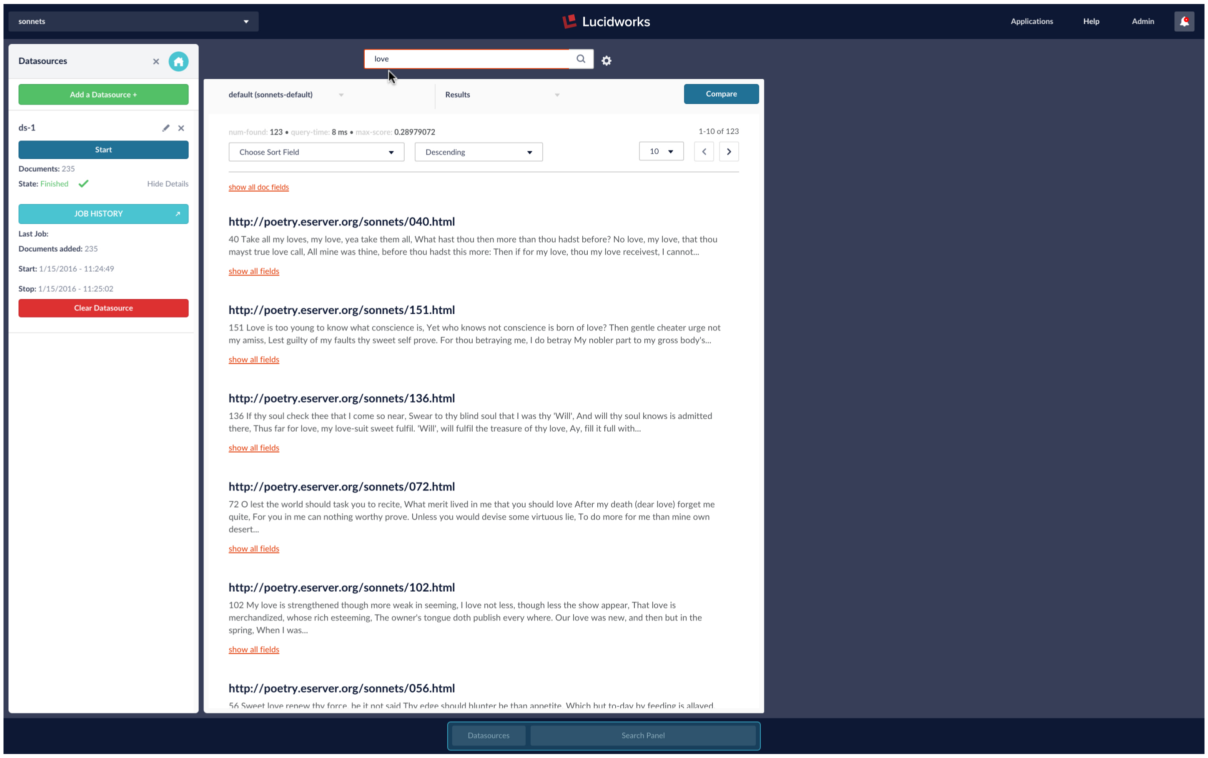
- _lw_data_source_s ● □
- parsing_s ● □
- _lw_batch_id_s ● □
- url ● ☑
- _lw_data_source_pi... ● □
- body.links.type ● □
- contentEncoding_s ● □
- content_txt ● □

http://poetry.eserver.org/sonnets/113.html

113 Since I left you, mine eye is in my mind, And that which governs
Seems seeing, but effectually is out: For it no form delivers to the he

body.links.params.type	["text/css"]
title	["Shakespeare: Sonr
_lw_data_source_collection_s	sonnets
resourcename_s	http://poetry.eserve
parent_s	http://poetry.eserve
dc_title	["Shakespeare: Sonnets"]
signature_s	"1ba0f5-674-4ab447c288a00"
body_links_anchor_ss	["Up to the EServer!","Poetry Collection!","The Sonnets"]
Content_Type_Hint	["text/html; charset=iso-8859-1"]
xParsedBy_ss	["org.apache.tika.parser.DefaultParser","org.apache.tika.parser.html.HtmlParser"]
body_links_targetUri_ss	["http://poetry.eserver.org/sonnets/basic.css","http://eserver.org/","http://poetry.eserver.org/","http://poetry.eserver.org/sonnets/"]
_lw_data_source_type_s	lucid.anda/web
_lw_data_source_s	ds-1
parsing_s	ok
_lw_batch_id_s	be115d6ded474091956260eaaeb155cc
url	["http://poetry.eserver.org/sonnets/113.html"]
_lw_data_source_pipeline_s	Documents_Parsing
body.links.type	["link","a","a","a"]
contentEncoding_s	ISO-8859-1
content_txt	["\n\t\t\t\n\t\t\t\t\t\n\n\t\t\t\t\t\n\t\t\t\t\t11 3\n\t\t\t\t\t\n\n\t\t\t\t\t\n\n\t\t\t\t\n\tSince I left you, mine eye is in my min

With this configuration in place, a search on the word "love" returns the following result:



love

Documents: 235
State: Finished ✓ Hide Details

Last Job:
Documents added: 235
Start: 1/15/2016 - 11:24:49
Stop: 1/15/2016 - 11:25:02

num-found: 123 • query-time: 8 ms • max-score: 0.28979072
1-10 of 123

show all doc fields

http://poetry.eserver.org/sonnets/040.html
40 Take all my loves, my love, yea take them all, What hast thou then more than thou hadst before? No love, my love, that thou mayst true love call, All mine was thine, before thou hadst this more: Then if for my love, thou my love receivest, I cannot...
show all fields

http://poetry.eserver.org/sonnets/151.html
151 Love is too young to know what conscience is, Yet who knows not conscience is born of love? Then gentle cheater urge not my amiss, Lest guilty of my faults thy sweet self prove. For thou betraying me, I do betray My nobler part to my gross body's...
show all fields

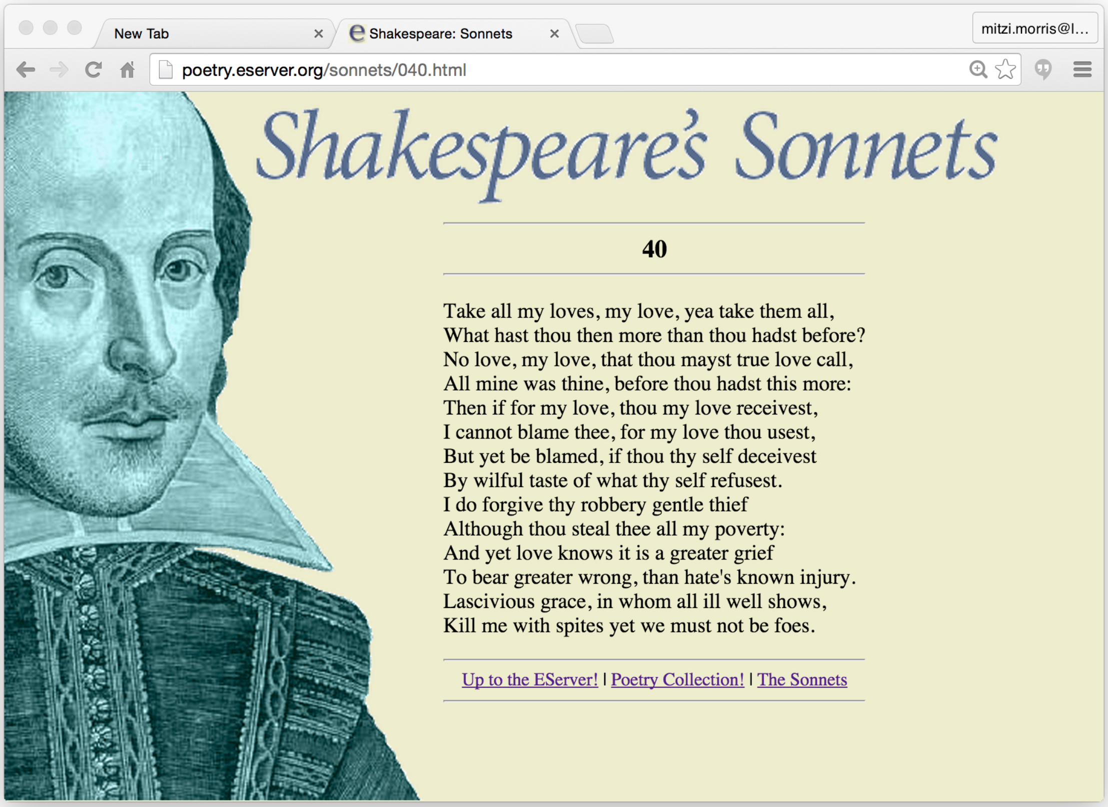
http://poetry.eserver.org/sonnets/136.html
136 If thy soul check thee that I come so near, Swear to thy blind soul that I was thy 'Will', And will thy soul knows is admitted there, Thus far for love, my love-suit sweet fulfil. 'Will', will fulfil the treasure of thy love, Ay, fill it full with...
show all fields

http://poetry.eserver.org/sonnets/072.html
72 O lest the world should task you to recite, What merit lived in me that you should love After my death (dear love) forget me quite, For you in me can nothing worthy prove. Unless you would devise some virtuous lie, To do more for me than mine own desert...
show all fields

http://poetry.eserver.org/sonnets/102.html
102 My love is strengthened though more weak in seeming, I love not less, though less the show appear, That love is merchandized, whose rich esteeming, The owner's tongue doth publish every where. Our love was new, and then but in the spring, When I was...
show all fields

http://poetry.eserver.org/sonnets/056.html
56 Sweet love renew thy force, be it not said Thy edge should blunter be than appetite, Which but to-day by feeding is allayed,

The most relevant document is Sonnet 40:


New Tab
Shakespeare: Sonnets
mitzi.morris@l...
poetry.eserver.org/sonnets/040.html
Shakespeare's Sonnets
Up to the EServer! | Poetry Collection! | The Sonnets

40

Take all my loves, my love, yea take them all,
What hast thou then more than thou hadst before?
No love, my love, that thou mayst true love call,
All mine was thine, before thou hadst this more:
Then if for my love, thou my love receivest,
I cannot blame thee, for my love thou usest,
But yet be blamed, if thou thy self deceivest
By wilful taste of what thy self refusest.
I do forgive thy robbery gentle thief
Although thou steal thee all my poverty:
And yet love knows it is a greater grief
To bear greater wrong, than hate's known injury.
Lascivious grace, in whom all ill well shows,
Kill me with spites yet we must not be foes.

It contains eight instances of the word "love", and one instance of "loves", more than in any other sonnet. To understand what we mean by relevancy, it is necessary to understand Lucene and Solr.

2.6. Lucene and Solr

Underlyingly, Fusion collections are Solr collections and Solr collections are comprised of Lucene indexes.

Lucene itself is a search API. Solr wraps Lucene in an web platform. Search and indexing are carried out via HTTP requests and responses. Solr generalizes the notion of a Lucene index to a Solr **collection**, a uniquely named, managed, and configured index which can be distributed ("sharded") and replicated across servers, allowing for scalability and high availability.

Lucene started out a search engine, designed to perform the following information retrieval task: given a set of query terms and a set of documents, find the subset of documents which are relevant for that query. Lucene provides a rich query language which allows for writing complicated logical conditions. Lucene now encompasses much of the functionality of a traditional DBMS, both in the kinds of data it can handle and the transactional security it provides.

Lucene maps discrete pieces of information, e.g., words, dates, numbers, to the documents in which they occur. This map is called an inverted index because the keys are document elements and the values are document ids, in contrast to other kinds of datastores where document ids are used as a key and the values are the document contents. This indexing strategy means that search requires just one lookup on an inverted index, as opposed to a document oriented search which would require a large number of lookups, one per document. Lucene treats a document as a list of named, typed fields. For each document field, Lucene builds an inverted index that maps field values to documents.

With this inverted index, it is easy to compute relevancy based on the overall frequency of terms in documents. An extremely relevant document for a query is a document that contains more of the terms in the query relative to all the other documents in the collection, as seen in the above search example.

2.7. Further Reading

An expanded version of this getting started page is available on the Lucidworks blog as a two-part series:

Search Basics for Data Engineers

Preliminary Data Analysis with Fusion

The Lucidworks blog also has a series of articles on getting started with Signals in Fusion:

Signals Basics for Data Engineers

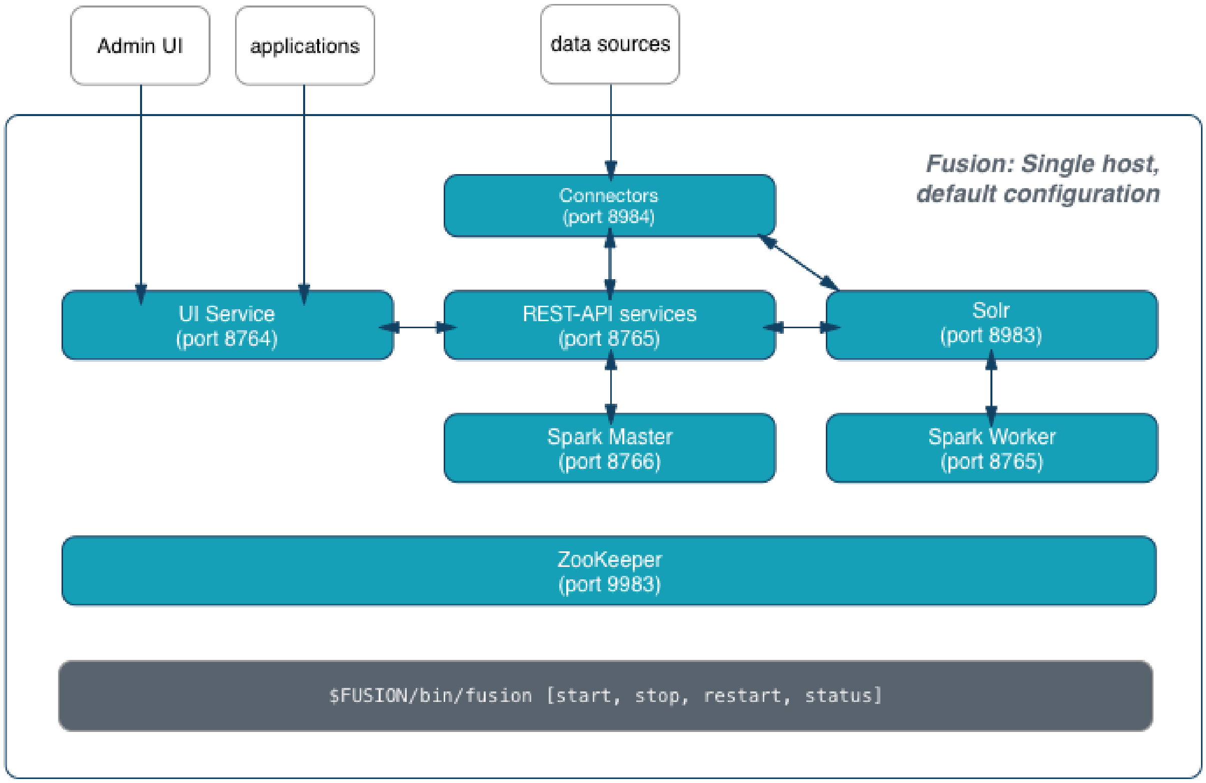
Better Search with Fusion Signals

Chapter 3. Fusion Architecture

The Fusion platform is designed to support enterprise search applications at any scale. Fusion can be deployed across multiple servers in order to store large amounts of data or to achieve high processing throughput or both, and the set of Fusion components running on each server can be adjusted to meet these processing requirements.

3.1. Fusion Platform Component Architecture

The Fusion platform is comprised of a series of Java programs, each of which runs in its own JVM. Apache ZooKeeper provides the shared, synchronized data store for all user and application configuration information. The following diagram shows the full set of Fusion processes that run on a single server and the default ports used by each, with arrows representing the flow of HTTP requests between components for document search and indexing:



Search queries, which originate from the search application, are sent to the Fusion UI for authentication. The Fusion UI sends the request to the Fusion API services component, which invokes the a query pipeline to build out the raw query and send the resulting query to Solr. Document indexing is carried out by Fusion datasources which send raw data to Fusion's connector services. The connector invokes an index pipeline to extract, transform, and otherwise enrich the raw data and sends the resulting document to Solr for indexing. Apache Spark is used for signal processing and aggregation. All Fusion processes across all servers in a Fusion deployment communicate with the ZooKeeper ensemble at the socket layer using ZooKeeper's Java API.

See these topics for details about each component:

- Fusion UI
- UI Service
- Connectors

- REST API Services
- Solr
- Spark
- ZooKeeper
- bin/fusion

3.2. Fusion Platform Deployment Architecture

For Enterprise applications that consist of very large collections or that require high-throughput or high availability or both, the Fusion deployment will consist of multiple servers. Each server is a Fusion *node*. All nodes in a Fusion deployment communicate with a common ZooKeeper cluster.

Every Fusion node in a deployment runs the Fusion API Services process. Beyond that, the set of processes running on a particular node depends on the processing and throughput needs of the search application.

- Running Solr on all Fusion nodes scales out document storage as well as providing data replication. (Alternatively, external SolrCloud clusters can be used to store Fusion collections, see Integrating with existing Solr instances.)
- Running Fusion Connectors on multiple nodes provides high throughput for indexing and updates, e.g., applications which run analytics over live data streams such as logfile indexing or mobile tracking devices.
- Running the Fusion UI on two or more nodes provides failover for Fusion’s authentication proxy.
- Running Apache Spark on multiple nodes provides processing power for applications that aggregate clicks and other signals, or that use Fusion machine learning components.

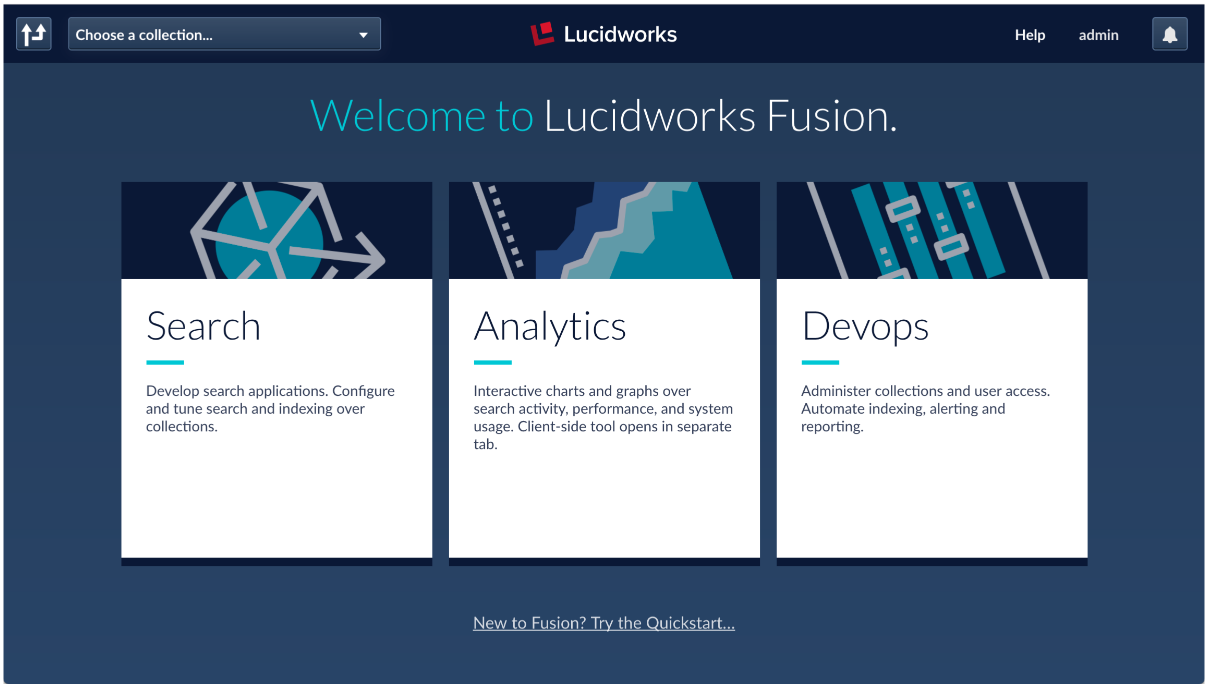
3.3. The Fusion UI

The Fusion UI provides a graphical interface to Fusion's REST APIs. To access it, point your browser to http://localhost:8764/, or the host on which the UI Service is running.

The first time you log in to the UI, you'll be prompted to create an admin account.

Note	Save your admin credentials. There is no password recovery system in Fusion.

Whenever you log in to Fusion (and it's not the first login), the Launcher lets you choose a context for your workflow:



Choose a collection...

Lucidworks

Help

admin

Welcome to Lucidworks Fusion.

Search

Develop search applications. Configure and tune search and indexing over collections.

Analytics

Interactive charts and graphs over search activity, performance, and system usage. Client-side tool opens in separate tab.

Devops

Administer collections and user access. Automate indexing, alerting and reporting.

New to Fusion? Try the Quickstart...

- The **Search** context provides all the tools you need to get your data into Fusion and develop a search-ready back end for your application.
- The **Analytics** context gives you access to Fusion's dashboard application, where you can customize your view into system metrics and search activity.
- Use the **Devops** context to administrate and monitor the Fusion system.

Since the Fusion UI is just an interface to the REST API service, you can also use the APIs to do almost anything that the UI can do.

3.4. UI service

The UI service is the interface between your applications and the REST API services. All HTTP requests from external clients must pass through the UI service, because it hosts the auth proxy. It also hosts the Web-based Fusion UI.

3.4.1. The auth proxy

The auth proxy generates the session cookie required by all REST API services. The REST API services accept no commands without this cookie. The auth proxy requires valid username/password credentials with every request, like this:

```
curl -u <user>:<pwd> localhost:8764/api/apollo/<servicename>
```

3.4.2. UI configuration

The default port is 8764, configurable as `UI_PORT` in `fusion/conf/config.sh`.

UI Web service configuration is in `fusion/apps/jetty/ui`.

This blog post explains how to secure the UI using SSL.

3.4.3. UI logs

UI service log files are in `fusion/var/log/ui`.

3.5. Connectors

Connectors are the built-in mechanism for pulling your data into Fusion. Fusion comes with a wide variety of connectors, each specialized for a particular data type. When you add a datasource to a collection, you specify the connector to use for ingesting data. See Connector Types for a complete list of connectors, with links to configuration reference information for each one.

3.5.1. Connector configuration

When you add a datasource to a collection, you select a connector and configure it. There are two ways to do this:

- Using the API
- Using the UI

Configuring Connectors Using The API

You can create or update a datasource with the Connector Datasources API, specifying the connector, its properties, and their values.

Example: *Create and configure a datasource to index Solr-formatted XML files*

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d '{"id":"SolrXML",
"connector":"lucid.solrxml", "type":"solrxml", "properties":{"path":"/Applications/solr-
4.10.2/example/exampledocs", "generate_unique_key":false, "collection":"MyCollection"}}'
http://localhost:8764/api/apollo/connectors/datasources
```

See the Connectors and Datasources Reference for details about configuration options.

Tip	Be sure the include the `collection` property; otherwise the datasource will not be available in the Fusion UI.

Example: *Change the `max_docs` value for the above datasource*

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d '{"id":"SolrXML", "connector":"lucid.solrxml",
"type":"solrxml", "properties":{"path":"/Applications/solr-4.10.2/example/exampledocs", "max_docs":10}}'
http://localhost:8764/api/apollo/connectors/datasources/SolrXML
```

Configuring Connectors Using The Fusion UI

- To create and configure a new datasource and its connector:

  1. Click **Applications** > **Collections**.
  2. Select a Collection, or click **Add a Collection** to create a new one.
  3. Click **Add a Datasource**.
  4. Select a datasource type; these correspond to Fusion’s Connectors.
  5. Edit the configuration fields in the datasource panel that appears.

     See the Connectors and Datasources Reference for details about configuration options.

6. Click **Save**.

- To change the connector configuration for an existing datasource:

  1. Navigate to **Applications** > **Collections**.
  2. Click your collection name.
  3. Click the datasource you want to change.
  4. Edit the configuration fields as needed.
  5. Click **Save**.

3.5.2. Connector logs

You can find connector logs in `fusion/var/log/connectors`.

3.6. REST API Services

Fusion includes many REST APIs, all described in detail in the REST API Reference. Almost every aspect of Fusion can be controlled via the APIs.

The Fusion UI provides another administrative interface to Fusion, using a subset of the REST APIs.

3.6.1. API Service configuration

The Web service for the APIs is configured in `fusion/apps/jetty/api`.

3.6.2. API Service logs

API log files are in `fusion/var/log/api`.

3.7. Solr

Solr is the search platform that powers Fusion. There are multiple aspects to Fusion's use of Solr:

- Fusion components manage Solr search and indexing and provide analytics over these collections. Fusion's analytics components depend on aggregations over information which is stored in a Solr collection.
- Fusion collections are all Solr collections.
- Application data is stored as one or more Solr collections.
- Fusion's own logs are stored as Solr collections.
- A few Fusion service APIs use Solr as a backing store, notably Parameter Sets.

3.7.1. Solr configuration

Fusion requires that Solr run with SolrCloud enabled.

Configuration for Solr's Web service is in `fusion/apps/jetty/solr`.

3.7.2. Solr logs

Solr log files are in `fusion/var/log/solr`.

3.7.3. Accessing the Solr UI

With Fusion installed out of the box, you can still access the Solr UI at http://localhost:8983/solr/.

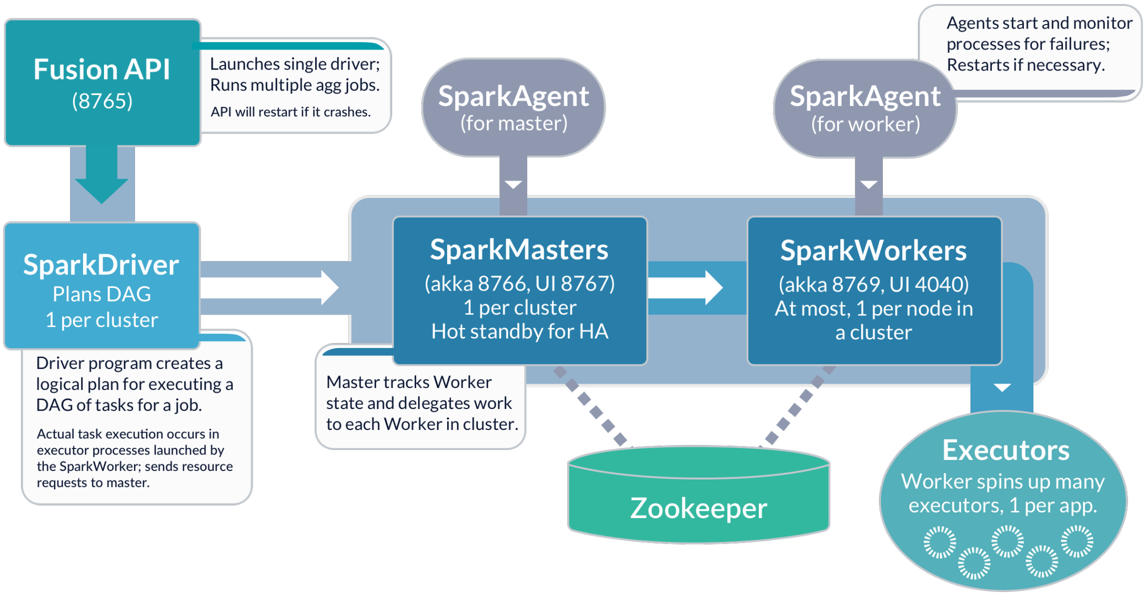
3.7.4. Solr documentation

Solr documentation and additional resources are available at http://lucene.apache.org/solr/resources.html.

You can also find plenty of Solr tips and technical discussions in our knowledge base, blog, and webinars. Lucidworks also maintains a search interface to Solr's community discussions at searchhub.org.

3.8. Apache Spark

Apache Spark is a fast and general execution engine for large-scale data processing jobs that can be decomposed into stepwise tasks which are distributed across a cluster of networked computers. Spark provides faster processing and better fault-tolerance than previous MapReduce implementations. The following schematic shows the Spark components available from Fuson:


Fusion API
(8765)
Launches single driver;
Runs multiple agg jobs.
API will restart if it crashes.
SparkAgent
(for master)
SparkAgent
(for worker)
Agents start and monitor processes for failures; Restarts if necessary.
SparkDriver
Plans DAG
1 per cluster
SparkMasters
(akka 8766, UI 8767)
1 per cluster
Hot standby for HA
SparkWorkers
(akka 8769, UI 4040)
At most, 1 per node in a cluster
Driver program creates a logical plan for executing a DAG of tasks for a job.
Actual task execution occurs in executor processes launched by the SparkWorker; sends resource requests to master.
Master tracks Worker state and delegates work to each Worker in cluster.
Zookeeper
Executors
Worker spins up many executors, 1 per app.

Fusion 2.1 introcuced the use of a Spark cluster for all signal aggregation processes. This use of Spark is managed entirely by Fusion and is hidden from view of a Fusion application developer or end user.

As of Fusion 2.4, the Spark Jobs API provides a set of REST API endpoints for configuring and running Spark jobs via Fusion.

The section Spark and Machine Learning covers how to use use Fusion’s Spark cluster to run your own Spark jobs for custom aggregations and Machine Learning tasks.

3.9. ZooKeeper

Apache ZooKeeper is a distributed configuration service, synchronization service, and naming registry.

Fusion uses ZooKeeper to configure and manage all Fusion components in a single Fusion deployment, therefore a ZooKeeper service must always be running as part of the Fusion deployment. For high availability, this should be an external 3-node ZooKeeper cluster. All Fusion Java components communicate with ZooKeeper using the ZooKeeper API.

For ZooKeeper installation instructions, see the ZooKeeper documentation.

You can find ZooKeeper's logs at `fusion/var/log/zookeeper`.

3.9.1. ZooKeeper Terminology

- **znode**: ZooKeeper data is organized into a hierarchal name space of data nodes called znodes. A znode can have data associated with it as well as child znodes. The data in a znode is stored in a binary format, but it is possible to import, export, and view this information as JSON data. Paths to znodes are always expressed as canonical, absolute, slash-separated paths; there are no relative reference.
- **ephemeral nodes**: An ephemeral node is a znode which exists only for the duration of an active session. When the session ends the znode is deleted. An ephemeral znode cannot have children.
- **server**: A ZooKeeper service consists of one or more machines; each machine is a server which runs in its own JVM and listens on its own set of ports. For testing, you can run several ZooKeeper servers at once on a single workstation by configuring the ports for each server.
- **quorum**: A quorum is a set of ZooKeeper servers. It must be an odd number. For most deployments, only 3 servers are required.
- **client**: A client is any host or process which uses a ZooKeeper service.

See the official ZooKeeper documentation for details about using and managing a ZooKeeper service.

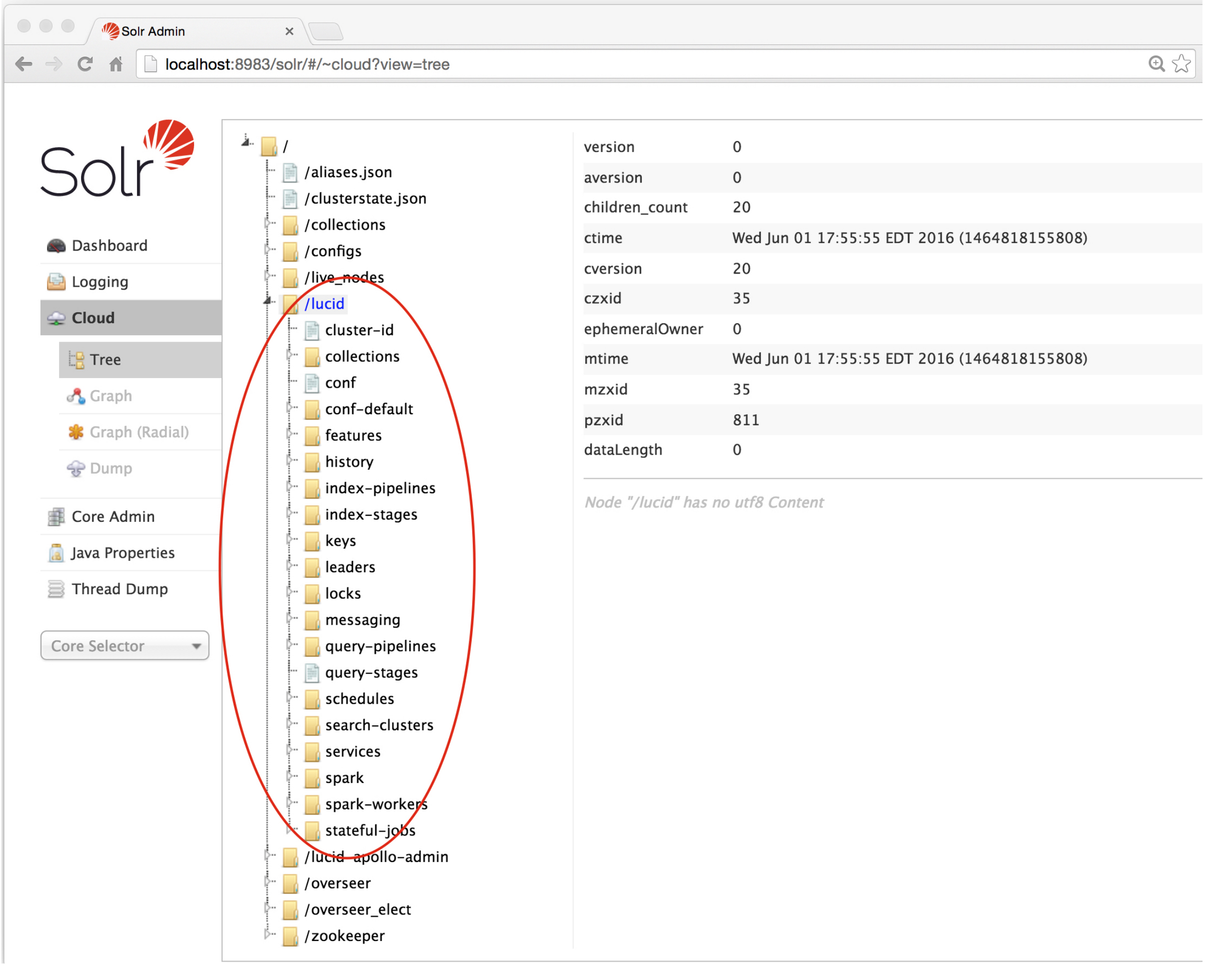
3.9.2. Fusion ZooKeeper Nodes

Fusion configuration data is stored in ZooKeeper under two znodes:

- Node `lucid` stores all application-specific configurations, including collection, datasource, pipeline, signals, aggregations, and associated scheduling, jobs, and metrics.
- Node `lucid-apollo-admin` stores all access control information, including all users, groups, roles, and realms.

The Solr Admin tool provides a ZooKeeper node browser tool. In the case of the Fusion default developer deployment, the Fusion runs scripts are configured to run the instances of both Solr and ZooKeeper which are included with the Fusion distribution, and therefore we take a fresh installation of a Fusion developer instance and use the embedded Solr's Admin tool to explore how Fusion's configurations are managed in ZooKeeper.

On initial install, the "lucid" znode contains the set of default configurations used by Fusion's services:

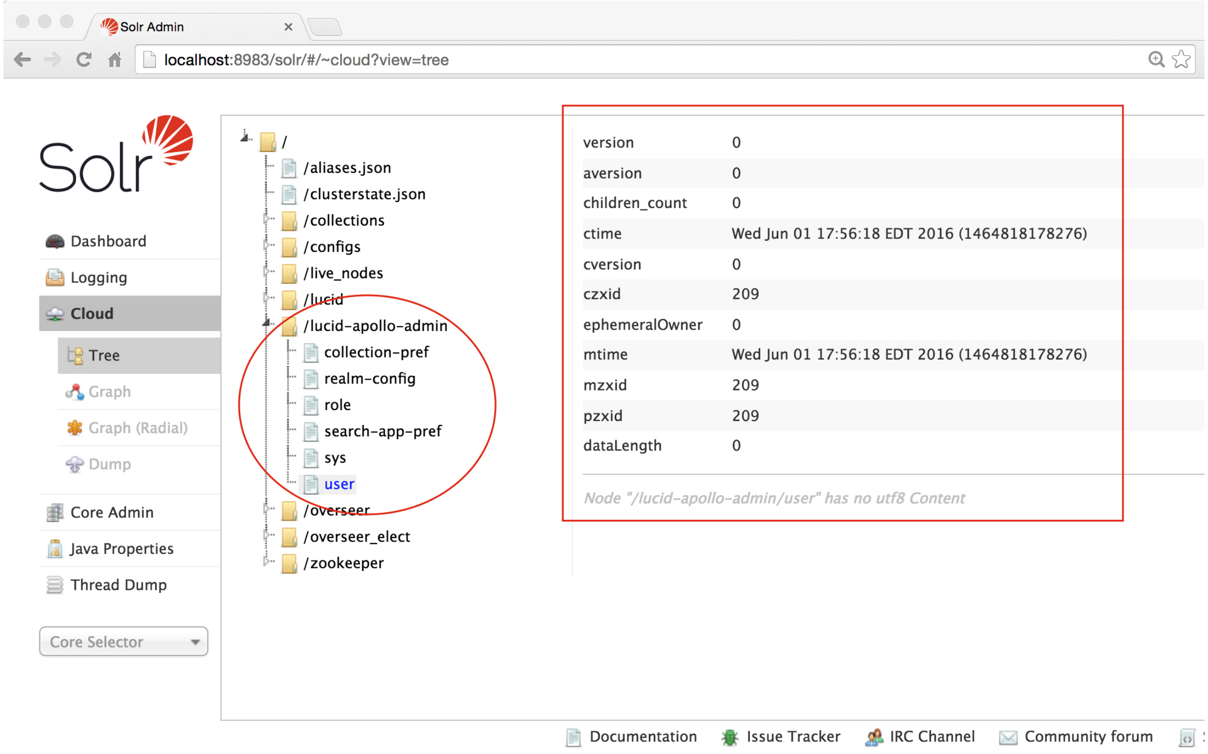


```
/
  /aliases.json
  /clusterstate.json
  /collections
  /configs
  /live_nodes
  /lucid
    cluster-id
    collections
    conf
    conf-default
    features
    history
    index-pipelines
    index-stages
    keys
    leaders
    locks
    messaging
    query-pipelines
    query-stages
    schedules
    search-clusters
    services
    spark
    spark-workers
    stateful-jobs
  /lucid-apollo-admin
  /overseer
  /overseer_elect
  /zookeeper
```

version	0
aversion	0
children_count	20
ctime	Wed Jun 01 17:55:55 EDT 2016 (1464818155808)
cversion	20
czxid	35
ephemeralOwner	0
mtime	Wed Jun 01 17:55:55 EDT 2016 (1464818155808)
mzxid	35
pzxid	811
dataLength	0

Node "/lucid" has no utf8 Content

The "lucid-apollo-admin" znode contains the set of nodes used by Fusion’s access control services:



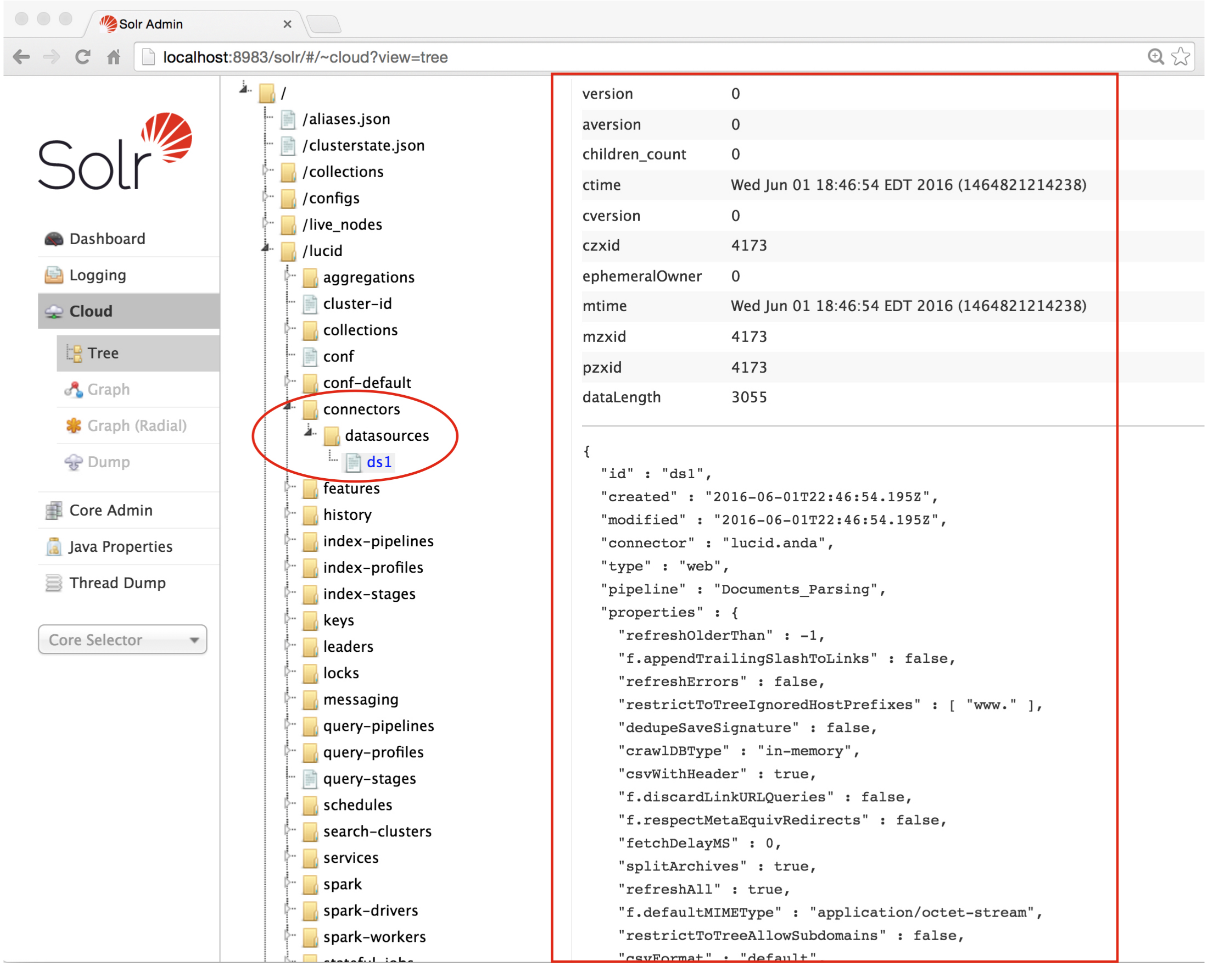
```
/
  /aliases.json
  /clusterstate.json
  /collections
  /configs
  /live_nodes
  /lucid
  /lucid-apollo-admin
    collection-pref
    realm-config
    role
    search-app-pref
    sys
    user
  /overseer
  /overseer_elect
  /zookeeper
```

version	0
aversion	0
children_count	0
ctime	Wed Jun 01 17:56:18 EDT 2016 (1464818178276)
cversion	0
czxid	209
ephemeralOwner	0
mtime	Wed Jun 01 17:56:18 EDT 2016 (1464818178276)
mzxid	209
pzxid	209
dataLength	0

Node "/lucid-apollo-admin/user" has no utf8 Content

In the above screenshot, the ZooKeeper node browser is browsing the contents of znode "lucid-apollo-admin/users" which is empty. The Fusion distribution ships without any user accounts. The initial user added to Fusion is the Fusion native realm "admin" user. This entry is only created on initial startup via the Fusion UI "set admin password" panel. Once you submit the admin password, the admin user account is created. Until Fusion contains as least the admin user account, you cannot use the system, because all Fusion requests require proper authorization.

Once the admin password is set, and you have created one or more Fusion collections and have populated them by running one or more datasources, these collections, datasources, pipelines, and other application configuration settings are stored under the "lucid" znode:

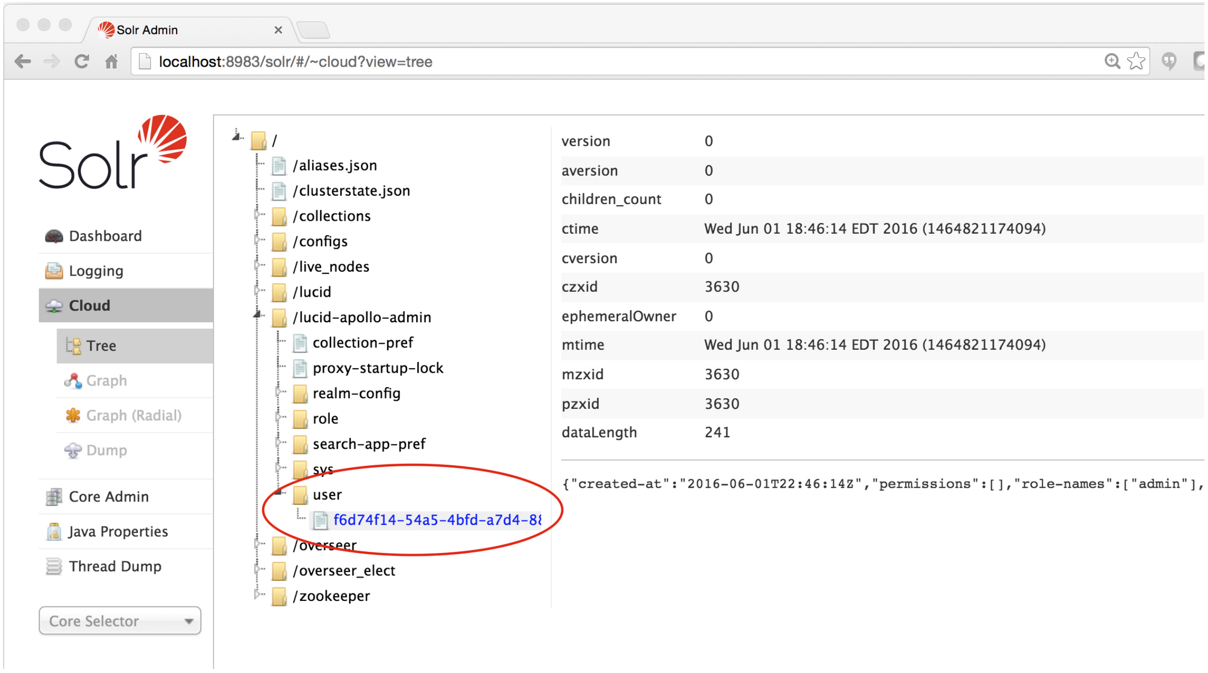


version	0
aversion	0
children_count	0
ctime	Wed Jun 01 18:46:54 EDT 2016 (1464821214238)
cversion	0
czxid	4173
ephemeralOwner	0
mtime	Wed Jun 01 18:46:54 EDT 2016 (1464821214238)
mzxid	4173
pzxid	4173
dataLength	3055

```
{
  "id" : "ds1",
  "created" : "2016-06-01T22:46:54.195Z",
  "modified" : "2016-06-01T22:46:54.195Z",
  "connector" : "lucid.anda",
  "type" : "web",
  "pipeline" : "Documents_Parsing",
  "properties" : {
    "refreshOlderThan" : -1,
    "f.appendTrailingSlashToLinks" : false,
    "refreshErrors" : false,
    "restrictToTreeIgnoredHostPrefixes" : [ "www." ],
    "dedupeSaveSignature" : false,
    "crawlDBType" : "in-memory",
    "csvWithHeader" : true,
    "f.discardLinkURLQueries" : false,
    "f.respectMetaEquivRedirects" : false,
    "fetchDelayMS" : 0,
    "splitArchives" : true,
    "refreshAll" : true,
    "f.defaultMIMEType" : "application/octet-stream",
    "restrictToTreeAllowSubdomains" : false,
    "csvFormat" : "default",
```

In the above screenshot, the ZooKeeper node browser is browsing the contents of znode "lucid/connectors/datasources/ds1". This datasource was used to populate a Fusion collection with documents retrieved via a webcrawl. Note that in the initial screenshot for znode "lucid", there is no "connectors" node at all.

The "lucid-apollo-admin" znode now contains one user accounts for user "admin":



```
version          0
aversion         0
children_count   0
ctime            Wed Jun 01 18:46:14 EDT 2016 (1464821174094)
cversion         0
czxid            3630
ephemeralOwner   0
mtime            Wed Jun 01 18:46:14 EDT 2016 (1464821174094)
mzxid            3630
pzxid            3630
dataLength       241
```

{"created-at":"2016-06-01T22:46:14Z","permissions":[],"role-names":["admin"],

3.10. bin/fusion

For every server in a Fusion deployment, the script `fusion/bin/fusion` is used to start, stop, and check the status of the Fusion instance running on that server.

3.10.1. Start Fusion

Run the script `fusion/bin/fusion` with the argument `start`:

```
$ cd /path/to/fusion
$ ./bin/fusion start
2016-10-04 18:52:04Z Starting Fusion ZooKeeper on port 9983
2016-10-04 18:52:14Z Starting Fusion Solr on port 8983
2016-10-04 18:52:39Z Starting Fusion Spark Master on port 8766
2016-10-04 18:52:39Z Starting Fusion Spark Worker on port 8769
2016-10-04 18:52:39Z Starting Fusion API Services on port 8765
2016-10-04 18:52:45Z Starting Fusion Connectors on port 8984
2016-10-04 18:52:50Z Starting Fusion UI on port 8764
```

3.10.2. Check the status of Fusion

Run the script `fusion/bin/fusion` with the argument `status`:

```
$ cd /path/to/fusion
$ ./bin/fusion status
2016-10-04 20:14:43Z process 2525 from pid file /home/ubuntu/fusion-2.4.1/var/connectors/connectors.pid is running
2016-10-04 20:14:43Z process 2230 from pid file /home/ubuntu/fusion-2.4.1/var/api/api.pid is running
2016-10-04 20:14:43Z process 1962 from pid file /home/ubuntu/fusion-2.4.1/var/spark-master/spark-master.pid is running
2016-10-04 20:14:43Z process 2072 from pid file /home/ubuntu/fusion-2.4.1/var/spark-worker/spark-worker.pid is running
2016-10-04 20:14:43Z process 2708 from pid file /home/ubuntu/fusion-2.4.1/var/ui/ui.pid is running
2016-10-04 20:14:43Z process 1793 from pid file /home/ubuntu/fusion-2.4.1/var/solr/solr.pid is running
2016-10-04 20:14:43Z process 1644 from pid file /home/ubuntu/fusion-2.4.1/var/zookeeper/zookeeper.pid is running
```

3.10.3. Stop Fusion

Run the script `fusion/bin/fusion` with the argument `stop`:

```
$ cd /path/to/fusion
$ ./bin/fusion stop
2016-10-04 20:18:30Z Stopping Fusion UI on port 8764
2016-10-04 20:18:42Z Stopping Fusion Connectors on port 8984
2016-10-04 20:18:53Z Stopping Fusion API Services on port 8765
2016-10-04 20:19:05Z Stopping Fusion Spark Worker on port 8769
2016-10-04 20:19:10Z Stopping Fusion Spark Master on port 8766
2016-10-04 20:19:15Z Stopping Fusion Solr on port 8983
2016-10-04 20:19:27Z Stopping Fusion ZooKeeper on port 9983
```

3.10.4. Troubleshooting

The Java Virtual Machine Process Status Tool utility at `/usr/bin/jps` is useful for reporting on all Fusion processes reported by script `fusion/bin/fusion`:

```
$ jps
1644 QuorumPeerMain   (ZK)
1793 start.jar        (solr)
1962 SparkAgent       (spark)
2072 SparkAgent
2230 start.jar        (api)
2389 SparkWorker
2410 SparkMaster
2525 start.jar        (connectors)
2708 start.jar        (ui)
3760 SparkDriver
4431 Jps
```

The process `QuorumPeerMain` is the ZooKeeper process used by Fusion. The 4 `start.jar` processes are Fusion’s Solr, API Services, Connectors, and UI. Spark SparkMaster and SparkWorker processes are used for Fusion signals processing and SparkAgent processes are used for the Spark shell.

If the `path/to/fusion/bin/fusion` script doesn’t run, or if it fails to start all services, see the Troubleshooting topic or the knowledge base for help.

Chapter 4. Collections

Fusion collections are Solr collections managed by Fusion. A Solr collection is a distributed index defined by a named configuration stored in ZooKeeper, with these properties:

- Number of shards

  Documents are distributed across this number of partitions.

- Document routing strategy

  How documents are assigned to shards.

- Replication factor

  How many copies of each document in the collection.

- Replica placement strategy

  Where to place replicas in the cluster.

Solr is the underlying engine which indexes, stores, and searches your data. Fusion manages Solr collections, manipulates data and queries before passing them to Solr, and provides analytics and monitoring features.

If your data is already stored in a Solr instance or cluster, you can manage this collection in Fusion by creating a Fusion collection which is configured to import the existing Solr collection. See Installation with an existing Solr instance or cluster.

4.1. Primary and Auxiliary Collections

In Fusion, the "Primary" collection is the collection which contains your application data, that is, the set of documents over which search and indexing happens. Fusion registers the collection name and information about the Solr cluster that manages this collection.

Note	All collection names should be considered to be case-insensitive, even though Fusion preserves case in referring to these collections.

If your application uses Fusion’s signals, analytics, or monitoring services, then Fusion will create a set of auxiliary collections in which to store signals, query, and other logfiles. Naming conventions relate auxiliary collections with the primary collection. Auxiliary collections have the same base name as the name of the primary collection plus a suffix which indicates the kind of auxiliary collection, e.g., the suffix for a query logs auxiliary collection is "_logs" so that for a primary collection named "COLL", Fusion creates an auxiliary collection named "COLL_logs". These auxiliary collections include:

- A search query logs collection, suffix "_logs".
- A pair of associated collections for signals and aggregated signals, suffixes "_signals", "_signals_aggr" respectively.

Note	Do not create primary collections with names that end in suffix "_logs", "_signals", or "_signals_aggr". Such names can only be used for Fusion auxiliary collections, which are created and managed by Fusion directly.

Fusion maintains a set of Solr collections which store Fusion’s own logfiles and other internal information. These are called System Collections, described below.

Note	Do not create primary collections named "logs", or which begin with "system_". These names are reserved for Fusion system collections.

Fusion uses ZooKeeper to register information about all collections, and the Fusion components and services related to a collection. The Fusion components associated with a collection include:

- Datasources
- Pipelines
- Profiles
- Signals and aggregations
- Analytics Dashboards

4.2. System Collections

Fusion uses *system collections* for internal purposes:

- **logs** indexes the log messages from the Fusion API services.
- **audit_logs** indexes all HTTP requests sent to the Fusion API services.
- **system_banana** stores configurations used by Fusion dashboards.
- **system_blobs** stores blobs in Solr. This is used to store model files for the NLP components and other binary files used by Fusion components.
- **system_messages** is used by Fusion’s Messaging Services.
- **system_metrics** stores information about the running process itself, such as the amount of memory in the system, the average response time for services, Solr heap size, etc. The data is polled at regular intervals according to the internal configuration variable: `com.lucidworks.apollo.metrics.poll.seconds`. This collection doesn’t appear until after the first set of metrics are collected.

4.3. Collection Configuration Properties

Collections have three configurable properties which are set to default values in the Fusion UI. They can be configured as appropriate for your application by creating the collection using the Fusion API service Collections API.

Property	Description
signals	Property `signals` determines whether or not to create an auxiliary collections "_signals" and "_signals_aggr". When creating a collection in the Fusion UI, this property defaults to **true**. When creating a collection using Fusion's API services, this property defaults to **false**.
searchLogs	Property `searchLogs` determines whether or not to create an auxiliary search query logs collection with suffix "_logs". When creating a collection in the Fusion UI, this property defaults to **true**. When creating a collection using Fusion's API services, this property defaults to **false**.
dynamicSchema	Property `dynamicSchema` always defaults to **false**. When `dynamicSchema` is **true**, Fusion and Solr use schemaless mode to administer search and indexing over that collection.

Signals are events with timestamps that can be used to improve search results. For more information about signals in Fusion, see the section Signals.

Search logs data is used for Search Query Reporting. The set of reports available includes most popular documents, queries that generated less than a minimum number of results, and search histograms.

The name schemaless mode is misleading: Solr always uses a schema when managing a collection. In schemaless mode, if a document contains a field not currently in the Solr schema, Solr processes the field value to determine what the field type should be defined as, and then adds a new field to the schema with the field name and field type. This behavior may be convenient during preliminary application development, but is rarely appropriate in a production environment, therefore the default is false.

4.4. Collection Profiles

Profiles are used to create pipeline aliases for a specific collection. In Fusion, index and query pipelines are not connected to a specific collection by default so that pipeline can be created once and re-used in several collections. This complicates the way that pipelines are used with collections. Profiles provide a shortcut.

- Index Profiles work with index pipelines for getting content into the system.
- Query Profiles work with query pipelines for user queries.

4.5. Solr Config Files

Solr uses several XML configuration files to specify the contents and behavoir of a Solr collection. Fusion provides a set of default configurations, but depending on your aplication needs, it may be necessary to edit these configuration files.

The "Configuration" section of the collection home panel contains option "Solr Config" which opens a "Solr Config" panel. This panel displays the set of Solr configuration XML files that you can edit. This is an advanced feature, for experienced Solr users. See the Solr wiki for more information on configuring Solr.

4.6. Stopwords Files

Fusion collections are Solr collections which are managed by Fusion. Solr itself manages a set of resources for a collection. Stopword lists are one such resource.

The Fusion UI provides a Stopwords manager tool which is reached from the "Stopwords" tab in the "Configuration" section of the collection home panel. Clicking on this tool allows you to view the contents of all configured stopwords files in an editable browser.

4.7. Synonyms Files

Fusion collections are Solr collections which are managed by Fusion. Solr itself manages a set of resources for a collection. These resources include stopword lists and synonyms.

There are two ways of specifying synonyms: as a set of interchangeable tokens, or as a mapping from one token to one or more other tokens. A set of interchangeable tokens is specified as a series of comma-sperated words:

```
Television, Televisions, TV, TVs
```

This will be used to expand any document or query by expanding any single token in this set to all tokens in this set.

A mapping from one token to one or more other tokens is specified as follows:

```
pixima => pixma
```

In this case, tokens on the left hand side of the rule will be replaced by the tokens on the right hand side.

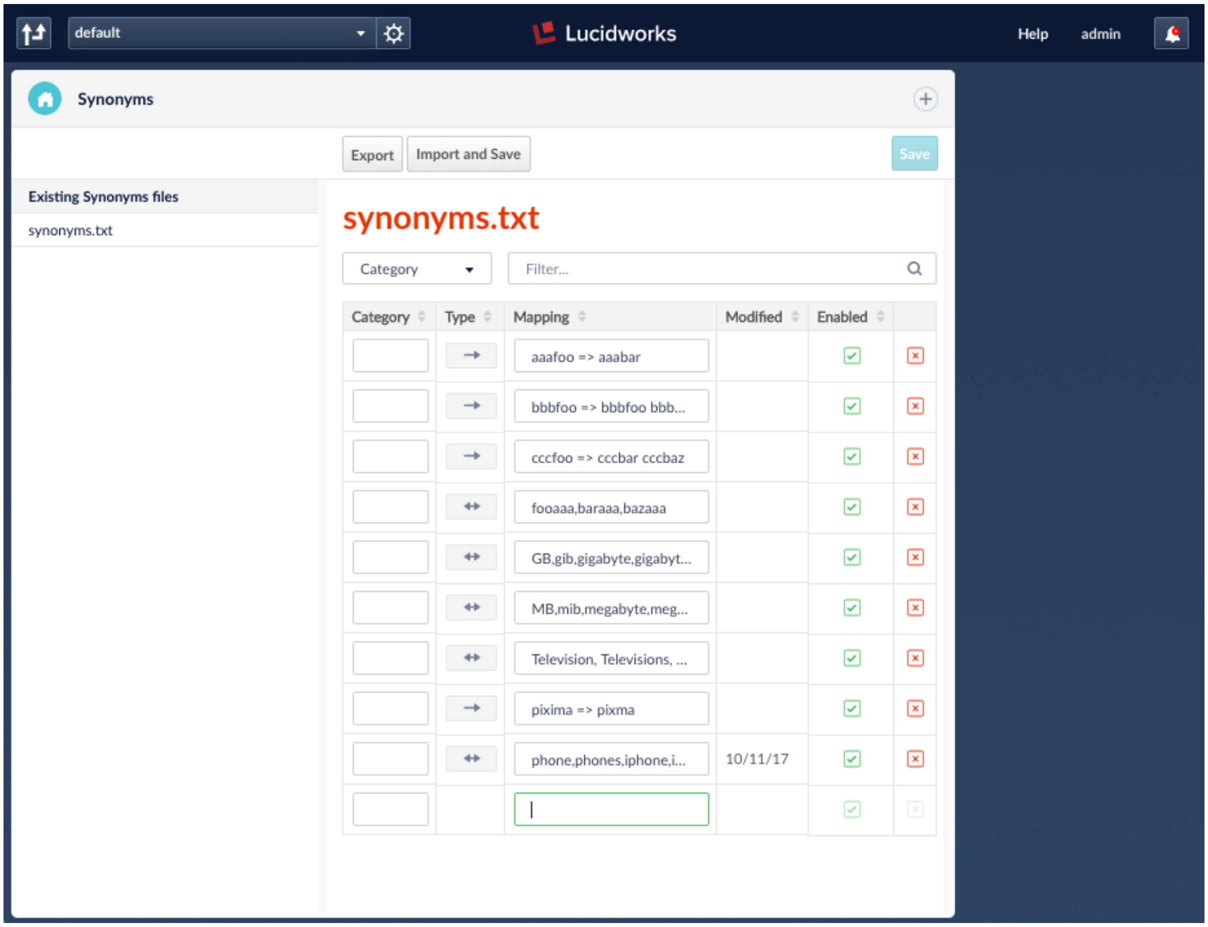
The "Configuration" section of the collection home panel contains options for managing each of these resources from inside Fusion.

You can view this file as a text file from the "Solr Config" panel:

```
#some test synonym mappings unlikely to appear in real input text
aaafoo => aaabar
bbbfoo => bbbfoo bbbbar
cccfoo => cccbar cccbaz
fooaaa,baraaa,bazaaa

# Some synonym groups specific to this example
GB,gib,gigabyte,gigabytes
MB,mib,megabyte,megabytes
Television, Televisions, TV, TVs
#notice we use "gib" instead of "GiB" so any WordDelimiterFilter coming
#after us won't split it into two words.

# Synonym mappings can be used for spelling correction too
pixima => pixma
```

4.7.1. Synonyms Manager

The Fusion UI provides a Synonyms manager tool which is reached from the "Synonyms" tab in the "Configuration" section of the collection home panel. Once opened, the Synonyms manager panel will display the synonyms.txt as a series of editable objects, 1 per line.



Category	Type	Mapping	Modified	Enabled	
	→	aaafoo => aaabar		☑	☒
	→	bbbfoo => bbbfoo bbb...		☑	☒
	→	cccfoo => cccbar cccbaz		☑	☒
	↔	fooaaa,baraaa,bazaaa		☑	☒
	↔	GB,gib,gigabyte,gigabyt...		☑	☒
	↔	MB,mib,megabyte,meg...		☑	☒
	↔	Television, Televisions, ...		☑	☒
	→	pixima => pixma		☑	☒
	↔	phone,phones,iphone,i...	10/11/17	☑	☒
				☑	☒

Moving the cursor out of the editor field will add the item, clicking "Save" in the upper right corner will apply it to the current collection immediately, storing the config file in the correct Fusion folder. Choosing "Export" will download a copy via the browser to the local user computer. "Import and Save" reads a local file into Fusion.

An additional feature of the Synonym manager in the Fusion UI is that label terms can be used to restrict the display of synonyms to a browsable subset by adding a "category" label to the synonym set or mapping rule. This category label is a convenience for working with data in the Fusion UI only. The information is not saved in the synonyms file used by Solr.

Chapter 5. Datasources

A collection includes one or more datasources. A datasource is a configuration that manages the import and indexing of data into the collection.

The Index Workbench provides a development environment for creating, configuring, and testing a datasource configuration. Every datasource configuration includes the following:

- Connector configuration, specifying the source and format of the incoming data.
- Parser configuration, describing a series of conditional parsing stages to transform the incoming data into PipelineDocument objects.
- Index pipeline configuration, consisting of stages that transform PipelineDocument objects into Solr documents to be indexed.

Collections and datasources can also be managed through the REST API.

In some cases it may make sense to bypass the connectors and use other ingest methods for your data.

5.1. Ingest and Indexing

You’ve got data, and you need to get it into Fusion. Here’s how it works:

- *Ingest* is the process of sending your data to Fusion via a compatible protocol.
    - Connectors are Fusion components that ingest and parse specific kinds of data. There’s a Fusion connector for just about any data type.
    - Lucidworks provides plugins for Hive and Pig that allow them to send data to Fusion for ingest.
    - The Index Pipelines API allows you to push data directly to one of Fusion’s index pipelines.
- *Indexing* converts your data into an indexed collection in Fusion’s Solr core. It’s critical for ensuring that your data is stored in a format that’s ideal for your search application.
    - Index pipelines determine the details of the conversion.
    - Index pipeline stages are the building blocks of index pipelines.
    - The Index Pipeline Simulator is Fusion’s index pipeline development tool.

5.2. Importing Data with Connectors

Connectors are the built-in mechanism for pulling your data into Fusion. Fusion comes with a wide variety of connectors, each specialized for a particular data type. When you add a datasource to a collection, you specify the connector to use for ingesting data. See Connector Types for a complete list of connectors, with links to configuration reference information for each one.

5.2.1. Connector configuration

When you add a datasource to a collection, you select a connector and configure it. There are two ways to do this:

- Using the API
- Using the UI

Configuring Connectors Using The API

You can create or update a datasource with the Connector Datasources API, specifying the connector, its properties, and their values.

Example: *Create and configure a datasource to index Solr-formatted XML files*

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d '{"id":"SolrXML",
"connector":"lucid.solrxml", "type":"solrxml", "properties":{"path":"/Applications/solr-
4.10.2/example/exampledocs", "generate_unique_key":false, "collection":"MyCollection"}}'
http://localhost:8764/api/apollo/connectors/datasources
```

See the Connectors and Datasources Reference for details about configuration options.

Tip	Be sure the include the `collection` property; otherwise the datasource will not be available in the Fusion UI.

Example: *Change the `max_docs` value for the above datasource*

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d '{"id":"SolrXML", "connector":"lucid.solrxml",
"type":"solrxml", "properties":{"path":"/Applications/solr-4.10.2/example/exampledocs", "max_docs":10}}'
http://localhost:8764/api/apollo/connectors/datasources/SolrXML
```

Configuring Connectors Using The Fusion UI

- To create and configure a new datasource and its connector:
  1. Click **Applications** > **Collections**.
  2. Select a Collection, or click **Add a Collection** to create a new one.
  3. Click **Add a Datasource**.
  4. Select a datasource type; these correspond to Fusion’s Connectors.
  5. Edit the configuration fields in the datasource panel that appears.

     See the Connectors and Datasources Reference for details about configuration options.

   6. Click **Save**.

- To change the connector configuration for an existing datasource:

   1. Navigate to **Applications** > **Collections**.

   2. Click your collection name.

   3. Click the datasource you want to change.

   4. Edit the configuration fields as needed.

   5. Click **Save**.

5.3. Other Ingestion Methods

Usually, the simplest way to get data into Fusion is through its connectors. However, in some cases it makes sense to use other methods:

- Importing directly into Solr

  Fusion can read any data that has been imported into its Solr core. If you're already a proficient Solr user, you may already have mechanisms in place for doing this. You can continue importing your data directly into Solr, then use Fusion to read and manage it.

  The Fusion package also includes tools for direct import into Solr:

  - a Hive Serializer/Deserializer
  - a set of Pig functions

- Pushing to an index pipeline

  You can bypass the connectors and parsers to push documents directly to an index pipeline, using the Index-Pipelines REST API. You'll need to understand Fusion PipelineDocument Objects first.

5.4. Importing Data with Pig

You can use Pig to import data into Fusion, using the `lucidworks-pig-functions-2.2.1.jar` file found in `fusion/apps/connectors/resources/lucid.hadoop/jobs`.

5.4.1. Available Functions

The Pig functions included in the `lucidworks-pig-functions-2.2.1.jar` are three UserDefined Functions (UDF) and one Store function. These functions are:

- `com/lucidworks/hadoop/pig/SolrStoreFunc.class`
- `com/lucidworks/hadoop/pig/EpochToCalendar.class`
- `com/lucidworks/hadoop/pig/Extract.class`
- `com/lucidworks/hadoop/pig/Histogram.class`

5.4.2. Using the Functions

There are two approaches to using functions in Pig: to `REGISTER` them in the script, or to load them with your Pig command line request.

If using `REGISTER`, the Pig function jars must be put in HDFS in order to be used by your Pig script. It can be located anywhere in HDFS; you can either supply the path in your script or use a variable and define the variable with `-p` property definition.

The example below uses the second approach, loading the .jars with `-Dpig.additional.jars` system property when launching the script. With this approach, the .jars can be located anywhere on the machine where the script will be run.

When using the Pig functions, you can pass parameters for your script on the command line. The properties you will need to pass are the location of Solr and the collection to use; these are shown in detail in the example below.

5.4.3. Indexing to a Kerberos-Secured Solr Cluster

When a Solr cluster is secured with Kerberos for internode communication, Pig scripts must include the full path to a JAAS file that includes the service principal and the path to a keytab file that will be used to index the output of the script to Solr.

Two parameters provide the information the script needs to access the JAAS file:

`lww.jaas.file`

The path to the JAAS file that includes a section for the service principal who will write to the Solr indexes. For example, to use this property in a Pig script: +

```
set lww.jaas.file '/path/to/login.conf';
```

```
  +
  The JAAS configuration file *must* be copied to the same path on every node where a Node Manager is running (i.e., every node where map/reduce tasks are executed).
`lww.jaas.appname`::
  The name of the section in the JAAS file that includes the correct service principal and keytab path. For example, to use this property in a Pig script:
  +
```

```
set lww.jaas.appname 'Client';
```

Here is a sample section of a JAAS file:

```
Client { (1)
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="/data/solr-indexer.keytab" (2)
  storeKey=true
  useTicketCache=true
  debug=true
  principal="solr-indexer@SOLRSERVER.COM"; (3)
};
```

1. The name of this section of the JAAS file. This name will be used with the `lww.jaas.appname` parameter.
2. The location of the keytab file.
3. The service principal name. This should be a different principal than the one used for Solr, but must have access to both Solr and Pig.

5.4.4. Indexing to a SSL-Enabled Solr Cluster

When SSL is enabled in a Solr cluster, Pigs scripts must include the full paths to the `keystore` and `truststore` with their respective passwords.

```
set lww.keystore '/path/to/solr-ssl.keystore.jks'
set lww.keystore.password 'secret'
set lww.truststore '/path/to/solr-ssl.truststore.jks'
set lww.truststore.password 'secret'
```

Tip	The path (and secret configurations) should be the same in all YARN/MapReduce hosts.

5.4.5. Sample CSV Script

The following Pig script will take a simple CSV file and index it to Solr.

```
set solr.zkhost '$zkHost';
set solr.collection '$collection'; (1)

A = load '$csv' using PigStorage(',') as
(id_s:chararray,city_s:chararray,country_s:chararray,code_s:chararray,code2_s:chararray,latitude_s:chararray,l
ongitude_s:chararray,flag_s:chararray); (2)
--dump A;
B = FOREACH A GENERATE $0 as id, 'city_s', $1, 'country_s', $2, 'code_s', $3, 'code2_s', $4, 'latitude_s', $5,
'longitude_s', $6, 'flag_s', $7; (3)

ok = store B into 'SOLR' using com.lucidworks.hadoop.pig.SolrStoreFunc(); (4)
```

This relatively simple script is doing several things that help to understand how the Solr Pig functions work.

1. This and the line above define parameters that are needed by `SolrStoreFunc` to know where Solr is. `SolrStoreFunc` needs the properties `solr.zkhost` and `solr.collection`, and these lines are mapping the `zkhost` and `collection` parameters we will pass when invoking Pig to the required properties.
2. Load the CSV file, the path and name we will pass with the `csv` parameter. We also define the field names for each column in CSV file, and their types.
3. For each item in the CSV file, generate a document id from the first field (`$0`) and then define each field name and value in `name, value` pairs.
4. Load the documents into Solr, using the `SolrStoreFunc`. While we don't need to define the location of Solr here, the function will use the `zkhost` and `collection` properties that we will pass when we invoke our Pig script.

Warning	When using `SolrStoreFunc`, the document ID **must** be the first field.

When we want to run this script, we invoke Pig and define several parameters we have referenced in the script with the `-p` option, such as in this command:

```
./bin/pig -Dpig.additional.jars=/path/to/lucidworks-pig-functions-2.2.1.jar -p
csv=/path/to/my/csv/airports.dat -p zkHost=zknode1:2181,zknode2:2181,zknode3:2181/solr -p
collection=myCollection ~/myScripts/index-csv.pig
```

The parameters to pass are:

`csv`

The path and name of the CSV file we want to process.

`zkhost`

The ZooKeeper connection string for a SolrCloud cluster, in the form of `zkhost1:port,zkhost2:port,zkhost3:port/chroot`. In the script, we mapped this to the `solr.zkhost` property, which is required by the `SolrStoreFunc` to know where to send the output documents.

`collection`

The Solr collection to index into. In the script, we mapped this to the `solr.collection` property, which is required by the `SolrStoreFunc` to know the Solr collection the documents should be indexed to.

Tip

The zkhost parameter above is only used if you are indexing to a SolrCloud cluster, which uses ZooKeeper to route indexing and query requests.

If, however, you are not using SolrCloud, you can use the solrUrl parameter, which takes the location of a standalone Solr instance, in the form of http://host:port/solr.

In the script, you would change the line that maps solr.zkhost to the zkhost property to map solr.server.url to the solrUrl property. For example:

```
`set solr.server.url '$solrUrl';`
```

5.5. Importing Data with Hive

Fusion ships with a Serializer/Deserializer (SerDe) for Hive, included in the distribution as `lucidworks-hive-serde-2.2.1.jar` in `fusion/apps/connectors/resources/lucid.hadoop/jobs`.

5.5.1. Install the SerDe Jar to Hive

In order for Hive to work with Solr, the Hive SerDe jar must be added as a plugin to Hive.

From a Hive prompt, use the `ADD JAR` command and reference the path and filename of the SerDe jar for your Hive version.

```
   hive> ADD JAR lucidworks-hive-serde-2.2.1.jar;
```

This can also be done in your Hive command to create the table, as in the example below.

5.5.2. Create External Tables

An external table in Hive allows the data in the table to be used (read or write) by another system or application outside of Hive. For integration with Solr, the external table allows you to have Solr read from and write to Hive.

To create an external table for Solr, you can use a command similar to below. The properties available are described after the example.

```
hive> CREATE EXTERNAL TABLE solr (id string, field1 string, field2 int)
      STORED BY 'com.lucidworks.hadoop.hive.LWStorageHandler'
      LOCATION '/tmp/solr'
      TBLPROPERTIES('solr.server.url' = 'http://localhost:8888/solr',
                    'solr.collection' = 'collection1',
                    'solr.query' = '*:*');
```

In this example, we have created an external table named "solr", and defined a custom storage handler (`STORED BY 'com.lucidworks.hadoop.hive.LWStorageHandler'`).

The LOCATION indicates the location in HDFS where the table data will be stored. In this example, we have chosen to use `/tmp/solr`.

In the section TBLPROPERTIES, we define several properties for Solr so the data can be indexed to the right Solr installation and collection:

`solr.zkhost`

The location of the ZooKeeper quorum if using LucidWorks in SolrCloud mode. If this property is set along with the `solr.server.url` property, the `solr.server.url` property will take precedence.

`solr.server.url`

The location of the Solr instance if not using LucidWorks in SolrCloud mode. If this property is set along with the `solr.zkhost` property, this property will take precedence.

`solr.collection`

The Solr collection for this table. If not defined, an exception will be thrown.

`solr.query`

The specific Solr query to execute to read this table. If not defined, a default of `:` will be used. This property is not needed when loading data to a table, but is needed when defining the table so Hive can later read the table.

`lww.commit.on.close`

If true, inserts will be automatically committed when the connection is closed. True is the default.

`lww.jaas.file`

Used only when indexing to or reading from a Solr cluster secured with Kerberos. + This property defines the path to a JAAS file that contains a service principal and keytab location for a user who is authorized to read from and write to Solr and Hive. + The JAAS configuration file **must** be copied to the same path on every node where a Node Manager is running (i.e., every node where map/reduce tasks are executed). Here is a sample section of a JAAS file: +

```
Client { (1)
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  keyTab="/data/solr-indexer.keytab" (2)
  storeKey=true
  useTicketCache=true
  debug=true
  principal="solr-indexer@SOLRSERVER.COM"; (3)
};
```

```
  +
  1.  The name of this section of the JAAS file. This name will be used with the `lww.jaas.appname` parameter.
  2.  The location of the keytab file.
  3.  The service principal name. This should be a different principal than the one used for Solr, but must
have access to both Solr and Hive.
`lww.jaas.appname`::
  Used only when indexing to or reading from a Solr cluster secured with Kerberos.
  +
  This property provides the name of the section in the JAAS file that includes the correct service principal
and keytab path.
```

If the table needs to be dropped at a later time, you can use the DROP TABLE command in Hive. This will remove the metadata stored in the table in Hive, but will not modify the underlying data (in this case, the Solr index).

5.5.3. Query and Insert Content

Once the table is configured, any syntactically correct Hive query will be able to query the Solr index.

For example, to select three fields named "id", "field1", and "field2" from the "solr" table, you would use a query such as:

`hive> SELECT id, field1, field2 FROM solr;`

To join data from tables, you can make a request such as:

```
hive> SELECT id, field1, field2 FROM solr left
      JOIN sometable right
      WHERE left.id = right.id;
```

And finally, to insert data to a table, simply use the Solr table as the target for the Hive INSERT statement, such as:

```
hive> INSERT INTO solr
      SELECT id, field1, field2 FROM sometable;
```

5.5.4. Example Hive Table

Solr includes a small number of sample documents for use when getting started. One of these is a CSV file containing book metadata. This file is found in your Solr installation, at `$SOLR_HOME/example/exampledocs/books.csv`.

Using the sample `books.csv` file, we can see a detailed example of creating a table, loading data to it, and indexing that data to Solr.

```
CREATE TABLE books (id STRING, cat STRING, title STRING, price FLOAT, in_stock BOOLEAN, author STRING, series
STRING, seq INT, genre STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','; (1)

LOAD DATA LOCAL INPATH '/solr/example/exampledocs/books.csv' OVERWRITE INTO TABLE books; (2)

ADD JAR {packageUser}-hive-serde-{connectorVersion}.jar; (3)

CREATE EXTERNAL TABLE solr (id STRING, cat_s STRING, title_s STRING, price_f FLOAT, in_stock_b BOOLEAN,
author_s STRING, series_s STRING, seq_i INT, genre_s STRING) (4)
     STORED BY 'com.lucidworks.hadoop.hive.LWStorageHandler' (5)
     LOCATION '/tmp/solr' (6)
     TBLPROPERTIES('solr.zkhost' = 'zknode1:2181,zknode2:2181,zknode3:2181/solr',
                   'solr.collection' = 'gettingstarted',
                   'solr.query' = '*:*'),
                   'lww.jaas.file' = '/data/jaas-client.conf'; (7)


INSERT OVERWRITE TABLE solr SELECT b.* FROM books b;
```

1. Define the table `books`, and provide the field names and field types that will make up the table.
2. Load the data from the `books.csv` file.
3. Add the `lucidworks-hive-serde-2.2.1.jar` file to Hive. Note the jar name shown here omits the version information which will be included in the jar file you have. If you are using Hive 0.13, you must also use a jar specifically built for 0.13.
4. Create an external table named `solr`, and provide the field names and field types that will make up the table. These will be the same field names as in your local Hive table, so we can index all of the same data to Solr.
5. Define the custom storage handler provided by the `lucidworks-hive-serde-2.2.1.jar`.
6. Define storage location in HDFS.
7. Define the location of Solr (or ZooKeeper if using SolrCloud), the collection in Solr to index the data to, and the query to use when reading the table. This example also refers to a JAAS configuration file that will be used to authenticate to the Kerberized Solr cluster.

Chapter 6. Indexing Data

When a connector ingests data, it passes the data through an index pipeline for transformation prior to indexing. The format of your indexed data depends on the index pipeline, which is part of the datasource configuration.

An index pipeline consists of one or more configurable index pipeline stages, each performing a different type of transformation on the incoming data. Each connector has a default index pipeline, but you can modify these or create new ones.

The last stage in any index pipeline should be the Solr Indexer stage, which submits the documents to Solr for indexing.

The following pages provide details on how to configure and use Fusion’s index pipelines:

- Fusion PipelineDocument Objects describes the datatype on which pipelines operate.
- Pushing Documents to a Pipeline is a brief overview on how to quickly index content using the pipelines directly (instead of using a datasource).
- Index Profiles allow the use of one pipeline for multiple collections.
- Entity-Extraction covers Natural Language Processing (NLP) tools and techniques for finding names of people places and things.
- Blob Storage describes how to upload large binary objects to Fusion. Language model files needed for the entity extraction pipeline stages are loaded into Fusion by this mechanism.
- Time-Based Partitioning explains how to configure a collection to automatically index data by time slice.
- The Index Pipeline Simulator is a tool for previewing the output of a pipeline while you configure it.

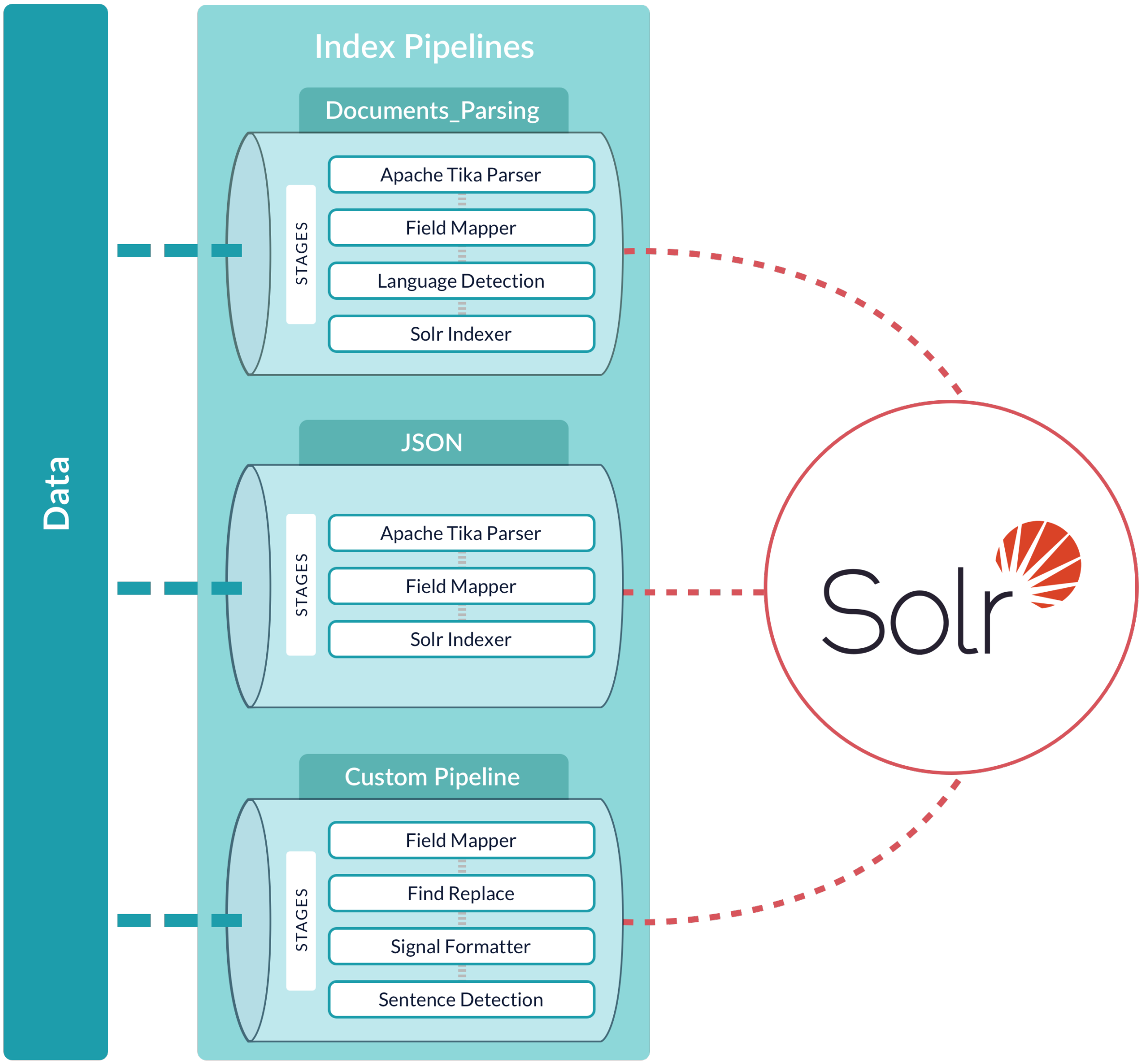
6.1. Index Pipelines

Index pipelines transform incoming data into PipelineDocument objects for indexing by Fusion's Solr core. An index pipeline consists of a series of configurable index pipeline stages, each performing a different transformation on the data before passing the result to the next stage in the pipeline. The final stage is the Solr Indexer stage, which transforms the PipelineDocument into a Solr document and submits it to Solr for indexing in a specific Collection.

Each configured datasource has an associated index pipeline and uses a connector to fetch data to parse and then input into the index pipeline.

Alternatively, documents can be submitted directly to an Index Pipeline via the REST API; see Pushing Documents to a Pipeline.

A pipeline can be re-used across multiple collections. Fusion provides a set of built-in pipelines. You can use the Index Workbench or the REST API to develop custom index pipelines to suit any datasource or application.



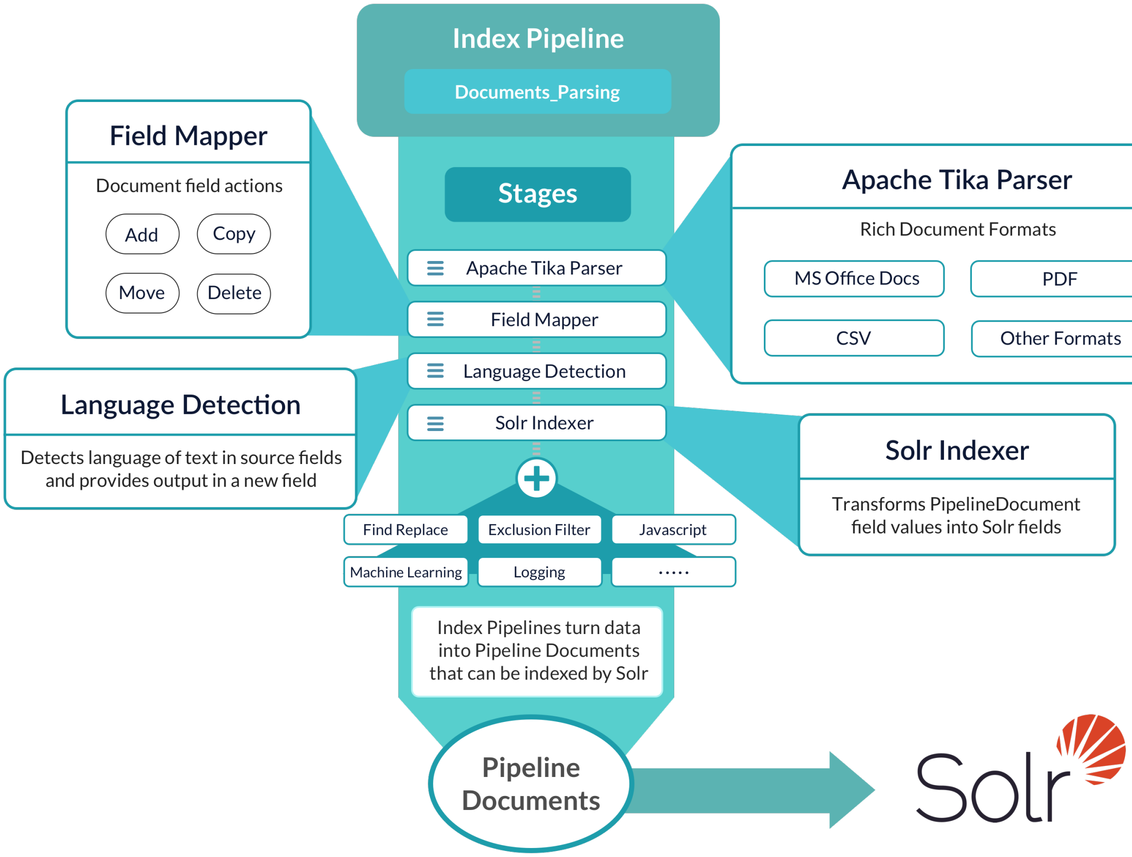
6.1.1. Collection-specific Pipelines

When a Fusion collection is created using the Fusion UI, a pair of index and query pipelines are created to that pipeline, where the pipeline name is the collection name with the suffix "-default". This pipeline consists of a Field Mapper index stage

Although default pipelines are created when a Fusion collection is created, they are not deleted when the collection is deleted. This is due to the fact that pipelines can be used across collections, therefore a named pipeline, although originally associated with a collection, may be used by several collections.

6.1.2. Pre-configured Pipelines

Fusion includes several pre-configured pipelines which which provide out-of-the-box processing capabilities and/or a starting point for customization. There are also a set of named pipelines which are used by Fusion services for logging, signal processing, and signal aggregation.



General Purpose Pipelines

- **CSV** - a pipeline for handling tabular data from CSV files, using these stages:
  1. CSV Parser index stage
  2. Field Mapper index stage (with no pre-defined mappings)
  3. Solr Indexer stage
- **Default_Data** - a pipeline for processing general key-value data, i.e., data which has already been parsed into key-value pairs.
  1. Field Mapper index stage
  2. Solr Indexer stage.
- **Discard** (Fusion 2.0) / **conn_noop**** - a pipeline used for testing datasource configurations which has no defined stages.

- **Documents_Parsing** (Fusion 2.0) - a pipeline used to parse and index documents.
  1. Apache Tika Parser index stage, which can process a wide variety of common document formats and is able to extract metadata as well.
  2. Field Mapper index stage, which has mapping rules for common document elements.
  3. Language Detection index stage
  4. Solr Indexer stage
- **Documents_Parsing_debug_logging** (Fusion 2.0) - this pipeline is an augmented version of the Documents_Parsing pipeline where a logging stage has been added before every processing stage.
- **JSON** - a pipeline for handling JSON data.
  1. JSON Parser index stage.
  2. Field Mapper index stage
  3. Solr Indexer stage
- **Logstash** - a pipeline for handling Logstash data.
  1. Field Mapper index stage which handles timestamp fields.
  2. Field Mapper index stage (empty)
  3. Solr Indexer stage
- **Source_Code** - a pipeline for extracting source code from Git and SVN repositories.
  1. Apache Tika Parser index stage, to extract raw data.
  2. Field Mapper index stage to map elements to proper Solr field names.
  3. Solr Indexer stage

Legacy Pipelines

- **conn_solr** - a pipeline used to parse and index documents. The initial stage is a Tika Parser index stage. The next stage is a Field Mapper index stage which has mapping rules for common document elements. The final stage is a Solr Indexer stage.
- **default** - a pipeline which consists of just a Solr Indexer stage, used to push documents which have been completely parsed and have appropriately named fields to Solr for indexing.

Internal Use Pipelines

- **_aggregation_default** - a pipeline which consists of a single Solr Indexer stage which sends aggregations to Solr.
- **_aggregation_rollup** - also a pipeline which consists of a single Solr Indexer stage which sends aggregations to Solr.
- **_signals_ingest** - a pipeline used to index raw signal data. It has three stages, a Format Signals stage, a Field Mapping stage and a Solr Indexer stage to index the raw signal events.
- **_system_metrics** - a pipeline which consists of a single Solr Indexer stage which sends internal information to the Fusion system_metrics collection.

6.2. Fusion PipelineDocument Objects

A PipelineDocument organizes the contents of each document submitted to the pipeline, document-level metadata, and processing commands into a list of fields where each field has a string name, a value, an associated metadata object and a list of annotations. A Solr Indexer stage transforms a PipelineDocument into a Solr document and submits it to Solr for indexing.

6.2.1. The PipelineDocument Java Object

Under the Fusion hood, a PipelineDocument is a Java object, see the PipelineDocument javadocs.

6.2.2. JSON representation of a PipelineDocument

The JSON representation of a PipelineDocument object has four fields:

- **id** : value is a string identifier.
- **commands** : value is a list of processing commands for the index (optional)
- **metadata** : value is single object containing a name : value pair (optional)
- **fields**: value is a list of field objects, where a field object consists of four fields:
    - name : value is a string containing the field name
    - value : value is a string containing the field value
    - metadata : value is a single object containing a name : value pair (optional)
    - annotations : value is a list of annotations (optional)

Pipeline stages add, remove, and update the fields of the PipelineDocument. The Solr Indexer stage transforms the list of PipelineDocument fields into a set of Solr document fields.

The commands field can be used to issue a commit at the end of document processing or to delete documents based on documents that match an included query.

If a pipeline includes a logging stage, the PipelineDocument will be pretty-printed to the Fusion connectors logfile (default location `fusion/var/log/connectors/connectors.log`). To see how this works, we set up a pipeline consisting of an initial logging stage, followed by a Apache Tika Parser stage, followed by another logging stage, followed by a Field Mapper stage.

We define a datasource named "email" configured with lucid.anda filesystem connector that submits the contents of a file named "test_email.eml" to the pipeline.

The initial logging stage from the connectors logfile is shown below. The raw bytes from the file are encoded as a BASE64 string in field "*raw_content*". After the initial logging stage (before Tika Parsing), the PipelineDocument object is:

```
{ "id" : "/Users/demo/test_email.eml",
  "metadata" : { },
  "commands" : [ ],
  "fields" : [
    { "name" : "_lw_batch_id_s",
      "value" : "14982991ada04c62a77cbb9ee4b32439",
      "metadata" : { },
      "annotations" : [ ] },
    { "name" : "_lw_data_source_s",
      "value" : "email",
      "metadata" : { },
      "annotations" : [ ] },
    { "name" : "_lw_data_source_collection_s",
      "value" : "foo",
      "metadata" : { },
      "annotations" : [ ] },
    { "name" : "_lw_data_source_pipeline_s",
      "value" : "conn_logging",
      "metadata" : { },
      "annotations" : [ ] },
    { "name" : "_lw_data_source_type_s",
      "value" : "lucid.anda/file",
      "metadata" : { },
      "annotations" : [ ] },
    { "name" : "lastModified_dt",
      "value" : "2015-02-28T14:14:32Z",
      "metadata" : { "creator" : "lucid.anda" },
      "annotations" : [ ] },
    { "name" : "_raw_content_",
      "value" :
"TUlNRS1WZXJzaW9uOiAxLjAKUmVjZWl2ZWQ6IGJ5IDEwLjIwMi4yMjguMTk3IHdpdGggSFRUUDsgRnJpLCAyNyBGZWIgMjAxNSAxOToxMzo0M
SAtMDgwMCAoUFNUKQpEYXRlOiBTYXQsIDI4IEZlYiAyMDE1IDE0OjEzOjQxICsxMTAwCkRlbGl2ZXJlZC1UbzogbWl0emkubW9ycmlzQGx1Y2l
kd29ya3MuY29tCk1lc3NhZ2UtSUQ6IDxDQU03UFJDVjVuYzQxMTJ2Ym5hdEtKNk05RFVEU0prVzFpeXcxUHhhLWJaWFVCZ1FlV3dAbWFpbC5nb
WFpbC5jb20+ClN1YmplY3Q6IHRoaXMgaXMgdGhlIHN1YmplY3Qgb2YgZW1haWwgbWVzc2FnZQpGcm9tOiBNaXR6aSBNb3JyaXMgPG1pdHppLm1
vcnJpc0BsdWNpZHdvcmtzLmNvbT4KVG86IE1pdHppIE1vcnJpcyA8bWl0emkubW9ycmlzQGx1Y2lkd29ya3MuY29tPgpDb250ZW50LVR5cGU6I
G11bHRpcGFydC9hbHRlcm5hdGl2ZTsgYm91bmRhcnk9MDQ3ZDdiZDc1ZjJhM2YzNGYzMDUxMDFkNWY5OQoKLS0wNDdkN2JkNzVmMmEzZjM0ZjM
wNTEwMWQ1Zjk5CkNvbnRlbnQtVHlwZTogdGV4dC9wbGFpbjsgY2hhcnNldD1VVEYtOAoKdGhpcyBpcyB0aGUgZmlyc3QgbGluZSBvZiB0aGUgY
m9keSBvZiBhbiBlbWFpbCBtZXNzYWdlLgoKYW5kIHRoaXMgaXMgdGhlIHNlY29uZCBsaW5lLgoKYW5kIHRoaXMgaXMgdGhlIGNsb3NpbmcsIGN
oZWVycywKCi0tMDQ3ZDdiZDc1ZjJhM2YzNGYzMDUxMDFkNWY5OQpDb250ZW50LVR5cGU6IHRleHQvaHRtbDsgY2hhcnNldD1VVEYtOAoKPGRpd
iBkaXI9Imx0ciI+dGhpcyBpcyB0aGUgZmlyc3QgbGluZSBvZiB0aGUgYm9keSBvZiBhbiBlbWFpbCBtZXNzYWdlLjxkaXY+PGJyPjwvZGl2Pjx
kaXY+YW5kIHRoaXMgaXMgdGhlIHNlY29uZCBsaW5lLjwvZGl2PjxkaXY+PGJyPjwvZGl2PjxkaXY+YW5kIHRoaXMgaXMgdGhlIGNsb3NpbmcsI
GNoZWVycyw8L2Rpdj48ZGl2Pjxicj48L2Rpdj48L2Rpdj4KCi0tMDQ3ZDdiZDc1ZjJhM2YzNGYzMDUxMDFkNWY5OS0tCg==" ,
      "metadata" : { },
      "annotations" : [ ] },
  ]
}
```

After processing by the Tika parser and the Field Mapping stage, the document contains additional fields, e.g.:

```
{ "id" : "/Users/mitzimorris/tmp/test_email.eml",
  "fields" : [ {
    "name" : "parsing_time_l",
    "value" : [ "java.lang.Long", 157 ],
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "subject",
    "value" : "this is the subject of email message",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  }, {
    "name" : "dcterms:created",
    "value" : "2015-02-28T03:13:41Z",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  }, ...
```

The following is an example of an empty PipelineDocument that issues a commit command on the index:

```
{ "fields" : [ ],
  "metadata" : { },
  "commands" : [ {
    "name" : "commit",
    "params" : { }
  } ]
}
```

6.2.3. Submitting PipelineDocuments Directly to a Pipeline via the REST API

PipelineDocuments can be submitted to a pipeline as a POST request to the Fusion REST API path:

`/api/apollo/index-pipelines/<id>/collections/<collectionName>/index`

where <id> is the name of an specific pipeline and <collectionName> is the name of a specific collection. The content type header to use for this format is:

`application/vnd.lucidworks-document`

Example

Send two documents to collection named "docs" using the "conn_solr" pipeline:

```
curl -u user:pass -X POST -H "Content-Type: application/vnd.lucidworks-document" -d '[{"id":"myDoc1",
"fields":[{"name":"title", "value":"My first document"}, {"name":"body", "value":"This is a simple
document."}]}, {"id":"myDoc2", "fields":[{"name":"title", "value":"My second document"}, {"name":"body",
"value":"This is another simple document."}]}]' http://localhost:8764/api/apollo/index-
pipelines/conn_solr/collections/docs/index
```

Limitations of using the index-pipelines REST API

All fields of a pipeline document sent to a pipeline as an HTTP POST request are treated as plain text strings. If you encode data in a binary format as a BASE64-encoded string, you must then add a stage to decode that data before the Tika Parser stage, else it will be parsed as plain text by Tika.

6.3. Index Pipeline Stages

An Index Pipeline takes content and transforms it into a document suitable for indexing by Solr via a series of modular operations called stages. The objects sent from stage to stage are PipelineDocument objects. Fusion provides many specialized index stages as well as a JavaScript Index stage that allows for custom processing via a JavaScript program. The general outline of theExtract/Transform/Load processing performed by an index pipeline is:

- Raw content is parsed into one or more PipelineDocument objects.
- Any number of intermediate stages operate on the document fields directly, or, in the case of specialized NLP tools, add annotations to a document.
- Finally, the PipelineDocument is sent to Solr for indexing.

A pipeline stage definition associates a unique ID with a set of properties. Pipeline definitions are stored in ZooKeeper for reuse across pipelines and search applications. The Fusion UI provides stage-specific panels used to define and configure each pipeline stage. Alternatively, JSON can be used to specify the sequence of pipeline stages and registered via the Fusion REST API. Some stages require additional resources, e.g., text files which contain lists of names, synonyms, places, or binary files which NLP language models. These resources can be uploaded via the Fusion UI or the REST API.

Available index pipeline stages are listed below:

- Apache Tika Parser
- CSV Parser
- Date Parsing
- Detect Language
- Exclusion Filter
- Field Mapper
- Find Replace
- Fusion Pipeline
- Gazetteer Lookup Extractor
- HTML Transform
- Indexing RPC
- JDBC
- JSON Parser
- Javascript
- Language Detection
- Logging
- Logging Message
- Machine Learning
- Multi-value Resolver
- OpenNLP NER Extractor

- Part of Speech
- Regular Expression Extractor
- Regular Expression Filter
- Regular Expression Replacement
- Sentence Detection
- Set Property
- Short Field Filter
- Signal Formatter
- Solr Indexer
- Solr Partial Update Indexer
- XML Transform

6.4. Index Profiles

Index profiles allow your applications to send documents for indexing to a consistent endpoint (the profile alias) and change the backend index pipeline as needed. The profile is also a simple way to use one pipeline for multiple collections without any one collection "owning" the pipeline.

Associating a profile with a pipeline is simply a mapping, and the profile is considered an alias for the pipeline. The mapping can be managed in the UI or with the Index Profiles API.

The Profiles tab shows the index profiles on the left and the query profiles on the right. Hover over the name of a profile, and an **edit** button will appear to allow you to change the pipeline the profile is mapped to. Hover over the name of a pipeline, and you will be able to jump to edit that pipeline.

Click **Add Profile** to add a profile. The next screen will show a form allowing you to define the profile name and either select an existing pipeline or create a new pipeline with the name you choose. Click **Create** to save the new profile.

6.5. Entity Extraction

Fusion includes extensive entity extraction capabilities. Entity extraction is configured as an index pipeline stage and there are several stage types to correspond to the different types of entity extraction you'd like to perform on your documents. The different types are described in more detail below.

Many of the entity extraction capabilities require models or lookup files, and we have provided a number of these by default. You can find the files in `fusion/data/nlp/` but in order to use them in an index pipeline stage, you will need to load them to Solr using the Blob Store API.

6.5.1. Loading Models and Lookup Files to Solr

To load the files, you simply need to make a PUT request with the Blob Store API, as in this example:

```
curl -u admin:pass -X PUT --data-binary @data/nlp/models/en-sent.bin -H 'Content-type: application/octet-
stream' http://localhost:8764/api/apollo/blobs/sentenceModel.bin
```

Note that the endpoint is the ID of the file. You will use the ID you've assigned with the endpoint in the index pipeline definition to indicate the model or lookup file to use. If the ID is omitted from the request (which is possible with a POST request), a random ID will be assigned and it will be difficult to tell one stored blob from another.

6.5.2. Entity Extraction Capabilities

Lookup Lists

The lookup lists are the most numerous of the available files. Many of these are simple lists that you will want to add values to. However, some may be robust enough for your needs.

The available lists are found in `fusion/data/nlp/gazetteer`.

To use a lookup list-based entity extraction, you would configure a Gazetteer Lookup Extractor Index Stage as part of your pipeline.

OpenNLP Models

We have also included entity extraction models from the OpenNLP project. These models are based on news articles and may not be suitable for all entity extraction needs. The Fusion-supplied models are located in `fusion/data/nlp/models`.

If you have created your own model for your data, you can load it to the blob store and use it in the nlp index stage as described above.

To use an OpenNLP-based entity extraction, you would configure a OpenNLP NER Extractor Index Stage as part of your pipeline.

Regular Expression Extraction

The regular expression extraction allows using a regular expression to find entities in documents that should be extracted. The extracted entities will then be copied to a field defined by you.

To use regular expression extraction, you would configure a Regular Expression Extractor Index Stage as part of your pipeline.

Regex Field Filter

The Regex Field Filter stage allows you to remove a field based on a regular expression. This removes the entire field from the document, there is not yet an option for removing only specific entities found in the field or for excluding entire documents based on values found in a field.

To use Regex Field Filter, you would configure a Regular Expression Filter Index Stage as part of your pipeline.

Exclusion Lists

An exclusion list is a list of known items that should be removed from a document. This removes the entire field from the document, there is not yet an option for removing only specific entities found in the field or for excluding entire documents based on values found in a field.

Note that the list must be loaded to Solr using the Blob Store API before it can be used with a Fusion index pipeline.

To use exclusion list filtering, you would configure a Exclusion Filter Index Stage as part of your pipeline.

Filter Short Fields

The Filter Short Fields stage removes entities that are equal to or smaller than a defined character limit.

To use the Filter Short Fields stage, you would configure a Short Field Filter Index Stage as part of your pipeline.

6.6. Pushing Documents to a Pipeline

Documents can be sent directly to an index pipeline using the Index-Pipelines REST API. The request path is:

`/api/apollo/index-pipelines/<id>/collections/<collectionName>/index`

where <*id*> is the name of an specific pipeline and <*collectionName*> is the name of an specific collection.

These requests are sent as a POST request. The request header specifies the format of the contents of the request body.

To send in a streaming list of JSON documents to the index pipeline you can send the JSON file which holds these objects to the API listed above with "application/json" as the content type. If your JSON file is a list/array of many items, the pipeline will operate in a streaming way and index the docs as necessary.

Example:

The JSON document called "myJsonDoc.json" holds 4.3M entries. Send the document to the index pipeline with the following command:

```
curl -u user:password -X POST -H 'Content-Type: application/json' -d@myJsonDoc.json
"http://localhost:8764/api/apollo/index-pipelines/<id>/collections/<collectionName>/index"
```

Documents can be created on the fly using the PipelineDocument JSON notation. See Fusion PipelineDocument Objects for details and an example of how to do this.

6.6.1. Indexing PDFs and MS Office Documents

If you can access the filesystem in which the PDFs or MS Office documents reside, you can index these documents using properly configured datasource with the appropriate connector for that filesystem type. See Connectors and Datasources Reference for a list of all Fusion connectors.

If, however, there are obstacles to using the connectors, it may be simpler to index these types of documents with an index pipeline. The pipelines can only be used with REST API calls, and there is complete documentation in the section Index Pipelines API.

When sending the documents, it’s important to set the content type header properly for the content being sent. This is not a complete list, but here are some frequently used content types:

- PDF documents: `application/pdf`
- MS Office:
  - .docx: `application/vnd.openxmlformats-officedocument.wordprocessingml.document`
  - .xlsx: \`application/vnd.openxmlformats-officedocument.spreadsheetml.sheet \`
  - .pptx: \`application/vnd.vnd.openxmlformats-officedocument.presentationml.presentation \`
  - More types: http://filext.com/faq/office_mime_types.php
- Text: `text/json`, `text/xml`, `text/csv`, etc.

Examples

Index a PDF document through the 'conn_solr' index pipeline to a collection named 'docs'. The pre-configured 'conn_solr' pipeline includes stages to parse documents with Tika, map fields, and index the documents to Solr (in that order).

```
curl -u user:pass -X POST -H "Content-Type: application/pdf" --data-binary @/solr/core/src/test-
files/mailing_lists.pdf http://localhost:8764/api/apollo/index-pipelines/conn_solr/collections/docs/index
```

Index one of the example Solr XML documents found in the `./example/exampledocs` directory of Solr. For this example to work well, the default pipeline was modified to include a field mapping stage in addition to indexing the documents to Solr. In this example, the custom pipeline is named 'docs-default' and the collection is 'docs'.

```
curl -u user:pass -X POST -H "Content-Type: text/xml" --data-binary @/Applications/solr-
4.10.0/example/exampledocs/hd.xml http://localhost:8764/api/apollo/index-pipelines/docs-
default/collections/docs/index
```

6.6.2. Indexing CSV Files

In the usual case, to index a CSV (or TSV) file, the file is split into records, one per row, and each row is indexed as a seperate document. Datasources which use crawlers that are based on either the lucid.anda or lucid.fs framework can do the CSV splitting as part of the connector process.

Alternatively, the index pipeline can include a CSV Parser stage.

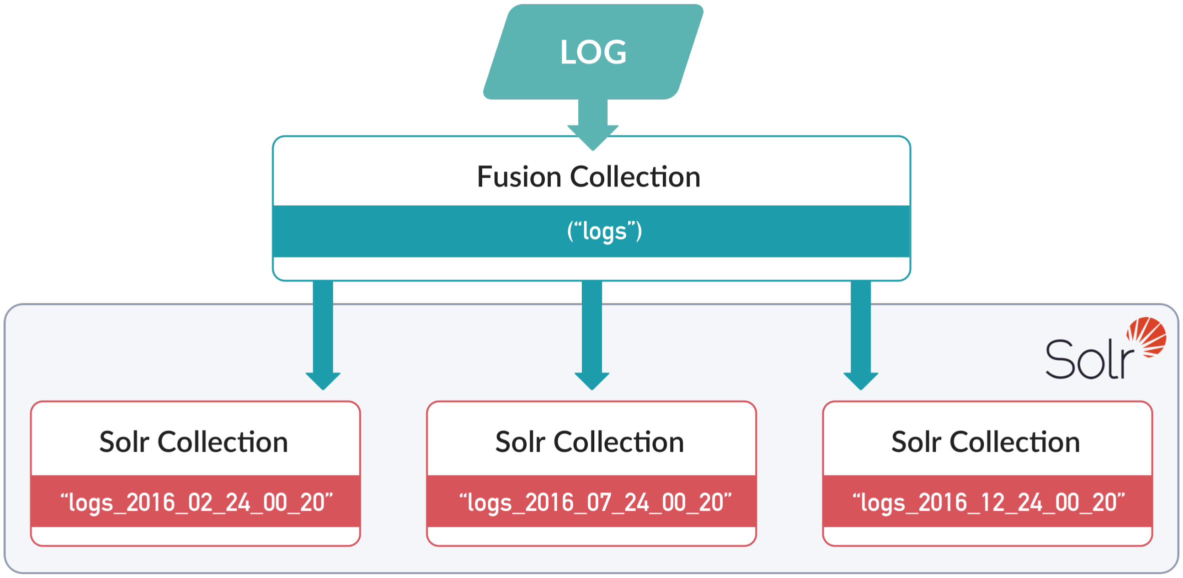
6.7. Blob Storage

Fusion accepts large binary objects (blobs) for upload, and stores them in Solr. Blob uploads are used to install models, lookup lists, JDBC drivers, connectors, and more.

In Fusion versions earlier than 3.1, it is only possible to upload these blobs with the Blob Store API.

6.8. Time-Based Partitioning

A Fusion collection can be configured to map to multiple Solr collections, known as partitions in this context, where each partition contains data from a specific time range. An example is time-based partitioning for logs:



Once a collection is configured for time-base partitioning, Fusion automatically ages out old partitions and creates new ones, using the configured partition sizes, expiration intervals, and so on. No manual maintenance is needed.

This feature is not enabled by default. Enable it for each collection using the Collection Features API.

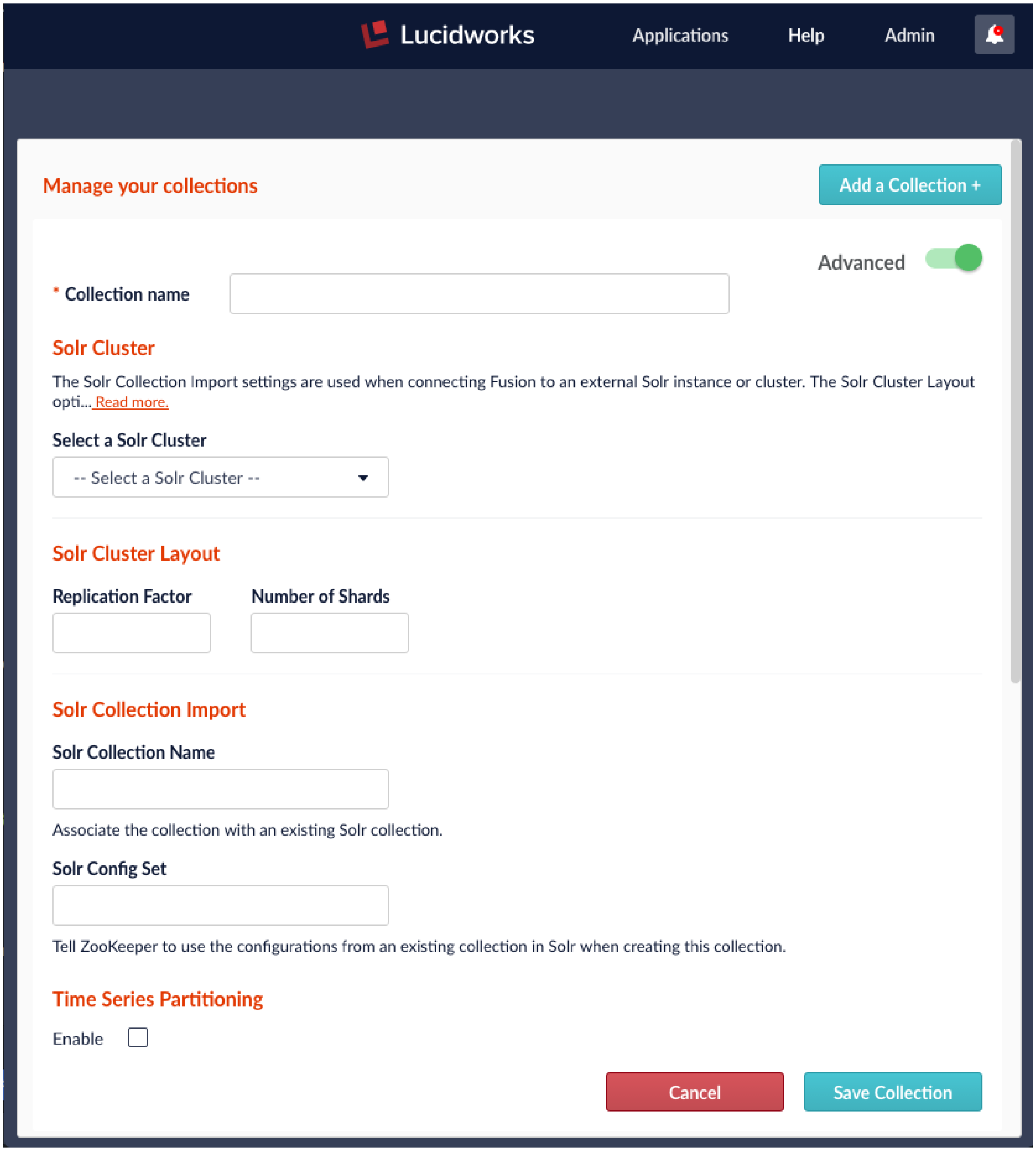
Note	Fusion cannot retroactively partition data that has already been indexed. It can only perform time-based partitioning on incoming data.

6.8.1. Enabling time-based partitioning

- In the UI, you can only enable time-based partitioning for *new* collections.
- In the API, you can only enable time-based partitioning for *existing* collections.

Enablement using the Fusion UI

1. While creating a collection, click **Advanced**.
2. Scroll down to "Time Series Partitioning".
3. Click **Enable**.



Manage your collections

Advanced ●

* Collection name

Solr Cluster

The Solr Collection Import settings are used when connecting Fusion to an external Solr instance or cluster. The Solr Cluster Layout opti... Read more.

Select a Solr Cluster: -- Select a Solr Cluster --

Solr Cluster Layout

Replication Factor

Number of Shards

Solr Collection Import

Solr Collection Name

Associate the collection with an existing Solr collection.

Solr Config Set

Tell ZooKeeper to use the configurations from an existing collection in Solr when creating this collection.

Time Series Partitioning

Enable □

4. Save the collection.

Currently, you cannot use the UI to enable time-based partitioning for an existing collection.

Enablement using the API

Use the Collection Features API to enable time-based partitioning for an existing collection.

Enable time-based partitioning using the default configuration:

```
curl -X PUT -H 'Content-type: application/json' -d '{"enabled": true}'
http://localhost:8765/api/v1/collections/<collection>/features/partitionByTime
```

No response is returned.

Submit an empty request to the same endpoint to verify that time-based partitioning is enabled:

```
curl -X GET http://localhost:8765/api/v1/collections/<collection>/features/partitionByTime
```

Response:

```
{
  "name" : "partitionByTime",
  "collectionId" : "<collection>",
  "params" : { },
  "enabled" : true
}
```

To change the configuration, see the options and examples below.

6.8.2. Configuration options

When time series indexing is enabled for a collection, you can configure these options using the UI or the Collections API. None are required.

UI Label, API Name	**Description**
Timestamp Field Name `timestampFieldName`	The name of the field from which to read timestamps. The default is "timestamp".
Partition Time Period `timePeriod`	The time range for each partition. The default is one day.
Max Active Partitions `maxActivePartitions`	The number of partitions to keep active.
Delete Expired Partitions `deleteExpired`	"True" to automatically delete partitions that fall outside of the `maxActivePartitions` window, at intervals of `scheduleIntervalMinutes`. The default is "false".
Preemptive Create Enabled `preemptiveCreateEnabled`	"True" (the default) to create partitions in advance.
Schedule Interval `scheduleIntervalMinutes`	The interval, in minutes, at which to perform background maintenance, including preemptively creating partitions (`preemptiveCreateEnabled`) and deleting expired partitions (`deleteExpired`). The default is five minutes.
Partition Num Shards `numShards`	The number of shards per partition. The default is the value configured for the main Fusion collection.

UI Label, API Name	Description
Partition Replication Factor `replicationFactor`	The number of copies to keep, per partition. The default is the value configured for the main Fusion collection.
Partition Config Name `configName`	The name of the Solr configuration set to be applied to new partitions; the default is the configuration used by the primary collection.

6.8.3. Examples

Create a new collection called "TimeSeries1":

```
curl -X PUT -H 'Content-type: application/json' -d '{
  "solrParams": {
    "numShards": 1,
    "replicationFactor": 1
  }
}' http://localhost:8765/api/v1/collections/TimeSeries1
```

Enable and configure time-based partitioning for the "TimeSeries1" collection:

```
curl -X PUT -H 'Content-type: application/json' -d '{
  "enabled": true,
  "timestampFieldName": "ts",
  "timePeriod": "5MINUTES",
  "scheduleIntervalMinutes": 1,
  "preemptiveCreateEnabled": false,
  "maxActivePartitions": 4,
  "deleteExpired": true
}' http://localhost:8765/api/v1/collections/TimeSeries1/features/partitionByTime
```

Verify that time-based partitioning is enabled:

```
curl -X GET http://localhost:8765/api/v1/collections/TimeSeries1/features/partitionByTime
```

Import some sample data into this collection:

```
curl -X POST -H "Content-type:application/vnd.lucidworks-document" -d '[
  {
    "id": "1",
    "fields": [
      {
        "name": "ts",
        "value": "2016-02-24T00:00:01Z"
      },
      {
        "name": "partition_s",
        "value": "eventsim_2016_02_24_00_00"
      }
    ]
  },
  {
    "id": "2",
    "fields": [
      {
        "name": "ts",
        "value": "2016-02-24T00:05:01Z"
      },
      {
        "name": "partition_s",
        "value": "eventsim_2016_02_24_00_05"
      }
    ]
  },
  {
    "id": "3",
    "fields": [
      {
        "name": "ts",
        "value": "2016-02-24T00:10:01Z"
      },
      {
        "name": "partition_s",
        "value": "eventsim_2016_02_24_00_10"
      }
    ]
  },
  {
    "id": "4",
    "fields": [
      {
        "name": "ts",
        "value": "2016-02-24T00:15:01Z"
      },
      {
        "name": "partition_s",
        "value": "eventsim_2016_02_24_00_15"
      }
    ]
  },
  {
    "id": "5",
    "fields": [
      {
        "name": "ts",
        "value": "2016-02-24T00:20:01Z"
      },
      {
        "name": "partition_s",
        "value": "eventsim_2016_02_24_00_20"
      }
    ]
  }
]' http://localhost:8765/api/v1/index-pipelines/TimeSeries1-default/collections/TimeSeries1/index
```

View the Solr configuration for this collection:

```
curl -X GET "http://localhost:8765/api/v1/query-pipelines/TimeSeries1-
default/collections/TimeSeries1/select?q=*:*"
```

The response includes a list of active Solr collections that correspond to this Fusion collection:

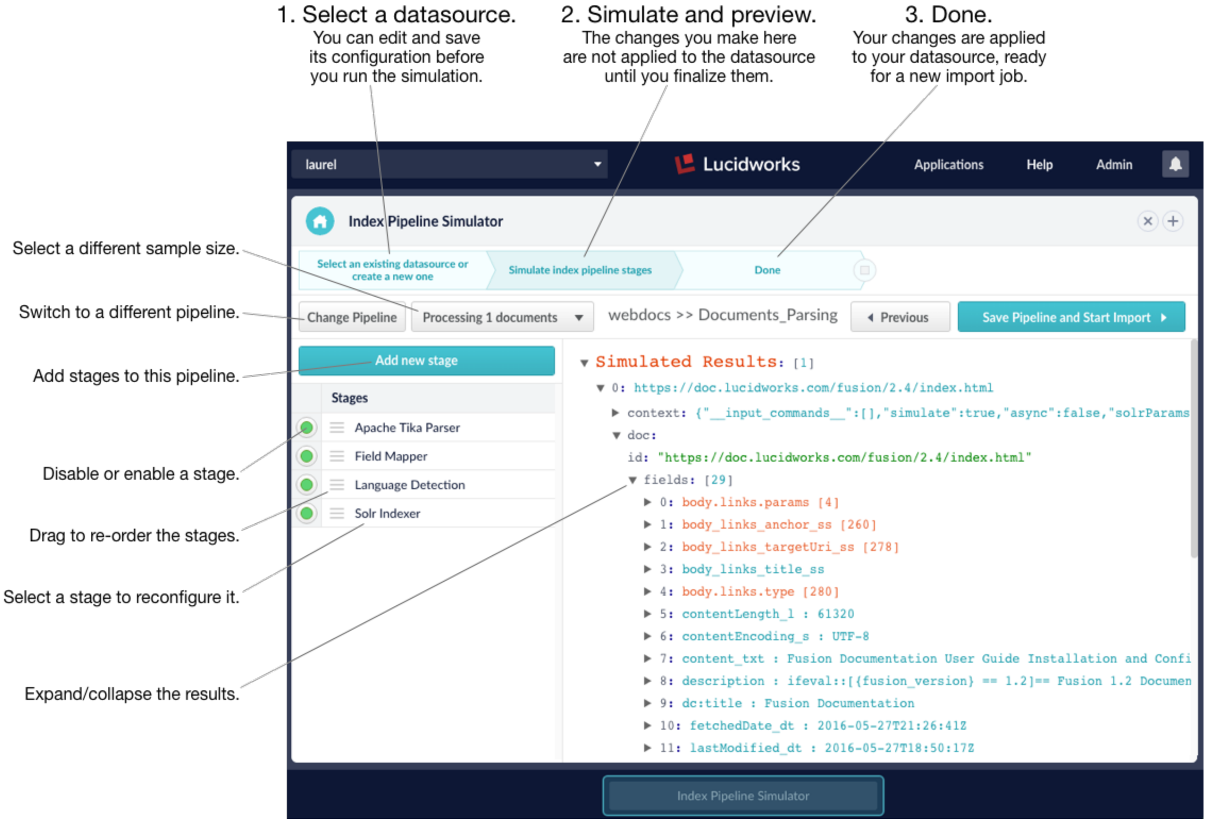
```
<str
name="collection">TimeSeries1_2016_02_24_00_05,TimeSeries1_2016_02_24_00_10,TimeSeries1_2016_02_24_00_15,TimeS
eries1_2016_02_24_00_20</str>
```

6.9. Index Pipeline Simulator

The Index Pipeline Simulator is a tool for developing index pipelines and previewing their output before you apply them to your live datasources.

The simulator displays a sample of your data as it would be indexed after processing by the configured pipeline stages. You can enable, disable, add, remove, or reconfigure the pipeline stages to see how your changes affect the data sample.

None of your changes are applied to your live datasources until you click the **Save Pipeline and Start Import** button.



1. Select a datasource.
You can edit and save
its configuration before
you run the simulation.

2. Simulate and preview.
The changes you make here
are not applied to the datasource
until you finalize them.

3. Done.
Your changes are applied
to your datasource, ready
for a new import job.

Select a different sample size.

Switch to a different pipeline.

Add stages to this pipeline.

Disable or enable a stage.

Drag to re-order the stages.

Select a stage to reconfigure it.

Expand/collapse the results.

```
▼ Simulated Results: [1]
  ▼ 0: https://doc.lucidworks.com/fusion/2.4/index.html
    ▶ context: {"__input_commands__":[],"simulate":true,"async":false,"solrParams
    ▼ doc:
      id: "https://doc.lucidworks.com/fusion/2.4/index.html"
      ▼ fields: [29]
        ▶ 0: body.links.params [4]
        ▶ 1: body_links_anchor_ss [260]
        ▶ 2: body_links_targetUri_ss [278]
        ▶ 3: body_links_title_ss
        ▶ 4: body.links.type [280]
        ▶ 5: contentLength_l : 61320
        ▶ 6: contentEncoding_s : UTF-8
        ▶ 7: content_txt : Fusion Documentation User Guide Installation and Confi
        ▶ 8: description : ifeval::[{fusion_version} == 1.2]== Fusion 1.2 Documen
        ▶ 9: dc:title : Fusion Documentation
        ▶ 10: fetchedDate_dt : 2016-05-27T21:26:41Z
        ▶ 11: lastModified_dt : 2016-05-27T18:50:17Z
```

Here’s how to run an index pipeline simulation:

1. Navigate to the collection you want to work with.
2. In the Home panel, click **Index Pipeline Simulator**.
3. Select a datasource.
    - You can click an existing datasource and optionally modify its configuration.

Note	Any changes you make here will overwrite the original datasource configuration.

    - To create a new datasource, click the **Add** + button.

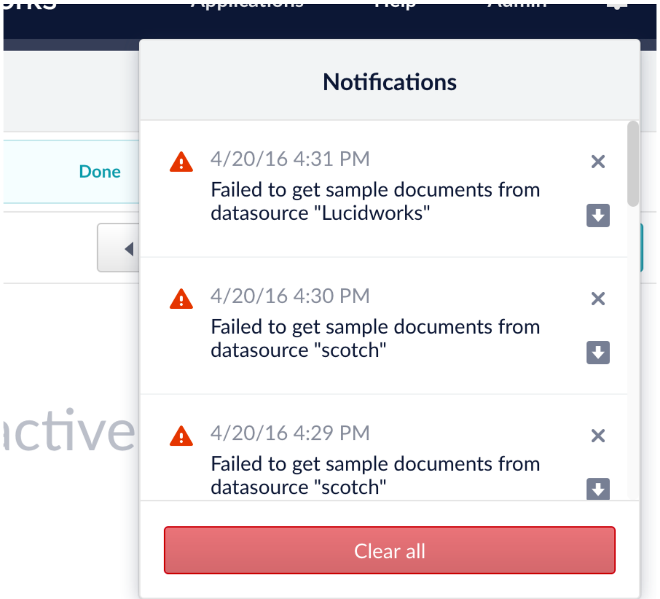
      You’ll be prompted to select a connector and import some data to preview. When you finish, the new datasource is added to this collection.
4. Click **Start simulation**.

Fusion collects a data sample from the selected source with which to run the simulation.

5. Select a pipeline to load.
6. Click **Load pipeline**.

   Your data sample is displayed in the format in which it would be indexed using the selected pipeline.

   You can enable, disable, add, remove, or reconfigure any pipeline stage to see a preview of how your changes affect the sample data.

Tip	 Applications Help Admin Done ictive Clear all Notifications 4/20/16 4:31 PM Failed to get sample documents from datasource "Lucidworks" 4/20/16 4:30 PM Failed to get sample documents from datasource "scotch" 4/20/16 4:29 PM Failed to get sample documents from datasource "scotch" If no data is displayed, download the error log from the Notifications menu in the upper right.

Note	When you click **Save** in the pipeline stage configuration panel, you’re saving a temporary copy of the stage configuration. *Your live datasource is not affected.*

7. Click **Save Pipeline and Start Import** to finalize your changes.

8. To start indexing data using the updated pipeline, click **Go to datasource**, then click **Start Crawl**.

6.10. Custom JavaScript Stages For Index Pipelines

The JavaScript Index stage allows you to write a custom processing logic using JavaScript to manipulate Pipeline Documents and the index pipeline context. which will be compiled by the JDK into Java bytecode that is executed by the Fusion pipeline. The first time that the pipeline is run, Fusion compiles the JavaScript program into Java bytecode using the JDK's JavaScript engine.

For a JavaScript Index stage, the JavaScript code must return either: a single document or array of documents; or the null value or an empty array. In the latter case, no further processing is possible, which means that the document will not be indexed or updated.

6.10.1. JavaScript Index Stage Global Variables

JavaScript is a lightweight scripting language. The JavaScript in a JavaScript stage is standard ECMAScript. What a JavaScript program can do depends on the container in which it runs. For a JavaScript Index stage, the container is a Fusion index pipeline. The following global pipeline variables are available:

Name	Type	Description
`doc`	PipelineDocument	The contents of each document submitted to the pipeline. See: PipelineDocument Objects for a complete description of this object.
`ctx`	Context	A reference to the container which holds a map over the pipeline properties. Used to update or modify this information for downstream pipeline stages.
`collection`	String	The name of the Fusion collection being indexed or queried.
`solrServer`	BufferingSolrServer	The Solr server instance that manages the pipeline's default Fusion collection. All indexing and query requests are done by calls to methods on this object. See SolrClient for details.
`solrServerFactory`	SolrClientFactory	The SolrCluster server used for lookups by collection name which returns a Solr server instance for a that collection, e.g. `var productsSolr = solrServerFactory.getSolrServer("products");`

Note	The now-deprecated global variable "_context" refers to the same object as "ctx".

The JavaScript in a JavaScript Index stage must return either a single document or an array of documents. This can be accomplished by either:

- a series of statements where the final statement evaluates to a document or array of documents
- a function which returns a document or an array of documents

As of Fusion 2.4, all pipeline variables referenced in the body of the JavaScript function must be passed in as arguments to the function. E.g., in order to access the PipelineDocument in global variable 'doc', the JavaScript function **must** be written as:

```
function doWork(doc) {
    // do some work ...
    return doc;
}
```

The allowed set of function declarations are:

```
function doWork(doc) {  ... return doc; }
function doWork(doc, ctx) {  ... return doc; }
function doWork(doc, ctx, collection) {  ... return doc; }
function doWork(doc, ctx, collection, solrServer) {  ... return doc; }
function doWork(doc, ctx, collection, solrServer, solrServerFactory) {  ... return doc; }
```

The order of these arguments is according to the (estimated) frequency of use. The assumption is that most processing only requires access to the document object itself, and the next-most frequent type of processing requires only the document and read-only access of some context parameters. If you need to reference the solrServerFactory global variable, you must use the 5-arg function declaration.

In order to use other functions in your JavaScript program, you can define and use them, as long as the final statement in the program returns a document or documents.

Global variable `logger`

The global variable named `logger` writes messages to the logfile of the server running the pipeline. This variable is truly global and doesn't need to be declared as part of the function parameter list.

Since Fusion's connectors service does the index pipeline processing, these log messages go into the logfile: `fusion/var/log/connector/connector.log`. There are 5 methods available, which each take either a single argument (the string message to log) or two arguments (the string message and an exception to log). The five methods are, "debug", "info", "warn", and "error".

6.10.2. JavaScript Index Stage Examples

Add a field to a document

```
function (doc) {
  doc.addField('some-new-field', 'some-value');
  return doc;
}
```

Join two fields

The following example conjoins separate latitude and longitude fields into a single geo-coordinate field, whose field name follows Solr schema conventions and ends in "_p". It also removes the original latitude and longitude fields from the document.

```
function(doc) {
  var value = "";
  if (doc.hasField("myGeo_Lat") && doc.hasField("myGeo_Long"))   {
    value = doc.getFirstFieldValue("myGeo_Lat") + "," + doc.getFirstFieldValue("myGeo_Long");
    doc.addField("myGeo_p", value);
    doc.removeFields("myGeo_Lat");
    doc.removeFields("myGeo_Long");
    logger.debug("conjoined Lat, Long: " + value);
  }
  return doc;
}
```

Return an array of documents

```
function (doc) {
  var subjects = doc.getFieldValues("subjects");
  var id = doc.getId();
  var newDocs = [];
  for (i = 0; i < subjects.size(); i++) {
     var pd = new com.lucidworks.apollo.common.pipeline.PipelineDocument(id+'-'+i );
     pd.addField('subject',  subjects.get(i));
     newDocs.push( pd  );
  }
  return newDocs;
}
```

Parse a JSON-escaped string into a JSON object

While it’s simpler to use a JSON Parsing index stage, the following code example shows you how to parse a JSON-escaped string representation into a JSON object.

This code parses a JSON object into an array of attributes, and then find the attribute "tags" which has as its value a list of strings. Each item in the list is added to a multi-valued document field named "tag_ss".

```
var imports = new JavaImporter(Packages.sun.org.mozilla.javascript.internal.json.JsonParser);
function(doc) {
    with (imports) {
        myData = JSON.parse(doc.getFirstFieldValue('body'));
        logger.info("parsed object");
        for (var index in myData) {
            var entity = myData[index];
            if (index == "tags") {
                for (var i=0; i<entity.length;i++) {
                    var tag = entity[i][0];
                    doc.addField("tag_ss",tag);
                }
            }
        }
    }
    doc.removeFields("body");
    return doc;
}
```

Do a lookup on another Fusion collection

```
function doWork(doc, ctx, collection, solrServer, solrServerFactory) {
    var imports = new JavaImporter(
        org.apache.solr.client.solrj.SolrQuery,
        org.apache.solr.client.solrj.util.ClientUtils);
    with(imports) {
        var sku = doc.getFirstFieldValue("sku");
        if (!doc.hasField("mentions")) {
            var mentions = ""
            var productsSolr = solrServerFactory.getSolrServer("products");
            if( productsSolr != null ){
                var q = "sku:"+sku;
                var query = new SolrQuery();
                query.setRows(100);
                query.setQuery(q);
                var res = productsSolr.query(query);
                mentions = res.getResults().size();
                doc.addField("mentions",mentions);
            }
        }
    }
    return doc;
}
```

Reject a document

If the function returns `null` or an empty array, it will not be indexed or updated into Fusion.

```
function doWork(doc) {
 if (!doc.hasField("required_field")) {
    return null;
 }
 return doc;
}
```

6.10.3. Debugging and Troubleshooting

To debug a JavaScript Index stage you can:

- Check the Fusion api server logs for compilation errors.
- Check the Fusion connectors server logs for runtime processing errors.
- Use the `logger` object for print debugging (in the Fusion connectors logfile).
- Use the Pipeline Preview tool (not available in Fusion 2.0, 2.1, or 2.2).

6.10.4. The JavaScript Engine Used by Fusion

The JavaScript engine used by Fusion is the Nashorn engine from Oracle. See The Nashorn Java API for details.

Upgrading to the latest Nashorn engine

The default version of the Nashorn engine used by Fusion versions 2.4.1 and earlier is the nashorn-0.1-jdk7.jar which contains many bugs that have since been fixed in the official JDK 1.8 version. In order to use the latest version of the Nashorn engine, you must:

- Have an up-to-date version of Java 8 installed.
- Remove the nashorn-0.1-jdk7.jar from the Fusion classpaths:
  - `cd fusion`
  - `find . -name "nashorn-0.1-jdk7.jar" -print -exec rm -i {} \;`

Creating and accessing Java types

The following information is taken from Oracle’s JavaScript programming guide section 3, Using Java From Scripts.

To create script objects that access and reference Java types from Javascript use the `Java.type()` function:

```
var ArrayList = Java.type("java.util.ArrayList");
var a = new ArrayList;
```

Chapter 7. Search

- Datasources are the configurations that import and index data into a collection.
- Query pipelines filter, transform, and augment Solr queries and responses in order to return all and only the most relevant search results.
- Signals and aggregations are ways to collect and compile data for analysis or to boost search results in the future.
- Search applications are the front-end interfaces that you build on top of Fusion using its REST API.

7.1. Query Pipelines

A Query Pipeline transforms a set of inputs into a Solr query request and it can execute requests and manipulate the Solr response as well, via a set of modularized operations called Query Stages. The objects sent from stage to stage are Request objects and Response objects.

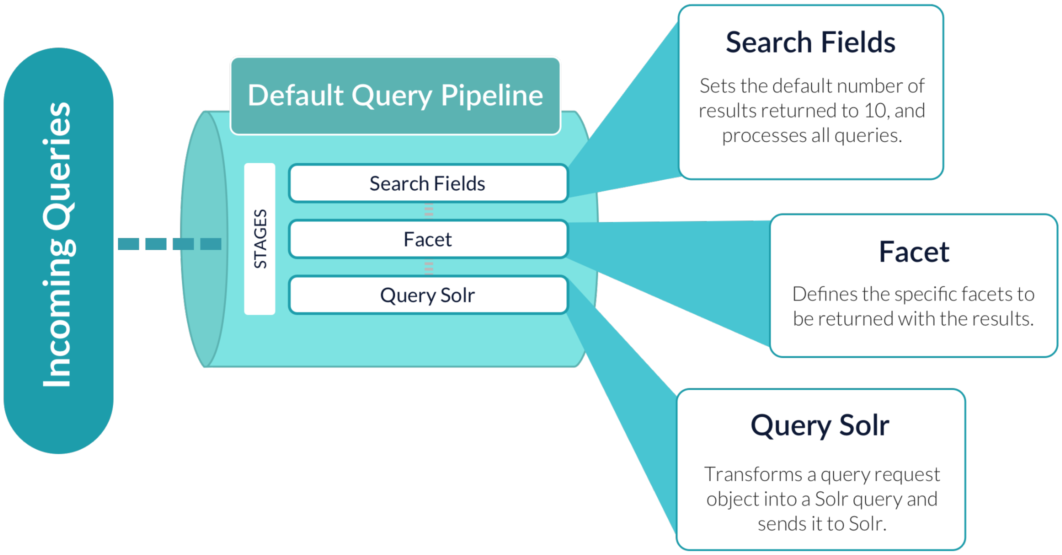
Fusion stores pipeline names and definitions, allowing a pipeline to be reused across applications. Pipeline definitions can be modified, so that as an application evolves, the pipelines used by that application can evolve accordingly. During application development, the Fusion UI can be used to develop and debug a Query Pipeline.

The available stage types allow setting specific parameters for the query, such as the number of results to return or the query parser to use. You can also define facets and recommendations to be returned with the results. If Access Control Lists (ACLs) are in use, you can apply a security-trimming stage to apply user access restrictions to the results.

For details about the available REST APIs, see Query Pipelines API and Query Stages API.

7.1.1. Default Query Pipelines

Fusion ships with one default query pipeline named 'default'.


Incoming Queries
Default Query Pipeline
STAGES
Search Fields
Facet
Query Solr
Search Fields
Sets the default number of results returned to 10, and processes all queries.
Facet
Defines the specific facets to be returned with the results.
Query Solr
Transforms a query request object into a Solr query and sends it to Solr.

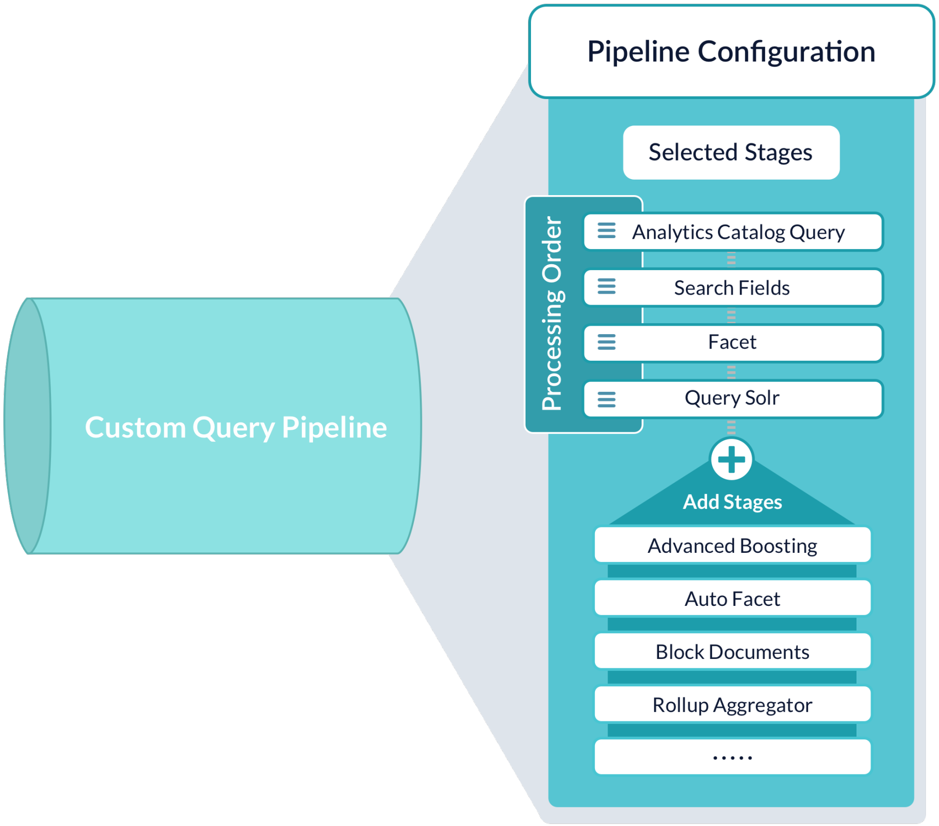
This pipeline has the following pre-configured stages:

- A Recommendation Boosting stage which boosts based on signals (if signals are collected and sent to the app).
- A Search Fields stage which sets the default number of results returned to 10. The property "skip" is false and the property "condition" is empty, so that all queries are processed.
- A Facet stage. No facet fields are specified. As configured, this stage has no effect on the pipeline.
- A Query Solr stage which sends the full query request to Solr.

When a collection is created, a default query pipeline for the collection is created. The pipeline name is the collection name with '-default' appended. For example, collection *foo* will have default query pipeline *foo-default*. This pipeline has the same configuration as the pipeline named 'default'.

7.1.2. Custom Query Pipelines

Using the Query Workbench or the REST API, you can develop custom pipelines to suit any search application. Start with any of Fusion's built-in query pipelines, then add, remove, and re-order the pipeline stages as needed to produce the appropriate query results.



7.2. Fusion Query Request Objects

A Fusion Query Request organizes the contents of each query submitted to a Fusion query pipeline. A Request object is comprised of the HTTP method, a set of HTTP headers, and a set of query parameters.

7.2.1. The Request Java API

Under the Fusion hood, a Request is a Java object. The link to the public API javadoc page is: Request

7.2.2. JSON representation of a Request object

The JSON representation of a Request object has three fields:

- **httpMethod**: value is a string, one of the defined HTTP methods (verbs), usually GET.
- **headers** : value is an object consisting of the set of defined header fields on the request.
- **params** : value is an object consisting of the set of query parameter names and corresponding parameter values.

The httpMethod and headers reflect the initial pipeline request. The Fusion Request object public API doesn’t provide methods to update these fields.

The Fusion Request object public API provides methods to get and set the param fields and their values.

Query pipeline stages add, remove, and update the values in the params field. To see how the query parameters are affected by query pipeline stage, we insert a Logging Query stage at the very beginning of a Query Pipeline and capture the request object printed to the Fusion log file fusion/logs/api/api.log. The example query pipeline used here is called "medsamp-default." This pipeline is composed of the following sequence of stages:

- Logging Query
- Query Fields
- Logging Query
- Facet
- Logging Query
- Solr Query

Using the Admin Search tool, we submit the query "oxygen". The initial Logging Query stage shows the Request object. The "params" field takes an object consisting of a set of param names, param values. The param name "q" takes as its value the list of query terms, which in this case, is a list consisting of the word "oxygen":

```
{
  "headers" : {
    "Accept-Collections" : [ "*" ],
    "Fusion-User-Auth-Realm" : [ "native" ],
    "Host" : [ "172.16.2.12:8765" ],
    "Content-Length" : [ "0" ],
    "Fusion-Session-Id" : [ "e7085302-f5c5-4401-ba79-a4ead8a9f65a" ],
    "Accept-Encoding" : [ "gzip, deflate" ],
    "User-Agent" : [ "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/41.0.2272.89 Safari/537.36" ],
    "Via" : [ "1.0 apollo-proxy" ],
    "Fusion-User-Role" : [ "admin" ],
    "Accept" : [ "application/json, text/plain, */*" ],
    "Content-Type" : [ "application/json; charset=UTF-8" ],
    "Fusion-User-Name" : [ "admin" ],
    "Fusion-User-Id" : [ "a4096f7e-2de8-4910-beb4-93f30a3a5eb6" ]
  },
  "params" : {
    "echoParams" : [ "all" ],
    "start" : [ "0" ],
    "lw.pipelineId" : [ "medsamp-default" ],
    "q" : [ "oxygen" ],
    "json.nl" : [ "arrarr" ],
    "wt" : [ "json" ]
  },
  "httpMethod" : "GET"
}
```

The Query Fields stage updates the Request object, and the second Logging Query stage reflects these changes. Here we have omitted the "headers", as they are unchanged (and unchangable). The Query Fields stage added params "fl" (fields list - document fields returned), "qf" (query fields - the fields used for search), and "defType" (the default scoring type) to the params set.

```
{
  "headers" : { ... }
  "params" : {
    "fl" : [ "article-title_txt", "article-vernacular-title_txt", "article-author-lastname_txt", "mesh-
heading_ss", "pmid_txt", "article-language_s", "article-abstract_txt", "article-author-affiliation_txt" ],
    "echoParams" : [ "all" ],
    "start" : [ "0" ],
    "lw.pipelineId" : [ "medsamp-default" ],
    "q" : [ "oxygen" ],
    "qf" : [ "article-title_txt", "article-abstract_txt" ],
    "json.nl" : [ "arrarr" ],
    "wt" : [ "json" ],
    "rows" : [ "10" ],
    "defType" : [ "edismax" ]
  },
  "httpMethod" : "GET"
}
```

The Facets stage further updates the Request object, and the third Logging Query stage reflects these changes. The params set now includes param "facet".

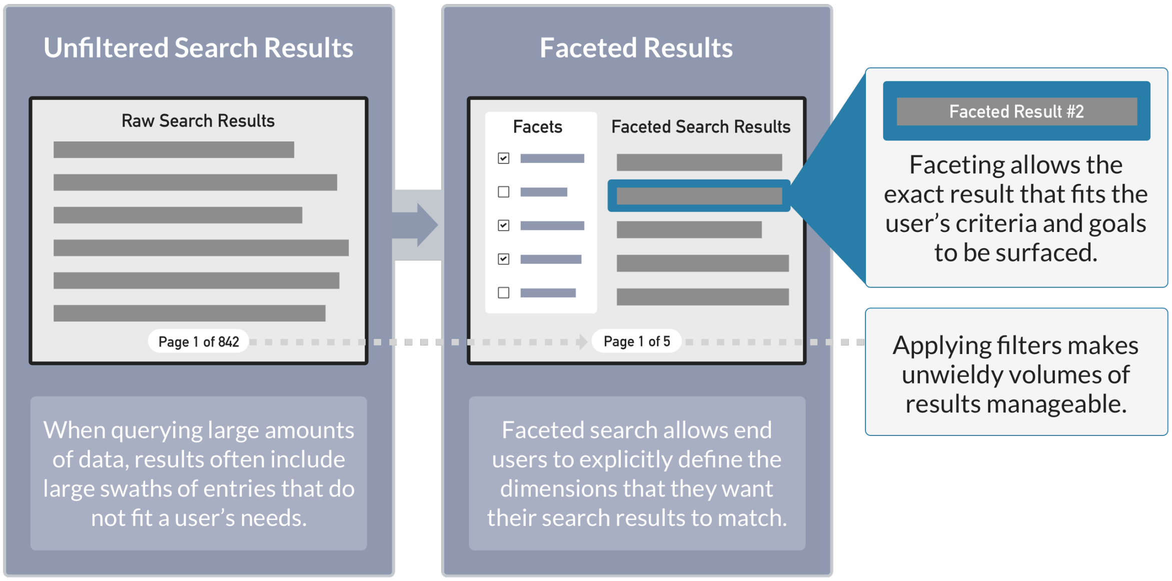
```
 {
  "headers" : { ... }
  "params" : {
    "facet" : [ "true" ],
    "fl" : [ "article-title_txt", "article-vernacular-title_txt", "article-author-lastname_txt", "mesh-
heading_ss", "pmid_txt", "article-language_s", "article-abstract_txt", "article-author-affiliation_txt" ],
    "echoParams" : [ "all" ],
    "start" : [ "0" ],
    "lw.pipelineId" : [ "medsamp-default" ],
    "q" : [ "oxygen" ],
    "qf" : [ "article-title_txt", "article-abstract_txt" ],
    "json.nl" : [ "arrarr" ],
    "wt" : [ "json" ],
    "rows" : [ "10" ],
    "defType" : [ "edismax" ]
  },
  "httpMethod" : "GET"
}
```

7.3. Fusion Query Response Objects

A Fusion query Response object contains the Solr response as well as a set of HTTP headers. It is used to improve the search experience by refining, expanding, filtering, or otherwise modifying the Solr response.

Under the Fusion hood, a Response is a Java object. The link to the public API javadoc page is: Response.

7.4. Faceting



Unfiltered Search Results
Raw Search Results
Page 1 of 842
When querying large amounts of data, results often include large swaths of entries that do not fit a user's needs.
Faceted Results
Facets
Faceted Search Results
Page 1 of 5
Faceted search allows end users to explicitly define the dimensions that they want their search results to match.
Faceted Result #2
Faceting allows the exact result that fits the user's criteria and goals to be surfaced.
Applying filters makes unwieldy volumes of results manageable.

Faceting is the name given to a set of computed counts over a search result which are returned together with the documents which match the search query. Facets are most often used to create additional navigational controls on the search results page or panel which allow users to expand and restrict their search criteria in a natural way, without having to construct complicated queries. For example, popular e-commerce facets include product category, price range, availability, and user ratings.

Fusion leverages Solr's Faceting search components.

7.4.1. Field faceting

In Solr the most straightforward kind of faceting is field faceting, in which Solr's FacetComponent computes the top values for a field and returns the list of those values along with a count of the subset of documents in the search results which match that term. Field faceting works best over fields which contain a single label or set of labels from a finite, controlled lexicon such as product category. Fusion's Facet Query Stage can be used to configure field faceting as part of the search query pipeline.

7.4.2. Range faceting

Range facets are used for fields which contain date or number values. Values can be grouped into ranges by specifying additional query parameters.

To configure range faceting, use the Additional Query Parameters Stage to specify Solr range faceting parameters.

7.4.3. Faceting concepts

Key Facet Concepts:

Term

A specific value from a field.

Limit

The maximum number of terms to be returned.

Offset

The number of top facet values to skip in the response (just like paging through search results and choosing an offset of 51 to start on page 2 when showing 50 results per page).

Sort

The order in which to list facet values: `count` ordering is by documents per term, descending, and `index` ordering is sorted on term values themselves.

Missing

The number of documents in the results set which have no value for the facet field.

Choice of facet method

(advanced) Specify Solr algorithm used to calculate facet counts. (See Facet Method Configuration for details). One of: +

- `enum` - small number of distinct categories
- `fc` ("field cache") - many different values in the field, each document has low number of values, multi-valued field
- `fcs` ("single value string fields") - good for rapidly changing indexes

7.4.4. Further Reading

https://lucidworks.com/blog/2014/10/03/pivot-facets-inside-and-out/

https://lucidworks.com/blog/2015/01/29/you-got-stats-in-my-facets/

https://lucidworks.com/blog/2016/08/12/pivoting-to-the-query-using-pivot-facets-to-build-a-multi-field-suggester/

7.5. Search Query Pipeline Stages

A query pipeline is made up of a series of query stages that process incoming search queries.

A pipeline stage definition associates a unique ID with a set of properties. These definitions are registered with the Fusion API service and stored in ZooKeeper for re-use across pipelines and search applications.

Fusion includes a number of specialized query stages as well as a JavaScript stage that allows advanced processing via a JavaScript program.

7.5.1. Configuring query pipeline stages

- In the Fusion UI, the Query Workbench provides an environment for configuring the stages in a query pipeline.
- The Query Stages API is used used to create, list, update, or delete query stages using JSON. See also the Query Pipelines API.

7.5.2. Conditional query processing

Query Pipeline stages are used to modify Request objects and Response objects. Each stage can include a conditional JavaScript expression (the 'condition' property in its configuration) that can access these objects.

For example, this condition first checks that the property "fusion-user-name" is specified in the Request object, then checks for a particular value:

```
request.hasParam("fusion-user-name") && request.getFirstParam("fusion-user-name").equals("SuperUser");
```

7.5.3. Reference topics

See these reference topics for complete details about each query pipeline stage:

- Active Directory Security Trimming
- Advanced Boosting
- Auto Facet
- Block Documents
- Boost Documents
- Email
- Facet
- Javascript
- JDBC
- Landing Pages
- Logging
- Logging Message
- Machine Learning
- PagerDuty

- Recommendation Boosting
- Return QueryParams
- Rollup Aggregator
- Search Fields
- Security Trimming
- Set Query Params Stage
- Slack
- Solr Query
- Stored Parameters
- SubQuery
- User Recommendation Boosting

7.6. Query Profiles

Query profiles allow you to consistently point your search application at a static endpoint, but give you the flexibility to change the actual query pipeline being used.

For example, an e-commerce site may want to create a query pipeline to support a month-long promotion. Once the pipeline is configured, it can be easily enabled by changing the profile in use by the front-end application to use the new pipeline.

A profile can be created or modified with the UI or with a REST API. The UI is described below, the REST API is described in the section Query Profiles API.

7.6.1. Query Profiles in the UI

The Profiles tab shows the index profiles on the left and the query profiles on the right. Hover over the name of a profile, and an **edit** button will appear to allow you to change the pipeline the profile is mapped to. Hover over the name of a pipeline, and you will be able to jump to edit that pipeline.

Click **Add Profile** to add a profile. The next screen will show a form allowing you to define the profile name and either select an existing pipeline or create a new pipeline with the name you choose. Click **Create** to save the new profile.

7.7. Fusion UI Search Tool

The Fusion UI Search Tool simulates the typical search application experience by running a user query over a collection using a specified query pipeline. This tool can display the query as well as the results, so that specific queries can be embedded in an application. It offers ways to customize the display of the Solr query results, but not the search parameters. You can also use it to compare the results of two query pipelines.

Query pipeline configuration is done through the Query Pipeline tool.

To access this tool, you can either click on the icon for the Fusion UI Search Tool, shown above, which is found on the main Fusion UI page, (the page displayed immediately after initial login and/or is reached by clicking on the Lucidworks Fusion label in the very top left-hand corner of the UI), or you can enter the URL http://<host:port>/search, where <host> and <port> point to your Fusion UI instance.

7.7.1. Search Tool Elements

The search tool provides drop-down selectors for the Collection to be searched and a text box for the search query. Next to these is a gear icon. The gear icon toggles a configuration panel which is used to select the search pipeline that will carry out the search and to specify the number of search results to display and how these results are displayed. See section Search Tool Options, below.

The space below the selectors is used to display documents and/or system messages.

The left-hand side is used to display facets. Facets will only be displayed if the query-pipeline response contains facets and according to the configuration settings (see below).

7.7.2. Search Results Display

The search results display panel displays the following:

- The number of results found and the query time.
- If the "Display Query URL" option is on, it will display the Solr query sent to the collection
- A set of controls used to display the documents retrieved by the query (if any). The set of controls are, from left to right
  - a drop-down menu used to select the Solr response field to sort on
  - a drop-down menu used to select the sort order
  - controls to page through the search results
  - a control box allowing you to choose the number of results per page, either 10, 20, 50, or 100 results per page.
- The documents themselves

7.7.3. Search Tool Options

The configuration panel toggles between three displays: "General", "Document", and "Facets".

The general search tool configuration options are:

- Results Per-Page: The maximum amount of results that can be displayed for each query.
- Display Solr Response Highlighting: If the Query Pipeline contains highlighting information, you can choose to show

or hide it here. To add highlighting to a query pipeline, add the following two parameters to the `set-query-params` stage: `hl=true` and `hl.fl=*`.

- Display Query URL: Display the full URL of the Query Pipeline search request.

The document configuration options are used to specify:

- "Primary" - a field which will be treated as the document title. The value of this field is displayed in bold text. The field name is omitted.
- "Secondary" - a field which will be treated as the document summary. The value of this field is displayed below the document title, in a smaller font. The field name is omitted.
- "Additional" - fields which will be displayed if the "more fields" label is expanded. These will display as a set of field name:value pairs.

The facets configuration option allow you to choose which facets are displayed to the left of the documents. This only applies if the query pipeline has a facets stage.

7.7.4. Comparing Query pipelines

You can use the search tool to compare the search results from two different query pipelines.

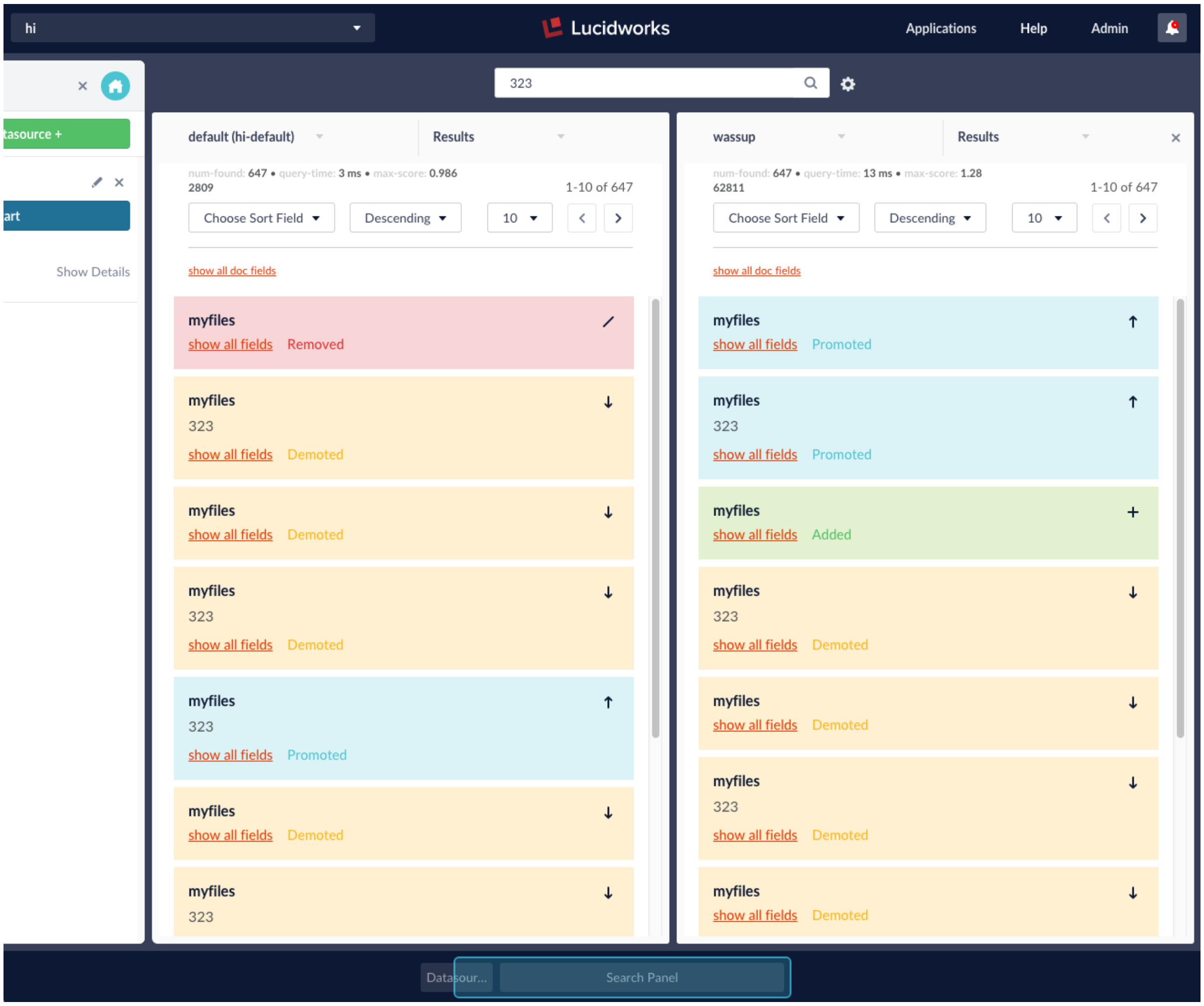
1. In the search panel, click the **Compare** button.

   A second search panel appears on the right, indicating that no pipeline is selected for comparison.

2. In the right-hand search panel, select the query pipeline to compare.

   You can select a preconfigured pipeline from the dropdown list, or click the **Duplicate** button to create a copy of the current pipeline for this datasource.

   Now the search results are displayed as a diff:



3. Modify the pipeline on the right as desired.

    - You can block or boost results directly in the Search Panel.
    - To edit one of the query pipelines, click **Home** > **Query Pipelines** > <**pipeline name**>.

7.8. Using Query Pipelines with SolrJ

Fusion Pipelines can be used in conjunction with a SolrJ client, allowing you to use the power of Fusion pipelines with an existing Solr installation.

7.8.1. Authentication with SolrJ

Fusion user authentication and authorization is carried out by the Fusion UI service. For details on how Fusion handles authentication and authorization, please see Access Control.

When using SolrJ, however, there are two approaches that can be used: basic authentication and passing credentials in the URL. Once the authentication has occurred, the roles that have been assigned to the user provide the authorization.

Basic Authentication

The Basic authentication approach looks very similar to the session-based approach. However, some classes are changed.

```
  HttpClient client = HttpClientBuilder.create().useSystemProperties()
      .addInterceptorLast(new PreEmptiveBasicAuthenticator(user, password))
      .build();
  HttpSolrServer server = new HttpSolrServer(url, client);
 // ...
public static class PreEmptiveBasicAuthenticator implements HttpRequestInterceptor {
  private final UsernamePasswordCredentials credentials;
  public PreEmptiveBasicAuthenticator(String user, String pass) {
    credentials = new UsernamePasswordCredentials(user, pass);
  }
  public void process(HttpRequest request, HttpContext context) throws HttpException, IOException {
    request.addHeader(BasicScheme.authenticate(credentials,"US-ASCII",false));
  }
}
```

URL Credentials

The URL credential approach provides the ability to pass the authentication properties in the URL, as in this example:

```
  String url = "http://user:pass@localhost:8764/api/apollo/solr/demo";
  HttpSolrServer server = new HttpSolrServer(url);
```

7.8.2. Example

The example below demonstrates the use of query profiles and query pipelines for querying a collection named 'test' with a query pipeline named 'default', using the basic authentication approach.

```
package com.lucidworks.apollo.testQueryPipeline;

import org.apache.http.HttpRequestInterceptor;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.solr.client.solrj.SolrServerException;
import org.apache.solr.client.solrj.impl.HttpSolrServer;
import org.apache.solr.client.solrj.impl.XMLResponseParser;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.apache.solr.common.params.ModifiableSolrParams;
public class TestQPSolrJ {
    public static void main(String[] args) throws SolrServerException {
        /**
         * Request URL points to Fusion UI which uses authentication proxy.
         * Example uses basic authentication
         * URL is Apollo query-pipelines endpoint:
         * http://{hostname}:{port_name}/api/apollo/query-
pipelines/{pipeline_name}/collections/{collection_name}
         */
        String url = "http://localhost:8764/api/apollo/query-pipelines/default/collections/demo";
        String user = "admin";
        String password = "password123";
        HttpClient client = HttpClientBuilder.create().useSystemProperties()
            .addInterceptorLast(new PreEmptiveBasicAuthenticator(user, password))
            .build();
        HttpSolrServer solrServer = new HttpSolrServer(url, client);
        solrServer.setParser(new XMLResponseParser());
        ModifiableSolrParams solrParams = new ModifiableSolrParams();
        solrParams.add("q", "*:*");
        QueryResponse out = solrServer.query(solrParams);
        System.out.println("QTime is " + out.getQTime());
        System.out.println("RH is " + out.getResponseHeader().toString());
    }
    public static class PreEmptiveBasicAuthenticator implements HttpRequestInterceptor {
        private final UsernamePasswordCredentials credentials;
        public PreEmptiveBasicAuthenticator(String user, String pass) {
            credentials = new UsernamePasswordCredentials(user, pass);
        }
        public void process(HttpRequest request, HttpContext context) throws HttpException, IOException {
            request.addHeader(BasicScheme.authenticate(credentials,"US-ASCII",false));
        }
    }

}
```

For more information about SolrJ, see the Apache Solr Reference Guide section Using SolrJ.

7.9. Search Query Reporting

Fusion includes several reports that can help you gain insight into the performance of your search application and the behavior of your users.

In order to use these reports, you must have the 'searchLogs' feature enabled for a collection. This will create a parallel collection named '<collection>_logs'. When requests for report data is sent to the main collection, the data is pulled from the *_logs collection.

7.9.1. Available Reports

lessThanN

This report provides a way to discover queries that returned less than a defined number of results.

In addition to defining the number of results for inclusion in the report, you can also limit by a date range.

topQueries

This report shows the queries that were performed most often. It reports the user’s entire query.

While it does not take a minimum number parameter, it can be limited by a date range.

topN

The TopN report finds the most popular terms. This is distinct from the topQueries report because it reports on terms and not the entire query.

Possible fields are those that appear when investigating a particular item:

- collection_s: the collection used for the query
- id: the document ID
- q_txt: the query text
- q_s:
- qtime_l: the length of time the query portion of the response
- totaltime_l: the total length of time for the response
- numdocs_l: the number of documents found for the query
- timestamp_dt: the timestamp of the query
- req_facet_ss: if facets were requested
- req_echoParams_ss: the setting for "echoParams" during the query
- req_q_ss: the request itself
- req_facet.field_ss: the facet fields requested
- req_rows_ss: the number of rows requested
- *version*: The document version at the time of the query

topClicked

This report shows the items that were clicked most often. It requires that Signals have been enabled for a collection, that click events have been recorded, and that the click signals have been aggregated. See the section on Signals for more information.

In addition to accepting a number to define how many items to return, it can also be limited by date range.

histo

This report provides a histogram that matches the query parameters. A field is defined as the basis for the data, and then a range of values and an interval are provided.

The available fields are:

- qtime_l: the length of time the query portion of the response
- totaltime_l: the total length of time for the response
- numdocs_l: the number of documents found for the query
- timestamp_dt: the timestamp of the query
- req_rows_ss: the number of rows requested

dateHisto

This report provides a histogram of queries based on a provided date range.

7.10. Custom JavaScript Stages for Query Pipelines

The JavaScript Query stage allows you to write a custom processing logic using JavaScript to manipulate search requests and responses. The first time that the pipeline is run, Fusion compiles the JavaScript program into Java bytecode using the JDK’s JavaScript engine.

The JavaScript Query stage allows you to run JavaScript functions over search requests and responses by manipulating variables called "request" and "response" which are Request and Response objects, respectively.

7.10.1. JavaScript Query Stage Global Variables

JavaScript is a lightweight scripting language. The JavaScript in a JavaScript stage is standard ECMAScript. What a JavaScript program can do depends on the container in which it runs. For a JavaScript Query stage, the container is a Fusion query pipeline. The following global pipeline variables are available:

Name	Type	Description
`request`	Request	The Solr query information.
`response`	Response	The Solr response information.
`params`	Context	A reference to the container which holds a map over the pipeline properties. Used to update or modify this information for downstream pipeline stages.
`ctx`	Context	A reference to the container which holds a map over the pipeline properties. Used to update or modify this information for downstream pipeline stages.
`_context`	Context	Another reference to the same object as `ctx`, included because some stages use this name instead.
`collection`	String	The name of the Fusion collection being indexed or queried.
`solrServer`	BufferingSolrServer	The Solr server instance that manages the pipeline’s default Fusion collection. All indexing and query requests are done by calls to methods on this object. See SolrClient for details.

Name	Type	Description
`solrServerFactory`	SolrClientFactory	The SolrCluster server used for lookups by collection name which returns a Solr server instance for a that collection, e.g. `var productsSolr = solrServerFactory.getSolrServer("products");`

Global variable `logger`

The global variable named `logger` writes messages to the logfile of the server running the pipeline. Since Fusion's api service does the query pipeline processing, these log messages go into the logfile: `fusion/var/log/api/api.log`. There are 5 methods available, which each take either a single argument (the string message to log) or two arguments (the string message and an exception to log). The five methods are, "debug", "info", "warn", and "error".

7.10.2. JavaScript Query Stage Examples

Add a parameter to the query request

```
request.addParam("foo", "bar");
```

Add a parameter to the query response

This example contains a simple JavaScript function which adds information to the query response. Repeated calls to this function build out the response.

```
function add_to_response(key, list) {
  if (list.length > 0) {
      response.initialEntity.appendStringList(key, Java.to(list, Java.type('java.util.List')));
  }
}
add_to_response('banners', ctx.getProperty('banners'));
add_to_response('landing-pages', ctx.getProperty('redirects'));
```

7.10.3. Debugging and Troubleshooting

To debug a JavaScript Index stage you can:

- Check the Fusion api server logs for errors.
- Use the `logger` object for print debugging (in the Fusion api logfile).
- Use the Pipeline Preview tool (only available in Fusion 1.x)

7.10.4. The JavaScript Engine Used by Fusion

The JavaScript engine used by Fusion is the Nashorn engine from Oracle. See The Nashorn Java API for details.

Upgrading to the latest Nashorn engine

The default version of the Nashorn engine used by Fusion versions 2.4.1 and earlier is the nashorn-0.1-jdk7.jar which contains many bugs that have since been fixed in the official JDK 1.8 version. In order to use the latest version of the Nashorn engine, you must:

- Have an up-to-date version of Java 8 installed.
- Remove the nashorn-0.1-jdk7.jar from the Fusion classpaths:
  - `cd fusion`
  - `find . -name "nashorn-0.1-jdk7.jar" -print -exec rm -i {} \;`

Creating and accessing Java types

The following information is taken from Oracle's JavaScript programming guide section 3, Using Java From Scripts.

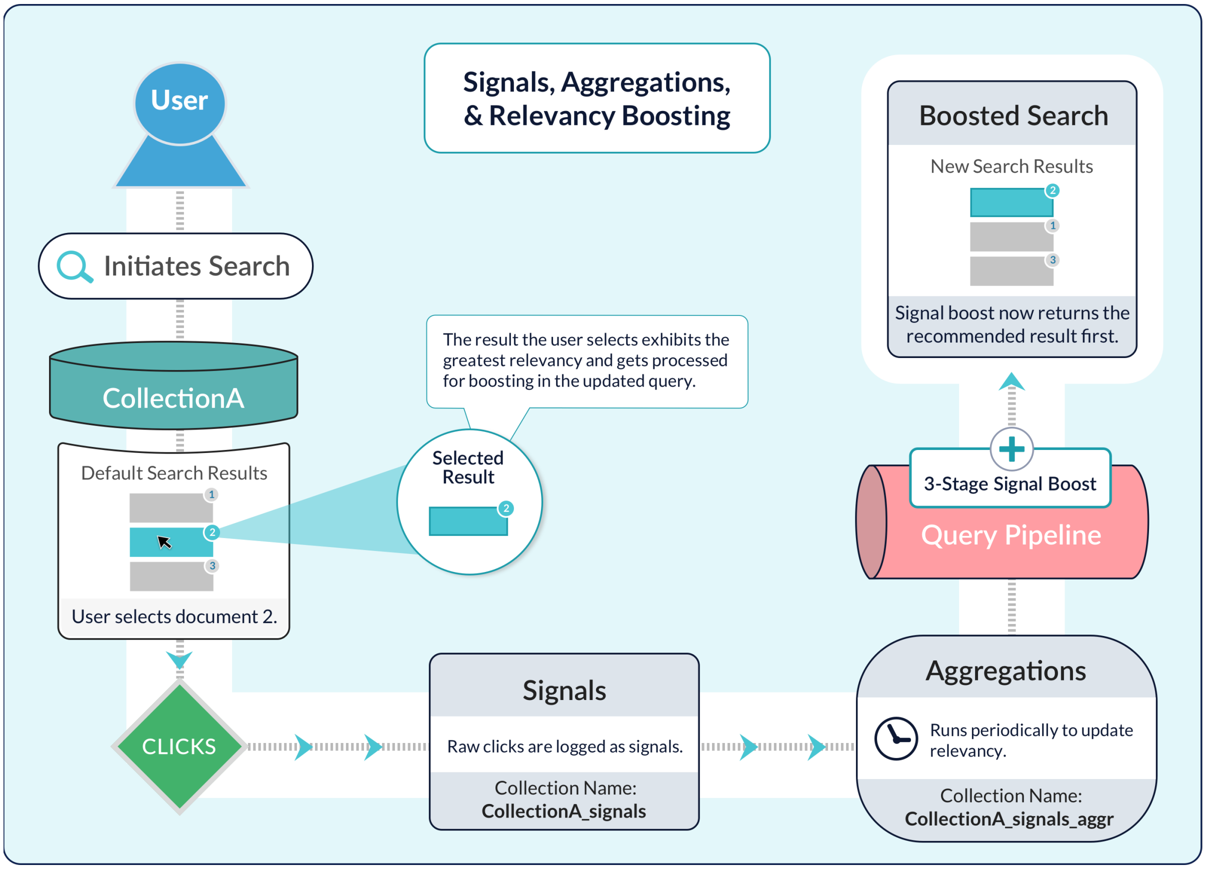
To create script objects that access and reference Java types from Javascript use the `Java.type()` function:

```
var ArrayList = Java.type("java.util.ArrayList");
var a = new ArrayList;
```

Chapter 8. Signals and Aggregations

In addition to the basic search experience enabled through query pipelines, Fusion provides ways to develop an enhanced search experience for your end users and provide useful data for your analytics team. The primary mechanisms for doing this are signals and aggregations.

By collecting signals and aggregating them, you compile a body of data that allows you to develop a sophisticated search experience, with rich search results for your end users, based on past user behavior.


Signals, Aggregations, & Relevancy Boosting
User
Initiates Search
CollectionA
Default Search Results
1
2
3
User selects document 2.
Selected Result
2
The result the user selects exhibits the greatest relevancy and gets processed for boosting in the updated query.
CLICKS
Signals
Raw clicks are logged as signals.
Collection Name: CollectionA_signals
Aggregations
Runs periodically to update relevancy.
Collection Name: CollectionA_signals_aggr
Query Pipeline
3-Stage Signal Boost
Boosted Search
New Search Results
2
1
3
Signal boost now returns the recommended result first.

Signals and aggregated signals are stored each in their own collection. These collections are associated with a primary collection, so that a collection named "products" will have two related collections: "products_signals" and "products_signals_aggr". By default, when using the UI to create a collection, a "signals" and "aggregated signals" collection are also created.

8.1. Signals

Signals are events that are collected for analysis or to enhance the search experience for end users. Common types of signal events include clicks, purchases, downloads, ratings, and so on.

Generally, signals are only useful after aggregation.

8.2. Aggregations

Aggregations are processed signals. An *aggregator* reads the raw signals and returns interesting summaries, ranging from simple sums to sophisticated statistical functions.

Crucially, it must be possible to relate the documents in an aggregated signals collection to documents in the primary collection, in order to use the aggregated signals for recommendations and/or boosting of searches over the primary collection.

8.3. The cold start problem

The "cold start" problem means it is hard to personalize the search experience when insufficient signals have been aggregated. For example, it is hard to offer recommendations to users who have never visited before, or for queries that have never been issued before, or for items that have been recently introduced into the system.

Fusion provides solutions for this problem using its query pipelines. A query pipeline that includes stages for blocking, boosting, or recommending based on signals can also include stages that provide fallbacks. In the case where there is not enough data to provide specialized blocking, boosting, or recommendations, the pipeline can return a simpler set of search results using Solr’s normal relevancy calculation.

A common solution to the cold start problem is to sort or boost on a certain field to provide pseudo-recommendations when more specific recommendations are not available. For example, you can sort on the `sales_rank` field to recommend the most popular products, or boost on the `date_added` field to recommend the newest items.

8.4. Signals

A *signal* is a recorded event related to one or more documents in a collection. Signals can record any kind of event that is interesting to your organization. Queries and clicks are the most common types of signals, as they are useful for tracking what users search for and what actions they take.

Signals are indexed in a secondary collection which is linked to the primary collection by the naming convention `<primarycollectionname>_signals`. So, if your main collection is named `products`, the associated signals collection is named `products_signals`. The signals collection is created automatically when signals are enabled for the primary collection.

Signals are indexed just like ordinary documents. The signals collection can be searched like any other collection, for example to retrieve a user's search history or last viewed items.

Signals are most useful when they are *aggregated* into a set of summaries that can be used to enrich the search experience through recommendations and boosting. Like the signals collection, a `<primarycollectionname>_signals_aggr` collection is created automatically when signals are enabled for a primary collection.

8.4.1. Enabling and disabling signals

Using the Fusion UI, when you create a collection, signals are enabled and a signals collection created by default.

Using the API, the `/collections/{collection}/features/{feature}` endpoint enables or disables signals for any collection:

Check whether signals are enabled for a collection

```
curl -u user:pass http://localhost:8765/api/v1/collections/<collection-name>/features/signals
```

Enable signals for a collection

```
curl -u user:pass -X PUT -H "Content-type: application/json" -d '{"enabled" : true}'
http://localhost:8765/api/v1/collections/<collection-name>/features/signals
```

Disable signals for a collection

```
curl -u user:pass -X PUT -H "Content-type: application/json" -d '{"enabled" : false}'
http://localhost:8765/api/v1/collections/<collection-name>/features/signals
```

8.4.2. Signal document structure

A raw signal is stored as a Solr document with the following fields, which are derived from the raw signal as follows:

Field	Description
`id` *Optional*	The signal ID. If no ID is supplied, one will be automatically generated.

Field	Description
`type` *Required*	The signal type that is being sent. This value is used during aggregation to filter events of the same type. Types can be mixed in aggregation jobs, if needed. The type can consist of any string you choose. For consistency, always send events of the same type with the same type value. During indexing, type values will be moved to a field named `type_s`.
`params` *Optional*	The params allow flexible definition of the fields you care about and will use later for signal aggregation: • `docId` – A unique document ID This is stored in the Solr raw signal document as field `doc_id_s`. • `userId` – A unique user ID This is stored in the Solr raw signal document as field `user_id_s`. • `query` – A query string; for example, a user's search This is copied to the Solr raw signal document as both fields `query_s` and `query_t`. Some cleanup occurs to convert the string to lowercase, decode URL encoding, and replace white space with single space characters. The original query is saved in field `query_orig_s`. • `filterQueries` – A list of strings, such as filters on the search query This is copied to the Solr raw signal document as both `filters_s` and `filters_orig_ss`. • `collection` – The primary collection name • `weight` – A float value representing the relative weight of this signal This is saved in the field `weight_d`. • `count` – A positive integer value representing the incremented count of signals This is saved in the field `count_i`.

Field	Description
timestamp	The timestamp of the signal event. • When using the Signals API, this property is optional; it defaults to the current server time. • When using the Signal Formatter index stage, one of the following fields must be present: timestamp, timestamp_tdt, timestamp_dt, or epoch.

Here is the JSON representation of one click signal, taken from an example dataset of synthetic clickstream data:

```
{ "params": {
      "docId": "2125233",
      "filterQueries": ["cat00000","abcat0100000", "abcat0101000", "abcat0101001"],
      "query": "Televisiones Panasonic  50 pulgadas" }
 "type":"click",
 "timestamp": "2011-09-01T23:44:52.533000Z",
}
```

8.4.3. The default signals index pipeline

When indexing signals, a default index pipeline named `_signals_ingest` will be used unless you specify a different index pipeline.

The `_signals_ingest` pipeline has three stages:

1. Format Signals stage
2. Field Mapping stage
3. Solr Indexer stage

If you prefer different options in the signals index pipeline, you can pass a query parameter when indexing signals that contains the name of your custom index pipeline.

If you create a custom pipeline, it must include a Field Mapping stage and a Solr Indexer stage (see Index Pipeline Stages for more details), which sends the documents to Solr. Additionally, the Solr Indexer stage must have the `enforce_schema` property set to "true".

8.4.4. Removing signals

The aggregator includes an option to delete signals after they have been processed. If, however, you have chosen not to remove signals during aggregation, you can also run a "delete" query in Solr to delete documents from the signals collection.

8.4.5. Video tutorial

This video tutorial explains how to boost searches using click signals and aggregations:

8.5. Aggregations

Signals are most useful when they are aggregated into a set of summaries that can be used to enrich the search experience through recommendations and boosting.

When signals are enabled for a "primary" collection, a `<primarycollectionname>_signals` collection and a `<primarycollectionname>_signals_aggr` collection are created automatically.

Note	Aggregations must be configured from the collection that contains the signals data (usually `<primarycollectionname>_signals`), not the primary (`<primarycollectionname>`) collection. You can find the `_signals` collection by navigating to **Applications** > **Collections** and expanding your original collection to display its system collections.

8.5.1. Aggregation Pipelines

Aggregated events are indexed, and use a default pipeline named "aggr_rollup". This pipeline contains one stage, a Solr Indexer stage to index the aggregated events.

You can create your own custom index pipeline to process aggregated events differently if you choose.

8.5.2. Aggregation Functions

The section Aggregator Functions documents the available set of aggregation functions.

Custom aggregation functions can be defined via a JavaScript stage. The options described in Aggregator Scripting provide more detail on the objects available for scripts.

8.5.3. Aggregation properties

The aggregation process is specified by an aggregation type consisting of the following list of properties:

Name	Description
`id`	Aggregation ID
`groupingFields`	List of signal field names
`signalTypes`	List of signal types
`aggregator`	Symbolic name of the aggregator implementation
`selectQuery`	Query string, default `:`
`sort`	Ordering of aggregated signals
`timeRange`	String specifying time range, e.g., `[* TO NOW]`
`outputPipeline`	Pipeline ID for processing aggregated events
`outputCollection`	Output collection name
`rollupPipeline`	Rollup pipeline ID

Name	Description
`rollupAggregator`	Name of the aggregator implementation used for rollups
`sourceRemove`	Boolean, default is false
`sourceCatchup`	Boolean, default is true
`outputRollup`	Boolean, default is true
`aggregates`	List of aggregation functions
`params`	Arbitrary parameters to be used by specific aggregator implementations

8.5.4. EventMinerAggregator

This aggregator type produces synthetic co-occurrence documents for predefined fields, based on co-occurring data in events in the same session.

This aggregator assumes that events have the following fields:

- `timestamp_tdt` - event timestamp
- `user_id_s` - user ID
- `query_s` - (optional) query that is related to this event
- `doc_id_s` - (optional) document ID that is related to this event

A session is then defined as a series of events created by a given `user_id_s` with timestamps falling within a session timeout limit (which is configurable via aggregator parameters).

Example document produced by this aggregator:

```
"id": "29d2c95e48154a4ebdc03158b0dd7875-25170",
  "entity_type_s":"doc_id_s",
  "entity_id_s": "2548405",
  "co_occurring_docIds_ss": [
    "2938114"
  ],
  "co_occurring_docIds_counts_is": [2],
  "in_queries_ss": [
    "tennis",
    "nicktoons kinect",
    "virtua tennis 4"
  ],
 "in_queries_counts_is":[3, 1 ,1],
  "in_session_ids_ss": [
    "f2de89f97e1638614795d3b03c50d5b1",
    "eae6242c6b58526d2a039d4bd95a85b6",
    "d87d264a528fd7ba162c20209bd3ca8a"
  ],
  "in_session_ids_counts_is":[1, 1, 1],
  "in_user_ids_ss": [
    "4c79f9ebcf7d50ba5d25a3fca0343929ff05c822",
    "2943dea5408b6f947bbd42aa28f9d8006bfb366a",
    "3fa43872d2275d5e6463ff2de1d95dca51d0003f"
  ],
  "in_user_ids_counts_is":[1, 1, 1]
```

This example illustrates the following:

- This is a synthetic co-occurrence document for an entity of type doc_id_s, with entity ID "2548405" (that is, a document with this ID). Other similar documents are produced for sessions, queries and users as the primary entity ID.
- The `in_queries_ss` field contains queries that led to this document (in this case, since events were click-throughs, these are the queries that resulted in clicks on a link to this document).
- The `related_docIds_ss` field shows documents that users also clicked within the same search session (thus implicitly indicating that they are related to this one).
- The `in_user_ids_ss` and `in_session_ids_ss` fields contains user IDs and session IDs respectively, where the click events for this document originated from.

This document can be viewed as a row in an N x N co-occurrence matrix, with the dimensions being (in this case): `doc_id_s`, `user_id_s`,`query_s`. Alternatively, it can be viewed as a vertex of a given type (e.g. `doc_id_s`) in a graph, with the fields representing edges to other vertices of different types.

The EventMinerAggregator accumulates this co-occurrence data from events in sessions, per each entity type, and then periodically outputs co-occurrence documents when its internal LRU cache becomes full. It also flushes all remaining entries from the cache at the end of a job.

This helps the aggregator to limit the total memory budget, while keeping a reasonably long context of co-occurring entities.

The size of this internal cache is adjustable via a configuration parameter. A side-effect of this approach is that there may be multiple partial co-occurrence documents produced if related entities occur in a longer context than the LRU cache is able to handle - consumers of the aggregated data should be prepared to roll-up these multiple partial documents.

8.5.5. Aggregation job configuration

The groupingFields should use just `user_id_s`, and optionally the "sort" parameter should be set to `timestamp_tdt asc` - this way the sessionization process will work most efficiently. On the other hand, sorting by timestamp requires more work on the Solr-side, so it may be omitted, with the possible side-effect that there will be additional partial documents created.

8.5.6. EventMinerAggregator configuration parameters

These parameters are passed in the "params" property of the aggregator configuration:

- `maxSessionTime`: Optional integer, default value: 300. This specifies the maximum time (in seconds) to use for the definition of a session i.e. a series of actions by the same user in a given time period. Normally you want to keep this value fairly small, as events that occur close together in time are more likely to be related than those further apart.
- `maxElementsPerField`: Optional integer, default value: 10. Configures the maximum number of elements to store in each field of the aggregated documents.
- `maxCacheSize`: Optional integer, default value: 25000. This controls the maximum size of the internal LRU ("Least recently used") cache. Depending on the volume of data being processed, smaller values will result in more partial documents being created, while larger values will lead to a higher memory usage during the aggregation run.

8.5.7. Example Configuration

```
{
    "id": "event-miner-aggregation",
    "groupingFields": [
        "user_id_s"
    ],
    "signalTypes": [ ],
    "selectQuery": "*:*",
    "sort": "timestamp_tdt asc",
    "timeRange": "[* TO NOW]",
    "aggregator": "em",
    "sourceRemove": false,
    "sourceCatchup": false,
    "outputRollup": false,
    "aggregates": [ ],
    "params": {
        "maxSessionTime": 600,
        "maxElementsPerField": 20
        "maxCacheSize": 10000
    }
}
```

Notes:

- The input documents will be grouped together based on their `user_id_s` values.
- The `selectQuery` is set to `:`, which means match all values in all fields when returning the initial set of input records for processing.
- The `timeRange` specifies all records with a timestamp up to NOW, where the latter will be set by the system to the

time the aggregation job starts.

- The value for aggregator is set to `em`, which is a short label for the "EventMinerAggregator".
- The definition of the settings in the params section is as defined earlier.

8.5.8. Aggregator Functions

Aggregator Functions provide many ways to customize signals aggregations. These functions execute a specified operation on data coming from source event fields and accumulate the new value in a target field of the aggregated result.

Functions are implemented in a aggregator job definition, as a list within the `aggregates` property. Each function definition includes the function type, source fields, target fields, and additional parameters as needed for the function type. Specifically, each function takes the following properties (unless otherwise noted); additional parameters are noted in the function descriptions below.

- type: the function type.
- sourceFields: the list of fields from source events. Data will be retrieved from these fields as inputs to the function.
- targetField: the name of the target field where the aggregated result will be stored.
- params: any additional parameters for the specific function type, as described below.

The "sourceFields" and "targetField" field names in function specifications can be optionally prefixed with "event:" or "result:". If there are no prefixes the sourceFields take values from the current event being aggregated, and the targetField takes (or updates) the value in the current partial aggregated result. With these prefixes values can be processed and e.g. the original event can be updated, or event fields can be considered taking into account the accumulated values in the result.

Examples:

Override default input field source:

```
"sourceField": "result:tweet_split_ss"
```

Override default target field source:

```
"targetField": "event:tweet_split_ss"
```

Arithmetic Functions

Arithmetic functions operate on all valid numeric values (including string fields that are parseable into double numbers) from source fields and compute a single result to the target field.

sum

A sum of numeric values, as a double number.

Example:

```
  {
    "type" : "sum",
    "sourceFields" : [ "count_i" ],
    "targetField" : "sum_count_d"
  }
```

sumOfLogs

A sum of natural logs of numeric value, as a double number.

Example:

```
 {
    "type": "sumOfLogs",
    "sourceFields": [ "script_d" ],
    "targetField": "script_sum_logs_d"
 }
```

sumOfSquares

A sum of squares of numeric value, as a double number.

Example:

```
{
  "type" : "sumOfSquares",
  "sourceFields" : [ "params.position_s" ],
  "targetField" : "sumOfSquares_position_d"
}
```

count

A count of source values, as a long number.

Example:

```
{
   "type": "count",
   "sourceFields": [ "id" ],
   "targetField": "count_d"
 }
```

geoMean

A geometric mean of numeric values, as a double number.

Example:

```
{
  "type" : "geoMean",
  "sourceFields" : [ "params.position_s" ],
  "targetField" : "geoMean_position_d"
}
```

mean

An arthimetic mean of numeric values, as a double number.

Example:

```
 {
   "type" : "mean",
   "sourceFields" : [ "params.position_s" ],
   "targetField" : "mean_position_d"
 }
```

min

The minimum numeric value.

Example:

```
 {
   "type" : "min",
   "sourceFields" : [ "params.position_s" ],
   "targetField" : "min_position_d"
 }
```

max

The maximum numeric value.

```
 {
   "type" : "max",
   "sourceFields" : [ "params.position_s" ],
   "targetField" : "max_position_d"
 }
```

decay_sum

A sum of time-based exponentially decayed numeric values. The difference between the aggregationTime and the event time will be decayed using an exponential function with a half-life equaling 30 days.

This function has some additional properties:

- halfLife: the number of seconds for the half-life decay function.
- timestampField: the name of the field that contains the source event's timestamp. By default, this is `timestamp_dt`.
- defaultWeight: the weight of an event if all values from source fields are missing. The default is 0.1f, and this is expressed as a float.

Example:

```
{
   "type" : "decay_sum",
   "sourceFields" : [ "weight_d" ],
   "targetField" : "decay_sum_weight_d",
   "params" : { }
 }
```

String Functions

String functions operate all values from source fields treated as strings.

cat

A concatenation of string values.

This function has some additional properties:

- separator: the character to use as a delimiter between values. The default is a single space.
- maxStringLength: the maximum length of the concatenated values (including separators). When this limit is exceeded, additional values are discarded. The default value is 10485760 characters (10 * 1024 * 1024).
- maxValueCount: the maximum number of values to concatenate. Any values collected after this limit are discarded. The default is 100.

Example:

```
{
  "type" : "cat",
  "sourceFields" : [ "user_id_s" ],
  "targetField" : "cat_user_id_txt",
  "params" : { }
}
```

ucat

A concatenation of unique string values.

This function has some additional properties:

- separator: the character to use as a delimiter between values. The default is a single space.
- maxStringLength: the maximum length of of the concatenated values (including separators). When this limit is exceeded, additional values are discarded. The default value is 10485760 characters (10 * 1024 * 1024).
- maxValueCount: the maximum number of values to concatenate. Any values collected after this limit are discarded. The default is 100.

Example:

```
{
  "type" : "ucat",
  "sourceFields" : [ "user_id_s" ],
  "targetField" : "ucat_user_id_txt",
  "params" : { }
}
```

split

A simple regex-based string splitting function.

The following function params are supported:

- regex - (string, required) a regular expression used for splitting.
- lower - (boolean, optional, false by default) after the regex has been applied the resulting parts are optionally lower-cased (using US locale).

Example:

```
 {
    "type" : "split",
    "sourceFields" : [ "query_s" ],
    "targetField" : "event:query_split",
    "params" : {
      "regex": "\\s+",
      "lower": true
    }
}
```

In the example above, the raw signal event field "query_s" is first split on whitespace and then lower-cased, and the result is put back into the raw signal event field "query_split".

replace

A simple regex-based string replace. The java.util.regex.Pattern syntax is supported for the regex matching and replacement.

The following function params are supported:

- regex - (string, required) a pattern to match.
- replace - (string, required) replacement.

Example:

```
{
    "type" : "replace",
    "sourceFields" : [ "query_split" ],
    "targetField" : "event:query_split_clean",
    "params" : {
      "regex": "\\P{Alpha}+",
      "replace": "_"
    }
}
```

In the example above, this function takes the "query_split" values and replaces all non-alphabetic characters with underscores, and stores the result in the event's field "query_split_clean". As an extended example, this function would follow after the example **split** function and would add the field "query_split_clean" to the raw signal event. The "query_split_clean" field could be aggregated via other aggregation functions.

Collection Functions

Collection functions simply collect values from the source fields and add them as multiple values to the target field.

discard

This function discards all values from source fields and the target field. This modifies the source event and any in-progress aggregation result. This creates side-effects for subsequent functions, so should be used with care.

Example:

```
{
  "type" : "discard",
  "sourceFields" : [ "user_id_s" ],
  "targetField" : "collect_user_id_ss",
  "params" : { }
}
```

collect

Collect values from source fields.

This function has one additional property, 'maxValueCount', which defines the number of fields to collect from source fields. Any fields collected after this limit are discarded. The default is 100.

Example:

```
{
  "type" : "collect",
  "sourceFields" : [ "user_id_s" ],
  "targetField" : "collect_user_id_ss",
  "params" : { }
}
```

ucollectCollect unique values from source fields.

This function has one additional property, 'maxValueCount', which defines the number of fields to collect from source fields. Any fields collected after this limit are discarded. The default is 100.

Example:

```
{
  "type" : "ucollect",
  "sourceFields" : [ "user_id_s" ],
  "targetField" : "unique_user_id_ss",
  "params" : { }
}
```

Statistical Functions

Statistical functions compute scalar and matrix statistics. When the function has multiple results, such as for matrix or vector results, the data is stored in multiple fields.

varianceThe square of standard deviation of numeric values, as a double number.

Example:

```
{
  "type" : "covariance",
  "sourceFields" : [ "params.position_s", "position_rnd_1", "position_rnd_2" ],
  "targetField" : "cov_position_d",
  "params" : { }
}
```

stddev

The standard deviation of numeric values, as a double number.

Example:

```
{
  "type" : "stddev",
  "sourceFields" : [ "params.position_s" ],
  "targetField" : "stddev_position_d",
  "params" : { }
}
```

cardinality

An estimate of the number of unique elements in the set of values from source fields (which are treated as strings). This uses the HyperLogLog implementation.

This function has one additional property, 'error', which defines the acceptable probability of error from real value, specifically the standard deviation from real results. Smaller values cause exponentially higher RAM consumption during processing. For example, the default, 0.1, uses ~8Kb of RAM, while tests have shown 0.0001 uses ~64Mb.

Example:

```
{
  "type" : "cardinality",
  "sourceFields" : [ "params.position_s" ],
  "targetField" : "cardinality_position_l",
  "params" : { }
}
```

skewness

The measure of asymmetry of the distribution around its mean. This function is performed on numeric values and is expressed as a double number.

Example:

```
{
  "type" : "skewness",
  "sourceFields" : [ "params.position_s" ],
  "targetField" : "skewness_position_d",
  "params" : { }
}
```

kurtosis

The adjusted Pearson’s kurtosis of numeric values, expressed as a double. This provides a comparison of the shape of the distribution to that of the normal distribution.

Example:

```
{
  "type" : "kurtosis",
  "sourceFields" : [ "params.position_s" ],
  "targetField" : "kurtosis_position_d",
  "params" : { }
}
```

quantiles

The quantiles of numeric values, stored as a double number in 0-N.targetField, or as a list of values in the target field (depending on the 'multivalued' property, described below). This implementation uses the T-Digest structure.

This function has the following additional properties:

- quantiles: the number of quantiles. The default is 10.
- multiValued: when true, all quantiles will be stored as multiple values in the target field. If false, then multiple values will be created in the format '0.targetField' to 'N.targetField'.

Example:

```
{
  "type" : "quantiles",
  "sourceFields" : [ "params.position_s" ],
  "targetField" : "quantiles_position_ss",
  "params" : {
    "multiValued" : true
  }
}
```

topk

An estimate of the top-K elements in the source fields and their frequency. The result is stored in three multi-valued fields, each with the same number of values. The three fields are:

- counts.targetField: integer counts (frequencies) of elements.
- values.targetField: elements.
- errors.targetField: estimation errors.

This function has one additional property, 'k', which is the number of elements to report. The default is 10.

Example:

```
{
  "type" : "topk",
  "sourceFields" : [ "params.position_s" ],
  "targetField" : "topk_position_ss",
  "params" : { }
}
```

covariance

A covariance matrix of numeric values from N > 1 source fields, with no smoothing. Missing or invalid values are treated as 0.0. A row of missing values is ignored. The resulting covariance matrix is stored in `N * (N - 1)` fields following the naming pattern 'sourceField1.sourceField2.targetField'.

If source fields contain multiple values, only the first value from each source field will be used.

This implementation runs in a constant and small memory budget.

Example:

```
{
  "type" : "covariance",
  "sourceFields" : [ "params.position_s", "position_rnd_1", "position_rnd_2" ],
  "targetField" : "cov_position_d",
  "params" : { }
}
```

correlation

A correlation matrix of numeric values from N > 1 source fields. This implementation is based on the covariance function. The resulting correlation matrix is stored in `N * (N - 1)` fields following the naming pattern 'sourceField1.sourceField2.targetField'.

Example:

```
{
  "type" : "correlation",
  "sourceFields" : [ "params.position_s", "position_rnd_1", "position_rnd_2" ],
  "targetField" : "corr_position_d",
  "params" : { }
}
```

histogram

An approximate histogram of values and their counts in source fields, using the T-Digest algorithm. Results are stored as corresponding multiple values in 'means.targetField' (for double values) and 'counts.targetField' (for integer values).

Example:

```
{
  "type" : "histogram",
  "sourceFields" : [ "params.position_s" ],
  "targetField" : "histogram_position_ss",
  "params" : { }
}
```

sigmoid

This function uses hyperbolic tangent (tanh) to limit the impact of source values according to an s-shaped curve. The following parameters control the shape of the curve:

- weight - controls the range of values. Default weight is 1.0, which means that the sigmoid function values will range between (-1, 1). E.g. weight = 2.0 means that values will range between (-2, 2).
- intercept - sets the constant shift of function values. Default is 0, which means that sigmoid(0) = 0 and sigmoid(Inf) → 1. E.g. intercept = 2.0 means that sigmoid(0) = 2.0 and sigmoid(Inf) = 3.0.
- slope - this parameter controls the slope of the function, i.e. how quickly it reaches saturation. Default value is 1.0. E.g. slope = 2 will cause the function to saturate quickly, slope = 0.1 will cause the function to saturate for larger values of source.
- final - boolean, default is true. This controls how the sigmoid is applied to the source value. First, all numeric values from source fields are summed. Then, if final = false the current sum is passed to the sigmoid function and added to the previous total. If final = true then the current sum is added to the total and the sigmoid function is applied only at the end of the aggregation.

Example:

```
{
  "type" : "sigmoid",
  "sourceFields" : [ "params.position_s" ],
  "targetField" : "sigmoid_position_ss",
  "params" : {
      "weight" : 2.0,
      "intercept" : 10.0,
      "slope" : 0.5,
      "final" : true
   }
}
```

Logical Functions

when

A logical function where processing will continue only if this function evaluates to true.

This function takes one additional property, 'expr', which is a JavaScript expression that must evaluate to a Boolean true/false. This property takes the same objects as the 'expr' function, described above. If this property is missing, the function will evaluate to true when any sourceField or targetField is present.

Example:

```
{
  "type" : "when",
  "sourceFields" : [ "params.position_s" ],
  "params" : {
    "expr" : "parseFloat(params_position_s) < 3"
  }
}
```

unless

A logical function where processing will continue only if this function evaluates to false.

This function takes one additional property, 'expr', which is a JavaScript expression that must evaluate to a Boolean true/false. This property takes the same objects as the 'expr' function, described above. If this property is missing, the function will evaluate to false when any sourceField or targetField is present.

Example:

```
{
  "type" : "unless",
  "sourceFields" : [ "params.position_s" ],
  "params" : {
    "expr" : "parseFloat(params_position_s) > 1"
  }
}
```

Scripting Functions

script

A scripted function. Scripts are evaluated as snippets, not as a function, and are expected to operate directly on the source event and the result. Their final values are discarded, since snippets in JavaScript are treated as expressions that evaluate to a specific value.

This function ignores the sourceFields and targetFields properties. Instead, the snippets are passed the following properties:

- startScript: the script defined is executed when the aggregation for the next unique tuple starts.
- aggregateScript: the script defined is executed for each source event.
- finishScript: the script is defined when all events for the current tuple have been processed and the result is about to be returned.

Example:

```
  {
    "type" : "script",
    "sourceFields" : [ ],
    "params" : {
      "aggregateScript" : "result.addField('script_event_id_ss', event.getFieldValue('id'));"
    }
  }, {
    "type" : "script",
    "sourceFields" : [ ],
    "params" : {
      "aggregateScript" : "event.addField('position_rnd_1', event.getFieldValue('params.position_s') + 1.0 -
Math.random());event.addField('position_rnd_2', event.getFieldValue('params.position_s') + 5.0 - Math.random()
* 10.0);"
    }
  }
```

expr

A script expression. The script is evaluated as a snippet, and its final value is assigned to the targetField.

This function has only one additional propery, 'expr', which contains the script expression.

Example:

```
  {
    "type" : "expr",
    "sourceFields" : [ "query_s", "filters_s" ],
    "targetField" : "expr_s",
    "params" : {
      "expr" : "v = ''; if (value != null) v = value + ' '; v + query_s + '_' + filters_s"
    }
  }, {
    "type" : "expr",
    "sourceFields" : [ "params.position_s" ],
    "targetField" : "expr_d",
    "params" : {
      "expr" : "v = 0; if (value != null) v = parseFloat(value); v + parseFloat(params_position_s)"
    }
  }
```

Special Functions

noop

A function that does nothing (is non-operational). This is a fallback function when invalid function parameters or execution errors are encountered.

Example:

```
   {
    "type" : "noop"
  }
```

8.5.9. Aggregator Scripting

The aggregation jobs that process signals can be customized with using JavaScript. There are several options for scripts, and each option will be executed at a different point of the aggregation process. The options available at each stage of the process will vary, as explained for each option below.

Scripts are run after the main logic of the class they are customizing. This allows overriding default behavior of the class if needed.

The scripts are defined in the aggregation job using the Signals Aggregator API, with the 'params' property. Here is an example of declaring a script in an aggregator definition, using the the specialFields script option:

```
{
  "id" : "r1",
  "signalTypes" : [ "click" ],
  "selectQuery" : "*:*",
  "timeRange" : "[* TO NOW]",
  "params" : {
    "specialFields" : "unless_pos_gt_1_ss,when_pos_lt_3_ss"
  }
}
```

Note:

In many cases, the scripts defined will be executed many times during the aggregation job (i.e., for every event). For this reason, it’s good practice to keep the scripts as simple as possible to avoid a negative impact on system performance. The initScript option includes a "_context" object that can be used for storing values that may require lengthy initialization or heavy computation.

initScript

A JavaScript defined with this option is executed wen the signal aggregator instance (i.e., the specific aggregator job) is initialized. The following objects are available to the script:

- logger: an SLF4J Logger object.
- aggregator: the aggregator instance.
- initArgs: the intiation arguments.
- _context: the current scripting context. This can be used for storing small objects between executions of other scripted methods.

startScript

A script defined with this option is executed when a new tuple is about to be aggregated. All of the objects available to initScript are available to startScript, plus:

- type: the aggregation type, which is a string. Currently only the 'click' type is supported.
- aggregationTime: the reference point from which the aggregation is calculated, which is expressed in epoch time, an integer.
- currentTuple: a map of field names and values for the current tuple being aggregated.

aggregateScript

A script defined with this option is executed when a new event is being processed for the current tuple. All of the objects available to initScript and startScript are available, plus:

- event: the current event for aggregation.This is a SolrDocument.
- result: the aggregated result so far. This will also contain the original tuple fields. This is a SolrDocument.

If this script is present, it overrides the default logic for processing events. This means that the script must completely process the events as desired; it’s not possible to build on existing rules. Note also that defining an aggregateScript will override any options defined as specialFields, described below.

It’s possible to emit more than one result of aggregation for any given group of source events. This may be invoked in scripts, like the following snippet:

```
doc = $.prepareResult();
$.emit(doc);
```

"$" is a reference to the current instance of aggregation function. The "prepareResult" method finishes calculations of some of the more complex functions (e.g. topK, percentiles, correlation, etc) and updates the result PipelineDocument (note: after this function is called the current "result" document is discarded, and a new PipelineDocument will be created to hold results of aggregating the following events, and the returned document can’t be used for incremental calculations). An example use for this functionality would be to extract the month part of the date from a set of events which are sorted by timestamps, in order to produce aggregated results for every month within the current tuple defined by groupingFields.

finishScript

A script defined with this option is executed when all of the events for the current tuple have been processed and it’s time to return the aggregated result. All of the objects available to initScript, startScript and aggregator script are available, plus:

- result: the final aggregated result. This is a SolrDocument.

specialFields

A script defined with this option uses a comma-separated, a whitespace-separated, or a JSON list of field names that are exempt from the default processing logic. These fields will **not** be processed in any way, which means they will not be included in the aggregated result.

If an aggregatorScript has been defined, it will be used instead of this option.

The default processing logic is as follows:

- skip any fields declared in specialFields;
- skip the event ID field (id);
- if the field value is a Number, then sum up all values as a Double;
- if field name ends with '_s' or '_dt', retain only the first value and discard all other values (these dynamic fields are single-value only);
- otherwise add all values as-is to the result.

halfLife

This option allows defining a time period, in milliseonds, for the half-life decay formula. This formula is used when determining boost values for clicked documents: documents that have not been clicked in a longer period of time will not receive as high of a boost as documents that have been clicked more recently.

The default value is equivalent to 30 days (i.e., 2,592,000,000 ms).

weightScript

A script defined with this option is used when weighting the current event. It must evaluate to a numeric value, but has the following additional objects available:

- event: the current event for aggregation. This is a SolrDocument.
- result: the aggregated result so far. This will also contain the original tuple fields. This is a SolrDocument.
- eventFlag: the flag that indicates if the event is the result of a previous aggregation ("aggr") or is a new event ("event").
- eventTime: the timestamp of the event.
- eventWeight: the initial event weight, expressed as a float.
- defaultWeight: the default weight (if the script fails or the eventWeight is not entered properly), expressed as a float.
- position: the click position. This is 0-based, or 0 if not available. The data is retrieved from the 'params_position_s' field of the event.

8.5.10. Example Aggregation with Sample Data

Located in the `fusion/examples/signals` directory of your installation are a set of files that can show you how signals aggregation can work. There are three files that work together:

- `signals.sh` - this is a script that will load signals and run an aggregation job from start to finish.
- `signals.json` - a sample data set of 20,000 signal events. These are 'click' events.
- `aggregation_definition.json` - a sample aggregation job definition that will demonstrate some of the capabilities of the aggregator.

You can simply start the `signals.sh` script and sit back and wait for it to finish. Below we'll step through what the script is doing.

Creating a Collection and Loading Signals

The script will create a collection in your system named 'lucidworks102'. This collection will have signals enabled, which will create two additional system collections, 'lucidworks102_signals' and 'lucidworks102_signals_aggr'.

The script will then load the sample signal events contained in the `signals.json` file. This is about 20,000 click events that reflect queries users performed and document IDs the users clicked on. These signal events will be loaded to the 'lucidworks102_signals' collection.

Defining the Aggregation Job

Once the signal events have been loaded, the script will create an aggregation job and begin processing the events. The aggregated events will be stored in the 'lucidworks102_signals_aggr'. Below we'll step through this sample job in detail and note some interesting configurations. The line numbers correspond to the lines in `aggregation_definition.json`. Note that many of the functions referred to here are described in full in the section Aggregator Functions.

First, we assign the ID of the aggregation job and define the signalTypes and the aggregator we will use.

```
{
  "id": "1",
  "signalTypes": [
    "click"
  ],
```

In this case, we are using 'click' signalTypes, but we are not limited to only those types. We could instead define any string, as long as it exists in the type field of the events stored in the signals collection. If necessary, we could mix types, as needed.

Note that we have not defined the fields that we will use to create tuples for aggregation. Since we have not defined these, the defaults will be used: 'doc_id_s', 'query_s', and ' filters_s'. We have also not defined an 'aggregator' property so by default 'click' aggregation will occur and 'weight_d' fields will be calculated and added to the aggregated records.

In the next step, we define an 'aggregates' property, which will contain all of the functions we'd like to perform with the events data. The functions will be performed in the order they are defined here.

```
  "aggregates": [
    {
      "type": "count",
      "sourceFields": [
        "id"
      ],
      "targetField": "count_d"
    },
```

This first step counts the number of 'id' fields in the events data, and puts that number in a field named 'count_d'.

Next we define a script function.

```
    {
      "type": "script",
      "params": {
        "aggregateScript": "event.addField('script_d',2.0);result.setField('script_d',2.0);"
      }
    },
```

The aggregateScript is called for every event record in the source collection of signals. This script defines that a field 'script_d' should be added to each source record, and should also be added to the result record (i.e., the record after aggregation).

Next we define a sumOfLogs function.

```
    {
      "type": "sumOfLogs",
      "sourceFields": [
        "script_d"
      ],
      "targetField": "script_sum_logs_d"
    },
```

This definition uses the 'script_d' field that the previous step just created in each record.

Next we define a collect function.

```
    {
      "type": "collect",
      "sourceFields": [
        "id"
      ],
      "targetField": "ids_ss"
    },
```

This function will collect all 'id' fields from the source event, and put the ids in a field named 'id_ss'.

Finally, we define an expression function.

```
    {
      "type": "expr",
      "sourceFields": [
        "query_s",
        "filters_s"
      ],
      "targetField": "expr_t",
      "params": {
        "expr": "v = ''; if (value != null) v = value + ' | '; v + query_s + ' & ' + filters_s"
      }
    }
  ],
```

This expression concatenates values from the 'query_s' and 'filters_s' fields and appends them to the previous value of the target field (in this case, 'expr_t'). We have included a conditional block at the beginning of the expression because when this expression is first invoked the result ("value") may be 'null' and JavaScript would convert it to a literal "null" string.

This is the end of the aggregates section of the configuration, so we've closed the JSON tags appropriately.

The last property for this job is to set the timeRange.

```
  "timeRange": "[* TO NOW]"
}
```

This simple range will get all events for all dates and times.

Loading the Configuration and Starting the Job

If you are using the `signals.sh` script, you don't need to load the configuration or start the aggregator job. However, we've included the API calls below for your reference.

It's possible to load the configuration as a file via the Signals Aggregator API, with a call such as:

```
curl -u user:pass -X POST -H 'Content-Type: application/json'
http://localhost:8764/api/apollo/aggregator/aggregations -d @aggregation_definition.json
```

If this was successful, the response will be empty. While the file name is `aggregation_definition.json`, the ID of the job as defined in the file is '1', so that is the ID this job will be known as.

Once the job definition has been loaded, you can start the job when ready:

```
curl -u user:pass -X POST http://localhost:8764/api/apollo/aggregator/jobs/lucidworks102_signals/1
```

The job will likely run for a while, so you can check it periodically by doing a GET request to the same endpoint:

```
curl -u user:pass http://localhost:8764/api/apollo/aggregator/jobs/lucidworks102_signals/Aggr1
```

Once the job is finished, you are ready to work with your aggregated signals.

Sample Aggregated Event

Once the aggregation job has completed, we can take a look at a specific aggregated event.

```
{
    "id": "0886f6d2e02c47d5b11fc809dd8508bc-358",
    "attr_params.filterQueries_": [
        "cat00000",
        "abcat0200000",
        "abcat0204000",
        "pcmcat144700050004"
    ],
    "filters_s": "abcat0200000 $ abcat0204000 $ cat00000 $ pcmcat144700050004",
    "count_d": 5,
    "doc_id_s": "1118988",
    "script_d": 4,
    "query_t": "shure",
    "aggr_type_s": "click@doc_id_s-query_s-filters_s",
    "params.position_s": "0",
    "query_s": "shure",
    "filters_orig_ss": [
        "abcat0204000",
        "pcmcat144700050004",
        "abcat0200000",
        "cat00000"
    ],
    "weight_d": 1.0778248953069447e-11,
    "flag_s": "aggr",
    "expr_t": "shure & abcat0200000 $ abcat0204000 $ cat00000 $ pcmcat144700050004 | shure & abcat0200000 $ abcat0204000 $ cat00000 $ pcmcat144700050004 | shure & abcat0200000 $ abcat0204000 $ cat00000 $ pcmcat144700050004 | shure & abcat0200000 $ abcat0204000 $ cat00000 $ pcmcat144700050004 | shure & abcat0200000 $ abcat0204000 $ cat00000 $ pcmcat144700050004",
    "ids_ss": [
        "12a3a546-95b2-4a86-8d36-88aebc89eb5f",
        "4f074782-74b5-4a05-9bdf-1f447a60735d",
        "6006eb92-6bcb-417e-828a-e0408590c76e",
        "be02f44d-4731-44f3-94ac-faede6e7d4e3",
        "f80f886c-3825-4d90-81ce-89f53a987d92"
    ],
    "script_sum_logs_d": 3.4657359027997265,
    "aggr_id_s": "0886f6d2e02c47d5b11fc809dd8508bc",
    "type_s": "click",
    "query_orig_s": "shure",
    "attr_query_orig_s_": [
        "shure",
        "Shure"
    ],
    "count_i": 5,
    "timestamp_dt": "2014-09-12T22:05:22.818Z",
    "_version_": 1479078865938677800
}
```

Let’s walk through the fields of this aggregated document:

- First, the id consists of the job id, a dash, then a sequential number of the aggregated document.
- The fields that made up our tuple, in this case filters_s, doc_id, and query_s since we used the default, are shown

verbatim from the input signals.

- The aggr_type_s field consists of the aggregator used ('click') and the tuple dimensions ('doc_id_s-query_s-filters_s').
- The weight_d field has been added to each aggregated record because we performed a 'click' aggregation. If we had chosen a 'simple' aggregation instead, we would not have a 'weight_d' field but in other respects the aggregated record will be very similar.
- The count_d is 5, which shows how many source events comprised this aggregated record.
- The flag_s field is set to 'aggr' to indicate that this is an aggregated event.
- The script_sum_logs_d is the calculated sumOfLogs result, which used the values from the 'script_d' field in the original records added by the 'script' function.
- The id_ss is the result of the 'collect' function, in which we asked for all the event ids.
- The expr_t field is the output of the 'expr' function including the concatenated values from the 'query_s' and 'filters_s' fields.
- The aggr_id is the unique ID of the aggregation job. Note that this ID is one component of this aggregated event ID.
- The type_s is the type of the source event. If more than one type was chosen for the aggregation job, each type would appear separated by a pipe ('|') delimiter (such as, 'view|click|buy').
- The count_i field is the count of source events for this aggregated result.
- The timestamp_dt field is a reference time for the aggregation. This time is either when the aggregation was executed, or was supplied as part of the aggregation definition with the 'time' property.

Once the records have been aggregated, they can be used for Recommendations, or for other uses within your organization.

8.6. DateTime Processing

8.6.1. DateUtils - Uniform API for Date Parsing and Formatting

Date and time processing is difficult due to the complexity of rules (and exceptions from rules) specific to historical changes, calendar systems, locales, time zones, and DST rules (daylight saving time).

Fusion uses com.lucidworks.apollo.common.util.DateUtils for date / time parsing and formatting, and developers are strongly encouraged to use this class for parsing date formats.

The DateUtils class uses the Joda Time library, which was the basis for the java.util.time API in JDK 8 but is compatible with earlier versions of Java. It add support for parsing abbreviated time zone names (e.g. PST). Although use of abbreviated time zone names has been deprecated because many of them are ambiguous, they are still in wide use. It also supports parsing full time zone names in any letter case (Joda Time accepts only canonical mixed-case names, e.g. "America/New_York"). Robust identification of time zone is helpful for reasoning about time intervals, because time zone rules cover phenomena like DST changes with gap and overlap hours, leap seconds, administrative changes to offsets, etc, etc, for which a simple offset from UTC is insufficient.

8.6.2. Supported Formats

Supported formats for date / time parsing can be divided into three disjoint groups:

- ISO 8601 formats ("Solr formats"), represented as yyyy-MM-dd’T’HH:mm:ss.SSS’Z'. The milliseconds part is optional, with precision varying between 0-3 digits.
- Fusion "zoned format" - a modification of a common Internet format that specifies day of week and uses full names for time zones, represented as EEE yyyy-MM-dd HH:mm:ss.SSS ZZZ in US locale. This format is useful for storing and easy processing of date/time with zone, as it’s unambiguous, exact and easy to parse.
- Epoch formats - represented in Java API as objects of java.util.Date or the number of seconds, or the number of milliseconds since epoch (either as a number or as a String). These formats by definition don’t contain any time zone information, and if permitted they are treated as absolute instants in the default time zone (see below).
- Common global formats - i.e. formats that explicitly specify the timezone.
- Common local formats - i.e. formats that don’t specify the timezone.

The definition of formatting symbols used below can be found in the Joda Time DateTimeFormat documentation.

ISO 8601 / Solr Formats

See https://cwiki.apache.org/confluence/display/solr/Working+with+Dates

```
      "yyyy-MM-dd'T'HH:mm:ss'Z'", // Solr format without milliseconds
      "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'", // standard Solr format, with literal "Z" at the end
      "yyyy-MM-dd'T'HH:mm:ss.SS'Z'", // standard Solr format, with literal "Z" at the end
      "yyyy-MM-dd'T'HH:mm:ss.S'Z'" // standard Solr format, with literal "Z" at the end
```

Common Global Formats

```
      "EEE yyyy-MM-dd HH:mm:ss.SSS zzz",
      "yyyy-MM-dd'T'HH:mm:ss.SSSZ", // with numeric +-HHmm timezone at the end
      "yyyy-MM-dd'T'HH:mm:ss.SSSZZ", // with numeric +-HH:mm timezone at the end
      "yyyy-MM-dd'T'HH:mm:ss.SSSz", // with symbolic XXX timezone at the end
      "yyyy-MM-dd'T'HH:mm:ssz", // with symbolic XXX timezone at the end
      "yyyy-MM-dd'T'HH:mm:ssZ", // with offset
      "EEE MMM d HH:mm:ss z yyyy",
      "EEE MMM d HH:mm:ss Z yyyy",
      "EEE MMM d HH:mm:ss z yyyy",
      "EEE MMM d HH:mm:ss.SSS z yyyy",
      "EEE, dd MMM yyyy HH:mm:ss zzz", // RFC 1123, with either short or full time zone
      "EEEE, dd-MMM-yy HH:mm:ss zzz", // RFC 1036
      "yyyy-MM-dd HH:mm:ss Z",
      "yyyy-MM-dd HH:mm:ss ZZ",
      "yyyy-MM-dd HH:mm:ss z",
      "yyyy-MM-dd HH:mm:ss.SSS Z",
      "yyyy-MM-dd HH:mm:ss.SSS ZZ",
      "yyyy-MM-dd HH:mm:ss.SSS z",
      "yyyy-MM-dd HH:mm:ss zzz", // with full time zone (e.g. America/New_York)
      "yyyy-MM-dd'T'HH:mm:ss'GMT'Z", // with literal "GMT" and offset
      "yyyy-MM-dd'T'HH:mm:ss.SSS'GMT'Z", // with literal "GMT" and offset
      "yyyy-MM-dd'T'HH:mm:ss'UTC'Z", // with literal "UTC" and offset
      "yyyy-MM-dd'T'HH:mm:ss.SSS'UTC'Z", // with literal "UTC" and offset
      "yyyy-MM-dd HH:mm:ss 'UTC'Z",
      "yyyy-MM-dd HH:mm:ss.SSS 'UTC'Z",
      "yyyy-MM-dd HH:mm:ss 'GMT'Z",
      "yyyy-MM-dd HH:mm:ss.SSS 'GMT'Z"
```

Note: Joda-Time cannot parse ZZZ when zone name is in incorrect case (e.g. "EUROPE/WARSAW" will fail), for this reason we use zzz which accepts any letter case.

Common Local Formats

```
      "yyyy-MM-dd'T'HH:mm:ss",
      "yyyy-MM-dd",
      "yyyy-MM-dd hh:mm:ss",
      "yyyy-MM-dd HH:mm:ss",
      "yyyy-MM-dd HH:mm:ss.SSS",
      "EEE MMM d HH:mm:ss yyyy", // ANSI C
      "EEE MMM d HH:mm:ss.SSS yyyy" // ANSI C
```

```
Missing parts of the data are filled in with defaults - e.g. time zone is filled in with the default time zone
(see below), missing time data is filled with 00:00:00.000
```

8.6.3. DateUtils Usage

Instances of DateUtils can be created with the following arguments:

- requireTimezone - (boolean, required) when this argument is true then only patterns with timezone component are accepted (unless custom patterns are provided). Epoch formats are not accepted. If false then any recognizable pattern is used.
- formats - (list of strings, optional) a list of formats to use instead of the built-in ones.

- locale - (string, optional) locale to use for parsing, or null for the platform default locale.
- defaultTimeZone - (string, optional) time zone name to use as default (when using local formats), or null for the platform default time zone. Time zone names can be provided in short or long form, or as a fixed offset [-+]HH:mm.Further information is provided in the Javadoc of the class. The default instance of DateUtils uses common global formats (with ISO8601 and epoch formats), requireTimezone == true, locale "en-US" and time zone "UTC".

Chapter 9. Dashboards

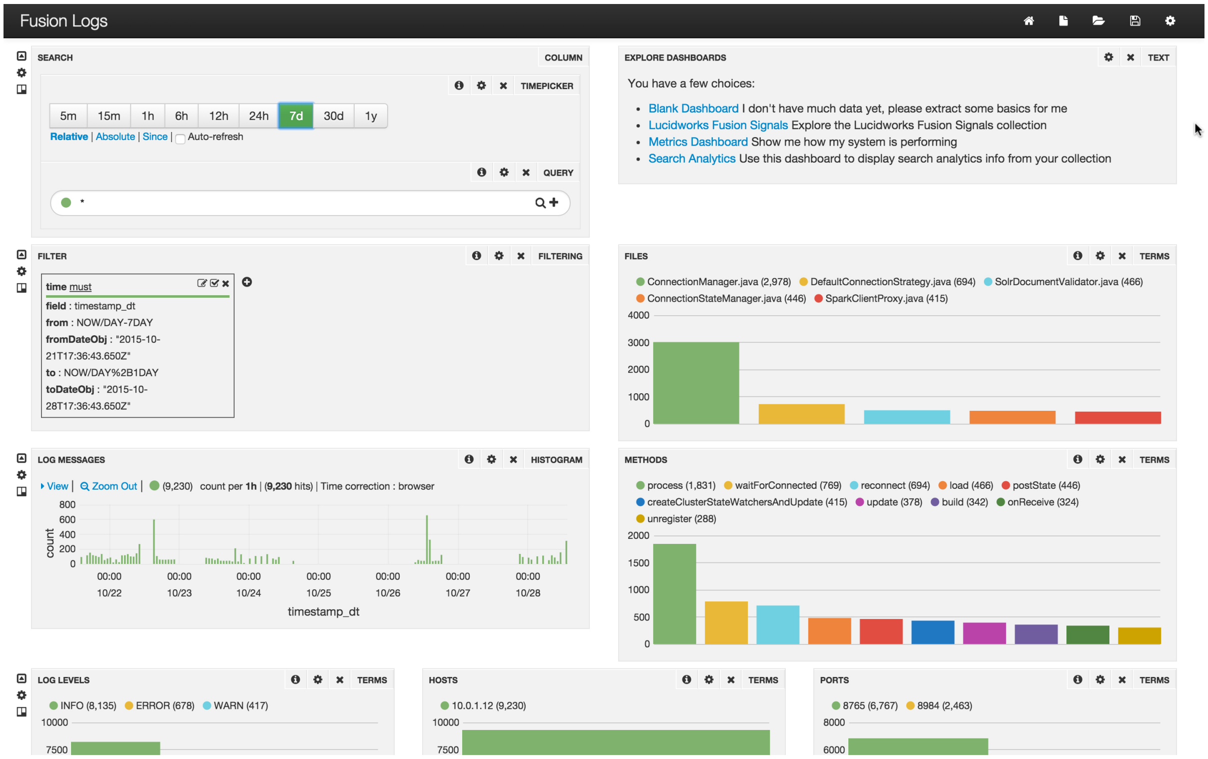
Dashboards are Fusion’s built-in analytics component.

A Fusion dashboard contains one or more controls for search query inputs and one or more quantitive displays over the results for that query. Dashboards run as a client-side application in a web browser, using JavaScript components for HTML5. Solr facets provide the quantifications required for visualizations, which can be charts, graphs, tables, and maps (for geospatial data). Dashboards also have tabular displays for drilling down to the individual documents in a results set.

Client-side JavaScript provides a dynamic, responsive browsing experience. The underlying browser application is "Banana", an open-source visualization tool based on Kibana 3. Where Kibana communicates with Elasticsearch, Banana communicates with Solr. Banana can be downloaded from: https://github.com/LucidWorks/banana/.

Fusion Dashboards were first developed for logfile analysis. Logfile entries contain a timestamp plus some amount of additional event information. Fusion’s date / time processing aggregates these timestamps at different levels of granularity allowing for visualization of historical trends in system and user behaviors.

The default Fusion dashboard is a Log Analytics dashboard over the Fusion logfiles:



Fusion Logs

EXPLORE DASHBOARDS

You have a few choices:

- Blank Dashboard I don't have much data yet, please extract some basics for me
- Lucidworks Fusion Signals Explore the Lucidworks Fusion Signals collection
- Metrics Dashboard Show me how my system is performing
- Search Analytics Use this dashboard to display search analytics info from your collection

FILTER

time must

field : timestamp_dt

from : NOW/DAY-7DAY

fromDateObj : "2015-10-21T17:36:43.650Z"

to : NOW/DAY%2B1DAY

toDateObj : "2015-10-28T17:36:43.650Z"

LOG MESSAGES

▸ View | Zoom Out | ● (9,230) count per 1h | (9,230 hits) | Time correction : browser

FILES

● ConnectionManager.java (2,978) ● DefaultConnectionStrategy.java (694) ● SolrDocumentValidator.java (466) ● ConnectionStateManager.java (446) ● SparkClientProxy.java (415)

METHODS

● process (1,831) ● waitForConnected (769) ● reconnect (694) ● load (466) ● postState (446) ● createClusterStateWatchersAndUpdate (415) ● update (378) ● build (342) ● onReceive (324) ● unregister (288)

LOG LEVELS

● INFO (8,135) ● ERROR (678) ● WARN (417)

HOSTS

● 10.0.1.12 (9,230)

PORTS

● 8765 (6,767) ● 8984 (2,463)

Fusion Dashboards are equally useful for signal analysis. Signals dashboards are similar to the Log Analytics dashboard, because Fusion signals must always contain a timestamp field.

The Getting Started tutorial is a step-by-step guide to setting up a new dashboard for time-series data.

Fusion Dashboards are general enough so that Fusion collections over documents that don’t contain timestamp information can still be analyzed as a non time-series dashboard.

9.1. Dashboard Access and Authorizations

If you are logged into Fusion and have a valid Fusion session cookie, you can launch the Dashboards application directly from a browser via the URL:

```
http://host:*port*/banana/index.html#/dashboard
```

Configuration settings control which dashboard is displayed when the application is opened. For the initial Fusion distribution, this is set to the Fusion Logs dashboard, a pre-configured dashboard over Fusion logs for the system collection.

At a minimum, dashboard access requires the following Fusion user permissions:

- Read access to the collection over which the analytics are being run
- Read access to the system collection `system_banana`

A user who has only these minimal permissions can view and interact with the analytics dashboard, but will not be able to use any other part of the Fusion UI.

9.2. Dashboard Components

A Dashboard is a named layout which consists of:

- input panels which compose and submit the Solr query
- display panels which provide information and visualizations of the results set.

Each time the input controls are updated, a new query is sent to Fusion which then repopulates the dashboard with the results set, causing all displays to update automatically.

9.3. Pre-configured Dashboards

Fusion includes a set of pre-configured dashboards:

- Fusion Logs - Dashboard over Fusion system collection "logs".
- Fusion Metrics - dashboard over Fusion system collection "system_metrics".
- Lucidworks Signals Collection - dashboard for the example signals dataset included with Fusion distribution.
- Search Analytics - configured for any Fusion primary collection which has a secondary "_logs" collection.

9.4. Getting Started

See the Getting Started tutorial for a step-by-step guide to setting up a new dashboard for time-series data.

9.5. Get Started with Fusion Dashboards

In this tutorial we build an analytics dashboard over signal data.

The Fusion distribution includes a directory named "fusion/examples/signals", where "fusion" is the top-level directory of the Fusion distribution, which contains:

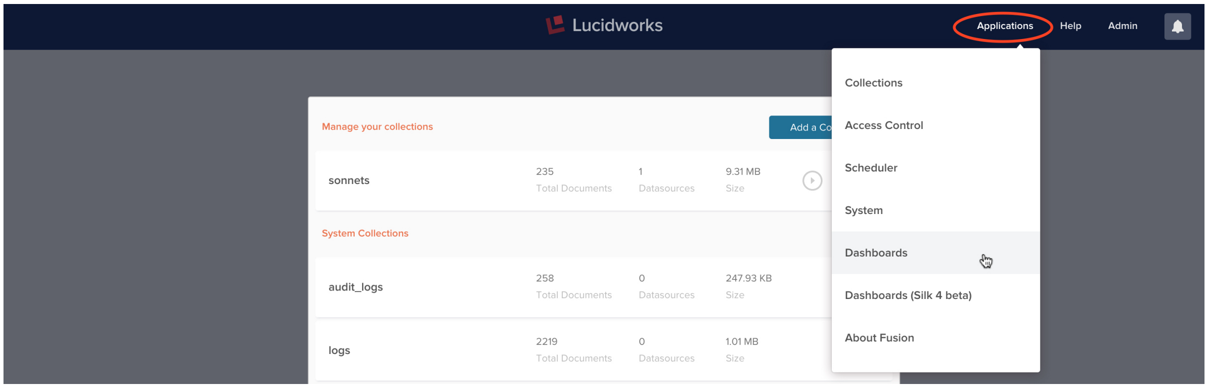
- The data file `signals.json`, a synthetic dataset of 20,000 click-signal events, based on a set of Best Buy query logs from 2011. This file contains a list of JSON objects, where each object contains information about a search query, a set of search categories, and the item ultimately clicked on.
- The script file `signals.sh`, which loads the raw signal data and then runs aggregations.

First we load the raw signal data into a Fusion collection. Then we create a new time-series dashboard and populate it with input and display panels.

9.5.1. Dashboard Controls

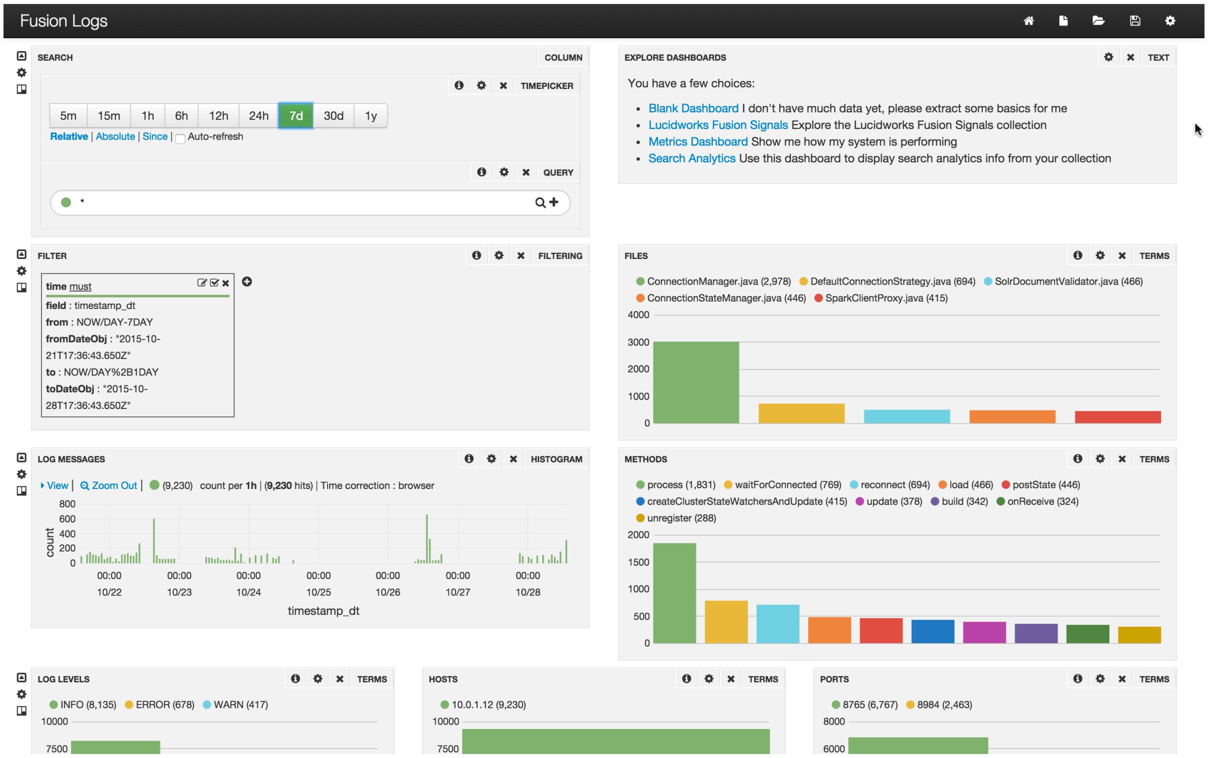
Here’s how to interact with the Dashboard interface.

You can launch the Dashboards tool from the Fusion UI, via the Fusion Applications menu:



sonnets	235 Total Documents	1 Datasources	9.31 MB Size
audit_logs	258 Total Documents	0 Datasources	247.93 KB Size
logs	2219 Total Documents	0 Datasources	1.01 MB Size

When launched from the Fusion UI, the Dashboards tool appears in a new tab titled Banana 3. The title changes to the name of the dashboard. The initial dashboard is the built-in Fusion Logs dashboard:



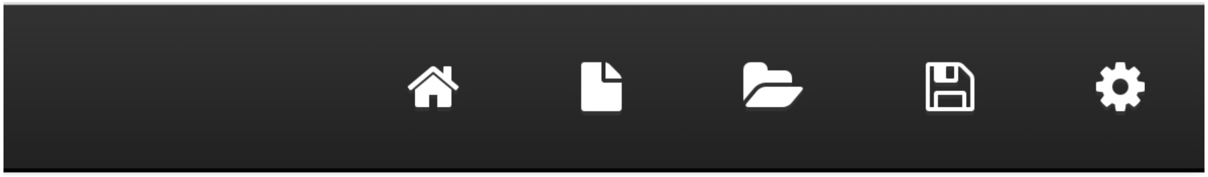
Fusion Logs

You have a few choices:

- Blank Dashboard I don't have much data yet, please extract some basics for me
- Lucidworks Fusion Signals Explore the Lucidworks Fusion Signals collection
- Metrics Dashboard Show me how my system is performing
- Search Analytics Use this dashboard to display search analytics info from your collection

time must
field : timestamp_dt
from : NOW/DAY-7DAY
fromDateObj : "2015-10-21T17:36:43.650Z"
to : NOW/DAY%2B1DAY
toDateObj : "2015-10-28T17:36:43.650Z"

The left side of the top menu bar displays the Dashboard title. The right side of the top menu contains a set of controls:



- The home icon is the "Go to Default Dashboard" control. The initial default dashboard is the "Fusion Logs" dashboard.
- The sheet of paper icon opens the "Create Dashboard" dialog.
- The folder icon opens the "Load Dashboard" dialog.
- The diskette icon opens the "Save Dashboard" dialog.
- The gear icon opens the "Configure Dashboard" dialog for the current dashboard, which has tabs:
  - "General" : title and page style
  - "Rows" : for adding, deleting and arranging the rows
  - "Controls" : loading, save, and export options
  - "Solr" : configure Solr server and set global query parameters.

9.5.2. Create the Raw Signals Collection

In this example, we create a Log Analytics Dashboard using a synthetic log dataset. This file contains a list of JSON objects, where each object contains information about a search query, a set of search categories, and the item ultimately clicked on.

The first JSON object in this dataset is:

```
    "timestamp": "2011-09-01T23:44:52.533Z",
    "params": {
      "query": "Televisiones Panasonic  50 pulgadas",
      "docId": "2125233",
      "filterQueries": [
        "cat00000",
        "abcat0100000",
        "abcat0101000",
        "abcat0101001"
      ]
    },
    "type": "click"
```

The top-level attributes of this object are:

- **type** - A required field for all signals. In the example dataset, all signals are of type click.
- **timestamp** - The time at which this click was logged.
- **params** - This attribute contains a set of key-value pairs for search-query event information.

In this dataset, the information captured includes the free-text search query entered by the user, the document id of the item clicked on, and the set of Best Buy site categories that the search was restricted to. These are codes for categories and sub-categories such as "Electronics" or "Televisions".

The example script `signals.sh` loads the raw signal via a POST request to the Fusion REST API endpoint: `/api/apollo/signals/<collectionName>` where <*collectionName*> is the name of the primary collection itself.

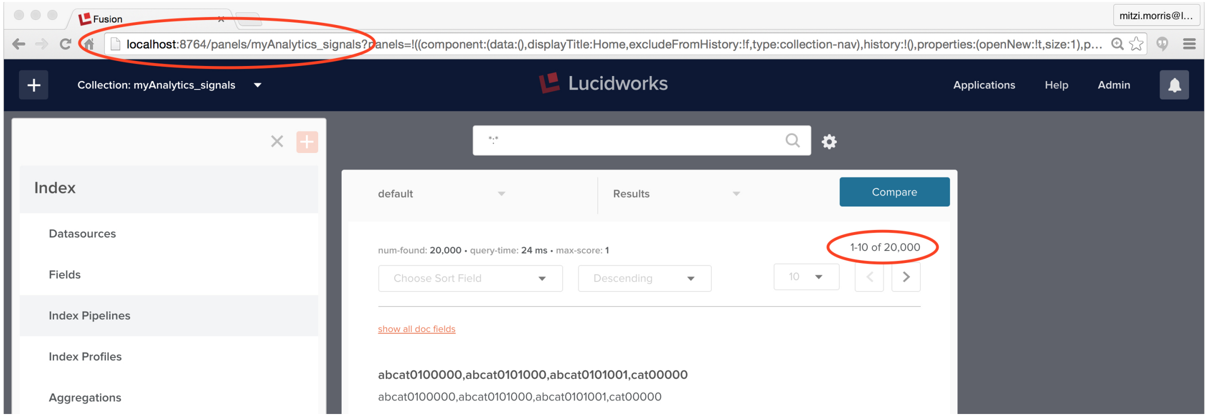
I create the primary collection "myAnalytics" from the Fusion UI. By default, this collection has feature "signals" enabled, therefore I can load the raw signal data into Fusion collection "myAnalytics_signals" by sending a POST request to the endpoint: `/api/apollo/signals/myAnalytics`

To run this command on a local Fusion installation using all default ports, the request is:

```
curl -u <user>:<password> -X POST\
 -H 'Content-type:application/json'\
 http://localhost:8764/api/apollo/signals/myAnalytics?commit=true\
 --data-binary @<some-path>/signals.json
```

This command succeeds silently.

To verify that the collection "myAnalytics_signals" exists, view the collection details in the Fusion UI via URL: "localhost:8764/panels/myAnalytics_signals".



localhost:8764/panels/myAnalytics_signals?panels=!((component:(data:(),displayTitle:Home,excludeFromHistory:!f,type:collection-nav),history:!(),properties:(openNew:!t,size:1),p...

:

num-found: 20,000 • query-time: 24 ms • max-score: 1

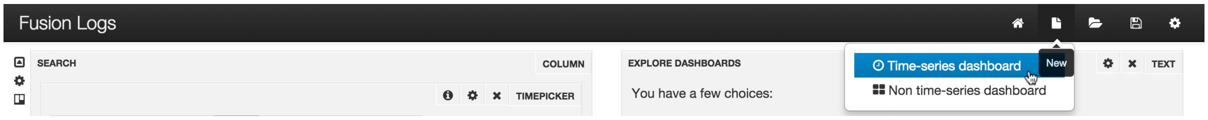
1-10 of 20,000

abcat0100000,abcat0101000,abcat0101001,cat00000

abcat0100000,abcat0101000,abcat0101001,cat00000

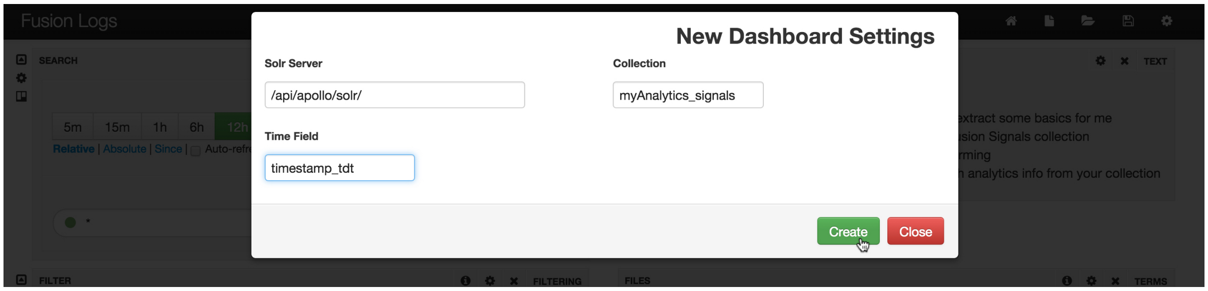
9.5.3. Create the Dashboard

Time-series dashboards show trends over time by using the timestamp field to aggregate query results. To create a time-series dashboard over the collection "myAnalytics_signals", from the Dashboards application, use the "New" control to open the "Create Dashboard" dialog and choose the option "new time-series dashboard"



You have a few choices:

The next step is configuring the dashboard settings:



New Dashboard Settings

Solr Server: /api/apollo/solr/

Collection: myAnalytics_signals

Time Field: timestamp_tdt

The newly created dashboard is called "New Time Series Dashboard". It contains 4 rows: * Row 1 contains the input controls and a display panel showing total hits. * Row 2 displays the time range specified in the search query * Row 3 contains a visual display of hits per date-time interval * Row 4 contains a table over all documents in the result set.

In the example above, the dashboard is populated with results for the following Solr query parameters:

```
q=*:*&
fq=timestamp_tdt:[NOW/MINUTE-15MINUTE TO NOW/MINUTE+1MINUTE]&
wt=json&rows=0
```

The example signals data was collected during the months of August through October 2011. After the "Time Range" panel is configured with this date range, the new Solr query parameters are:

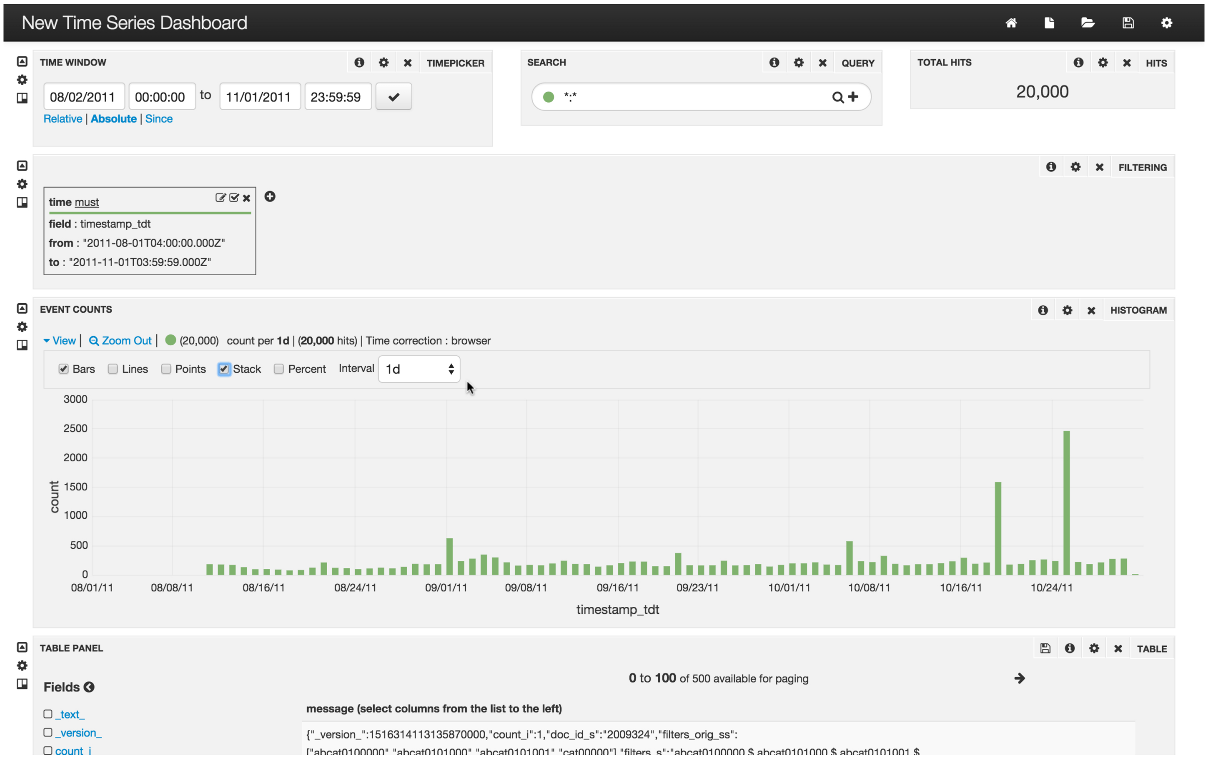
```
q=*:*&
fq=timestamp_tdt:[2011-08-01T04:00:00.000Z TO 2011-11-01T03:59:59.000Z]&
wt=json&rows=0
```

This query returns all 20,000 raw signals and the panel is updated accordingly.

Configure the Histogram Panel

Solr facets provide the counts displayed by the Histogram. The facet parameters are derived from the TimePicker selection and the controls on the Histogram display panel. These controls are used to select the type of count display and the time interval over which counts are aggregated.

After this interval is set to 1-day, the dashboard shows the daily activity for the timespan:



New Time Series Dashboard

TIME WINDOW — TIMEPICKER: 08/02/2011 00:00:00 to 11/01/2011 23:59:59 — Relative | Absolute | Since

SEARCH — QUERY: ● *:*

TOTAL HITS — HITS: 20,000

FILTERING:
time must
field : timestamp_tdt
from : "2011-08-01T04:00:00.000Z"
to : "2011-11-01T03:59:59.000Z"

EVENT COUNTS — HISTOGRAM

View | Zoom Out | ● (20,000) count per 1d | (20,000 hits) | Time correction : browser

☑ Bars ☐ Lines ☐ Points ☑ Stack ☐ Percent Interval 1d

TABLE PANEL — TABLE

Fields

0 to 100 of 500 available for paging

☐ _text_
☐ _version_
☐ count_i

message (select columns from the list to the left)

{"_version_":1516314113135870000,"count_i":1,"doc_id_s":"2009324","filters_orig_ss":
["abcat0100000" "abcat0101000" "abcat0101001" "cat00000"] "filters_s":"abcat0100000 $ abcat0101000 $ abcat0101001 $

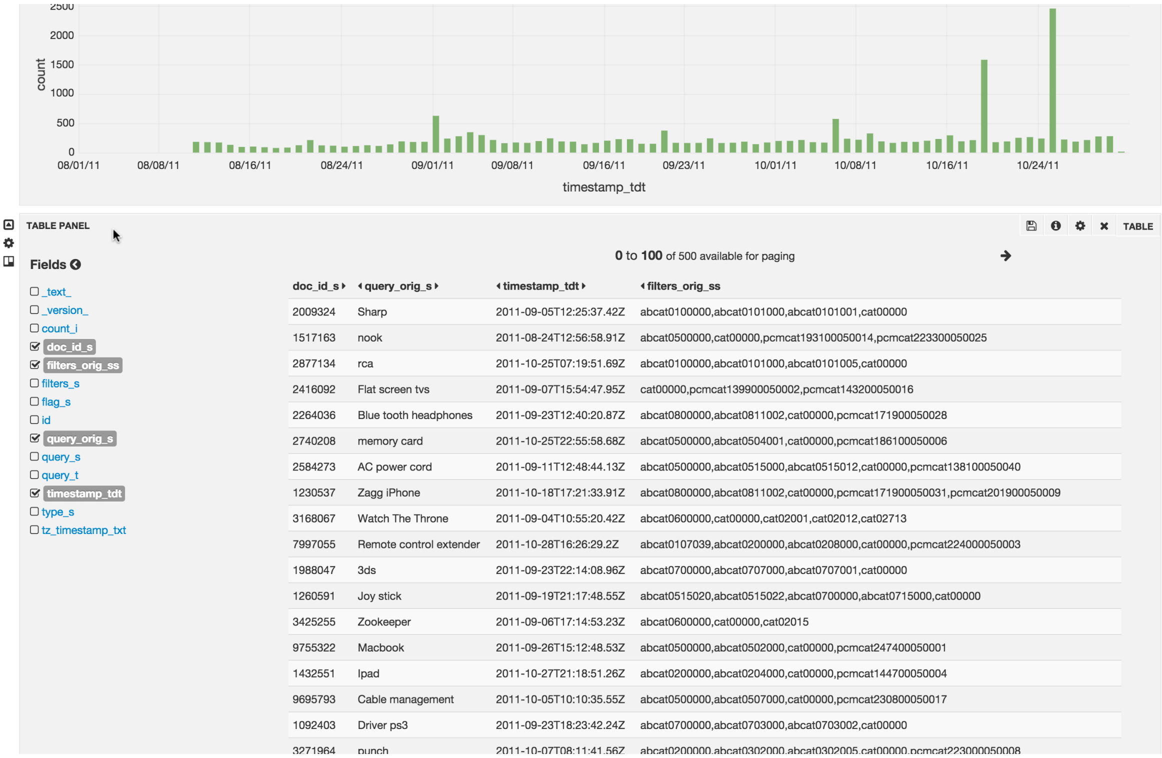
Clicking on the "info" icon on the Histogram controls shows the Solr query parameters:

```
q=*:*&
wt=json&rows=0&
fq=timestamp_tdt:[2011-08-01T04:00:00.000Z TO 2011-11-01T03:59:59.000Z]&
facet=true&
facet.range=timestamp_tdt&
facet.range.start=2011-08-01T04:00:00.000Z&
facet.range.end=2011-11-01T03:59:59.000Z&
facet.range.gap=+1DAY
```

Configure the Table Panel

The table panel displays the documents in the results set, 1 per row. The default display shows all document fields. The table "Fields" control is used to select the fields. The table header row column headers have arrows which allow shifting the column position left or right.

After selecting a subset of the document fields and ordering the display accordingly, the table panel shows the essential raw signal document fields:



- ☐ _text_
- ☐ _version_
- ☐ count_i
- ☑ doc_id_s
- ☑ filters_orig_ss
- ☐ filters_s
- ☐ flag_s
- ☐ id
- ☑ query_orig_s
- ☐ query_s
- ☐ query_t
- ☑ timestamp_tdt
- ☐ type_s
- ☐ tz_timestamp_txt

0 to **100** of 500 available for paging

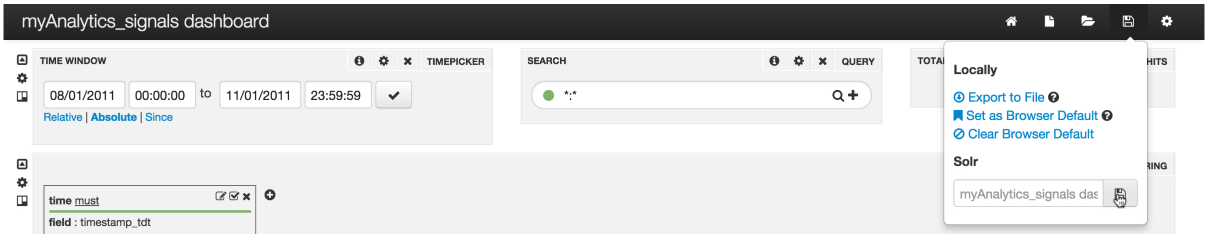
doc_id_s	query_orig_s	timestamp_tdt	filters_orig_ss
2009324	Sharp	2011-09-05T12:25:37.42Z	abcat0100000,abcat0101000,abcat0101001,cat00000
1517163	nook	2011-08-24T12:56:58.91Z	abcat0500000,cat00000,pcmcat193100050014,pcmcat223300050025
2877134	rca	2011-10-25T07:19:51.69Z	abcat0100000,abcat0101000,abcat0101005,cat00000
2416092	Flat screen tvs	2011-09-07T15:54:47.95Z	cat00000,pcmcat139900050002,pcmcat143200050016
2264036	Blue tooth headphones	2011-09-23T12:40:20.87Z	abcat0800000,abcat0811002,cat00000,pcmcat171900050028
2740208	memory card	2011-10-25T22:55:58.68Z	abcat0500000,abcat0504001,cat00000,pcmcat186100050006
2584273	AC power cord	2011-09-11T12:48:44.13Z	abcat0500000,abcat0515000,abcat0515012,cat00000,pcmcat138100050040
1230537	Zagg iPhone	2011-10-18T17:21:33.91Z	abcat0800000,abcat0811002,cat00000,pcmcat171900050031,pcmcat201900050009
3168067	Watch The Throne	2011-09-04T10:55:20.42Z	abcat0600000,cat00000,cat02001,cat02012,cat02713
7997055	Remote control extender	2011-10-28T16:26:29.2Z	abcat0107039,abcat0200000,abcat0208000,cat00000,pcmcat224000050003
1988047	3ds	2011-09-23T22:14:08.96Z	abcat0700000,abcat0707000,abcat0707001,cat00000
1260591	Joy stick	2011-09-19T21:17:48.55Z	abcat0515020,abcat0515022,abcat0700000,abcat0715000,cat00000
3425255	Zookeeper	2011-09-06T17:14:53.23Z	abcat0600000,cat00000,cat02015
9755322	Macbook	2011-09-26T15:12:48.53Z	abcat0500000,abcat0502000,cat00000,pcmcat247400050001
1432551	Ipad	2011-10-27T21:18:51.26Z	abcat0200000,abcat0204000,cat00000,pcmcat144700050004
9695793	Cable management	2011-10-05T10:10:35.55Z	abcat0500000,abcat0507000,cat00000,pcmcat230800050017
1092403	Driver ps3	2011-09-23T18:23:42.24Z	abcat0700000,abcat0703000,abcat0703002,cat00000
3271964	punch	2011-10-07T08:11:41.56Z	abcat0200000,abcat0302000,abcat0302005,cat00000,pcmcat223000050008

9.5.4. Save the Dashboard

The work done to create and configure this new time series dashboard is all work that has been done client-side. In order to save this dashboard for future sessions, it must be saved back to Fusion.

Before saving the dashboard, we rename it to "myAnalytics_signals dashboard" using the "Dashboard Settings" controls, which are accessed by clicking the gear icon on the top menu bar.

The top menu bar diskette icon opens the Save dialog. The option "Solr" saves the configured dashboard to Fusion.



08/01/2011 00:00:00 to 11/01/2011 23:59:59

:

myAnalytics_signals das

time must
field : timestamp_tdt

The top menu bar folder icon opens the Load dialog. From this controls you can either reload or delete a saved dashboard configuration.

When the dashboard configuration is reloaded, the configured Solr query or queries will be rerun and the panel displays will be populated with the results.

9.5.5. Conclusion

In this section we have demonstrated how to create, configure, and save a new dashboard. The essential principles are:

- Dashboards are client-side application.
- Input panels create and submit queries to Fusion

- Dashboard displays are highly configurable.
- Configurations can be saved on the server.

For more on this topic, see the Lucidworks blog:

https://lucidworks.com/blog/2015/04/20/noob-notes-log-analytics-fusion/

9.6. Log Analytics Dashboards

You can build a log analytics dashboard for any Fusion collection that contains one or more logfiles. Each logfile entry contains a timestamp plus some amount of additional event information. The time field must be a Solr trie-date field (with field suffix "_tdt").

The more information and structure that can be extracted from the logfile and modeled in a Solr document, the more possibilities for analytics and visualizations. At a minimum log data must include:

- a timestamp in an allowed standard date/time format
- text message(s) which be unstructured or semi-structured

9.6.1. Search query controls

- query
- timepicker
- dateRange

9.6.2. Display controls

At a minimum, a log analytics dashboard contains a histogram.

9.7. Signals Dashboards

Signals dashboards are a type of time-series dashboard that you can use to monitor signals collections, using the signal timestamp as the time field.

The time field must be a Solr trie-date field (with field suffix "_tdt").

The default dashboard layout for a time-series dashboard is:

Row	Title	Type	Description
1	Time Window	Timepicker	input control which selects timestamp range, faceting granularity
1	Search	Query	keyword search terms
1	Total Hits	Hits	number of results for Solr query
2	-	Filtering	displays Solr params corresponding to Timepicker selections
3	Event Counts	Histogram	binned results as bar chart where X-axis is timeline and Y-axis is signals per date-time interval
4	Table Panel	Table	documents in results set, displayed 1 document per row. Table panel has controls over which fields to display, order in which fields are displayed

9.7.1. Signals Analytics Components

Search query controls

- query
- timepicker
- dateRange

Display controls

At a minimum, a signals analytics dashboard contains a histogram. If the signal contains discrete labels, additional display panels should be added to drill down on the signal contents.

9.8. Non-Time Series Dashboards

Fusion provides the option of creating a non time-series dashboard over the contents of a Fusion collection. For non-time series data, the "filtering" widget controls the faceting.

9.8.1. Search query controls

- query
- filtering

9.8.2. Display controls

If no faceting or quantitative information is present, a table can be used to drill-down on the documents in the Fusion collection.

9.9. Dashboard Layouts

Dashboard layouts are controlled by the Dashboard Settings menu. The dashboard setting menu is toggeled open and close.

Note	Always remember to click "Save" to save your work.

9.9.1. Configuring Rows

A Dashboard consists of one or more rows, each of which contains one or more panels.

The Dashboard display contains a set of control icons on each row allowing you to hide/unhide, position, and delete that row. To add a row, enter a title for the new row, the height of the row, if it is editable, and then click **Create Row**. After the row is created, it will appear in the table of rows, and you can click the up or down arrows to arrange the new row with the existing rows.

9.9.2. Configuring Panels

To add a panel to a row, click the + icon to the right of the last panel in a row; if this icon does not appear, the row is full.

You can also click the 'gear' icon to the left of the first panel. This will bring up the row configuration popup, where the last tab is 'Add Panel'.

When adding a panel, the configuration screen varies depending on the required properties for each panel type. From the Add Panel tab, all of the available properties of each type are displayed, allowing you to define the panel name, the data properties, and any queries that should be used to limit the data used in the panel.

When editing a panel, however, the view is split between three tabs: General, for name and size configuration; Panel, for the data properties; and Queries, for defining queries.

9.9.3. Nested Panel Layout using Column Panels

A column panel is a container panel for other display panels. The properties of a column panel allow you to define the panels you would like included.

Choose the type of panel, and then define the properties according to the type chosen. When you have finished configuring the included panel, click Create Panel.

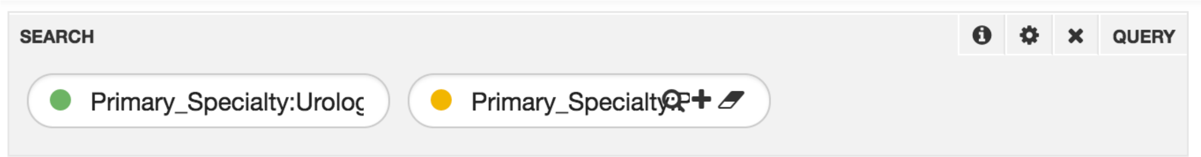
Below the Add Panel to Column area, the configured panels will be shown as a list, in the order they will appear on the dashboard. From this list, you can change the height of each panel, remove panels, change the display order, or temporarily hide panels.

9.10. Input Panels

Input panels let you control which data output panels display. The values you specify in input panels become parts of the Solr queries that output panels use to obtain data.

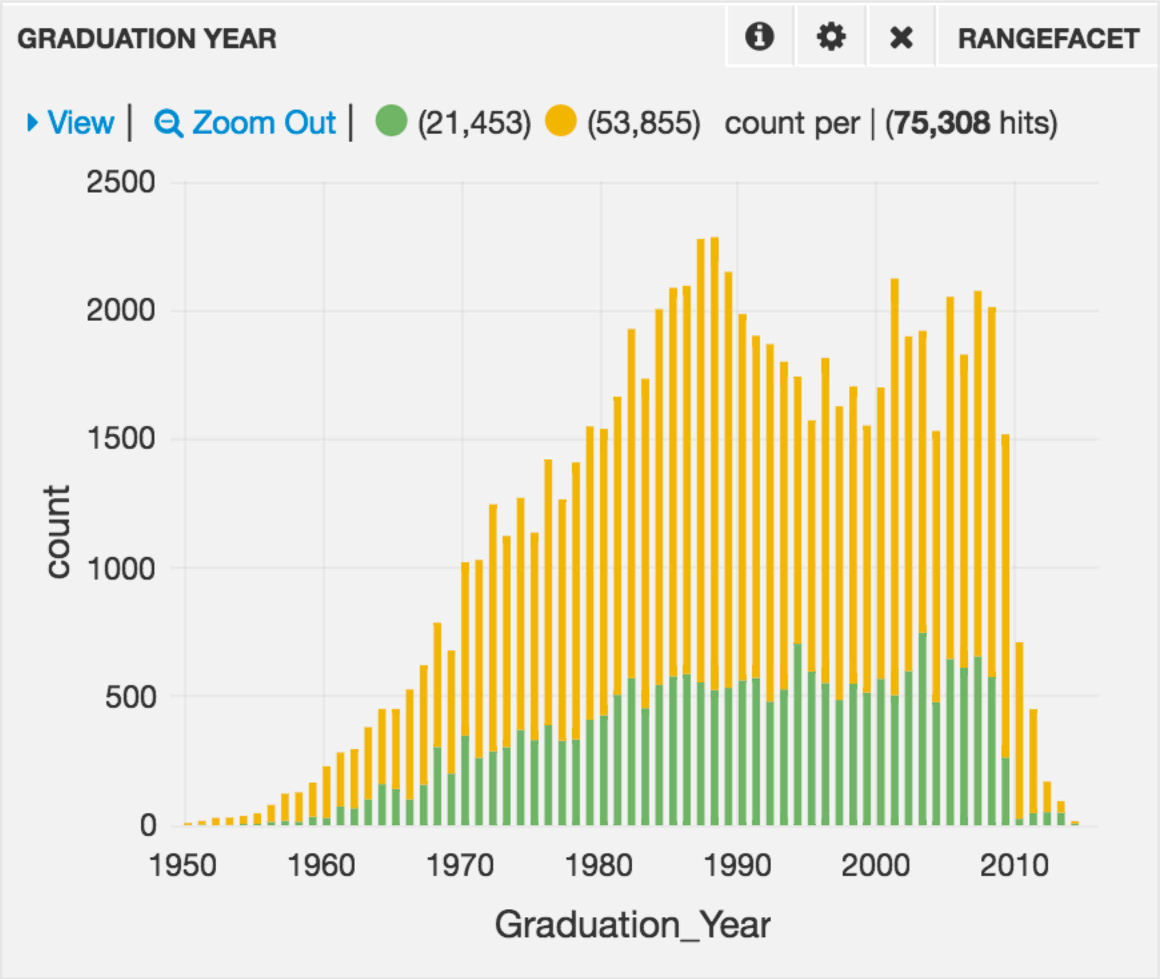
You can use these input panels:

- **Query Panel** – Enter a free-form query (one or more query terms) in a search bar. Add additional search bars to a query panel to submit separate queries. Some visualization panels (for example, Rangefacet) keep the data separate so you can compare it. This is an example of a Query panel with two search boxes. The parameters for query strings are `Primary_Specialty:Urology` and `Primary_Specialty:Psychiatry`. Here is the Query Panel and the resulting Rangefacet panel:

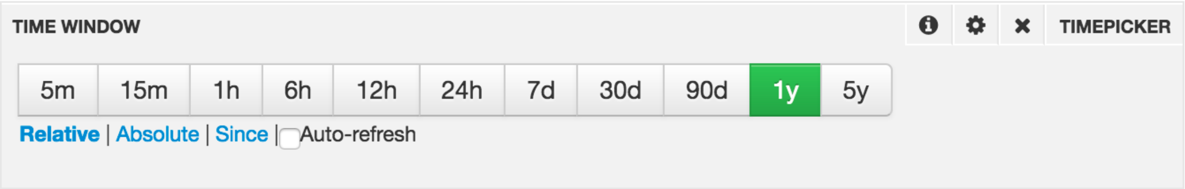


● Primary_Specialty:Uroloç

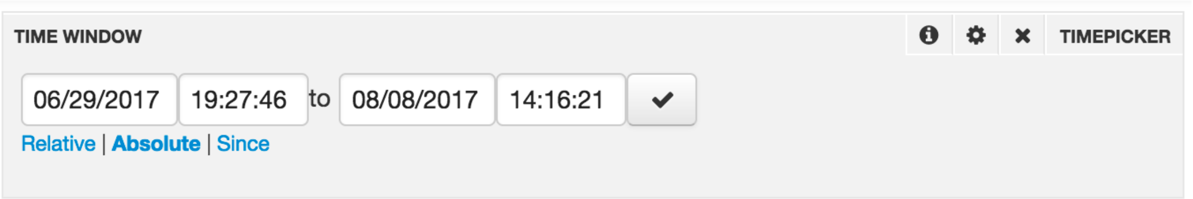
● Primary_Specialty:P



- **Time Picker Panel** – Apply a time range to time-series data. The time range can be:
  - Relative – A time range starting now and reaching backward in time, for example, the last 15 minutes or one hour



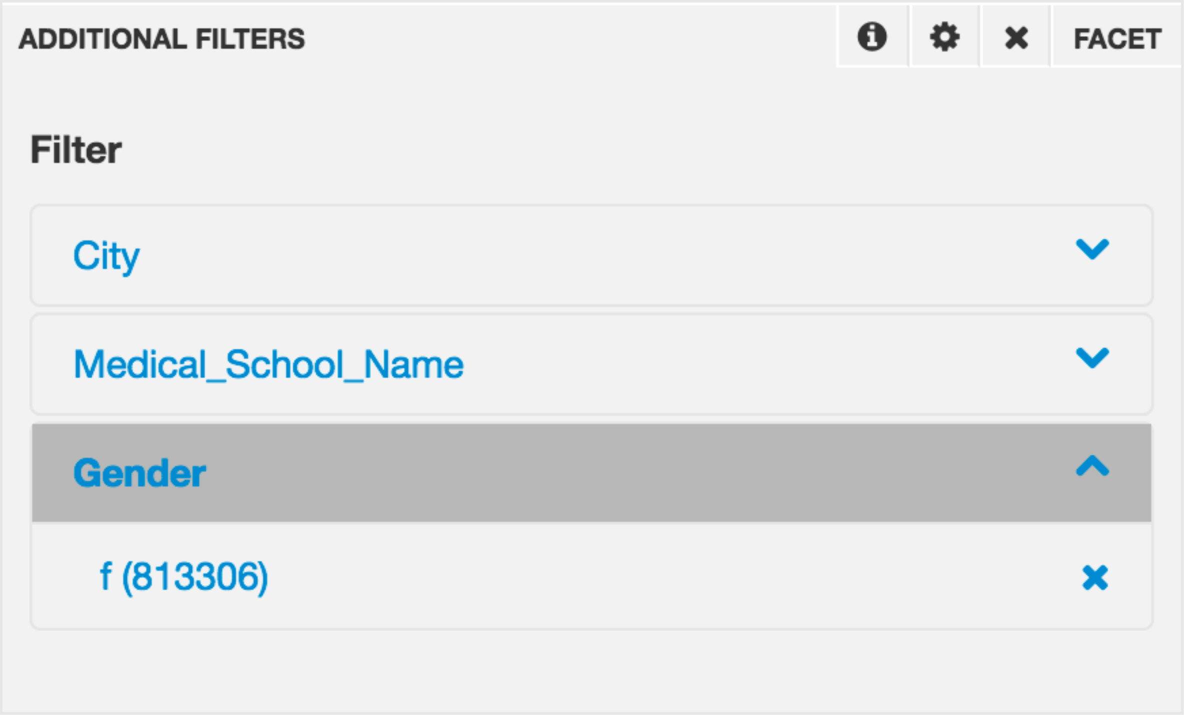
  - Absolute – A specific time-and-date range, for example, from 06/01/2017 00:00:00 to 06/30/17 23:59:59



06/29/2017 19:27:46 to 08/08/2017 14:16:21 ✓

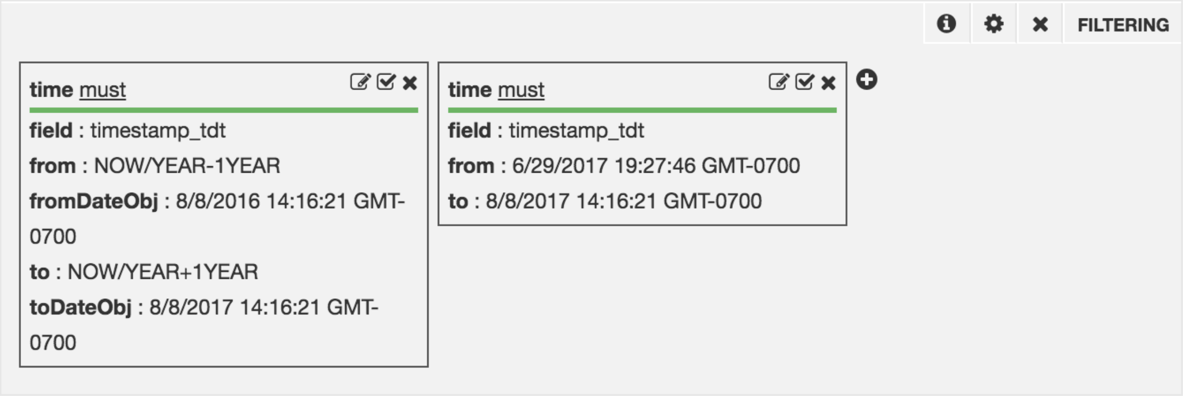
  - Since – A time range since a specific date and time, for example, since 01/01/2017 00:00:00.

- **Facet** – A Facet panel can facet any data field.



f (813306)

- **Filtering** – A Filtering panel lets you apply field-based filters to the hits returned by the query. The filters apply to all display panels.



time must

field : timestamp_tdt

from : NOW/YEAR-1YEAR

fromDateObj : 8/8/2016 14:16:21 GMT-0700

to : NOW/YEAR+1YEAR

toDateObj : 8/8/2017 14:16:21 GMT-0700

time must

field : timestamp_tdt

from : 6/29/2017 19:27:46 GMT-0700

to : 8/8/2017 14:16:21 GMT-0700

> **Important**: You must use a Filtering panel somewhere on your dashboard, so that all panels work correctly when you interact with data.

9.10.1. Query Syntax

Enter a search term or phrase in the search box of the Query panel. Autocompletion provides a list of possibly related prior searches. Finish typing your search query or select an autocompleted query, and then click Search .

Enter queries in a Query panel using Apache Lucene Query Parser syntax. You enter the parameter for the query string (for example, `Susan` or `Gender:F`), not the query string (for example, `q=Susan` or `q=Gender:F`).

Rules for the Simplest Cases

Here are some syntax rules for the simplest cases:

- A single term is either `field:value` or `value`. With `value`, the search is over all fields.
- For an exact-case match, you must specify the field name.
- Surround a term that contains spaces in *double quotation marks* (" ").
- You can't specify `:value or :"value"` to search over all fields; that syntax doesn't work.
- To retrieve all records, use the search term `:`.
- For OR logic, enter `OR` between the terms, or just use spaces. For AND logic, enter `AND` between the terms.

Examples

These are examples of the query syntax:

Goal	Syntax and example
Single term in any field; no blanks in term	`term` (matches any case)
Single term in any field; blanks in term	`"term"` (matches any case)
Multiple terms, each in any field; with OR logic; no blanks in terms	`term1` `term2` …
Multiple terms, each in any field; with OR logic; blanks in terms	`"term1"` `"term2"` … (matches any case)

Goal	Syntax and example
Multiple terms, each in any field; with AND logic (in the same record); no blanks in terms	`term1 AND term2` …
Multiple terms, each in any field; with AND logic (in the same record); blanks in terms	`"term1" AND "term2"` … (matches any case)
Single term in a specific field; no blanks in term	`field:term` (matches any case)
Single term in a specific field; blanks in term	`"field:term"` (matches *exact* case)
Multiple terms, each in a specific field; with OR logic; no blanks in terms	`field1:term1` `field2:term2` …
Multiple terms, each in a specific field; with OR logic; blanks in terms	`"field1:term1"` `"field2:term2"` … (matches *exact* case)
Multiple terms, each in each in a specific field; with AND logic (in the same record); no blanks in terms	`field1:term1 AND field2:term2` …
Multiple terms, each in a specific field; with AND logic (in the same record); blanks in terms	`"field1:term1" AND "field2:term2"` … (matches *exact* case)

For more information about the query syntax, see Standard Query Parser Parameters.

Tip You can use a Text panel to advise the user regarding the syntax of search terms. Also, if you want the dashboard user to explore subsets of the data, use a Filtering panel or a Facet panel to achieve that. Don't expect users to enter complex search expressions. For example, add the field `gender` as a facet, instead of expecting the user to add `AND gender:male` to a search expression.

9.10.2. Inspect a Panel Query

You can't inspect the panel query in the Query panel. You can inspect the panel query in other panels, for example, in a Histogram or Heatmap panel. You can see the contributions that the different parts of the query make. In this example, there are no global query parameters.

Query:

```
start_station_name:"Broadway and E 14 St" AND gender:Male
```

Panel query for a Histogram panel:

The part of the query from the Query panel is the first part, from `q=` through `Male`.

```
q=start_station_name%3A%22Broadway%20and%20E%2014%20St%22%20AND%20gender%3AMale&wt=json&rows=0&fq=start_time:[
2014-01-28T20:16:59.000Z%20TO%202014-03-
15T05:36:55.000Z]&facet=true&facet.range=start_time&facet.range.start=2014-01-
28T20:16:59.000Z&facet.range.end=2014-03-15T05:36:55.000Z&facet.range.gap=%2B12HOUR
```

9.10.3. Query Panel

The query panel provides a search box to allow real-time filtering of data.

It is a best practice to include one of this type of panel on your dashboard. With this panel, you can add, remove, label, pin and color queries.

There are no specific properties for a query panel.

9.10.4. Timepicker Panel

The timepicker panel controls the time range filters. This control is should be included on log and signal analytics dashboards, and any other dashboard based on time-series data.

The configuration properties are:

- **Default Mode**: The options are **relative**, which provides a series of relative timeframes (such as 30 days ago, 1 year, etc.); **absolute**, where you define the start and end dates; or **since**, where you define only the starting date, with the current date assumed.
- **Time Field**: The field to use for time-based data.
- **Relative Time Options**: When the mode is set to relative, you can provide a comma-separated list of relative time options, such as "5m,1h,2d". If you use the default range, you should set the panel to span at least 6, to prevent the time selections from overrunning the edges of the panel.
- **Default Timespan**: The time option that should be selected as a default.
- **Auto-refresh**: When the mode is set to either relative or since, you may want your dashboard to automatically refresh with the latest data. These options allow you to configure auto-refresh:
    - **Enable**: Select to enable auto-refresh.
    - **Interval**: The interval, in seconds, to refresh.
    - **Minimum Interval:** The minimum interval, in seconds. to refresh.

9.11. Display Panels

Display panels on a dashboard display information about the data in a single collection. Fusion has these types of display panels:

- **Layout** – Use these display panels to organize the panels on a dashboard.
- **Textual Information** – Use these panels to display textual information. For example, use a Table panel to displays the values of fields in a set of records.
- **Graphical Visualization** – Use these panels to help users to visualize data. For example, display category data in a bar chart and geographical data on a map.

9.11.1. What Data is Displayed

These things determine which data Fusion displays in a display panel:

- Where the query is sent (to a Fusion query pipeline or to Solr)
- The collection that contains the data. You can specify this when you create a dashboard, or permit the user to choose the collection.
- Global query parameters (optional)
- Input panel configuration settings (possibly including a panel query)
- A user’s selection of the collection (if permitted)
- A user’s query
- A user’s interactions with the data

9.11.2. Layout Panels

These layout panels can help you organize a dashboard:

- **Column** – Lay out panels in a column within a row or part of a row. (Instead of the usual left-to-right layout within rows.)
- **Text** – Add text to a dashboard, for example, to instruct the user regarding how to use the dashboard, or to describe the data.

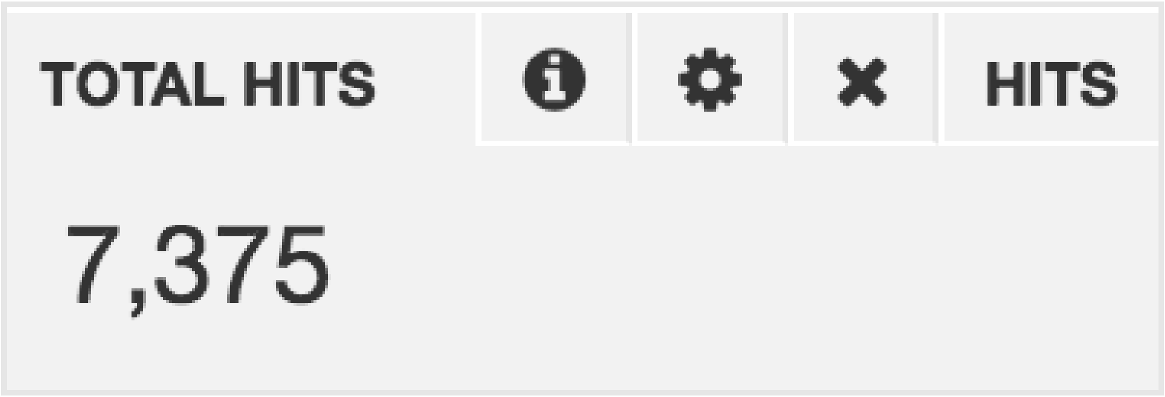
9.11.3. Textual Information Panels

These panels display textual information:

- **Doc Viewer** – Display documents as single pages of information, and let the user page through documents. Paging through documents doesn’t affect the data displayed in other panels.
- **Full Text Search** – Provide full text search capability.

  Note: Make a Full Text Search panel wide. Start with Span = 12. Depending on the content, you might be able to make it narrower. If you make a Full Text Search panel too narrow, content and controls can be inaccessible.

- **Hits** – A Hits panel displays statistical information about the hits returned by the query, including all filters that are applied to the query. The default information is the count (number of hits). You can display the count, minimum, mean, maximum, sum, standard deviation, the sum of squares, and/or the number of hits that lack a value for a field.



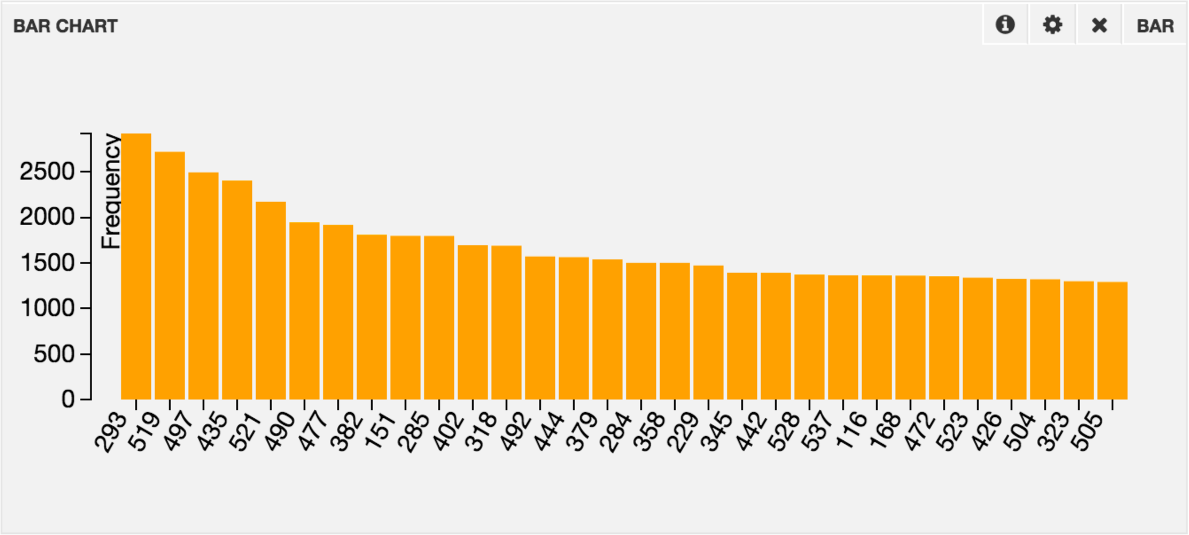
- **Table** – Display data in a table.

  Note: Make a Table panel wide. Start with Span = 12. Depending on the content, you might be able to make it narrower. If you make a Table panel too narrow, content and controls can be inaccessible.

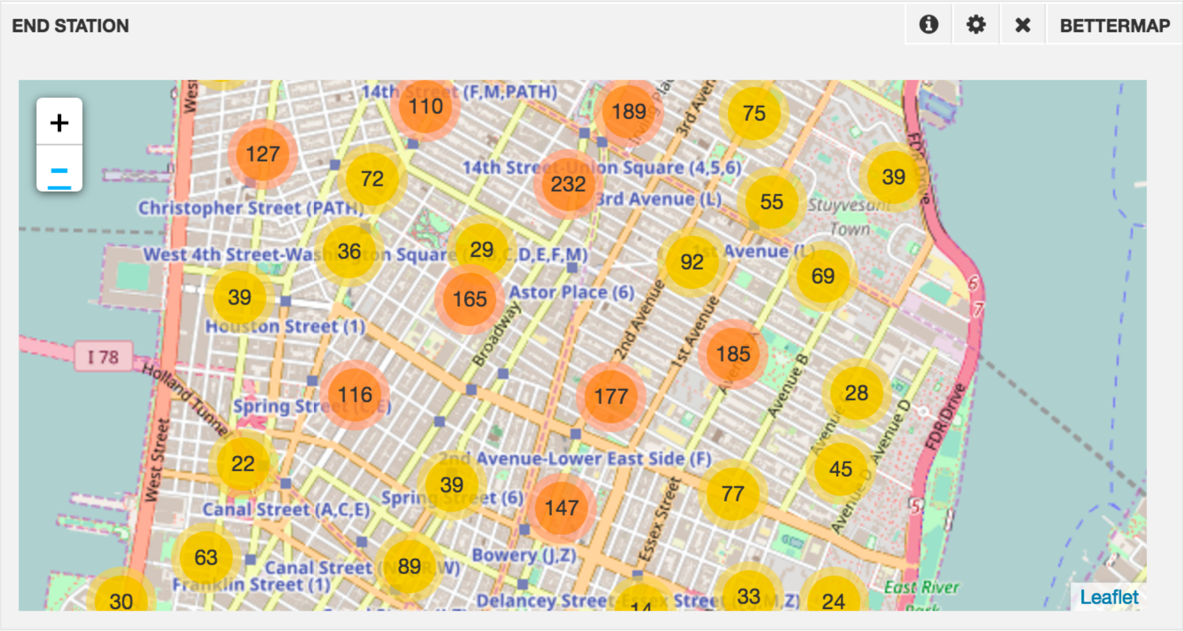
9.11.4. Graphical Visualization Panels

Most display panels are graphical visualization panels. They visually present the data requested in the input panel(s), subject to configuration of the display panel and to any global filtering. Panels for visualizing information graphically are:

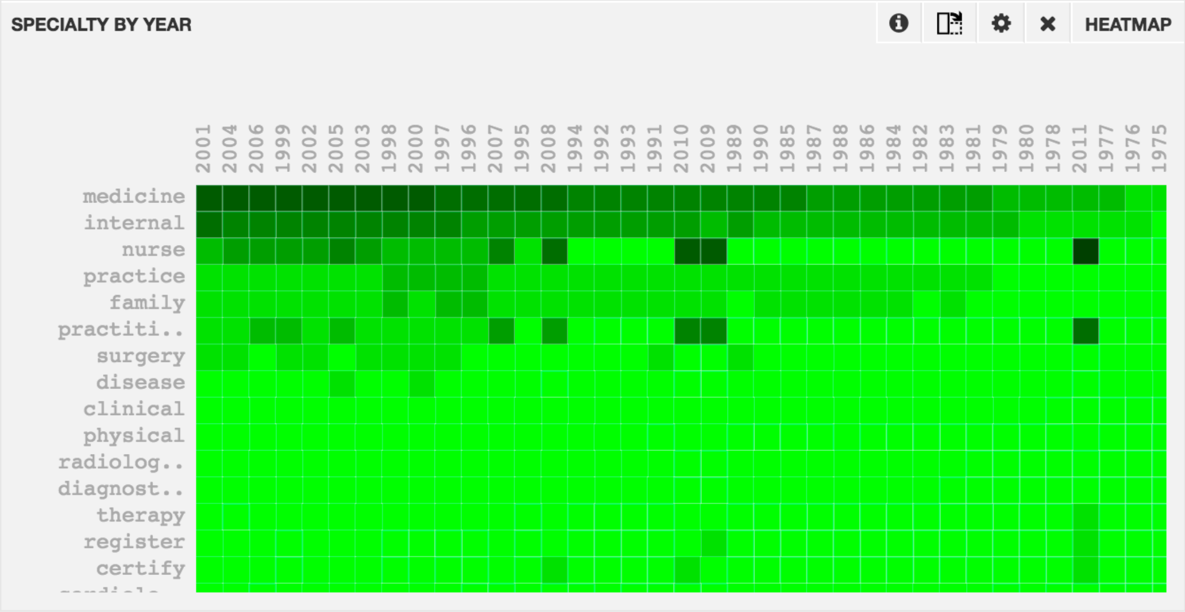
- **Bar** – Graph data as frequencies in a bar chart.



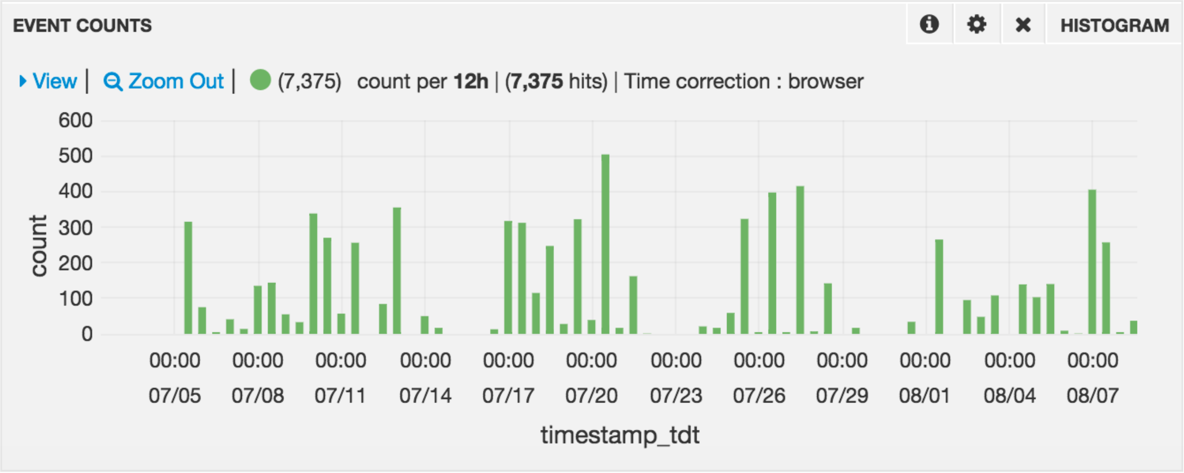
- **Better Map** – Display geolocated points in clustered groups on a map. As you zoom out, points are clustered into fewer groups. As you zoom in, points are clustered into fewer groups (and at some point are not clustered). The Better Map panel *doesn't* use the geospatial search capabilities of Solr. It transfers more data than the Map panel and generally requires more computation, while showing less data. If you have a time filter, the panel will show the most recent points in your search, up to your defined limit. This panel is best used after filtering the results through other queries and filter queries, or when you want to inspect a recent sample of points.



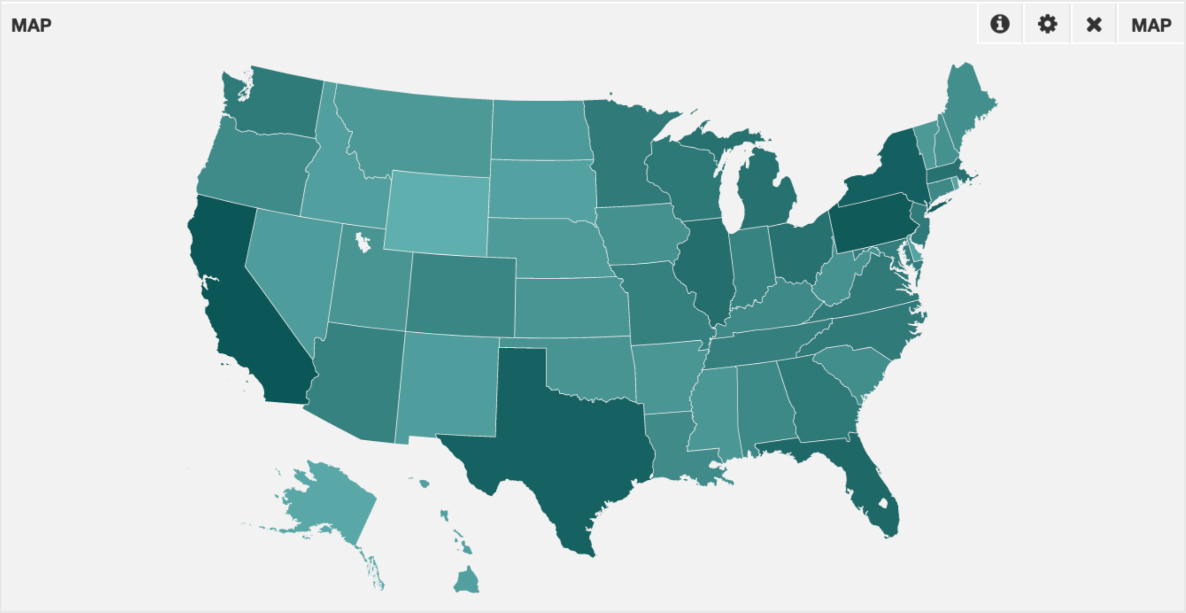
- **Heat Map** – Display a heat map, that is, a graphical representation of data along two facet axes.



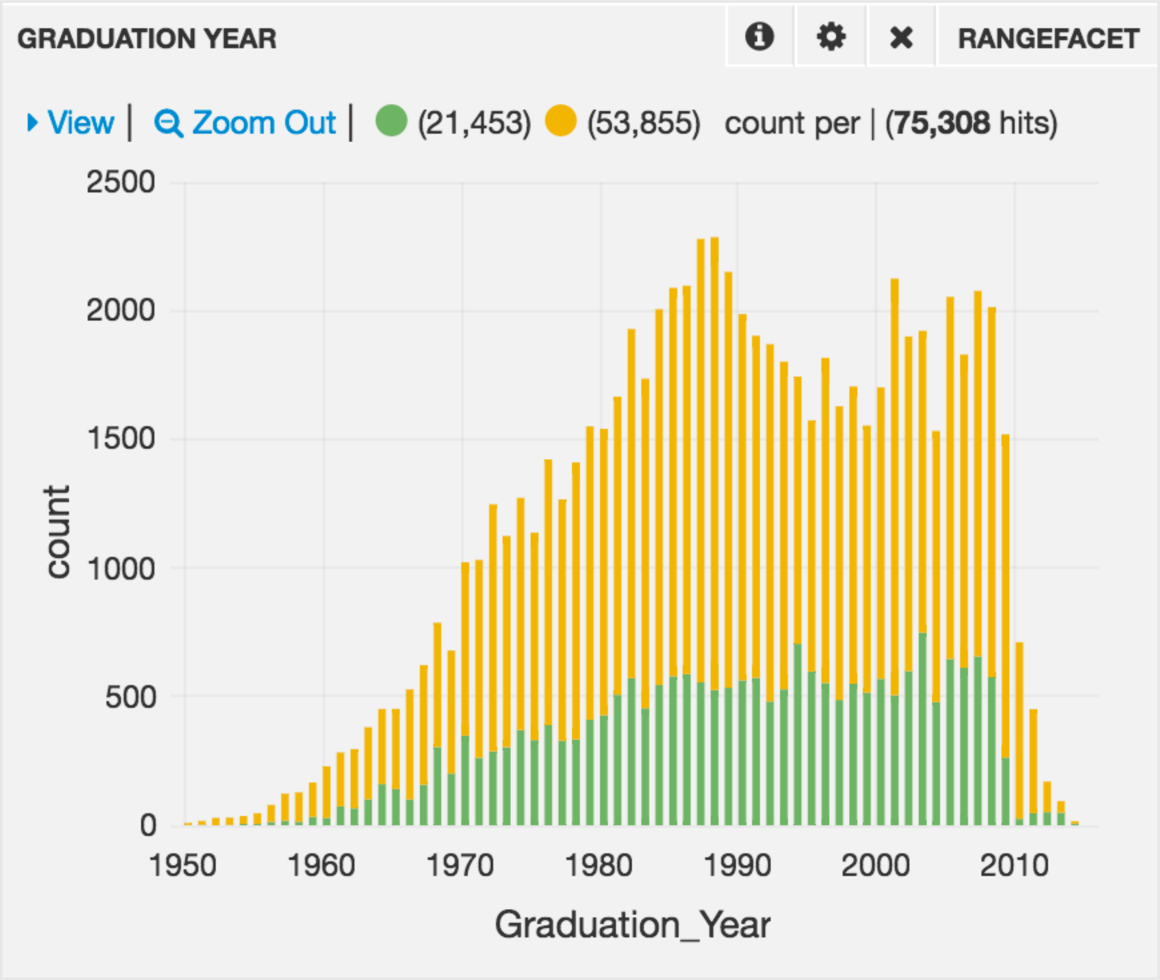
- **Histogram** – Display a histogram. When used in `count` mode, this is a bucketed chart of the current query, including all applied time and non-time filters. When used in `values` mode, the histogram plots the value of a specific field over time, and lets the user group the values based on the values of a second field.



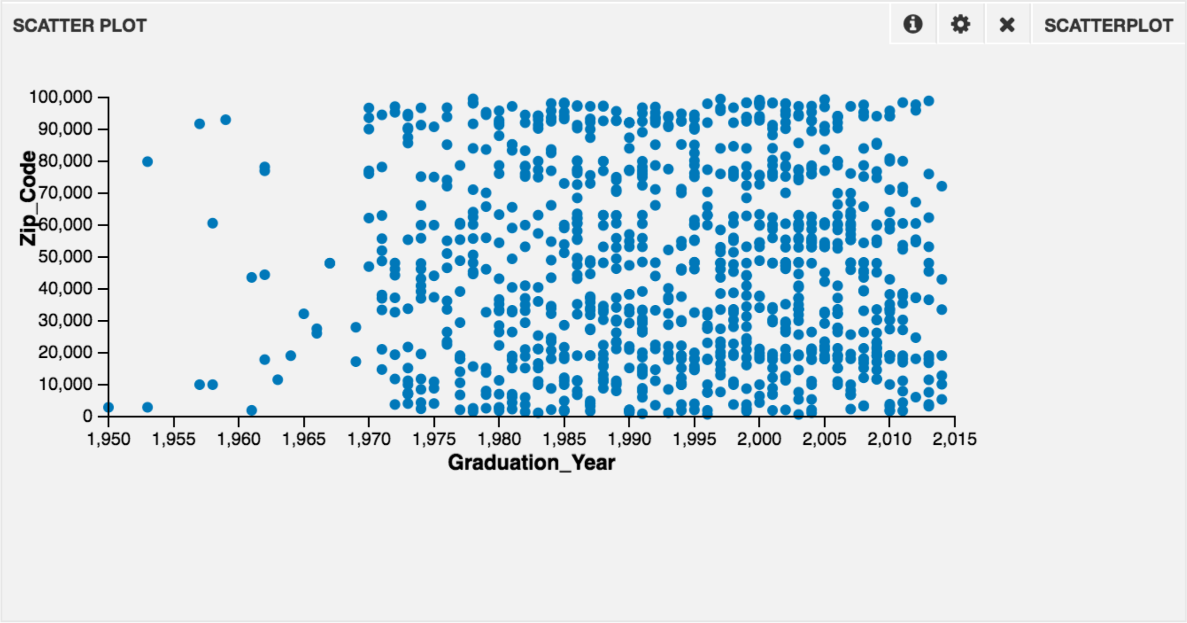
- **Map** – Display a map of shaded regions using a field that contains a 2-letter country code or U.S. state code. Regions with more hits are shaded darker. Instead of a count, you can choose to shade regions based on the minimum, maximum, mean, or sum. The map panel uses facets, so it is important that you set field values to the appropriate 2-letter codes at index time.



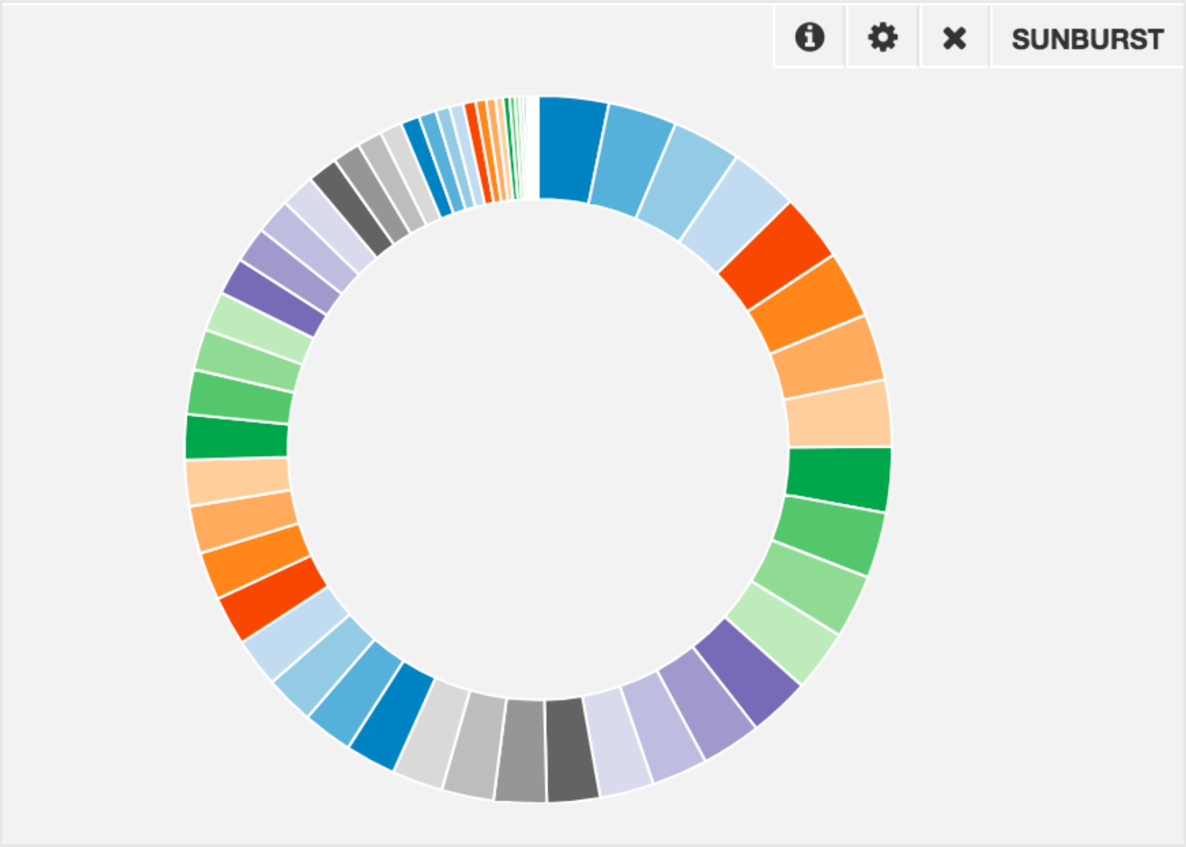
- **Multi-series** – Graph multiple fields of the same type that vary with time in the same graph. The X-axis is used for the time series field. Values in the other fields are graphed along the Y-axis.
- **Range Facet** – Display a histogram of a numeric field. The Range Facet panel is similar to the time series histogram. It lets you select ranges and zooming in/out to the desired numeric range. Range selections in the panel are reflected across the entire dashboard. With multiple search boxes in a Query panel, the Rangefacet panel can display multiple data sets, as shown in this example:



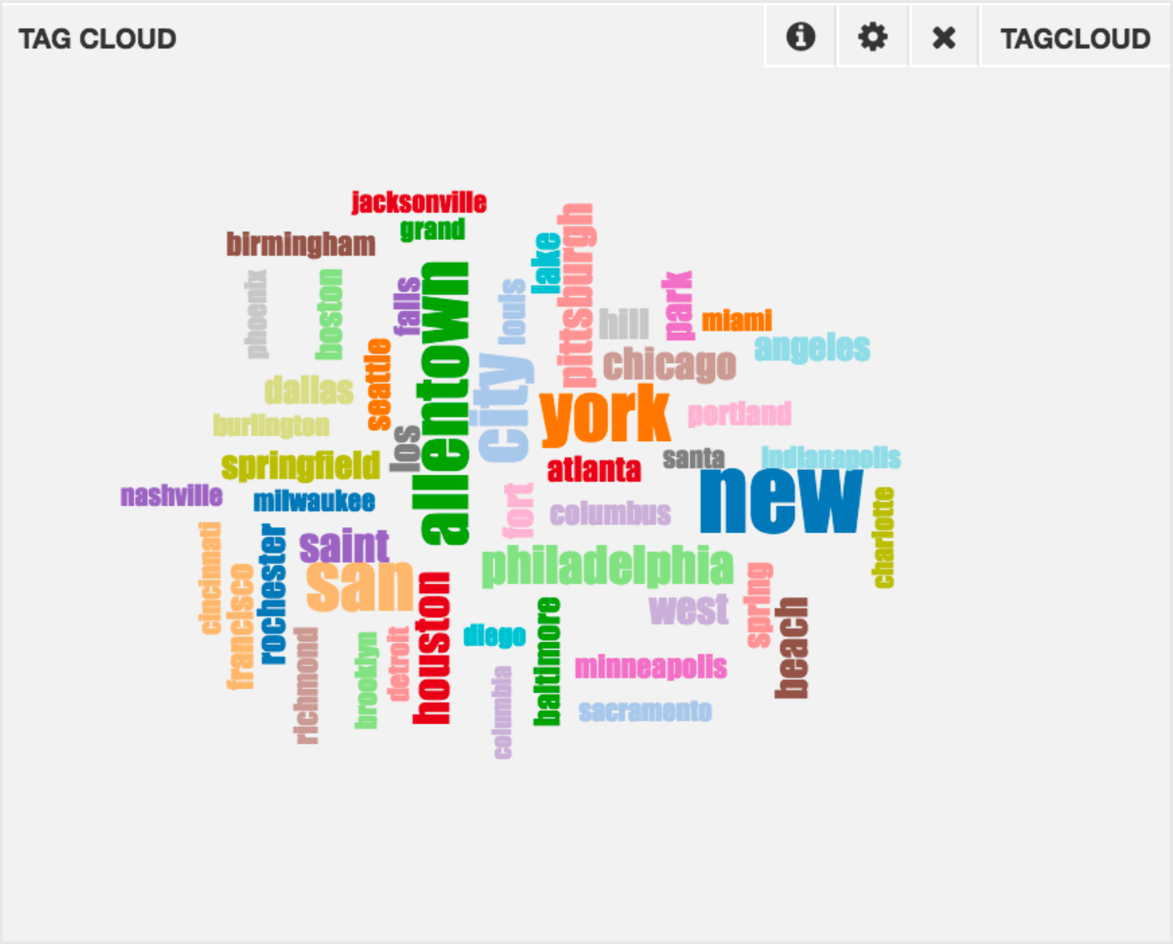
- **Scatter Plot** – Display a scatter plot between two variables or four variables.



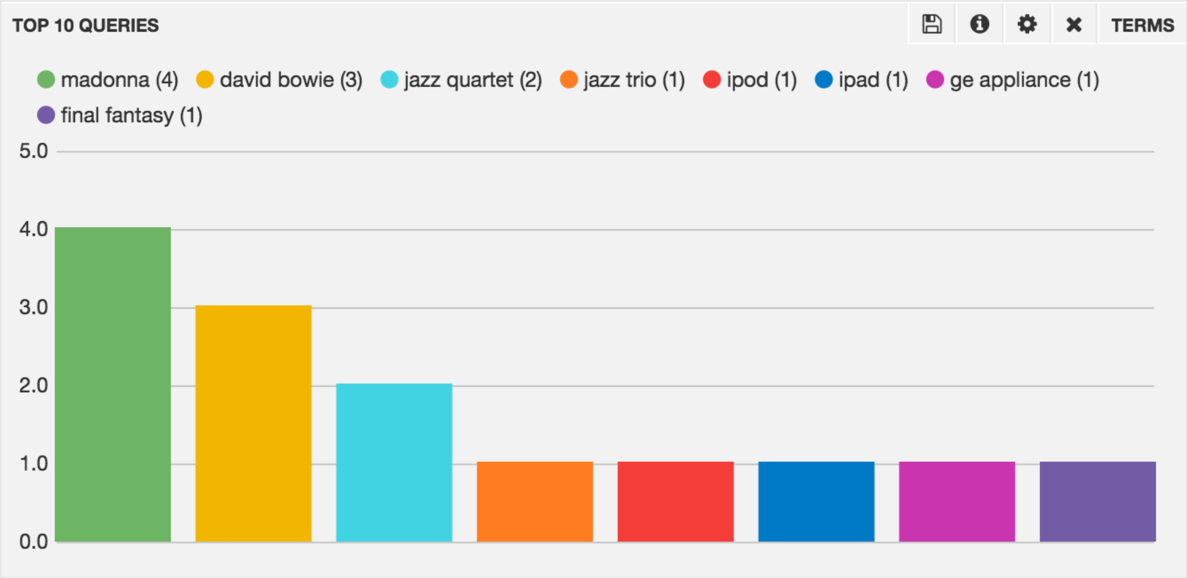
- **Sunburst** – Display a sunburst plot based on facets.

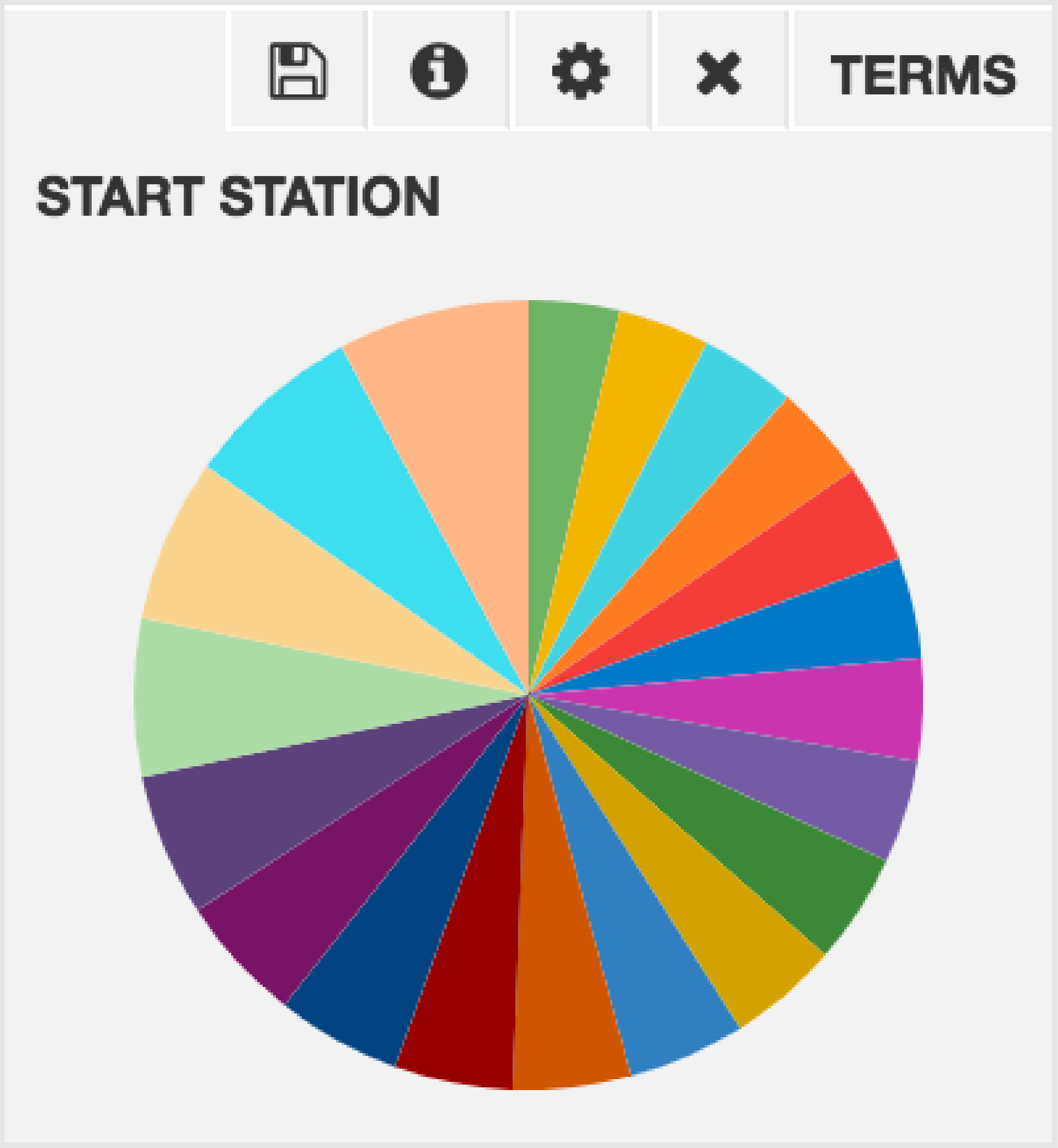


- **Tag Cloud** – Display a tag cloud of the top N words in a field.



- **Terms** – Display the count, minimum, maximum, mean, or sum for a Solr facet as a bar chart, pie chart, or table.




TERMS
START STATION

9.11.5. Bettermap Panel

The bettermap panel displays geographic points in clustered groups on a map.

This panel type does not use the terms facet and it does query sequentially. This means that it transfers more data and is generally heavier to compute, while showing less actual data

If you have a time filter, it will attempt to show to most recent points in your search, up to your defined limit.

9.11.6. Filtering Panel

A Filtering Panel shows the ranges applied to the query facets. It is a feedback/information panel that shows the limits applied to the data. A Filtering Panel should always be included on an analytics dashboard as an aid to interpreting/understanding the other visualizations on the dasboard.

9.11.7. Heatmap Panel

The heatmap panel provides a heat map for representing pivot facet counts.

The Panel tab defines what data is displayed in the panel and how it is displayed.

Row Field: The field from Solr that will provide data for rows.

Column Field: The field from Solr that will provide data for columns.

Rows Limit: The maximum number of rows to display.

Heatmap Color: The base color for the heatmap. The intensity of the color in any cell is proportional to the pivot facet count for that cell.

Transposed

9.11.8. Histogram Panel

The histogram panel provides a binned display of queries per time interval, using Solr’s range facets for data.

Configuration Options

Mode

The value for the y-axis. The options are **count** or **values**.

When choosing values, you must select a field to use as the basis for the values. This field must have a numeric field type. You can also select a Group By Field, which cannot be a multi-valued field, and creates multiple charts.

Time_field

The value for the x-axis. This is the field to use for display of the time information for the histogram.

Chart Settings

There are several chart settings.

- **Bars**: Enable to show bars on the chart.
- **Lines**: Enable to show lines on the chart. When enabled a few more settings will be available:
  - **Line Fill**: The fill area, from 0-10.
  - **Line Width**: The width of the lines, in pixels.
  - **Smooth**: Enable to remove 0 values from lines.
- **Points**: Enable to show points on the chart.
- **Stack**: Enable to stack multiple series together.
- **Percent**: Enable to show the stack as a percentage of the total (only displayed when stack is enabled).
- **Legend**: Enable to show the legend.
- **xAxis**: Enable to display the x-axis values.
- **yAxis**: Enable to display the y-axis values.
- **Time correction**: If time correction should be applied to use the browser’s timezone. Select 'utc' to always display times in UTC.
- **Selectable**:
- **Zoom Links**: Enable to allow users to zoom out in time.
- **View Options**: Enable to show an options section.
- **Auto-interval**: Enable to allow the chart to automatically scale the intervals.
- **Resolution**: When Auto-interval is enabled, a best effort will be made to show this number of bars.

Tooltip Settings

The tooltip settings control the display of data when users hover over a line or bar on the chart.

- **Stacked Values**: When using stacked values, this defines if the data be displayed as cumulative, or as individual values.

- **Display Query**: If an alias is set, it will be shown in the tooltip. If no alias is set, enable this to show the entire query.

9.11.9. Hits Panel

The hits panel shows the total hits for the current query input to a query panel.

The properties allow providing a title for the panel, the style of results, and the font size.

9.11.10. Map Panel

A map panel displays a map of shaded regions using any field that contains a 2-letter country or US state code. Regions with more hits are shaded darker.

This uses the Solr terms facet, so it is important that you set it to the correct field.

The following properties are defined for a map panel:

Field: the Solr field that has the 2-letter codes that will be used for the location data.

Max: a maximum number of countries to plot. The default is 100.

Map: the style of map to display. If your data spans the world, you can choose the world map. If your data is focused on Europe or the US instead, you can choose the Europe or US maps respectively.

Mode: the approach to summarizing the data. You can chose count, mean, maximum, minimum or sum. In order to use any mode other than count, the field type must be numeric.

Decimal Points: the number of digits to display after a decimal points.

9.11.11. Table Panel

The table panel allows you to create a table of field values from the Solr index that match the filters applied to the dashboard. While only the fields selected are displayed, clicking on any entry expands to show all fields of the document.

You can also export the documents, if needed, to CSV, XML or JSON by clicking the "Export" icon in the upper right of the panel.

In the display, you can add new fields on the fly, but clicking a field name from the list in the left side of the panel.

Configuration Options

Add Column

Enter the field(s) you would like to see in the table. You can enter as many fields as you'd like, but too many columns may require you to scroll horizontally to see all of your data.

Click the '+' button to add the field, and you should see it listed in the **Columns** section to the right. Click on a column to remove it from the list.

Options

The display options allow defining how the table is displayed.

- **Header**: Enable to see a header row in the table.
- **Sorting**: Enable to be able to sort the data in the table.
- **Sort**: Choose the field to sort on. Also choose the sort order by selecting the up or down carat to the right of the column name.
- **Font Size**: Choose the font size for the display of the data.
- **Trim Factor**: If the data is too long to fit in the column at your desired width, you can configure the point at which the data will be "trimmed" from display.

Paging

The paging options allow control over how to deal with large amounts of data that may be easier to work with on separate pages.

- **Show Controls**: Enable to show page forward and back options.
- **Overflow**: If the data expands beyond the height of the window, you can choose to **scroll** through the records, or enable **overflow** to expand the size of the panel to fit all of the data for that page.
- **Per Page**: The number of items to display on each page.
- **Page Limit**: The number of pages to display.
- **Pageable**: This will automatically update based on the values of Per Page and Page Limit, to show the total number of items that will be displayed in the table.

9.11.12. Terms Panel

The terms panel displays the results of a Solr facet as a pie chart, bar chart, or a table.

A statistics field can be displayed as min/max/mean/sum, faceted by the Solr facet field, also displayed either as a pie chart, bar chart or a table.

A terms panel takes several properties, described below.

- **Field**: The field to use as the basis of the facets that are used for display.
- **Length**: The maximum number of terms to display.
- **Order**: The sort order of the facets.
- **Style**: The style of chart, either **bar**, **pie** or **table**.
- **Legend**: If you choose bar or pie as the style, you can then choose no legend, or to display it above or below the data.
- **Font Size**: If you choose table, you will be given the option to define the font size.
- **Missing**: Enable to display missing values.
- **Other**:
- **Donut**: If you choose pie chart as the style, you can choose to display the chart as a donut, with an empty circle in the middle.
- **Tilt**: If you choose pie chart as the style, you can choose to tilt the chart as an added effect.
- **Labels**: Enable to show labels in the chart for your data.
- **Mode**: The mode for the data. Choose **count**, **mean**, **min**, **max**, or **sum**. If choosing any mode other than count, the Stats Field selected must be a numeric field.
- **Stats Field**: If you choose any mode other than count, you must then specify the field to use for statistics. This field must be a numeric field.
- **Display Precision**: Choose the number of digits to display after a decimal point, as appropriate.

9.11.13. Text Panel

A text panel displays static text, which can be either unformatted plain text or formatted using either Markdown or HTML.

To configure a text panel, first choose the text format mode, then enter the panel contents. For plain text displays, the font size can be specified.

9.11.14. Ticker Panel

The ticker panel provides a stock-ticker style representation of how queries are moving over time.

When configuring a ticker panel, there is one primary property, "Time Ago", which defines the point in time to use as the basis for comparison.

For example, if the time is 1:10pm, your time picker was set to "Last 10m", and the Time Ago parameter was set to '1d', the panel would show how much the current query results have changed since 1:00 to 1:10pm yesterday.

Chapter 10. Dev Ops

Performance monitoring tools and reports are covered in System Metrics.

Task scheduling tools are covered in Schedules.

Fusion provides system messages and notifications via Messaging and Alerting.

10.1. System Metrics

Fusion continuously indexes System and Solr metrics to the system collection `system_metrics`.

There are around 600 different metrics available. In this topic we've highlighted a few that are likely to be the most useful or interesting to you.

The `/system/metrics` endpoint of the System API lists all the metrics that the system is currently collecting. Metrics are returned for the current instance only; Fusion instances do not aggregate metrics between nodes.

10.1.1. Types of Metrics Collected

There are several types of metrics:

- Gauges: These are single values, valid for the point in time at which the metrics are collected.
- Counters: These are values that are incremented or decremented over time.
- Meters: These measure the rate of events over time. They include a mean rate, as well as a 1-, 5- and 15-minute moving average. Most of these moving averages are exponentially weighted, so that more recent values contribute more heavily than older values; exceptions to this rule have the word "unweighted" in their name.
- Histograms: These measure the distribution of values. They will report the minimum, maximum, mean, and the values at the 50th, 75th, 95th, 98th, 99th, and 99.9th percentiles.
- Timers: A timer is a meter combined with a histogram; it measures the length of time that a particular operation takes (both mean duration and moving averages) as well as the distribution of those durations.

Many of the metrics are for internal use by the system. However, Fusion may ask for a dump of the metrics data (using the System API endpoint) to help diagnose performance issues. Some metrics are also subject to change pending performance tuning and additional testing.

10.1.2. Metrics of Particular Interest

Slow Web Service Calls

For each web service endpoint in the system, the system keeps a list of the last several requests whose request time has been in the 99th percentile – that is, examples of the top 1% of slow requests for that endpoint. These are recorded as `com.lucidworks.apollo.resources.serviceName.methodName.weighted.slow.examples`, where serviceName is the name of the service and methodName is the name of a valid method for that service.

This information might be helpful when diagnosing performance issues. Here is an example of the 5 slowest calls to the getCollectionMetrics method of the CollectionResource service:

```
"com.lucidworks.apollo.resources.CollectionResource.getCollectionMetrics.weighted.slow.examples" : {
      "value" : [ {
        "requestUri" : "http://localhost:8765/api/v1/collections/lws5_metrics/stats",
        "queryParams" : { },
        "userPrincipal" : null,
        "method" : "GET",
        "cookies" : { }
      }, {
        "requestUri" : "http://localhost:8765/api/v1/collections/logs/stats",
        "queryParams" : { },
        "userPrincipal" : null,
        "method" : "GET",
        "cookies" : { }
      }, {
        "requestUri" : "http://localhost:8765/api/v1/collections/logs/stats",
        "queryParams" : { },
        "userPrincipal" : null,
        "method" : "GET",
        "cookies" : { }
      }, {
        "requestUri" : "http://localhost:8765/api/v1/collections/lws5_metrics/stats",
        "queryParams" : { },
        "userPrincipal" : null,
        "method" : "GET",
        "cookies" : { }
      }, {
        "requestUri" : "http://localhost:8765/api/v1/collections/lws5_metrics/stats",
        "queryParams" : { },
        "userPrincipal" : null,
        "method" : "GET",
        "cookies" : { }
      } ]
    }
```

System Memory

There are several memory-related metrics reported:

- `mem.heap.used`: the current amount of heap memory, in bytes, used by the system.
- `mem.heap.max`: the maximum amount of heap memory, in bytes, that the system could use.
- `mem.heap.usage`: the percentage (0 - 1.0) of available heap memory that the system is currently using (this is equal to `mem.heap.used` / `mem.heap.max`).
- `mem.non-heap.used`: the current amount of non-heap memory (also called "off-heap memory"), in bytes, used by the system.
- `mem.non-heap.max`: the maximum amount of non-heap memory, in bytes, that the system could use.
- `mem.non-heap.usage`: the percentage (0 - 1.0) of available non-heap memory that the system is currently using (this is equal to `mem.non-heap.used` / `mem.non-heap.max`).
- `mem.total.used`: the current total amount of memory (heap plus non-heap), in bytes, used by the system.
- `mem.total.max`: the maximum amount of total memory (heap plus non-heap), in bytes, that the system could use.

Here is an example of `mem.heap.used`:

```
{
  "version" : "3.0.0",
  "gauges" : {
    "mem.heap.used" : {
      "value" : 94783360
    }
  },
  "counters" : { },
  "histograms" : { },
  "meters" : { },
  "timers" : { }
}
```

Query and Index Pipeline Stage Metrics

For each query pipeline and index pipeline stage, Fusion collects aggregate performance metrics for successful executions and for errors. All executions for each stage are stored in a metric named `stages.stageType.stageName.process`, where stageType is the type of stage, and stageName is the name of a specific stage.

Here is an example of a request to get the performance metrics for an index pipeline stage named 'solr-default' (`stages.solr-index.solr-default.process`), which is included with Fusion:

```
{"version" : "3.0.0",
  "gauges" : { },
  "counters" : { },
  "histograms" : { },
  "meters" : { },
  "timers" : {
    "stages.solr-index.solr-default.process" : {
      "count" : 109195,
      "max" : 0.128585,
      "mean" : 0.004011065175097276,
      "min" : 0.0022500000000000003,
      "p50" : 0.0030645000000000004,
      "p75" : 0.0033495,
      "p95" : 0.005410449999999992,
      "p98" : 0.014195759999999965,
      "p99" : 0.02462230000000001,
      "p999" : 0.12850243700000002,
      "stddev" : 0.007408363728123277,
      "m15_rate" : 11.957732876922531,
      "m1_rate" : 8.784289947811962,
      "m5_rate" : 9.037172472578138,
      "mean_rate" : 9.214233776748047,
      "duration_units" : "seconds",
      "rate_units" : "calls/second"
    }
  }
}
```

This shows the number of uses of the stage ("count"), the maximum and minimum times, the mean, the 50th, 75th, 95th, 98th, 99th, and 99.9th percentiles (p50, p75, etc.), and the mean rates over 1-, 5- and 15-minute intervals ('m1_rate', etc.). In this case, the pipeline has been used 109,195 times, with a mean rate of 9.214 events per second, with only .003 events in the 50th percentile.

Metrics for successful completions of stages are stored in metrics named `stages.index.stageType.stage.stageName.ok` or `stages.query.stageType.stage.stageName.ok`, depending on if the stage is part of an index pipeline or a query pipeline. Here is an example of the mean rates for successful runs of the 'solr-default' index pipeline stage (`stages.index.solr-index.stage.solr-default.ok`):

```
{
  "version" : "3.0.0",
  "gauges" : { },
  "counters" : { },
  "histograms" : { },
  "meters" : {
    "stages.index.solr-index.stage.solr-default.ok" : {
      "count" : 110855,
      "m15_rate" : 5.270163206842968,
      "m1_rate" : 8.485969925086419,
      "m5_rate" : 8.06785229981572,
      "mean_rate" : 9.18230056255745,
      "units" : "events/second"
    }
  },
  "timers" : { }
}
```

This shows the number of uses of the stage ("count") and the mean rates over 1-, 5- and 15-minute intervals ('m1_rate', etc.). From the above, we can see that the solr-default stage has been executed 110,855 times, with a mean rate of 9.18 events per second.

If you prefer to see the metrics for the entire stage type, you can omit the stage name entirely, and simply get metrics for the stage type. This takes the form of `stages.index.stageType.ok` (for an index pipeline) or `stages.query.stageName.ok` (for a query pipeline). Here is an example, using the solr-index stage type:

```
{
  "version" : "3.0.0",
  "gauges" : { },
  "counters" : { },
  "histograms" : { },
  "meters" : {
    "stages.index.solr-index.ok" : {
      "count" : 116425,
      "m15_rate" : 6.178851947720613,
      "m1_rate" : 8.814380052133192,
      "m5_rate" : 8.585203640734829,
      "mean_rate" : 9.19499774409566,
      "units" : "events/second"
    }
  },
  "timers" : { }
}
```

In this example, we see that the solr-index stage has been successfully run 116,425 times, with a mean rate of 9.19 events per second.

Web Service Endpoint Metrics

For each web service endpoint, Fusion keeps a timer recording the duration and rate of requests. The duration is calculated using an exponentially-weighted moving average with a heavy bias toward measurements from the last 5 minutes.

These metrics have names in the form: `com.lucidworks.apollo.resources.serviceName.methodName.weighted.timer`, or for a specific example, `com.lucidworks.apollo.resources.CollectionResource.getCollectionMetrics.weighted.timer`:

```
"com.lucidworks.apollo.resources.CollectionResource.getCollectionMetrics.weighted.timer" : {
      "count" : 2624,
      "max" : 0.134712,
      "mean" : 0.031589107976653694,
      "min" : 0.022424000000000003,
      "p50" : 0.028440000000000003,
      "p75" : 0.036908,
      "p95" : 0.044644449999999995,
      "p98" : 0.05026944,
      "p99" : 0.05444051000000004,
      "p999" : 0.134693411,
      "stddev" : 0.00936497282768644,
      "m15_rate" : 0.07113433590025664,
      "m1_rate" : 0.06387037028343223,
      "m5_rate" : 0.06218407166715861,
      "mean_rate" : 0.0663172057583814,
      "duration_units" : "seconds",
      "rate_units" : "calls/second"
    }
```

Solr Request Metrics

The system keeps track of the performance of requests to each Solr server that it communicates with.

The metrics have names in the form `solr.solrIdentifier.requestType`. The solrIdentifier is the address of the Solr instance, and the requestType can be 'get-requests', 'post-requests' or 'put-requests'.

This example shows get-requests to a Solr instance that is found on '10.0.1.8' and port 8983:

```
{
  "version" : "3.0.0",
  "gauges" : { },
  "counters" : { },
  "histograms" : { },
  "meters" : { },
  "timers" : {
    "solr.10.0.1.8-8983.get-requests" : {
      "count" : 3170,
      "max" : 0.873981,
      "mean" : 0.2451200904669261,
      "min" : 0.001678,
      "p50" : 0.318176,
      "p75" : 0.48169550000000005,
      "p95" : 0.53017705,
      "p98" : 0.5617982399999999,
      "p99" : 0.6281221800000003,
      "p999" : 0.8710894970000004,
      "stddev" : 0.2448979377578966,
      "m15_rate" : 0.020593265615577774,
      "m1_rate" : 0.032494324572729 69,
      "m5_rate" : 0.030788223074952624,
      "mean_rate" : 0.033875616252208286,
      "duration_units" : "seconds",
      "rate_units" : "calls/second"
    }
  }
}
```

From this we can see that there have been 3,170 GET requests to that Solr instance, and the mean response rate is .03 requests per second.

10.1.3. Changing Metric Collection Frequency

The default frequency to collect metrics is 60 seconds. Since the metrics are stored in a system collection (and a Solr instance), the data can grow to be quite large over time. If you do not need metrics collection to happen as frequently (perhaps during initial implmentation), you can change the frequency by modifying the `com.lucidworks.apollo.metrics.poll.seconds` configuration parameter with the Configurations API.

For example:

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d '600'
http://localhost:8764/api/apollo/configurations/com.lucidworks.apollo.metrics.poll.seconds
```

To disable metrics, you could set the `com.lucidworks.apollo.metrics.poll.seconds` parameter to '-1'.

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d '-1'
http://localhost:8764/api/apollo/configurations/com.lucidworks.apollo.metrics.poll.seconds
```

10.2. Schedules

Schedules in Fusion allow you to execute any Fusion service, Solr request, or other HTTP request on a defined timetable.

For example, you could schedule a Solr query to run at a specified time every day, or you could define a datasource to be re-crawled once a week. The schedules service does not execute any business logic; the service at the specified endpoint must provide this.

The Fusion scheduler is fault-tolerant and distributed across the nodes of your cluster. Several instances of the scheduler service can run on different nodes, but only one of them at a time executes and modifies schedules. This instance is elected as the schedule "leader", which occurs in ZooKeeper in a similar way to how SolrCloud node leaders are elected. The instances that are not the leader are on standby in case the leader goes down. The schedule job definitions are also kept in ZooKeeper, which allows them to be restored to any node whenever needed.

10.2.1. Scheduler job definitions

When defining a job with the scheduler service, there are two main aspects to configuration:

- The time properties that define when the job will run and how often it will repeat
- The call properties defining the call that will be executed

Time properties

These properties define the start time, end time, and repeat interval, if any.

- `startTime` defines when the job should first run.
- `endTime` defines when the job should no longer run.

  The `endTime` does not stop a running job; instead it has the same effect as setting the entire schedule to "inactive" at a certain date.

- `interval` is an integer.
  - `millisecond` or `ms`
  - `second` or `sec`
  - `minute` or `min`
  - `hour` or `hr`
  - `day`
  - `week`
  - `month`

    The interval can be "0", in which case the scheduled job only runs once. When the interval is higher than "0", then `repeatUnit` must also be defined.

- `repeatUnit` defines the unit of time to use in conjunction with the `interval`. The allowed values are:

These values are case-insensitive, meaning they can be entered in upper or lower case as you prefer.

Call properties

The call properties are where the actual task of the schedule is defined.

- `uri`

  This can take several forms:

  - An HTTP or HTTPS request: `<protocol>://<path>`
  - A Solr request: `solr://<collection>/…`

    For example, you could periodically issue a commit request to Solr. Or you could periodically run a query against a specific collection

  - A Fusion service request: `service://<serviceName>/<path>`

    The services available are stored in ZooKeeper. You can find them in the Admin UI under the "System" tab, or with a REST API call to the `/introspect` endpoint

- `method` is the HTTP method to use.
- `header` contains any additional required headers.
- `queryParams` contains any additional query parameters.

  For Solr requests, `queryParams` may be any valid query parameter for the specified URI.

- `entity` is the request body, if any.

10.2.2. How to define a scheduler job

There are two ways to define a scheduler job:

- In the Fusion UI, at **Applications** > **Scheduler**.

  See below for instructions.

- Using the Scheduler API.

  API examples are provided below.

Defining a scheduler job in the Fusion UI

1. Navigate to **Applications** > **Scheduler**.
2. Click **Add a Schedule**.
3. Enter the parameters for the scheduler job:
   - **Schedule Name** – Any arbitrary string (required)
   - **Service** – The endpoint and method for the service to run (required)

     Select the protocol:

     - `http://` or `https://`

    - `solr://{collection}/…`

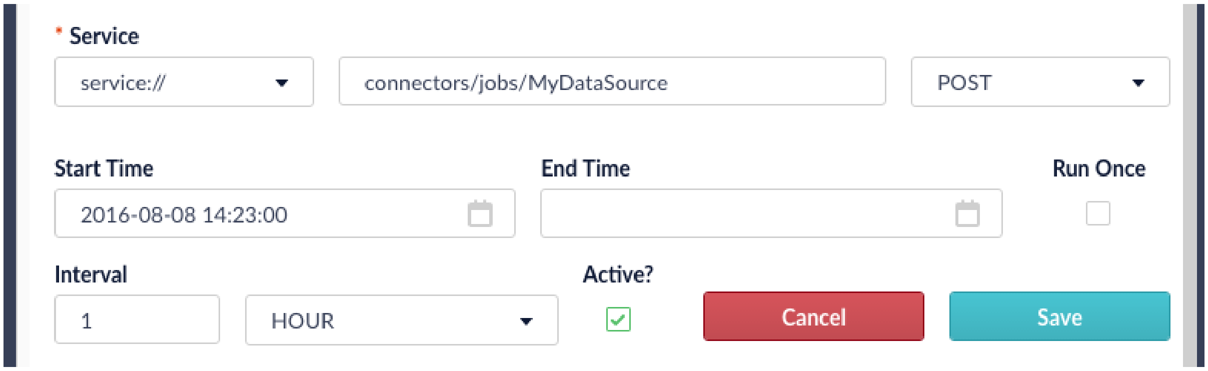
      A SolrCloud request.

    - `service://{serviceName}/{path}`

      A load-balanced Fusion service request.

  - **Start Time** – The date and time at which to begin running the first instance of this job
  - **End Time** – The date and time after which this job will be disabled
  - **Run Once** – To run the job at regular intervals, uncheck this option.
  - **Interval** – The interval at which to repeat this job
  - **Active?** – To disable the job, uncheck this option.

  Now your configuration should look something like this:



* Service: service:// | connectors/jobs/MyDataSource | POST

Start Time: 2016-08-08 14:23:00

End Time:

Run Once: ☐

Interval: 1 | HOUR

Active?: ☑

  In this example, the scheduler job crawls MyDataSource every hour. If you want this to happen every hour *on the hour*, you can set the start time to 11:00:00 (or any other hour). In the case of a crawl job like this one, you can check the job history by navigating to **Applications** > **Collections** > ***CollectionName*** > **Datasources** > ***DatasourceName*** > **Job History**.

4. Click **Save**.

API Examples

Each of these examples shows setting a schedule for an action in the system, using the Scheduler API. To see the results of a job, you will likely need to query the History API.

Issue a commit every 10 seconds

```
{"creatorType":"human", "creatorId":"me", "repeatUnit":"SECOND", "interval":10, "active":true,
"callParams":{"uri":"solr://myCollection/update", "method":"GET", "queryParams":{"stream.body":"<commit/>"}}}
```

In this example, we've defined the `callParams` with a URI for Solr that calls a collection named 'myCollection' and the 'update' `updateHandler`. The method is GET. The queryParams define the commit call for Solr. For timing, we've defined the job to run every 10 seconds.

Run a datasource every 20 minutes

```
{"creatorType":"human", "creatorId":"me", "repeatUnit":"MINUTE", "interval":20, "active":true,
"callParams":{"uri":"service://connectors/jobs/TwitterSearch", "method":"POST"}}
```

In this example, we've defined the callParams with a URI for Fusion that calls the TwitterSearch datasource job. The method is POST, which is the method to use when starting a crawl. There aren't any other properties needed to define the task. For timing, we've defined the job to run every 20 minutes.

Remove signals older than 1 month

```
{"creatorType":"human", "creatorId":"me", "repeatUnit":"MONTH", "interval":1, "active":true,
"callParams":{"uri":"solr://myCollection_signals/update",
"queryParams":"stream.body=<delete><query>timestamp_dt:[* TO NOW-1MONTH]</query></delete>", "method":"GET"}}
```

In this example, we're again calling Solr's 'update' updateHandler with a collection named 'myCollection_signals', which is the default location for signals. This time we've also defined queryParams to delete documents that match a date query that finds all documents older than 1 month old. For timing, we've set this to run once a month.

10.3. Messaging and Alerting

Fusion’s messaging services provide implementations to send messages and alerts to any application or device capable of displaying the supported message types. Read a primer on Fusion’s messaging services on our blog.

10.3.1. Supported Message Types

Fusion supports these types of messages and alerts:

- logging

  This service logs any message sent to it in the configured logger. You can use it in an index pipeline or a query pipeline.

- Slack

  Slack is a team messaging service with document integration and a focus on collaborative communication. See the Slack Index Stage and the Slack Query Stage.

- SMTP

  Email, via the Simple Mail Transfer Protocol. See the Email Index Stage and the Email Query Stage.

- PagerDuty

  Alerting and monitoring. See the PagerDuty Index Stage and the PagerDuty Query Stage.

10.3.2. Messaging Service Configuration

The Message Services as a whole can be configured via the Configurations API with these attributes:

Attribute	Description
`rateLimit`	The time, in milliseconds, to wait between sending messages on a per-second basis. This does not synchronize throttling between requests.
`storeAllMessages`	Boolean flag that indicates whether messages should be indexed and stored. By default, only scheduled messages are stored, as they need to be retrieved by the scheduler at a later time. Storing all messages can be useful for auditing the system, but it will have an impact on the system storage requirements.

Enabling Messaging Services

The logging service is enabled by default, but Slack, email, and PagerDuty messaging services must be explicitly enabled.

You can do this through the UI at **Applications** > **System** > **Messaging Services**, or through the Messaging API.

To see which messaging services are currently enabled:

```
curl -u <user>:<pass> http://localhost:8764/api/apollo/messaging/service
```

String Templates

String templates are libraries used for structured text generation outputs. They are a powerful way of doing variable substitution into a provided template using values contained in documents, requests, and contexts. String templates are made available in the Messaging Service System setup, where users can fill in these portions with document or query values from the working system.

See Messaging Services Templates for details.

10.3.3. Triggering Messages and Alerts

The messaging services can be invoked in several ways:

- Via the Scheduler API to send messages at designated intervals.
- Via the Messaging API.
- Through an index or query pipeline; see Messaging and Alerting Pipeline Stages below.

Note	By default, only scheduled messages are stored. To configure Fusion to store all messages, see Messaging Service Configuration above. The default collection for message storage is system_messages, which is created on startup.

Messaging and Alerting Pipeline Stages

The Messaging Service supports these pipeline stages:

- Email Message Query Stage
- Email Message Index Stage
- PagerDuty Message Query Stage
- PagerDuty Message Index Stage
- Slack Message Query Stage
- Slack Message Index Stage
- SetPropertyIndex Stage

The pipeline stages above send messages and alerts when specific conditions are met. Conditions can be specified using regular expressions, database lookups, and more. Any upstream stage can affect how Fusion behaves when a match occurs, so pay special attention to the order in which stages occur.

Additionally, the Set Property Index Stage allows conditions to be specified before messages are sent.

Setting Properties Upstream

Fusion includes two index pipeline stages that are useful for setting properties on indexed documents so that you can evaluate those properties in one of the messaging stages, either downstream in the index pipeline or in a query pipeline:

- The Set Property Index Stage can be used to set a property on a document, or a context, by evaluating one or more simple conditions. It is an index-only, conditional stage that allows the setting of properties without the use of JavaScript.

- The JavaScript Index Stage provides a more sophisticated means of setting properties upstream from the messaging stages.

Chapter 11. Access Control

11.1. User Authentication and Authorization

Fusion provides application security by restricting access to known users via a two-stage process consisting of:

- Authentication - users must sign on using a username and password.
- Authorization - each username is associated with one or more permissions which specify the Fusion UI components and REST API requests that user has access to. Permissions can be restricted to specific endpoints and path parameters. Roles are named sets of permissions which provide access to a specific function.

The access control component runs in the same process as the Fusion UI. It referred to as the "auth proxy" because it handles authentication and authorization for all requests to the Fusion REST API services.

All requests to Fusion must be authenticated, as described in section User Access Request Params.

11.2. User Account Admininistration

A Fusion Security Realm encapsulates a user database together with specific authentication and authorization mechanisms. This information is stored in ZooKeeper so that is it always available to all Fusion components across the deployment.

Fusion’s native security realm manages both authentication and authorization directly. All user information is stored in ZooKeeper: usernames, passwords, roles, and permissions. Stored passwords are encrypted using bcrypt, the strongest possible encryption algorithm available to all JDKs. Authentication consists of a password-hash comparison between the login password and the encrypted password. The native realm is the home of the Fusion admin user and is the default realm type.

Fusion can be configured to use the host domain’s security mechanism for user administration. The following configurations are possible:

- LDAP - Fusion stores a local user record in ZooKeeper. Authentication is performed by the LDAP server. LDAP group membership can be used to assign Fusion permissions.
- Kerberos - Fusion stores a local user record in ZooKeeper. SPNEGO is used for authentication via Kerberos.
- Kerberos authentication, LDAP authorization - Fusion stores a local user record in ZooKeeper. SPNEGO is used for authentication via Kerberos. LDAP group membership can be used to assign Fusion permissions.
- SAML - Fusion stores a local user record in ZooKeeper. The SAML 2.0 protocol is used to provide web-browser single sign-on.

See also:

- Authentication and Authorization APIs

11.3. Video tutorial

11.4. Users

All Fusion requests must come from a registered user.

11.4.1. Add Users

The first user who logs in becomes the user `admin`.

There are two approaches for adding users:

- **Manual** – Add users manually to a security realm that doesn't auto-create users.
- **Automatic** – For a security realm that uses an external authentication provider, Fusion can add users automatically. When creating the security realm, check **auto-create users**. Fusion creates a user the first time someone logs into Fusion.

When you add a new user manually, you must provide a unique username and valid password. All other information is optional. However, unless either roles or permissions are specified (or both), this user won't be able to do anything in Fusion.

If you specify API permissions in a user definition, those permissions *override* corresponding permissions defined in the user's roles. See Permissions for more information about how permissions supplied by multiple roles and by user definitions combine.

11.4.2. Manage Users in the Fusion UI

Only Fusion users with administrative privileges (for example, those who are assigned the built-in role `admin`) can manage users.

Manage users in the "USERS" panel of the Fusion UI "Access" component.

11.4.3. Manage Users via HTTP Requests to the Users API

See page Users API.

11.4.4. User Information

Fusion stores user information in Apache ZooKeeper.

Each User entry in ZooKeeper contains the following:

- `id`– A globally unique user ID (UUID), created by Fusion based on username, realm-name
- `realm-name`– The Fusion security realm name; the default is "native".
- `username`– The username string, which is unique within the specified security realm
- `permissions`– List of permissions that have been explicitly assigned to the user in the Fusion UI (in **Devops** > **Access Control**)
- `role-names`– List of roles assigned to the user in the Fusion UI (in **Devops** > **Access Control**)
- `created-at`– Timestamp; created by Fusion
- `updated-at`– Timestamp for the last edit; created by Fusion

The following JSON shows the ZooKeeper record for the Fusion admin user:

```
{
  "id":"57f539d2-3f53-4011-ad6f-257a3f00fc6b",
  "username":"admin",
  "realm-name":"native"
  "password-hash":"$2a$08$3I82umlXLPSshQIW6ngj.Or06DOVgDLGohGmCB9GC0yRtvy5Nfkn6",
  "permissions":[],
  "role-names":["admin"],
  "created-at":"2016-01-28T00:00:18Z"
}
```

The following JSON shows the ZooKeeper record for a user entry managed by Fusion:

```
{
  "id":"ae9b345a-79e2-4e6d-8620-e6ed4ed2cc16",
  "username":"firstname.lastname",
  "realm-name":"lwLDAP",
  "permissions":[{"path":"collections/**","methods":["GET"]}],
  "role-names":[],
  "created-at":"2016-04-01T21:17:36Z"
  "updated-at":"2016-04-01T21:42:15Z",
}
```

11.5. Roles

Roles are named sets of permissions that encapsulate the permissions needed for different kinds of users. Permissions grant users access to subsets of Fusion functionality. A role can specify UI permissions, API permissions, or both:

- UI permissions grant users access to parts of the Fusion UI
- API permissions grant users access to specific API commands for specific REST API endpoints.

See Permissions for information about how permissions supplied by multiple roles and by user definitions combine.

11.5.1. Where You Specify Roles

You can specify which roles to apply for a user in one or more of these places:

- **Security realm (directly)** – Under the heading **Roles**, specify the roles to always apply to all users in the security realm.
- **Security realm (from a group/role mapping)** – Security realms of types `ldap` and `trusted-http` can provide a list of groups to which the user belongs. The security realm can map the group names to role names.
- **User definition** – A user definition can specify roles for the user. These roles don't override the other roles. They are added to the other roles.

11.5.2. Default Roles

At initial startup, Fusion creates a set of default roles for common types of users.

admin

The admin role is the the equivalent to the Unix `root` or superuser. It allows full access to all Fusion services:

```
GET,POST,PUT,DELETE,PATCH,HEAD:/**
```

collection-admin

The collection-admin role allows a user to fully manage all Fusion collections.

```
GET:/query-pipelines/**
GET:/query-stages/**
GET:/connectors/**
GET:/reports/**
POST:/reports/**
GET,POST,PUT,DELETE,PATCH:/collections/**
```

search

The search role gives a user access to the Fusion UI Search component and allows search over all Fusion collections.

```
GET:/query-pipelines/*/collections/*/select
GET:/query-pipelines
GET:/solr/*/schema
GET:/prefs/apps/search/*
GET:/collections/**
GET:/solr/*/admin/luke
```

ui-user

The ui-user role allows a user to use the Fusion UI for system monitoring and reporting.

```
GET:/system/**
GET:/history/**
GET:/connectors/**
PUT:/usage/**
GET:/reports/**
GET:/solrAdmin/**
GET:/nodes/**
POST:/reports/**
GET:/collections/**
PATCH:/users/{id}:id=#ID
GET:/searchCluster/**
```

Note	The permission `PATCH:/users/{id}:id=#ID` uses the variable value `#ID` as a placeholder for the currently logged-in user ID. It is included so the Fusion UI "change password" feature is available to native realm users.

11.5.3. Role Information

Fusion stores role information in Apache ZooKeeper. Each role in a ZooKeeper entry contains the following:

- `id`– ID string, created by Fusion
- `name`– Role name string
- `desc`– Text description; optional
- `permissions`– A list of Fusion permission specifications
- `ui-permisions`– A list of names of Fusion UI components
- `created-at`– Timestamp; created by Fusion
- `updated-at`– Timestamp for last edit; created by Fusion

11.5.4. Manage Roles

Only Fusion users with admin privileges can manage roles.

Restricting access to a subset of Fusion's functionality requires several narrowly defined permissions. Path variables can be used to designate specific collections. As an example, it's possible to define a role which allows read-only access to Fusion dashboards for a specific collection:

- `GET:/solr/{id}/*:id=test` – Read-only access to the collection "test"

- `GET:/solr/{id}/admin/luke:id=test` – Also read-only access
- `GET:/solr/system_banana/*` – Read-only access to dashboards
- `GET:/collections/system_banana` – Read-only access to the collection where dashboard definitions are stored

Manage Roles in the Fusion UI

Manage roles in "ROLES" panel of the Fusion UI "Access" component.

To create a new role from the Fusion Admin UI, first you choose a unique role name, then edit the set of permissions. Specify API permissions one per line in the **Permissions** input box. There is a separate list of checkboxes which allow access to the Fusion UI components. If users who are assigned this role require access to the Fusion UI, then you must specify UI permissions in addition to REST API permissions.

Manage Roles via HTTP Requests to the Roles API

See page Roles API.

11.6. Permissions

Permissions determine what a user can do in Fusion. There are two kinds of permissions:

- **UI permissions** – Control which parts of the Fusion UI a user can access. These parts show up in menus and the user can view them. But the ability to *use* the functionality depends on API permissions. (The UI uses the API.)
- **API permissions** – Control which requests a user can submit to which REST API endpoints.

Fusion uses permissions for authorization as follows:

- UI permissions are positive (permission needs to be given) and additive (the user has the sum of all specified permissions. This is true of roles specified in a user definition, roles specified in a security realm, and roles determined dynamically based on groups in an LDAP authentication provider.
- API permissions specified in roles are positive (permission needs to be given) and additive (the user has the sum of all specified permissions; that is, for a specific endpoint, the most permissive permissions are used). This is true of roles specified in a user definition, roles specified in a security realm, and roles determined dynamically based on groups in an LDAP authentication provider.
- API permissions specified in the role(s) but not in the user definition are used.
- If an API permission for a specific endpoint is specified in both one or more roles *and* in the user definition, then the permissions in the user definition are used, *overriding* the permissions in the role(s). Use permissions in user definitions to give specific users permissions that are less permissive than the permissions for their role(s). Alternatively, you could define less permissive roles.

11.6.1. Specify UI Permissions

Specify UI permissions in roles.

11.6.2. Specify API Permissions

A Fusion API permission denotes an allowed request to a Fusion REST API endpoint or endpoints. A permissions specification consists of:

- HTTP request method or methods allowed. Multiple HTTP methods are separated by commas.
- REST API services endpoint, which can contain wildcards or named variables. All calls to the REST API start with "api/apollo", followed by the service name and any methods and parameters. The permissions specification includes everything following "api/apollo". The endpoint can include wildcards.

Wildcards make it easy to grant broad access to Fusion services. The wildcard symbol '*' matches all possible values for a single path segment and two wildcards match all possible values for any number of path segments. Granting access to a subset of Fusion's functionality requires a list of narrowly defined permissions.

A path segment can be a named variable enclosed in curly braces: `{variable-name}`. Variables are used when a wildcard would be too permissive and a single path segment too restrictive.

- Optionally, the allowed values for any named variables in the endpoint. The variable specification component specifies the restricted value or values for all named variables in the path. Each specification consists of the variable name, followed by "=" (the equals sign), followed by one or more values which are separated by commas. If the endpoint specification has multiple variable, the semi-colon character ";" is used as the separator between parameter specifications.

Permissions specifications are coded up as a string using the colon character ":" as the separator between the permission elements.

Here are some examples of permissions specifications:

- GET:/query-pipelines//**collections**//select – Search access to any Fusion collection.
- `GET,PUT:/collections/Collection345/synonyms/\**` – Permission to edit synonyms for the collection named "Collection345"
- `GET:/collections/{id}:id=Collection345,Collection346` – Read access to collections named "Collection345" and "Collection346"

In ZooKeeper, both User and Roles entries contain a list of Permission specifications. A Permission entry has three attributes: "methods", "path", and "params".

11.6.3. Example Permissions Set

Wildcards make it easy to give wide access to Fusion services. The permissions for the admin user can be written in a single line:

```
GET,POST,PUT,DELETE,PATCH,HEAD:/**
```

To restrict access to a single collection and a single Fusion facility requires a set of narrowly defined permissions. For example, the following set of permissions allows a user to run Fusion’s analytics dashboards over a collection named "test":

- `GET:/solr/{id}/*:id=test`– Read-only access to collection named "test"
- `GET:/solr/{id}/admin/luke:id=test`– Dashboards require read-only access to Solr utility luke to compile collection metrics.
- `GET:/solr/system_banana/*`– Read-only access to dashboards
- `GET:/collections/system_banana`– Read-only access to the collection where dashboard definitions are stored

Read-only access to the dashboard definitions collection means that the user cannot save the configured dashboard back to Fusion.

The following JSON shows how Fusion stores this list of permissions in ZooKeeper:

```
"permissions":[
  {"params": {"id":["mdb1"]},
   "path":"/solr/{id}/*",
   "methods":["GET"]},
  {"params":{"id":["mdb1"]},
   "path":"/solr/{id}/admin/luke",
   "methods":["GET"]},
  {"path":"/solr/system_banana/*",
   "methods":["GET"]},
  {"path":"/collections/system_banana",
   "methods":["GET"]}
  ]
```

11.7. Security Realms

Fusion uses *security realms* to authenticate users of the Fusion UI. Each user has an assigned security realm, which the user must choose when logging in. Choosing a different realm results in an authentication failure.

A security realm also provides a list of roles:

- The list always includes the role(s) that are specified in the security realm.
- (Optional) The security realm can reference one or more Fusion roles and/or get groups to which the user belongs from an external directory service that is the authentication provider. Fusion maps the group names to role names and adds these roles to the user’s list of roles.

Note: Fusion doesn’t use permissions from the LDAP for authorization of UI access or API requests. Fusion only obtains group names from the LDAP (optionally), which you can configure the security realm to map to role names.

Requests to the Fusion REST API must specify a security realm for per-request authentication, unless a session cookie is used (which contains information about the security realm).

Fusion authorizes requested operations based on API permissions specified for the user and for the user’s role(s). Fusion considers the role(s) specified in the user definition and in the security realm. Fusion creates a list of roles when a session is created, that is, when a user logs in or when the Sessions REST API creates a session. Authorization based on permissions is at request time.

You can define multiple security realms for a Fusion instance. This lets you give different sets of users different levels of access to specific Fusion collections.

11.7.1. Security Realm Types

When you create a security realm, you can choose among the following security realm types:

Native

Fusion has a single preconfigured security realm named *native*. The admin user is in the native realm. and is the default realm. The native realm also provides a fallback mechanism in case of LDAP server or communication failure.

This realm is required to bootstrap Fusion. Because all requests to Fusion require authentication and authorization, on initial startup you must access the Fusion UI to set the admin password. After Fusion has a valid admin password, it creates the admin account in the Fusion native realm.

For the native realm, Fusion manages all authentication and permissions information directly.

You can create Fusion user accounts and manage them using either the Fusion UI or the User API.

Stored passwords are encrypted using bcrypt, the strongest possible encryption algorithm available to all JDKs.

SSO Trusted HTTP

The "SSO Trusted HTTP" realm type (`trusted-http` in the REST API) is useful in single sign-on (SSO) environments.

If the SSO environment contains groups that make sense regarding partitioning Fusion functionality for Fusion users (that is, giving Fusion users different UI and API permissions based on the SSO groups), then you can configure an SSO Trusted HTTP security realm to return a list of group names, and then map the groups to Fusion roles in the security realm definition.

See Configuring Fusion for SSO.

Kerberos

In the case where a host domain uses Kerberos for authentication and LDAP for authorization, Fusion can be configured to do the same, by configuring a realm of type "LDAP" and then specifying Kerberos as the authentication mechanism.

Fusion stores a local user record in ZooKeeper and a mapping to the Kerberos principal.

SPNEGO is used for authentication via Kerberos.

See Configuring Fusion for Kerberos.

LDAP

You can use an LDAP as an authentication provider for Fusion. If the LDAP contains groups that make sense regarding partitioning Fusion functionality for Fusion users (that is, giving Fusion users different UI and API permissions based on the LDAP-group memberships of LDAP users), then you can configure an LDAP security realm to search for LDAP groups and to map the LDAP groups to Fusion roles.

Fusion stores a local user record in ZooKeeper, and authentication is performed by the LDAP server. User accounts can be managed by Fusion, or created automatically, in which case the Fusion user ID maps directly to the LDAP Distinguished Name (DN). Fusion permissions can be assigned automatically based on LDAP group membership.

See Configuring Fusion for LDAP.

SAML

Fusion stores a local user record in ZooKeeper and the URL and information about the SAML Identity Provider. The SAML 2.0 protocol is used to provide web browser single sign-on.

See Configuring Fusion for SAML.

11.7.2. Manage Security Realms

Only Fusion users with admin privileges can manage security realms. There are two ways to manage security realms:

In the Fusion UI

Navigate to **Applications** > **Access Control** > **Security Realms**.

Using the Realms API

Use the `http://localhost:8764/api/realm-configs/` endpoint to manage security realms. See the Realms API reference for details. In production environments, use port 8765.

11.8. Configuring Fusion for SSO

The "SSO Trusted HTTP" realm type (`trusted-http` in the REST API) is useful in single sign-on (SSO) environments.

If SSO is already set up in your environment, user identities and group information can be sent to Fusion through HTTP headers (REMOTE_USER, for example). The SSO Trusted HTTP realm type provides the configuration options for integrating this into Fusion’s authentication systems. It also supports allowing access to only a set of known client IPs, and mapping groups to Fusion roles.

Use the Realms API to configure this realm type:

```
curl -u user:pass -H 'content-type:application/json' -X POST :3000/api/realm-configs -d @./realm-config.json
```

Below is a sample configuration:

```
{"id":"test-id",
 "enabled":true,
 "name":"sso-test",
 "realmType":"trusted-http",
 "config":{"identityKey":"REMOTE_USER",
           "groups": {"key":"GROUPS",
                      "delimiter":"|",
                       "roleMapping": [["a","admin"], ["b","foo"]]},
           "allowedIps":["127.0.0.1", "0:0:0:0:0:0:0:1", "localhost"]}}
```

`identityKey`	The name of an HTTP headers entry. If this key is found in the headers map, it used as the identity of the client (username, for example). The `X-FORWARDED-FOR` header is inspected for this client IP first; the value is split on comma, and the first entry is taken. This would normally be used in cases where the client was forwarded to Fusion through one or more external proxy servers. If the `X-FORWARDED-FOR` header is not present in the request, the `REMOTE-ADDR` header value is used instead.
`groups`	Configuration keys for auth groups: • `key` + The name of an HTTP header, used as the source of group names. • `delimiter` + The character used to split the value (defaults to comma). • `roleMapping` + A set of 2-tuples, used for mapping the external group values to Fusion Roles.

`allowedIps`	Allow access to only a set of known client IPs. When this property is defined and the client IP is not included in it, the realm logic return a 401.

11.9. Configuring Fusion for LDAP

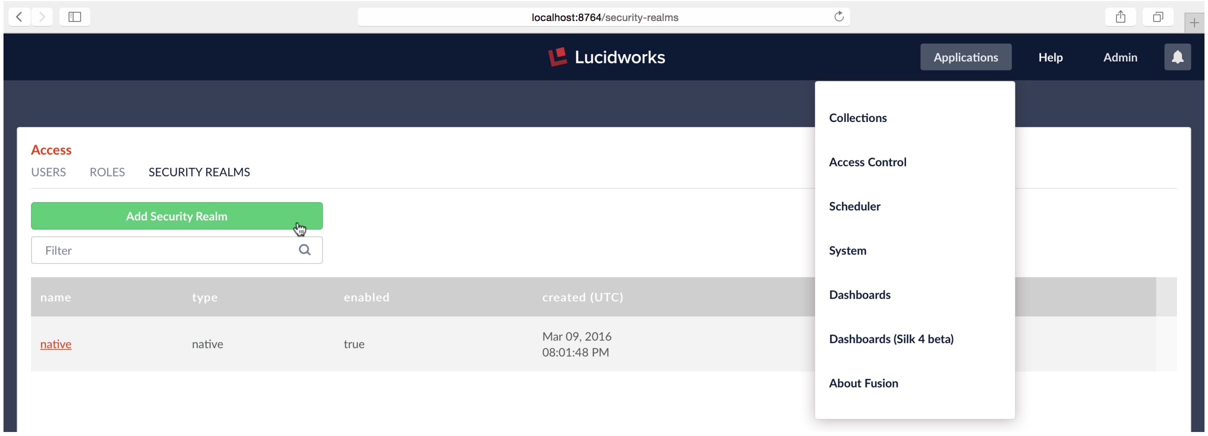
You can create security realms that use external LDAP servers for authentication. Optionally, Fusion can search in the LDAP for groups to which a user belongs, and then map those groups to Fusion roles. Fusion performs authorization using permissions stored in Fusion users and Fusion roles.

Note: Fusion *doesn't* use permissions from the LDAP for authorization of UI access or API requests. It only obtains group names (optionally), which are mapped to role names. If an Active Directory Security Query Trimming Stage is used, then directory-service permissions *are* used for trimming. If a connector supports security trimming, then connector permissions *are* used for trimming.

To configure Fusion to use an external LDAP as an authentication provider, you'll need to get information about the LDAP server(s) running on your system, either from your system or your sysadmin.

11.9.1. Fusion Configuration for an LDAP Realm

To configure an LDAP security realm from the Fusion UI, you must be logged in as a user with admin-level permissions. From the "Applications" menu, menu item "Access Control", panel "Security Realms", click on the "Add Security Realm" button:

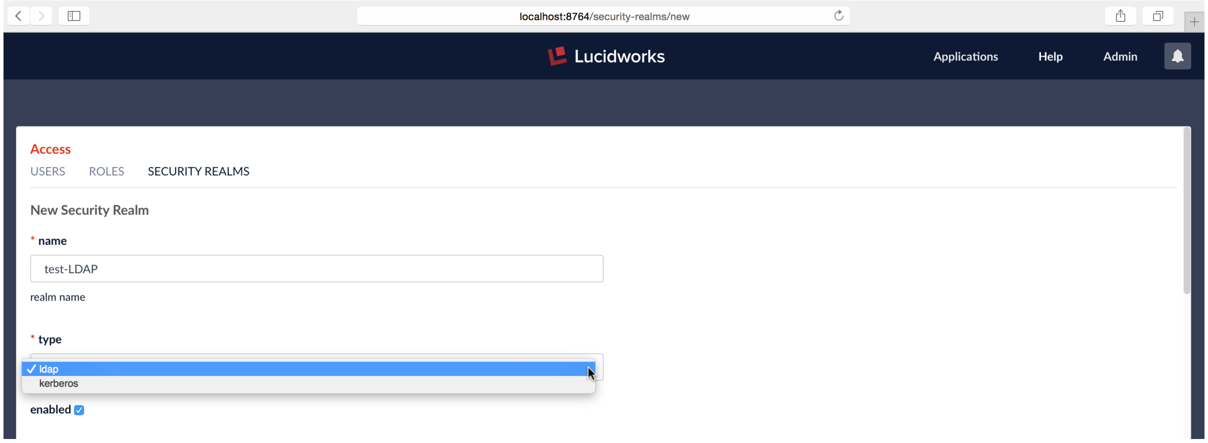


name	type	enabled	created (UTC)
native	native	true	Mar 09, 2016 08:01:48 PM

This opens an editor panel for a new Security Realm, containing controls and inputs for all required and optional configuration information.

Required Configuration Step One: Name and Type

The first step in setting up an LDAP security realm is filling out the required information at the top of the realm config panel:



New Security Realm

* name

test-LDAP

realm name

* type

✓ ldap
kerberos

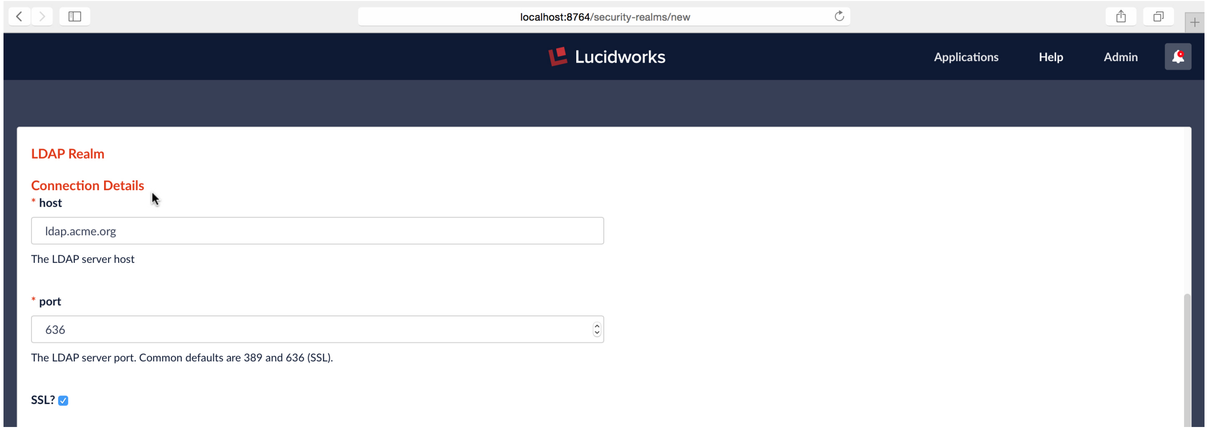
enabled ☑

The first three required configuration items are:

- name - must be unique, should be descriptive yet short.
- type - choose "LDAP" from the pulldown menu.
- "enabled" checkbox - default is true (i.e., the box is checked). The "enabled" setting controls whether or not Fusion allows user logins for this security realm.
- "auto-create users" checkbox - default is true (i.e., the box is checked). This controls whether or not a user account is created automatically upon initial authentication. If false, the user accounts can only be created by a Fusion user with admin privileges.

Required Configuration Step Two: Server and Port

The hostname and port of the LDAP server are required, along with whether or not the server is running over SSL.



LDAP Realm

Connection Details

* host

ldap.acme.org

The LDAP server host

* port

636

The LDAP server port. Common defaults are 389 and 636 (SSL).

SSL? ☑

Required Configuration Step Three: Authentication Method and DN Templates

There are three possible authentication methods:

- Bind - LDAP authentication is carried out via a single "Bind" operation.
- Search - LDAP authentication is carried out indirectly via a Search operation followed by a Bind operation.
- Kerberos - Kerberos authenticates Fusion and an LDAP Search operation is carried out to find group-level authorizations.

The Bind authentication method is used when the Fusion login username matches a part of the LDAP DN. The rest of the

LDAP DN is specified in the "DN Template" configuration entry, which uses a single pair of curly braces ({}) as a placeholder for the value of the Fusion username.

The Search authentication method is used when the username used for Fusion login *doesn't* match a part of the LDAP DN. The search request returns a valid user DN, which is used together with the user password for authentication via a Bind request.

The Search authentication method is generally required when working with Microsoft Active Directory servers. In this case, you need to know the username and password of *some* user who has sufficient privileges to query the LDAP server for user and group memberships; this user doesn't have to be *the* superuser. In addition to a privileged user DN and password, the Search authentication method requires crafting a search request. There are two parts to the request: the first part is the base DN of the LDAP directory tree which contains user account objects. The second part of the request is a Search Filter object which restricts the results to a matching subset of the information.

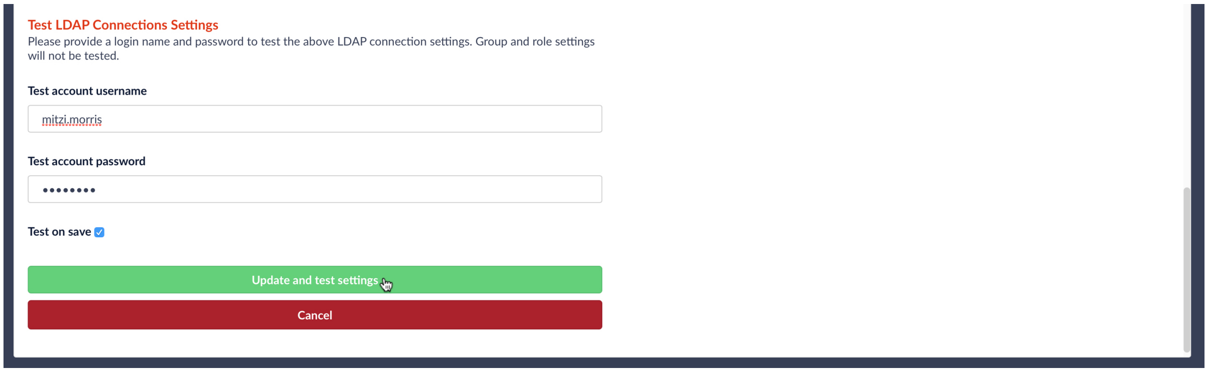
Optional Configuration: Roles and Groups Mappings

A Fusion role is a bundle of permissions tailored to the access needs of different kinds of users. Access to services and data for LDAP-managed users is controlled by mappings from LDAP users and groups to Fusion roles.

Roles can be assigned globally or restricted to specific LDAP groups. The security realm configuration panel contains a list of all Fusion roles with a checkbox for each, used to assign that role to all users in that realm. LDAP group names can be mapped directly to specific Fusion roles and LDAP group search and filter queries can also be used to map kinds of LDAP users to specific Fusion roles.

Testing the Configured Connection

The last part of the form allows you to test the LDAP realm config using a valid username and password:



Test LDAP Connections Settings

Please provide a login name and password to test the above LDAP connection settings. Group and role settings will not be tested.

Test account username

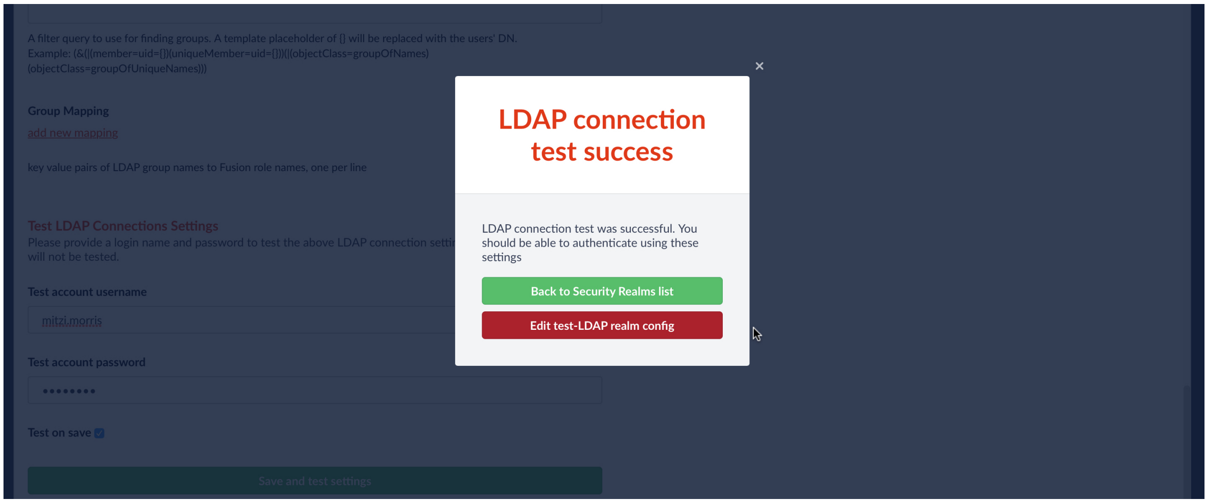
mitzi.morris

Test account password

••••••••

Test on save ☑

When the "Update and test settings" button is clicked, the username from the form is turned into a DN according to the DN template, and a Bind operation request is sent to the configured LDAP server. Fusion reports whether or not authentication was successful:



A filter query to use for finding groups. A template placeholder of {} will be replaced with the users' DN.
Example: (&(|(member=uid={})(uniqueMember=uid={}))(|(objectClass=groupOfNames)
(objectClass=groupOfUniqueNames)))

Group Mapping

add new mapping

key value pairs of LDAP group names to Fusion role names, one per line

Test LDAP Connections Settings

Please provide a login name and password to test the above LDAP connection setti
will not be tested.

Test account username

mitzi.morris

Test account password

••••••••

Test on save ☑

Save and test settings

LDAP connection test success

LDAP connection test was successful. You should be able to authenticate using these settings

Back to Security Realms list

Edit test-LDAP realm config

11.9.2. Basic LDAP Concepts and Terminology

The LDAP protocol is used to share information about users, systems, networks, and services between servers on the internet. LDAP servers are used as a central store for usernames, passwords, and user and group permissions. Applications and services use the LDAP protocol to send user login and password information to the LDAP server. The server performs name lookup and password validation. LDAP servers also store Access Control Lists (ACLs) for file and directory objects which specify the users and groups and kinds of access allowed for those objects.

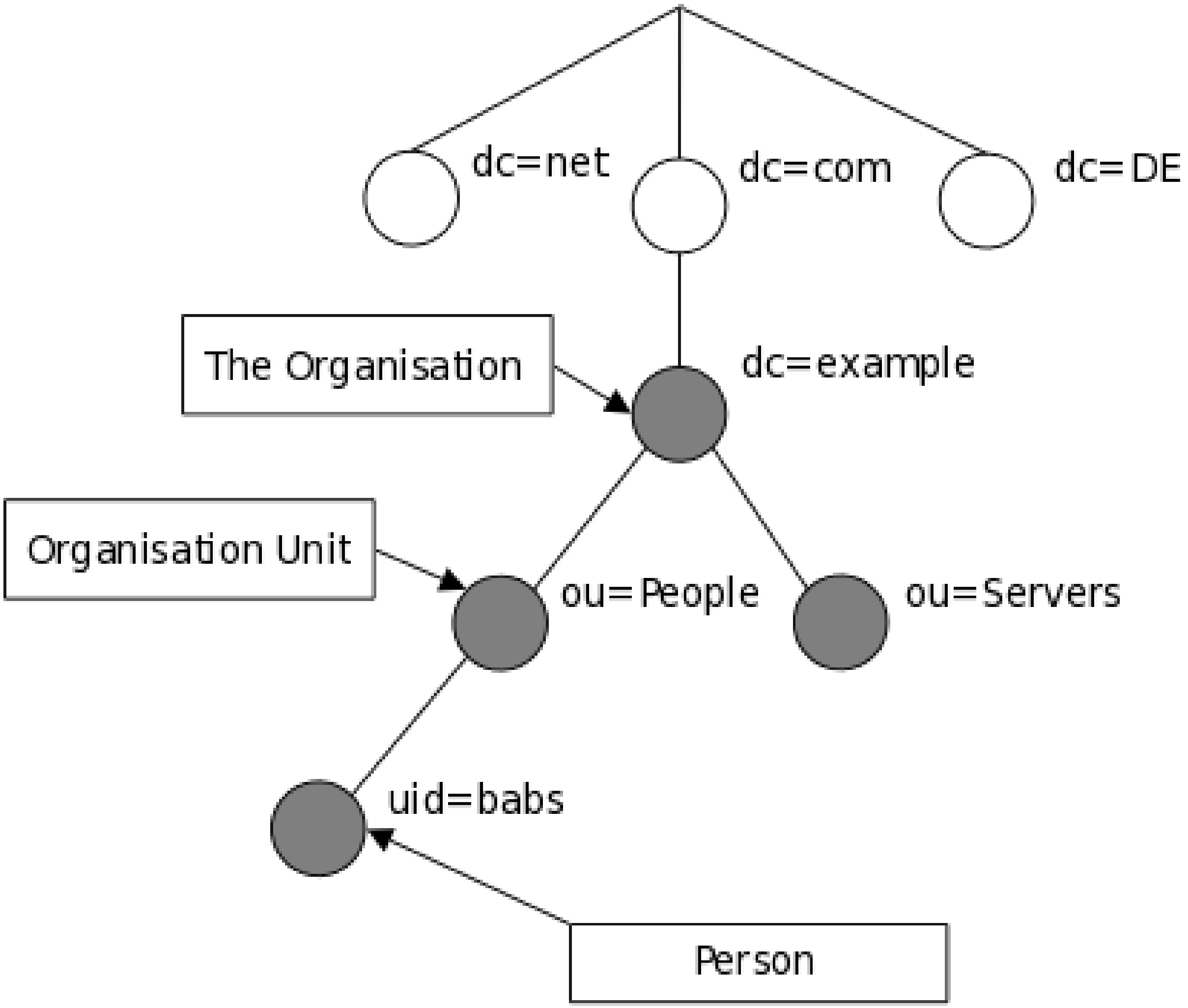
LDAP is an open standard protocol and there are many commercial and open-source LDAP servers available. Microsoft environments generally use Active Directory. Unix servers use AD or other LDAP systems such as OpenLDAP, although many Unix systems don't use LDAP at all. To configure Fusion for LDAP, you'll need to get information about the LDAP server(s) running on your system either from your sysadmin or via system utilities.

Directories and Distinguished Names

An LDAP information store is a Directory Information Tree (DIT). The tree is composed of entry nodes; each node has a single parent and zero or more child nodes. Every node must have at least one attribute which uniquely distinguishes it from its siblings which is used as the node's Relative Distinguished Name (RDN). A node's Distinguished Name (DN) is a globally unique identifier.

The string representation of a DN is specified in RFC 4514. It consists of the node's RDN followed by a comma, followed by the parent node's DN. The string representation of the RDN is the attribute-value pair name, connected by an equals ("=") sign. This recursive definition means that the DN of a node is composed by working from the node back through its parent and ancestor nodes up to the root node.

Here is a small example of a DIT:

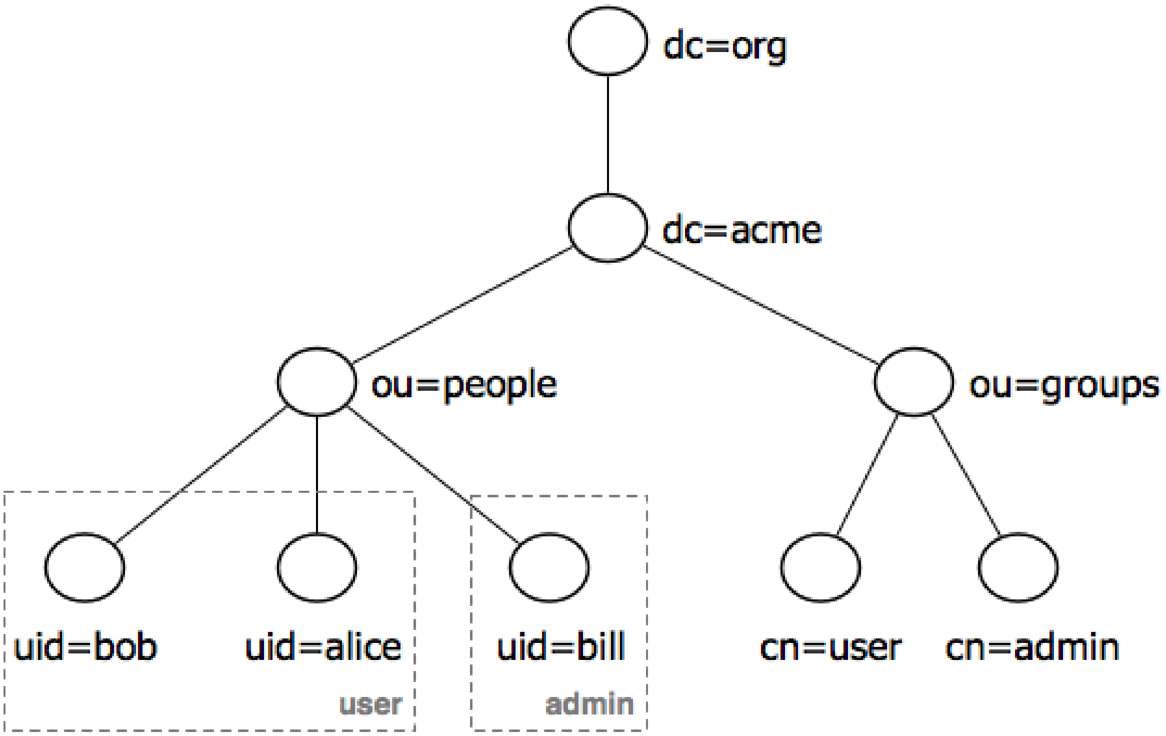


The person entry in this tree has the DN: "uid=babs, ou=people, dc=example, dc=com".

Attribute names include many short strings based on English words and abbreviations, e.g.:

Name	Description
cn	commonName
dc	domainComponent
mail	email address
ou	organizationalUnitName
sn	surname
uid	userId

LDAP entry attributes can refer to other LDAP entries by using the DN of the entry as value of that attribute. The following example of a directory which contains user and groups information shows how this works:



This tree contains two organizational units: "ou=people" and "ou=groups". The children of the "group" organizational unit are specific named groups, just as the child nodes of organization unit "people" are specific users. There are three user entries with RDNs "uid=bob", "uid=alice", "uid=bill" and two groups with RDNs "cn=user" and "cn=admin". The dotted lines and group labels around the person nodes indicates group membership. This relationship is declared on the groups nodes by adding an attributes named "member" whose value is a users DN. In the LDAP data interchange format (LDIF), this is written:

```
cn=user,ou=groups,dc=acme,dc=org
    member: uid=bob,ou=people,dc=acme,dc=org
    member: uid=alice,ou=people,dc=acme,dc=org
cn=admin,ou=groups,dc=acme,dc=org
    member: uid=bill,ou=people,dc=acme,dc=org
```

See the Wikipedia's LDAP entry for details.

LDAP Protocol Operations

For authentication purposes, Fusion sends Bind operation requests to the LDAP server. The Bind operation authenticates clients (and the users or applications behind them) to the directory server, establishes authorization identity used for subsequent operations on that connection, and specifies the LDAP protocol version that the client will use.

Depending on the way that the host system uses LDAP to store login information about users and groups, it may be necessary to send Search operation requests to the LDAP server as well. The Search operation retrieves partial or complete copies of entries matching a given set of criteria.

LDAP filters specify which entries should be returned. These are specified using prefix notation. Boolean operators are "&" for logical AND, "|" for logical OR, e.g., "A AND B" is written "(&(A)(B))". To tune and test search filters for a Unix-based LDAP system, see the ldapsearch command line utility documentation. For Active Directory systems, see AD

Syntax Filters.

11.10. Configuring Fusion for Kerberos

To configure the Fusion UI service to use Kerberos for user authentication, you must create a Kerberos security realm.

Kerberos is a system that provides authenticated access for users and services on a network. Instead of sending passwords in plaintext over the network, encrypted passwords are used to generate time-sensitive tickets that are used for authentication. SPNEGO provides a mechanism for extending Kerberos to Web applications through the standard HTTP protocol.

Kerberos uses symmetric-key cryptography and a trusted third party called a Key Distribution Center (KDC) to authenticate users to a suite of network services. (By users we mean both end users and client programs). The computers managed by that KDC and any secondary KDCs constitute a realm. When a user authenticates to the KDC, the KDC sends a set of credentials (a ticket) specific to that session back to the user's machine. Kerberos-aware services use the ticket on the user's machine for authentication instead of requiring sign-on with a password. Because tickets are used rather than passwords, this provides the convenience of Single Sign-On (SSO) in addition to security.

A Kerberized process is one that has been configured so it can get tickets from a KDC and negotiate with Kerberos-aware services. When a user sends an HTTP request, Fusion tries to authenticate using the Kerberos/SPNEGO protocol. If the request was sent from a browser, Fusion doesn't display the initial sign-on panel; instead on login, the user sees the main Fusion collections panel.

To Kerberize Fusion, you must:

- Configure the Kerberos client on the server that the Fusion UI service will be running on so that it can talk to the KDC (section Configuring the Kerberos client below).
- Configure the security realm of the Fusion UI (section Configuring Fusion Authentication for Kerberos Realm below).
- Depending on the encryption used by the KDC, you may also need to install additional Java security libraries into the Fusion distribution. These are freely available from Oracle, see download and installation instructions below.

To do this, you need following information, which you can get from your sysadmin:

- Kerberos realm name - in most cases, this is your domain name in upper case.
- KDC name - usually "kerberos." + your domain name.
- Kerberos principal name and password. A principal is a unique identity to which Kerberos can assign tickets. When the entity is a client program, this is called the Service Principal name.
- A keytab file which holds the encrypted credentials.

The usual scenario in an enterprise organization is to have a Kerberos admin create a service principal with a random key password. Then, the admin generates a keytab, which is then used for Fusion service principal authentication.

The Kerberos commands needed for configuration and testing are:

- kinit - obtain and cache a ticket from the KDC (i.e., domain login)
- kdestroy - destroys credentials (i.e., domain logout)
- klist - lists cached credentials
- ktutil - create or add credentials to a keytab file

If your browser is not already configured to use the Kerberos/SPNEGO, you will need to do so in order to test the Fusion configuration.

11.10.1. Configuring the Kerberos client

Step 1: Edit the Kerberos configuration file.

To configure your local Kerberos client so that it can talk to the Kerberized server, you must edit Kerberos configuration file named `krb5.conf`. On most Unix systems, this file is located at `/etc/krb5.conf`.

This file contains Kerberos configuration information, including the locations of KDCs and admin servers for the Kerberos realms of interest, defaults for the current realm and for Kerberos applications, and mappings of hostnames onto Kerberos realms.

If your organization realm name is "MYORG.ORG", and your KDC server is named "kerberos.myorg.org", then you edit two entries. The first entry is `libdefaults`. Set MYORG.ORG as the default realm:

```
[libdefaults]
  default_realm = MYORG.ORG
```

The second entry is `realms`. Add MYORG.ORG as a realm:

```
[realms]
  MYORG.ORG = {
    kdc = kerberos.myrealm.com
  }
```

For example, for realm LUCIDWORKS.IO, the krb5.conf file is just like the above example, except that instead of "myorg.org" we specify "lucidworks.io".

Step 2: Authenticate to Kerberos

The command `kinit` is the Kerberos authentication command. To get started, you authenticate to Kerberos using the Kerberos principal name and password (which you may need to obtain from your sysadmin). For this example, the principal name is "prince".

```
> kinit prince
```

The `kinit` command prompts for a password. Successful authentication is silent. Unsuccessful authentication results in an error message.

The command `klist` shows all cached Kerberos credentials. To check that you’ve successfully authenticated, run this command:

```
> klist
```

Output should be in this form, but with your data:

```
Credentials cache: API:C980F9F5-415C-4A3E-9C67-883C7D5FFFBE
        Principal: prince@MYORG.ORG

  Issued                Expires               Principal
May  6 15:14:55 2015  May  7 01:12:47 2015  krbtgt/MYORG.ORG@MYORG.ORG
```

11.10.2. The Service Principal Keytab file

The usual scenario in an enterprise organization is to have a Kerberos admin create a service principal with a random key password. Then, the admin generates a keytab, which is then used for Fusion service principal authentication. If you are your own Kerberos admin, then you will need to create this file for yourself.

Step 3. Create a Keytab file

The command `ktutil` creates the service principal keytab file which holds the encrypted credentials that the Fusion UI Proxy will use for Kerberos authentication. In order to generate the keytab file, you must have a set of cached credentials, therefore, first run the `kinit` command (step 2).

From the command line, run the command `ktutil`. You must enter your password twice.

```
> ktutil -k http-myrealm.org.keytab add -p HTTP/myrealm.org@MYORG.ORG -e aes256-cts-hmac-sha1-96 -V 0
```

The -k argument specifies the name of the keytab file which will be created or updated. The command "add" takes the following argument flags:

- "-p" : service principal, format <service>/<fully.qualified.domain>@REALM
- "-e" : encryption type. Depending on the encryption type, you may need to download additional Java security libraries for strong encryption.
- "-V" : key version number (kvno). Key version numbers are used in the Kerberos V5 protocol to distinguish between different keys in the same domain.

If successful, this command creates a keytab file called "http-myrealm.org.keytab". Note the directory you’re in - you’ll need this full path when creating the Proxy realm-config later.

Step 4. Test the Keytab file

The location of this keytab file will be used to configure UI Proxy configuration. Before configuring the Fusion UI Proxy, you should check that the keytab file is valid. Testing the keytab requires the following sequence of steps:

- Clear any existing credentials via command `kdestroy`.
- Log in using the keytab as an argument to the command `kinit -kt <keytab file> <principal>`, where `<principal>` is the name of a principal within the keytab file.
- Examine your credentials via command `klist`.
- Clear credentials via command `kdestroy`, which removes any existing credentials, effectively logging you out of Kerberos.

To remove cached credentials, use the `kdestroy` command. This command succeeds silently. To check that credentials have been removed, re-run the klist command:

```
> kdestroy
> klist
klist: krb5_cc_get_principal: No credentials cache file found
```

Use the keytab to login as the service principal, without being prompted for a password:

```
> kinit -t http-myrealm.org.keytab HTTP/myrealm.org
```

Examine your credentials via the command `klist`. The output should be similar to this:

```
Credentials cache: API:51D488FF-5CD9-4E16-98FA-B47743F5B4ED
        Principal: HTTP/myrealm.org@MYORG.ORG

  Issued                Expires               Principal
Apr  1 09:15:02 2015  Apr  1 19:13:42 2015  krbtgt/MYORG.ORG@MYORG.ORG
```

Logout again with `kdestroy`.

11.10.3. Configuring Fusion Authentication for Kerberos Realm

Once you have tested both the user and service principal logins, you must create the service principal realm-config in the Fusion Authentication Proxy. This allows the Proxy to authenticate to Kerberos as the service principal, without a password.

Step 5. Configure the Fusion Realm

Fusion security realms can be configured either via the Fusion UI Admin tool or the Fusion REST API. The advantage of using the Fusion UI Admin Tool is that a single panel's worth of configuration requires a series of calls to the REST API. It is important to understand the set of configuration properties collected by the Fusion UI and how they are used by the REST API.

A security realm for Kerberos has the following properties:

- name : unique string identifier
- realmType : "kerberos"
- enabled : whether or not the realm is available for users to use with system authentication
- config : this property is required for the realm type "kerberos". It takes two key-value pairs:
    - principal : the principal service name
    - keytab : this must be the full path to the keytab file.

To configure Fusion via the Fusion UI Admin tool, you should be logged in to Fusion as "admin" or as a user who has super-admin privileges.

From the Fusion Admin tool, choose the "Access" control from the left hand side nav bar, and go to the Access tab "Security Realms" (URL: <server>:<port>/admin/security-realms).

Click on the "Add Security Realm" button. On the New Security Realm form, choose type "kerberos" from the pulldown menu so that there are input boxes for the Kerberos realm properties "Service Principal" and "Keytab path". Choose a realm name, check the "enabled" checkbox, and enter the service principal and the full path to the keytab file.

The New Security Realm form also controls the default roles assigned to a user the first time that they access Fusion via this realm, using the Kerberos/SPNEGO protocol. For example, once you have defined a Kerberos realm "my-kerberos-realm" for domain "MYORG.ORG", when user "any.user" in domain "MYORG.ORG" authenticates to Fusion for the first time via this realm, they will be added to the set Fusion users as username "any.user@MYORG.ORG" and they will have all default roles.

It is prudent to allow the minimum set of default roles, as all users will have these permissions. Some users will require admin privileges and a few users will require super-admin privileges. There should always be a user with super-admin privileges that can authenticate to Fusion using the native security realm and can then grant permissions to individual users as needed.

11.10.4. Kerberos/SPNEGO HTTP Authentication

SPNEGO provides a mechanism for extending Kerberos to applications that use the HTTP protocol including web browsers and the `curl` command-line utility.

Step 6. Configure the HTTP client

When a user sends a request to the Kerberized Fusion UI, a SPNEGO request (http[s]) is made. If the user is not already authenticated, the Fusion authentication proxy will yield a 401 status code and a Negotiate header. This status/header response triggers compatible clients to fetch a local ticket from their Kerberos "ticket tray". This ticket is then encoded and sent back to the Fusion. The Fusion authentication proxy will then decode the ticket, and perform a SPN.doAs(user) authentication request to the KDC/Authentication Service. Depending on the results, the proxy then successfully executes the original request (along with a session cookie) or a 401 (without the Negotiate). Clients can either choose to use the session cookie or continue authenticating on every request.

Configuring Web Browsers and `curl` for SPNEGO

The `--negotiate` option enables SPNEGO in `curl`.

IE and Safari require no additional configuration to use SPNEGO.

To configure Firefox, access the low level configuration page by loading the about:config page. Then go to the network.negotiate-auth.trusted-uris preference and add the hostname or the domain of the web server that is HTTP Kerberos SPNEGO protected (if using multiple domains and hostname use comma to separate them).

The Chrome browser must be launched from the command line with several added parameters.

To run Chrome on linux:

```
> google-chrome --enable-plugins --args\
  --auth-server-whitelist="*KERBEROS_DOMAIN"\
  --auth-negotiate-delegate-whitelist="*KERBEROS_DOMAIN"\
  --auth-schemes="basic,digest,ntlm,negotiate"
```

To run Chrome on a Mac:

```
> open 'Google Chrome.app' --args\
  --auth-server-whitelist="*ROGUECLOUD.COM"\
  --auth-negotiate-delegate-whitelist="*KERBEROS_DOMAIN"\
  --auth-schemes="basic,digest,ntlm,negotiate"
```

To run Chrome on Windows:

```
chrome.exe --auth-server-whitelist="*KERBEROS_DOMAIN"\
  --auth-negotiate-delegate-whitelist="*ROGUECLOUD.COM"\
  --auth-schemes="basic,digest,ntlm,negotiate"
```

For more information, see Using a Web Browser to Access an URL Protected by Kerberos HTTP SPNEGO and http://www.roguelynn.com/words/apache-kerberos-for-django/.

Session cookies

A successful Kerberos/SPNEGO login will yield a session cookie, this cookie is identical to the cookie yielded by the Fusion authentication proxy's current POST-login-mechanism.

The expiration policy on the cookie is currently fixed at 8 hours. But has a 1 hour "idle" max, which means if you don't make a request for 1 hour, the cookie is invalidated. Otherwise the lifetime is pushed ahead until the 8 hour max is met.

The name of the cookie is "id" and the value is a UUID. This UUID is a key that maps to an in-memory value containing the real user ID.

11.10.5. Testing and Troubleshooting

Once you have configured the Kerberos security realm, you can test it by logging out of Fusion and shutting down the browser.

Check that you have a valid Kerberos authentication ticket via the klist command.

Now open a new browser session and access the Fusion installation, <domain>:<UI port>.

This should take you directly to the main Fusion panel, bypassing the the Fusion "Welcome" login panel. To view your login profile, click on the profile icon on the top nav bar. Your user name should be your login name + "@" + your domain name.

If instead, you see the Fusion login panel, your browser is not configured for SPNEGO.

If the Fusion display consists only of an empty top nav bar, this indicates an authentication failure. Check that the path to your keytab file is correct. Then check the Fusion logs.

If your KDC uses "AES256 CTS mode with HMAC SHA1-96" for key encryption, the proxy will log this error when attempting to authenticate:

```
GSSException: Failure unspecified at GSS-API level (Mechanism level: Encryption type AES256 CTS mode with HMAC
SHA1-96 is not supported/enabled)
```

To get around this, the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy File will need to be downloaded and installed. It can be downloaded from:

- http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html

Place the jars in your JAVA_HOME/jre/lib/security/ directory, then restart Fusion.

Clicking on the "logout" icon on the top nav bar (rightmost icon or a padlock) takes you back to the main Fusion panel. If you destroy your Kerberos credentials cache via the "kdestroy" command, the next time you logout of Fusion, you will be logged out and the browser will display the Fusion login panel.

11.10.6. References and Tutorials

https://en.wikipedia.org/wiki/Kerberos_%28protocol%29

http://www.roguelynn.com/words/explain-like-im-5-kerberos/

http://thekspace.com/home/component/content/article/54-kerberos-and-spnego.html

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Managing_Smart_Cards/Using_Kerberos.html#about-kerberos

http://www.oracle.com/technetwork/articles/idm/weblogic-sso-kerberos-1619890.html - section "Install Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files"

http://www.cisco.com/c/en/us/support/docs/security-vpn/kerberos/16087-1.html

11.11. Configuring Fusion for SAML

SAML 2.0 is a standard for exchanging authentication and authorization data between security domains. The SAML protocol allows web-browser single sign-on (SSO) through a sequence of messages sent to and from the browser, which is the intermediary between Fusion and the SAML authority acting as the Identity Provider (IdP).

Fusion has been tested with the following IdPs (but any IdP should work):

- ADFS
- Okta
- OneLogin
- OpenAM
- Shibboleth

To configure Fusion to use SAML 2.0 for user authentication and authorization you must create a SAML security realm. In addition to configuring the Fusion security realm, you must configure the SAML identity provider to recognize the Fusion application.

Once Fusion is configured for a SAML realm, this realm is added to the list of available realms on the initial Fusion sign-on panel. When the SAML realm is chosen from the list of available realms, the browser then redirects to the IdP which handles user authentication. Upon successful authentication, the IdP sends a response back to the browser which contains authentication and authorization information as well as the URL of the Fusion application. The browser redirects back to the Fusion URL, passing along the SAML message with the user authentication and authorization information. Fusion then issues as session cookie which is used for subsequent user access.

11.11.1. Fusion Configuration for a SAML Realm

You must get the following information about the SAML Identity Provider either from your sys admin or from the IdP directly:

- Identity Provider URL - the URL used by the SAML authority for single sign-on. Usually a URL which ends in "saml/sso", e.g. "https://www.my-idp.com/<my-app-path>/sso/saml"
- Issuer - SAML Issuer Id. A unique ID for that authority, e.g. "http://www.my-idp.com/exk686w2xi5KTuSXz0h7".
- Certificate Fingerprint - the contents of the SAML authority certificate, *without the certificate header and footer*. You must get this certificate from the SAML Identity Provider. The certificate is a text file which has a pair of header and footer lines which say "BEGIN CERTIFICATE" and "END CERTIFICATE", respectively. The fingerprint consists of the lines between the header and the footer. You can cut and paste this information into the text box on the Fusion UI.
- User ID Attribute - an optional attribute. The Identity Provider contains the user database. By default, the Fusion username is the same as the login name known to the Identity Provider. When another field or attribute in the user record stored by the IdP should be used as the Fusion username, that attribute name is the value of the User ID Attribute.

All Fusion security realms require the following information:

- name - must be unique, should be descriptive yet short.
- type - value is "SAML" (one of the choices on the Fusion UI Security Realms config panel).
- "enabled" - default vale is true. The "enabled" setting controls whether or not Fusion allows user logins for this

security realm.

- "auto-create users" - default is true. This controls whether or not a user account is created automatically upon initial authentication. If false, new user accounts can only be created by a Fusion user with admin privileges.

11.11.2. SAML Authority Identify Provider Configuration for Fusion

The Fusion application must be registered with the SAML Identity Provider. The amount of information varies depending on the SAML authority.

All systems will require the Fusion URL to redirect to upon successful login; this is the protocol, server, and port for the Fusion application, and path "api/saml", e.g. "https://www.my-fusion-app.com:8764/api/saml". If the Fusion application is running behind a load-balancer, then this URL is the load-balancer URL plus path "api/saml". Note that the load-balancer should be session-sticky in order for the sequence of messages that comprise the SAML protocol to run to completion successfully.

Some authorities may require additional information. In particular the SAML 2.0 "AudienceRestriction" tag may be part of the SAML message. This tag specifies the domain for which the SAML trust conditions are valid, which is usually the domain in which the Fusion app is running, e.g. "https://www.my-fusion-app".

11.11.3. Example SAML Realm Configuration

The Fusion endpoint "api/realms-config" returns a JSON list of all the configuration objects for all realms. After configuring a SAML realm named "saml-test" using the okta.com developer preview tool, the configuration object for this realm is:

```
{
 "name":"saml-test",
 "realmType":"saml",
 "enabled":true,
 "config":{
     "autoCreateUsers":true,
     "idpUrl":"https://dev-417804.oktapreview.com/app/dev417804_1/exk686w2xi5KTuSXz0h7/sso/saml",
     "issuer":"http://www.okta.com/exk686w2xi5KTuSXz0h7",

"certificateFingerprint":"MIIDpDCCAoygAwIBAgIGAVQr4A4NMA0GCSqGSIb3DQEBBQUAMIGSMQswCQYDVQQGEwJVUzETMBEG\nA1UECA
wKQ2FsaWZvcm5pYTEWMBQGA1UEBwwNU2FuIEZyYW5jaXNjbzENMAsGA1UECgwET2t0YTEU\nMBIGA1UECwwLU1NPUHJvdmlkZXIxEzARBgNVBA
MMCmRldi00MTc4MDQxHDAaBgkqhkiG9w0BCQEW\nDWluZm9Ab2t0YS5jb20wHhcNMTYwNDE5MDAxNTI0WhcNMjYwNDE5MDAxNjI0WjCBkjELMA
kGA1UE\nBhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExFjAUBgNVBAcMDVNhbiBGcmFuY2lzY28xDTALBgNV\nBAoMBE9rdGExFDASBgNVBAsM
C1NTT1Byb3ZpZGVyMRMwEQYDVQQDDApkZXYtNDE3ODA0MRwwGgYJ\nKoZIhvcNAQkBFg1pbmZvQG9rdGEuY29tMIIBIjANBgkqhkiG9w0BAQEF
AAOCAQ8AMIIBCgKCAQEA\nuNz0trRFPw2d4xXoUvX2oWeZolTVeFtaTnB9SWUyjK4og0WdT7rNdBg10eTvB2ezBwXCf24UGGui\nr1kjkZjiHD
qDxKtzQYWpGuzLCjh/4PxKFGDaiUNKcE1Ig5myiEBTvMvv99XtrcI75QdUGDhbMiBr\n2PR5FukWOYepzlBzqY0JSDzX9NYJBKPkz+syK4mj0I
6dqtYOU+bcTvjF9sR7jiHtQ+d0Zl8rz1Ca\nyuE3mNUtFJ1IJrY/RArhH1AB6mXbV/de1CXmGhKQbqQAbx9SiKtki9n84gKEwuWdV0jIqcBLGx
UQ\ngjbsaIVqed2oX+7F2fh6t0g/I8NPnOWXOTvA+wIDAQABMA0GCSqGSIb3DQEBBQUAA4IBAQCPt+DR\nliIsHO/iVnmLFPGfqrCO/gMv++xn
q2wOB19YX7HhT1GIY2YZoVphrQpXH3OO0/8AZJ8ApCmq0E9x\nxUTQwQPBanVqlyLtu1Hr0c6dbAqcd5PtaEe6Ci33nayWPydhOmitvIyb/WtW
tbel9HcdUkoGvkAl\ng305jnxkhwGLJm4jkzYe+eaYhd6oG3/JcHqKDGYGf7i2Z+ny0D7vxTeBQ+8PbZfsUKg0PlKyTocb\nydSmDPISsA1xOH
5zlw+hzFIdKgD5vW7QZLvpIclNc2hqki/nWl/CHut0TnUuP/V3boREmhDu395n\n/u72pgNANfP7+2DTBb+CBTjGUsAxpKRF",
     "userIdAttribute":""
 },
 "roleNames":["search"],
 "id":"52e1c0d2-5e00-4c76-a3d4-57f1381bdb4f",
 "createdAt":"2016-04-19T19:49:04Z",
 "updatedAt":"2016-04-19T20:06:56Z"
}
```

11.11.4. References

http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

http://docs.oasis-open.org/security/saml/v2.0/saml-glossary-2.0-os.pdf

11.12. User Access Request Params

Fusion requests must come from a known user, i.e., a user with a unique user id (UUID). Fusion’s ZooKeeper registry tracks all users across all realms. Usernames must be unique within a realm. Fusion creates a globally unique user ID for all users based on the combination of username and realm.

All requests to the Fusion REST API require either a username, password, and security realm name, or the session cookie which contains the unique user ID.

11.12.1. Per-Request Authentication

To pass authentication information with each request, the realmName is specified as a query parameter on the request itself:

```
curl -u joe.smith:password123 "http://www.acme.com:8764/api/apollo/collections?realmName=acmeLDAP"
```

The default realmName parameter is "native", so for native authentication, this parameter can be omitted.

11.12.2. Session Cookies

The Fusion UI service endpoint "api/session" can be used to generate a session cookie which contains the unique user id via a POST request whose body consists of a JSON object which contains the username, password information. For users belonging to a realm other than the native realm, the request parameter "realmName" must be specified. The command to generate a session cookie for the admin user with password "password123" is:

```
curl \
 -c cookie -i -X POST -H "Content-type:application/json" -d @- \
 http://localhost:8764/api/session?realmName=native \
<<EOF
 { "username" : "admin" , "password" : "password123" }
EOF
```

The curl command takes any number of specialized arguments, followed by the URL of the request endpoint. The arguments used here are:

- `-c` : filename of cookies file. If it exists, cookies are added to it. You can use `-c -` which writes to the terminal window (std out).
- `-i` : include the HTTP-header in the output. Used here to see the cookie returned with the response.
- `-X` : request method, in this case `POST`
- `-H` : request header. The `api/session` endpoint requires `Content-type:application/json`.
- `-d` : Pass POST body as part of the command-line request. To get ready the body from a file, use the syntax `-d @<filename>`. The argument `-d @-` reads the data from stdin.

The header output shows the cookie information:

```
HTTP/1.1 201 Created
Set-Cookie: id=996e4adf-bd04-4058-a926-8ea8ca08c05a;Secure;HttpOnly;Path=/api
Content-Length: 0
Server: Jetty(9.2.11.v20150529)
```

Once the session cookie file has been created, it can be sent along in all subsequent requests to the REST API. For the curl command-line client, the `-b` flag is used to send the contents of the cookie file to the server along with the request.

The following command sends a GET request to the Fusion REST API Collections service to check the status of the "system_metrics" collection. The `-b` flag sends in a freshly generated session cookie.

```
> curl -b cookie -i http://localhost:8764/api/apollo/collections/system_metrics

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Encoding: gzip
Vary: Accept-Encoding, User-Agent
Content-Length: 278
Server: Jetty(9.2.11.v20150529)

{
  "id" : "system_metrics",
  "createdAt" : "2016-03-04T23:29:47.779Z",
  "searchClusterId" : "default",
  "commitWithin" : 10000,
  "solrParams" : {
    "name" : "system_metrics",
    "numShards" : 1,
    "replicationFactor" : 1
  },
  "type" : "METRICS",
  "metadata" : { }
}
```

If the session cookie has expired, the system returns a 401 Unauthorized code:

```
> curl -b cookie -i http://localhost:8764/api/apollo/collections/system_metrics

HTTP/1.1 401 Unauthorized
Content-Type: application/json; charset=utf-8
Content-Length: 31
Server: Jetty(9.2.11.v20150529)

{"code":"session-idle-timeout"}
```

Chapter 12. Spark and Machine Learning

Apache Spark is an open-source cluster-computing framework. Spark improves on previous previous MapReduce implementations by use of Resilient Distributed Datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner. To schedule and run jobs on the nodes in the cluster, Spark uses Akka which is a toolkit and runtime for building highly concurrent, distributed, and resilient message-driven applications on the JVM.

Fusion manages a Spark cluster which is used for all signal aggregation processes. As of Fusion 2.4, this Spark cluster can also be used to train and compile machine learning models as well as to run experiment-management tasks via the Spark Jobs API.

The topics covered in the section are:

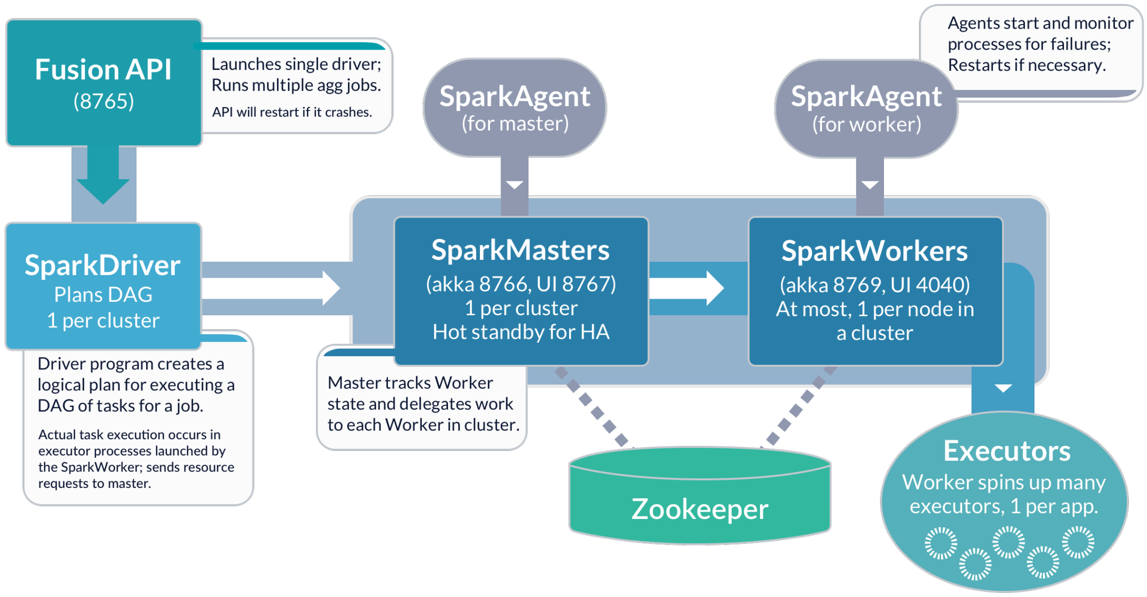
- Spark Concepts and Terminology - Spark integration in Fusion - basic terminology and schematics.
- Spark Getting Started - Starting Spark processes and working with the shell and the Spark UI.
- Spark Drivers - Running different kinds of Spark applications.
- Spark Configuration - Configuration settings for Spark applications.
- Scaling Spark Aggregations - Working with very large data sets.
- Troubleshooting - Common mistakes and how to fix them.
- Machine Learning Models in Fusion - Example of using a Spark cluster to train a sentiment classifier for tweets.

12.1. Further Reading

- Machine Learning in Lucidworks Fusion
- Apache Spark Key Terms, Explained
- Apache Spark on Wikipedia
- Akka (toolkit) on Wikipedia
- Reactive App presentation

12.2. Spark Concepts and Terminology

The following schematic shows the Spark components available from Fusion:



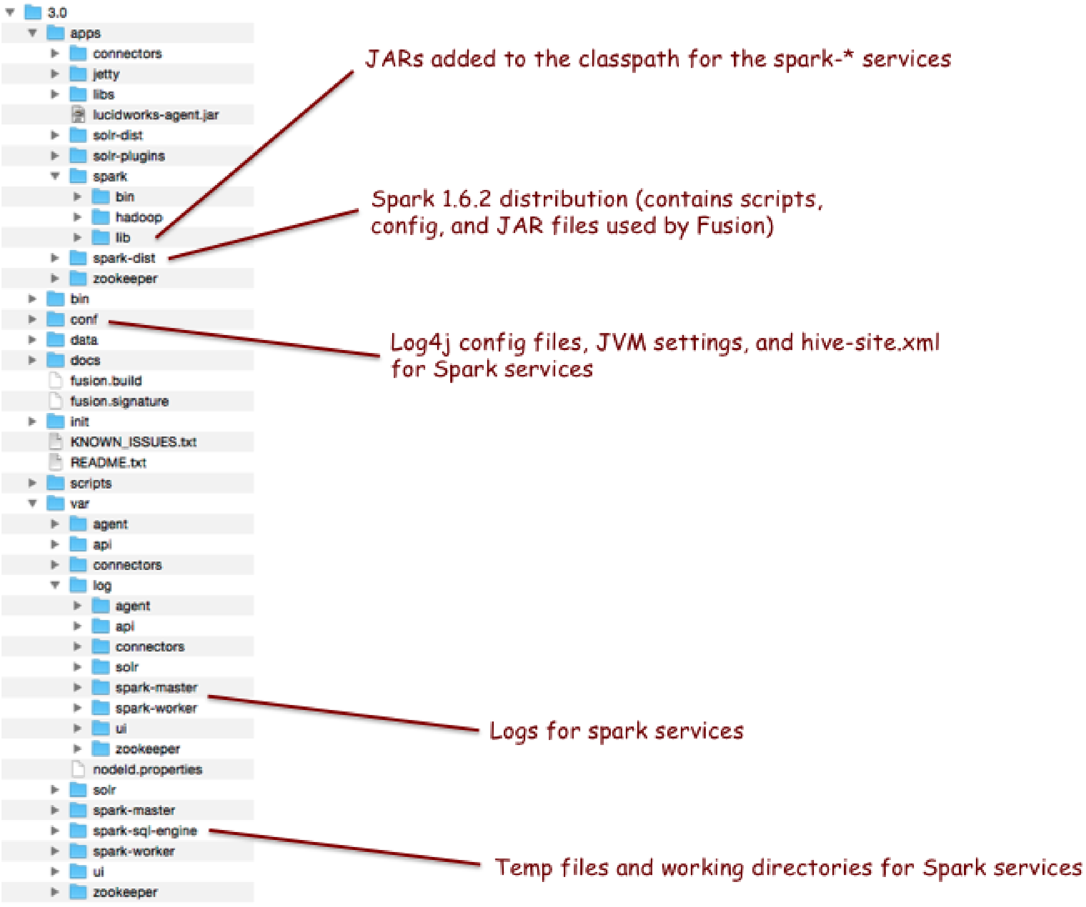
12.2.1. Spark Fusion Terminology

- **application**: an active SparkContext in Spark Master UI, which consists of a classpath and configuration. Jobs submitted to the cluster always run as classes in a specific application, ie. using the application’s classpath and configuration.
- **SparkDriver**: JVM process launched by the Fusion API service to execute Fusion jobs in Spark. SparkDriver creates and manages SparkContext for the Fusion application, and stops SparkContext when it’s no longer needed.
- **spark-master**: Agent-managed Fusion service that coordinates worker processes and applications in a Spark cluster. You should run at least 2 spark-master processes per cluster to achieve high-availability. ZooKeeper determines which spark-master process is the leader and handles fail-over.
- **spark-worker**: Agent-managed Fusion service that launches executors for Spark applications. Spark-workers communicate with the master to launch executors for an application.
- **spark-sql-engine**: Agent-managed Fusion service that runs Spark’s thrift-based SQL engine. Provides JDBC access to a Spark cluster.
- **spark-shell**: Wrapper script provided with Fusion to launch the Spark Scala REPL shell with the correct master URL (pulled from Fusion’s API) and shaded Fusion JAR added.
- **CoarseGrainedExecutorBackend**: Executor process(es) launched by a spark-worker to execute the tasks for a specific application, such as the spark-shell.
- **dynamic allocation**: Spark configuration setting that allows the Spark master to reclaim allocated resources (cpu & memory) from an "application" if it is not using the resources.
- **shaded JAR**: The Fusion API service creates an uber-jar containing all of the dependencies needed to use spark-solr and Fusion classes within a spark job. The classes that conflict with classes on spark’s classpath are shaded to ensure that Fusion classes use the correct version.
- **akka**: Akka is a toolkit and runtime for building highly concurrent, distributed, and resilient message-driven applications on the JVM. Akka uses the Actor model to hide all the thread-related code and provides simple

interfaces which allow you to more easily implement a scalable and fault-tolerant system. Spark is built on top of Akka.

12.2.2. Spark Fusion Directories

In Fusion 3.0, the following directories are used for Spark logs and components:



```
3.0
  apps
    connectors
    jetty
    libs
    lucidworks-agent.jar
    solr-dist
    solr-plugins
    spark
      bin
      hadoop
      lib
    spark-dist
    zookeeper
  bin
  conf
  data
  docs
  fusion.build
  fusion.signature
  init
  KNOWN_ISSUES.txt
  README.txt
  scripts
  var
    agent
    api
    connectors
    log
      agent
      api
      connectors
      solr
      spark-master
      spark-worker
      ui
      zookeeper
    nodeId.properties
    solr
    spark-master
    spark-sql-engine
    spark-worker
    ui
    zookeeper
```

JARs added to the classpath for the spark-* services

Spark 1.6.2 distribution (contains scripts, config, and JAR files used by Fusion)

Log4j config files, JVM settings, and hive-site.xml for Spark services

Logs for spark services

Temp files and working directories for Spark services

12.3. Spark Getting Started

The public GitHub repo https://github.com/lucidworks/fusion-spark-bootcamp contains examples and labs for learning how to use Fusion’s Spark features.

In this section, you’ll walk through some basic concepts of using Spark in Fusion. For further exposure, you should work through the labs in the Fusion Spark Bootcamp.

12.3.1. Starting Spark Master and Worker Processes

In Fusion 3.0, the Fusion run script `fusion/bin/fusion` with the argument `start` doesn’t start the spark-master and spark-worker processes. This reduces the number of Java processes needed to run Fusion and therefore reduces memory and CPU consumption.

Jobs that depend on Spark, e.g. aggregations, will still execute in what Spark calls "local" mode. When in local mode, Spark executes tasks in-process in the driver application JVM. Local mode is intended for jobs that consume/produce small datasets. One caveat of using local mode is that a persistent Spark UI is not available, but you can access the driver/job application UI at port `:4040` while the local SparkContext is running.

Tip

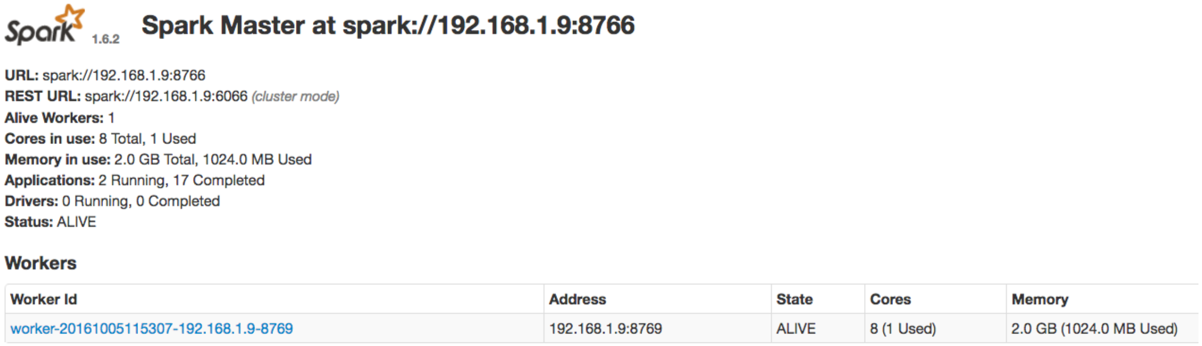
To allow the spark-master and spark-worker processes to start and stop via `bin/fusion start` and `bin/fusion stop`, we recommend adding them to the `group.default` definition in `conf/fusion.properties`:

```
group.default = zookeeper, solr, api, connectors,
ui, spark-master, spark-worker
```

To scale-out Spark in Fusion to support larger data sets and to speed up processing, you should start the spark-master and spark-worker processes:

```
$ cd /path/to/fusion
$ bin/spark-master start
$ bin/spark-worker start
```

After starting the master and worker services, direct your browser to http://localhost:8767 to view the Spark UI. The display should be similar to:



Spark 1.6.2 Spark Master at spark://192.168.1.9:8766

URL: spark://192.168.1.9:8766
REST URL: spark://192.168.1.9:6066 *(cluster mode)*
Alive Workers: 1
Cores in use: 8 Total, 1 Used
Memory in use: 2.0 GB Total, 1024.0 MB Used
Applications: 2 Running, 17 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20161005115307-192.168.1.9-8769	192.168.1.9:8769	ALIVE	8 (1 Used)	2.0 GB (1024.0 MB Used)

Check the following logs if you do not see the master UI or at least one worker in the ALIVE state.

```
fusion/var/log/spark-master/spark-master.log
fusion/var/log/spark-worker/spark-worker.log
```

Use the Fusion API to get the status of the Spark master via the following request:

```
curl http://localhost:8765/api/v1/spark/master/status
```

This should return a response of the form:

```
[ {
  "url" : "spark://192.168.1.9:8766",
  "status" : "ALIVE",
  "workers" : [ {
    "id" : "worker-20161005175058-192.168.1.9-8769",
    "host" : "192.168.1.9",
    "port" : 8769,
    "webuiaddress" : "http://192.168.1.9:8082",
    "cores" : 8,
    "coresused" : 0,
    "coresfree" : 8,
    "memoryused" : 0,
    "memoryfree" : 2048,
    "state" : "ALIVE",
    "lastheartbeat" : 1475711489460
  } ], ...
```

If you have multiple Spark masters running in a Fusion cluster, each will be shown in the status but only one will be ALIVE; the other masters will be in STANDBY mode.

Tip	If you are operating a multi-node Spark cluster, we recommend running multiple Spark master processes to achieve high-availability if the active one fails, the standby will take over.

12.3.2. Running a Job in the Spark Shell

Once you have started the Spark master and worker, launch the Fusion Spark shell wrapper:

```
$ bin/spark-shell
```

It may take a few minutes to load the first time as the script needs to download the shaded Fusion JAR file from the API service. When the shell is initialized, you'll see the prompt:

```
scala>
```

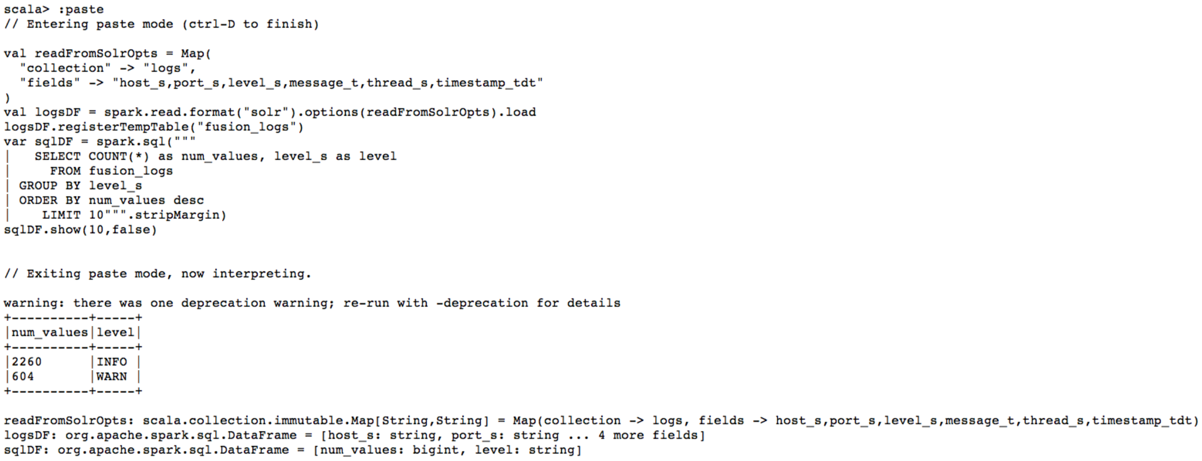
Type `:paste` to activate paste mode in the shell and paste in the following Scala code:

```
val readFromSolrOpts = Map(
  "collection" -> "logs",
  "fields" -> "host_s,port_s,level_s,message_t,thread_s,timestamp_tdt"
)
val logsDF = sqlContext.read.format("solr").options(readFromSolrOpts).load
logsDF.registerTempTable("fusion_logs")
var sqlDF = sqlContext.sql("""
|   SELECT COUNT(*) as num_values, level_s as level
|     FROM fusion_logs
| GROUP BY level_s
| ORDER BY num_values desc
|    LIMIT 10""".stripMargin)
sqlDF.show(10,false)
```

Note	In Fusion 2.4.x, you'll need to specify the Solr zkhost in the readFromSolrOpts Map, but in 3.x we set it automatically using Fusion configuration API.

Make sure the zkhost val is correct for your environment!

Type Ctrl+D to execute the script. Your results should look similar to the following:



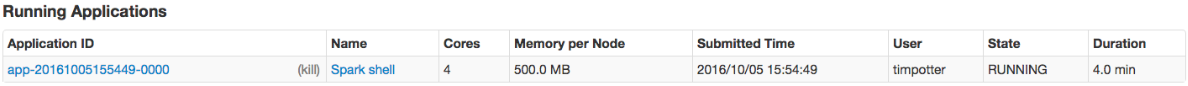
```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val readFromSolrOpts = Map(
  "collection" -> "logs",
  "fields" -> "host_s,port_s,level_s,message_t,thread_s,timestamp_tdt"
)
val logsDF = spark.read.format("solr").options(readFromSolrOpts).load
logsDF.registerTempTable("fusion_logs")
var sqlDF = spark.sql("""
|   SELECT COUNT(*) as num_values, level_s as level
|     FROM fusion_logs
| GROUP BY level_s
| ORDER BY num_values desc
|    LIMIT 10""".stripMargin)
sqlDF.show(10,false)


// Exiting paste mode, now interpreting.

warning: there was one deprecation warning; re-run with -deprecation for details
+----------+-----+
|num_values|level|
+----------+-----+
|2260      |INFO |
|604       |WARN |
+----------+-----+

readFromSolrOpts: scala.collection.immutable.Map[String,String] = Map(collection -> logs, fields -> host_s,port_s,level_s,message_t,thread_s,timestamp_tdt)
logsDF: org.apache.spark.sql.DataFrame = [host_s: string, port_s: string ... 4 more fields]
sqlDF: org.apache.spark.sql.DataFrame = [num_values: bigint, level: string]
```

Congratulations, you just ran your first Fusion Spark job that reads data from Solr and performs a simple aggregation!

12.3.3. The Spark Shell UI

The Spark Master UI allows us to dig into the details of the Spark job. This handy guide helps you understand the Spark UI: https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-webui.html.

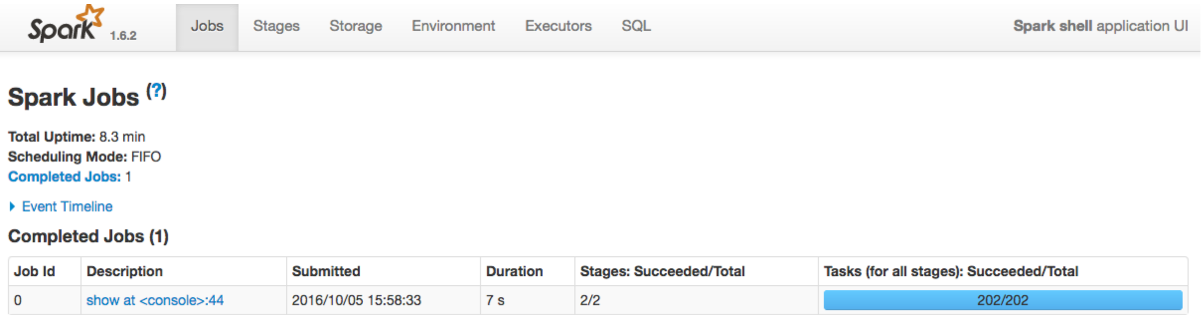
In your browser (http://localhost:8767), there should be a job named "Spark shell" under running applications (the application ID will be different than the following screenshot):



Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20161005155449-0000 (kill)	Spark shell	4	500.0 MB	2016/10/05 15:54:49	timpotter	RUNNING	4.0 min

Click on the application ID and then on the **Application Detail UI** link:



Spark Jobs (?)

Total Uptime: 8.3 min
Scheduling Mode: FIFO
Completed Jobs: 1

Completed Jobs (1)

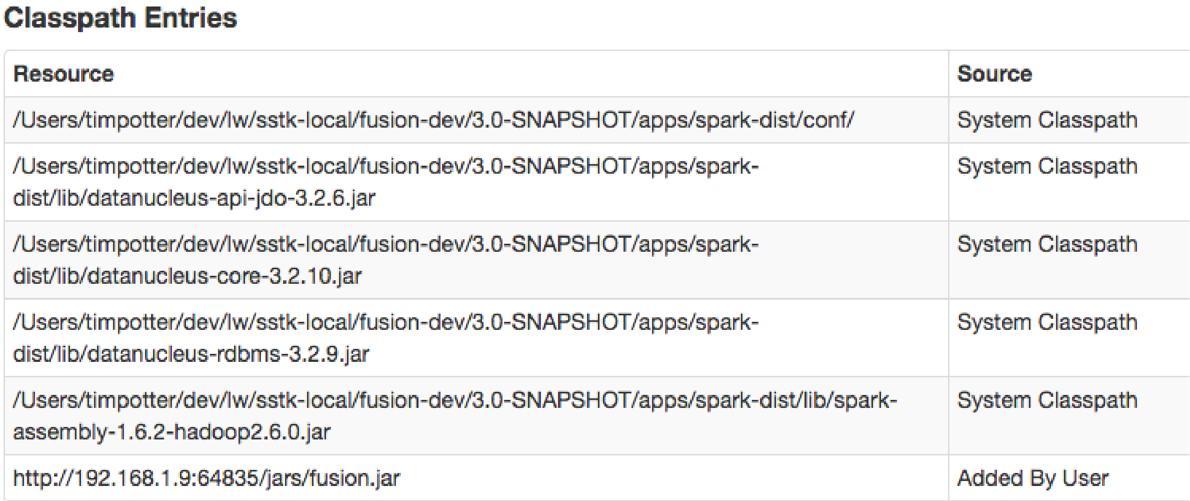
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	show at <console>:44	2016/10/05 15:58:33	7 s	2/2	202/202

Notice the tabs at the top that allow you to dig into details about the running application. Take a moment to explore the UI. Can answer the following questions about your application:

- How many tasks were needed to execute this job?
- Which JARs were added to the classpath for this job? Hint: look under the Environment tab.
- How many executor processes were used to run this job? Why? Hint: take a look at all the Spark configuration properties under the Environment tab.
- How many rows were read from Solr for this job? Hint: look under the SQL tab

For the above run, the answers are:

- 202 tasks were needed to execute this job.
- The Environment tab shows that one of the JAR files is named "fusion.jar" and was "Added By User". The fusion.jar file is a copy of the shaded JAR created by the API service:



Classpath Entries

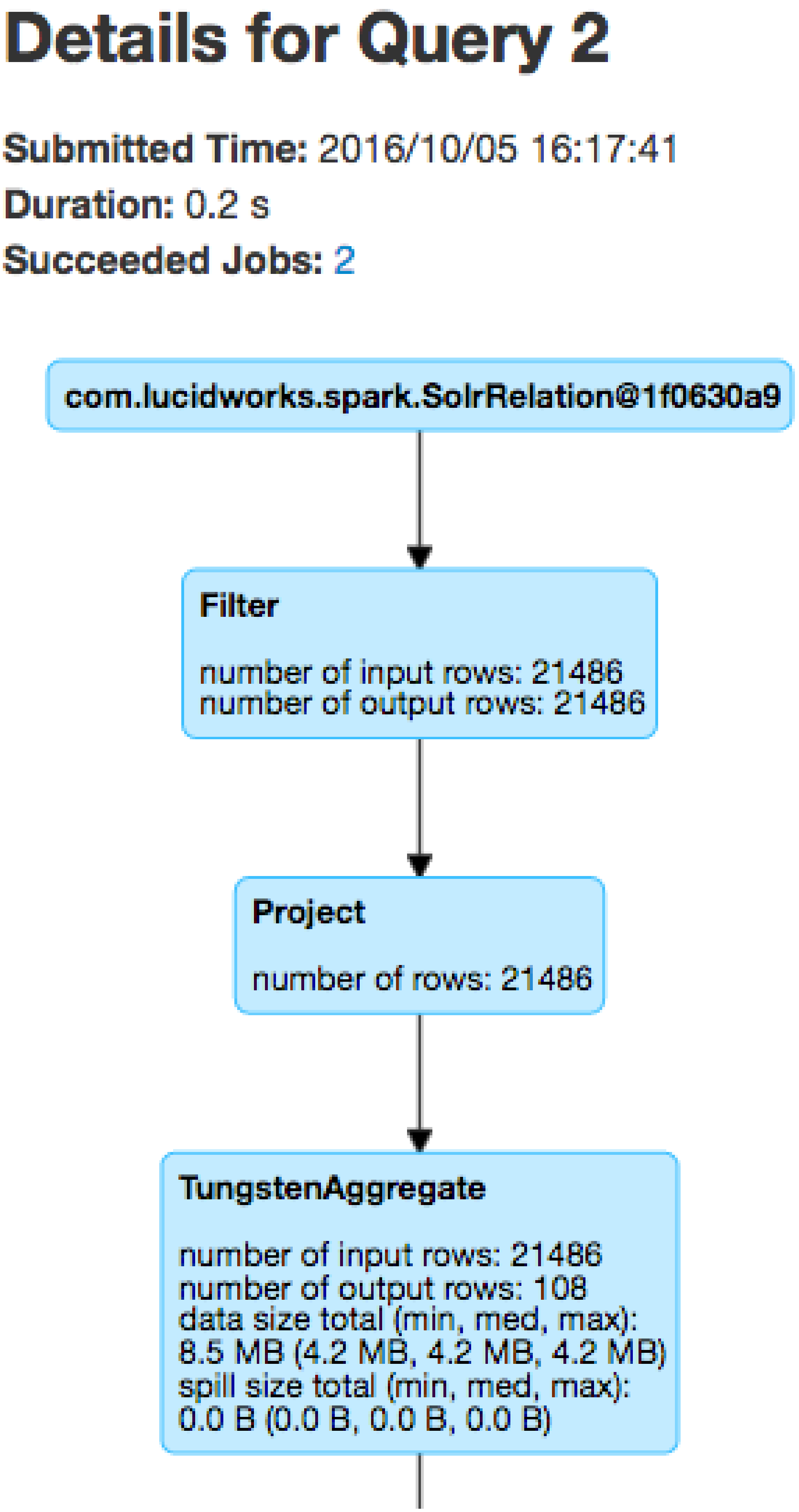
Resource	Source
/Users/timpotter/dev/lw/sstk-local/fusion-dev/3.0-SNAPSHOT/apps/spark-dist/conf/	System Classpath
/Users/timpotter/dev/lw/sstk-local/fusion-dev/3.0-SNAPSHOT/apps/spark-dist/lib/datanucleus-api-jdo-3.2.6.jar	System Classpath
/Users/timpotter/dev/lw/sstk-local/fusion-dev/3.0-SNAPSHOT/apps/spark-dist/lib/datanucleus-core-3.2.10.jar	System Classpath
/Users/timpotter/dev/lw/sstk-local/fusion-dev/3.0-SNAPSHOT/apps/spark-dist/lib/datanucleus-rdbms-3.2.9.jar	System Classpath
/Users/timpotter/dev/lw/sstk-local/fusion-dev/3.0-SNAPSHOT/apps/spark-dist/lib/spark-assembly-1.6.2-hadoop2.6.0.jar	System Classpath
http://192.168.1.9:64835/jars/fusion.jar	Added By User

- It took 2 executor processes to run this job. Each executor has 2 CPUs allocated to it and the `bin/spark-shell` script asked for 4 total CPUs for the shell application.
- This particular job read about 21K rows from Solr, but this number will differ based on how long Fusion has been running.

The key take-away is that you can see how Spark interacts with Solr using the UI.

12.3.4. Spark Job Tuning

Returning to the first question, why were 202 tasks were needed to execute this job?



Details for Query 2

Submitted Time: 2016/10/05 16:17:41

Duration: 0.2 s

Succeeded Jobs: 2

com.lucidworks.spark.SolrRelation@1f0630a9

Filter

number of input rows: 21486
number of output rows: 21486

Project

number of rows: 21486

TungstenAggregate

number of input rows: 21486
number of output rows: 108
data size total (min, med, max):
8.5 MB (4.2 MB, 4.2 MB, 4.2 MB)
spill size total (min, med, max):
0.0 B (0.0 B, 0.0 B, 0.0 B)

The reason is that SparkSQL defaults to using 200 partitions when performing distributed group by operations, see property: `spark.sql.shuffle.partitions`. Let's adjust that so Spark only uses 4 tasks since our data set is so small. In the Spark shell, execute the following Scala:

```
sqlContext.setConf("spark.sql.shuffle.partitions", "4")
```

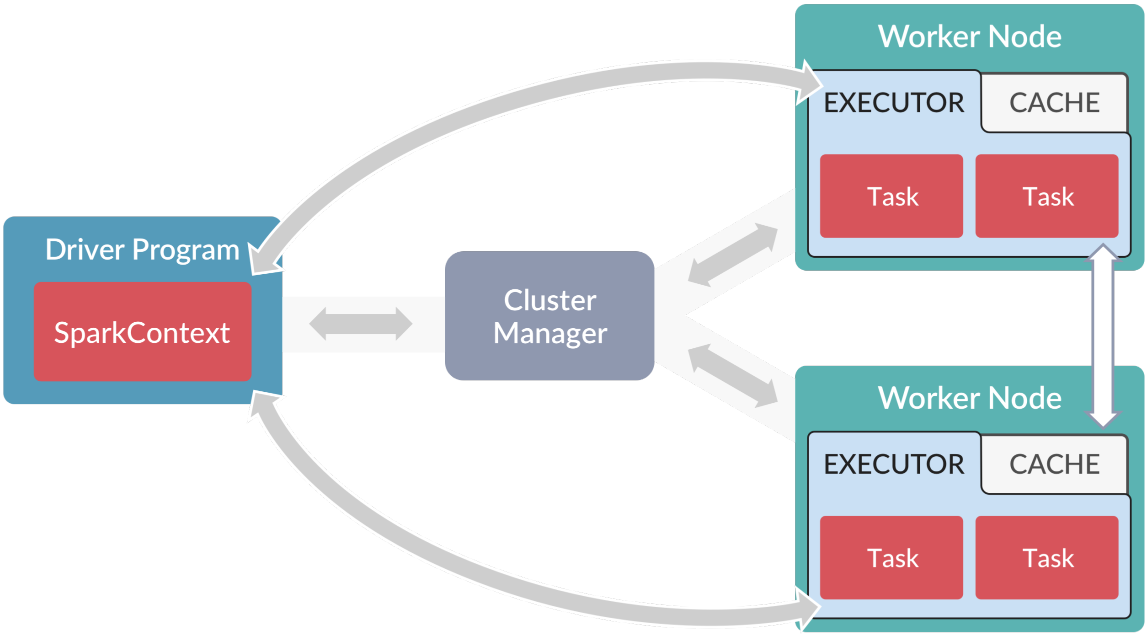
You just need to re-execute the final query and show commands:

```
val readFromSolrOpts = Map(
  "collection" -> "logs",
  "fields" -> "host_s,port_s,level_s,message_t,thread_s,timestamp_tdt"
)
val logsDF = sqlContext.read.format("solr").options(readFromSolrOpts).load
logsDF.registerTempTable("fusion_logs")
var sqlDF = sqlContext.sql("""
|   SELECT COUNT(*) as num_values, level_s as level
|     FROM fusion_logs
| GROUP BY level_s
| ORDER BY num_values desc
|    LIMIT 10""".stripMargin)
sqlDF.show(10,false)
```

Now if you look at the Job UI, you'll see a new job that executed with only 6 executors! You've just had your first experience with tuning Spark jobs.

12.4. Spark Driver Processes

A Spark "driver" is an application that creates a SparkContext for executing one or more jobs in the Spark cluster. The following diagram depicts the driver’s role in a Spark cluster:



In the diagram above, the spark-master service in Fusion is the Cluster Manager.

If your Spark job performs any collect operations, then the result of the collect (or collectAsMap) is sent back to the driver from all the executors. Consequently, if the result of the collect is too big too fit into memory, you will encounter OOM issues (or other memory related problems) when running your job.

All Fusion jobs run on Spark using a driver process started by the API service. There are three types of drivers in Fusion:

- Default driver: Executes built-in Fusion jobs, such as a signal aggregation job or a metrics rollup job.
- Scripted job driver: Executes custom script jobs; a separate driver is needed to isolate the classpath for custom Scala scripts.
- Spark-shell driver: Launch using `fusion/bin/spark-shell`.

12.4.1. Default Driver

Navigate to the Fusion UI and select the system metrics collection in the UI. Select one of the built-in Aggregation jobs, such as hourlyMetricsRollup-counters. In the diagram above, the spark-master service in Fusion is the Cluster Manager.

You must delete any existing driver applications before launching the job. Even if you haven’t started any jobs by hand, Fusion’s API service may have started one automatically, because Fusion ships with built-in jobs that run in the background which perform rollups of metrics in the system_metrics collection. Therefore, before you try to launch a job, you should run the following command:

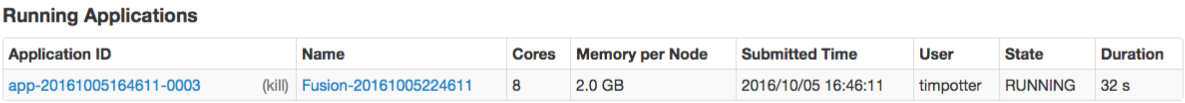
```
> curl -XDELETE http://localhost:8765/api/v1/spark/driver
```

Wait a few seconds and use the Spark UI to verify that no Fusion-spark application (e.g., "Fusion-20161005224611") is running.

In a terminal window or windows, set up a `tail -f` job on the api and spark-driver-default logs

```
$ cd /path/to/fusion
$ tail -f var/log/api/api.log var/log/api/spark-driver-default.log
```

Now, start any aggregation job from the UI. It doesn't matter whether or not this job performs any work; the goal of this exercise is to show what happens in Fusion and Spark when you run an aggregation job. You should see activity in both logs related to starting the driver application and running the selected job. The Spark UI will now show a Fusion-spark app:



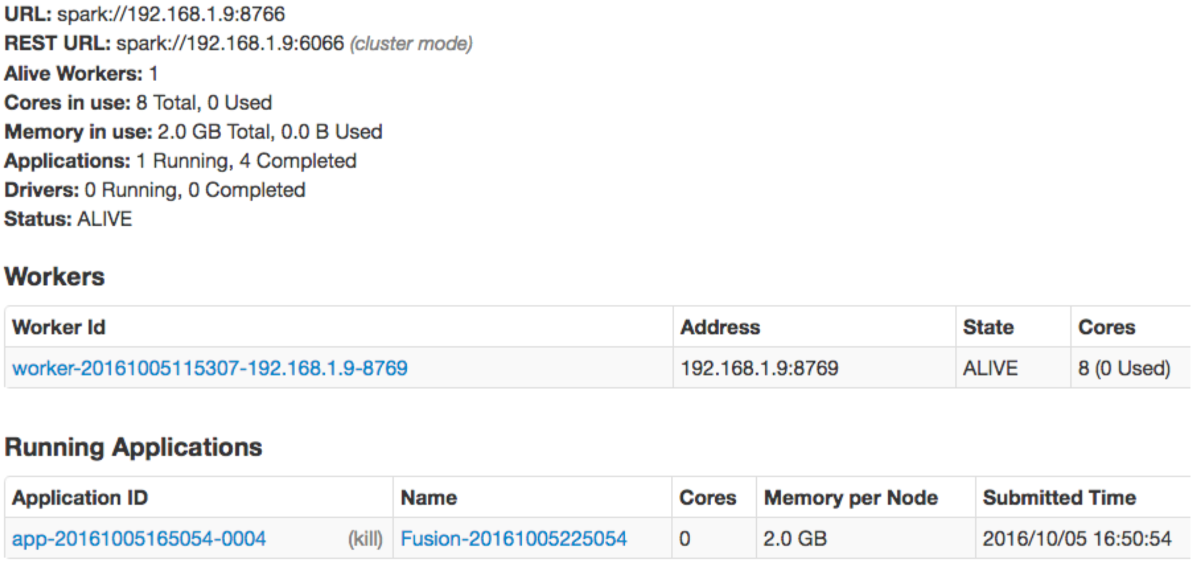
Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20161005164611-0003 (kill)	Fusion-20161005224611	8	2.0 GB	2016/10/05 16:46:11	timpotter	RUNNING	32 s

Use the `ps` command to get more details on this process:

```
$ ps waux | grep SparkDriver
```

The output should show that the Fusion SparkDriver is a JVM process started by the API service; it is not managed by the Fusion agent. Within a few minutes, the Spark UI will update itself:



URL: spark://192.168.1.9:8766
REST URL: spark://192.168.1.9:6066 *(cluster mode)*
Alive Workers: 1
Cores in use: 8 Total, 0 Used
Memory in use: 2.0 GB Total, 0.0 B Used
Applications: 1 Running, 4 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores
worker-20161005115307-192.168.1.9-8769	192.168.1.9:8769	ALIVE	8 (0 Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time
app-20161005165054-0004 (kill)	Fusion-20161005225054	0	2.0 GB	2016/10/05 16:50:54

Notice that the application no longer has any cores allocated and that all of the memory available is not being used (0.0B Used of 2.0 GB Total). This is because we launch our driver applications with `spark.dynamicAllocation.enabled=true`. This setting allows the Spark master to reclaim CPU & memory from an application if it is not actively using the resources allocated to it.

Both driver processes (default and scripted) manage a SparkContext. For the default driver, the SparkContext will be shut down after waiting a configurable (`fusion.spark.idleTime`: default 5 mins) idle time. The scripted driver shuts down the SparkContext after every scripted job is run to avoid classpath pollution between jobs.

12.4.2. Scripted Driver

Fusion supports custom script jobs, see: https://doc.lucidworks.com/fusion/2.4/REST_API_Reference/Spark-Jobs-API.html#job-types

Script jobs require a separate driver to isolate the classpath for custom Scala scripts, as well as to isolate the classpath between the jobs, so that classes compiled from scripts don't pollute the classpath for subsequent scripted jobs. For this reason the SparkContext that each scripted job uses is immediately shut down after the job is finished and recreated for new jobs. This adds some startup overhead for scripted jobs. Refer to the apachelogs lab in the Fusion Spark Bootcamp project for a complete example of a custom script job.

To troubleshoot problems with a script job, start by looking for errors in the script driver log `fusion/var/log/api/spark-driver-scripted.log`.

12.4.3. Spark Drivers in a Multi-node Cluster

To find out which node is running the Spark driver which node is running the driver when running a multi-node Fusion deployment which has several nodes running Fusion's API services, you can query the driver status via the following call to the Spark Jobs API endpoint:

```
$ curl http://localhost:8765/api/v1/spark/driver/status
```

This returns a status report:

```
{
  "/spark-drivers/15797426d56T537184c2" : {
    "id" : "15797426d56T537184c2",
    "hostname" : "192.168.1.9",
    "port" : 8601,
    "scripted" : false
  }
}
```

Note	This endpoint only exists in 3.x and 2.4.3 and higher.

12.5. Spark Configuration

Spark has a number of configuration properties. In this section, we'll cover some of the key settings you'll need to use Fusion's Spark integration.

For the full set of Fusion's spark-related configuration properties, see the Spark Jobs API.

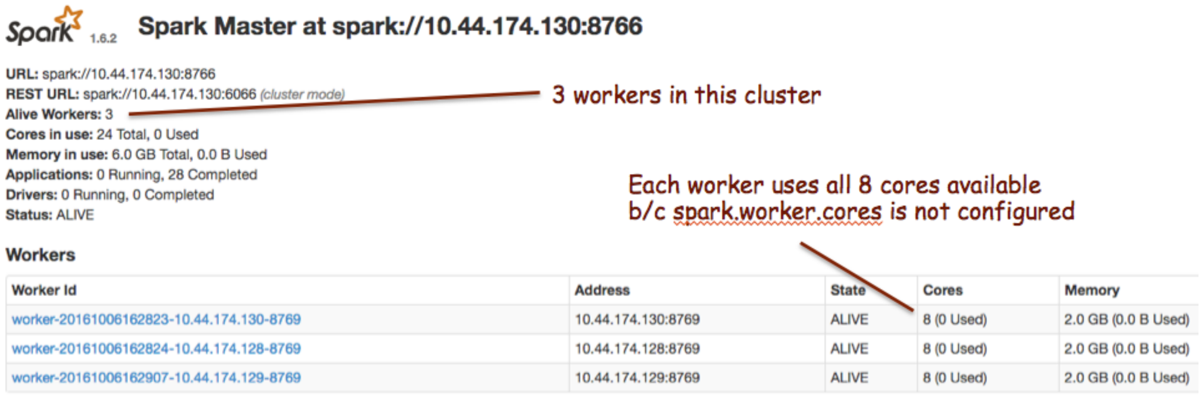
12.5.1. Spark Master / Worker Resource Allocation

Note	If you co-locate Spark workers and Solr nodes on the same server, then be sure to reserve some CPU for Solr to avoid a compute intensive Spark job from starving Solr of CPU resources.

Number of Cores Allocated

When a worker process joins the Spark cluster, it looks in the Fusion configuration (stored in Fusion's ZooKeeper) for the property `spark.worker.cores` (Fusion 2.4) or `fusion.spark.worker.cores` (Fusion 3.x). If this setting is left unspecified, then the worker will use all available cores.

For example, in the screenshot below, we have a 3-node cluster, where each worker uses all available cores (8) for a total of 24 cores:



Spark Master at spark://10.44.174.130:8766

URL: spark://10.44.174.130:8766
REST URL: spark://10.44.174.130:6066 *(cluster mode)*
Alive Workers: 3
Cores in use: 24 Total, 0 Used
Memory in use: 6.0 GB Total, 0.0 B Used
Applications: 0 Running, 28 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

3 workers in this cluster

Each worker uses all 8 cores available
b/c spark.worker.cores is not configured

Workers

Worker Id	Address	State	Cores	Memory
worker-20161006162823-10.44.174.130-8769	10.44.174.130:8769	ALIVE	8 (0 Used)	2.0 GB (0.0 B Used)
worker-20161006162824-10.44.174.128-8769	10.44.174.128:8769	ALIVE	8 (0 Used)	2.0 GB (0.0 B Used)
worker-20161006162907-10.44.174.129-8769	10.44.174.129:8769	ALIVE	8 (0 Used)	2.0 GB (0.0 B Used)

To change the CPU usage per worker, you need to use the Fusion configuration API to update this setting, as in the following examples.

Fusion 2.4

```
$ curl -u <name>:<password> -H 'Content-type:application/json' -X PUT -d '6' \
http://localhost:8764/api/apollo/configurations/spark.worker.cores
```

Fusion 3.0

```
$ curl -u <name>:<password> -H 'Content-type:application/json' -X PUT -d '6' \
http://localhost:8764/api/apollo/configurations/fusion.spark.worker.cores
```

You can also over-allocate cores to a spark-worker, which usually is recommended for hyper-threaded cores by setting the property `spark-worker.envVars` to `SPARK_WORKER_CORES=<number of cores>` in the `fusion.properties` file on all nodes hosting a spark-worker. For example, a r4.2xlarge instance in EC2 has 8 CPU cores, but the following configuration will improve utilization and performance:

```
spark-worker.envVars=SPARK_WORKER_CORES=16
```

After making this change to your spark worker nodes, you must restart the spark-worker process on each node:

```
$ cd /path/to/fusion
$ bin/spark-worker restart
```

Memory Allocation

The amount of memory allocated to each worker process is controlled by Fusion property `fusion.spark.worker.memory` which specifies the total amount of memory available for all executors spun up by that Spark Worker process. This is the quantity seen in the memory column against a worker entry in the Workers table.

Note	The JVM memory setting (-Xmx) for the spark-worker process configured in the `fusion.properties` file in Fusion 3.x and in the `spark-worker` script in Fusion 2.x controls how much memory the spark-worker needs to manage executors and not how much memory should be made available to your job(s). Typically, 512m to 1g is sufficient for the spark-worker JVM process.

Fusion 3.0:

```
$ curl -u <name>:<password> -H 'Content-type:application/json' -X PUT -d '8g' \
http://localhost:8764/api/apollo/configurations/fusion.spark.worker.memory
```

Fusion 2.4:

```
$ curl -u <name>:<password> -H 'Content-type:application/json' -X PUT -d '8g' \
http://localhost:8764/api/apollo/configurations/spark.worker.memory
```

The Spark worker process manages executors for multiple jobs running concurrently. For certain types of aggregation jobs you can also configure the per executor memory, but this may impact how many jobs you can run concurrently in your cluster. Unless explicitly specified using the parameter `spark.executor.memory`, Fusion calculates the amount of memory that can be allocated to the executor

In Fusion 3.x, the aggregation Spark jobs always get half the memory of what is assigned to the workers. This is controlled by `fusion.spark.executor.memory.fraction` property which is set to default value `0.5`. E.g., Spark workers in 3.x have 4 Gb of memory by default and the executors for aggregator Spark jobs are assigned 2 Gb for each executor. To let fusion aggregation jobs use more memory of the workers, increase `fusion.spark.executor.memory.fraction` property to `1`. This property should be used instead of the spark executor memory property.

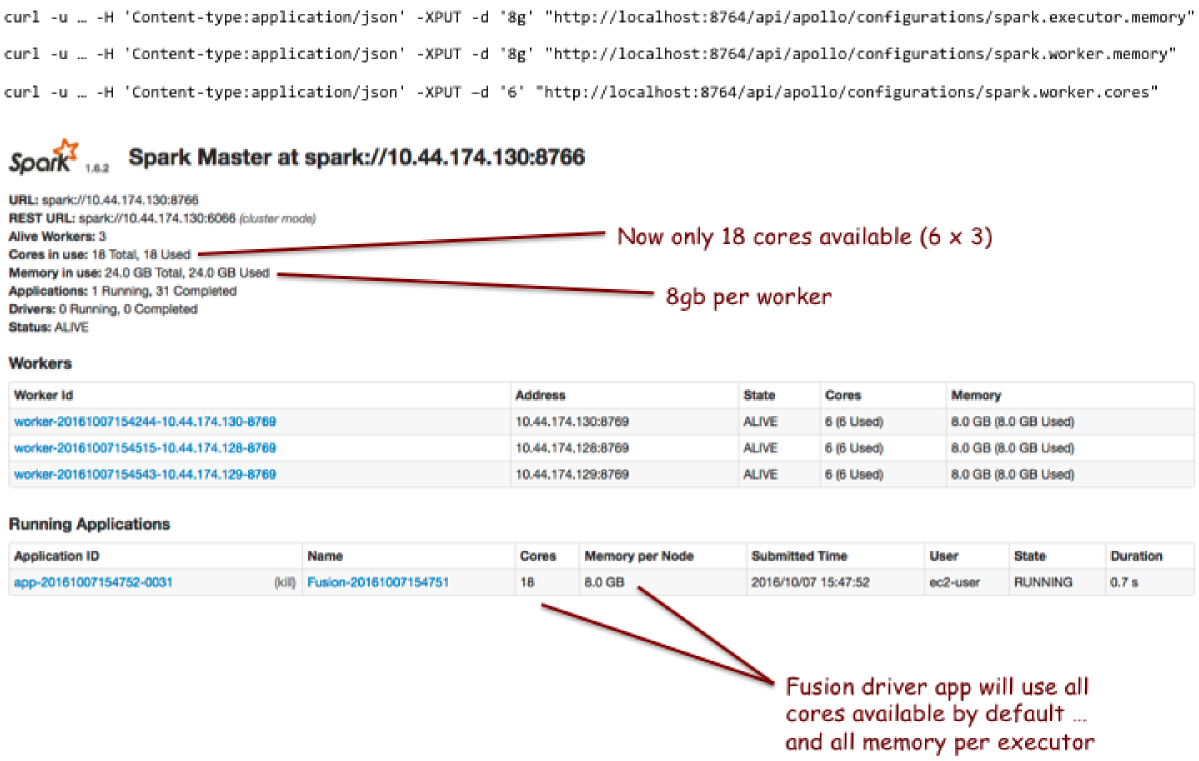
Fusion 3.0:

```
$ curl -u <name>:<password> -H 'Content-type:application/json' -X PUT -d '1' \
http://localhost:8764/api/apollo/configurations/fusion.spark.executor.memory.fraction
```

In Fusion 2.4, you must set the per-executor memory directly via property `spark.executor.memory`:

```
$ curl -u <name>:<password> -H 'Content-type:application/json' -X PUT -d '8g' \
http://localhost:8764/api/apollo/configurations/spark.executor.memory
```

After making these changes and restarting the workers, when we run a Fusion job, we get the following (screenshots taken from running instance of Fusion 2.4):



```
curl -u … -H 'Content-type:application/json' -XPUT -d '8g' "http://localhost:8764/api/apollo/configurations/spark.executor.memory"

curl -u … -H 'Content-type:application/json' -XPUT -d '8g' "http://localhost:8764/api/apollo/configurations/spark.worker.memory"

curl -u … -H 'Content-type:application/json' -XPUT -d '6' "http://localhost:8764/api/apollo/configurations/spark.worker.cores"
```

Spark Master at spark://10.44.174.130:8766

URL: spark://10.44.174.130:8766
REST URL: spark://10.44.174.130:6066 *(cluster mode)*
Alive Workers: 3
Cores in use: 18 Total, 18 Used
Memory in use: 24.0 GB Total, 24.0 GB Used
Applications: 1 Running, 31 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Now only 18 cores available (6 x 3)

8gb per worker

Workers

Worker Id	Address	State	Cores	Memory
worker-20161007154244-10.44.174.130-8769	10.44.174.130:8769	ALIVE	6 (6 Used)	8.0 GB (8.0 GB Used)
worker-20161007154515-10.44.174.128-8769	10.44.174.128:8769	ALIVE	6 (6 Used)	8.0 GB (8.0 GB Used)
worker-20161007154543-10.44.174.129-8769	10.44.174.129:8769	ALIVE	6 (6 Used)	8.0 GB (8.0 GB Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20161007154752-0031 (kill)	Fusion-20161007154751	18	8.0 GB	2016/10/07 15:47:52	ec2-user	RUNNING	0.7 s

Fusion driver app will use all cores available by default … and all memory per executor

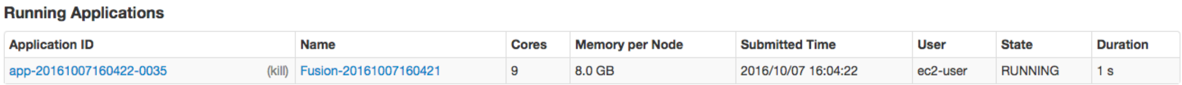
Cores per Driver Allocation

The configuration property `fusion.spark.cores.fraction` allows you to limit the number of cores used by the Fusion driver applications (default and scripted). For instance, in the screenshot above, we see 18 total CPUs available.

We set the cores fraction property to 0.5 via the following command:

```
$ curl -u <name>:<password> -H 'Content-type:application/json' -X PUT -d '0.5' \
http://localhost:8764/api/apollo/configurations/fusion.spark.cores.fraction
```

This cuts the number of available cores in half, as shown in the following screenshot:



Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20161007160422-0035 (kill)	Fusion-20161007160421	9	8.0 GB	2016/10/07 16:04:22	ec2-user	RUNNING	1 s

12.5.2. Ports used by Spark in Fusion

The following list shows the default port number used by Spark processes. If that a port number is not available, Spark will use the next available port by adding a ‘+1’ to the assigned port. E.g., if 4040 is not available, Spark will use 4041 (if available, or 4042 …etc).

Process	Port number
Spark master web UI	8767
Spark worker web UI	8082
SparkContext web UI	4040
Spark master listening port	8766
Spark worker listening port	8769
Spark driver listening port	random (spark.driver.port)
Spark executor listening port	random (spark.executor.port)
Spark blockmanager port	random (spark.blockManager.port)
Spark file server port	random (spark.fileserver.port)
Spark REPL class server port	random, can be overridden through spark.replClassServer.port

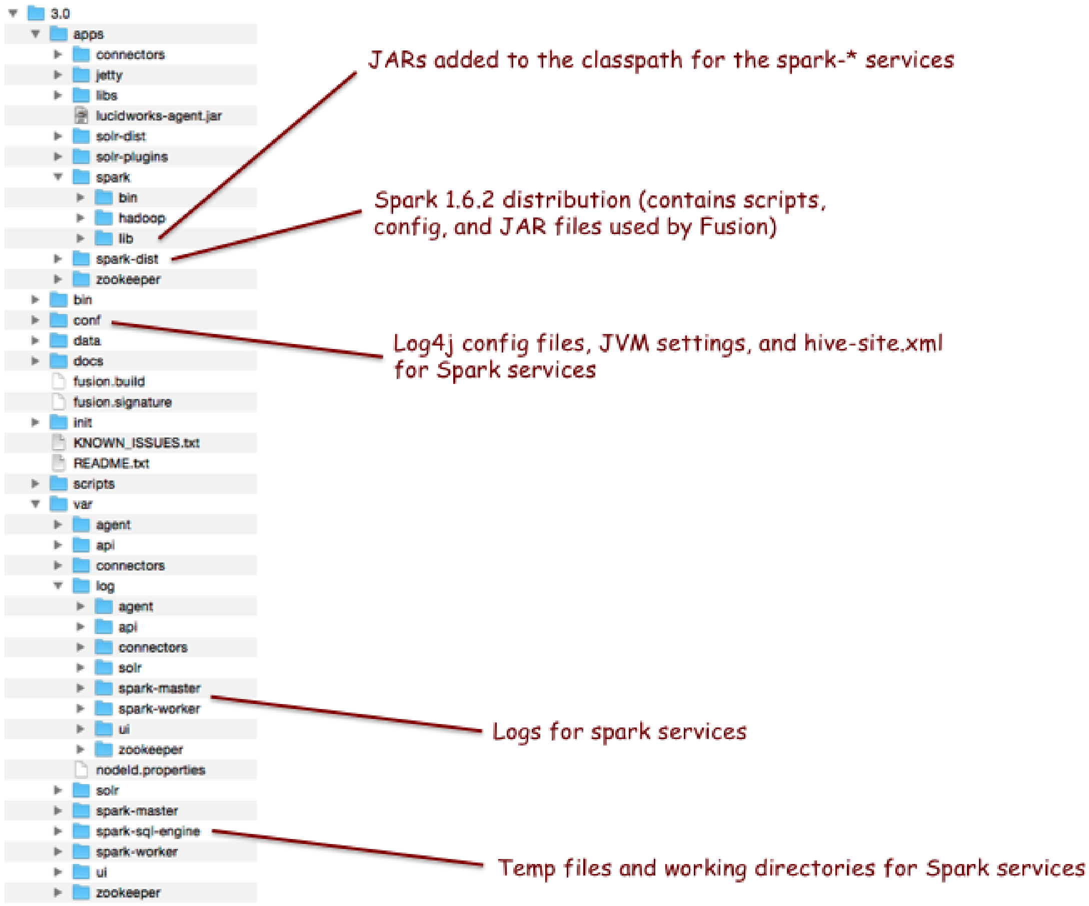
12.5.3. Directories and Temporary Files

Shaded jar file

The shaded jar file is downloaded to the `var/api/work` folder. If one of the jars in the (api) has changed, then a new shaded jar will be created with an updated name.

Temporary work directories

Temporary work dirs (spark-workDir-*) are created in `var/`’ when an application is running. They are removed after the driver is shut down or closed.



```
3.0
  apps
    connectors
    jetty
    libs
    lucidworks-agent.jar
    solr-dist
    solr-plugins
    spark
      bin
      hadoop
      lib
    spark-dist
    zookeeper
  bin
  conf
  data
  docs
  fusion.build
  fusion.signature
  init
  KNOWN_ISSUES.txt
  README.txt
  scripts
  var
    agent
    api
    connectors
    log
      agent
      api
      connectors
      solr
      spark-master
      spark-worker
      ui
      zookeeper
      nodeId.properties
    solr
    spark-master
    spark-sql-engine
    spark-worker
    ui
    zookeeper
```

JARs added to the classpath for the spark-* services

Spark 1.6.2 distribution (contains scripts, config, and JAR files used by Fusion)

Log4j config files, JVM settings, and hive-site.xml for Spark services

Logs for spark services

Temp files and working directories for Spark services

12.5.4. Connection Configurations for an SSL-enabled Solr cluster

You'll need to set these Java system properties used by SolrJ:

- `javax.net.ssl.trustStore`
- `javax.net.ssl.trustStorePassword`
- `javax.net.ssl.trustStoreTyep`

For the following Spark configuration properties:

- `spark.executor.extraJavaOptions`
- `fusion.spark.driver.jvmArgs`
- `spark.driver.extraJavaOptions`

```
$ curl -H 'Content-type:application/json' -X PUT \
  -d '-Djavax.net.ssl.trustStore=/opt/app/jobs/ssl/solrtrust.jks -Djavax.net.ssl.trustStorePassword=changeit
-Djavax.net.ssl.trustStoreType=jks' \
  "http://localhost:8765/api/v1/configurations/spark.executor.extraJavaOptions"

$ curl -H 'Content-type:application/json' -X PUT \
  -d '-Djavax.net.ssl.trustStore=/opt/app/jobs/ssl/solrtrust.jks -Djavax.net.ssl.trustStorePassword=changeit
-Djavax.net.ssl.trustStoreType=jks' \
  "http://localhost:8765/api/v1/configurations/fusion.spark.driver.jvmArgs"

$ curl -H 'Content-type:application/json' -X PUT \
  -d '-Djavax.net.ssl.trustStore=/opt/app/jobs/ssl/solrtrust.jks -Djavax.net.ssl.trustStorePassword=changeit
-Djavax.net.ssl.trustStoreType=jks' \
  "http://localhost:8765/api/v1/configurations/spark.driver.extraJavaOptions"
```

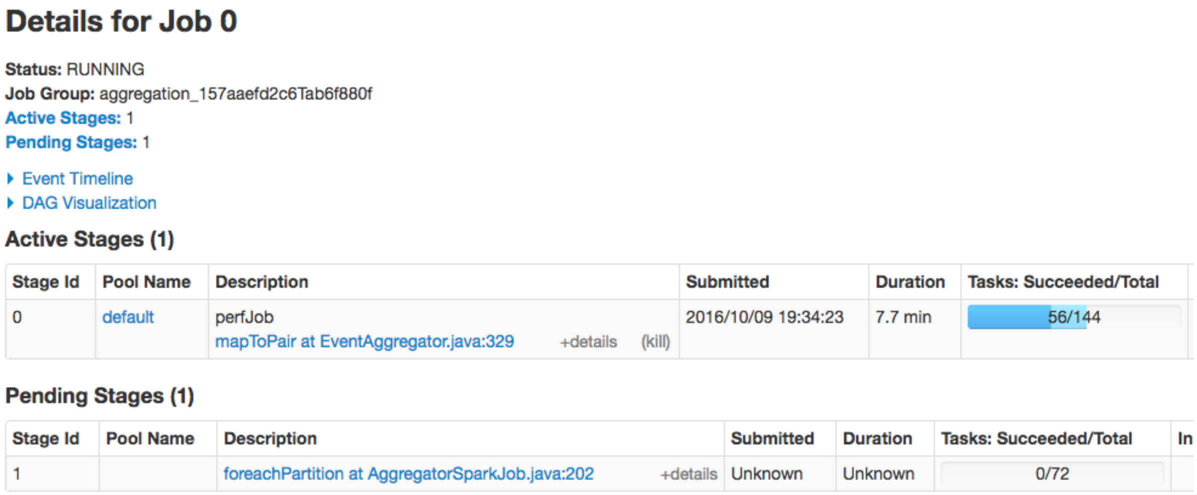
12.6. Scaling Spark Aggregations

For this section, we’ll walk through the process of running a simple aggregation on 130M signals (created from synthetic data).

One of the most common issues when running an aggregation job over a large signals data set is task timeout issues in Stage 2 (foreachPartition). This is typically due to slowness indexing aggregated jobs back into Solr or JavaScript functions. The solution is to increase the number of partitions of the aggregated RDD (the input to Stage 2). You need to increase the default parallelism by setting the following configuration property:

```
> curl -u ... -H 'Content-type:application/json' -X PUT -d '72'
"$FUSION_API/configurations/spark.default.parallelism"
```

After making the change, the foreachPartition stage of the job will use 72 partitions:



Details for Job 0

Status: RUNNING
Job Group: aggregation_157aaefd2c6Tab6f880f
Active Stages: 1
Pending Stages: 1

- Event Timeline
- DAG Visualization

Active Stages (1)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total
0	default	perfJob mapToPair at EventAggregator.java:329 +details (kill)	2016/10/09 19:34:23	7.7 min	56/144

Pending Stages (1)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	In
1		foreachPartition at AggregatorSparkJob.java:202 +details	Unknown	Unknown	0/72	

You can increase the number of rows read per page, default is 10000, by passing the rows parameter when starting your aggregation job, such as:

```
> curl -u ... -XPOST "$FUSION_API/aggregator/jobs/perf_signals/perfJob?rows=20000&sync=false"
```

For example, we were able to read 130M signals from Solr in 18 minutes at ~120K rows / sec using rows=20000 vs. 21 minutes using the default 10000.

12.7. Spark Troubleshooting

Troubleshooting Tips and Techniques

12.7.1. Job Hung in Waiting Status

Check the logs for a message that looks like:

```
2016-10-07T11:51:44,800 - WARN  [Timer-0:Logging$class@70] - Initial job has not accepted any resources; check
your cluster UI to ensure that workers are registered and have sufficient resources
```

If you see this, then it means your job has requested more CPU or memory than is available. For instance, if you ask for 4g but there is only 2g available, then the job will just hang in WAITING status.

12.7.2. Lost Executor Due to Heartbeat Timeout

If you see errors like the following:

```
2016-10-09T19:56:51,174 - WARN  [dispatcher-event-loop-5:Logging$class@70] - Removing executor 1 with no
recent heartbeats: 160532 ms exceeds timeout 120000 ms

2016-10-09T19:56:51,175 - ERROR [dispatcher-event-loop-5:Logging$class@74] - Lost executor 1 on ip-10-44-188-
82.ec2.internal: Executor heartbeat timed out after 160532 ms

2016-10-09T19:56:51,178 - WARN  [dispatcher-event-loop-5:Logging$class@70] - Lost task 22.0 in stage 1.0 (TID
166, ip-10-44-188-82.ec2.internal): ExecutorLostFailure (executor 1 exited caused by one of the running tasks)
Reason: Executor heartbeat timed out after 160532 ms
```

This is most likely due to an OOM in the executor JVM (preventing it from maintaining the heartbeat with the application driver). However, we've seen cases where tasks fail, but the job still completes, so you'll need to wait it out to see if the job recovers.

Another situation when this may occur is when a shuffle size (incoming data for a particular task) exceeds 2GB. This is hard to predict in advance because it depends on job parallelism and the number of records produced by earlier stages. The solution is to re-submit the job with increased job parallelism.

12.7.3. Spark Master won't start on EC2

http://deploymentzone.com/2014/01/06/aws-instances-and-java-net-unknownhostexception/

12.8. Machine Learning Models in Fusion

Fusion provides the following tools required for the model training process:

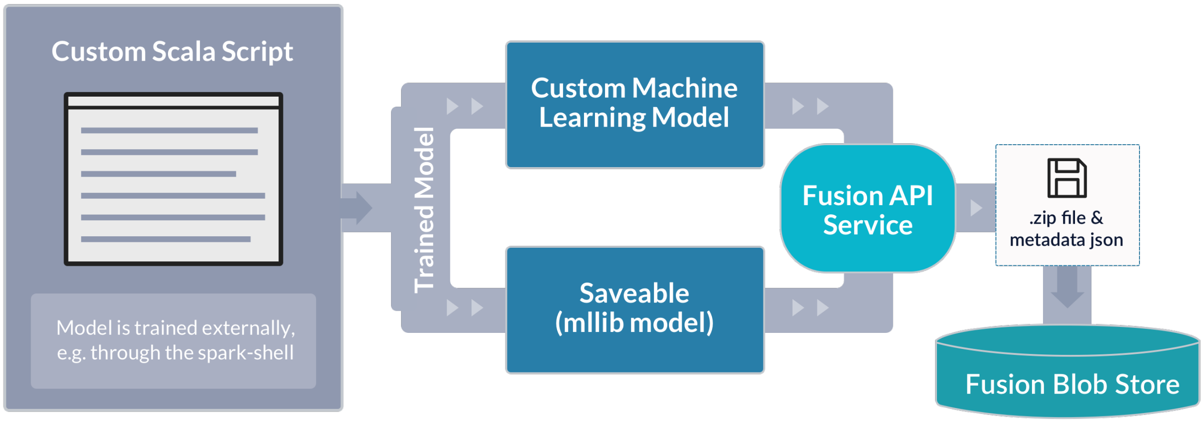
- Solr can easily store all your training data.
- Spark jobs perform the iterative machine learning training tasks.
- Fusion's blob store facility makes the final model available for processing new data.

12.8.1. Training Models

An example Scala script to train an SVM-based sentiment classifier for tweets is provided in the Lucidworks GitHub repo for the spark-solr project, see file SVMExample.scala. The SVMExample code illustrates how to read training data from Solr into Spark using the SparkSQL DataFrame API, how to run the iterations necessary to train an SVM model, and how to store the final model.

The following diagram depicts this process:

Supervised Machine Learning Model Training Workflow


Custom Scala Script
Model is trained externally, e.g. through the spark-shell
Trained Model
Custom Machine Learning Model
Saveable (mllib model)
Fusion API Service
.zip file & metadata json
Fusion Blob Store

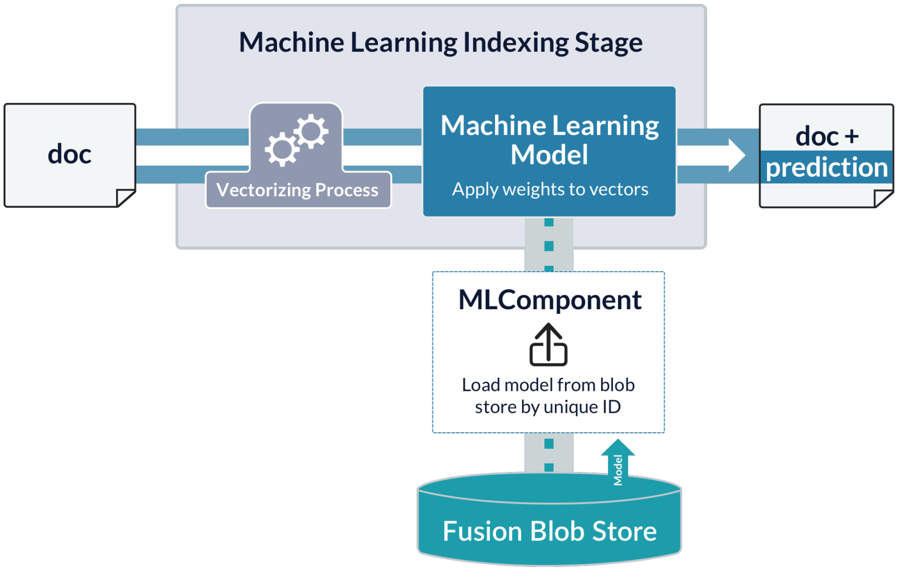
```
curl -u user:pass - X PUT -H "Content-type:application/zip"  \
    - - data-binary  @mllib-svm-sentiment.zip \
    "http://localhost:8764/api/apollo/blobs/tweets_sentiment_svm?modelType=spark-mllib"
```

The class com.lucidworks.spark.fusion.FusionMLModelSupport is used to save the final model into Fusion's blob store. It uploads the Spark MLlib model directory together with a file named spark-mllib.json.

12.8.2. Model Prediction

Fusion's blob store requires all stored objects have a unique ID. Once the model is stored in the Fusion blobstore, it is available to Fusion's index and query Machine Learning pipeline stages, which use the model to make predictions for new data in pipeline documents and queries. The following diagram shows how this process works:

Supervised Machine Learning Model Serving Workflow



12.8.3. Model Checking

To test the goodness of your model in Fusion, first create either a document index pipeline or a query processing pipeline which contains a Machine Learning stage that uses your model to make predictions on your data, and then send a document or query through that pipeline pipeline which contains data for which you know what the predicted value should be. For example, given a trained sentiment classifier and an index stage configured to use it, the following document should be classified as a highly positive tweet, with a value of (close to) 1.0 in the "sentiment_d" field:

```
{ "id":"tweets-2",
  "fields": [
    { "name": "tweet_txt",
      "value": "I am super excited that spring is finally here, yay! #happy" }
  ]
}
```

12.8.4. Metadata file "spark-mllib.json"

The file "spark-mllib.json" contains metadata about the model implementation, in particular, how it derives feature vectors from a document or query. The JSON object has the following attributes:

- "id" - a string label that used a unique ID for the Fusion blobstore, e.g., "tweets_sentiment_svm".
- "modelClassName" - the name of the spark-mllib class or the custom Java class that implements the com.lucidworks.spark.ml.MLModel interface.
- "featureFields" - a list of one or more field names.
- "vectorizer" - specified the processing required to derive a vector of features from the contents of the document fields listed in the "featureFields" entry.

The following example shows the "spark-mllib.json" file for the model with id "tweets_sentiment_svm":

```
{
  "id":"tweets_sentiment_svm",
  "modelClassName":"org.apache.spark.mllib.classification.SVMModel",
  "featureFields":["tweet_txt"],
  "vectorizer":[
    {
      "lucene-analyzer": {
         "analyzers":[{
           "name":"std_tok_lower",
           "tokenizer":{"type":"standard"},
        "filters":[{"type":"lowercase"}]}],
        "fields":[{"regex":".+","analyzer":"std_tok_lower"}]}
    }, {
       "hashingTF":{"numFeatures":"1000000"}
    }
  ]
}
```

The "vectorizer" consists of two steps: a lucene-analyzer step followed by a hashingTF step. The lucene-analyzer step can use any Lucene analyzer to perform text analysis. For more information about using the lucene-analyzer, see: https://lucidworks.com/blog/2016/04/13/spark-solr-lucenetextanalyzer/.

Other available vectorizer operations include: the MLlib normalizer, standard scaler, and ChiSq selector. See SVMExample.scala for an example of how to use the standard scaler.

Reference Manual

Chapter 13. Connectors Configuration Reference

13.1. Database Connectors

Content stored in a database comes in the form of a set of rows returned in response to a query. Content is fetched via a series of database queries. The rows returned are transformed into documents for the index. The query results set returned by a fixed query will change over time. Database rows lack the kind of metadata information available on files, so tracking changes to documents is not always possible.

Fusion provides the following database connectors:

- Couchbase which uses Couchbase's XDCR feature to transfer data in real time
- JDBC - for relational databases which can be accessed by a JDBC4 database driver, such as Oracle, MySQL, or Postgres
- MongoDb

13.2. Filesystem Connectors

A filesystem-based data store is a network of nodes to be traversed, where each node (e.g., a Unix file directory) provides information about its child nodes (e.g., the files in that directory) or references other nodes (e.g. links in an html document). The extent of the network of nodes to be traversed is discovered during the crawl. The crawler captures information about the node, e.g., file name, permissions, date of creation, last modification, and last access, as well as the contents of the nodes.

Fusion provides the following filesystem connectors:

- Box.com
- Dropbox
- FTP
- Google Drive
- HDFS This crawler doesn't use MapReduce for document processing, but instead treats HDFS as a simple filesystem.
- Local Filesystem
- S3 (deprecated)
- S3 Hadoop FS (Hadoop over Amazon)
- SolrXML - Specific to files in SolrXML format only; it cannot be used with any other kind of XML.
- Windows Share

13.3. Hadoop Cluster Connectors

Fusion provides connectors for the following Hadoop distributions:

- Apache 2.x
- Cloudera
- Hortonworks HDP

- MapR
- Pivotal

See section Hadoop Connector and Datasource Configuration for connector and datasource details.

13.4. Logging Connectors

- Logstash (deprecated) - collects logging and event streams from a Logstash instance

13.5. Push Content Connectors

- Push connector- push content to Fusion for indexing into Solr via a Fusion index pipeline

13.6. Repository Connectors

- Alfresco
- Azure
- Drupal
- GitHub
- JIRA
- Salesforce
- ServiceNow
- SharePoint
- Solr Index
- Subversion
- Zendesk

13.7. Script Connectors

- Javascript Connector - Javascript for custom retrieval of content from filesystems and websites

13.8. Social Media Connectors

- Jive
- Slack (deprecated)
- Twitter Search (deprecated)
- Twitter Stream (deprecated)

13.9. Web Connectors

- Web

Connectors

13.10. Alfresco Connector and Datasource Configuration

The Alfresco Connector is a crawler for the Alfresco server, which adheres to the Content Management Interoperability Services (CMIS) standard.

This connector was developed for the Alfresco Community 5.0d content repositories and hasn’t been tested on any other versions of Alfresco or any other CMIS server.

13.10.1. Configuration

Connector-specific Properties

Property	Description
f.cmis_password Password	type: `string` default value: '`’
f.cmis_repo_url CMIS Connection URL	The AtomPub connection URL, e.g., 'http://[host]:[port]/alfresco/api/-default-/public/cmis/versions/1.1/atom'. type: `string` default value: '`’
f.cmis_session_cache_size Session Cache Size	The maximum number of session connections to the repository to cache. Default is 5. type: `integer` default value: ‘5’
f.cmis_username Username	type: `string` default value: '`’
f.maxSizeBytes Max file size (bytes)	The maximum size, in bytes, of a document to crawl. type: `integer` default value: ‘4194304’

Security Trimming

Property	Description
enable_security_trimming Enable Security Trimming	type: `object`

Link Discovery

Property	Description
restrictToTree Restrict to sub-directories and child pages	If true, only documents found in a tree below the start links will be fetched. For Web crawls, this will only limit to the domain. type: `boolean` default value: 'true'
restrictToTreeAllowSubdomains Allow sub-domains in restrictToTree	If true, any sub-domain will be allowed, even if the crawl is restricted to the tree of items found below the start links. type: `boolean` default value: 'false'
restrictToTreeUseHostAndPath Use paths in restrictToTree	If true, the path in start links will be used to restrict items fetched. For example, if the start link is 'http://host.com/US', this option will limit all followed URLs to this path. type: `boolean` default value: 'false'
restrictToTreeIgnoredHostPrefixes Ignored host prefixes	List of host prefixes to ignore when checking links for restrictToTree link-legality checks. For example, 'www.' can be ignored so links with the same domain are allowed. type: `array` of `string`

Limit Documents

Property	Description
f.maxSizeBytes Max file size (bytes)	The maximum size, in bytes, of a document to crawl. type: integer default value: ‘4194304’
depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: integer default value: ‘-1’
maxItems Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: integer default value: ‘-1’
includeExtensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. type: array of string
includeRegexes Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: array of string
excludeExtensions Excluded file extensions	File extensions that should not to be fetched. This will limit this datasource to all extensions except this list. type: array of string
excludeRegexes Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: array of string default value: [` **.IMAP\s+Attachments.** ` ` **.IMAP\s+Home.** ` ` **.Data\s+Dictionary.** `]

Property	Description
delete Delete dead URIs	Set to true to remove documents from the index when they can no longer be accessed as unique documents. type: `boolean` default value: ‘true’
deleteErrorsAfter Fetch failure allowance	Number of fetch failures to tolerate before removing a document from the index. The default of -1 means no fetch failures will be removed. type: `integer` default value: ‘-1’

Crawl Performance

Property	Description
f.cmis_session_cache_size Session Cache Size	The maximum number of session connections to the repository to cache. Default is 5. type: `integer` default value: ‘5’
chunkSize Fetch batch size	The number of items to batch for each round of fetching. The default is 50. type: `integer` default value: ‘50’
fetchThreads Fetch threads	The number of threads to use during fetching. The default is 5. type: `integer` default value: ‘5’
fetchDelayMS Fetch delay	Number of milliseconds to wait between fetch requests. The default is 0. This property can be used to throttle a crawl if necessary. type: `integer` default value: ‘0’

Property	Description
fetchDelayMSPerHost Fetch delay per host	If true, the 'Fetch delay (ms)' property will be applied for each host. type: `boolean` default value: ‘false’
emitThreads Emit threads	The number of threads used to send documents from the connector to the index pipeline. The default is 5. type: `integer` default value: ‘5’
failFastOnStartLinkFailure Fail crawl if start links are invalid	If true, when Fusion cannot connect to any of the provided start links, the crawl is stopped and an exception logged. type: `boolean` default value: ‘true’
retryEmit Retry emits	Set to true for emit batch failures to be retried on a document-by-document basis. type: `boolean` default value: ‘true’

Dedupe

Property	Description
dedupe Dedupe documents	If true, documents will be deduplicated. Deduplication can be done based on an analysis of the content, on the content of a specific field, or by a JavaScript function. If neither a field nor a script are defined, content analysis will be used. type: `boolean` default value: ‘false’

Property	Description
dedupeSaveSignature Save dedupe signature	If true,the signature used for dedupe will be stored in a 'dedupeSignature_s' field. Note this may cause errors about 'immense terms' in that field. type: `boolean` default value: ‘false’
dedupeField Dedupe field	Field to be used for dedupe. Define either a field or a dedupe script, otherwise the full raw content of each document will be used. type: `string`
dedupeScript Dedupe script	Custom javascript to dedupe documents. The script must define a 'genSignature(content)\{}' function, but can use any combination of document fields. The function must return a string. type: `string`

Recrawl Rules

Property	Description
refreshAll Recrawl all items	Set to true to always recrawl all items found in the crawldb. type: `boolean` default value: ‘true’
refreshStartLinks Recrawl start links	Set to true to recrawl items specified in the list of start links. type: `boolean` default value: ‘false’
refreshErrors Recrawl errors	Set to true to recrawl items that failed during the last crawl. type: `boolean` default value: ‘false’

Property	Description
refreshOlderThan Recrawl age	Number of seconds to recrawl items whose last fetched date is longer ago than this value. type: `integer` default value: ‘-1’
refreshIDPrefixes Recrawl ID prefixes	A prefix to recrawl all items whose IDs begin with this value. type: `array` of `string`
refreshIDRegexes Recrawl ID regexes	A regular expression to recrawl all items whose IDs match this pattern. type: `array` of `string`
refreshScript Recrawl script	A JavaScript function ('shouldRefresh()') to customize the items recrawled. type: `string`
forceRefresh Force recrawl	Set to true to recrawl all items even if they have not changed since the last crawl. type: `boolean` default value: ‘false’

Crawl History

Property	Description
retainOutlinks Retain links in the crawldb	Set to true for links found during fetching to be stored in the crawldb. type: `boolean` default value: ‘true’
aliasExpiration Alias expiration	The number of crawls after which an alias will expire. The default is 1 crawl. type: `integer` default value: ‘1’

Property	Description
crawlDBType Crawl database type	The type of crawl database to use, in-memory or on-disk. type: `string` default value: ‘in-memory’ enum: \{ in-memory on-disk }
indexCrawlDBToSolr Index crawl database to Solr	EXPERIMENTAL: Set to true to index the crawl-database into a 'crawldb_' collection in Solr. type: `boolean` default value: ‘false’
reevaluateCrawlDBOnStart Reevaluate crawldb on start?	Reevaluate exisiting crawldb entries for legality on startup? type: `boolean` default value: ‘false’

Field Mapping

Property	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Property	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
pipeline Pipeline ID	The index pipeline used to process documents.

13.11. Azure Connector and Datasource Configuration

The Azure connector is used to crawl an Azure instance. Its connector type is "lucid.azure" and its plugin type is "azure".

Note	This connector is included with Fusion 2.1.3 and above. Older versions of Fusion had separate Azure Table and Azure Blob connectors.

13.11.1. Configuration

Property	Description
collection Collection	Collection documents will be indexed to. type: `string`
commit_on_finish Solr commit on finish	Set to true for a request to be sent to Solr after the last batch has been fetched to commit the documents to the index. type: `boolean` default value: ‘true’
max_bytes Max Bytes	The maximum size, in bytes, of a document to crawl. type: `integer` default value: ‘10485760’
max_connections Maximum number of connections	Maximum number of simultaneous connections to the repository. This value usually does not need to be changed. type: `integer` default value: ‘5000’
max_threads Maximum number of threads	The maximum number of threads to use for fetching data. Each thread will create a new connection to the repository, which may make overall throughput faster, but will also require more system resources including CPU and memory. type: `integer` default value: ‘5’

Property	Description
service_type Azure service type *required*	The Azure service type to crawl, either Blobs or Tables. type: `string` enum: \{ Azure Table Azure Blob }
storage_account Storage Account *required*	The Azure storage account name. type: `string` minLength: 1
storage_container Storage Container	The name of an Azure Blob container. type: `string` minLength: 1
table_filter_statement Table Filter Statement	A filter to apply to the crawl. This is using Azure's syntax for filtering table queries. type: `string`
tables Table name	The Azure table to index. type: `string` minLength: 1
token_secret Token Secret *required*	A valid Azure Access Key for authentication. type: `string` minLength: 1

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Initial Mappings	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.12. Box.com Connector and Datasource Configuration

The Box connector retrieves data from a Box.com cloud-based data repository. To fetch content from multiple Box users, you must create a Box app that uses OAuth 2.0 with JWT server authentication. For limited testing using a single user account, you can create a Box app that uses Standard OAuth 2.0 authentication.

Note In Fusion 3.1, the Box connector uses an improved approach for crawling large Box data repositories. If you use the Box connector, we recommend that you upgrade to Fusion 3.1 or later.

The `startLinks` defined for the datasource must include the Box numeric file and directory IDs. The root directory of any Box account has ID 0 (zero) - if you want to crawl your entire Box repository, you should enter '0'.

During the crawl, the human-readable path (e.g., `/FolderName/FileName.txt`) will be added to each document as the title field.

See a tutorial about the complete configuration process here (full-screen recommended):

13.12.1. Box Authorization, Access, and Refresh Tokens

Fusion supports two methods of authentication with the Box API:

- JSON Web Token (JWT)
- OAuth2

Authentication Using JWT

Box.com has released a Box Developer Edition. The Box Developer Edition offers a new functionality where app users will no longer have to create their own Box accounts to use an application.

App Auth uses the JSON Web Token (JWT) authentication architecture to establish a trusted connection with Box, allowing an application to provision and manage a new type of Box account that removes the friction of multiple logins for users or the difficulty of managing services.

For this option, Fusion needs the inputs below to crawl your Box data.

Required options are highlighted.

UI Label, **API Name**	**Description**
JWT App User ID `f.fs.appUserId`	Email address that you use to sign in to your Box co-admin account. Use the Co-admin account you created earlier for this.
JWT Public Key ID `f.fs.publicKeyId`	The public key prefix registered in Box Auth that you want to use to authenticate with
JWT Private Key File Path `f.fs.privateKeyFile`	Full path to the private-key file that Box downloaded (for the private key that corresponds to the public key you chose for JWT Public Key Id)

UI Label, **API Name**	**Description**
JWT Private Key File Password `f.fs.privateKeyPassword`	Passphrase for the private key (from the private-key file)

Tip	The biggest advantage to using the JWT App Auth Users approach is that you don't have to generate new refresh tokens. The public/private key file combination remain valid indefinitely.

Authentication Using OAuth2

In this case, Fusion uses OAuth 2.0 to authenticate to a normal Box user when accessing the Box API.

Fusion needs the inputs below to crawl your Box data.

Required options are highlighted.

UI Label, **API Name**	**Description**
OAuth Refresh Token `f.fs.refreshToken`	The refresh token requires us to authenticate using the Box.com account you want to do the crawling to get it. To speed the process, we created a handy downloadable utility. Use it like this: 1. Launch the executable JAR: `java -jar /path/to/box-oauth-generator-1.0.jar` 2. Enter the requested data. 3. Click **Get Redirect Token**. The utility obtains the `refresh_token` from Box.com using the specified credentials; it's valid for about 60 days.
Box OAuth Refresh Token File `f.fs.refreshTokenFile`	The filename in which to save the refresh token.

Tip	If you crawl your Box files at intervals of less than 60 days, Fusion will automatically update the `refresh_token` and store it in `fusion/data/connectors/container/lucid.anda/<datasourceID>`, where <datasourceID> is the ID of the datasource.

13.12.2. Configuration

Connector-specific Properties

Property	Description
f.addFileMetadata Add file metadata	Set to true to add information about documents found in the filesystem to the document, such as document owner, group, or ACL permissions. type: `boolean` default value: ‘true’
f.excludedMimeTypes Exclude MIME types	MIME types excluded from the crawl. type: `array` of `string`
f.fs.apiKey API Key	The Box API Key type: `string`
f.fs.apiSecret API Secret	The Box API Secret type: `string`
f.fs.appUserId JWT App User ID	(JWT only) The JWT App User ID with access to crawl. type: `string`
f.fs.generatedSharedLinksAccess Generated Shared Link Access	Only applicable when Generate Shared Links when Absent is selected… Sets the shared link access setting. Can be left blank (the default) or set to open, company or collaborators type: `string` default value: '`’
f.fs.generatedSharedLinksExpireDays Generated Shared Link Expries After Days	Only applicable when Generate Shared Links when Absent is selected… this will control how many days the shared links stay valid for. 0 for unlimited. type: `integer` default value: ‘0’

Property	Description
f.fs.isGenerateSharedLinkPermissionCanDownload Generated Shared Link Has Download Permission	Only applicable when Generate Shared Links when Absent is selected… On the box shared link, is the "can download" permission granted? type: `boolean` default value: ‘false’
f.fs.isGenerateSharedLinkPermissionCanPreview Generated Shared Link Has Preview Permission	Only applicable when Generate Shared Links when Absent is selected… On the box shared link, is the "can preview" permission granted? type: `boolean` default value: ‘false’
f.fs.isGenerateSharedLinkWhenAbsent Generate Shared Link When Absent	If this is selected, the crawler will automatically create a shared link for any non-shared documents it finds while crawling. Note: This will change all documents to 'Shared' in your Box view. Use with care. type: `boolean` default value: ‘false’
f.fs.privateKeyFile JWT Private Key File	(JWT only) Path to the private key file. type: `string`
f.fs.privateKeyPassword JWT Private Key Password	(JWT only) The password you entered for the private key file. type: `string`
f.fs.proxyHost Proxy host	The address to use when connecting through the proxy. type: `string`
f.fs.proxyPort Proxy port	The port to use when connecting through the proxy. type: `integer`
f.fs.proxyType Proxy type	Type of proxy to use, if any. Allowed values are 'HTTP' and 'SOCKS'. Leave empty for no proxy. type: `string`

Property	Description
f.fs.publicKeyId JWT Public Key Id	(JWT only) The public key prefix from the box.com public keys. type: `string`
f.fs.refreshToken OAuth Refresh Token	OAuth Refresh token (Not needed for JWT). type: `string`
f.fs.refreshTokenFile OAuth Refresh Token File	File that stores the refresh token for the next session type: `string` default value: ‘refresh_token.txt’
f.includedMimeTypes Include MIME types	MIME types included into the crawl. type: `array` of `string`
f.maxSizeBytes Maximum file size (bytes)	Maximum size, in bytes, of documents to fetch. type: `integer` default value: ‘4194304’
f.minSizeBytes Minimum file size (bytes)	Minimum size, in bytes, of documents to fetch. type: `integer` default value: ‘0’

Security Trimming

Property	Description
enable_security_trimming Enable Security Trimming	type: `object` object attributes: \{ }

Limit Documents

Property	Description
f.maxSizeBytes Maximum file size (bytes)	Maximum size, in bytes, of documents to fetch. type: `integer` default value: ‘4194304’
f.minSizeBytes Minimum file size (bytes)	Minimum size, in bytes, of documents to fetch. type: `integer` default value: ‘0’
f.addFileMetadata Add file metadata	Set to true to add information about documents found in the filesystem to the document, such as document owner, group, or ACL permissions. type: `boolean` default value: ‘true’
depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: ‘-1’
maxItems Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: ‘-1’
delete Delete dead URIs	Set to true to remove documents from the index when they can no longer be accessed as unique documents. type: `boolean` default value: ‘true’

Property	Description
deleteErrorsAfter Fetch failure allowance	Number of fetch failures to tolerate before removing a document from the index. The default of -1 means no fetch failures will be removed. type: `integer` default value: ‘-1’
f.includedMimeTypes Include MIME types	MIME types included into the crawl. type: `array` of `string`
f.excludedMimeTypes Exclude MIME types	MIME types excluded from the crawl. type: `array` of `string`

Dedupe

Property	Description
dedupe Dedupe documents	If true, documents will be deduplicated. Deduplication can be done based on an analysis of the content, on the content of a specific field, or by a JavaScript function. If neither a field nor a script are defined, content analysis will be used. type: `boolean` default value: ‘false’
dedupeSaveSignature Save dedupe signature	If true,the signature used for dedupe will be stored in a 'dedupeSignature_s' field. Note this may cause errors about 'immense terms' in that field. type: `boolean` default value: ‘false’
dedupeField Dedupe field	Field to be used for dedupe. Define either a field or a dedupe script, otherwise the full raw content of each document will be used. type: `string`

Property	Description
dedupeScript Dedupe script	Custom javascript to dedupe documents. The script must define a 'genSignature(content)\{}' function, but can use any combination of document fields. The function must return a string. type: `string`

Recrawl Rules

Property	Description
refreshAll Recrawl all items	Set to true to always recrawl all items found in the crawldb. type: `boolean` default value: ‘true’
refreshStartLinks Recrawl start links	Set to true to recrawl items specified in the list of start links. type: `boolean` default value: ‘false’
refreshErrors Recrawl errors	Set to true to recrawl items that failed during the last crawl. type: `boolean` default value: ‘false’
refreshOlderThan Recrawl age	Number of seconds to recrawl items whose last fetched date is longer ago than this value. type: `integer` default value: ‘-1’
refreshIDPrefixes Recrawl ID prefixes	A prefix to recrawl all items whose IDs begin with this value. type: `array` of `string`

Property	Description
refreshIDRegexes Recrawl ID regexes	A regular expression to recrawl all items whose IDs match this pattern. type: `array` of `string`
refreshScript Recrawl script	A JavaScript function ('shouldRefresh()') to customize the items recrawled. type: `string`
forceRefresh Force recrawl	Set to true to recrawl all items even if they have not changed since the last crawl. type: `boolean` default value: ‘false’

Crawl Performance

Property	Description
chunkSize Fetch batch size	The number of items to batch for each round of fetching. The default is 50. type: `integer` default value: ‘50’
fetchThreads Fetch threads	The number of threads to use during fetching. The default is 5. type: `integer` default value: ‘5’
fetchDelayMS Fetch delay	Number of milliseconds to wait between fetch requests. The default is 0. This property can be used to throttle a crawl if necessary. type: `integer` default value: ‘0’

Property	Description
emitThreads Emit threads	The number of threads used to send documents from the connector to the index pipeline. The default is 5. type: `integer` default value: ‘5’
retryEmit Retry emits	Set to true for emit batch failures to be retried on a document-by-document basis. type: `boolean` default value: ‘true’
failFastOnStartLinkFailure Fail crawl if start links are invalid	If true, when Fusion cannot connect to any of the provided start links, the crawl is stopped and an exception logged. type: `boolean` default value: ‘true’

Crawl History

Property	Description
retainOutlinks Retain links in the crawldb	Set to true for links found during fetching to be stored in the crawldb. type: `boolean` default value: ‘true’
aliasExpiration Alias expiration	The number of crawls after which an alias will expire. The default is 1 crawl. type: `integer` default value: ‘1’
crawlDBType Crawl database type	The type of crawl database to use, in-memory or on-disk. type: `string` default value: ‘in-memory’ enum: \{ in-memory on-disk }

Property	Description
indexCrawlDBToSolr Index crawl database to Solr	EXPERIMENTAL: Set to true to index the crawl-database into a 'crawldb_' collection in Solr. type: `boolean` default value: ‘false’

Field Mapping

Property	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Property	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
pipeline Pipeline ID	The index pipeline used to process documents.

13.13. Couchbase Connector and Datasource Configuration

The Couchbase connector uses the Cross-Datacenter Replication (XDCR) feature of Couchbase to retrieve data stored in Couchbase continuously in real-time.

This connector has been tested for compatibility with Couchbase Server 2.5.1 Enterprise Edition.

13.13.1. Indexing and Commits

Because this connector retrieves data continuously, two properties are available to control the frequency of commits to Solr, which makes the documents available for user queries. The properties define the maximum number of documents to queue for a commit (set to 50,000 by default) and the maximum amount of time to wait between commits (set to 120 seconds, or 2 minutes). Documents will be committed when one of those thresholds is reached first, meaning that if 2 minutes have passed and there are only 20,000 documents, a commit will occur. Similarly, if only 1 minute has passed and there are 50,000 documents in the queue, a commit will occur. These properties can be adjusted for your own requirements if needed.

Note	This connector retrieves data continuously. You can limit the number of documents it fetches during testing by setting the maximum number of documents retrieved, or you can manually stop the connector with the Fusion UI or Connector Datasources API.

13.13.2. Splitting Couchbase Documents

Because Couchbase has a flexible data model, documents may have a nested JSON structure. It is possible to split nested documents with a splitpath property, which uses an XPath-style path to the element to split on. These paths do not accept wildcards.

For example, if you have a document that looks like this:

```
{
  "first": "John",
  "last": "Doe",
  "grade": 8,
  "exams": [
      {
        "subject": "Maths",
        "test"   : "term1",
        "marks":90},
        {
         "subject": "Biology",
         "test"   : "term1",
         "marks":86}
      ]
}
```

If we want to split this document on the 'exams' element and create two documents each with a different subject, we would define `"splitpath":"/exams"` in our datasource definition (if using the Fusion UI to configure the datasource, you would enter the path without quotes).

The output from retrieving the document will look like this:

```
{
  "first": "John",
  "last": "Doe",
  "grade": 8,
  "exams": [
      {
        "subject": "Maths",
        "test"   : "term1",
        "marks":90
      }
    ]
},
{
  "first": "John",
  "last": "Doe",
  "grade": 8,
  "exams": [
        {
         "subject": "Biology",
         "test"   : "term1",
         "marks":86
        }
      ]
}
```

13.13.3. Field Mapping with Couchbase

The Couchbase connector has built-in field mapping allows mapping Couchbase fields to fields in your schema. The mapping configuration defines a field from your schema and an XPath-style path to the field in the Couchbase JSON document.

The field mapping can accept wildcards and double-wildcards to map fields automatically. Wildcards can be used, but only at the end of the path definition.

- `field_name=""` and `field_path=/docs/*` - maps all the fields under docs to the same name as given in JSON.
- `field_name=""` and `field_path=/docs/**` - maps all the fields under docs and their children fields to the same name as given in JSON.
- `field_name=searchField` and `field_path=/docs/*` - maps all the fields under /docs to a single field named 'searchField'.
- `field_name=searchField` and `field_path=/docs/**` - maps all the fields under /docs and their children fields to a single field named 'searchField'.

If mapping is not defined, a default mapping will be assigned, in the format of the second example above, i.e., `field_name=""` and `field_path=/docs/**.`

Example

This example some simple field mapping, using a single document such as this:

```
{
  "first": "John",
  "last": "Doe",
  "grade": 8,
  "exams": [
      {
        "subject": "Maths",
        "test"   : "term1",
        "marks": 90 },
        {
         "subject": "Biology",
         "test"   : "term1",
         "marks": 86 }
      ]
}
```

When we configure the datasource, we can define our field mapping as follows:

```
"field_mapping": [
{
    "field_name":"points_i",
    "field_path":"/exams/marks"
},
{
    "field_name":"",
    "field_path":"/**"
}
]
```

Two mappings are defined. The first will map the '/exams/marks' field from Couchbase to the 'points_i' field in Solr. The second maps all top-level and child fields from Couchbase to either the same field name in Solr or to a dynamic field rule.

After retrieving the document, it will look like this:

```
{
  "first_s": "John",
  "last_s": "Doe",
  "grade_i": 8,
  "exams": [
      {
        "subject_s": "Maths",
        "test_s"   : "term1",
        "points_i":90},
        {
         "subject_s": "Biology",
         "test_s"   : "term1",
         "points_i":86}
      ]
}
```

The 'marks' field from the original document has been mapped to the 'points_i' field; most of the other fields have been mapped to appropriate dynamic field rules.

Note that the representation of the document above is after it has been retrieved from Couchbase, but before it has been processed by the index pipelines. Since the index pipelines contain several stage types that can further transform the document, such as the Apache Tika Parser stage and the Field Mapping stage, the document that ends up indexed to Solr may be different from the document representation above. Some small iterations of crawling are recommended to be sure the documents are indexed as required.

13.13.4. Configuration

Property	Description
client_host Couchbase client host	Host where Couchbase replica is running, which must be reachable from Couchbase server. type: `string` default value: '127.0.0.1' minLength: 1
client_port Couchbase client port	Port number where Couchbase client will be started. type: `integer` default value: '9876'
cluster_name Cluster name *required*	Connector's cluster name in Couchbase server. type: `string` minLength: 1
collection Collection	Collection documents will be indexed to. type: `string`
commit_on_finish Solr commit on finish	Set to true for a request to be sent to Solr after the last batch has been fetched to commit the documents to the index. type: `boolean` default value: 'true'

Property	Description
max_docs Max documents	The maximum number of documents to process before stopping. The default, '-1', means no maximum and this connector will continue processing content indefinitely. type: `integer` default value: ‘-1’
num_vbuckets Number of Couchbase VBuckets	Number of VBuckets used by Couchbase. Values should be 64 if Couchbase server is installed on Mac OS or 1024 for other platforms. Note that the number of VBuckets must be consistent for both Couchbase server and Couchbase Connector. Connector will try to obtain correct value from Couchbase server and the value specified will be used in case of failure. type: `integer` enum: \{ 64 1024 }
password Password *required*	Couchbase server’s valid password. type: `string`
server_host Couchbase server host	Host where Couchbase server is running. type: `string` default value: ‘127.0.0.1’ minLength: 1
server_port Couchbase server port	Port number where Couchbase server is running. type: `integer` default value: ‘8091’

Property	Description
source_buckets Source buckets *required*	Couchbase buckets to synchronize with. type: `array` of `object` minimum number of items (`minItems`): 1 object attributes: \{ `fieldmapping` : \{ display name: Field mappings type: `array` of `object` } `name` *(required)* : \{ display name: Bucket name type: `string` minLength : 1 } `splitpath` : \{ display name: Splitpath type: `string` default value: ‘/’ } }
username Username *required*	Couchbase server’s valid username. type: `string`
verify_access Validate access	Set to true to require successful connection to the filesystem before saving this datasource. type: `boolean` default value: ‘true’

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Initial Mappings	Description

unmapped
Unmapped fields
	If fields do not match any of the field mapping rules, these rules will apply.
	type: `object`
	object attributes: \{
	`operation` : \{
	display name: Operation
	type: `string`
	default value: ‘copy’
	description : The type of mapping to perform: move, copy, delete, add, set, or keep
	enum: \{ copy move delete set add keep }
	`} +`
	`` `source` : \{ + ``
	`display name: Source Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped. +`
	`} +`
	`` `target` : \{ + ``
	`display name: Target Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped to. +`
	`} +`
	}

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.14. Dropbox Connector and Datasource Configuration

The Dropbox connector retrieves data from a Dropbox cloud-based data repository.

To fetch content from Dropbox, you must have an OAuth token accessible to the crawler so it can properly authenticate.

If you want to crawl all of the files owned by the account, you can simply add '/' as the startLink.

13.14.1. Dropbox Authentication

In order to authenticate to Dropbox, you must have a access token. This is obtained by first creating an app in the Dropbox platform as follows:

- Go to https://www.dropbox.com/developers/apps and sign in with your Dropbox account.
- Navigate to the "Your apps" page (from "App Console" in the left-hand navigation bar)
- Click "Create app". You will be taken through a short wizard-like selection process. You will be presented with the following questions:
  - What type of app do you want to create? Choose "Dropbox API app"
  - What type of data does your app need to store on Dropbox? Choose "Files and datastores"
  - Can your app be limited to its own folder? Choose "No"
  - What type of files dose your app need access to? Choose "All file types"
  - Provide a name for the app and click "Create App".
- After saving your app, you will see the settings screen for the new app. Scroll to the OAuth 2 section, then click the Generate button under "Generated access token". A long string will appear on the screen. Copy or save this token so that you can use when configuring the datasource.

Paste this access token into the "Dropbox OAuth Access Token" field in the Fusion UI. If using the REST API, this is the value of the property 'f.fs.oAuthAccessToken'.

13.14.2. Configuration

Connector-specific Properties

Property	Description
f.addFileMetadata Add file metadata	Set to true to add information about documents found in the filesystem to the document, such as document owner, group, or ACL permissions. type: `boolean` default value: ‘true’
f.fs.oAuthAccessToken Dropbox OAuth Access Token	Dropbox OAuth Token for the registered application. type: `string`

Property	Description
f.maxSizeBytes Maximum file size (bytes)	Maximum size, in bytes, of documents to fetch. type: `integer` default value: ‘4194304’
f.minSizeBytes Minimum file size (bytes)	Minimum size, in bytes, of documents to fetch. type: `integer` default value: ‘0’

Limit Documents

Property	Description
f.maxSizeBytes Maximum file size (bytes)	Maximum size, in bytes, of documents to fetch. type: `integer` default value: ‘4194304’
f.minSizeBytes Minimum file size (bytes)	Minimum size, in bytes, of documents to fetch. type: `integer` default value: ‘0’
f.addFileMetadata Add file metadata	Set to true to add information about documents found in the filesystem to the document, such as document owner, group, or ACL permissions. type: `boolean` default value: ‘true’
restrictToTree Restrict to sub-directories and child pages	If true, only documents found in a tree below the start links will be fetched. For Web crawls, this will only limit to the domain. type: `boolean` default value: ‘true’

Property	Description
depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: '-1'
maxItems Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: '-1'
includeExtensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. type: `array` of `string`
includeRegexes Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: `array` of `string`
excludeExtensions Excluded file extensions	File extensions that should not to be fetched. This will limit this datasource to all extensions except this list. type: `array` of `string`
excludeRegexes Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: `array` of `string`
delete Delete dead URIs	Set to true to remove documents from the index when they can no longer be accessed as unique documents. type: `boolean` default value: 'true'

Property	Description
deleteErrorsAfter Fetch failure allowance	Number of fetch failures to tolerate before removing a document from the index. The default of -1 means no fetch failures will be removed. type: `integer` default value: '-1'

Crawl Performance

Property	Description
chunkSize Fetch batch size	The number of items to batch for each round of fetching. The default is 50. type: `integer` default value: '50'
fetchThreads Fetch threads	The number of threads to use during fetching. The default is 5. type: `integer` default value: '5'
fetchDelayMS Fetch delay	Number of milliseconds to wait between fetch requests. The default is 0. This property can be used to throttle a crawl if necessary. type: `integer` default value: '0'
emitThreads Emit threads	The number of threads used to send documents from the connector to the index pipeline. The default is 5. type: `integer` default value: '5'
retryEmit Retry emits	Set to true for emit batch failures to be retried on a document-by-document basis. type: `boolean` default value: 'true'

Property	Description
failFastOnStartLinkFailure Fail crawl if start links are invalid	If true, when Fusion cannot connect to any of the provided start links, the crawl is stopped and an exception logged. type: `boolean` default value: ‘true’

Dedupe

Property	Description
dedupe Dedupe documents	If true, documents will be deduplicated. Deduplication can be done based on an analysis of the content, on the content of a specific field, or by a JavaScript function. If neither a field nor a script are defined, content analysis will be used. type: `boolean` default value: ‘false’
dedupeSaveSignature Save dedupe signature	If true,the signature used for dedupe will be stored in a 'dedupeSignature_s' field. Note this may cause errors about 'immense terms' in that field. type: `boolean` default value: ‘false’
dedupeField Dedupe field	Field to be used for dedupe. Define either a field or a dedupe script, otherwise the full raw content of each document will be used. type: `string`
dedupeScript Dedupe script	Custom javascript to dedupe documents. The script must define a 'genSignature(content)\{}' function, but can use any combination of document fields. The function must return a string. type: `string`

Recrawl Rules

Property	Description
refreshAll Recrawl all items	Set to true to always recrawl all items found in the crawldb. type: `boolean` default value: ‘true’
refreshStartLinks Recrawl start links	Set to true to recrawl items specified in the list of start links. type: `boolean` default value: ‘false’
refreshErrors Recrawl errors	Set to true to recrawl items that failed during the last crawl. type: `boolean` default value: ‘false’
refreshOlderThan Recrawl age	Number of seconds to recrawl items whose last fetched date is longer ago than this value. type: `integer` default value: ‘-1’
refreshIDPrefixes Recrawl ID prefixes	A prefix to recrawl all items whose IDs begin with this value. type: `array` of `string`
refreshIDRegexes Recrawl ID regexes	A regular expression to recrawl all items whose IDs match this pattern. type: `array` of `string`
refreshScript Recrawl script	A JavaScript function ('shouldRefresh()') to customize the items recrawled. type: `string`

Property	Description
forceRefresh Force recrawl	Set to true to recrawl all items even if they have not changed since the last crawl. type: `boolean` default value: ‘false’

Crawl History

Property	Description
retainOutlinks Retain links in the crawldb	Set to true for links found during fetching to be stored in the crawldb. type: `boolean` default value: ‘true’
aliasExpiration Alias expiration	The number of crawls after which an alias will expire. The default is 1 crawl. type: `integer` default value: ‘1’
crawlDBType Crawl database type	The type of crawl database to use, in-memory or on-disk. type: `string` default value: ‘in-memory’ enum: \{ in-memory on-disk }
indexCrawlDBToSolr Index crawl database to Solr	EXPERIMENTAL: Set to true to index the crawl-database into a 'crawldb_' collection in Solr. type: `boolean` default value: ‘false’
reevaluateCrawlDBOnStart Reevaluate crawldb on start?	Reevaluate exisiting crawldb entries for legality on startup? type: `boolean` default value: ‘false’

Field Mapping

Property	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Property	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
pipeline Pipeline ID	The index pipeline used to process documents.

13.15. Drupal 7.x Connector and Datasource Configuration

This connector uses Drupal’s Services 7.x3.11Module REST API.

13.15.1. Configuration

Connector-specific Properties

Property	Description
f.cacheSize Cache size (number of entries)	The number of entries to cache when making REST requests. type: `integer` default value: ‘2000’
f.comment Drupal comment service	Name of the Comment resource to be able to index comment data. If you did not create an alias for the 'comment' object, keep the default. type: `string` default value: ‘comment’
f.drupal_password Drupal password	Password to access the REST service, if required. type: `string` default value: '`’
f.drupal_username Drupal username	Optional username, only required if the REST service requires authentication. type: `string` default value: '`’
f.endpoint Drupal Endpoint	Name of the REST endpoint defined when you added the REST service to Drupal. type: `string` default value: ‘rest’

Property	Description
f.file Drupal file service	Name of the File resource to be able to index file data. If you did not create an alias for the 'file' object, keep the default. type: `string` default value: ‘file’
f.node Drupal node service	Name of the Node resource to be able to index node data. If you did not create an alias for the 'node' object, keep the default. type: `string` default value: ‘node’
f.pageSize Node Page Size (number of nodes per page)	The number of items that will be returned. The Drupal default without this value is 20, this allows you to request more items and reduce the overall number of Node requests to fetch all content. type: `integer` default value: ‘100’
f.taxonomy_term Drupal Taxonomy Term service	Name of the Taxonomy Term resource to be able to index taxonomy term data. If you did not create an alias for the 'taxonomy_term' object, keep the default. type: `string` default value: ‘taxonomy_term’
f.taxonomy_vocabulary Drupal Taxonomy Vocabulary service	Name of the Taxonomy Vocabulary resource to be able to index taxonomy data. If you did not create an alias for the 'taxonomy_vocabulary' object, keep the default. type: `string` default value: ‘taxonomy_vocabulary’
f.timeoutMS Connection timeout (ms)	Time in ms to wait for a server response. type: `integer` default value: ‘10000’

Property	Description
f.user Drupal user service	Name of the User resource to be able to login, if authenticating to the REST service is required. If you did not create an alias for the 'user' object, keep the default. type: `string` default value: ‘user’

Drupal URL Aliases

Property	Description
f.endpoint Drupal Endpoint	Name of the REST endpoint defined when you added the REST service to Drupal. type: `string` default value: ‘rest’
f.node Drupal node service	Name of the Node resource to be able to index node data. If you did not create an alias for the 'node' object, keep the default. type: `string` default value: ‘node’
f.user Drupal user service	Name of the User resource to be able to login, if authenticating to the REST service is required. If you did not create an alias for the 'user' object, keep the default. type: `string` default value: ‘user’
f.file Drupal file service	Name of the File resource to be able to index file data. If you did not create an alias for the 'file' object, keep the default. type: `string` default value: ‘file’

Property	Description
f.comment Drupal comment service	Name of the Comment resource to be able to index comment data. If you did not create an alias for the 'comment' object, keep the default. type: `string` default value: ‘comment’
f.taxonomy_vocabulary Drupal Taxonomy Vocabulary service	Name of the Taxonomy Vocabulary resource to be able to index taxonomy data. If you did not create an alias for the 'taxonomy_vocabulary' object, keep the default. type: `string` default value: ‘taxonomy_vocabulary’
f.taxonomy_term Drupal Taxonomy Term service	Name of the Taxonomy Term resource to be able to index taxonomy term data. If you did not create an alias for the 'taxonomy_term' object, keep the default. type: `string` default value: ‘taxonomy_term’

Limit Documents

Property	Description
depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: ‘-1’
maxItems Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: ‘-1’
includeExtensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. type: `array` of `string`

Property	Description
includeRegexes Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: `array` of `string`
excludeExtensions Excluded file extensions	File extensions that should not to be fetched. This will limit this datasource to all extensions except this list. type: `array` of `string`
excludeRegexes Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: `array` of `string`
delete Delete dead URIs	Set to true to remove documents from the index when they can no longer be accessed as unique documents. type: `boolean` default value: ‘true’
deleteErrorsAfter Fetch failure allowance	Number of fetch failures to tolerate before removing a document from the index. The default of -1 means no fetch failures will be removed. type: `integer` default value: ‘-1’

Crawl Performance

Property	Description
f.pageSize Node Page Size (number of nodes per page)	The number of items that will be returned. The Drupal default without this value is 20, this allows you to request more items and reduce the overall number of Node requests to fetch all content. type: `integer` default value: ‘100’

Property	Description
f.cacheSize Cache size (number of entries)	The number of entries to cache when making REST requests. type: `integer` default value: ‘2000’
f.timeoutMS Connection timeout (ms)	Time in ms to wait for a server response. type: `integer` default value: ‘10000’
chunkSize Fetch batch size	The number of items to batch for each round of fetching. The default is 50. type: `integer` default value: ‘50’
fetchThreads Fetch threads	The number of threads to use during fetching. The default is 5. type: `integer` default value: ‘5’
fetchDelayMS Fetch delay	Number of milliseconds to wait between fetch requests. The default is 0. This property can be used to throttle a crawl if necessary. type: `integer` default value: ‘0’
fetchDelayMSPerHost Fetch delay per host	If true, the 'Fetch delay (ms)' property will be applied for each host. type: `boolean` default value: ‘false’

Property	Description
emitThreads Emit threads	The number of threads used to send documents from the connector to the index pipeline. The default is 5. type: `integer` default value: ‘5’
failFastOnStartLinkFailure Fail crawl if start links are invalid	If true, when Fusion cannot connect to any of the provided start links, the crawl is stopped and an exception logged. type: `boolean` default value: ‘true’
retryEmit Retry emits	Set to true for emit batch failures to be retried on a document-by-document basis. type: `boolean` default value: ‘true’

Dedupe

Property	Description
dedupe Dedupe documents	If true, documents will be deduplicated. Deduplication can be done based on an analysis of the content, on the content of a specific field, or by a JavaScript function. If neither a field nor a script are defined, content analysis will be used. type: `boolean` default value: ‘false’
dedupeSaveSignature Save dedupe signature	If true,the signature used for dedupe will be stored in a 'dedupeSignature_s' field. Note this may cause errors about 'immense terms' in that field. type: `boolean` default value: ‘false’

Property	Description
dedupeField Dedupe field	Field to be used for dedupe. Define either a field or a dedupe script, otherwise the full raw content of each document will be used. type: `string`
dedupeScript Dedupe script	Custom javascript to dedupe documents. The script must define a 'genSignature(content)\{}' function, but can use any combination of document fields. The function must return a string. type: `string`

Recrawl Rules

Property	Description
refreshAll Recrawl all items	Set to true to always recrawl all items found in the crawldb. type: `boolean` default value: ‘true’
refreshStartLinks Recrawl start links	Set to true to recrawl items specified in the list of start links. type: `boolean` default value: ‘false’
refreshErrors Recrawl errors	Set to true to recrawl items that failed during the last crawl. type: `boolean` default value: ‘false’
refreshOlderThan Recrawl age	Number of seconds to recrawl items whose last fetched date is longer ago than this value. type: `integer` default value: ‘-1’

Property	Description
refreshIDPrefixes Recrawl ID prefixes	A prefix to recrawl all items whose IDs begin with this value. type: `array` of `string`
refreshIDRegexes Recrawl ID regexes	A regular expression to recrawl all items whose IDs match this pattern. type: `array` of `string`
refreshScript Recrawl script	A JavaScript function ('shouldRefresh()') to customize the items recrawled. type: `string`
forceRefresh Force recrawl	Set to true to recrawl all items even if they have not changed since the last crawl. type: `boolean` default value: 'false'

Crawl History

Property	Description
retainOutlinks Retain links in the crawldb	Set to true for links found during fetching to be stored in the crawldb. type: `boolean` default value: 'true'
aliasExpiration Alias expiration	The number of crawls after which an alias will expire. The default is 1 crawl. type: `integer` default value: '1'
crawlDBType Crawl database type	The type of crawl database to use, in-memory or on-disk. type: `string` default value: 'in-memory' enum: \{ in-memory on-disk }

Property	Description
indexCrawlDBToSolr Index crawl database to Solr	EXPERIMENTAL: Set to true to index the crawl-database into a 'crawldb_' collection in Solr. type: `boolean` default value: ‘false’
reevaluateCrawlDBOnStart Reevaluate crawldb on start?	Reevaluate exisiting crawldb entries for legality on startup? type: `boolean` default value: ‘false’

Field Mapping

Property	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` _ (required)_ : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }

Property	Description
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: 'false'
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: 'false'
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: 'com.lucidworks.connectors.db.impl.MapDbConnectorDb' minLength: 1

General Configuration

Property	Description
pipeline Pipeline ID	The index pipeline used to process documents.

13.16. FTP Connector and Datasource Configuration

Retrieve documents using the File Transfer Protocol (FTP).

The configuration property "url" specifies the protocol (`ftp`), the host address, and the path to crawl. By default, all files linked to from this URL will be processed. There are several configuration properties available to limit the crawl.

13.16.1. Configuration

Property	Description
add_failed_docs Add failed documents	Set to true to add documents even if they partially fail processing. Failed documents will be added with as much metadata as available, but may not include many expected fields. type: `boolean` default value: ‘false’
bounds Crawl bounds	Limits the the crawl to a specific directory sub-tree, hostname or domain. type: `string` default value: ‘tree’ enum: \{ tree host domain none }
collection Collection	Collection documents will be indexed to. type: `string`
commit_on_finish Solr commit on finish	Set to true for a request to be sent to Solr after the last batch has been fetched to commit the documents to the index. type: `boolean` default value: ‘true’

Property	Description
crawl_depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: ‘-1’ exclusiveMinimum: false minimum: -1
crawl_item_timeout Fetch timeout	Time in milliseconds to fetch any individual document. type: `integer` default value: ‘600000’ exclusiveMinimum: true minimum: 0
exclude_paths Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: `array` of `string` default value: [ ]
include_extensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. However, files with possible matching MIME types but non-matching file extensions will be skipped. The file extensions should be entered with no dots, with whitespace separating items in a list (e.g., 'pdf zip'). type: `array` of `string`
include_paths Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: `array` of `string`

Property	Description
index_directories Index directories	Set to true to add directories to the index as documents. If set to false, directories will not be added to the index, but they will still be traversed for documents. type: `boolean` default value: ‘false’
max_bytes Maximum file size (bytes)	Maximum size, in bytes, of documents to fetch. type: `integer` default value: ‘10485760’ exclusiveMinimum: false minimum: -1
max_docs Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: ‘-1’ exclusiveMinimum: false minimum: -1
max_threads Fetch threads	The maximum number of threads to use for fetching data. Each thread will create a new connection to the repository, which may make overall throughput faster, but will also require more system resources including CPU and memory. type: `integer` default value: ‘1’

Property	Description
maximum_connections Maximum fetch connections	Maximum number of concurrent connections to the filesystem. A large number of documents could cause a large number of simultaneous connections to the repository and lead to errors or degraded performance. In some cases, reducing this number may help performance issues. type: `integer` default value: ‘1000’
password Password	Password for the user. type: `string`
splitter Split CSV/TSV and archives	Set to true to split CSV/TSV files into a document per row, or to split archives such as .zip, tar.gz, etc., into individual files. type: `object` object attributes: \{ }
url Start link *required*	A starting URI for this datasource. The URI must be fully-qualified, and include the protocol, host, port and path, as appropriate. type: `string` minLength: 1 pattern: .:.
username Username	Username with permissions to access the repository, if necessary. type: `string`
verify_access Validate access	Set to true to require successful connection to the filesystem before saving this datasource. type: `boolean` default value: ‘true’

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` _ (required)_ : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Initial Mappings	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: 'false'
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: 'com.lucidworks.connectors.db.impl.MapDbConnectorDb' minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.17. GitHub Connector and Datasource Configuration

The GitHub connector retrieves data from GitHub repositories using the GitHub REST API.

GitHub repository access may require authentication, in which case GitHub credentials or oAuth token Authentication will be used to perform GitHub requests. See the GitHub Authentication page for more information.

13.17.1. Configuration

Connector-specific Properties

Property	Description
f.blobs Content (blobs)	type: `boolean` default value: ‘false’
f.branches Branches	type: `boolean` default value: ‘false’
f.collaborators Collaborators	type: `boolean` default value: ‘false’
f.commit_comments Commit comments	type: `boolean` default value: ‘false’
f.commit_diffs Commit diffs	type: `boolean` default value: ‘false’
f.commits Commits	type: `boolean` default value: ‘true’
f.github_accept_self_signed_ssl_certificate Accept self signed SSL certificate	type: `boolean` default value: ‘true’
f.github_base_url GitHub Base URL	type: `string` default value: ‘https://github.com’
f.github_enterprise GitHub Enterprise	type: `boolean` default value: ‘false’

Property	Description
f.github_oauth_token GitHub oAuth Token	type: `string`
f.github_password GitHub password	type: `string`
f.github_username GitHub username	type: `string`
f.issue_comments Issue comments	type: `boolean` default value: ‘false’
f.issues Issues	type: `boolean` default value: ‘true’
f.milestones Milestones	type: `boolean` default value: ‘false’
f.pull_request_comments Pull request comments	type: `boolean` default value: ‘false’
f.pull_requests Pull requests	type: `boolean` default value: ‘true’
f.releases Releases	type: `boolean` default value: ‘false’
f.tags Tags	type: `boolean` default value: ‘false’

Limit Documents

Property	Description
depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: '-1'
maxItems Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: '-1'
includeExtensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. type: `array` of `string`
includeRegexes Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: `array` of `string`
excludeExtensions Excluded file extensions	File extensions that should not to be fetched. This will limit this datasource to all extensions except this list. type: `array` of `string`
excludeRegexes Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: `array` of `string`
delete Delete dead URIs	Set to true to remove documents from the index when they can no longer be accessed as unique documents. type: `boolean` default value: 'true'

Property	Description
deleteErrorsAfter Fetch failure allowance	Number of fetch failures to tolerate before removing a document from the index. The default of -1 means no fetch failures will be removed. type: `integer` default value: '-1'

Crawl Performance

Property	Description
chunkSize Fetch batch size	The number of items to batch for each round of fetching. The default is 50. type: `integer` default value: '50'
fetchThreads Fetch threads	The number of threads to use during fetching. The value is 1 when 'enterprise:false', if 'enterprise:true' the threads can be incremented type: `integer` default value: '1'
fetchDelayMS Fetch delay	Number of milliseconds to wait between fetch requests. The default is 0. This property can be used to throttle a crawl if necessary. type: `integer` default value: '0'
emitThreads Emit threads	The number of threads used to send documents from the connector to the index pipeline. The default is 5. type: `integer` default value: '5'

Property	Description
retryEmit Retry emits	Set to true for emit batch failures to be retried on a document-by-document basis. type: `boolean` default value: ‘true’
failFastOnStartLinkFailure Fail crawl if start links are invalid	If true, when Fusion cannot connect to any of the provided start links, the crawl is stopped and an exception logged. type: `boolean` default value: ‘true’

Dedupe

Property	Description
dedupe Dedupe documents	If true, documents will be deduplicated. Deduplication can be done based on an analysis of the content, on the content of a specific field, or by a JavaScript function. If neither a field nor a script are defined, content analysis will be used. type: `boolean` default value: ‘false’
dedupeSaveSignature Save dedupe signature	If true,the signature used for dedupe will be stored in a 'dedupeSignature_s' field. Note this may cause errors about 'immense terms' in that field. type: `boolean` default value: ‘false’
dedupeField Dedupe field	Field to be used for dedupe. Define either a field or a dedupe script, otherwise the full raw content of each document will be used. type: `string`

Property	Description
dedupeScript Dedupe script	Custom javascript to dedupe documents. The script must define a 'genSignature(content)\{}' function, but can use any combination of document fields. The function must return a string. type: `string`

Recrawl Rules

Property	Description
refreshAll Recrawl all items	Set to true to always recrawl all items found in the crawldb. type: `boolean` default value: 'true'
refreshStartLinks Recrawl start links	Set to true to recrawl items specified in the list of start links. type: `boolean` default value: 'false'
refreshErrors Recrawl errors	Set to true to recrawl items that failed during the last crawl. type: `boolean` default value: 'false'
refreshOlderThan Recrawl age	Number of seconds to recrawl items whose last fetched date is longer ago than this value. type: `integer` default value: '-1'
refreshIDPrefixes Recrawl ID prefixes	A prefix to recrawl all items whose IDs begin with this value. type: `array` of `string`

Property	Description
refreshIDRegexes Recrawl ID regexes	A regular expression to recrawl all items whose IDs match this pattern. type: `array` of `string`
refreshScript Recrawl script	A JavaScript function ('shouldRefresh()') to customize the items recrawled. type: `string`
forceRefresh Force recrawl	Set to true to recrawl all items even if they have not changed since the last crawl. type: `boolean` default value: 'false'

Crawl History

Property	Description
crawlDBType Crawl database type	The type of crawl database to use, in-memory or on-disk. type: `string` default value: 'in-memory' enum: \{ in-memory on-disk }
indexCrawlDBToSolr Index crawl database to Solr	EXPERIMENTAL: Set to true to index the crawl-database into a 'crawldb_' collection in Solr. type: `boolean` default value: 'false'

Field Mapping

Property	Description

mappings
Field mappings
	List of mapping rules
	type: `array` of `object`
	object attributes: \{
	`operation` : \{
	display name: Operation
	type: `string`
	default value: ‘copy’
	description : The type of mapping to perform: move, copy, delete, add, set, or keep
	enum: \{ copy move delete set add keep }
	`} +`
	`` `source` _ (required)_ : \{ + ``
	`display name: Source Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped. +`
	`} +`
	`` `target` : \{ + ``
	`display name: Target Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped to. +`
	`} +`
	`}`

reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Property	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: 'false'
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: 'com.lucidworks.connectors.db.impl.MapDbConnectorDb' minLength: 1

General Configuration

Property	Description
pipeline Pipeline ID	The index pipeline used to process documents.

13.18. Google Drive Connector and Datasource Configuration

The Google Drive connector is used to index the documents in a Google Drive account.

13.18.1. Google Drive authentication

There are two methods of Google Drive authentication for Fusion:

- authentication for access to site-wide documents

  Use a Google account with admin-level access rights to configure access to all shared documents owned by the users in your organization.

- authentication for access to per-user documents

  This type of authentication gives you access to your own documents in Google Drive. Admin-level access rights are not required.

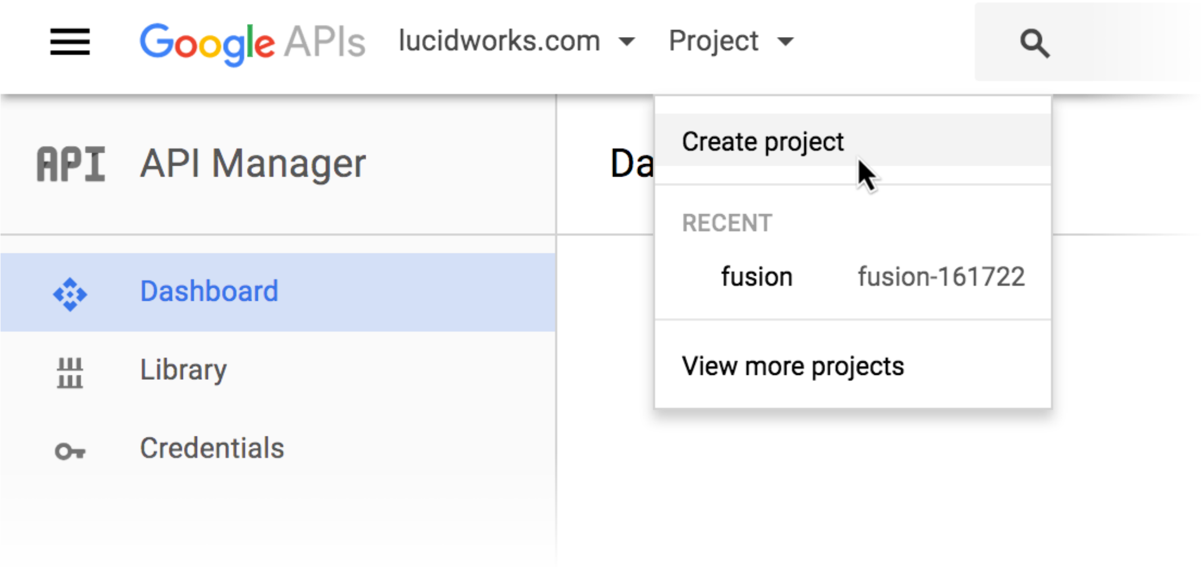
In both cases, you’ll get a client ID, client secret, and refresh token from Google. These become part of your datasource configuration in Fusion.

Authentication for access to site-wide documents

In order to access all the shared documents by users in your organization, you must configure the Google Drive API and the Admin SDK. See the instructions in the knowledge base.

How to configure authentication for access to site-side documents

1. Log in to Google as a user with *admin-level access rights*.
2. Go to https://console.developers.google.com/.
3. Create a Google project for Fusion:
4. In the upper left, open the **Project** menu and select **Create Project**:

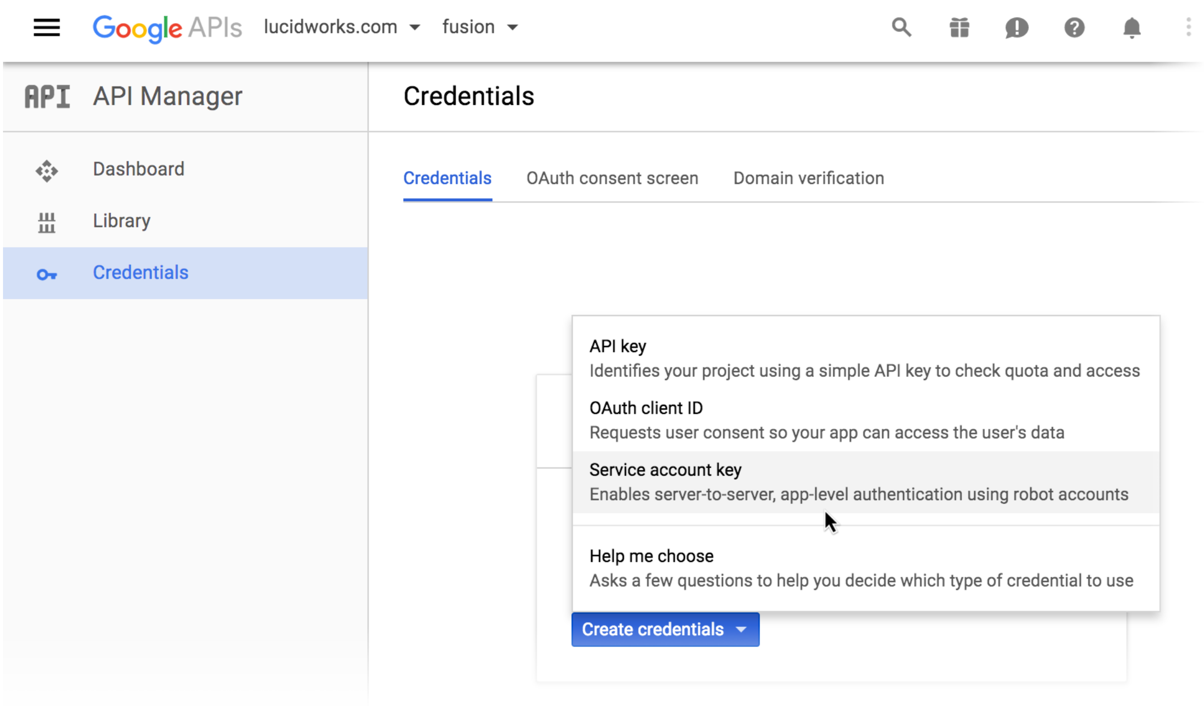


5. Enter a new project name, such as "fusion".

6. Click **Create**.
7. In the new project, click **Enable API**.
8. Under "Google Apps APIs", click **Drive API**.
9. Click **Enable**.

   Google may prompt you to create credentials. Do not create credentials here; we’ll do that a few steps later.

10. Click **Library**, then **Admin SDK**.
11. Click **Enable**.
12. Create a service account key:
13. Navigate to **Credentials** > **Create Credentials** > **Service account key**:



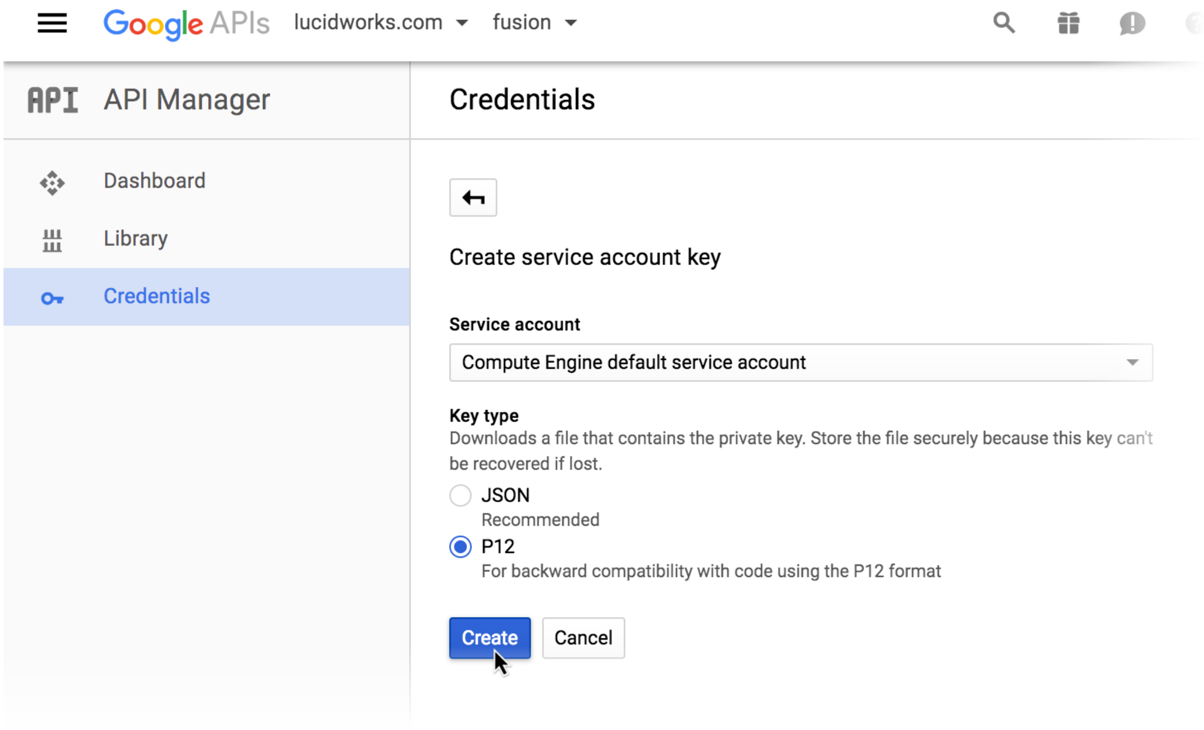
API key
Identifies your project using a simple API key to check quota and access

OAuth client ID
Requests user consent so your app can access the user's data

Service account key
Enables server-to-server, app-level authentication using robot accounts

Help me choose
Asks a few questions to help you decide which type of credential to use

14. From the **Service account** list, select **Compute Engine default service account**.
15. Under "Key type", select **P12**.
16. Click **Create**.



Credentials

Create service account key

Service account

Compute Engine default service account

Key type

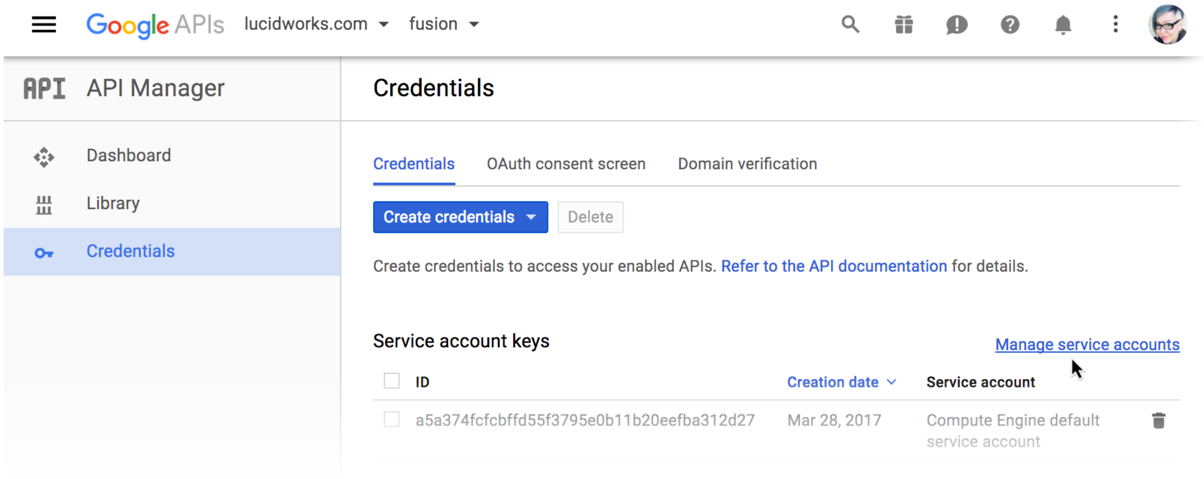
Downloads a file that contains the private key. Store the file securely because this key can't be recovered if lost.

○ JSON
Recommended

● P12
For backward compatibility with code using the P12 format

A new private key downloads automatically to your local drive. Google prompts you to store it securely and save the displayed password. The key and password will not be provided to you again.

17. Create a service account:
18. Click **Manage service accounts**:



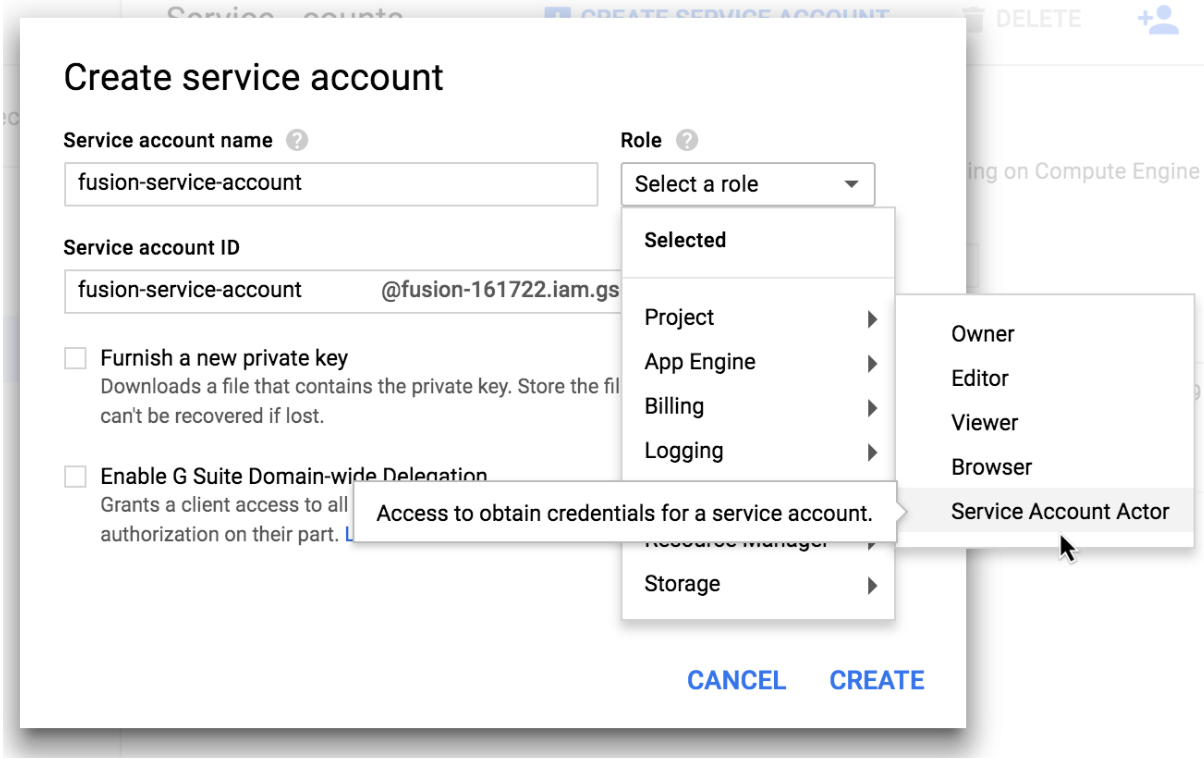
Credentials

Create credentials to access your enabled APIs. Refer to the API documentation for details.

Service account keys

□ ID	Creation date	Service account
□ a5a374fcfcbffd55f3795e0b11b20eefba312d27	Mar 28, 2017	Compute Engine default service account

19. Click **Create Service Account**.
20. Enter a service account name, such as "fusion-service-account".
21. From the **Role** list, select **Project** > **Service account actor**:



Create service account

Service account name: fusion-service-account

Role: Select a role

Service account ID: fusion-service-account @fusion-161722.iam.gs

□ Furnish a new private key
Downloads a file that contains the private key. Store the fil
can't be recovered if lost.

□ Enable G Suite Domain-wide Delegation
Grants a client access to all
authorization on their part. L

Access to obtain credentials for a service account.

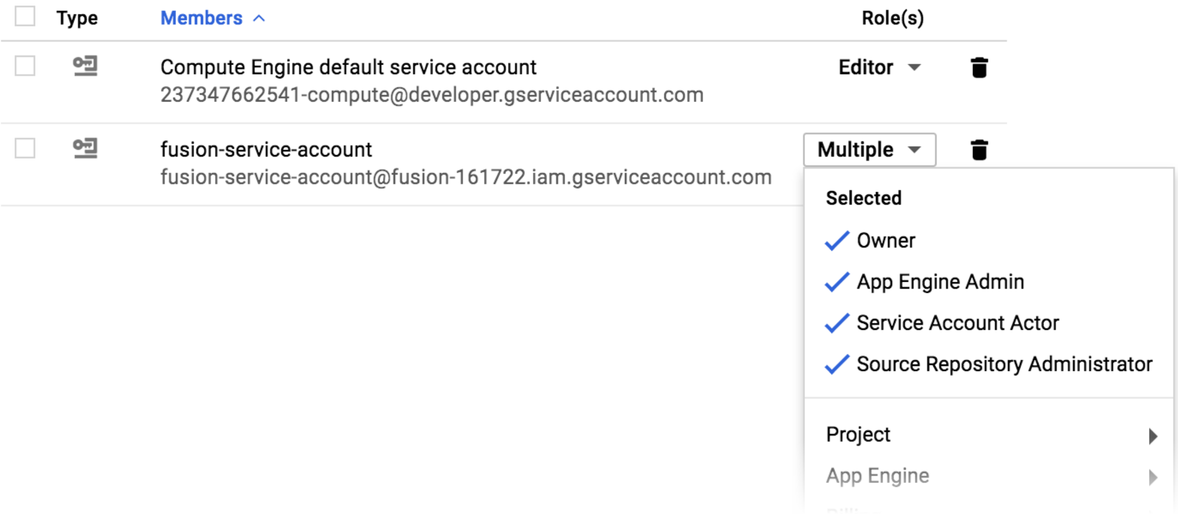
22. Select **Enable G Suite Domain-wide Delegation**.
23. Enter an arbitrary name under **Product name for the consent screen**.
24. Click **Create**.

    Google displays the list of service accounts.

25. Next to the "fusion-service-account", click **View Client ID**.

    You may need to scroll to the right in order to see this link.

26. Copy the client ID and service account name. Save them in a separate location.
27. Click the menu in the upper left and select **IAM & Admin**.
28. Next to the "fusion-service-account" project, select its permissions as shown below:



□	Type	Members ^	Role(s)
□		Compute Engine default service account 237347662541-compute@developer.gserviceaccount.com	Editor
□		fusion-service-account fusion-service-account@fusion-161722.iam.gserviceaccount.com	Multiple

Selected
✓ Owner
✓ App Engine Admin
✓ Service Account Actor
✓ Source Repository Administrator

29. Go to https://admin.google.com.
30. Navigate to **Admin Console** > **Security**.

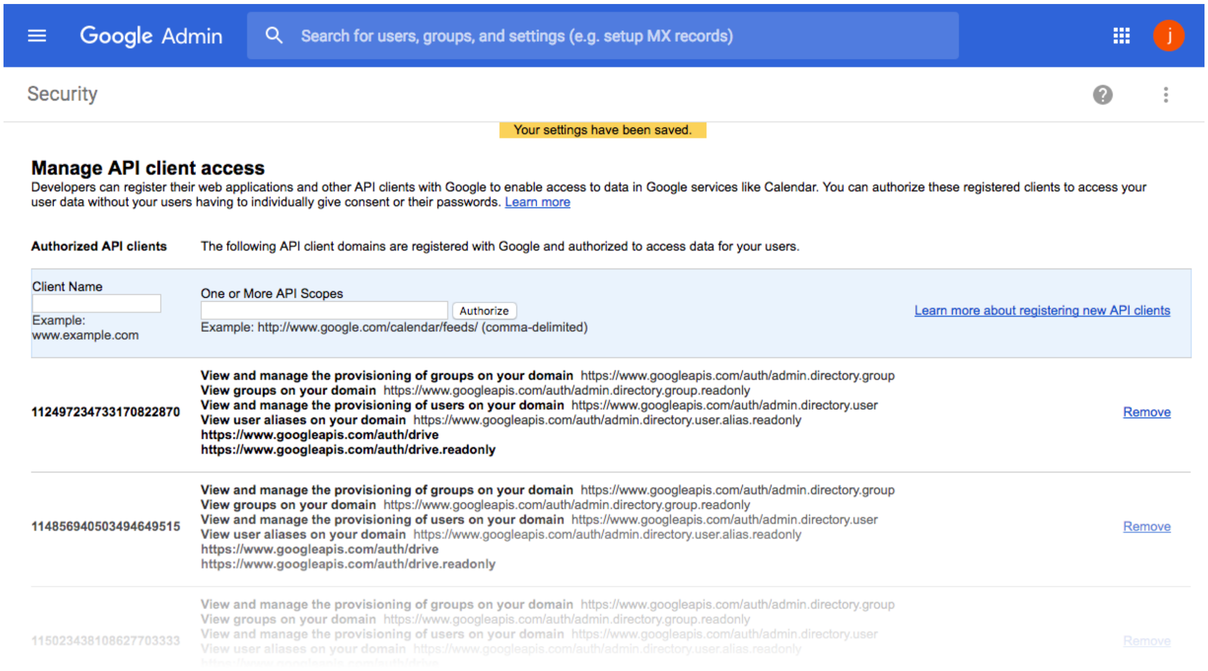
    Remember, you must be logged in as a user with admin-level access rights.

31. Navigate to **Show more** > **Advanced settings** > **Manage API client access**.
32. Create a new API client:
    a. In the **Client Name** field, enter the client ID from your service account (above).
    b. In the **One or More API Scopes** field, enter the following:

    ```
    https://www.googleapis.com/auth/admin.directory.group,https://www.googleapis.com/auth/admin.directory.group.readonly,https://www.googleapis.com/auth/admin.directory.user,https://www.googleapis.com/auth/admin.directory.user.alias.readonly,https://www.googleapis.com/auth/drive,https://www.googleapis.com/auth/drive.readonly
    ```

    c. Click **Authorize**.

    The new API client authorization appears in the list:



Google Admin
Search for users, groups, and settings (e.g. setup MX records)
j
Security

Your settings have been saved.

Manage API client access

Developers can register their web applications and other API clients with Google to enable access to data in Google services like Calendar. You can authorize these registered clients to access your user data without your users having to individually give consent or their passwords. Learn more

Authorized API clients The following API client domains are registered with Google and authorized to access data for your users.

Client Name
Example: www.example.com

One or More API Scopes [Authorize]
Example: http://www.google.com/calendar/feeds/ (comma-delimited)

Learn more about registering new API clients

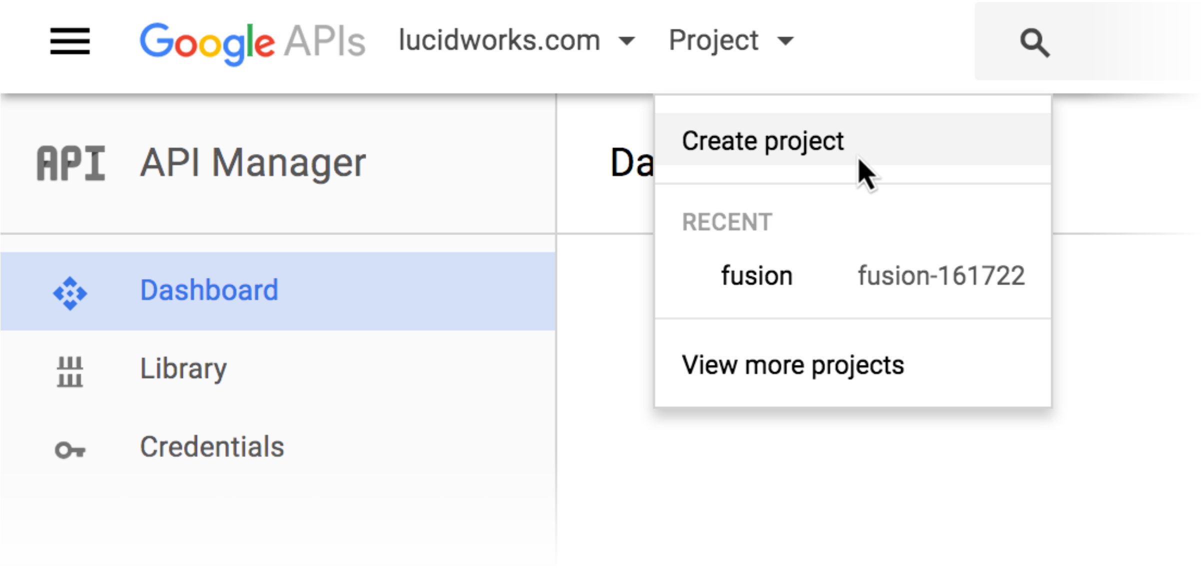
112497234733170822870	**View and manage the provisioning of groups on your domain** https://www.googleapis.com/auth/admin.directory.group **View groups on your domain** https://www.googleapis.com/auth/admin.directory.group.readonly **View and manage the provisioning of users on your domain** https://www.googleapis.com/auth/admin.directory.user **View user aliases on your domain** https://www.googleapis.com/auth/admin.directory.user.alias.readonly **https://www.googleapis.com/auth/drive** **https://www.googleapis.com/auth/drive.readonly**	Remove
114856940503494649515	**View and manage the provisioning of groups on your domain** https://www.googleapis.com/auth/admin.directory.group **View groups on your domain** https://www.googleapis.com/auth/admin.directory.group.readonly **View and manage the provisioning of users on your domain** https://www.googleapis.com/auth/admin.directory.user **View user aliases on your domain** https://www.googleapis.com/auth/admin.directory.user.alias.readonly **https://www.googleapis.com/auth/drive** **https://www.googleapis.com/auth/drive.readonly**	Remove
115023438108627703333	**View and manage the provisioning of groups on your domain** https://www.googleapis.com/auth/admin.directory.group **View groups on your domain** https://www.googleapis.com/auth/admin.directory.group.readonly **View and manage the provisioning of users on your domain** https://www.googleapis.com/auth/admin.directory.user **View user aliases on your domain** https://www.googleapis.com/auth/admin.directory.user.alias.readonly **https://www.googleapis.com/auth/drive**	Remove

Authentication for access to per-user documents

These instructions allow you to configure Google to allow Fusion to crawl a specific user's Google Drive, including documents that other users have shared with them.

How to configure authentication for access to per-user documents

1. Log in to Google as a user with *admin-level access rights*.
2. Go to https://console.developers.google.com/.
3. Create a Google project for Fusion:
4. In the upper left, open the **Project** menu and select **Create Project**:



5. Enter a new project name, such as "fusion".
6. Click **Create**.
7. Create the client ID and client secret:
8. In the new project, click **Enable API**.
9. Under "Google Apps APIs", click **Drive API**.
10. Click **Enable**.

    Google may prompt you to create credentials, if this is the first time you've enabled this API.

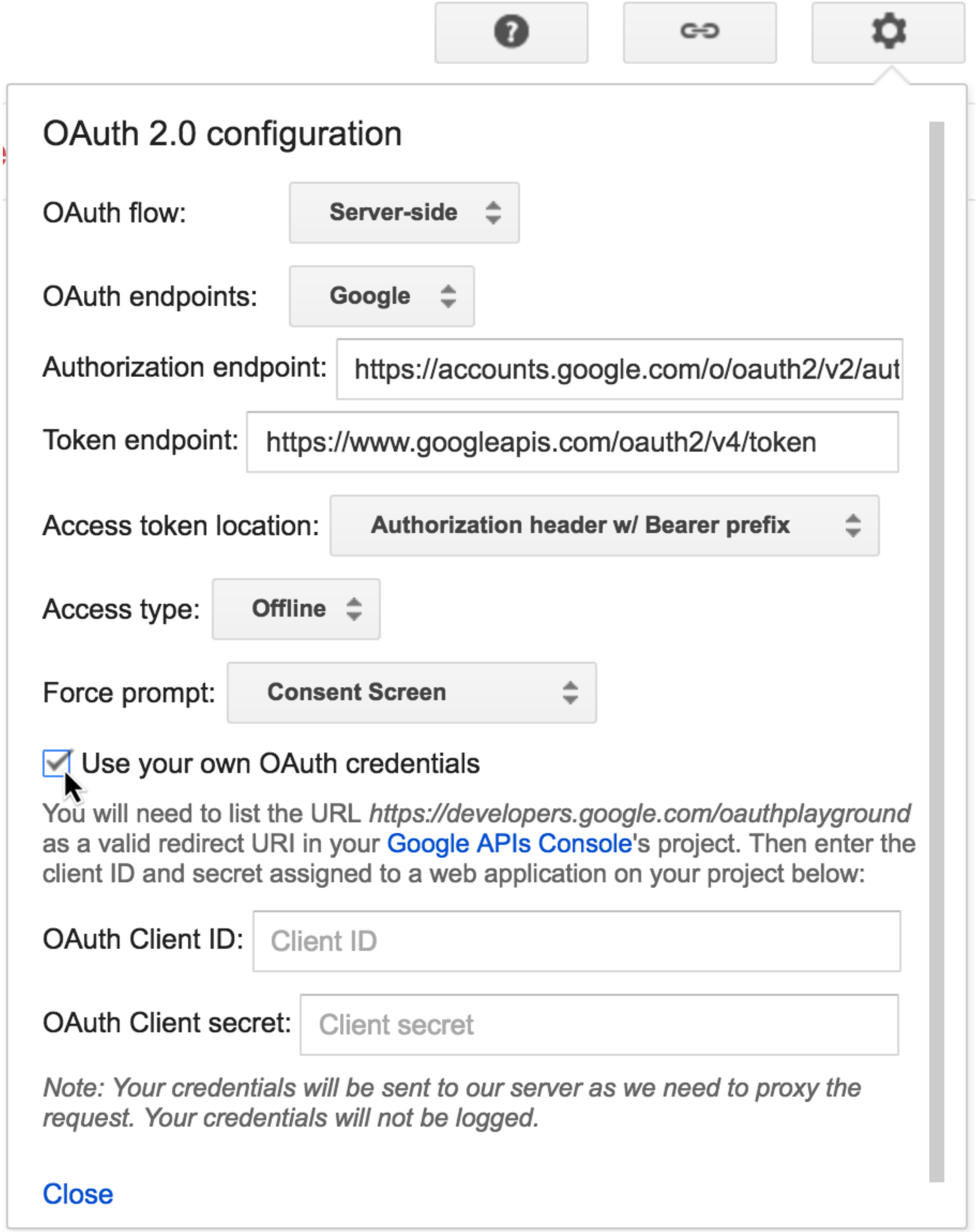
11. Click **Credentials**, then **Create Credentials** > **Oauth client ID**.
12. Select **Web application**.
13. Enter a name for this Web application, such as "Fusion search".
14. In the **Authorized Javascript origins** field, enter "https://developers.google.com".
15. In the **Authorized redirect URIs** field, enter "https://developers.google.com/oauthplayground" and press Return on your keyboard.
16. In the **Authorized redirect URIs** field, enter "http://<fusion-host>:8764/oauth-redirect", specifying the hostname of your Fusion instance.
17. Click **Create**.

    Google displays the new client ID and client secret.

18. Copy the client ID and client secret. Save them in a separate location.
19. Click **OK**.
20. Go to https://developers.google.com/oauthplayground/.
21. In the upper right, click the gear icon.

    The OAuth 2.0 configuration window opens.

22. Select **Use your own OAuth credentials**.



OAuth 2.0 configuration

OAuth flow: Server-side

OAuth endpoints: Google

Authorization endpoint: https://accounts.google.com/o/oauth2/v2/aut

Token endpoint: https://www.googleapis.com/oauth2/v4/token

Access token location: Authorization header w/ Bearer prefix

Access type: Offline

Force prompt: Consent Screen

☑ Use your own OAuth credentials

You will need to list the URL *https://developers.google.com/oauthplayground* as a valid redirect URI in your Google APIs Console's project. Then enter the client ID and secret assigned to a web application on your project below:

OAuth Client ID: Client ID

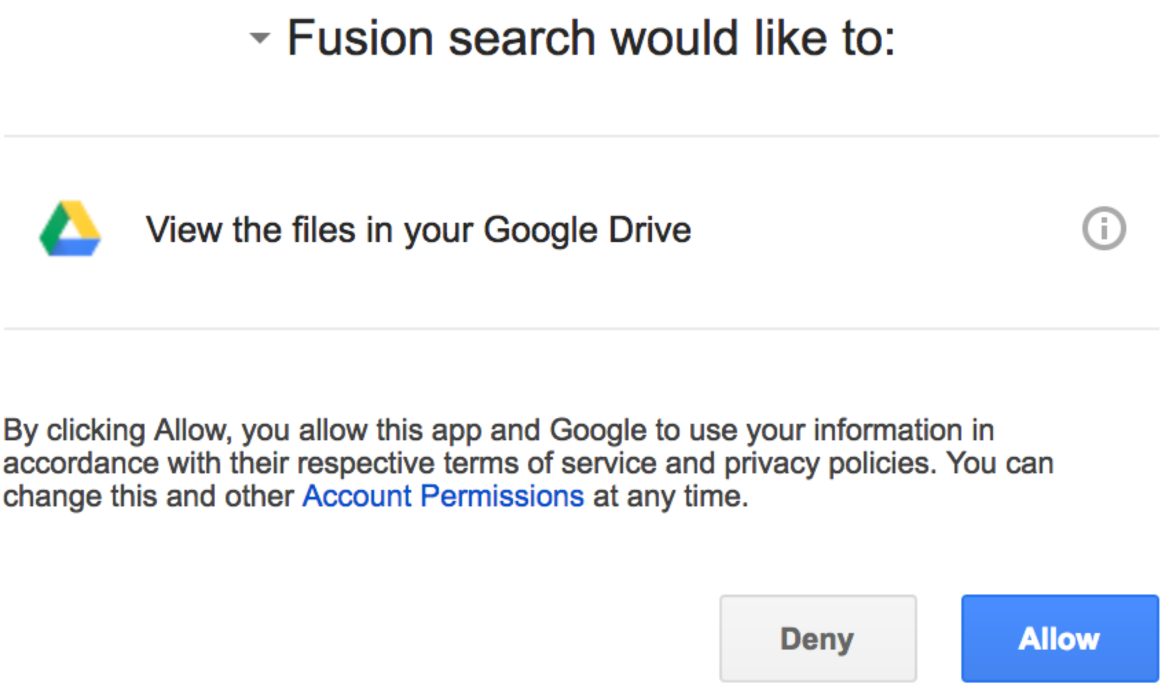
OAuth Client secret: Client secret

Note: Your credentials will be sent to our server as we need to proxy the request. Your credentials will not be logged.

Close

23. Enter your client ID and client secret.
24. Click **Close**.
25. Add the credentials to the datasource configuration in the Fusion UI:
26. In the Google Drive datasource configuration panel, enter a string for the **Datasource ID**.
27. Enter the Google client ID and client secret.
28. Click **Get Refresh Token**.

A new browser window opens, and Google prompts you for permission to access the documents:



Fusion search would like to:

View the files in your Google Drive

By clicking Allow, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other Account Permissions at any time.

29. Click **Allow**.

    Fusion automatically populates the **Google Drive Oauth Refresh Token** field.

30. In the **Startlinks** field, enter a starting URL to which this user has access.

    See below for details about the format for this value.

31. Click **Save**.

13.18.2. Configuration

The **StartLinks** values for this connector must be one of the following:

- `root` to crawl the whole organization or per-user drive.
- A folder ID.

  For example, if your folder's URL is `https://drive.google.com/drive/folders/0B1u0p7N096R6MWgma3gwUj4j` then the start link in the connector configuration should be `0B1u0p7N096R6MWgma3gwUj4j`.

- A document ID.

  For example, if your document's URL is `https://docs.google.com/document/d/10HRr5gD0OetzgEL9fsyxryf_AfiKqDQ8cn12YXQ/edit` then the start link in the connector configuration should be `10HRr5gD0OetzgEL9fsyxryf_AfiKqDQ8cn12YXQ`.

Connector-specific Properties

Property	Description
f.addFileMetadata Add file metadata	Set to true to add information about documents found in the filesystem to the document, such as document owner, group, or ACL permissions. type: `boolean` default value: ‘true’
f.fs.clientID Google User Account OAuth Client ID	For Google Account configuration only - Google OAuth Client ID for a registered application with access to the Drive API. type: `string`
f.fs.clientSecret Google User Account OAuth Client Secret	For Google Account configuration only - Google OAuth client secret for the registered application. type: `string`
f.fs.mime_type_excludes Mime Type Excludes	A comma-separated list of the Mime types to exclude from this crawl. This is only used if the "Mime Type Includes" field is empty. type: `string` default value: ‘image/jpeg’
f.fs.mime_type_includes Mime Type Includes	A comma-separated list of the Mime types to include in this crawl. Includes supercede excludes. type: `string`
f.fs.refreshToken Google User Account OAuth Refresh Token	For Google Account configuration only - OAuth Refresh Token to allow re-authorization of the connector to the Drive API. type: `string`
f.fs.serviceAccountEmail Service Account Email	For Service Account configuration only - NOTE: Required only when 'Apply Group Security Filtering' is checked. This service account email must be assigned to your Google project as a Service Actor in the console. It must have ability to list groups for a user, list users, and read google drive content. type: `string`

Property	Description
f.fs.serviceAccountId Service Account ID	For Service Account configuration only - specifies the Service Account ID to use to connect Fusion to Google Drive. type: `string`
f.fs.serviceAccountPrivateKeyFile Service Account P12 Private Key File	For Service Account configuration only - specifies the private key file in P12 private key format. type: `string`
f.fs.serviceAccountPrivateKeyFilePassword Service Account P12 Private Key File Password	For Service Account configuration only - specifies the private P12 key's password. type: `string` default value: 'notasecret'
f.fs.userExcludeList User Exclude List	By default, all users' files are crawled. Enter a comma-separated list of usernames (email addresses) to exclude from the crawl. If the last character of the email address is an '*', then all email addresses that start with that prefix will be excluded. type: `string`
f.fs.userSearchQuery User Search Query	By default, all users' files are crawled. If you specify this field, you can specify a query to specify the users who will get crawled. Example: email=nick.jones* type: `string`
f.maxSizeBytes Maximum file size (bytes)	Maximum size, in bytes, of documents to fetch. type: `integer` default value: '4194304'
f.minSizeBytes Minimum file size (bytes)	Minimum size, in bytes, of documents to fetch. type: `integer` default value: '0'

Security Trimming

Property	Description
enable_security_trimming Enable Security Trimming	type: `object` object attributes: \{ }

Limit Documents

Property	Description
f.maxSizeBytes Maximum file size (bytes)	Maximum size, in bytes, of documents to fetch. type: `integer` default value: '4194304'
f.minSizeBytes Minimum file size (bytes)	Minimum size, in bytes, of documents to fetch. type: `integer` default value: '0'
f.addFileMetadata Add file metadata	Set to true to add information about documents found in the filesystem to the document, such as document owner, group, or ACL permissions. type: `boolean` default value: 'true'
depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: '-1'
maxItems Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: '-1'

Property	Description
delete Delete dead URIs	Set to true to remove documents from the index when they can no longer be accessed as unique documents. type: `boolean` default value: ‘true’
deleteErrorsAfter Fetch failure allowance	Number of fetch failures to tolerate before removing a document from the index. The default of -1 means no fetch failures will be removed. type: `integer` default value: ‘-1’

Crawl Performance

Property	Description
chunkSize Fetch batch size	The number of items to batch for each round of fetching. The default is 50. type: `integer` default value: ‘50’
fetchThreads Fetch threads	The number of threads to use during fetching. The default is 5. type: `integer` default value: ‘5’
fetchDelayMS Fetch delay	Number of milliseconds to wait between fetch requests. The default is 0. This property can be used to throttle a crawl if necessary. type: `integer` default value: ‘0’
emitThreads Emit threads	The number of threads used to send documents from the connector to the index pipeline. The default is 5. type: `integer` default value: ‘5’

Property	Description
retryEmit Retry emits	Set to true for emit batch failures to be retried on a document-by-document basis. type: `boolean` default value: ‘true’
failFastOnStartLinkFailure Fail crawl if start links are invalid	If true, when Fusion cannot connect to any of the provided start links, the crawl is stopped and an exception logged. type: `boolean` default value: ‘true’

Dedupe

Property	Description
dedupe Dedupe documents	If true, documents will be deduplicated. Deduplication can be done based on an analysis of the content, on the content of a specific field, or by a JavaScript function. If neither a field nor a script are defined, content analysis will be used. type: `boolean` default value: ‘false’
dedupeSaveSignature Save dedupe signature	If true,the signature used for dedupe will be stored in a 'dedupeSignature_s' field. Note this may cause errors about 'immense terms' in that field. type: `boolean` default value: ‘false’
dedupeField Dedupe field	Field to be used for dedupe. Define either a field or a dedupe script, otherwise the full raw content of each document will be used. type: `string`

Property	Description
dedupeScript Dedupe script	Custom javascript to dedupe documents. The script must define a 'genSignature(content)\{}' function, but can use any combination of document fields. The function must return a string. type: `string`

Recrawl Rules

Property	Description
refreshAll Recrawl all items	Set to true to always recrawl all items found in the crawldb. type: `boolean` default value: 'true'
refreshStartLinks Recrawl start links	Set to true to recrawl items specified in the list of start links. type: `boolean` default value: 'false'
refreshErrors Recrawl errors	Set to true to recrawl items that failed during the last crawl. type: `boolean` default value: 'false'
refreshOlderThan Recrawl age	Number of seconds to recrawl items whose last fetched date is longer ago than this value. type: `integer` default value: '-1'
refreshIDPrefixes Recrawl ID prefixes	A prefix to recrawl all items whose IDs begin with this value. type: `array` of `string`

Property	Description
refreshIDRegexes Recrawl ID regexes	A regular expression to recrawl all items whose IDs match this pattern. type: `array` of `string`
refreshScript Recrawl script	A JavaScript function ('shouldRefresh()') to customize the items recrawled. type: `string`
forceRefresh Force recrawl	Set to true to recrawl all items even if they have not changed since the last crawl. type: `boolean` default value: 'false'

Crawl History

Property	Description
retainOutlinks Retain links in the crawldb	Set to true for links found during fetching to be stored in the crawldb. type: `boolean` default value: 'true'
aliasExpiration Alias expiration	The number of crawls after which an alias will expire. The default is 1 crawl. type: `integer` default value: '1'
crawlDBType Crawl database type	The type of crawl database to use, in-memory or on-disk. type: `string` default value: 'in-memory' enum: \{ in-memory on-disk }

Property	Description
indexCrawlDBToSolr Index crawl database to Solr	EXPERIMENTAL: Set to true to index the crawl-database into a 'crawldb_' collection in Solr. type: `boolean` default value: ‘false’

Field Mapping

Property	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` _ (required)_ : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Property	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: 'false'
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: 'com.lucidworks.connectors.db.impl.MapDbConnectorDb' minLength: 1

General Configuration

Property	Description
pipeline Pipeline ID	The index pipeline used to process documents.

13.19. HDFS Connector and Datasource Configuration

Hadoop Distributed File System (HDFS). It traverses the Hadoop file system as it would a regular Unix filesystem.

This connector can only be used with the default Hadoop shipped with Solr.

See also the Hadoop connector, a connector for HDFS filesystem which uses MapReduce to distribute the crawl processes. When there is a lot of content to process, the MapReduce-enabled connector will be significantly faster.

13.19.1. Configuration

Property	Description
add_failed_docs Add failed documents	Set to true to add documents even if they partially fail processing. Failed documents will be added with as much metadata as available, but may not include many expected fields. type: `boolean` default value: 'false'
bounds Crawl bounds	Limits the the crawl to a specific directory sub-tree, hostname or domain. type: `string` default value: 'tree' enum: \{ tree host domain none }
collection Collection	Collection documents will be indexed to. type: `string`
commit_on_finish Solr commit on finish	Set to true for a request to be sent to Solr after the last batch has been fetched to commit the documents to the index. type: `boolean` default value: 'true'
converter Converter	Fully-qualified classname for a custom converter to produce valid SolrInputDocuments extracted from Hadoop Sequence or MapReduce files. type: `string`

Property	Description
crawl_depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: ‘-1’ exclusiveMinimum: false minimum: -1
crawl_item_timeout Fetch timeout	Time in milliseconds to fetch any individual document. type: `integer` default value: ‘600000’ exclusiveMinimum: true minimum: 0
exclude_paths Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: `array` of `string` default value: [ ]
include_extensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. However, files with possible matching MIME types but non-matching file extensions will be skipped. The file extensions should be entered with no dots, with whitespace separating items in a list (e.g., 'pdf zip'). type: `array` of `string`
include_paths Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: `array` of `string`

Property	Description
index_directories Index directories	Set to true to add directories to the index as documents. If set to false, directories will not be added to the index, but they will still be traversed for documents. type: `boolean` default value: ‘false’
kerberos_keytab Kerberos keytab	Full path to the Kerberos keytab file. type: `string` default value: '`’
kerberos_user Kerberos principal	Kerberos principal name, i.e., 'username@YOUR-REALM.COM'. type: `string` default value: '`’
max_bytes Maximum file size (bytes)	Maximum size, in bytes, of documents to fetch. type: `integer` default value: ‘10485760’ exclusiveMinimum: false minimum: -1
max_docs Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: ‘-1’ exclusiveMinimum: false minimum: -1

Property	Description
max_threads Fetch threads	The maximum number of threads to use for fetching data. Each thread will create a new connection to the repository, which may make overall throughput faster, but will also require more system resources including CPU and memory. type: `integer` default value: ‘1’
maximum_connections Maximum fetch connections	Maximum number of concurrent connections to the filesystem. A large number of documents could cause a large number of simultaneous connections to the repository and lead to errors or degraded performance. In some cases, reducing this number may help performance issues. type: `integer` default value: ‘1000’
splitter Split CSV/TSV and archives	Set to true to split CSV/TSV files into a document per row, or to split archives such as .zip, tar.gz, etc., into individual files. type: `object` object attributes: \{ }
url Start link *required*	A starting URI for this datasource. The URI must be fully-qualified, and include the protocol, host, port and path, as appropriate. type: `string` minLength: 1 pattern: .:.
verify_access Validate access	Set to true to require successful connection to the filesystem before saving this datasource. type: `boolean` default value: ‘true’

Property	Description
with_kerberos Use Kerberos authentication	Set to true to use Kerberos to authenticate to HDFS for access to the content. type: `boolean` default value: ‘false’

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` _ (required)_ : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

<table>
<tr><th>Initial Mappings</th><th>Description</th></tr>
<tr><td><b>unmapped</b>

Unmapped fields</td><td>If fields do not match any of the field mapping rules, these rules will apply.

type: <code>object</code>

object attributes: \{
 <code>operation</code> : \{
  display name: Operation
  type: <code>string</code>
  default value: ‘copy’
  description : The type of mapping to perform: move, copy, delete, add, set, or keep
  enum: \{ copy move delete set add keep }
<pre>
    } +
  `source` : \{ +
    display name: Source Field +
    type: `string` +
    description : The name of the field to be
mapped. +
    } +
  `target` : \{ +
    display name: Target Field +
    type: `string` +
    description : The name of the field to be
mapped to. +
    } +
</pre> }</td></tr>
</table>

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.20. Hadoop Connector and Datasource Configuration

The Hadoop Connector is a MapReduce-enabled crawler which leverages the scaling qualities of Apache Hadoop.

A Hadoop connector creates a series of MapReduce-enabled jobs which index raw content into Fusion. The Hadoop connector can be run using on of the following Apache Hadoop distributions:

- Apache Hadoop v2.x
- Cloudera CDH v4.x, v5.x
- Hortonworks Data Platform v2.x
- MapR v4.x, v5.x
- Pivotal HD v1.1

The Hadoop connector name is `lucid.hadoop` and for each Hadoop distribution has its own plugin type. All plugin types take the same set of configuration properties.

There is a non-MapReduce enabled connector for HDFS filesystem; see page HDFS Connector and Datasource Configuration for details.

The Hadoop crawlers take full advantage of the scaling abilities of the MapReduce architecture and will use all of the nodes available in the cluster just like any other MapReduce job. This has significant ramifications for performance since it is designed to move a lot of content, in parallel, as fast as possible (depending on the system's capabilities), from its raw state to the Fusion index. The Hadoop crawlers work in three stage:

1. Create one or more SequenceFiles from the raw content. This can be done in one of two ways:
2. If the source files are available in a shared Hadoop filesystem, prepare a list of source files and their locations as a SequenceFile. The raw contents of each file are not processed until step 2.
3. If the source files are not available, prepare a list of source files and the raw content, stored as a BehemothDocument. This process is currently done sequentially and can take a significant amount of time if there is a large number of documents and/or if they are very large.
4. Run a MapReduce job to extract text and metadata from the raw content using Apache Tika. This is similar to the Fusion approach of extracting content from crawled documents, except it is done with MapReduce.
5. Run a MapReduce job to send the extracted content from HDFS to the index pipeline for further processing.

Note:

The first step of the crawl process converts the input content into a SequenceFile. In order to do this, the entire contents of that file must be read into memory so that it can be written out in the SequenceFile. Thus, you should be careful to ensure that the system does not load into memory a file that is larger than the Java heap size of the process. In certain cases, Behemoth can work with existing files such as SequenceFiles to convert them to Behemoth SequenceFiles. Contact Lucidworks for possible alternative approaches.

The processing approach is currently "all or nothing" when it comes to ingesting the raw content and all 3 steps must be completed each time, regardless of whether the raw content hasn't changed. Future versions may allow the crawler to restart from the SequenceFile conversion process. In the meantime, incremental crawling is not supported for this connector.

13.20.1. Hadoop Installation and Configuration

The Connector services must be able to access the Hadoop client in file `$HADOOP_HOME/bin/hadoop`, so it must either be installed on one of the nodes of the Hadoop cluster (such as the `nameNode`), or a client supported by your specific distribution must be installed on the same server as the Connectors. The Hadoop client must be configured properly to access the Hadoop cluster so the crawler is able to access the Hadoop cluster for content processing.

Please note, instructions for setting up any of the supported Hadoop distributions is beyond the scope of this document. We recommend reading one of the many tutorials found online or one of the books on Hadoop.

This connector writes to the `hadoop.tmp.dir and the /tmp` directory in HDFS, so Fusion should be started by a user who has read/write permissions for both.

Permission Issues

Using any flavor of Hadoop, you will need to be aware of the way Hadoop and systems based on Hadoop (such as CDH, MapR, etc.) handle permissions for services that communicate with other nodes.

Hadoop services execute under specific user credentials: a quadruplet consisting of user name, group name, numeric user id, numeric group id. Installations that follow the manual usually use user 'mapr' and group 'mapr' (or similar), with numeric ids assigned by the operating system (e.g., uid=1000, gid=20). When the system is configured to enforce user permissions (which is the default in some systems), any client that connects to Hadoop services has to use a quadruplet that exists on the server. This means that ALL values in this quadruplet must be equal between the client and the server, i.e., an account with the same user, group, uid, and gid must exist on both client and server machines.

When a client attempts to access a resource on Hadoop filesystems (or the JobTracker, which also uses this authentication method) it sends its credentials, which are looked up on the server, and if an exactly matching record is found then those local permissions will be used to determine read/write access. If no such account is found then the user is treated as "other" in the sense of the permission model.

This means that the crawlers for the HDFS data source should be able to crawl Hadoop or MapR filesystems without any authentication, as long as there is a read (and execute for directories) access for "other" users granted on the target resources. Authenticated users will be able to access resources owned by their equivalent account.

However, the Hadoop crawling described on this page require write access to a `/tmp` directory to use as a working directory. In many cases, this directory does not exist, or if it does, it doesn't have write access to "other" (not authenticated) users. Therefore users of these data sources should make sure that there is a `/tmp` directory on the target filesystem that is writable using their local user credentials, be it a recognized user, group, or "other". If a local user is recognized by the server then it's enough to create a `/tmp` directory that is owned by that user. If there is no such user, then the `/tmp` directory must be modified to have write permissions for "other" users. The working directory can be modified to be another directory that can be used for temporary working storage that has the correct permissions.

Configuration for a Kerberos Hadoop Cluster

Kerberos is a system that provides authenticated access for users and services on a network. Instead of sending passwords in plaintext over the network, encrypted passwords are used to generate time-sensitive tickets which are used for authentication. Kerberos uses symmetric-key cryptography and a trusted third party called a Key Distribution Center (KDC) to authenticate users to a suite of network services. When a user authenticates to the KDC, the KDC sends a set of credentials (a ticket) specific to that session back to the user's machine.

To work with a Kerberized Hadoop cluster you must have a set of credentials. These are generated by running the "kinit" program. The datasource can be configured to run this program, in which case, the following information must be specified: the full path to the program, the Kerberos principal name, the location of a keytab file and the name of the file in which to store the ticket.

For more information on Kerberized Hadoop Clusters, see http://www.cloudera.com/content/cloudera/en/documentation/archives/cloudera-manager-4/v4-5-2/Configuring-Hadoop-Security-with-Cloudera-Manager/cmeechs_topic_4_5.html.

13.20.2. Configuration

Property	Description
collection Collection	Collection documents will be indexed to. type: `string`
hadoop_home Hadoop home *required*	Path to the Hadoop home directory where $HADOOP_HOME/bin/hadoop can be found. The connector requires access to either a full Hadoop installation, or a Hadoop client provided by your Hadoop distribution that has been configured to access the Hadoop installation. type: `string` minLength: 1
job_jar Job Jar *required*	Path and name of the Hadoop job jar. Unless you are using a custom job jar, the default provided by Fusion is preferred. type: `string` default value: ‘lucidworks-hadoop-job-2.2.1.jar’ minLength: 1
job_jar_args Job JAR args *required*	Job Jar arguments, including the IngestMapper and other parameters for the Hadoop job. type: `string` minLength: 1
kinit_cache 'kerberos' cache	Full path of 'kerberos' cache. If this path does not exist, it will be created. type: `string` default value: '`’

Property	Description
kinit_cmd 'kinit' command	Full path to the 'kinit' binary. type: `string` default value: ‘kinit’
kinit_keytab 'kerberos' keytab	Full path to the Kerberos keytab file. type: `string` default value: '`’
kinit_user 'kerberos' principal	Kerberos principal name, i.e., username@YOUR-REALM.COM type: `string` default value: '`’
run_kinit Run 'kinit'	If your Hadoop installation requires job requests to authenticate with Kerberos, this option will allow Fusion to run 'kinit' to get a valid ticket. type: `boolean` default value: ‘false’

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: array of object object attributes: \{ operation : \{ display name: Operation type: string default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: boolean default value: ‘false’

Initial Mappings	Description

unmapped
Unmapped fields
	If fields do not match any of the field mapping rules, these rules will apply.
	type: `object`
	object attributes: \{
	`operation` : \{
	display name: Operation
	type: `string`
	default value: ‘copy’
	description : The type of mapping to perform: move, copy, delete, add, set, or keep
	enum: \{ copy move delete set add keep }
	`    } +`
	``  `source` : \{ +``
	`    display name: Source Field +`
	``    type: `string` +``
	`    description : The name of the field to be mapped. +`
	`    } +`
	``  `target` : \{ +``
	`    display name: Target Field +`
	``    type: `string` +``
	`    description : The name of the field to be mapped to. +`
	`    } +`
	`  }`

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.21. JDBC Connector and Datasource Configuration

The JDBC connector fetches documents from a relational database via SQL queries. Under the hood, this connector implements the Solr DataImportHandler (DIH) plugin.

13.21.1. SQL queries for document retrieval

The JDBC connector is configured with the SQL statement used to retrieve a resultset from the database. Each row in the result set is treated as a Solr document. This statement is specified as the value of the required property `sql_select_statement`.

The column names will be used as Solr document field names; use the SQL "AS" keyword to rename column names as needed. Column and table names should be treated as if they are case-insentive, even though some databases allow use of mixed case names. All Solr documents must have a unique key, which is the Fusion "id" field. Therefore, the results set must contain a column "id" which can be used as a unique key for the resulting document.

```
SELECT customer_id AS id, * from customers
```

Delta queries

Delta queries provide incremental updates to the contents of a collection by indexing only those records in the database which have been changed since the database was last indexed by this connector. The SQL statement is specified as the value of the property `delta_sql_query`.

Delta queries select only primary key values, therefore, the query must use the primary key and it must also have a "WHERE" clause which specifies a "last_modified" condition as follows:

```
SELECT customer_id AS id from customers WHERE last_modified > $
```

The dollar-sign character '$' is required; it holds the last successful import time from the database.

Nested queries

Nested queries are used to index information which is stored in the database across a series of tables where the is a one-to-many or many-to-many reletionship between them. This statement is specified as the value of the property `nested_queries`.

A nested query is used in conjunction with the SQL query specified by the `sql_select_statement` statement. The dollar-sign character '$' specifies the primary key in the resultset retrieved by the `sql_select_statement` statement.

The following example shows the pair of query, nested query statements used to index list of tags assigned to documents:

```
SELECT id FROM document
SELECT tag FROM tag INNER JOIN document_tag ON document_tag.tag_id=tag.id WHERE document_tag.doc_id=$
```

13.21.2. Uploading a JDBC driver

The Connector JDBC API allows you to upload JDBC drivers. It can also be used to find the driver class name needed for

JDBC datasource configuration.

The URL used to upload a .jar file containing a JDBC driver via a POST request is:

```
/api/apollo/connectors/plugins/lucid.jdbc/resources/jdbc?collection=<collectionName>
```

where <collectionName> is the collection for which the driver will be used.

Here is an example of using the REST API to upload a Postgres driver:

```
curl -u user:pass -X POST --form file=@/path/postgresql-9.3-1101.jdbc4.jar
http://localhost:8764/api/apollo/connectors/plugins/lucid.jdbc/resources/jdbc?collection=docs
```

Drivers are uploaded to a directory named `fusion/data/connectors/lucid.jdbc/jdbc/<collection>`, where <collection> is the collection name used during the upload.

You can then use GET requests with the same endpoint to validate that the driver has been properly loaded and is available to Fusion.

```
$curl -u user:pass http://localhost:8764/api/apollo/connectors/plugins/lucid.jdbc/resources?collection=docs

[ {
  "name" : "postgresql-9.3-1101.jdbc4.jar",
  "properties" : {
    "type" : "jar"
  }
}, {
  "name" : "org.postgresql.Driver",
  "properties" : {
    "type" : "class"
  }
} ]
```

Your database driver version should be the the most recent one that is compatible with your database and the Java version used to run Fusion.

13.21.3. Indexing binary data

The JDBC connector in Fusion does not automatically discover and index binary data you may have stored in your database (such as PDF files). However, you can configure Fusion to recognize and extract binary data correctly by modifying the datasource configuration file. This file is created when the datasource is first run, and then it is created in `fusion/data/connectors/lucid.jdbc/datasources/ datasourceID/conf`. The name of the file will include the name of the datasource, as in `dataconfig_datasourceName.xml`. If you are familiar with Solr's DIH, you will recognize this as a standard dataconfig.xml file.

Follow these steps to modify the configuration file:

1. Add a `name` attribute for the database containing your binary data to the `dataSource` entry.
2. Set the `convertType` attribute for the `dataSource` to `false`. This prevents Fusion from treating binary data as strings.
3. Add a `FieldStreamDataSource` to stream the binary data to the Tika entity processor.

4. Specify the `dataSource` name in the `root` entity.
5. Add an entity for your `FieldStreamDataSource` using the `TikaEntityProcessor` to take the binary data from the `FieldStreamDataSource`, parse it, and specify a field for storing the processed data.
6. Reload the Solr core to apply your configuration changes.

13.21.4. Troubleshooting

When using the JDBC connector, it is recommended that you work closely with your database administrator to formulate efficient and robust queries.

One source of possible problems is the driver being used. In some cases, indexing may fail due to problems with the driver, in particular older versions of Oracle's JDBC driver. If you have checked that your connection information is correct and your database is allowing the connection, you may want to research if there are any known bugs with the driver you are using.

With Oracle databases, note that column names not enclosed in double-quotes are converted to upper-case, but Solr field names are case sensitive. If your column-to-field mapping is not happening properly, check your SQL statement for any lower-case names not enclosed in double-quotes.

Dates can also be problematic. Solr has a different date format than many relational databases. If you want date and time fields to be indexed properly, you may need to convert database dates into the proper format using date/string convert functions. In Oracle this is the TO_CHAR function; in Microsoft SQL, this is the DATEPART function.

In MySQL databases, dates are allowed to be 0-strings, such as 0000-00-00, which is not acceptable to JDBC. If you have legacy date data you may need to add the query parameter "zeroDateTimeBehavior=convertToNull" to your JDBC request string, as in ` jdbc:mysql://localhost/myDatabase?zeroDateTimeBehavior=convertToNull `. This will convert the zero-string dates to null values that can be added to the index.

Finally, database timeouts are another problematic area. There are several possible solutions to this, from increasing the timeout in the JDBC request (with "netTimeoutForStreamingResults"), altering the SQL statement to page the results, or dumping the records to CSV and indexing them with another connector.

13.21.5. Configuration

Property	Description
clean_in_full_import_mode Clean in full import mode	Clean old records when doing a full import. This will remove all records from the index before re-indexing. type: `boolean` default value: 'true'
collection Collection	Collection documents will be indexed to. type: `string`

Property	Description
commit_on_finish Solr commit on finish	Set to true for a request to be sent to Solr after the last batch has been fetched to commit the documents to the index. type: `boolean` default value: ‘true’
delta_import_query Delta Import SQL Query	A SQL statement to retrieve the delta records. If not defined, the original SQL Statement will be used to retrieve delta records, which may be error prone in the case of complex statements. If that occurs, you can define a secondary SQL satement that will only be used for delta imports. type: `string`
delta_sql_query Delta SQL Query	A SQL statement to select the delta (records added or changed) since the last run of the datasource. For example, 'SELECT id FROM customers WHERE last_modified > $'. The $ indicates the date and time of the last import. type: `string`
driver JDBC Driver *required*	The class name of the JDBC driver to use to connect to the database. Only JDBC4 drivers will appear in this list. If a JDBC3 driver has been uploaded but does not appear in the list, it may be possible to manually enter the class name. type: `string` minLength: 1
fetch_size JDBC fetch size	The number of documents to retrieve per batch. type: `integer`
max_docs Max Documents	The maximum number of documents to crawl.Use -1 to index all documents found. type: `integer` default value: ‘-1’

Property	Description
nested_queries Nested queries	type: `array` of `string`
password Password *required*	The password of the account used for authentication and data access. type: `string`
primary_key Primary Key	The column name of the primary key for the table. type: `string`
sql_select_statement SQL Statement *required*	A SQL SELECT statement to choose the records to be retrieved. type: `string`
url URL *required*	A URL to the database, starting with jdbc and vendor name, e.g., 'jdbc:mysql://localhost/test' type: `string` minLength: 1
username Username *required*	The username of a database account used for authentication and data access. type: `string`
verify_access Validate access	Set to true to require successful connection to the filesystem before saving this datasource. type: `boolean` default value: ‘true’

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` _ (required)_ : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Initial Mappings	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: object object attributes: \{ operation : \{ display name: Operation type: string default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: boolean default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: boolean default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: 'false'
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: 'com.lucidworks.connectors.db.impl.MapDbConnectorDb' minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.22. JIRA Connector and Datasource Configuration

The JIRA connector retrieves data from a instance of Atlassian's JIRA issue tracking system.

It uses the JIRA REST API to retrieve the following JIRA elements:

- Projects
- Issues
- Comments
- Worklogs
- Attachments

JIRA access to projects and issues may be restricted to certain users. Access of restricted information requires a JIRA username and password.

The JIRA connector first requests a list of all projects, and for each project, if finds all issues. For each issue, the information on summary, priority, assignee, etc. is retrieved. Worklogs, comments, and attachments are treated as new entries (issue links) and thus are indexed as a new document, not as a component of the issue itself.

13.22.1. Configuration

Connector-specific Properties

Property	Description
f.cacheSize Cache size (number of entries)	Number of entries to cache when making REST requests. type: `integer` default value: '2000'
f.index_attachments Index Attachments	Select to index attachments on issues. type: `boolean` default value: 'true'
f.index_comments Index Comments	Select to index comments on issues. type: `boolean` default value: 'true'
f.index_issues Index Documents	Select to index issues. type: `boolean` default value: 'true'

Property	Description
f.index_projects Index Projects	Select to index project-level information. type: `boolean` default value: ‘false’
f.index_worklogs Index Worklogs	Select to index worklogs on issues. type: `boolean` default value: ‘false’
f.jira_password JIRA password	Password for the adminstrator user, if required. type: `string` default value: '`’
f.jira_username JIRA username	Username for accessing the JIRA REST API. This user must be a project-level administrator to index single projects, or a global or system-level administrator to index all projects. Leave empty if your JIRA system does not require authentication to see projects, issues, comments and attachments. type: `string` default value: '`’
f.maxResults Max results per page	The number of items to retrieve with each request. type: `integer` default value: ‘500’
f.maxSizeBytes Maximum file size (bytes)	Maximum size, in bytes, of a project description, issue description, worklog, issue comment, or attachment. type: `integer` default value: ‘32000’

Property	Description
f.minSizeBytes Minimum file size (bytes)	Minimum size, in bytes, of a project description, issue description, worklog, issue comment, or attachment. type: `integer` default value: ‘0’
f.timeoutMS Connection timeout (ms)	Time in milliseconds to wait for server response. type: `integer` default value: ‘10000’

Security Trimming

Property	Description
enable_security_trimming Enable Security Trimming	type: `object`

Limit Documents

Property	Description
f.maxSizeBytes Maximum file size (bytes)	Maximum size, in bytes, of a project description, issue description, worklog, issue comment, or attachment. type: `integer` default value: ‘32000’
f.minSizeBytes Minimum file size (bytes)	Minimum size, in bytes, of a project description, issue description, worklog, issue comment, or attachment. type: `integer` default value: ‘0’
f.maxResults Max results per page	The number of items to retrieve with each request. type: `integer` default value: ‘500’

Property	Description
f.index_projects Index Projects	Select to index project-level information. type: `boolean` default value: ‘false’
f.index_issues Index Documents	Select to index issues. type: `boolean` default value: ‘true’
f.index_comments Index Comments	Select to index comments on issues. type: `boolean` default value: ‘true’
f.index_attachments Index Attachments	Select to index attachments on issues. type: `boolean` default value: ‘true’
f.index_worklogs Index Worklogs	Select to index worklogs on issues. type: `boolean` default value: ‘false’
depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: ‘-1’
maxItems Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: ‘-1’

Property	Description
includeExtensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. type: `array` of `string`
includeRegexes Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: `array` of `string`
excludeExtensions Excluded file extensions	File extensions that should not to be fetched. This will limit this datasource to all extensions except this list. type: `array` of `string`
excludeRegexes Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: `array` of `string`
delete Delete dead URIs	Set to true to remove documents from the index when they can no longer be accessed as unique documents. type: `boolean` default value: ‘true’
deleteErrorsAfter Fetch failure allowance	Number of fetch failures to tolerate before removing a document from the index. The default of -1 means no fetch failures will be removed. type: `integer` default value: ‘-1’

Crawl Performance

Property	Description
chunkSize Fetch batch size	The number of items to batch for each round of fetching. The default is 50. type: `integer` default value: ‘50’
fetchThreads Fetch threads	The number of threads to use during fetching. The default is 5. type: `integer` default value: ‘5’
fetchDelayMS Fetch delay	Number of milliseconds to wait between fetch requests. The default is 0. This property can be used to throttle a crawl if necessary. type: `integer` default value: ‘0’
fetchDelayMSPerHost Fetch delay per host	If true, the 'Fetch delay (ms)' property will be applied for each host. type: `boolean` default value: ‘false’
emitThreads Emit threads	The number of threads used to send documents from the connector to the index pipeline. The default is 5. type: `integer` default value: ‘5’
retryEmit Retry emits	Set to true for emit batch failures to be retried on a document-by-document basis. type: `boolean` default value: ‘true’

Property	Description
failFastOnStartLinkFailure Fail crawl if start links are invalid	If true, when Fusion cannot connect to any of the provided start links, the crawl is stopped and an exception logged. type: `boolean` default value: ‘true’

Dedupe

Property	Description
dedupe Dedupe documents	If true, documents will be deduplicated. Deduplication can be done based on an analysis of the content, on the content of a specific field, or by a JavaScript function. If neither a field nor a script are defined, content analysis will be used. type: `boolean` default value: ‘false’
dedupeSaveSignature Save dedupe signature	If true,the signature used for dedupe will be stored in a 'dedupeSignature_s' field. Note this may cause errors about 'immense terms' in that field. type: `boolean` default value: ‘false’
dedupeField Dedupe field	Field to be used for dedupe. Define either a field or a dedupe script, otherwise the full raw content of each document will be used. type: `string`
dedupeScript Dedupe script	Custom javascript to dedupe documents. The script must define a 'genSignature(content)\{}' function, but can use any combination of document fields. The function must return a string. type: `string`

Recrawl Rules

Property	Description
f.timeoutMS Connection timeout (ms)	Time in milliseconds to wait for server response. type: `integer` default value: ‘10000’
f.cacheSize Cache size (number of entries)	Number of entries to cache when making REST requests. type: `integer` default value: ‘2000’
refreshAll Recrawl all items	Set to true to always recrawl all items found in the crawldb. type: `boolean` default value: ‘true’
refreshStartLinks Recrawl start links	Set to true to recrawl items specified in the list of start links. type: `boolean` default value: ‘false’
refreshErrors Recrawl errors	Set to true to recrawl items that failed during the last crawl. type: `boolean` default value: ‘false’
refreshOlderThan Recrawl age	Number of seconds to recrawl items whose last fetched date is longer ago than this value. type: `integer` default value: ‘-1’
refreshIDPrefixes Recrawl ID prefixes	A prefix to recrawl all items whose IDs begin with this value. type: `array` of `string`

Property	Description
refreshIDRegexes Recrawl ID regexes	A regular expression to recrawl all items whose IDs match this pattern. type: `array` of `string`
refreshScript Recrawl script	A JavaScript function ('shouldRefresh()') to customize the items recrawled. type: `string`
forceRefresh Force recrawl	Set to true to recrawl all items even if they have not changed since the last crawl. type: `boolean` default value: ‘false’

Crawl History

Property	Description
retainOutlinks Retain links in the crawldb	Set to true for links found during fetching to be stored in the crawldb. type: `boolean` default value: ‘true’
aliasExpiration Alias expiration	The number of crawls after which an alias will expire. The default is 1 crawl. type: `integer` default value: ‘1’
crawlDBType Crawl database type	The type of crawl database to use, in-memory or on-disk. type: `string` default value: ‘in-memory’ enum: \{ in-memory on-disk }

Property	Description
indexCrawlDBToSolr Index crawl database to Solr	EXPERIMENTAL: Set to true to index the crawl-database into a 'crawldb_' collection in Solr. type: `boolean` default value: ‘false’
reevaluateCrawlDBOnStart Reevaluate crawldb on start?	Reevaluate exisiting crawldb entries for legality on startup? type: `boolean` default value: ‘false’

Field Mapping

Property	Description

mappings
Field mappings
	List of mapping rules
	type: `array` of `object`
	object attributes: \{
	`operation` : \{
	display name: Operation
	type: `string`
	default value: ‘copy’
	description : The type of mapping to perform: move, copy, delete, add, set, or keep
	enum: \{ copy move delete set add keep }
	} +
	``` `source` _ (required)_ : \{ + ```
	display name: Source Field +
	``` type: `string` + ```
	description : The name of the field to be mapped. +
	} +
	``` `target` : \{ + ```
	display name: Target Field +
	``` type: `string` + ```
	description : The name of the field to be mapped to. +
	} +
	}

Property	Description
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
pipeline Pipeline ID	The index pipeline used to process documents.

13.23. Javascript Connector and Datasource Configuration

The Javascript connector allows users to write ad-hoc document retrieval routines to fetch content from filesystems and websites.

It provides a property `f.script` which is a JavaScript program that is compiled by the JDK. This program returns a content item which is handed off to the fetcher.

The script engine works exactly the same as the JavaScript Index and JavaScript Query Pipeline stages. The JavaScript program must be standard ECMAScript.

You can use any Java class available to the connectors JDK classloader to manipulate that object within a function. As in Java, to access Java classes by their simple names instead of their fully specified class names, e.g. to be able to write `String` instead of `java.lang.String`, these classes must be imported. The java.lang package is not imported by default, because its classes would conflict with Object, Boolean, Math, and other built-in JavaScript objects. To import a Java class, use the JavaImporter object and the `with` statement, which limits the scope of the imported Java packages and classes.

```
var imports = new JavaImporter(java.lang.String);
...
with (imports) {
    var name = new String("foo"); ...
}
```

For global variables, you can reference these objects using the `Java.type` API extension. See this tutorial for details: http://winterbe.com/posts/2014/04/05/java8-nashorn-tutorial/

13.23.1. The JavaScript Program

The Javascript context provides the following variables:

- id, type java.lang.String - the id of the object to fetch. This is almost always the URI of the datasource to connect to and fetch content.
- lastModified, type long - the time since the epoch from which the item was last touched.
- signature, type java.lang.String - an optional string meant to be used to compare versions of the ID being fetched, e.g. an ETag in a web-crawl.
- content, type crawler.common.MutableObject - a Content object that can be modified and returned, for fine grained control over the return. See the section on return types below.
- _fetcher, type Fetcher - the current Fetcher instance (usually type JavascriptFetcher), used to interact with the Fetcher, including getting a WebFetcher instance using`_fetcher.getWebFetcher()`
- `_context`, type java.util.Map - a map used to store data to persist across calls to `fetch()`, e.g. an instance of WebFetcher obtained using`_fetcher.getWebFetcher().`

The program must return one of the following kinds of objects:

- String — A string object. This will be converted to UTF-8 bytes and added as the raw content on a common.crawler.Content object and returned from the fetch() method
- byte [] — A byte array. This array will be set on a common.crawler.Content object and returned from the fetch()

method

- common.crawler.MutableContent: If you wish to have complete control over the return from fetch(), make changes to the content object provided in the Context and return it. DO NOT CREATE A NEW OBJECT.
- An array of Objects. These will be converted to Embedded Content (the Fetcher will return a parent Content object that has a "Container" discardMessage. The Embedded Content on that container will consist of calling toString() on the objects in the array. Thus, it is best if the array is simply
- A JavaScript Map. The map will be converted to fields on the Content item returned

If the JavaScript script is implemented as a function, the return statement must return one of the above types. If the script is not function-based, than the last line in the script must evaluate to one of these object types.

13.23.2. Examples

Return content as a java.lang.String

```
var str = new java.lang.String("Java");
str;
```

Return content as a byte array

```
var bytes = new java.lang.String("Java");
bytes.getBytes('UTF-8');
```

Return content as a MutableContent object

```
var bytes = new java.lang.String("Java");
content.setRawContent(bytes.getBytes('UTF-8'));
content
```

Return content as a JavaScript array

```
var strings = ["hi", "bye"];
strings;
```

Return content as a JavaScript map

```
var map = {"hi": "bye", "bye": "hi", "number":1};
map;
```

Leverage the Fetcher

```
var webFetcher = _context.get("webFetcher");
if (null == webFetcher) {
  webFetcher = _fetcher.getWebFetcher();
  // it's possible to pass config options to getWebFetcher() as a map as well, e.g.:
  // _fetcher.getWebFetcher({"f.discardLinkURLQueries" : false });
  _context.put("webFetcher", webFetcher);
}
var webContent = webFetcher.fetch(id, lastModified, signature);
var jsoupDoc = webContent.getDocument();
if (null !== jsoupDoc) {
  // modify the Jsoup document or web-content as-needed here, adding new links, removing sections etc.
  // ...
  // ...
  webContent.setRawContent(jsoupDoc.toString().getBytes("UTF-8"));
}
webContent;
```

13.23.3. Configuration

Connector-specific Properties

Property	Description
f.script Script	JavaScript program to fetch documents. type: `string`

Link Discovery

Property	Description
restrictToTree Restrict to sub-directories and child pages	If true, only documents found in a tree below the start links will be fetched. For Web crawls, this will only limit to the domain. type: `boolean` default value: 'false'
restrictToTreeAllowSubdomains Allow sub-domains in restrictToTree	If true, any sub-domain will be allowed, even if the crawl is restricted to the tree of items found below the start links. type: `boolean` default value: 'false'

Property	Description
restrictToTreeUseHostAndPath Use paths in restrictToTree	If true, the path in start links will be used to restrict items fetched. For example, if the start link is 'http://host.com/US', this option will limit all followed URLs to this path. type: `boolean` default value: ‘false’
restrictToTreeIgnoredHostPrefixes Ignored host prefixes	List of host prefixes to ignore when checking links for restrictToTree link-legality checks. For example, 'www.' can be ignored so links with the same domain are allowed. type: `array` of `string` default value: [` www. `]

Limit Documents

Property	Description
depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: ‘-1’
maxItems Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: ‘-1’
includeExtensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. type: `array` of `string`
includeRegexes Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: `array` of `string`

Property	Description
excludeExtensions Excluded file extensions	File extensions that should not to be fetched. This will limit this datasource to all extensions except this list. type: `array` of `string`
excludeRegexes Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: `array` of `string`

Crawl Performance

Property	Description
chunkSize Fetch batch size	The number of items to batch for each round of fetching. The default is 50. type: `integer` default value: '50'
fetchThreads Fetch threads	The number of threads to use during fetching. The default is 5. type: `integer` default value: '5'
fetchDelayMS Fetch delay	Number of milliseconds to wait between fetch requests. The default is 0. This property can be used to throttle a crawl if necessary. type: `integer` default value: '0'
emitThreads Emit threads	The number of threads used to send documents from the connector to the index pipeline. The default is 5. type: `integer` default value: '5'

Property	Description
retryEmit Retry emits	Set to true for emit batch failures to be retried on a document-by-document basis. type: `boolean` default value: ‘true’
failFastOnStartLinkFailure Fail crawl if start links are invalid	If true, when Fusion cannot connect to any of the provided start links, the crawl is stopped and an exception logged. type: `boolean` default value: ‘true’

Dedupe

Property	Description
dedupe Dedupe documents	If true, documents will be deduplicated. Deduplication can be done based on an analysis of the content, on the content of a specific field, or by a JavaScript function. If neither a field nor a script are defined, content analysis will be used. type: `boolean` default value: ‘false’
dedupeSaveSignature Save dedupe signature	If true,the signature used for dedupe will be stored in a 'dedupeSignature_s' field. Note this may cause errors about 'immense terms' in that field. type: `boolean` default value: ‘false’
dedupeField Dedupe field	Field to be used for dedupe. Define either a field or a dedupe script, otherwise the full raw content of each document will be used. type: `string`

Property	Description
dedupeScript Dedupe script	Custom javascript to dedupe documents. The script must define a 'genSignature(content)\{}' function, but can use any combination of document fields. The function must return a string. type: `string`

Recrawl Rules

Property	Description
refreshAll Recrawl all items	Set to true to always recrawl all items found in the crawldb. type: `boolean` default value: ‘true’
refreshStartLinks Recrawl start links	Set to true to recrawl items specified in the list of start links. type: `boolean` default value: ‘false’
refreshErrors Recrawl errors	Set to true to recrawl items that failed during the last crawl. type: `boolean` default value: ‘false’
refreshOlderThan Recrawl age	Number of seconds to recrawl items whose last fetched date is longer ago than this value. type: `integer` default value: ‘-1’
refreshIDPrefixes Recrawl ID prefixes	A prefix to recrawl all items whose IDs begin with this value. type: `array` of `string`

Property	Description
refreshIDRegexes Recrawl ID regexes	A regular expression to recrawl all items whose IDs match this pattern. type: `array` of `string`
refreshScript Recrawl script	A JavaScript function ('shouldRefresh()') to customize the items recrawled. type: `string`
forceRefresh Force recrawl	Set to true to recrawl all items even if they have not changed since the last crawl. type: `boolean` default value: ‘false’
delete Delete dead URIs	Set to true to remove documents from the index when they can no longer be accessed as unique documents. type: `boolean` default value: ‘true’
deleteErrorsAfter Fetch failure allowance	Number of fetch failures to tolerate before removing a document from the index. The default of -1 means no fetch failures will be removed. type: `integer` default value: ‘-1’

Crawl History

Property	Description
retainOutlinks Retain links in the crawldb	Set to true for links found during fetching to be stored in the crawldb. type: `boolean` default value: ‘true’

Property	Description
aliasExpiration Alias expiration	The number of crawls after which an alias will expire. The default is 1 crawl. type: `integer` default value: ‘1’
crawlDBType Crawl database type	The type of crawl database to use, in-memory or on-disk. type: `string` default value: ‘in-memory’ enum: \{ in-memory on-disk }
indexCrawlDBToSolr Index crawl database to Solr	EXPERIMENTAL: Set to true to index the crawl-database into a 'crawldb_' collection in Solr. type: `boolean` default value: ‘false’

Field Mapping

Property	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: 'copy' description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: 'false'

Property	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
pipeline Pipeline ID	The index pipeline used to process documents.

13.24. Jive Connector and Datasource Configuration

Retrieve content from a Jive instance.

Important	If you are upgrading from Fusion 2.0 or earlier, you must remove the old connector, add the new one, and then install the Lucidworks add-on for Jive. See the instructions below.

13.24.1. Updating the Jive connector

Perform all of the steps below as the user that owns the Fusion installation files and processes.

Removing the Old Jive Connector

1. Remove all existing Jive datasources, using either the UI or the REST API. Any datasource that is not a Jive type of datasource can be retained.

   a. In the UI, go to the Datasources screen for each collection that includes a Jive datasource, and select the X icon on the right side of the row. Confirm the delete action to remove the datasource.

   b. Using the Connector Datasources API, you can perform a call similar to this one:

   ```
   curl -u user:password -X DELETE http://localhost:8764/api/apollo/connectors/datasources/<id>
   ```

   Be sure to use the correct username and password, and also replace the hostname, port and datasource ID as appropriate.

2. Stop Fusion.

3. On the filesystem, find the file `fusion/apps/connectors/plugins/luicid.anda/connectors.json`. From this file, remove the following line:

   ```
   "jive": "com.lucidworks.connectors.anda.type.JiveAndaType",
   ```

   Be sure to save your changes.

4. In the directory `fusion/apps/connectors/plugins/lucid.anda/lib/`, remove the file `lucid.jive.jar` by deleting it or saving it outside of your Fusion installation directory tree (i.e., to a `/tmp` directory or similar).

Adding the New Jive Connector

1. In the directory `fusion/apps/connectors/plugins`, make a new directory named `lucid.jive`.

2. Copy the files `connectors.json` and `lucid.jive.jar` to the directory created in the previous step, `fusion/apps/connectors/plugins/lucid.jive`.

3. In order for the UI to display the correct name for the new connector, we must update a datasource mapping file.

   In `fusion/apps/jetty/ui/webapps/root/WEB-INF/classes/public/data/datsource-name-map.json`, remove the following line:

```
"lucid.anda.jive" : "Jive",
```

And add the following line (anywhere, as long as it is a new line):

```
"lucid.jive.jive" : "Jive",
```

When adding the new line, be sure to add a comma to the previous line in order to preserve correct JSON. If this file cannot be parsed, you will see odd-looking datasource names in the UI.

4. Start Fusion.

Installing the Jive Add-on

The add-on allows using OAuth for authenticating the crawler to the Jive API. This is a recommended way for Jive users to use the API, as documented by Jive at https://community.jivesoftware.com/docs/DOC-157031. The add-on is used to get a client ID and secret that are supplied to the connector.

If the client ID and secret cannot be retrieved or used, then Basic authentication is used as a fallback.

How to install the Jive Add-on

Following the instructions at https://community.jivesoftware.com/docs/DOC-141123, use the Jive UI to upload the add-on located in your Fusion distribution: `fusion/apps/connectors/resources/lucid.jive/lucidworks-jive-addon.zip`

13.24.2. Security Trimming of Results

If "Enable Security Trimming" is enabled, the Jive connector will use the `visibility` field while indexing permission metadata on content. This data is stored in the `acl_ss` field for each document.

The value of the `visibility` field impacts the permissions assigned to a document. The following list describes how the types of permissions found in the `visibility` field of a document are used.

All	The value stored in the `acl_ss` field will be "all".
People	The document will include a list of users who are authorized to view the content. This list will be stored in the `acl_ss` field as user email addresses.
Place	A request will be made to determine the type of group. The group type will determine the permissions stored.
Open or Members Only	The value stored in the `acl_ss` field will be "all".
Private or Secret	The value stored in the `acl_ss` field will be the name of the group.

13.24.3. Configuration

Property	Description
batch_size Batch Size	Number of items to fetch in each batch request. The Jive API limit is 100. If you are having problems with Jive timing out on requests, decreasing the batch size might help the crawl continue, but will extend the overall time the crawl takes. type: `integer` default value: '95'
collection Collection	Collection documents will be indexed to. type: `string`
contents_to_crawl Contents to crawl *required*	List of content types to crawl, separated by commas and no spaces. The connector can only retrieve objects from Jive's Content entity, as described in https://developers.jivesoftware.com/api/v3/cloud/rest/ContentEntity.html. type: `string` default value: 'All' minLength: 1
diagnostic_mode Diagnostic Mode	Enable to print more detailed information to the logs about each request. type: `boolean` default value: 'false'
enable_security_trimming Enable Security Trimming	Security trimming restricts query results to records the user is allowed to access by indexing user access information with other document metadata. Enable this option to index user access metadata. type: `boolean` default value: 'false'

Property	Description
excluded_content_types Excluded content types	Content MIME types to exclude from fetching. Files with a content type contained in this list will not be downloaded but available metadata will be added to the index. By default, this list includes many content types Apache Tika is not able to process. type: `array` of `string` default value: [`video/x-ms-wm`, `text/csv`, `application/postscript`, `video/x-ms-wmv`, `image/vnd.dwg`, `image/tiff`, `audio/x-aiff`, `audio/x-ms-wma`, `application/mac-binhex40`, `application/winhlp`, `application/x-font-ttf`, `video/x-ms-asf`, `application/illustrator`, `chemical/x-pdb`, `application/x-iso9660-image`, `application/x-msdownload`, `audio/x-pn-realaudio`, `application/x-ms-installer`, `image/gif`, `video/x-msvideo`, `application/vnd.visio`, `audio/mpeg`, `application/x-emf`, `image/vnd.dxf`, `application/x-dosexec`, `application/x-shockwave-flash`, `audio/midi`, `image/vnd.adobe.photoshop`, `application/octet-stream`, `application/x-java-jnilib`, `image/x-ms-bmp`, `image/jpeg`, `application/x-msaccess`, `video/mp4`, `text/plain`, `video/quicktime`, `application/x-dtbresource+xml`, `text/x-expect`, `application/vnd.ms-htmlhelp`, `audio/basic`, `video/mpeg`, `application/x-quattro-pro`, `application/vnd.rn-realmedia`, `audio/x-wav`, `application/vnd.ms-cab-compressed`, `audio/ogg`, `video/ogg`, `application/ogg`, `audio/vorbis`, `audio/x-oggflac`, `audio/x-oggpcm`, `audio/opus`, `audio/speex`, `video/daala`, `video/theora`, `video/x-dirac`, `video/x-ogm`, `video/x-ogguvs`, `video/x-oggyuv`, `video/x-oggrgb`, `audio/x-flac`, `application/x-msmetafile`, `application/x-roxio-toast`, `application/mp4`, `audio/mp4`, `audio/mp4a-latm`, `video/mp4v-es`, `video/x-m4v`, `audio/mpeg4-generic`, `audio/vnd.sealedmedia.softseal.mpeg`, `audio/x-mpegurl`, `video/bmpeg`, `video/mpeg4-generic`, `application/x-msdownload;format=pe`, `application/x-dosexec`, `application/x-msdownload;format=pe32`, `application/x-msdownload;format=pe64`, `application/x-msdownload;format=pe-itanium`, `application/x-msdownload;format=pe-armLE`, `application/x-msdownload;format=pe-arm7`, `application/x-tika-msoffice`,]
fetch_announcements Fetch Announcements	When enabled system and place announcements are fetched type: `boolean` default value: ‘true’

Property	Description
jive_client_id Jive Add-on Client Id	Jive Client ID provided by the Jive Add-on. If the Add-on has been installed in your Jive instance, OAuth will be used to authenticate to the Jive instance. If this is not provided, the username and password will be used instead. type: `string`
jive_client_secret Jive Add-on Client Secret	Jive Secret Key for the provided Client ID. type: `string`
jive_instance_url Jive instance URL *required*	Address of your Jive instance. type: `string` minLength: 1
jive_password Jive password *required*	Password for the provided user. type: `string` minLength: 1
jive_username Jive username *required*	Username to access the Jive instance. If this is provided and the Client ID is also provided, the Client ID will be preferred for OAuth-based auhtentication to Jive. The username and password are defined as a backup in case OAuth is not available. type: `string` minLength: 1
max_file_size Maximum file size (bytes)	Maximum size, in bytes, of documents to fetch. Files with size greater than this value will not be downloaded but metadata will be added. type: `integer` default value: '10485760'

Property	Description
places_filter Places Filter	Filter to control the indexation of contents under specific places, this can not be set with Places to crawl at the same time type: `array` of `object` object attributes: \{ `object_id` *(required)* : \{ display name: Object Id type: `string` } `object_type_id` *(required)* : \{ display name: Object Type Id type: `string` minLength : 1 } }
places_to_crawl Places to crawl	List of places to crawl, separated by commas and no spaces type: `string` default value: ‘All’
proxy_address HTTP proxy address	Address of the HTTP proxy, if required. This should be entered in the format: ://: type: `string`
request_delay Request delay (ms)	The amount of time, in milliseconds, to wait before perform each request. type: `integer` default value: ‘0’
requests_timeout Connection timeout (ms)	Time in milliseconds to wait for server response. type: `integer` default value: ‘60000’

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: array of object object attributes: \{ operation : \{ display name: Operation type: string default value: 'copy' description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: boolean default value: 'false'

Initial Mappings	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.25. Local Filesystem Connector and Datasource Configuration

A filesystem-based data store is a network of nodes to be traversed, where each node (such as a Unix file directory) provides information about its child nodes (such as the files in that directory) or references other nodes (such as links in an HTML document).

The crawler captures information about the node, e.g., filename, permissions, date of creation, last modification, and last access, as well as the contents of the nodes. The extent of the network of nodes to be traversed is discovered during the crawl.

The connector provides rules to limit the crawl and re-crawling. These rules use datasource configuration properties to limit the extent of the network (depth of nodes to explore) as well as limiting processing to a subset of files based on file names and file size. An overall limit can be set on number of files retrieved during a crawl.

13.25.1. Configuration

Connector-specific Properties

Property	Description
f.addFileMetadata Add file metadata	Set to true to add information about documents found in the filesystem to the document, such as document owner, group, or ACL permissions. type: `boolean` default value: ‘true’
f.maxSizeBytes Maximum file size (bytes)	Maximum size, in bytes, of documents to fetch. type: `integer` default value: ‘4194304’
f.minSizeBytes Minimum file size (bytes)	Minimum size, in bytes, of documents to fetch. type: `integer` default value: ‘0’

Limit Documents

Property	Description
restrictToTree Restrict to sub-directories and child pages	If true, only documents found in a tree below the start links will be fetched. For Web crawls, this will only limit to the domain. type: `boolean` default value: ‘true’
depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: ‘-1’
f.maxSizeBytes Maximum file size (bytes)	Maximum size, in bytes, of documents to fetch. type: `integer` default value: ‘4194304’
f.minSizeBytes Minimum file size (bytes)	Minimum size, in bytes, of documents to fetch. type: `integer` default value: ‘0’
maxItems Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: ‘-1’
includeExtensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. type: `array` of `string`
includeRegexes Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: `array` of `string`

Property	Description
excludeExtensions Excluded file extensions	File extensions that should not to be fetched. This will limit this datasource to all extensions except this list. type: `array` of `string`
excludeRegexes Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: `array` of `string`
f.addFileMetadata Add file metadata	Set to true to add information about documents found in the filesystem to the document, such as document owner, group, or ACL permissions. type: `boolean` default value: 'true'

Complex Documents

Property	Description
splitCSV Split CSV and archives	Set to true to split CSV/TSV files into a document per row and split archives into individual files. type: `object` object attributes: \{ }
splitArchives Split archives	Set to true for .zip, .tar, .tar.gz, .tgz, .jar, .bzip, .bzip2, .cpio, and .dump files to be opened and documents found indexed as distinct documents. type: `boolean` default value: 'true'

Crawl Performance

Property	Description
chunkSize Fetch batch size	The number of items to batch for each round of fetching. The default is 50. type: `integer` default value: '50'
fetchThreads Fetch threads	The number of threads to use during fetching. The default is 5. type: `integer` default value: '5'
fetchDelayMS Fetch delay	Number of milliseconds to wait between fetch requests. The default is 0. This property can be used to throttle a crawl if necessary. type: `integer` default value: '0'
emitThreads Emit threads	The number of threads used to send documents from the connector to the index pipeline. The default is 5. type: `integer` default value: '5'
retryEmit Retry emits	Set to true for emit batch failures to be retried on a document-by-document basis. type: `boolean` default value: 'true'
failFastOnStartLinkFailure Fail crawl if start links are invalid	If true, when Fusion cannot connect to any of the provided start links, the crawl is stopped and an exception logged. type: `boolean` default value: 'true'

Dedupe

Property	Description
dedupe Dedupe documents	If true, documents will be deduplicated. Deduplication can be done based on an analysis of the content, on the content of a specific field, or by a JavaScript function. If neither a field nor a script are defined, content analysis will be used. type: `boolean` default value: ‘false’
dedupeSaveSignature Save dedupe signature	If true,the signature used for dedupe will be stored in a 'dedupeSignature_s' field. Note this may cause errors about 'immense terms' in that field. type: `boolean` default value: ‘false’
dedupeField Dedupe field	Field to be used for dedupe. Define either a field or a dedupe script, otherwise the full raw content of each document will be used. type: `string`
dedupeScript Dedupe script	Custom javascript to dedupe documents. The script must define a 'genSignature(content)\{}' function, but can use any combination of document fields. The function must return a string. type: `string`

Recrawl Rules

Property	Description
delete Delete dead URIs	Set to true to remove documents from the index when they can no longer be accessed as unique documents. type: `boolean` default value: ‘true’

Property	Description
deleteErrorsAfter Fetch failure allowance	Number of fetch failures to tolerate before removing a document from the index. The default of -1 means no fetch failures will be removed. type: `integer` default value: '-1'
refreshAll Recrawl all items	Set to true to always recrawl all items found in the crawldb. type: `boolean` default value: 'true'
refreshStartLinks Recrawl start links	Set to true to recrawl items specified in the list of start links. type: `boolean` default value: 'false'
refreshErrors Recrawl errors	Set to true to recrawl items that failed during the last crawl. type: `boolean` default value: 'false'
refreshOlderThan Recrawl age	Number of seconds to recrawl items whose last fetched date is longer ago than this value. type: `integer` default value: '-1'
refreshIDPrefixes Recrawl ID prefixes	A prefix to recrawl all items whose IDs begin with this value. type: `array` of `string`
refreshIDRegexes Recrawl ID regexes	A regular expression to recrawl all items whose IDs match this pattern. type: `array` of `string`

Property	Description
refreshScript Recrawl script	A JavaScript function ('shouldRefresh()') to customize the items recrawled. type: `string`
forceRefresh Force recrawl	Set to true to recrawl all items even if they have not changed since the last crawl. type: `boolean` default value: ‘false’

Crawl History

Property	Description
retainOutlinks Retain links in the crawldb	Set to true for links found during fetching to be stored in the crawldb. type: `boolean` default value: ‘true’
aliasExpiration Alias expiration	The number of crawls after which an alias will expire. The default is 1 crawl. type: `integer` default value: ‘1’
crawlDBType Crawl database type	The type of crawl database to use, in-memory or on-disk. type: `string` default value: ‘in-memory’ enum: \{ in-memory on-disk }
indexCrawlDBToSolr Index crawl database to Solr	EXPERIMENTAL: Set to true to index the crawl-database into a 'crawldb_' collection in Solr. type: `boolean` default value: ‘false’

Field Mapping

Property	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` _ (required)_ : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Property	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
pipeline Pipeline ID	The index pipeline used to process documents.

13.26. Logstash Connector and Datasource Configuration

Note	This connector is deprecated in Fusion 2.4, and removed in Fusion 3.0. See the new LogStash Output Plugin in Lucidworks Labs.

Logstash is an open-source log management tool which takes inputs from one or more logfiles and parses and filters them according to a set of configurations and produces as output a stream of JSON objects. The Logstash connector uses Logstash 1.4.2 to send documents to a Fusion pipeline.

The Fusion archive includes a Logstash deployment, located in the directory fusion/apps/connectors/resources/lucid.logstash/logstash1.4.2. This deployment includes a custom ruby class lucidworks_pipeline_output.rb which collects Logstash outputs and sends them to a Fusion pipeline.

13.26.1. Configuration

The connector is "lucid.logstash" and the plugin type is "logstash".

The connector takes a required property "Logstash Configuration" which is the Logstash configuration in JavaScript. The Fusion UI Admin Tool provides a JavaScript-aware input box which so that you can create and edit your Logstash configuration directly from Fusion.

The Logstash configuration script has three clauses:

- input
- filter
- output

The connector takes an optional property "buffer_size" which is the number of documents to buffer. When the buffer limit is reached, all documents in the buffer are sent to Solr for batch indexing. The default buffer size is 10. In the Fusion UI Admin Tool, choosing the "Advanced" option on the Datasource configuration panel exposes an input box labeled "Buffer size".

13.26.2. Running a Datasource Job

Once started, a Logstash connector will continue to run indefinitely. The Fusion UI Admin Tool Datasource panel provides controls to start, stop, abort, and clear the datasource. For file-based inputs, Logstash tracks last line read in a file in a "since_db" file. Clearing the datasource requires removing these files.

13.26.3. Examples

The Logstash configuration is specified in JavaScript. The specification consists of the following three clauses:

- input - specifies how to listen for incoming data
- filter - data-specific Logstash filter definitions.
- output - additional outputs, as needed.

All three clauses must be present, but the filter and output clauses can be empty.

Simple Logstash Configuration for reading an input file:

```
input {
    file {
        path => "/var/log/mylogfile.log"
    }
}
filter {
}
output {
}
```

Simple Logstash Configuration for listening for incoming data on a specific port:

```
input {
    tcp {
        port => 10234
    }
}
filter {
}
output {
}
```

Filter clause for pure JSON data:

```
filter {
    json {source => "message" }
}
```

13.26.4. Tutorial

The Lucidworks blog post Data Analytics Using Fusion and Logstash walks through the process of developing a Logstash script and configuring and running a datasource.

13.27. MongoDB Datasource and Connector Configuration

<a href="http://lucidworks.com/connectors/"><img src="/project/pdfs/assets/images/common/icons/Icon-Download-128x128.png" format="png" alt="[Download</a>" width="45.0" tmp="false">] <a href="http://lucidworks.com/connectors/">Connector download</a>

Retrieve data from a MongoDB instance.

For the initial data ingest from a MongoDB database, select the option to do a full synchronization of the content in MongoDB. Subsequent ingest runs can use the `oplog` in MongoDB to discover new content and updates to existing content (updated or removed documents). A full synchronization can be done by selecting the full synchronization option and re-running the data ingest job.

13.27.1. Configuration

Property	Description
batch_size_solr_commit Batch size Solr commit	The number of documents every time solr_commit will be made. type: `integer` default value: ‘1000’
collection Collection	Collection documents will be indexed to. type: `string`
collections MongoDB Collections to index	The MongoDB collections to index, in the format 'databaseName.collection'. Multiple collections can be separated by commas. The default '.' means that all databases and all collections of each database will be crawled (limited by user access). type: `string` default value: ‘.’ minLength: 1
commit_on_finish Solr commit on finish	Set to true for a request to be sent to Solr after the last batch has been fetched to commit the documents to the index. type: `boolean` default value: ‘true’

Property	Description
list_credentials Credentials	Credentials for Mongo databases type: `array` of `object` object attributes: \{ `database` : \{ display name: Database type: `string` } `password` : \{ display name: Password type: `string` } `username` : \{ display name: Username type: `string` } }
list_hosts Hosts *required*	Host and ports of Mongo nodes type: `array` of `object` default value: [`{"host"⇒"localhost", "port"⇒27017}`,] object attributes: \{ `host` : \{ display name: Host type: `string` } `port` : \{ display name: Port type: `integer` } }
process_oplog Process OPLog *required*	Process updates from the oplog. Disable this option to perform a full synchronization of content in MongoDB collections with the index. type: `boolean` default value: ‘true’

Property	Description
read_preferences Read Preference Modes	Read preference describes how MongoDB clients route read operations to the members of a replica set. type: `string` default value: ‘primary’ enum: \{ primary primary preferred secondary secondary preferred nearest }
tag_set_list Read Preference Tag Sets	A list of Tag Sets used for non-primary read modes type: `array` of `object` default value: [ ] object attributes: \{ `tag_set` : \{ display name: Tag Set type: `array` of `object` default value: '`’ } }
verify_access Validate access	Set to true to require successful connection to the filesystem before saving this datasource. type: `boolean` default value: ‘true’

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: array of object object attributes: \{ operation : \{ display name: Operation type: string default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: boolean default value: ‘false’

Initial Mappings	Description

unmapped
Unmapped fields
	If fields do not match any of the field mapping rules, these rules will apply.
	type: `object`
	object attributes: \{
	`operation` : \{
	display name: Operation
	type: `string`
	default value: ‘copy’
	description : The type of mapping to perform: move, copy, delete, add, set, or keep
	enum: \{ copy move delete set add keep }
	`} +`
	`` `source` : \{ + ``
	`display name: Source Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped. +`
	`} +`
	`` `target` : \{ + ``
	`display name: Target Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped to. +`
	`} +`
	`}`

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.28. Solr Push Endpoint Datasource and Configuration

The Solr Push Endpoint accepts documents and pushes them to Solr using the Fusion index pipelines.

In Fusion 3.0 and earlier, this is called the Push connector.

It uses the embedded JettySolrRunner to push the documents. This requires defining a port for the JettySolrRunner that is not already in use by any other process. The documents can then be sent to Fusion at that port, and they will be consumed by Fusion.

13.28.1. Configuration

Property	Description
collection Collection	Collection documents will be indexed to. type: `string`
commit_on_finish Solr commit on finish	Set to true for a request to be sent to Solr after the last batch has been fetched to commit the documents to the index. type: `boolean` default value: 'true'
port Port *required*	The port that will be used to push content to Solr. type: `integer`
url URL	shows the endpoint where the external application will send documents to. type: `string`

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: array of object object attributes: \{ operation : \{ display name: Operation type: string default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: boolean default value: ‘false’

Initial Mappings	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: 'false'
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: 'com.lucidworks.connectors.db.impl.MapDbConnectorDb' minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.29. S3 Connector and Datasource Configuration

The S3 connector can access AWS S3 buckets in native format.

13.29.1. Bucket Permissions

The connector uses the S3 API to request data from S3. It calls the `listBucket` service, which lists all buckets owned by the user account supplied to the connector.

When creating an S3 datasource using the UI, Fusion automatically verifies that the user information supplied has access to the bucket defined in the URL property. If the bucket is not in the list returned by S3, datasource creation may fail. At crawl time, if the bucket is not in the list returned by S3, the crawl will fail.

Permission errors when trying to create or crawl the datasource may be caused by incorrect username or password, or they may be due to user account permissions. The user account must have List Bucket permissions for the account which owns the bucket that the crawler is trying to access.

13.29.2. Configuration

Property	Description
add_failed_docs Add failed documents	Set to true to add documents even if they partially fail processing. Failed documents will be added with as much metadata as available, but may not include many expected fields. type: `boolean` default value: ‘false’
bounds Crawl bounds	Limits the the crawl to a specific directory sub-tree, hostname or domain. type: `string` default value: ‘tree’ enum: \{ tree host domain none }
collection Collection	Collection documents will be indexed to. type: `string`
commit_on_finish Solr commit on finish	Set to true for a request to be sent to Solr after the last batch has been fetched to commit the documents to the index. type: `boolean` default value: ‘true’

Property	Description
crawl_depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: ‘-1’ exclusiveMinimum: false minimum: -1
crawl_item_timeout Fetch timeout	Time in milliseconds to fetch any individual document. type: `integer` default value: ‘600000’ exclusiveMinimum: true minimum: 0
exclude_paths Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: `array` of `string` default value: [ ]
include_extensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. However, files with possible matching MIME types but non-matching file extensions will be skipped. The file extensions should be entered with no dots, with whitespace separating items in a list (e.g., 'pdf zip'). type: `array` of `string`
include_paths Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: `array` of `string`

Property	Description
index_directories Index directories	Set to true to add directories to the index as documents. If set to false, directories will not be added to the index, but they will still be traversed for documents. type: `boolean` default value: ‘false’
max_bytes Maximum file size (bytes)	Maximum size, in bytes, of documents to fetch. type: `integer` default value: ‘10485760’ exclusiveMinimum: false minimum: -1
max_docs Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: ‘-1’ exclusiveMinimum: false minimum: -1
max_threads Fetch threads	The maximum number of threads to use for fetching data. Each thread will create a new connection to the repository, which may make overall throughput faster, but will also require more system resources including CPU and memory. type: `integer` default value: ‘1’

Property	**Description**
maximum_connections Maximum fetch connections	Maximum number of concurrent connections to the filesystem. A large number of documents could cause a large number of simultaneous connections to the repository and lead to errors or degraded performance. In some cases, reducing this number may help performance issues. type: `integer` default value: ‘1000’
password AWS Secret Key *required*	The AWS Secret Key associated with the Access Key. type: `string`
splitter Split CSV/TSV and archives	Set to true to split CSV/TSV files into a document per row, or to split archives such as .zip, tar.gz, etc., into individual files. type: `object` object attributes: \{ }
url S3 URL *required*	A fully-qualified S3 URL, including bucket and sub-bucket paths, as required, e.g., 's3://{bucketName}/{path}'. type: `string` minLength: 1 pattern: .:.
username AWS Key *required*	An AWS Access Key ID that can access the content. type: `string`
verify_access Validate access	Set to true to require successful connection to the filesystem before saving this datasource. type: `boolean` default value: ‘true’

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: array of object object attributes: \{ operation : \{ display name: Operation type: string default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` _ (required)_ : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: boolean default value: ‘false’

Initial Mappings	Description

unmapped
 	
Unmapped fields
	If fields do not match any of the field mapping rules, these rules will apply.
	
	type: `object`
	
	object attributes: \{
	`operation` : \{
	display name: Operation
	type: `string`
	default value: ‘copy’
	description : The type of mapping to perform: move, copy, delete, add, set, or keep
	enum: \{ copy move delete set add keep }
	
	`} +`
	`` `source` : \{ + ``
	`display name: Source Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped. +`
	`} +`
	`` `target` : \{ + ``
	`display name: Target Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped to. +`
	`} +`
	`}`

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: 'false'
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: 'com.lucidworks.connectors.db.impl.MapDbConnectorDb' minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.30. S3H Connector and Datasource Configuration

The S3H connector can access AWS S3 buckets that are stored as blocks, as they are in HDFS ("Hadoop over Amazon"). More information on the S3 Block approach is available from the Hadoop wiki.

13.30.1. Bucket Permissions

The connector uses the S3 API to request data from S3. It calls the `listBucket` service, which lists all buckets owned by the user account supplied to the connector.

When creating an S3 datasource with the UI, Fusion automatically verifies that the user information supplied has access to the bucket defined in the URL property. If the bucket is not in the list returned by S3, datasource creation may fail. At crawl time, if the bucket is not in the list returned by S3, the crawl will fail.

Permission errors when trying to create or crawl the datasource may be caused by incorrect username or password, or they may be due to user account permissions. The user account must have List Bucket permissions for the account which owns the bucket that the crawler is trying to access.

13.30.2. Configuration

Property	Description
add_failed_docs Add failed documents	Set to true to add documents even if they partially fail processing. Failed documents will be added with as much metadata as available, but may not include many expected fields. type: `boolean` default value: ‘false’
bounds Crawl bounds	Limits the the crawl to a specific directory sub-tree, hostname or domain. type: `string` default value: ‘tree’ enum: \{ tree host domain none }
collection Collection	Collection documents will be indexed to. type: `string`

Property	Description
commit_on_finish Solr commit on finish	Set to true for a request to be sent to Solr after the last batch has been fetched to commit the documents to the index. type: `boolean` default value: ‘true’
crawl_depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: ‘-1’ exclusiveMinimum: false minimum: -1
crawl_item_timeout Fetch timeout	Time in milliseconds to fetch any individual document. type: `integer` default value: ‘600000’ exclusiveMinimum: true minimum: 0
exclude_paths Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: `array` of `string` default value: [ ]
include_extensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. However, files with possible matching MIME types but non-matching file extensions will be skipped. The file extensions should be entered with no dots, with whitespace separating items in a list (e.g., 'pdf zip'). type: `array` of `string`

Property	Description
include_paths Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: `array` of `string`
index_directories Index directories	Set to true to add directories to the index as documents. If set to false, directories will not be added to the index, but they will still be traversed for documents. type: `boolean` default value: ‘false’
max_bytes Maximum file size (bytes)	Maximum size, in bytes, of documents to fetch. type: `integer` default value: ‘10485760’ exclusiveMinimum: false minimum: -1
max_docs Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: ‘-1’ exclusiveMinimum: false minimum: -1
max_threads Fetch threads	The maximum number of threads to use for fetching data. Each thread will create a new connection to the repository, which may make overall throughput faster, but will also require more system resources including CPU and memory. type: `integer` default value: ‘1’

Property	**Description**
maximum_connections Maximum fetch connections	Maximum number of concurrent connections to the filesystem. A large number of documents could cause a large number of simultaneous connections to the repository and lead to errors or degraded performance. In some cases, reducing this number may help performance issues. type: `integer` default value: ‘1000’
password AWS Secret Key *required*	The AWS Secret Key associated with the Access Key. type: `string`
splitter Split CSV/TSV and archives	Set to true to split CSV/TSV files into a document per row, or to split archives such as .zip, tar.gz, etc., into individual files. type: `object` object attributes: \{ }
url S3 Hadoop URL *required*	A fully-qualified S3N URL, format: s3://{bucketName}/{path} . type: `string` minLength: 1 pattern: .:.
username AWS Key *required*	An AWS Access Key ID that can access the content. type: `string`
verify_access Validate access	Set to true to require successful connection to the filesystem before saving this datasource. type: `boolean` default value: ‘true’

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: array of object object attributes: \{ operation : \{ display name: Operation type: string default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` _ (required)_ : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: boolean default value: ‘false’

Initial Mappings	Description

unmapped
Unmapped fields
	If fields do not match any of the field mapping rules, these rules will apply.
	type: `object`
	object attributes: \{
	`operation` : \{
	display name: Operation
	type: `string`
	default value: ‘copy’
	description : The type of mapping to perform: move, copy, delete, add, set, or keep
	enum: \{ copy move delete set add keep }
	`} +`
	`` `source` : \{ + ``
	`display name: Source Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped. +`
	`} +`
	`` `target` : \{ + ``
	`display name: Target Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped to. +`
	`} +`
	`}`

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.31. Salesforce Connector and Datasource Configuration

Salesforce REST API to extract data from a Salesforce repository via a Salesforce Connected App.

This service has an authentication procedure and API usage limits.

Note	You need to have the Consumer Key and Consumer Secret of the Salesforce Connected App, which must be created by a Salesforce administrator for the account. Instructions on how to do this follow the datasource configuration details.

Note	Salesforce authentication requires a security token as well as a user password. A security token is an automatically-generated key from Salesforce For more information on security tokens see Reset Your Security Token in the online help.

13.31.1. Security Trimming

When you enable security trimming for a Salesforce connector, the system uses the Fusion username and tries finding the identical ID in Salesforce. If it finds this ID, then it uses the permissions given to that ID in Salesforce and applies a filter to the search query. If the ID is not found then it applies a filter to block all documents from being shown.

The security trimming feature assumes that the Fusion username is the same as Salesforce alias. When retrieving data from Salesforce, the connector retrieves the user Id based on the Fusion username (Salesforce alias) via the query:

```
Select Id from User WHERE alias={username}
```

If it isn’t found, then the connector will create a security filter to deny access to all documents.

If you are logged into the Fusion UI or send a request to the REST API with username "admin", the Salesforce connector uses that as the value of property "salesforce_username". Therefore, in Fusion, you must have accounts which reflect the Salesforce accounts.

13.31.2. Creating a Salesforce Connected App

Before creating the datasource, a Salesforce administrator for the account must create a Salesforce Connected App in Salesforce:

- Go to Setup > Create > Apps and in the Connected Apps section and click the new button
- Add values to the following properties:
  - Connected App Name
  - API Name
  - Contact Email
  - Check the property Enable OAuth Settings
- More properties should be displayed:
  - Callback URL: You can use a dumb HTTPS URL e.g. https://mydomain.com/auth

  - Selected OAuth Scopes: Select Full Access(full) scope
- Save the new Connected App
- After this step a message will be displayed, click the continue button.
- In the section API (Enable OAuth Settings) the consumer key and consumer secret should be displayed, which will be used on the datasource

For the Salesforce credentials the user must be a System Administrator or one with no restrictions (able to access all the objects and fields)

13.31.3. Configuration

Property	Description
collection Collection	Collection documents will be indexed to. type: `string`
connected_app_client_id Salesforce Client Id *required*	Salesforce connected app Consumer Key type: `string` minLength: 1
connected_app_client_secret Salesforce Client secret *required*	Salesforce connected app Consumer Secret type: `string` minLength: 1
diagnostic_mode Diagnostic Mode	type: `boolean` default value: 'false'
enable_security_trimming Enable Security Trimming	Security trimming restricts query results to records the user is allowed to access by indexing user access information with other document metadata. Enable this option to index user access metadata. type: `boolean` default value: 'false'
is_production_environment Is production environment	Set to true for production instances and false for sandboxes. type: `boolean` default value: 'true'

Property	Description
objects_to_crawl Objects to crawl	List of objects to crawl, separated by commas and no spaces. type: `array` of `string` default value: [`Case`, `CaseComment`, `CaseHistory`, `CaseFeed`, `FeedComment`, `Asset`, `Account`, `Contact`, `Opportunity`, `Product2`,]
salesforce_password Salesforce password *required*	type: `string` minLength: 1
salesforce_username Salesforce username *required*	type: `string` minLength: 1
soql_query Custom SOQL query	Optional SOQL query to limit or join records fetched for indexing. type: `string` default value: '`’ maxLength: 15000

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Initial Mappings	Description

unmapped
 	
Unmapped fields
	If fields do not match any of the field mapping rules, these rules will apply.
	
	type: `object`
	
	object attributes: \{
	`operation` : \{
	display name: Operation
	type: `string`
	default value: ‘copy’
	description : The type of mapping to perform: move, copy, delete, add, set, or keep
	enum: \{ copy move delete set add keep }
	`} +`
	`` `source` : \{ + ``
	`display name: Source Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped. +`
	`} +`
	`` `target` : \{ + ``
	`display name: Target Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped to. +`
	`} +`
	`}`

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.32. ServiceNow Connector and Datasource Configuration

The ServiceNow Datasource retrieves data from ServiceNow repository via the ServiceNow REST API. ServiceNow records are stored in named tables.

The ServiceNow connector fetches records from these tables and transforms them into Fusion documents according to the datasource configuration. Access to the ServiceNow requires both a ServiceNow username and password, as well as an OAuth client password and token.

13.32.1. Configuration

Property	Description
batch_size Batch size	Number of records to fetch in batch requests. The default is 100 to avoid the ServiceNow REST request timeout of 60 seconds. Only increase the batch size if you are sure a higher batch size will not trigger the timeout. type: `integer` default value: ‘100’ exclusiveMaximum: false exclusiveMinimum: false maximum: 10000 minimum: 1
collection Collection	Collection documents will be indexed to. type: `string`
diagnostic_mode Diagnostic Mode	Enable to print more detailed information to the logs about each request. type: `boolean` default value: ‘false’
enable_security_trimming Enable Security Trimming	Security trimming restricts query results to records the user is allowed to access by indexing user access information with other document metadata. Enable this option to index user access metadata. type: `boolean` default value: ‘false’

Property	Description
exclude_field_name_value Exclude records by field values	Records can be excluded based on values of specific fields. Enter exclusions in the format fieldName=fieldValue, e.g., workflow_state=review. type: `array` of `string`
oauth_application_client_id OAuth application Client Id *required*	OAuth application Client ID. This ID is created after registering the OAuth endpoint for the ServiceNow instance. type: `string` minLength: 1
oauth_application_client_secret OAuth application Client Secret *required*	OAuth application Client Secret. This key is created after registering the OAuth endpoint for the ServiceNow instance. type: `string` minLength: 1
servicenow_instance_url ServiceNow Instance URL *required*	The ServiceNow instance address. type: `string` minLength: 1
servicenow_password ServiceNow password *required*	Password to access the ServiceNow instance. type: `string` minLength: 1
servicenow_username ServiceNow username *required*	A user with access to all of the tables configured below. This user should also have access to related tables, as appropriate, in order to retrieve content referenced in other tables. type: `string` minLength: 1

Property	Description
tables_to_crawl ServiceNow tables to crawl	ServiceNow tables to fetch content from. At least one table name should be entered type: `array` of `string` minimum number of items (`minItems`): 1

Field Mapping

Initial Mappings	Description

mappings
Field mappings
	List of mapping rules
	type: `array` of `object`
	object attributes: \{
	`operation` : \{
	display name: Operation
	type: `string`
	default value: ‘copy’
	description : The type of mapping to perform: move, copy, delete, add, set, or keep
	enum: \{ copy move delete set add keep }
	`} +`
	`` `source` _ (required)_ : \{ + ``
	`display name: Source Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped. +`
	`} +`
	`` `target` : \{ + ``
	`display name: Target Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped to. +`
	`} +`
	`}`

reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Initial Mappings	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.33. SharePoint Connector and Datasource Configuration

The SharePoint connector retrieves content and metadata from an on-premises SharePoint repository.

To retrieve content from cloud-based SharePoint repositories, see the SharePoint Online connector.

This connector can access a SharePoint repository running on the following platforms:

- Microsoft SharePoint 2010
- Microsoft SharePoint 2013
- Microsoft SharePoint 2016

See this tutorial about configuring a SharePoint datasource and enabling security trimming:

When crawling, the connector discovers SharePoint contents in the following order: sites, then sub-sites (children). A site may contain:

- Sub-sites
- Generic Lists
    - List Items
        - Attachments
- Document Libraries
    - Folders
    - Documents

When the connector re-crawls a SharePoint repository, each previously crawled URL is accessed before any newly discovered objects, but no order is guaranteed. The connector uses a cache to store retrieved parent objects to avoid unnecessary requests. The last modified date of each object is retrieved to determine if it has changed since the last crawl. If it has changed, a new request is made to retrieve the changes. If it has not changed, the object is skipped and no additional request is made.

The connector uses SOAP to connect to and retrieve documents, lists, and other objects for indexing. It does not access a SharePoint site in the same way that a regular user does, and it needs additional privileges to use the SOAP interface to SharePoint.

The connector can be configured to work with Active Directory (AD) or LDAP to retrieve the ACLs for each object, which can then be used for security trimming at query time. In order to use security trimming to restrict user access to SharePoint objects, the the authenticated user must have sufficient privileges to read every document in the system and determine which users can access them. The permissions requirements are explained below.

13.33.1. SharePoint Permissions

SharePoint security trimming restricts access to documents based on user permissions. There are two types of permissions in SharePoint:

- Site permissions, which are:
    - managed by SharePoint
    - customizable for each site or subsite

  - inherited by subsites as the default permissions
  - grantable to users and groups
- User permissions, which are:
  - assigned by group membership, when groups have been configured and provided permissions
  - assigned directly to the user

These permissions are stored as ACLs. When the SharePoint server is configured with security trimming set to "true", then documents retrieved from SharePoint have the set of all ACLs stored in a `acl_ss` field on each document.

At search time, the ACLs are used to verify if a user has access to a document. This is configured in a query pipeline with a Security Trimming Query Stage.

To crawl all the sites and subsites, the authenticated user must belong the site administrators group. If not, Fusion can still crawl and complete the job, but the crawled data will be limited by the user's privileges. In addition, a WARNING message will appear in the `connector.log` indicating that the user is not site administrator and therefore unable to get sites from site collections. The message starts with `Authorization Error (401)`.

Required Permissions

The SharePoint datasource must be configured with the name of a user who has sufficient permissions to crawl the entire site. These permissions require use of a custom Permission Policy. The required permissions correspond to the concept of Site Collection Auditor, a permission type which is not the same as a Site Administrator, but requires almost all of the Site Administrator privileges.

You will need to work with your SharePoint administrator to ensure that the account used by Fusion has all of the permissions listed in the following table:

Permission Type	Permission	Description
Site Collection Auditor		Full Read access for the entire site collection, including reading permissions and configuration data.
List	View Items	View items in lists and documents in document libraries.
List	Open Items	View the source of documents with server-side file handlers.
List	View Versions	View past versions of a list item or document.
Site	Browse Directories	Enumerate files and folders in a Web site using SharePoint Designer and WebDAV interfaces.
Site	View Pages	View pages in a Web site.
Site	Enumerate Permissions	Enumerate permissions on the Web site, list, folder, document, or list item.
Site	Browse User Information	View information about users of the Web site.

Permission Type	Permission	Description
Site	Use Remote Interfaces	Use SOAP, WebDAV, Client Object Model, or SharePoint Designer interfaces to access the Web site.
Site	Open	Open a Web site, list or folder in order to access items inside that container.

Troubleshooting Permission Issues

When the connector is configured using a SharePoint username without sufficient privileges, the Fusion connectors log file `fusion/var/log/connectors/connectors.log` contains an error like the following:

```
crawler.common.sharepoint.exception.SharePointException: Server was unable to process request. ---> Attempted
to perform an unauthorized operation. at
crawler.common.sharepoint.service.BaseService.analyzeResponse(BaseService.java:194) ~[classes/:?] at
crawler.common.sharepoint.service.SiteDataService.getContentBySiteOrList(SiteDataService.java:169)
~[classes/:?] at com.lucidworks.permissions.Main.test1(Main.java:50) [classes/:?] at
com.lucidworks.permissions.Main.main(Main.java:32) [classes/:?]
```

This user’s permissions may be sufficient to connect via SOAP and read the documents, but not sufficient to get the ACLs and other associated metadata. This may result in complete lack of access to documents, or access to unauthorized documents. Confirm that the configured SharePoint user has the required privileges.

13.33.2. Configuration

Connector-specific Properties

Property	Description
f.avoid_ssl_hostname_verification Avoid SSL hostname verification	Enable this when using self-signed certificates, or in cases where the CN on an SSL certificate does not match the IP of the server. type: `boolean` default value: ‘true’
f.domain SharePoint domain	Authentication domain for the SharePoint user. type: `string`
f.enable_http_headers_debugging Enable HTTP headers debugging	Prints DEBUG level information to the logs. type: `boolean` default value: ‘false’

Property	Description
f.log_sharepoint_xml Log Sharepoint Soap XML	When analyzing SharePoint crawls, it can be helpful to log the Soap XML messages between Fusion and SharePoint. By selecting this, SharePoint will log the xml of all messages to the connectors log. type: `boolean` default value: ‘false’
f.maxSizeBytes Maximum file size (bytes)	Maximum size, in bytes, of a document to crawl. type: `integer` default value: ‘4194304’
f.minSizeBytes Minimum file size (bytes)	Minimum size, in bytes, of a document to crawl. type: `integer` default value: ‘0’
f.password SharePoint password	Password for the SharePoint user. type: `string`
f.remove_prepended_ids Remove prepended IDs	If fields have been defined to include PrependIds, this option will remove those IDs before indexing. type: `boolean` default value: ‘true’
f.replace_invalid_xml_entities Replace invalid xml entities	Replace invalid xml entities to avoid the error: "Unmarshalling Error: Illegal character entity: expansion character …" type: `boolean` default value: ‘false’
f.sharepoint_services_timeout Sharepoint services timeout	Time in milliseconds to wait for a server response. type: `integer` default value: ‘600000’

Property	Description
f.username SharePoint username	Name of a SharePoint user who has the required permissions to access SharePoint via the SOAP API. type: `string`

Authentication

Property	Description
f.username SharePoint username *required*	Name of a SharePoint user who has the required permissions to access SharePoint via the SOAP API. type: `string`
f.password SharePoint password *required*	Password for the SharePoint user. type: `string`
f.domain SharePoint domain *required*	Authentication domain for the SharePoint user. type: `string`
f.avoid_ssl_hostname_verification Avoid SSL hostname verification	Enable this when using self-signed certificates, or in cases where the CN on an SSL certificate does not match the IP of the server. type: `boolean` default value: ‘true’
f.sharepoint_services_timeout Sharepoint services timeout	Time in milliseconds to wait for a server response. type: `integer` default value: ‘600000’
f.enable_http_headers_debugging Enable HTTP headers debugging	Prints DEBUG level information to the logs. type: `boolean` default value: ‘false’

Property	Description
f.remove_prepended_ids Remove prepended IDs	If fields have been defined to include PrependIds, this option will remove those IDs before indexing. type: `boolean` default value: ‘true’
f.log_sharepoint_xml Log Sharepoint Soap XML	When analyzing SharePoint crawls, it can be helpful to log the Soap XML messages between Fusion and SharePoint. By selecting this, SharePoint will log the xml of all messages to the connectors log. type: `boolean` default value: ‘false’
f.replace_invalid_xml_entities Replace invalid xml entities	Replace invalid xml entities to avoid the error: "Unmarshalling Error: Illegal character entity: expansion character …" type: `boolean` default value: ‘false’

Security Trimming

Property	Description
enable_security_trimming Enable Security Trimming	type: `object` object attributes: \{ }

Limit Documents

Property	Description
includeRegexes Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: `array` of `string`

Property	Description
excludeRegexes Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: `array` of `string`
includeExtensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. type: `array` of `string`
excludeExtensions Excluded file extensions	File extensions that should not to be fetched. This will limit this datasource to all extensions except this list. type: `array` of `string`
depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: '-1'
maxItems Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: '-1'
f.maxSizeBytes Maximum file size (bytes)	Maximum size, in bytes, of a document to crawl. type: `integer` default value: '4194304'
f.minSizeBytes Minimum file size (bytes)	Minimum size, in bytes, of a document to crawl. type: `integer` default value: '0'

Property	Description
delete Delete dead URIs	Set to true to remove documents from the index when they can no longer be accessed as unique documents. type: `boolean` default value: ‘true’
deleteErrorsAfter Fetch failure allowance	Number of fetch failures to tolerate before removing a document from the index. The default of -1 means no fetch failures will be removed. type: `integer` default value: ‘-1’

Complex Documents

Property	Description
splitCSV Split CSV and archives	Set to true to split CSV/TSV files into a document per row and split archives into individual files. type: `object` object attributes: \{ }
splitArchives Split archives	Set to true for .zip, .tar, .tar.gz, .tgz, .jar, .bzip, .bzip2, .cpio, and .dump files to be opened and documents found indexed as distinct documents. type: `boolean` default value: ‘true’

Dedupe

Property	Description
dedupe Dedupe documents	If true, documents will be deduplicated. Deduplication can be done based on an analysis of the content, on the content of a specific field, or by a JavaScript function. If neither a field nor a script are defined, content analysis will be used. type: `boolean` default value: ‘false’
dedupeSaveSignature Save dedupe signature	If true,the signature used for dedupe will be stored in a 'dedupeSignature_s' field. Note this may cause errors about 'immense terms' in that field. type: `boolean` default value: ‘false’
dedupeField Dedupe field	Field to be used for dedupe. Define either a field or a dedupe script, otherwise the full raw content of each document will be used. type: `string`
dedupeScript Dedupe script	Custom javascript to dedupe documents. The script must define a 'genSignature(content)\{}' function, but can use any combination of document fields. The function must return a string. type: `string`

Recrawl Rules

Property	Description
refreshAll Recrawl all items	Set to true to always recrawl all items found in the crawldb. type: `boolean` default value: ‘true’

Property	Description
refreshStartLinks Recrawl start links	Set to true to recrawl items specified in the list of start links. type: `boolean` default value: ‘false’
refreshErrors Recrawl errors	Set to true to recrawl items that failed during the last crawl. type: `boolean` default value: ‘false’
refreshOlderThan Recrawl age	Number of seconds to recrawl items whose last fetched date is longer ago than this value. type: `integer` default value: ‘-1’
refreshIDPrefixes Recrawl ID prefixes	A prefix to recrawl all items whose IDs begin with this value. type: `array` of `string`
refreshIDRegexes Recrawl ID regexes	A regular expression to recrawl all items whose IDs match this pattern. type: `array` of `string`
refreshScript Recrawl script	A JavaScript function ('shouldRefresh()') to customize the items recrawled. type: `string`
forceRefresh Force recrawl	Set to true to recrawl all items even if they have not changed since the last crawl. type: `boolean` default value: ‘false’

Crawl Performance

Property	Description
chunkSize Fetch batch size	The number of items to batch for each round of fetching. The default is 50. type: `integer` default value: ‘50’
fetchThreads Fetch threads	The number of threads to use during fetching. The default is 5. type: `integer` default value: ‘5’
fetchDelayMS Fetch delay	Number of milliseconds to wait between fetch requests. The default is 0. This property can be used to throttle a crawl if necessary. type: `integer` default value: ‘0’
emitThreads Emit threads	The number of threads used to send documents from the connector to the index pipeline. The default is 5. type: `integer` default value: ‘5’
retryEmit Retry emits	Set to true for emit batch failures to be retried on a document-by-document basis. type: `boolean` default value: ‘true’
failFastOnStartLinkFailure Fail crawl if start links are invalid	If true, when Fusion cannot connect to any of the provided start links, the crawl is stopped and an exception logged. type: `boolean` default value: ‘true’

Crawl History

Property	Description
crawlDBType Crawl database type	The type of crawl database to use, in-memory or on-disk. type: `string` default value: ‘in-memory’ enum: \{ in-memory on-disk }
indexCrawlDBToSolr Index crawl database to Solr	EXPERIMENTAL: Set to true to index the crawl-database into a 'crawldb_' collection in Solr. type: `boolean` default value: ‘false’

Field Mapping

Property	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` _ (required)_ : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }

Property	Description
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
pipeline Pipeline ID	The index pipeline used to process documents.

13.34. Slack Connector and Datasource Configuration

The Slack connector is used to retrieve data from a Slack service. The connector sends requests to the Slack REST API.

Note	This connector is deprecated in Fusion 2.4.

13.34.1. Configuration

Property	Description
collection Collection	Collection documents will be indexed to. type: `string`
enable_security_trimming Enable Security Trimming	Add permissions to each document to control access type: `boolean` default value: ‘false’
im_messages IM Messages	Index IM / Direct Messages type: `boolean` default value: ‘false’
politeness Throttle	Throttle Rate Of API Calls type: `boolean` default value: ‘true’
private_groups Private Groups	Index Private Groups type: `boolean` default value: ‘false’
public_channels Public Channels	Index Public Channels type: `boolean` default value: ‘true’

Property	Description
throttle_interval Throttle Rate	Throttle Rate Of API Calls To One Request Per N milliseconds type: integer default value: ‘1000’
token API authentication token *required*	type: string minLength: 1

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: array of object object attributes: \{ operation : \{ display name: Operation type: string default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

Initial Mappings	Description
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.35. Solr Connector and Datasource Configuration

A Solr connector pulls documents from an external standalone Solr instance or SolrCloud cluster using Solr's javabin response type and streaming response parser.

For Solr v4.7 and greater, cursorMark deep-paging is used. For earlier versions of Solr, standard paging (start+rows) is used.

The following Solr components and parameters can be configured:

- collection/core (also allows default/empty core)
- query (: by default)
- filter queries
- query parser
- request handler (defaults to /select)
- stored fields to retrieve

Also, since cursorMark deep paging should be used when possible:

- sort spec (default: id asc)

This connector can be configured to store information about datasources and the data ingested in a ConnectorDB crawldb instance.

13.35.1. Limitations

1. Cannot do incremental crawls. (May be possible to do so in the future using source Solr docs' version field.)
2. Cannot do manual filtered deep paging.
3. Doesn't check that both sort spec and field list contain uniqueKey field.
4. Cannot handle encrypted connection to Solr

13.35.2. Configuration

Property	Description
collection Collection	Collection documents will be indexed to. type: `string`
commit_on_finish Solr commit on finish	Set to true for a request to be sent to Solr after the last batch has been fetched to commit the documents to the index. type: `boolean` default value: 'true'

Property	Description
solr_base_url Standalone Solr server base URL	If using a single Solr instance, enter the base URL, e.g., http://solrhost.example.com:8983/solr/. type: `string`
solr_field_list Stored fields to retrieve	Fields to fetch from the source Solr instance/cluster, which must be stored fields. Multiple field names should be separated with commas. type: `string` default value: ‘*’ minLength: 1
solr_filter_queries Filter queries to execute	Filter queries to select documents from the source Solr instance/cluster. Multiple filter queries should be separated with commas. type: `string`
solr_page_size Page size	Number of rows per request to Solr. type: `integer` default value: ‘100’
solr_query Query to execute	Query to select documents from the source Solr instance/cluster. If not defined, the default query **:** will be used. type: `string` default value: ‘**:**’ minLength: 1
solr_query_parser Query parser	The query parser to use for the request. type: `string`

Property	Description
solr_request_handler Solr request handler	The request handler to use for the request to the Solr instance/cluster. type: `string` default value: ‘/select’ minLength: 1
solr_sort_spec Sort specification	Sort order for the request. The uniqueKey field must be included as one of the sorted fields. type: `string` default value: ‘id asc’ minLength: 1
source_collection Source collection	Collection or Core name in the source Solr instance/cluster. If not defined, the default core will be used. type: `string`
verify_access Validate access	Set to true to require successful connection to the filesystem before saving this datasource. type: `boolean` default value: ‘true’
zk_host_string SolrCloud ZooKeeper host string	If using a SolrCloud instance, enter the ZooKeeper connect string, e.g., zkServerA:2181,zkServerB:2181,zkServerC:2181/solr. type: `string`

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Initial Mappings	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.36. SolrXML Connector and Datasource Configuration

The SolrXML connector indexes XML files formatted according to Solr's XML structure. It is not a generic XML file crawler; it can only index SolrXML-formatted documents.

Per the Solr standard, all XML files must include the `<add>` tag in order for the documents to be added to the Fusion index.

13.36.1. The SolrXML Format

As described in the Solr Reference Guide section on using Solr's updateHandlers, an XML document formatted for Solr must conform to a very specific structure. There are three general elements that are used:

- <add> introduces one or more documents to be added to the index.
- <doc> introduces the fields that make up a single document.
- <field> defines the content for each field of the document.

For example, this is very simple XML including only one document:

```
<add>
  <doc>
   <field name="id">doc1</field>
   <field name="title">My Solr Document</field>
   <field name="body">This is the body of my document.</field>
  </doc>
</add>
```

The fields can be any field that is defined in your schema, or you can use dynamic field rules to create fields during indexing.

The elements can take some attributes to define document overwrites, commit rules and field or document boosts. See the Solr Reference Guide section on XML-formatted updates for more details.

13.36.2. Configuration

Property	Description
collection Collection	Collection documents will be indexed to. type: `string`
commit_on_finish Solr commit on finish	Set to true for a request to be sent to Solr after the last batch has been fetched to commit the documents to the index. type: `boolean` default value: 'true'

Property	Description
exclude_paths Exclude paths	An array of regular expression patterns that indicate documents to be excluded from the index. Multiple expressions can be separated by commas. type: `array` of `string` default value: [ ]
generate_unique_key Generate unique key?	If true, a unique identifier will be added to each document. In most cases, this is an 'id' field, unless it was changed in your implementation. If your documents already include an ID field, you can set this to false. type: `boolean` default value: ‘true’
include_datasource_metadata Include datasource metadata?	Set True to add '_lw_data_source_s' and '_lw_data_source_type_s' fields to each document in addition to fields found in the file. These fields will ensure these documents are associated with this datasource for faceting, information shown in the UI, or later document removal. type: `boolean` default value: ‘true’
include_paths Include paths	An array of regular expression patterns that indicate documents to be included in the index. Multiple expressions can be separated by commas. type: `array` of `string` default value: [`.*\.xml`,]
max_docs Max documents	The maximum number of documents to crawl. Use -1 to index all documents found. type: `integer` default value: ‘-1’

Property	Description
path Path *required*	Name of the file to read, or directory containing files to read. type: `string` minLength: 1
url URL	Read-only value that shows the absolute path. type: `string` minLength: 1
verify_access Validate access	Set to true to require successful connection to the filesystem before saving this datasource. type: `boolean` default value: ‘true’

Field Mapping

Initial Mappings	**Description**
mappings Field mappings	List of mapping rules type: array of object object attributes: \{ operation : \{ display name: Operation type: string default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: boolean default value: ‘false’

Initial Mappings	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.37. Subversion Connector and Datasource Configuration

This connector requires a Subversion client that is compatible which javahl.

Fusion ships with binaries for Windows and Linux which can be used if a javahl-compatible client is not already installed. These are found in subdirectories of the fusion/apps/connectors/resources/lucid.anda/subversion directory.

13.37.1. Configuration

Connector-specific Properties

Property	Description
f.subversion_batch_size Subversion number of revisions in batch	Number of records to fetch in each batch request. type: `integer` default value: ‘100’
f.subversion_configuration_directory Subversion configuration directory	Path to a configuration directory that includes a file named 'servers' to suport self-signed certificates. If the Subversion repository does not use SSL or does not use self-signed certificates, this can be left empty. type: `string` default value: '`’
f.subversion_native_library Subversion JavaHL library path	Path to the libsvnjavahl-1.so (Linux) or libsvnjavahl-1.dll (Windows) library. If using the libaries included with Fusion, you do not need to specify the path. type: `string` default value: ‘fusion-svn-binaries’
f.subversion_password Subversion Password	Password for user, if required. type: `string` default value: '`’
f.subversion_username Subversion Username	Username with permissions to access the repository, if the Subversion repository requires authentication. type: `string` default value: '`’

Limit Documents

Property	Description
depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: '-1'
maxItems Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: '-1'
includeExtensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. type: `array` of `string`
includeRegexes Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: `array` of `string`
excludeExtensions Excluded file extensions	File extensions that should not to be fetched. This will limit this datasource to all extensions except this list. type: `array` of `string`
excludeRegexes Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: `array` of `string`
delete Delete dead URIs	Set to true to remove documents from the index when they can no longer be accessed as unique documents. type: `boolean` default value: 'true'

Property	Description
deleteErrorsAfter Fetch failure allowance	Number of fetch failures to tolerate before removing a document from the index. The default of -1 means no fetch failures will be removed. type: `integer` default value: '-1'

Crawl Performance

Property	Description
f.subversion_batch_size Subversion number of revisions in batch	Number of records to fetch in each batch request. type: `integer` default value: '100'
chunkSize Fetch batch size	The number of items to batch for each round of fetching. The default is 50. type: `integer` default value: '50'
fetchThreads Fetch threads	The number of threads to use during fetching. The default is 5. type: `integer` default value: '5'
fetchDelayMS Fetch delay	Number of milliseconds to wait between fetch requests. The default is 0. This property can be used to throttle a crawl if necessary. type: `integer` default value: '0'
emitThreads Emit threads	The number of threads used to send documents from the connector to the index pipeline. The default is 5. type: `integer` default value: '5'

Property	Description
retryEmit Retry emits	Set to true for emit batch failures to be retried on a document-by-document basis. type: `boolean` default value: ‘true’
failFastOnStartLinkFailure Fail crawl if start links are invalid	If true, when Fusion cannot connect to any of the provided start links, the crawl is stopped and an exception logged. type: `boolean` default value: ‘true’

Dedupe

Property	Description
dedupe Dedupe documents	If true, documents will be deduplicated. Deduplication can be done based on an analysis of the content, on the content of a specific field, or by a JavaScript function. If neither a field nor a script are defined, content analysis will be used. type: `boolean` default value: ‘false’
dedupeSaveSignature Save dedupe signature	If true,the signature used for dedupe will be stored in a 'dedupeSignature_s' field. Note this may cause errors about 'immense terms' in that field. type: `boolean` default value: ‘false’
dedupeField Dedupe field	Field to be used for dedupe. Define either a field or a dedupe script, otherwise the full raw content of each document will be used. type: `string`

Property	Description
dedupeScript Dedupe script	Custom javascript to dedupe documents. The script must define a 'genSignature(content)\{}' function, but can use any combination of document fields. The function must return a string. type: `string`

Recrawl Rules

Property	Description
refreshAll Recrawl all items	Set to true to always recrawl all items found in the crawldb. type: `boolean` default value: 'true'
refreshStartLinks Recrawl start links	Set to true to recrawl items specified in the list of start links. type: `boolean` default value: 'false'
refreshErrors Recrawl errors	Set to true to recrawl items that failed during the last crawl. type: `boolean` default value: 'false'
refreshOlderThan Recrawl age	Number of seconds to recrawl items whose last fetched date is longer ago than this value. type: `integer` default value: '-1'
refreshIDPrefixes Recrawl ID prefixes	A prefix to recrawl all items whose IDs begin with this value. type: `array` of `string`

Property	Description
refreshIDRegexes Recrawl ID regexes	A regular expression to recrawl all items whose IDs match this pattern. type: `array` of `string`
refreshScript Recrawl script	A JavaScript function ('shouldRefresh()') to customize the items recrawled. type: `string`
forceRefresh Force recrawl	Set to true to recrawl all items even if they have not changed since the last crawl. type: `boolean` default value: 'false'

Crawl History

Property	Description
retainOutlinks Retain links in the crawldb	Set to true for links found during fetching to be stored in the crawldb. type: `boolean` default value: 'true'
aliasExpiration Alias expiration	The number of crawls after which an alias will expire. The default is 1 crawl. type: `integer` default value: '1'
crawlDBType Crawl database type	The type of crawl database to use, in-memory or on-disk. type: `string` default value: 'in-memory' enum: \{ in-memory on-disk }

Property	Description
indexCrawlDBToSolr Index crawl database to Solr	EXPERIMENTAL: Set to true to index the crawl-database into a 'crawldb_' collection in Solr. type: `boolean` default value: ‘false’
reevaluateCrawlDBOnStart Reevaluate crawldb on start?	Reevaluate exisiting crawldb entries for legality on startup? type: `boolean` default value: ‘false’

Field Mapping

Property	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

Property	Description
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
pipeline Pipeline ID	The index pipeline used to process documents.

13.38. Twitter Search Connector and Datasource Configuration

The Twitter Search connector Twitter Search, uses Twitter's search API to query Twitter for tweets that match specific parameters. It allows querying for any keyword, location or other query terms.

13.38.1. Registering for Twitter Credentials

In order to successfully configure either Twitter connector, you must first register your application with Twitter and accept their terms of service. The registration process will provide you with the required OAuth tokens you need to access either API. To get the tokens, follow these steps:

1. Make sure you have a Twitter account, and go to https://dev.twitter.com/ and sign in.
2. After signing in, choose 'My Applications' from the pull down menu at the upper right that shows a thumbnail of your Twitter profile picture (if you have one). Then choose "Create New App" and fill out the required details. The callback field can be skipped, but you must accept the Terms of Service. To save your information, choose "Create Your Twitter Application" to register your application.
3. The next page will contain the Consumer Key and Consumer Secret, which you will need to configure the data source in Fusion.
4. At the bottom of the same page, choose "Create My Access Token".
5. The next page will contain the Access Token and Token Secret, which you will also need to configure the data source in Fusion.

While you need a Twitter account to register an application, you do not use your Twitter username and password to configure this data source. The APIs will only use the Consumer Key, Consumer Secret, Access Token, and Token Secret information as authentication, so store it where you can access it while configuring the data source.

13.38.2. Configuration

Property	Description
access_token Access Token *required*	The OAuth Access Token is provided by Twitter when registering the application. type: `string` minLength: 1
collection Collection	Collection documents will be indexed to. type: `string`
commit_on_finish Solr commit on finish	Set to true for a request to be sent to Solr after the last batch has been fetched to commit the documents to the index. type: `boolean` default value: 'true'

Property	Description
consumer_key Consumer Key *required*	The OAuth Consumer Key is provided by Twitter when registering the application. type: `string` minLength: 1
consumer_secret Consumer Secret *required*	The OAuth Consumer Secret is provided by Twitter when registering the application. type: `string` minLength: 1
lang Language	Restrict Tweets to the given language type: `string` default value: ‘en’ enum: \{ aa ab ae af ak am an ar as av ay az ba be bg bh bi bm bn bo br bs ca ce ch co cr cs cu cv cy da de dv dz ee el en eo es et eu fa ff fi fj fo fr fy ga gd gl gn gu gv ha he hi ho hr ht hu hy hz ia id ie ig ii ik in io is it iu iw ja ji jv ka kg ki kj kk kl km kn ko kr ks ku kv kw ky la lb lg li ln lo lt lu lv mg mh mi mk ml mn mo mr ms mt my na nb nd ne ng nl nn no nr nv ny oc oj om or os pa pi pl ps pt qu rm rn ro ru rw sa sc sd se sg si sk sl sm sn so sq sr ss st su sv sw ta te tg th ti tk tl tn to tr ts tt tw ty ug uk ur uz ve vi vo wa wo xh yi yo za zh zu }
max_docs Max Documents	The maximum number of documents to pull down, as a long. -1 for no limit type: `integer` default value: ‘-1’
queries Queries	A list of queries to perform type: `array` of `string`

Property	Description
sleep Sleep	The amount of time, in milliseconds, to sleep when listening so as to not get throttled type: `integer` default value: ‘10000’
token_secret Token Secret *required*	The OAuth Token Secret is provided by Twitter when registering the application. type: `string` minLength: 1

Field Mapping

Initial Mappings	Description

mappings
 	
Field mappings
	List of mapping rules
	
	type: `array` of `object`
	
	object attributes: \{
	`operation` : \{
	display name: Operation
	type: `string`
	default value: ‘copy’
	description : The type of mapping to perform: move, copy, delete, add, set, or keep
	enum: \{ copy move delete set add keep }
	
	`} +`
	`` `source` _ (required)_ : \{ + ``
	`display name: Source Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped. +`
	`} +`
	`` `target` : \{ + ``
	`display name: Target Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped to. +`
	`} +`
	}

Initial Mappings	Description
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: boolean default value: ‘false’
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: object object attributes: \{ operation : \{ display name: Operation type: string default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: boolean default value: ‘false’

Property	Description
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.39. Twitter Stream Connector and Datasource Configuration

The Twitter Stream connector uses Twitter’s streaming API to continually index Twitter. The datasource can be configured to limit tweets or it can be run indefinitely, until Twitter cuts off your access or you stop the datasource. This connector will only retrieve tweets created after the datasource has been started.

This connector type is "lucid.twitter.search" and the plugin type is "twitter_search".

13.39.1. Registering for Twitter Credentials

In order to successfully configure either Twitter connector, you must first register your application with Twitter and accept their terms of service. The registration process will provide you with the required OAuth tokens you need to access either API. To get the tokens, follow these steps:

1. Make sure you have a Twitter account, and go to https://dev.twitter.com/ and sign in.
2. After signing in, choose 'My Applications' from the pull down menu at the upper right that shows a thumbnail of your Twitter profile picture (if you have one). Then choose "Create New App" and fill out the required details. The callback field can be skipped, but you must accept the Terms of Service. To save your information, choose "Create Your Twitter Application" to register your application.
3. The next page will contain the Consumer Key and Consumer Secret, which you will need to configure the data source in Fusion.
4. At the bottom of the same page, choose "Create My Access Token".
5. The next page will contain the Access Token and Token Secret, which you will also need to configure the data source in Fusion.

While you need a Twitter account to register an application, you do not use your Twitter username and password to configure this data source. The APIs will only use the Consumer Key, Consumer Secret, Access Token, and Token Secret information as authentication, so store it where you can access it while configuring the data source.

13.39.2. Configuration

Property	Description
access_token Access Token *required*	The OAuth Access Token is provided by Twitter when registering the application. type: `string` minLength: 1
collection Collection	Collection documents will be indexed to. type: `string`

Property	Description
commit_on_finish Solr commit on finish	Set to true for a request to be sent to Solr after the last batch has been fetched to commit the documents to the index. type: `boolean` default value: ‘true’
consumer_key Consumer Key *required*	The OAuth Consumer Key is provided by Twitter when registering the application. type: `string` minLength: 1
consumer_secret Consumer Secret *required*	The OAuth Consumer Secret is provided by Twitter when registering the application. type: `string` minLength: 1
filter_follow Filter Follow	Set of users (user ids) to track type: `array` of `string`
filter_locations Filter Locations	Set of bounding boxes (e.g. 'left,bottom,right,top' lat/long coordinates) type: `array` of `string`
filter_track Filter Track	Keywords or phrases to track type: `array` of `string`
max_docs Max Documents	The maximum number of documents to pull down, as a long. -1 for no limit type: `integer` default value: ‘-1’

Property	Description
sleep Sleep	The amount of time, in milliseconds, to sleep when listening so as to not get throttled type: `integer` default value: ‘10000’
token_secret Token Secret *required*	The OAuth Token Secret is provided by Twitter when registering the application. type: `string` minLength: 1
url URL	The URL used by the Twitter Stream type: `string` default value: ‘https://stream.twitter.com’

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Initial Mappings	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.40. Web Connector and Datasource Configuration

The Web connector is used to retrieve data from a Web site using HTTP and starting from a specified URL.

13.40.1. Limiting Crawl Scope

The connector works by going to the seed page (the "startURIs" specified in the configuration form), collecting the content for indexing, and extracting any links to other pages. It then follows those links to collect content on other pages, extracting links to those pages, etc.

When creating a Web data source, pay attention to the "Max depth" and "Restrict To Tree" parameters (also known as "c.depth" and "c.restrictToTree" in the REST API). These properties will help limit the scope of your crawl to prevent an "unbounded" crawl that could continue for a long time, particularly if you are crawling a site with links to many pages outside the main site. An unbounded crawl may also cause memory errors in your system.

The connector keeps track of URIs it has seen, and many of the properties relate to managing the resulting database of entries. If the connector finds a standard redirect, it will track that the redirected URI has an alias, and will not re-evaluate the URI on its next runs until the alias expiration has passed. Documents that were found to be duplicates, if de-duplication is enabled, are also added to the alias list and are not re-evaluated until the alias expiration has passed.

Regular expressions can be used to restrict the crawl either by defining URI patterns that should be followed or URI patterns that should not be followed.

Additionally, specific patterns of the URI can also be defined to define URIs that should not be followed.

13.40.2. Extracting Content from Pages

The connector supports several approaches to extracting and filtering content from pages. When analyzing the HTML of a page, the connector can specifically include or exclude elements based on the HTML tag, the tag ID, or the tag class (such as a 'div' tag, or the '#content' tag ID).

Specific tags can be selected to become fields of the document if needed. For example, all content from <h1> tags can be pulled into a 'h1' field, and with field mapping be transformed into document titles.

For even more advanced capabilities, you can use jsoup selectors to find elements in the content to include or exclude from the content.

While field mapping is generally a function of the index pipeline, you can define some initial mapping to occur during the crawl. The 'initial mappings' property for each web datasource is pre-defined with three mappings, to move 'fetchedDates' to a 'fetchedDates_dts' field, to move 'lastModified' to a 'lastModified_dt' field and to move 'length' to a 'length_l' field.

Finally, the crawler is able to do de-duplication of crawled content. You can define a specific field to use for this de-duplication (such as title, or another field), or you can use the full raw content as the default.

13.40.3. Sitemap Processing

As of Fusion 1.1.2, crawling sitemaps is supported. Simply add the URL(s) of the sitemap to the f.sitemapURLs property ("Sitemap URLs" in the UI) and all of the URLs found in a sitemap will be added to the list of URLs to crawl. If your site has a sitemap index, i.e., a sitemap that points to other sitemaps, that is also supported and the URLs found through each sitemap will be added to the list of URLs to crawl.

If you want to configure your datasource to only crawl the sitemap file, you must add the sitemap URL to both the startLinks property (because that is a required property for a datasource) and also to the f.sitemapsURL property so it is properly treated as a sitemap by the connector when it starts.

13.40.4. Website Authentication

The Web connector supports Basic, Digest, Form and NTLM authentication to websites.

The credentials for a crawl are stored in a credentials file that should be placed in `fusion/data/connectors/container/lucid.anda/datasourceName ` where the "datasourceName" corresponds to the name given to the datasource. After creating a datasource, this directory should be created for you. The file should be a JSON formatted file, ending with the '.json' file extension. When defining the datasource, you would pass the name of the file with the 'Authentication file' property in the UI (or 'f.credentialsFile' property if using the REST API).

All types of authentication require the credentials file to include a property called "type" which defines the type of authentication to use. After that, the required properties will vary depending on the type of authentication chosen.

Form-based Authentication

To use basic form-based authentication, use "form" for the type. The other properties are:

- ttl - The "time to live" for the session that will be created after authentication. This will have the crawler log in again after the specified time so the crawl activity doesn’t fail due to an expired session. This value is defined in seconds.
- action - The action to take to log in, i.e., the URL for the login form.
- params - The parameters for the form, likely the username and password, but any other required properties. In the example below, we are passing two parameters, the 'os_username' and the 'os_password' that are the properties expected by the system we would like to crawl.

Here is an example using form-based authentication:

```
[ {
        "credential" : {
            "type" : "form",
            "ttl" : 300000,
            "action" :
"http://some.server.com/login.action?os_destination=%2Fpages%2Fviewpage.action%3Ftitle%3DAcme%2B5%2BDocumentat
ion%26spaceKey%3DAcme5",
            "params" : {
                "os_username" : "username",
                "os_password" : "password"
            }
        }
  } ]
```

Complex Form-based Authentication

Some websites do not manage their own authentication, but rather trust a third-party authority to authenticate the user. An example of this would be websites that use SAML to log in a user via a central single-signon authority. In order to configure fusion to log in to a website like this, use "smartForm" for the type. The other properties are:

- ttl - the "time to live" for the session that will be created after authentication. This will have the crawler re-login after the specified time so the crawl activity doesn’t fail due to an expired session. This value is defined in seconds.

- loginUrl - the URL on which the first page that initializes the login chain is located
- params - a list of parameters to use for the form logins, likely the username and password, but could be other required properties. In the example below, we are passing two parameters, the 'os_username' and the 'os_password' that are the properties expected by the system we would like to crawl. Additionally we expect that once that login has happened, that a new form will be presented to the user which then posts back to where we came from. No data need to be entered in this form, which is why we include an empty `{ }` in the params list.

Here is an example using form-based authentication:

```
[ {
        "credential" : {
            "type" : "smartForm",
            "ttl" : 300000,
            "loginUrl" : "http://some.example.com/login",
            "params" : [{
                "os_username" : "username",
                "os_password" : "password"
            }, {

            } ]
        }
  } ]
```

In order to figure out what params you need to specify, turn off JavaScript in your browser and walk through the login chain. Though you normally only see a single login form on your screen, you may be surprised to find many more forms you need to submit before you get logged in when JavaScript is not available to perform those form submissions automatically. Each form in that chain needs to be represented in list of `params`. If no user input is required, simply include an empty `{ }`.

Basic and Digest Authentication

Basic and Digest authentication are simple HTTP authentication methods still in use in some places. To use either of these types use "basic" or "digest" in the credentials file for the 'type' property. Other properties are:

- host - the host of the site.
- port - the port, if any.
- userName - the username to use for authentication.
- password - the password for the userName.
- realm - the realm for the site, if any.

Example basic auth configuration:

```
[ {
        "credential" : {
            "type" : "basic",
            "ttl" : 300000,
            "userName" : "usr",
            "password" : "pswd",
            "host":"hostname.exampledomain.com”
            "port": 443
        }
  }
]
```

NTLM Authentication

To use NTLM authentication, use "ntlm" in the credentials file for the 'type' property. The other properties available are:

- host - the host of the site.
- port - the port, if any.
- userName - the username to use for authentication.
- password - the password for the userName.
- realm - the realm for the site, if any.
- domain - the domain.
- workstation - the workstation, as needed.

Example NTLM credential configuration:

```
[ {"credential" :
   { "type" : "ntlm",
     "ttl" : 300000,
     "port" : 80,
     "host" : "someHost",
     "domain" : "someDomain",
     "userName" : "someUser",
     "password" : "XXXXXXXX"
   }
} ]
```

13.40.5. Configuration

Connector-specific Properties

Property	Description
f.allowAllCertificates Allow all HTTPS certificates	If true, security checks will be performed on all SSL/TLS certificate signers and origins. This means self-signed certificates would not be supported. type: `boolean` default value: ‘false’
f.appendTrailingSlashToLinks Add trailing slash to link URLs	If true, a trailing '/' will be added to link URLs when the URL does not end in a dot ('.'). type: `boolean` default value: ‘false’
f.cookieSpec Cookie spec	type: `string` default value: ‘browser-compatibility’ enum: \{ browser-compatibility rfc-2965 best-match ignore-all }
f.credentialsFile Authentication credentials filename	Name of the file that contains the credentials for sites that require authentication to access. This file must be located in '$FUSION_HOME/data/connectors/container/lucid.anda/'. type: `string`
f.defaultCharSet Default character set	Default character set to use when one is not declared in the HTTP headers. type: `string` default value: ‘UTF-8’
f.defaultMIMEType Default MIME type	Default MIME type to use when one is not declared in the HTTP headers. type: `string` default value: ‘application/octet-stream’

Property	Description
f.discardLinkURLQueries Discard queries in link URLs	If true, query parameters found in URLs will be removed before being added to the discovery queue. type: `boolean` default value: ‘false’
f.excludeSelectors Jsoup exclusive selectors	Jsoup-formatted selectors for elements to exclude from the crawled content. Syntax for jsoup selectors is available at http://jsoup.org/apidocs/org/jsoup/select/Selector.html. type: `array` of `string`
f.excludeTagClasses Excluded tag classes	HTML tag classes of elements to exclude from the crawled content. type: `array` of `string`
f.excludeTagIDs Excluded tag IDs	HTML tag IDs of elements to exclude from the crawled content. type: `array` of `string`
f.excludeTags Excluded tags	HTML tag names of elements to exclude from the crawled content. type: `array` of `string`
f.filteringRootTags Root elements to filter	Root HTML elements whose child elements will be used to extract content. By default 'body' and 'head' elements are already included. type: `array` of `string` default value: ‘bodyhead’
f.includeSelectors Jsoup inclusive selectors	Jsoup-formatted selectors for elements to include in the crawled content. type: `array` of `string`
f.includeTagClasses Included tag classes	HTML tag classes of elements to include in the crawled content. type: `array` of `string`

Property	Description
f.includeTagIDs Included tag IDs	HTML tag IDs of elements to include in the crawled content. type: `array` of `string`
f.includeTags Included tags	HTML tag names of elements to include in the crawled content. type: `array` of `string`
f.maxSizeBytes Max file size (bytes)	Maximum size, in bytes, of a document to fetch. type: `integer` default value: '4194304'
f.obeyRobots Obey robots.txt	If true, Allow, Disallow and other rules found in a robots.txt file will be obeyed. type: `boolean` default value: 'true'
f.obeyRobotsDelay Obey robots.txt Crawl-Delay	If true, Crawl-Delay rules in robots.txt will be obeyed. Disabling this option will speed up crawling, but is considered negative behavior for sites you do not control. type: `boolean` default value: 'true'
f.proxy HTTP proxy address	Address of the HTTP proxy, if required.. This should be entered in the format host:port. type: `string`
f.respectMetaEquivRedirects Respect refresh redirects	If true, the connector will follow metatags with refresh redirects such as . type: `boolean` default value: 'false'

Property	Description
f.scrapeLinksBeforeFiltering Scrape links before filtering	If true, links will be extracted from documents before any other document processing has ocurred. By default, links are extracted after all other document processing. type: `boolean` default value: ‘false’
f.selectorFields Jsoup selector fields	List of Jsoup selectors for elements to put into their separate field in the index. The field will have the same name as the element. Syntax for jsoup selectors is available at http://jsoup.org/apidocs/org/jsoup/select/Selector.html. type: `array` of `string`
f.sitemapURLs Sitemap URLs	URLs for sitemaps, to be used a basis for link discovery. Rules found in sitemaps will not be processed. type: `array` of `string`
f.tagClassFields Tag-class fields	HTML tag classes of elements to put into their own field in the index. The field will have the same name as the tag class. type: `array` of `string`
f.tagFields Tag fields	HTML tags of elements to put into their own field in the index. The field will have the same name as the tag. type: `array` of `string`
f.tagIDFields Tag-ID fields	HTML tag IDs of elements to put into their own field in the index. The field will have the same name as the tag ID. type: `array` of `string`
f.timeoutMS Connection timeout (ms)	Time in milliseconds to wait for server response. type: `integer` default value: ‘10000’

Property	Description
f.userAgentEmail HTTP user-agent email address	Email address to use as part of connector identification. type: `string` default value: '`’
f.userAgentName HTTP user-agent name	Name the connector should use when identifying itself to a website in order to crawl it. type: `string` default value: ‘Lucidworks-Anda/2.0’
f.userAgentWebAddr HTTP user-agent web address	Web address to use as part of connector identification. type: `string` default value: '`’

Crawl Authorization

Property	Description
f.proxy HTTP proxy address	Address of the HTTP proxy, if required.. This should be entered in the format host:port. type: `string`
f.allowAllCertificates Allow all HTTPS certificates	If true, security checks will be performed on all SSL/TLS certificate signers and origins. This means self-signed certificates would not be supported. type: `boolean` default value: ‘false’
f.credentialsFile Authentication credentials filename	Name of the file that contains the credentials for sites that require authentication to access. This file must be located in '$FUSION_HOME/data/connectors/container/lucid.anda/'. type: `string`

Property	Description
f.obeyRobots Obey robots.txt	If true, Allow, Disallow and other rules found in a robots.txt file will be obeyed. type: `boolean` default value: ‘true’

Link Discovery

Property	Description
f.sitemapURLs Sitemap URLs	URLs for sitemaps, to be used a basis for link discovery. Rules found in sitemaps will not be processed. type: `array` of `string`
restrictToTreeAllowSubdomains Allow sub-domains in restrictToTree	If true, any sub-domain will be allowed, even if the crawl is restricted to the tree of items found below the start links. type: `boolean` default value: ‘false’
restrictToTreeUseHostAndPath Use paths in restrictToTree	If true, the path in start links will be used to restrict items fetched. For example, if the start link is 'http://host.com/US', this option will limit all followed URLs to this path. type: `boolean` default value: ‘false’
restrictToTreeIgnoredHostPrefixes Ignored host prefixes	List of host prefixes to ignore when checking links for restrictToTree link-legality checks. For example, 'www.' can be ignored so links with the same domain are allowed. type: `array` of `string` default value: [` www. `]
f.respectMetaEquivRedirects Respect refresh redirects	If true, the connector will follow metatags with refresh redirects such as . type: `boolean` default value: ‘false’

Limit Documents

Property	Description
restrictToTree Restrict to sub-directories and child pages	If true, only documents found in a tree below the start links will be fetched. For Web crawls, this will only limit to the domain. type: `boolean` default value: ‘true’
depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: ‘-1’
maxItems Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: `integer` default value: ‘-1’
includeExtensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. type: `array` of `string`
includeRegexes Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: `array` of `string`
excludeExtensions Excluded file extensions	File extensions that should not to be fetched. This will limit this datasource to all extensions except this list. type: `array` of `string`
excludeRegexes Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: `array` of `string`

Property	Description
f.maxSizeBytes Max file size (bytes)	Maximum size, in bytes, of a document to fetch. type: `integer` default value: ‘4194304’

Complex Documents

Property	Description
splitCSV Split CSV and archives	Set to true to split CSV/TSV files into a document per row and split archives into individual files. type: `object` object attributes: \{ }
splitArchives Split archives	Set to true for .zip, .tar, .tar.gz, .tgz, .jar, .bzip, .bzip2, .cpio, and .dump files to be opened and documents found indexed as distinct documents. type: `boolean` default value: ‘true’

Crawler ID

Property	Description
f.userAgentName HTTP user-agent name	Name the connector should use when identifying itself to a website in order to crawl it. type: `string` default value: ‘Lucidworks-Anda/2.0’
f.userAgentEmail HTTP user-agent email address	Email address to use as part of connector identification. type: `string` default value: '`’

Property	Description
f.userAgentWebAddr HTTP user-agent web address	Web address to use as part of connector identification. type: `string` default value: '`’

Document Parsing

Property	Description
f.appendTrailingSlashToLinks Add trailing slash to link URLs	If true, a trailing '/' will be added to link URLs when the URL does not end in a dot ('.'). type: `boolean` default value: ‘false’
f.defaultCharSet Default character set	Default character set to use when one is not declared in the HTTP headers. type: `string` default value: ‘UTF-8’
f.defaultMIMEType Default MIME type	Default MIME type to use when one is not declared in the HTTP headers. type: `string` default value: ‘application/octet-stream’
f.filteringRootTags Root elements to filter	Root HTML elements whose child elements will be used to extract content. By default 'body' and 'head' elements are already included. type: `array` of `string` default value: [` body ` ` head `]
f.scrapeLinksBeforeFiltering Scrape links before filtering	If true, links will be extracted from documents before any other document processing has ocurred. By default, links are extracted after all other document processing. type: `boolean` default value: ‘false’

Property	Description
f.includeTags Included tags	HTML tag names of elements to include in the crawled content. type: `array` of `string`
f.includeTagClasses Included tag classes	HTML tag classes of elements to include in the crawled content. type: `array` of `string`
f.includeTagIDs Included tag IDs	HTML tag IDs of elements to include in the crawled content. type: `array` of `string`
f.includeSelectors Jsoup inclusive selectors	Jsoup-formatted selectors for elements to include in the crawled content. type: `array` of `string`
f.excludeTags Excluded tags	HTML tag names of elements to exclude from the crawled content. type: `array` of `string`
f.excludeTagClasses Excluded tag classes	HTML tag classes of elements to exclude from the crawled content. type: `array` of `string`
f.excludeTagIDs Excluded tag IDs	HTML tag IDs of elements to exclude from the crawled content. type: `array` of `string`
f.excludeSelectors Jsoup exclusive selectors	Jsoup-formatted selectors for elements to exclude from the crawled content. Syntax for jsoup selectors is available at http://jsoup.org/apidocs/org/jsoup/select/Selector.html. type: `array` of `string`
f.tagFields Tag fields	HTML tags of elements to put into their own field in the index. The field will have the same name as the tag. type: `array` of `string`

Property	Description
f.tagIDFields Tag-ID fields	HTML tag IDs of elements to put into their own field in the index. The field will have the same name as the tag ID. type: `array` of `string`
f.tagClassFields Tag-class fields	HTML tag classes of elements to put into their own field in the index. The field will have the same name as the tag class. type: `array` of `string`
f.selectorFields Jsoup selector fields	List of Jsoup selectors for elements to put into their separate field in the index. The field will have the same name as the element. Syntax for jsoup selectors is available at http://jsoup.org/apidocs/org/jsoup/select/Selector.html. type: `array` of `string`

Crawl Performance

Property	Description
chunkSize Fetch batch size	The number of items to batch for each round of fetching. The default is 50. type: `integer` default value: ‘50’
fetchThreads Fetch threads	The number of threads to use during fetching. The default is 5. type: `integer` default value: ‘5’
fetchDelayMS Fetch delay	Number of milliseconds to wait between fetch requests. The default is 0. This property can be used to throttle a crawl if necessary. type: `integer` default value: ‘0’

Property	Description
fetchDelayMSPerHost Fetch delay per host	If true, the 'Fetch delay (ms)' property will be applied for each host. type: `boolean` default value: ‘true’
emitThreads Emit threads	The number of threads used to send documents from the connector to the index pipeline. The default is 5. type: `integer` default value: ‘5’
failFastOnStartLinkFailure Fail crawl if start links are invalid	If true, when Fusion cannot connect to any of the provided start links, the crawl is stopped and an exception logged. type: `boolean` default value: ‘true’
retryEmit Retry emits	Set to true for emit batch failures to be retried on a document-by-document basis. type: `boolean` default value: ‘true’
f.timeoutMS Connection timeout (ms)	Time in milliseconds to wait for server response. type: `integer` default value: ‘10000’
f.obeyRobotsDelay Obey robots.txt Crawl-Delay	If true, Crawl-Delay rules in robots.txt will be obeyed. Disabling this option will speed up crawling, but is considered negative behavior for sites you do not control. type: `boolean` default value: ‘true’

Dedupe

Property	Description
dedupe Dedupe documents	If true, documents will be deduplicated. Deduplication can be done based on an analysis of the content, on the content of a specific field, or by a JavaScript function. If neither a field nor a script are defined, content analysis will be used. type: `boolean` default value: ‘false’
dedupeSaveSignature Save dedupe signature	If true,the signature used for dedupe will be stored in a 'dedupeSignature_s' field. Note this may cause errors about 'immense terms' in that field. type: `boolean` default value: ‘false’
dedupeField Dedupe field	Field to be used for dedupe. Define either a field or a dedupe script, otherwise the full raw content of each document will be used. type: `string`
dedupeScript Dedupe script	Custom javascript to dedupe documents. The script must define a 'genSignature(content)\{}' function, but can use any combination of document fields. The function must return a string. type: `string`

Recrawl Rules

Property	Description
refreshAll Recrawl all items	Set to true to always recrawl all items found in the crawldb. type: `boolean` default value: ‘true’

Property	Description
refreshStartLinks Recrawl start links	Set to true to recrawl items specified in the list of start links. type: `boolean` default value: ‘false’
refreshErrors Recrawl errors	Set to true to recrawl items that failed during the last crawl. type: `boolean` default value: ‘false’
refreshOlderThan Recrawl age	Number of seconds to recrawl items whose last fetched date is longer ago than this value. type: `integer` default value: ‘-1’
refreshIDPrefixes Recrawl ID prefixes	A prefix to recrawl all items whose IDs begin with this value. type: `array` of `string`
refreshIDRegexes Recrawl ID regexes	A regular expression to recrawl all items whose IDs match this pattern. type: `array` of `string`
refreshScript Recrawl script	A JavaScript function ('shouldRefresh()') to customize the items recrawled. type: `string`
forceRefresh Force recrawl	Set to true to recrawl all items even if they have not changed since the last crawl. type: `boolean` default value: ‘false’

Property	Description
delete Delete dead URIs	Set to true to remove documents from the index when they can no longer be accessed as unique documents. type: `boolean` default value: ‘true’
deleteErrorsAfter Fetch failure allowance	Number of fetch failures to tolerate before removing a document from the index. The default of -1 means no fetch failures will be removed. type: `integer` default value: ‘-1’

Crawl History

Property	Description
f.discardLinkURLQueries Discard queries in link URLs	If true, query parameters found in URLs will be removed before being added to the discovery queue. type: `boolean` default value: ‘false’
retainOutlinks Retain links in the crawldb	Set to true for links found during fetching to be stored in the crawldb. type: `boolean` default value: ‘true’
aliasExpiration Alias expiration	The number of crawls after which an alias will expire. The default is 1 crawl. type: `integer` default value: ‘1’
crawlDBType Crawl database type	The type of crawl database to use, in-memory or on-disk. type: `string` default value: ‘in-memory’ enum: \{ in-memory on-disk }

Property	Description
indexCrawlDBToSolr Index crawl database to Solr	EXPERIMENTAL: Set to true to index the crawl-database into a 'crawldb_' collection in Solr. type: `boolean` default value: ‘false’

Field Mapping

Property	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` _ (required)_ : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Property	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
pipeline Pipeline ID	The index pipeline used to process documents.

13.41. Websphere Connector and Datasource Configuration

WebSphere Commerce Suite, a.k.a IBM WebSphere is an IBM product that offers full stack software for companies that do eCommerce business. User interactions with the IBM storefront eCommerce website can be captured as Fusion signals and can be used for product recommendations on the storefront.

WebSphere search is powered by Solr and the product is distributed with Solr jar files and deployment scripts for different development and production configurations. In order to use the WebSphere-deployed Solr with Fusion index and query pipelines, is it necessary to install and configure an additional set of Solr plugins. The plugins, installation and configuration information are publically available from:

- https://github.com/LucidWorks/fusion-solr-plugins

13.42. Windows Share Connector and Datasource Configuration

The Windows Share connector can access content in a Windows Share or Server Message Block (SMB)/Common Internet File System (CIFS) filesystem.

See this tutorial about configuring a Windows Share datasource and enabling security trimming:

13.42.1. Access Control Lists (ACLs)

The connector is able to retrieve and store ACL details when crawling with the 'smb' type. There are several properties available to define how the datasource should read the user and group information found in Active Directory, and when security trimming is enabled, document results will take user authorizations into consideration.

For each document, the `acl` field is populated with data that can be used at search time to filter the results so that only people that have been granted access at the user level or through group membership can see them. Two kinds of tokens are stored: Allow and Deny. The format used is as follows:

Allow:`WINA<SID>`

Deny:`WIND<SID>`

Where `SID` is the security identifier commonly used in Microsoft Windows systems. There are some well known SIDs that can be used in the `acl` field to make documents that are crawled through some other mechanism than by using SMB data source behave, from the `acl pow`, the same way as the crawled SMB content:

SID	Description
S-1-1-0	Everyone.
S-1-5-domain-500	A user account for the system administrator. By default, it is the only user account that is given full control over the system.
S-1-5-domain-512	Domain Admins: a global group whose members are authorized to administer the domain. By default, the Domain Admins group is a member of the Administrators group on all computers that have joined a domain.
S-1-5-domain-513	Domain Users.

Note that some of the listed SIDs contain a `domain` token. This means that the actual SIDs differ from system to system. To find out the SIDs for particular user in particular system you can use the information provided by the Windows command line tool `whoami` by executing command `whoami /all`.

You can populate the `acl` field in your documents with these Windows SIDs to make them searchable in Fusion. For example, if you wanted to make some documents available to "Everyone" you would populate the `acl` field with the `WINAS-1-1-0` token. If you wanted to make all docs from one data source available to everybody you can use the literal definitions in the data source configuration.

13.42.2. Configuration

Property	Description
add_failed_docs Add failed documents	Set to true to add documents even if they partially fail processing. Failed documents will be added with as much metadata as available, but may not include many expected fields. type: `boolean` default value: ‘false’
bounds Crawl bounds	Limits the the crawl to a specific directory sub-tree, hostname or domain. type: `string` default value: ‘tree’ enum: \{ tree host domain none }
collection Collection	Collection documents will be indexed to. type: `string`
commit_on_finish Solr commit on finish	Set to true for a request to be sent to Solr after the last batch has been fetched to commit the documents to the index. type: `boolean` default value: ‘true’
crawl_depth Max crawl depth	Number of levels in a directory or site tree to descend for documents. type: `integer` default value: ‘-1’ exclusiveMinimum: false minimum: -1

Property	Description
crawl_item_timeout Fetch timeout	Time in milliseconds to fetch any individual document. type: `integer` default value: '600000' exclusiveMinimum: true minimum: 0
enable_security_trimming Enable security trimming	Set to true to fetch and index access control information from files. type: `object` object attributes: \{ }
exclude_paths Exclusive regexes	Regular expressions for URI patterns to exclude. This will limit this datasource to only URIs that do not match the regular expression. type: `array` of `string` default value: [ ]
include_extensions Included file extensions	File extensions to be fetched. This will limit this datasource to only these file extensions. However, files with possible matching MIME types but non-matching file extensions will be skipped. The file extensions should be entered with no dots, with whitespace separating items in a list (e.g., 'pdf zip'). type: `array` of `string`
include_paths Inclusive regexes	Regular expressions for URI patterns to include. This will limit this datasource to only URIs that match the regular expression. type: `array` of `string`

Property	Description
index_directories Index directories	Set to true to add directories to the index as documents. If set to false, directories will not be added to the index, but they will still be traversed for documents. type: boolean default value: ‘false’
kerberos_keytab Kerberos keytab	Full path to the Kerberos keytab file. type: string default value: '`’
kerberos_user Kerberos principal	Kerberos principal name, i.e., 'username@YOUR-REALM.COM'. type: string default value: '`’
max_bytes Maximum file size (bytes)	Maximum size, in bytes, of documents to fetch. type: integer default value: ‘10485760’ exclusiveMinimum: false minimum: -1
max_docs Max items	Maximum number of documents to fetch. The default (-1) means no limit. type: integer default value: ‘-1’ exclusiveMinimum: false minimum: -1

Property	Description
max_threads Fetch threads	The maximum number of threads to use for fetching data. Each thread will create a new connection to the repository, which may make overall throughput faster, but will also require more system resources including CPU and memory. type: `integer` default value: ‘1’
maximum_connections Maximum fetch connections	Maximum number of concurrent connections to the filesystem. A large number of documents could cause a large number of simultaneous connections to the repository and lead to errors or degraded performance. In some cases, reducing this number may help performance issues. type: `integer` default value: ‘10000’
password Password	Password for the user. Do not set a password if Kerberos authentication is used. type: `string`
splitter Split CSV/TSV and archives	Set to true to split CSV/TSV files into a document per row, or to split archives such as .zip, tar.gz, etc., into individual files. type: `object` object attributes: \{ }
url Start link *required*	A starting URI for this datasource. The URI must be fully-qualified, and include the protocol, host, port and path, as appropriate. type: `string` minLength: 1 pattern: .:.

Property	Description
username Username	A user in the Windows domain with READ permissions for the Windows Share. Do not set username if Kerberos authentication is used. type: `string`
verify_access Validate access	Set to true to require successful connection to the filesystem before saving this datasource. type: `boolean` default value: ‘true’
windows_domain Windows Domain	Authentication domain for the user. Not required if using Kerberos authentication. type: `string`
with_kerberos Use Kerberos Authentication	Select to use Kerberos for authentication instead of username and password. Kerberos authentication requires information for the advanced properties 'Kerberos principal' and 'Kerberos keytab'. If 'false', Username and Password must be entered. type: `boolean` default value: ‘false’

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: array of object object attributes: \{ operation : \{ display name: Operation type: string default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: boolean default value: ‘false’

Initial Mappings	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: 'false'
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: 'com.lucidworks.connectors.db.impl.MapDbConnectorDb' minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

13.43. Zendesk Connector and Datasource Configuration

The Zendesk connector uses the Zendesk REST API to retrieve tickets and their associated comments and attachments from a Zendesk repository.

It retrieves all tickets with all fields (e.g., customer, assignee, priority, status) as well as access restrictions for users and groups. ACLs can be used for security trimming at query time. The types of items retrieved are:

- tickets and their associated metrics: e.g., time elapsed until first response, time to close.
- ticket comment counts, comment ids
- ticket comment attachment URLs

Items retrieved are returned as individual Solr documents, therefore, for a given Zendesk ticket, there will be multiple documents:

- the Zendesk ticket itself
- one document per comment
- one document per comment attachment

Documents have fields for Zendesk type and reference field to parent documents, e.g., a comment document will have field "ticket_id" pointing back to the Zendesk ticket.

Incremental recrawls allow updates to the Fusion collection to add new tickets and record further changes to existing tickets without having to retrieve the entire Zendesk repository contents.

13.43.1. Authorization

The Zendesk user must have administrator privileges in order to retrieve all tickets and associated information. All communication with the Zendesk API is encrypted with SSL.

13.43.2. Required Configuration Properties

A Zendesk datasource must be configured with the following properties:

- Authentication Key - username or email
- Authentication Value - the password or API token
- Token Auth - a flag to indicate whether the auth key/value should be treated as username/password or email/token
- Base URL - the API url to an instance of the Zendesk API
- Organization ID - set to restrict indexing to only tickets that belong to the Organization

13.43.3. Configuration Properties

Property	Description
auth_key Authentication Key *required*	An email address of a valid Zendesk user which must have an admin role. type: `string` minLength: 1
auth_value Authentication Value *required*	This value can be a user password, or an API token. If an API token is used, the Token Auth must be set to true. type: `string`
base_url Base API URL *required*	A properly formatted v2 Zendesk API URL. Example: https://yourcompany.zendesk.com/api/v2 type: `string`
collection Collection	Collection documents will be indexed to. type: `string`
organization_id Organization ID	The ID of a Zendesk Organization. This optional setting will allow the Connector to skip records that don't have the same Organization ID. type: `string`
use_token_auth Token Auth *required*	Used to determine how the authentication credentials will be formatted. If set to true, the auth_key value will be formatted according to the Zendesk token auth specifications (/). type: `boolean` default value: 'true'

Field Mapping

Initial Mappings	Description
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’

Initial Mappings	Description
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + \`source\` : \{ + display name: Source Field + type: \`string\` + description : The name of the field to be mapped. + } + \`target\` : \{ + display name: Target Field + type: \`string\` + description : The name of the field to be mapped to. + } + }

ConnectorDb Configuration

Property	Description
aliases Process Aliases?	Keep track of original URI-s that resolved to the current URI. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
inlinks Process Inlinks?	Keep track of incoming links. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’

Property	Description
inv_aliases Process Inverted Aliases?	Keep track of target URI-s that the current URI resolves to. This negatively impacts performance and size of DB. type: `boolean` default value: ‘false’
type Implementation Class Name	Fully qualified class name of ConnectorDb implementation. type: `string` default value: ‘com.lucidworks.connectors.db.impl.MapDbConnectorDb’ minLength: 1

General Configuration

Property	Description
description Description	Optional description for datasource instance.
id Datasource ID	Unique name for datasource instance.
pipeline Pipeline ID	The index pipeline used to process documents.

Chapter 14. Pipeline Stages Reference

Index Pipeline stages are used to create and modify PipelineDocument objects to control how incoming data is indexed in Fusion’s Solr core.

Query Pipeline stages are used to modify Request objects and Response objects to control how query results are returned to the end user.

14.1. Pipeline Stage Properties

Fusion associates a pipeline stage definition with a unique ID and stores the definition so that stages can be reused across pipelines and applications. In addition to an ID, all stages have the following properties:

- **type** (required): an enumeration, one of the defined Fusion pipeline stage types, e.g., "index-logging". If the Fusion UI is used to define the stage, this property is filled in automatically.
- **label** (optional) : a string field with a maximum length of 255 characters. The label is displayed on the Fusion UI.
- **skip** (optional): a boolean value, If true, pipeline processing bypasses this stage altogether. The default is **false**.
- **condition** (optional): a JavaScript expression that evaluates to true (1) or false (0). If this condition evaluates to false, this stage is skipped. The default is **true**.

14.2. Conditional Processing

Each Fusion pipeline stage contains a "Condition" field at the beginning of the stage configuration found in the Fusion UI. A conditional script can be used to dynamically turn a stage on/off. For example, in the index pipeline you can create a field mapping stage and create a condition to only execute the stage if a document contains the field 'sample_field', and to skip the stage otherwise.

```
doc.hasField("sample_field");
```

For the example above, if the pipeline document has the field 'sample_field', then the conditional script returns 'true' and the pipeline stage is executed. If the pipeline document does not have the 'sample_field', the condition returns 'false' and the pipeline stage is skipped.

The JavaScript expression specified in the condition property of a pipeline stage has access to the pipeline objects. For example, query processing can be conditional on information in the query Request object:

```
request.hasParam("fusion-user-name") && request.getFirstParam("fusion-user-name").equals("SuperUser");
```

The above condition first checks that the property "fusion-user-name" is specified, then checks for a particular value.

14.3. Index Pipeline Stages

Index Pipeline stages are used to create and modify PipelineDocument objects. Use the Index Workbench to configure stages in a pipeline and preview the results.

See these reference topics for details about each index pipeline stage:

- Apache Tika Parser
- CSV Parser
- Date Parsing
- Detect Language
- Exclusion Filter
- Field Mapper
- Find Replace
- Fusion Pipeline
- Gazetteer Lookup Extractor
- HTML Transform
- Indexing RPC
- JDBC
- JSON Parser
- Javascript
- Language Detection
- Logging
- Logging Message
- Machine Learning
- Multi-value Resolver
- OpenNLP NER Extractor
- Part of Speech
- Regular Expression Extractor
- Regular Expression Filter
- Regular Expression Replacement
- Sentence Detection
- Set Property
- Short Field Filter
- Signal Formatter
- Solr Indexer
- Solr Partial Update Indexer
- XML Transform

14.3.1. Apache Tika Parser Index Stage

The Apache Tika Parser index stage type includes rules for parsing documents with Apache Tika. Fusion uses Tika v1.13. (Note that components of the Solr distribution included with Fusion contian their own Tika jar files; these are not used by Fusion.)

To parse a CSV document, you should use a CSV Parsing Index Stage instead of an Apache Tika Parser stage.

Configuration

When using Fusion's REST-API, the ID of this stage is: `tika-parser`.

Configuration Properties

Property	Description, Type
addFailedDocs Add failed documents	type: `boolean` default value: 'false'
addOriginalContent Add original document content (raw bytes)	type: `boolean` default value: 'true'
contentEncoding Content transport encoding of the content (per RFC1341)	type: `string` default value: 'binary' enum: \{ binary base64 }
contentField Field name where content is expected	type: `string` default value: '*raw_content*'

Property	Description, Type
csvConfig Configuration for the CSV files parser	type: object object attributes: \{ comment : \{ display name: Comment character type: string default value: ‘#’ maxLength : 1 } condition : \{ display name: Condition type: string description : Define a conditional script that must result in true or false. This can be used to determine if the stage should process or not. } delimiter : \{ display name: Delimiter type: string default value: ‘,’ minLength : 1 } emptyValue : \{ display name: Empty value type: string default value: ‘' } `headers : \{ display name: Set Headers type: array of string } headersHandling : \{ display name: Headers Handling type: string default value: '`parse’ enum: \{ dynamic ignore parse } } + `ignoreLeadingWhitespaces` : \{ + display name: Ignore leading whitespaces + type: `boolean` + default value: '`false`' + } + `ignoreTrailingWhitespaces` : \{ + display name: Ignore trailing whitespace + type: `boolean` + default value: '`false`' + } + `inheritFields` : \{ + display name: Inherit fields + type: `boolean` + default value: '`false`' +

Property	Description, Type
csvParsing Do CSV parsing	type: `boolean` default value: ‘true’
excludeContentTypes Content types to exclude	List of content types to exclude from parsing type: `array` of `string`
extractHtmlLinks Extract XHTML links	Collect links explicitly declared in document structure (e.g. using HTML tags, bookmarks, etc) type: `boolean` default value: ‘true’
extractOtherLinks Extract other links	Use regex-based heuristic extractor to collect likely links from plain text content in all fields. type: `boolean` default value: ‘false’
flattenCompound Flatten compound documents	type: `boolean` default value: ‘false’
includeContentTypes Content types to include	List of content types to parse type: `array` of `string`
includeImages Include images	type: `boolean` default value: ‘false’
keepOriginalStructure Return original XML and HTML instead of Tika XML output	type: `boolean` default value: ‘false’
returnXml Return parsed content as XML or HTML	type: `boolean` default value: ‘false’

Property	Description, Type
zipBombCompressionRatio Maximum input-to-output byte ratio	Maximum number of output bytes Fusion will generate per input byte. If you are indexing highly compressed files, you may increase this value to avoid triggering 'Zip Bomb' detection type: `integer` default value: ‘100’

14.3.2. Call Pipeline Index Stage

The Call Pipeline index stage (called the Fusion Pipeline stage in versions earlier than 3.0) provides a means of composing index pipelines.

Configuration

When using Fusion's REST-API, the ID of this stage is: `apollo`.

Configuration Properties

Property	Description, Type
collection Collection *required*	type: `string`
pipeline Pipeline ID *required*	type: `string`

14.3.3. CSV Parsing Index Stage

The CSV Parsing Index stage (previously called the CSV Parser stage) parses CSV content from a document field into new documents. It will produce as many documents are there are rows in the CSV input, excluding comment and header rows.

The following CSV snippet consists of a row of column headers and 5 rows of data, (6 lines total):

```
Name,Description
Arnold_Schwarzenegger,"Austrian-American bodybuilder, actor, politician"
Anthony_Hopkins,"Actor"
Albert_Brooks,"Actor, voice actor, writer, comedian and director"
Britney_Spears,"American musician, singer, songwriter, actress, author"
Brigitte_Bardot,"French actress and animal welfare activist"
```

Running this input through a CSV parsing stage which has been configured to use the column headers as field names produces 5 documents:

(document)	**field "Name"**	**field "Description"**
(1)	Arnold_Schwarzenegger	Austrian-American bodybuilder, actor, politician
(2)	Anthony_Hopkins	Actor
(3)	Albert_Brooks	Actor, voice actor, writer, comedian and director
(4)	Britney_Spears	American musician, singer, songwriter, actress, author
(5)	Brigitte_Bardot	French actress and animal welfare activist

There are two configuration properties for proper handling of the CSV header columns:

- `headers` configuration property can be used to define the mapping from CSV columns to document fields using a list of field names which are applied to the columns in the order in which they are specified.
- `headersHandling` configuration property specifies how to handle the first row of CSV data. It takes one of three possible values:
  - `parse` - Assumes the input document contains headers in the first row. The column headers are used as the field name. This is the default option.
  - `dynamic` - Assumes the input does not contain headers. Treats the first line of the CSV file as data. If field names have been specified they will be used, else dynamic field names will be created for the parsed values.
  - `ignore` - Assumes the input document contains headers and ignores them. If field names have been specified they will be used, else dynamic field names will be created for the parsed values.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `csv-parser`.

Configuration Properties

Property	Description, Type
comment Comment character	type: `string` default value: ‘#’ maxLength: 1
delimiter Delimiter	type: `string` default value: ‘,’ minLength: 1
emptyValue Empty value	type: `string` default value: '`’
headers Set Headers	type: `array` of `string`
headersHandling Headers Handling	type: `string` default value: ‘parse’ enum: \{ dynamic ignore parse }
ignoreLeadingWhitespaces Ignore leading whitespaces	type: `boolean` default value: ‘false’
ignoreTrailingWhitespaces Ignore trailing whitespace	type: `boolean` default value: ‘false’
inheritFields Inherit fields	Should generated documents include the original document’s fields type: `boolean` default value: ‘false’

Property	**Description, Type**
lineSeparator Line Separator	type: `string` default value: ‘\n’ minLength: 1
nullValue Null value	type: `string` default value: '`’
quote Quote character	type: `string` default value: ‘"’ maxLength: 1
quoteEscape Quote escape	type: `string` default value: ‘"’ maxLength: 1
skipEmptyLines Skip empty lines	type: `boolean` default value: ‘true’
sourceEncoding Source field encoding	The content transport encoding of the data stored in the source field (per RFC1341) type: `string` default value: ‘binary’ enum: \{ binary base64 }
sourceField Source field	The document field where the CSV content should be read from type: `string` default value: ‘*raw_content*’

14.3.4. Date Parsing Index Stage

The Date Parsing Stage (previously called the Date Parser stage) is an index pipeline stage that performs parsing and normalization of date/time data in document fields which uses the Fusion DateUtils library. The resulting date/time information is available both as an timestamp in UTC time zone as well as a local date/time in the original local time zone.

The time zone name, offset and the epoch time are stored in separate fields, too. Additionally the formatted dates can be split into their components, and each component added to separate document fields.

Note that this stage works only with data that consists solely of the date/time information, i.e. it will not work correctly if dates are a part of a larger piece of text.

Timestamp splitting options

Splitting options help in processing timestamp information without resorting to scripting - e.g. in order to index day of week information it's more convenient and faster to split the timestamp in this stage, and then just discard other components that are not needed (using a field mapping stage), rather than using a JavaScript stage to parse and split the timestamp manually.

Please note that time zone name and time zone offset, as well as epoch time, are always added as separate fields regardless of the splitting options. E.g. for a field named `test` these values will be added as fields `tz.test`, `tz_offset.test`, and `epoch.test`.

The option `splitLocal` splits the timestamp in its original timezone, while the option `splitUTC` first converts the timestamp to UTC and then splits it. The resulting date and time components are stored in fields that follow patterns <part>.local.<fieldName> and <part>.utc.<fieldName> respectively.

The following parts are extracted and added to the document:

- **year** - year component
- **month** - month in year, from 1 to 12
- **day** - day in month, from 1 to 31
- **yday** - day in year, from 1 to 356
- **weekday** - day of week, 1 being Monday and 7 being Sunday
- **week** - week in year, from 1 to 52. Note: in the standard ISO8601 week algorithm, the first week of the year is that in which at least 4 days are in the year. As a result of this definition, day 1 of the first week may be in the previous year, which will be indicated by weekyear. The opposite is also true - last day of the last week may be in the next year, and weekyear will show the next year.
- **weekyear** - year corresponding to the week value. This can be either the current year or previous one, or the next one.
- **hour** - hour in day, from 0 to 23
- **min** - minute in hour, from 0 to 59
- **sec** - second in minute, from 0 to 59
- **ms** - millisecond in second, from 0 to 999

Example: given this normalized timestamp in the original timezone `2015-01-01 00:00:00.000 Europe/Warsaw` in a field `test`, the corresponding normalized UTC timestamp will be `2014-12-31T23:00:00.00Z`.

Example: `splitLocal` parsing

The following table shows the additional fields added to a document as the result of applying `splitLocal` parsing to the contents a field named `test` which contains the value `2015-01-01 00:00:00.000 Europe/Warsaw`:

Field name	value
tz.test	Europe/Warsaw
tz_offset.test	+01:00
epoch.test	1420066800000

Example: `splitUTC` parsing

The following table shows the additional fields added to a document as the result of applying `splitUTC` parsing to the contents a field named `test` which contains the value `2015-01-01 00:00:00.000 Europe/Warsaw`:

Field name	value
tz.test	Europe/Warsaw
tz_offset.test	+01:00
epoch.test	1420066800000
year.utc.test	2014
year.local.test	2015
month.utc.test	12
month.local.test	1
day.utc.test	31
day.local.test	1
yday.utc.test	365
yday.local.test	1
weekday.utc.test	3
weekday.local.test	4
week.utc.test	1
week.local.test	1
weekyear.utc.test	2015
weekyear.local.test	2015
hour.utc.test	23
hour.local.test	0
min.utc.test	0
min.local.test	0

Field name	value
sec.utc.test	0
sec.local.test	0
ms.utc.test	0
ms.local.test	0

Note: The following:

- weekday is different - UTC day of week is Wednesday, and local day of week is already Thursday.
- yday in UTC points to the last day of the year, while it’s the first day of the year in local time zone, similarly with day.
- week and weekyear are the same in both cases - because according to the ISO 8601 definition all days of this week belong to year 2015 so it doesn’t matter whether it’s Wednesday or Thursday.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `date-parsing`.

Configuration Properties

Property	Description, Type
dateFormats Date Formats	Custom date formats, or empty for default formats type: `array` of `string`
defaultLocale Default locale	Locale to assume if different from Locale.ENGLISH. This uses IETF BCP 47 codes. type: `string`
defaultTimezone Default timezone	Timezone to assume if one is not present in the incoming date type: `string`
ignoreInvalid Ignore invalid	When false invalid date strings will cause the whole document to be rejected. When true invalid values are silently discarded type: `boolean` default value: ‘false’

Property	Description, Type
requireTimezone Require timezone	Accept only formats that explicitly specify the timezone type: `boolean` default value: ‘false’
sourceFields Source Fields	type: `array` of `string`
splitLocal Split local date into parts	Split local date (in the local timezone) into parts and store in .local. fields type: `boolean` default value: ‘false’
splitUTC Split UTC date into parts	Split UTC date (in the UTC timezone) into parts and store in .utc. fields type: `boolean` default value: ‘false’

14.3.5. Exclusion Filter Index Stage

The Exclusion Filter index stage is used to remove fields or documents that match items in a pre-defined exclusion list.

There are two ways to supply an exclusion list:

- Upload a file containing a newline-separated list, using the Blob Store. When configuring the index stage, reference the list by its blob name in the `location` property (**Exclusion List URI** in the Fusion UI).
- When configuring the index stage, enter an array of values for exclusion in the `excludeValues` property (**Exclusion List** in the Fusion UI).

The Exclusion Filter stage can be configured using one or both of these methods; Fusion combines them into one list. If `regexPattern` is configured, the pattern is applied to the field before the result is compared to the combined list.

By default, any matching *field* is excluded from indexing. To exclude the whole document, set `skipDocument` to "true" (**Skip Document** in the Fusion UI).

Uploading an exclusion list

Before you can configure the `location` property, you must upload one or more exclusion lists to Fusion using the Blob Store API.

Fusion comes with an example exclusion list at `fusion/data/nlp/excludes/excludes.txt`. Here is an example of how to upload this file using `curl`, where `admin:pass` are the credentials for an admin-level user:

```
curl -u admin:pass -X PUT --data-binary @data/nlp/excludes/excludes.txt -H 'Content-type: text/plain'
http://localhost:8764/api/apollo/blobs/excludes.txt
```

Example

Use an exclusion list for entities found in the author field:

```
{
    "type" : "exclusion-filter",
    "id" : "iw",
    "filters" : [ {
      "sourceField" : "author_s",
      "location" : "excludes.txt",
      "caseSensitive" : false
    } ],
    "skip" : false
  } ]
}
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `exclusion-filter`.

Configuration Properties

Property	Description, Type
filters Filtering Rules	type: `array` of `object` object attributes: \{ `caseSensitive` *(required)* : \{ display name: Case Sensitive type: `boolean` default value: ‘false’ } `excludeValues` : \{ display name: Values to Exclude type: `array` of `string` } `location` : \{ display name: Exclusion List URI (Blob name) type: `string` reference : /blobs } `regexPattern` : \{ display name: Regex expression type: `object` `group` : \{"default"⇒0, "title"⇒"Regex Capture Group", "type"⇒"integer"} `pattern` *(required)* : \{"format"⇒"regex", "title"⇒"Regex expression", "type"⇒"string"} } `skipDocument` : \{ display name: Skip document type: `boolean` default value: ‘false’ } `sourceField` *(required)* : \{ display name: Source Field type: `string` } }

14.3.6. Field Mapping Index Stage

The Field Mapping stage was renamed for Fusion 3.0; it was called the Field Mapper stage in previous versions.

A Field Mapping stage is used to do customized mapping of the fields in an Index Pipeline document to fields in a the Solr schema.

Field Mapping Stage Properties

A Field Mapping stage specification consists of three things:

- a unique ID
- a set of mapping rules that specify operations applied to named fields as a triple: `{ source, target, operation }`.
- a set of rules called "unmapped" rules which specify operations applied to fields whose name doesn’t match any of the mapping rules, also a triple `{ source, target, operation }`.

Mapping Rules and Unmapped Rules

Each rule has the following properties:

Property	Description
source	The name of the source field. This will be the name of the field in the Pipeline document that should be mapped to another field. Java regular expressions can be used in the source field by surrounding the regular expression with forward slashes ('/'). For example, `/(.)text(.\)/` is a valid expression that will find field names in the incoming document that contain the string 'text' between any number of preceding or succeeding characters. If a regular expression is not used, the value supplied for the source will be treated as a literal field name and will be matched ignoring the case (for example, "text" will match "tExt" or "Text", etc.).
target	The name of the target field. If the value for the `source` was a regular expression, then this can also be a regular expression. It can also contain substitutions using references to capture groups (using Java’s Matcher.replaceAll). Otherwise, the source field name will be simply substituted by the value of target according to the operation rules described below.

Property	Description
operation	What to do with the field during mapping. Several options are available: • **copy**: Content contained in fields matching `source` will be copied to `target`. • **move**: Content contained in fields matching `source` will be moved to `target` (it may also help to think of this as the field name being replaced by the value of `target`). • **delete**: Content contained in fields matching `source` will be dropped from the document and not indexed. In this case, the `target` can be null or not defined at all. • **add**: The literal value of `target` will be added to the `source` if `source` is a regular expression. If `source` is not a regular expression, `target` will be added as a new field. • **set**: The literal value of `target` will be set as the new value of `source` if `source` is a regular expression. If `source` is not a regular expression, `target` will be set as a new field. • **keep**: Content contained in fields matching `source` will be retained and unchanged, and the fields will be added to a list of known fields and they will not be affected by however the `renameUnknown` rule has been set.

Note that the mapping rules are applied in the order in which they are defined, which may have an impact on the final effects of the mapping process.

Field Mapping Behavior

The field mapping rules are applied in a specific order.

1. A copy of the Pipeline document is prepared. All further operations are applied to this copy.
2. The rules are traversed only once, in the order of their declaration in the `mapping` property. This means it is possible to do multiple operations on a field. However, note that if fields are moved (renamed), further operations should reference the new field name.
3. Before each rule is evaluated, the current list of field names is prepared and sorted in alphabetic ascending order.
4. The current rule is applied to field values for each matching name from the list of names prepared in step 3. New field names resulting from the current rule do not effect the snapshot list of field names; in order for a rule to be applied to a new field name, it will be included in a later round of the evaluation cycle.
5. The process is repeated for each rule, and a list of matching source fields is noted.
6. If the document contains any fields that were not affected by any mapping rule, the `renameUnknown` option is applied if it has been set to true.

7. Finally, the resulting transformed document is returned to the next stage of the index pipeline.

Examples

Map several fields:

```
{
    "id": "mapping-text",
    "type": "field-mapping",
    "mappings": [
        {
            "operation": "move",
            "source": "plaintextcontent",
            "target": "body"
        },
        {
            "operation": "add",
            "source": "content-length",
            "target": "fileSize"
        },
        {
            "operation": "move",
            "source": "/file(.*)/",
            "target": "lastModified"
        },
        {
            "operation": "delete",
            "source": "last-printed"
        },
        {
            "operation": "copy",
            "source": "mimetype",
            "target": "content_type"
        }
    ],
    "unmapped": {
        "source": "/(.*)/",
        "target": "$1_ss",
        "operation": "move"
    },
    "skip" : false
}
```

Set the `urlX` *field based on the value of the* `employee_id` *field:*

```
{
    "id": "set-field",
    "type": "field-mapping",
    "mappings": [
        {
            "operation": "set",
            "source": "urlX",
            "target": "https://mydomain.com/<employee_id>"
        }
    ],
    "skip" : false
}
```

Configuration

When using Fusion's REST-API, the ID of this stage is: `field-mapping`.

Configuration Properties

Property	Description, Type
mappings Field mappings	List of mapping rules type: `array` of `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: 'copy' description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` _ (required)_ : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

Property	Description, Type
reservedFieldsMappingAllowed Allow system fields (special fields reserved for internal use) mapping?	type: `boolean` default value: ‘false’
unmapped Unmapped fields	If fields do not match any of the field mapping rules, these rules will apply. type: `object` object attributes: \{ `operation` : \{ display name: Operation type: `string` default value: ‘copy’ description : The type of mapping to perform: move, copy, delete, add, set, or keep enum: \{ copy move delete set add keep } } + `source` : \{ + display name: Source Field + type: `string` + description : The name of the field to be mapped. + } + `target` : \{ + display name: Target Field + type: `string` + description : The name of the field to be mapped to. + } + }

14.3.7. Find and Replace Index Stage

The Find and Replace stage is used to standardize or remove a set of specific passages, phrases, or words from a document field using exact string matching. This stage is configured with one or more rules. Each rule specifies one or more texts to match against and a *single* text string that is the replacement value. When the replacement string is the empty string, this will delete the matched text from the document field.

There are two different ways of setting up a Find and Replace stage:

- Using Fusion's blob store to store the list of texts to be standardized. This is known as "Find List Replace" For each standardization, there is a file which contains the set of texts to be standardized, one per line, with file suffix `.lst`. The `.lst` file is uploaded into Fusion's ../../REST_API_Reference/Blob-Store-API.html[Blob Store]. The required properties for each rule are: the document field to scan; the name of the list file uploaded to the blob store, inclusive of suffix `.lst`; and the replace text value.
- Specifying a set of find/replace pairs directly using the Fusion UI or REST API. This is known as "Find Replace". The requires properties for each rule are: the document field to scan; a list of find/replace pairs.

Configuration

When using Fusion's REST-API, the ID of this stage is: `find-replace`.

Configuration Properties

Property	Description, Type
findListReplaceRules Find List Replace rules	type: `array` of `object` object attributes: \{ `caseSensitive` : \{ display name: Case sensitive type: `boolean` default value: 'true' } `listLocation` *(required)* : \{ display name: Blob name of the list type: `string` reference : /blobs } `replacementValue` : \{ display name: Replacement Value type: `string` } `sourceField` *(required)* : \{ display name: Source Field type: `string` } }

Property	Description, Type
findReplaceRules Find replace rules	type: `array` of `object` object attributes: \{ `caseSensitive` : \{ display name: Case sensitive type: `boolean` default value: ‘true’ } `keyValues` *(required)* : \{ display name: find and replace strings list type: `array` of `object` } `sourceField` *(required)* : \{ display name: Source Field type: `string` } }

14.3.8. Gazetteer Lookup Extraction Index Stage

The Gazetteer Lookup Extraction index stage (called the Gazetteer Lookup Extractor stage in versions earlier than 3.0) uses predefined lists of words and phrases to process specified text fields in a document. A gazetteer is a set of lookup lists over names of people, places, or things. These lookup lists are used to find occurrences of these names in text. The matched items are saved into separate fields on the document for downstream processing.

Gazetteers and OpenNLP Tools

The following video shows how to configure a Gazetteer Lookup Extraction stage in combination with OpenNLP:

Uploading Lookup Lists to Fusion Blob Store

Fusion includes a number of lookup lists in the directory fusion/data/nlp/gazetteer. To use the supplied lists or a list of your own data, each must list be uploaded to Fusion using the Blob Store API in order to make the list contents available to the Gazetteer Lookup Extraction stage.

For example, to identify color names, you would first compile a list of color terms, one entry per line in a text file with suffix `.lst` and then upload that file using the Fusion REST API endpoint `` api/apollo/blobs/<listfilename>, as per the following example which uses the \`curl `` command-line utility, where 'admin' is the name of a user with admin privileges, and 'pass' is that user’s password:

```
curl -u admin:pass -X PUT --data-binary @data/nlp/gazetteer/colours.lst -H 'Content-type: text/plain'
http://localhost:8764/api/apollo/blobs/colours.lst
```

Name Lookup Example

Define a lookup-extractor to identify mentions of certain celebrities in text field `description_t`*:*

```
{
  "type" : "lookup-extractor",
  "id" : "peopleLookup",
  "rules" : [ {
    "source" : [ "description_t" ],
    "target" : "celebrities_ss",
    "entityTypes" : [ {
      "name" : "person_female",
      "definitions" : [ "person_female.lst" ]
    } ],
    "additionalEntities" : [ {
      "name" : "players",
      "definitions" : [ "sharapova", "murray" ]
    }, {
      "name" : "actors",
      "definitions" : [ "pitt", "jolie" ]
    } ],
    "caseSensitive" : false
  } ],
  "skip" : false
}
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `lookup-extractor`.

Configuration Properties

Property	Description, Type
rules Extraction Rules	type: `array` of `object` object attributes: \{ `additionalEntities` : \{ display name: Additional Entities type: `array` of `object` } `caseSensitive` : \{ display name: Case Sensitive type: `boolean` default value: ‘false’ } `entityTypes` *(required)* : \{ display name: Entity Types type: `array` of `object` } `source` *(required)* : \{ display name: Source Fields type: `array` of `string` minItems : 1 } `target` *(required)* : \{ display name: Target Field type: `string` } }

14.3.9. HTML Transformation Index Stage

The HTML Transformation stage (called the HTML Transform stage in versions earlier than 3.0) is used to process HTML by means of a set of explicit mapping rules. This stage is usually used in tandem with an Apache Tika Parser stage; it provides custom processing of HTML content, instead of the Tika defaults. It uses the JSoup library which has a rich syntax for selecting HTML and CSS tags and elements. The JSoup selector patterns are used to map HTML elements to PipelineDocument fields. For example, you could process navigational div elements one way, and contentful div elements another way.

The HTML Transformation stage can be used to create multiple child records from HTML document fragments and relate them back to the parent record via a parent ID field. The Additional Metadata property provides the ability to add additional fields.

Required Pipeline Stages and Configuration

The pipeline must have a Tika Parser stage before the HTML Transformation stage. The Tika Parser **must** be configured as follows:

- UI checkbox "Add original document content" / REST API property "addOriginalContent" set to false
- UI checkbox "Return parsed content as XML or HTML" / REST API property "keepOriginalStructure" set to true
- UI checkbox "Return original XML and HTML instead of Tika XML output" / REST API property "returnXml" set to true

For some versions of Fusion you may need to add a Field Mapping stage after the HTML Transformation stage to remove the following fields from the document:

- *raw-content*
- Content-Type
- Content-Length
- parsing
- parsing_time

HTML Stage Configuration Example

Definition of an HTML Transformation stage to extract image links and text:

```
{
  "type": "html-transform",
  "recordSelector": "#main-content",
  "parentIdField": "page_s",
  "bodyField": "body",
  "mappings": [
   { "selectRule": "div",
     "attribute": "",
     "field": "main-content_txt",
     "multivalue": true
   },
   { "selectRule": "a",
     "attribute": "text",
     "field": "links_txt",
     "multivalue": true
   }  ],
  "keepParent": false,
  "skip": false,
  "label": "html-main-content"
}
```

Configuration

When using Fusion's REST-API, the ID of this stage is: `html-transform`.

Configuration Properties

Property	Description, Type
bodyField Body Field Name	type: `string` default value: 'body'
keepParent Should keep parent document?	type: `boolean` default value: 'false'

Property	Description, Type
mappings HTML Mappings	type: `array` of `object` object attributes: \{ `attribute` : \{ display name: Attribute type: `string` } `field` *(required)* : \{ display name: Field type: `string` } `multivalue` : \{ display name: Multi Value type: `boolean` default value: ‘false’ } `selectRule` *(required)* : \{ display name: Select Rule type: `string` } }
metadata Additional Metadata	type: `array` of `object` object attributes: \{ `field` *(required)* : \{ display name: Field type: `string` } `value` *(required)* : \{ display name: Value type: `string` } }
parentIdField Parent ID Field Name	type: `string`
recordSelector Record Selector	type: `string`

14.3.10. REST Query Index Stage

The REST Query index stage (called the Query REST (RPC) Client stage in versions earlier than 3.0) performs Remote Procedure Calls to external services. It merges the response from that service with a document being processed by the pipeline. Calls to the external system are made for each document.

This stage can call external systems in any of the following ways:

- any HTTP or HTTPS request: `<http-protocol>://<path>`
- a Solr request: `solr://<collection>/…`
- a Fusion service request: `service://<serviceName>/<path>`

The call syntax allows the use of document field variables. For example, variable `${docField}` will be replaced with the value of the field named `docField` for the document being processed.

Several types of responses are also supported:

- POJOs
- JSON
- XML

Call Parameters

In the Fusion UI, select **include** to display the fields for the call parameters (`params` in the REST API). Parameters are specified as key-value pairs:

Endpoint URI / `uri`	A fully-qualified service URI. This can be an HTTP call, a Solr request, or a Fusion service call • any HTTP or HTTPS request: `<http-protocol>://<path>` • a Solr request: `solr://<collection>/…` • a Fusion service request: `service://<serviceName>/<path>`
Call Method / `method`	The method to use for the RPC call. Supported methods are `GET`, `POST`, `PUT`, and `DELETE`.
Query parameters / `queryParams`	Query parameters to be passed with the request.
Request protocol headers / `headers`	Request protocol headers, such as `"Content-Type" : "application/json"`.
Request entity (as string) / `entity`	The request body. This parameter can only be configured via the REST API; see the example below. In the Fusion UI, a known issue prevents configuring the request entity.

The `params` can also take variable substitution expressions. Variables are expressed as `${fieldName}`, where `fieldName` is a name of the current document's field (or `id` for the document's id). Only the first value of a multi-valued field is used for substitution, and this value is treated as a string.

For example, a `queryParams` could be constructed as follows:

```
"params" : {
    "uri" : "solr://collection1/select",
    "method" : "GET",
    "queryParams" : {
      "q" : "{!field f=a_txt v=$prodName}",
      "prodName" : "${prodName}"
}
```

In this example, the variable `${prodName}` will be replaced with the string value of the 'prodName' field in the current document being processed. If the current document contains "iPhone 6" as the value of 'prodName', the resulting query in this example will be a fielded search `q=a_txt:iPhone 6`.

Mapping Rules

The **mappingRules** property takes the following, specified as key-value pairs:

- path - an XPath expression. It’s assumed that this expression always returns a node list, and each returned node is processed separately and its converted value is added to a multi-valued field.
- target - the name of the target field in the current document where the value(s) will be stored.
- append - if true, values extracted from the RPC response will be appended to the target multi-valued field. When this is false, the default, existing values in the target field will be discarded and replaced with the values from the RPC response.
- xml - if true, the extracted DOM nodes will be separately serialized to XML and the resulting XML-formatted text will be added to the target field. When this is false, the default, the extracted DOM nodes will be flattened and converted to a list of fields. Field names in this case will correspond to XML element names, dot-separated, and element attributes will be represented as fields with @attributeName suffix.

Examples

Create a REST Query stage to merge results from another Solr system:

```
{
  "type" : "indexing-rpc",
  "id" : "BasicSolrCall",
  "mappingRules" : [ {
    "path" : "//result/doc[1]/str[@name='foo_s']/text()",
    "target" : "foo_s",
    "append" : true,
    "xml" : false
  }, {
    "path" : "//result/doc/arr[@name='a_txt']/*",
    "target" : "doc_txt",
    "append" : true,
    "xml" : false
  } ],
  "debug" : true,
  "params" : {
    "uri" : "solr://collection1/select",
    "method" : "GET",
    "queryParams" : {
      "q" : "a_txt:${doc_value}"
    },
    "headers" : { }
  },
  "skip" : false,
  "label" : "indexing-rpc"
}
```

Upload a stopword list:

```
{
 "type" : "indexing-rpc",
 "id" : "demo",
 "debug" : true,
 "params" : {
   "uri" : "http://localhost:8765/api/v1/stopwords/movies",
   "method" : "PUT",
   "headers" : { "Content-Type" : "application/json" },
   "entity" : "New stopword list"
 },
 "skip" : false,
 "label" : "Indexing RPC Demo",
 "type" : "indexing-rpc"
}
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `indexing-rpc`.

Configuration Properties

Property	Description, Type
debug Add debugging information	type: `boolean`
mappingRules Mapping of returned values (as XPath expressions) to document fields	type: `array` of `object` object attributes: \{ `append` : \{ display name: Append to existing values in target field type: `boolean` default value: ‘false’ } `path` *(required)* : \{ display name: XPath expression type: `string` } `target` *(required)* : \{ display name: Target field type: `string` } `xml` : \{ display name: Add as an XML fragment type: `boolean` default value: ‘false’ } }

Property	Description, Type
params Call parameters	type: object object attributes: \{ entity : \{ display name: Request entity (as string) type: object } headers : \{ display name: Request protocol headers type: object } method : \{ display name: Call method type: string description : One of GET, POST, PUT, or DELETE enum: \{ get put post delete } } + \`queryParams\` : \{ + display name: Query parameters + type: \`object\` + } + \`uri\` : \{ + display name: Endpoint URI + type: \`string\` + } + }

14.3.11. JavaScript Index Stage

For a complete description of the JavaScript Index stage and additional examples, see: Custom JavaScript Stages For Index Pipelines.

Examples

Drop a document by ID

```
function(doc) {
  var id = doc.getId();
  if (id !== null) {
    var pattern = "https://www.mydomain.com/links/contact/?";

    // 0 means the pattern was found so drop the doc
    return (id.indexOf(pattern) == 0) ? null : doc;
  }

  return doc;
}
```

Format Date to Solr Date

```
// For example:
// From: 26/Mar/2015:14:38:48 -0700
// To:   2015-03-26T14:38:48Z  (Solr format)
function(doc) {
  if (doc.getId() !== null) {
    var inboundPattern = "dd/MMM/yyyy':'HH:mm:ss Z";  // modify this to match the format of the inbound date
    var solrDatePattern = "yyyy-MM-dd'T'HH:mm:ss'Z'"; // leave this alone
    var dateFieldName = "apachelogtime"; // change this to your date field name

    var solrFormatter = new java.text.SimpleDateFormat(solrDatePattern);
    var apacheParser = new java.text.SimpleDateFormat(inboundPattern);

    var dateString = doc.getFirstFieldValue(dateFieldName);
    logger.info("**** dateString: " + dateString);
    var inboundDate = apacheParser.parse(dateString);
    logger.info("**** inboundDate: " + inboundDate.toString());
    var solrDate =  solrFormatter.format(inboundDate);
    logger.info("**** solrDate: " + solrDate.toString());

    doc.setField(dateFieldName, solrDate.toString());
  }

  return doc;
}
```

Replace whitespace and newlines

```
function(doc) {
  if (doc.getId() !== null) {
    var fields = ["col1", "col2", "col3"];

    for (i = 0; i < fields.length; i++ ) {
      var field = fields[i];
      var value = doc.getFirstFieldValue(field);
      logger.info("BEFORE: Field " + field + ": *" + value + "*");

      if (value != null) {
        value = value.replace(/^\s+/, "");  // remove leading whitespace
        logger.info("AFTER: Field " + field + ": *" + value + "*");
        value = value.replace(/\s+$/, "");  // remove trailing whitespace
        logger.info("AFTER: Field " + field + ": *" + value + "*");
        value = value.replace(/\s+/g, " ");  // multiple whitespace to one space
        logger.info("AFTER: Field " + field + ": *" + value + "*");

        doc.setField(field, value);
      }
    }
  }

  return doc;
}
```

Split the values in a field

```
//Split On a delimiter. In this case, a newline
function(doc){
  if (doc.getId() !== null) {
    var fromField = "company2_ss";
    var toField = "company2_ss";
    var delimiter = "\n";

    var oldList = doc.getFieldValues(fromField);

    var values = [];

    // parse the entries one at a time
    doc.removeFields(toField); // clear out the target field
    for (i = 0; i < oldList.size(); i++) {
      values[i] = oldList.get(i);

      // get the list of strings split by the delimiter
      newList = values[i].split(delimiter);
      for(j = 0; j < newList.length; j++ ){
        doc.addField(toField, newList[j]);
      }
    }
  }
  return doc;
}
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `javascript-index`.

Configuration Properties

Property	Description, Type
script Script Body *required*	type: `string`

14.3.12. JDBC Index Stage

The JDBC Index Stage is used to connect to a database, lookup one or more values, and then inject them into the context. The properties for setting stage are identical to the JDBC Query Stage, except for the "rows" property, which defaults to **-1** for the index stage (which returns all rows in the database), and defaults to **10** returned rows for the query stage.

Note	You must first upload the JDBC driver to Fusion, see the Connector JDBC API page.

Example

An example of a JDBC Index Stage setup

Upload stage config via POST to Fusion REST API endpoint `api/apollo/index-stages/instances`

```
curl -u user:password -X POST -H "Content-Type: application/json" -d '{"id": "jdbc-index-test","type":"jdbc-index-lookup","driver":"postgresql-9.3-1101-jdbc4.jar","connection":"jdbc:postgresql:database","username":"user","password":"password1","preparedStatement":"select ID as id from DATABASE;"}' http://localhost:8764/api/apollo/index-stages/instances
```

Response

```
{
  "type" : "jdbc-index-lookup",
  "id" : "jdbc-index-test",
  "driver" : "postgresql-9.3-1101-jdbc4.jar",
  "connection" : "jdbc:postgresql:database",
  "username" : "user",
  "password" : "password1",
  "preparedStatement" : "select ID as id from DATABASE;",
  "fetchSize" : -1,
  "join" : true,
  "rows" : -1,
  "skip" : false,
  "label" : "jdbc-index-lookup",
  "type" : "jdbc-index-lookup"
}
```

Configuration

When using Fusion's REST-API, the ID of this stage is: `jdbc-index-lookup`.

Configuration Properties

Property	Description, Type
connection Connection URI *required*	The connection string for the database type: `string`

Property	Description, Type
driver JDBC Driver *required*	The fully qualified class name of the JDBC Driver to use. type: `string`
fetchSize Fetch Size	The JDBC Fetch Size to use. If -1, use the driver default. type: `integer` default value: ‘-1’
join Join with Document	If true, the results will be added on to the document using the prefix key and the row id, else the results will be put in the pipeline context using type: `boolean` default value: ‘true’
password Password *required*	The password to connect to the database. type: `string`
prefix Result Prefix Key	The string to use as a prefix for all values type: `string`
preparedStatement SQL Prepared Statement *required*	The SQL Prepared Statement to execute when bound with values. type: `string`
preparedStatementKeys Prepared Statement Keys	The keys in the Pipeline Context to use to map request attributes into the prepared statement. These must map to the '?'s in your prepared statement. They must also be able to be resolved as the first parameter of that name in a request. type: `array` of `string`

Property	Description, Type
rows Rows	The number of rows to return. -1 for all rows (be wary of memory usage for this) type: `integer` default value: '-1'
username Username *required*	The username to connect to the database. type: `string`

14.3.13. JSON Parsing Index Stage

A JSON Parsing Index stage (previously called the JSON Parser stage) parses JSON content from a document field into one or more new documents.

This stage uses Solr’s JsonRecordReader to create an index stage capable of splitting JSON into sub-documents. For details on the use of this stage in Solr, see this Lucidworks blog post: Indexing Custom JSON Data.

Example Specification, Data, Results

Stage Specification

```
{ "type": "json-parsing",
  "skip": false,
  "id": "json-parsing",
  "sourceField": "data",
  "splitPath": "/exams",
  "mappingRules": [
      {"path": "/first", "field": "first"},
      {"path": "/last", "field": "last"},
      {"path": "/grade", "field": "grade"},
      {"path": "/exams/subject", "field": "subject"},
      {"path": "/exams/test", "field": "test"},
      {"path": "/exams/marks", "field": "marks"}
  ]
}
```

Data

```
{
  "first": "John3",
  "last": "Doe",
  "grade": 8,
  "exams": [
      {
        "subject": "Maths",
        "test"   : "term1",
        "marks":90},
        {
         "subject": "Biology",
         "test"   : "term1",
         "marks":86}
      ]
}
```

Results

Parsing this data, using the splitPath "/exams" and the six mapping rules above, produces two documents, one for each object in the list of exams.

The first document has the following field, value pairs:

```
* first : John
* last : Doe
* grade : 8
* test : term1
* subject: Maths
* marks : 90
```

The second has the following field, value pairs:

```
* first : John
* last : Doe
* grade : 8
* test : term1
* subject: Biology
* marks : 86
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `json-parser`.

Configuration Properties

Property	Description, Type
inheritFields Inherit fields	Should generated documents include the original document’s fields type: `boolean` default value: ‘false’
mappingRules JSON mapping rules	type: `array` of `object` object attributes: \{ `field` : \{ display name: Destination field name type: `string` description : If omitted, will default to the extracted JSON property } `path` *(required)* : \{ display name: JSON path to extract type: `string` } }

Property	Description, Type
sourceEncoding Source field encoding	The content transport encoding of the data stored in the source field (per RFC1341) type: `string` default value: ‘binary’ enum: \{ binary base64 }
sourceField Source Field	The document field where the JSON content should be read from type: `string` default value: ‘*raw_content*’
splitPath JSON split point *required*	type: `string` default value: ‘/’

14.3.14. Detect Language Index Stage

The Detect Language index stage (called the Language Detection stage in versions earlier than 3.0) operates over one of more fields in the Pipeline Document. The contents of each field are analyzed using the Language Detection Library for Java, which is an open-source project hosted on GitHub. The analyzer returns the id of the language which best matches the contents of that field, if any. These ids can be returned as an annotation on the Pipeline Document context, or as annotation on each field analyzed.

The language identification algorithm breaks the text in each source field into ngrams and compares them to sets of ngrams compiled from all the different language versions of the Wikipedia. This library will only produce reasonable results for document fields which are comparable in length, vocabulary, and style to the known texts compiled from the Wikipedia. Caveats are discussed below.

If a positive language identification is made, that information is added to the Pipeline Document according to the choice of configuration property "Output Type". If the language annotation is added to the PipelineDocument context object, the name of the context key string is specified by configuration property "Output Key". For Output Type configuration property "Document", per-field language annotations are added to the document using a parallel naming convention where the name of the language identification field starts with the name of the analyzed field and has an additional suffix string, default value "_lang". For example, if a document contains fields named "plot_summary_txt" and "user_reviews_txt" to be analyzed, if the software can detect the language, it will add fields "plot_summary_txt_lang" and "user_reviews_txt_lang".

Languages

The Language Detection Library for Java has build-in profiles for many languages. If there is a set of Wikipedia entries written in a language, it is likely that the Language Detection Library can identify texts written in this language.

Caveats

This library should produce reasonable results on document fields which are comparable in length, vocabulary, and style to the known texts compiled from the Wikipedia.

The documentation lists the following challenges:

- This software does not work as well when the input text to analyze is short, or unclean. For example tweets.
- When a text is written in multiple languages, the default algorithm of this software is not appropriate. You can try to split the text (by sentence or paragraph) and detect the individual parts. Running the language guesser on the whole text will just tell you the language that is most dominant, in the best case.
- This software cannot handle it well when the input text is in none of the expected (and supported) languages.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `lang-detect`.

Configuration Properties

Property	Description, Type
documentPostfix Document Postfix	The postfix to add to the source name when storing the results on the document (via the output type) type: `string` default value: '_lang'
outputKey Output Key	The name of the key to insert into the context if the output type is 'context'. The value is a map of source name to language. May be a String Template. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation type: `string` default value: 'languages'
outputType Output Type	Should we set the flag on the document or in the Pipeline Context? type: `string` enum: \{ document context }
source Source *required*	The fields/context keys to detect on. May be a String Template. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation type: `array` of `string` minimum number of items (`minItems`): 1

14.3.15. Logging Index Stage

The Logging Index stage prints messages to the Connectors log file, default location `fusion/var/log/connectors/connectors.log`.

The verbosity of this message is controlled by the property `detailed`. When detailed logging is true the current PipelineDocument object is pretty-printed to the Connectors log file.

In a production environment logging stages should be configured with property `skip` set to `true`, if possible. Use of detailed logging may impact performance.

Configuration

When using Fusion's REST-API, the ID of this stage is: `index-logging`.

Configuration Properties

Property	Description, Type
detailed Detailed Logging	type: `boolean` default value: 'false'

14.3.16. Write Log Message Index Stage

The Write Log Message Index stage (called the Log a Message stage in versions earlier than 3.0) is an extension of Fusion’s Logging Index Stage, which logs any message sent to the configured logging system using the Messaging Services.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `logging-message`.

Configuration Properties

Property	Description, Type
errorKey Message Response Failure Key	The name of the key to store a boolean if sending a message failed. If set, you can check the MessageResponse errorCode and other attributes for the reason. type: `string` default value: ‘messageResponseFailure’
from From *required*	Who the message is from. May be a string template similar to the body and subject. type: `string`
logLevel Log Level	The Log Level. May be: debug, info, warn, error type: `string` enum: \{ debug info warn error }
messageBodyTemplate Message Body Template *required*	A Message Template that is used to create the message body to send. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation for details on the template language type: `string` default value: '`Processing Document ’

Property	Description, Type
messageSubjectTemplate Message Subject Template *required*	A Message Template that is used to create the message subject to send. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation for details on the template language type: `string` default value: ‘Hello’
responseKey Message Response Context Key	The name of the key to store the MessageResponse under in the Pipeline Context. type: `string` default value: ‘messageResponse’
storeInContext Add to Pipeline Context	Put the generated Message later in the pipeline. type: `boolean` default value: ‘false’
to To *required*	Who to send the message to. May be a string template similar to the body and subject. type: `array` of `string`

14.3.17. Resolve Multivalued Fields Index Stage

The Resolve Multivalued Fields stage (previously called the Multi-Value Resolver stage) allows you to choose one value from a set of one or more field values using a set of pre-defined rules, based on either field name or field type. For each field name or field type rule, a strategy is defined. There are 6 available strategies:

- DEFAULT: leave all values intact. This may cause problems if a field or dynamic field rule is not defined as multi-valued in the schema.
- PICK_FIRST: choose the first value and discard all others.
- PICK_LAST: choose the last value and discard all others.
- PICK_BY_CREATOR: based on the name of the "creator" metatdata, choose the name of 'creator' as defined in the creatorStrategy property for the field name. Only the last matching value will be retained.
- PICK_MAX: choose the maximum value and discard all others. If the values are non-numeric, then all non-numeric values will be discarded.
- PICK_MIN: choose the minimum value and discard all others. If the values are non-numeric, then all non-numeric values will be discarded.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `multivalue-resolver`.

Configuration Properties

Property	Description, Type
creatorStrategy Creator Strategies	Mapping of fields to a 'creator' name type: `array` of `object` object attributes: \{ `creatorName` *(required)* : \{ display name: Creator Name type: `string` } `fieldName` *(required)* : \{ display name: Field Name type: `string` } }

Property	Description, Type

fieldStrategy
Field Strategies
	Mapping of fields to a strategy
	type: `array` of `object`
	object attributes: \{
	`fieldName` *(required)* : \{
	display name: Field Name
	type: `string`
	}
	`resolverStrategy` *(required)* : \{
	display name: Resolver Strategy
	type: `string`
	default value: ‘pick_first’
	enum: \{ pick_first pick_last pick_by_creator concatenate_unique pick_min pick_max default }
	`} +`
	`}`

typeStrategy
Type Strategies
	Mapping of types to a strategy. These supersede field strategies.
	type: `array` of `object`
	object attributes: \{
	`fieldName` *(required)* : \{
	display name: Field Name
	type: `string`
	}
	`resolverStrategy` *(required)* : \{
	display name: Resolver Strategy
	type: `string`
	default value: ‘pick_first’
	enum: \{ pick_first pick_last pick_by_creator concatenate_unique pick_min pick_max default }
	`} +`
	`}`

14.3.18. OpenNLP NER Extraction Index Stage

Named Entity Recognition (NER) is the task of finding the names of persons, organizations, locations, and/or things in a passage of free text. The OpenNLP NER Extraction index stage (previously called the OpenNLP NER Extractor stage) uses a set of rules to find named entities in a field in the Pipeline Document (the "source") and populates a new fields (the "target") with these entities.

This stage uses Apache OpenNLP project’s Named Entity Recognition tool (the Name Finder tool). The OpenNLP documentation states

> The Name Finder tool can detect named entities and numbers in text. To be able to detect entities the Name Finder needs a model. The model is dependent on the language and entity type it was trained for. The OpenNLP projects offers a number of pre-trained name finder models which are trained on various freely available corpora. They can be downloaded at our model download page. To find names in raw text the text must be segmented into tokens and sentences.

See this video tutorial for a demonstration of how to configure this stage:

Models are available from the OpenNLP models SourceForge repository.

The Fusion directory fusion/data/nlp contains a set of NER models for English, as well as sentence, token, and part-of-speech models.

Before they can be used, model files must be uploaded to Fusion using the Fusion Blob Store service via the REST API. Here is an example of how to upload the sentence model file from the fusion using the curl command-line utility, where admin is the name of a user with admin privileges, and pass is the password:

```
curl -u admin:pass -X PUT --data-binary @data/nlp/models/en-sent.bin -H 'Content-type: application/octet-
stream' http://localhost:8764/api/apollo/blobs/en-sent.bin
```

Example Specification

Specification of a stage which extracts names of people and places from field named 'body':

```
{
    "type" : "nlp-extraction",
    "id" : "nd",
    "rules" : [ {
      "source" : [ "body_s" ],
      "target" : "content",
      "sentenceModel" : "en-sent.bin",
      "tokenizerModel" : "en-token-1.bin",
      "entityTypes" : [ {
        "name" : "organization",
        "definition" : "en-ner-organization-1.bin"
      }, {
        "name" : "person",
        "definition" : "en-ner-person-1.bin"
      } ]
    } ],
    "skip" : false
  } ]
}
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `nlp-extractor`.

Configuration Properties

Property	Description, Type
rules Extractor Rules *required*	type: `array` of `object` minimum number of items (`minItems`): 1 object attributes: \{ `entityTypes` : \{ display name: Entity Types type: `array` of `object` } `sentenceModelLocation` *(required)* : \{ display name: Sentence Model type: `string` reference : /blobs } `source` *(required)* : \{ display name: Source Fields type: `array` of `string` minItems : 1 } `target` *(required)* : \{ display name: Target Field type: `string` } `tokenizerModelLocation` *(required)* : \{ display name: Tokenizer Model type: `string` reference : /blobs } }

14.3.19. Tag Part-of-Speech Index Stage

The Tag Part-of-Speech Index stage (previously called the Part of Speech stage) operates over one of more fields in the Pipeline Document. It marks sentences with part of speech information as annotations which can be used by downstream indexing stages. Therefore this stage requires a Detect Sentences stage defined over these fields earlier in the pipeline.

This stage uses Apache OpenNLP project's Part of Speech Tagger to mark tokens with their corresponding word type based on the token itself and the context of the token. The OpenNLP documentation states:

"A token might have multiple pos tags depending on the token and the context. The OpenNLP POS Tagger uses a probability model to predict the correct pos tag out of the tag set. To limit the possible tags for a token a tag dictionary can be used which increases the tagging and runtime performance of the tagger."

Fusion comes with a set of OpenNLP language models for english. These data files are found in the directory: `fusion/data/nlp/models`.

More models are available from the OpenNLP models SourceForge repository. Model files must be uploaded to Fusion using the Fusion Blob Store service via the REST API.

Part-of-speech Tagging in a NLP Pipeline

The following video shows how to use a Part-of-speech indexing stage as part of an NLP pipeline:

Stage Setup

Here is an example of how to upload a part-of-speech model file to the Fusion blob store:

INPUT

```
curl -u user:pass -X PUT --data-binary @en-pos-maxent.bin -H 'Content-type: text/plain'
http://localhost:8764/api/apollo/blobs/en-pos-maxent.bin
```

OUTPUT

```
{
  "name" : "en-pos-maxent.bin",
  "contentType" : "text/plain",
  "size" : 5696197,
  "modifiedTime" : "2015-07-15T06:57:48.636Z",
  "version" : 0,
  "md5" : "db2cd70395b9e2e4c6b9957015a10607"
}
```

This is an example setup of this stage using the previously loaded .bin file:

INPUT

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d '{"id":"TagPartofSpeech1", "type": "tag-part-of-speech","tokenizerModel":"en-pos-maxent.bin","posModel":"en-pos-perceptron.bin","source":
["sample","text","for","NLP"]}' http://localhost:8764/api/apollo/index-stages/instances
```

OUTPUT

```
{
  "type" : "tag-part-of-speech",
  "id" : "TagPartofSpeech1",
  "posModel" : "en-pos-perceptron.bin",
  "tokenizerModel" : "en-sent.bin",
  "source" : [ "sample", "text", "for", "NLP" ],
  "skip" : false,
  "label" : "tag-part-of-speech",
  "type" : "tag-part-of-speech"
}
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `part-of-speech`.

Configuration Properties

Property	Description, Type
posModel Part of Speech Model *required*	type: `string` reference: /blobs
source Source Fields *required*	type: `array` of `string` minimum number of items (`minItems`): 1
tokenizerModel Tokenizer Model *required*	type: `string` reference: /blobs

14.3.20. Regex Field Extraction Index Stage

The Regex Field Extraction stage (called the Regular Expression Extractor stage in versions earlier than 3.0) is used to extract entities from documents based on matching regular expressions. The resulting regex matches over the contents of the source field are copied to the target field. The regular expression, source, and target fields are defined properties of this stage.

If using the REST API, this stage type is named "regex-extractor".

Example Stage Specification

Define a regex-field-extraction stage to apply a regular expression that looks for storage capabilities of products when it appears in the product 'name' field, and store it in a special field:

```
{
  "type" : "regex-field-extraction",
  "id" : "storagesize-regex-extraction",
  "rules" : [ {
    "source" : [ "name" ],
    "target" : "storage_size_ss",
    "pattern" : "(\\d{1,20}\\s{0,3}(GB|MB|TB|KB|mb|gb|tb|kb))",
    "annotateAs" : "storage_size"
  } ],
  "skip" : false
}
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `regex-extractor`.

Configuration Properties

Property	Description, Type
rules Regex Rules	type: `array` of `object` object attributes: \{ `annotateAs` : \{ display name: Annotation Name type: `string` } `group` : \{ display name: Regex Capture Group type: `integer` default value: ‘0’ exclusiveMinimum : false minimum : 0 } `pattern` *(required)* : \{ display name: Regex Pattern type: `string` format : regex } `source` *(required)* : \{ display name: Source Fields type: `array` of `string` minItems : 1 } `target` *(required)* : \{ display name: Target Field type: `string` } }

14.3.21. Regex Field Filter Index Stage

The Regex Field Filter Index Stage (called the Regular Expression Filter stage in versions earlier than 3.0) removes a field or fields from a PipelineDocument according to a set of filters where each filter specifies a field name and a regular expression. If a field value matches the regular expression, the field is deleted from the document. The regex patterns follow Java regular expression pattern rules.

Example Stage Specification

Create a regex-filter to find Social Security Numbers and drop them from documents:

```
{
  "type" : "regex-field-replacement",
  "id" : "ssnFilter",
  "skip" : false,
  "filters" : [ {
    "sourceField" : "notes_t",
    "pattern" : "^\\d{3}-\\d{2}-\\d{4}$"
  } ]
}
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `regex-filter`.

Configuration Properties

Property	Description, Type
filters Filters	type: `array` of `object` object attributes: \{ `exclusive` : \{ display name: Filter is exclusive? type: `boolean` default value: ‘true’ description : Exclude only matching values when true, otherwise include only matching values. } `pattern` *(required)* : \{ display name: Filter Pattern type: `string` format : regex } `sourceField` *(required)* : \{ display name: Source Field type: `string` } }

Property	Description, Type
reservedFieldsFilteringAllowed Allow system fields (special fields reserved for internal use) filtering?	type: `boolean` default value: ‘false’

14.3.22. Send SMTP Email Index Stage

The Send SMTP Email index stage was renamed for Fusion 3.0; it was called the Send email (via SMTP) stage in previous versions.

This stage sends an SMTP message from Fusion, for alerting, reporting, and more, using Fusion’s Messaging Services.

Enabling Email Messaging

Before you can use the Email pipeline stage, you must enable Email messaging in Fusion:

1. Click **Applications** > **System** > **Messaging Services**.
2. Select **SMTP Message Service** from the drop-down menu.
3. Verify that the default settings are sufficient.
4. Click **Save message service**.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `smtp-message`.

Configuration Properties

Property	Description, Type
errorKey Message Response Failure Key	The name of the key to store a boolean if sending a message failed. If set, you can check the MessageResponse errorCode and other attributes for the reason. type: `string` default value: ‘messageResponseFailure’
from From *required*	Who the message is from. May be a string template similar to the body and subject. type: `string`
messageBodyTemplate Message Body Template *required*	A Message Template that is used to create the message body to send. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation for details on the template language type: `string` default value: '`Processing Document ’

Property	Description, Type
messageSubjectTemplate Message Subject Template *required*	A Message Template that is used to create the message subject to send. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation for details on the template language type: `string` default value: ‘Hello’
responseKey Message Response Context Key	The name of the key to store the MessageResponse under in the Pipeline Context. type: `string` default value: ‘messageResponse’
smtpPassword SMTP Password *required*	The SMTP password for the user credentials type: `string`
smtpUser SMTP Username *required*	The SMTP user to send the message from type: `string`
storeInContext Add to Pipeline Context	Put the generated Message later in the pipeline. type: `boolean` default value: ‘false’
to To *required*	Who to send the message to. May be a string template similar to the body and subject. type: `array` of `string`

14.3.23. Send PagerDuty Message Index Stage

This stage sends a PagerDuty Message from Fusion, for alerting, monitoring, and more, using Fusion’s Messaging Services.

Read more about PagerDuty integration in Fusion on our blog.

Enabling PagerDuty Messaging

Before you can use the PagerDuty pipeline stage, you must enable PagerDuty messaging in Fusion:

1. Click **Applications** > **System** > **Messaging Services**.
2. Select **PagerDuty Message Service** from the drop-down menu.
3. Enter the following information:
    - PagerDuty service key
    - PagerDuty Service API URL; this should be `https://events.pagerduty.com/generic/2010-04-15/create_event.json`.
4. Click **Save message service**.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `pagerduty-message`.

Configuration Properties

Property	Description, Type
client Client	The name of the monitoring client that is triggering this event. type: `string` default value: ‘Fusion’ minLength: 1
clientURL Client URL	The URL of the monitoring client that is triggering this event. type: `string` default value: ‘fusion-monitoring.yourdomain.com’ minLength: 1

Property	Description, Type
errorKey Message Response Failure Key	The name of the key to store a boolean if sending a message failed. If set, you can check the MessageResponse errorCode and other attributes for the reason. type: `string` default value: ‘messageResponseFailure’
eventType Event Type *required*	The Pager Duty Event Type. One of: trigger, acknowledge, resolve type: `string` default value: ‘trigger’ enum: \{ trigger acknowledge resolve }
incidentContextImages Incident Context Images	type: `array` of `object` object attributes: \{ `alt` : \{ display name: Alternate Text type: `string` description : HTM 'alt' tag for the image } `href` *(required)* : \{ display name: Target Link type: `string` description : URL to open when clicked on the image } `src` *(required)* : \{ display name: Source type: `string` description : HTML 'src' tag for the image. Should be always secure connection (https://) } }

Property	Description, Type
incidentContextLinks Incident Context Links	type: `array` of `object` object attributes: \{ `href` *(required)* : \{ display name: Target Link type: `string` description : URL to open when clicked on the link } `text` : \{ display name: Text type: `string` description : Arbitrary text explaining the URL } }
incidentDescription Description *required*	A short description of the problem that led to this trigger. This field (or a truncated version) will be used when generating phone calls, SMS messages and alert emails. It will also appear on the incidents tables in the PagerDuty UI. The maximum length is 1024 characters. type: `string` default value: ‘Sample Description’ maxLength: 1024 minLength: 1
incidentDetails Incident Details	type: `array` of `object` object attributes: \{ `name` *(required)* : \{ display name: Name type: `string` } `value` *(required)* : \{ display name: Value type: `string` } }

Property	Description, Type
incidentKey Incident Key	Identifies the incident to which this trigger event should be applied. If there’s no open (i.e. unresolved) incident with this key, a new one will be created. If there’s already an open incident with a matching key, this event will be appended to that incident’s log. type: `string` default value: '`Incident ’ minLength: 1
responseKey Message Response Context Key	The name of the key to store the MessageResponse under in the Pipeline Context. type: `string` default value: ‘messageResponse’
storeInContext Add to Pipeline Context	Put the generated Message later in the pipeline. type: `boolean` default value: ‘false’

14.3.24. Send Slack Message Index Stage

This stage sends a Slack message from Fusion, for alerting, reporting, and more, using Fusion's Messaging Services.

Enabling Slack Messaging

Before you can use the Slack pipeline stage, you must enable Slack messaging in Fusion:

1. Click **Applications** > **System** > **Messaging Services**.
2. Select **Slack Message Service** from the drop-down menu.
3. Enter the following information:
   - Slack auth token
   - Message template

     The default is `<subject> : <body>`, which are configured with `messageSubjectTemplate` and `messageBodyTemplate` below. See Messaging Services Templates for details on the template language.
   - Optionally, you can configure a proxy or the error reporting channel name.
4. Click **Save message service**.

Configuration

When using Fusion's REST-API, the ID of this stage is: `slack-message`.

Configuration Properties

Property	Description, Type
errorKey Message Response Failure Key	The name of the key to store a boolean if sending a message failed. If set, you can check the MessageResponse errorCode and other attributes for the reason. type: `string` default value: 'messageResponseFailure'
from From *required*	Who the message is from. May be a string template similar to the body and subject. type: `string`

Property	Description, Type
messageBodyTemplate Message Body Template *required*	A Message Template that is used to create the message body to send. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation for details on the template language type: `string` default value: '`Processing Document ’
messageSubjectTemplate Message Subject Template *required*	A Message Template that is used to create the message subject to send. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation for details on the template language type: `string` default value: ‘Hello’
responseKey Message Response Context Key	The name of the key to store the MessageResponse under in the Pipeline Context. type: `string` default value: ‘messageResponse’
storeInContext Add to Pipeline Context	Put the generated Message later in the pipeline. type: `boolean` default value: ‘false’
to To *required*	Who to send the message to. May be a string template similar to the body and subject. type: `array` of `string`

14.3.25. Detect Sentences Index Stage

The Detect Sentences index stage (called the Sentence Detection stage in versions earlier than 3.0) operates over one of more fields in the Pipeline Document and annotates each field with sentence boundary information. These annotations can be used by downstream indexing stages. A Detect Sentences stage can be used in tandem with a Tag Part-of-Speech Index Stage to provide part-of-speech annotations for the individual tokens in the field.

This stage uses Apache OpenNLP project's Sentence Detection tool. The OpenNLP documentation states:

The OpenNLP Sentence Detector can detect that a punctuation character marks the end of a sentence or not. In this sense a sentence is defined as the longest white space trimmed character sequence between two punctuation marks. The first and last sentence make an exception to this rule. The first non whitespace character is assumed to be the begin of a sentence, and the last non whitespace character is assumed to be a sentence end. _

Fusion comes with a set of OpenNLP language models for english. These data files are found in the directory: `fusion/data/nlp/models`. The included sentence model is `en-sent.bin`. More models are available from the OpenNLP models SourceForge repository.

Model files must be uploaded to Fusion using the Fusion Blob Store service via the REST API (see examples below).

Sentence Detection in a NLP Pipeline

The following video shows how to use a Sentence Detection index stage as part of an NLP pipeline:

Stage Setup

This is an example of how to upload a sentence model file to the Fusion blob:

INPUT

```
curl -u user:pass -X PUT --data-binary @en-pos-maxent.bin -H 'Content-type: text/plain'
http://localhost:8764/api/apollo/blobs/en-pos-maxent.bin
```

OUTPUT

```
{
  "name" : "en-sent.bin",
  "contentType" : "text/plain",
  "size" : 5696197,
  "modifiedTime" : "2015-07-15T06:57:48.636Z",
  "version" : 0,
  "md5" : "db2cd70395b9e2e4c6b9957015a10607"
}
```

This is an example setup of this stage using the previously loaded .bin file:

INPUT

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d '{"id":"DetectSentences1", "type": "detect-
sentences","sentenceModel":"en-sent.bin","source": ["A test sentence"]}'
http://localhost:8764/api/apollo/index-stages/instances
```

OUTPUT

```
{
  "type" : "detect-sentences",
  "id" : "DetectSentences1",
  "sentenceModel" : "en-sent.bin",
  "source" : [ "A test sentence" ],
  "skip" : false,
  "label" : "detect-sentences",
  "type" : "detect-sentences"
}
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `part-of-speech`.

Configuration Properties

Property	Description, Type
posModel Part of Speech Model *required*	type: `string` reference: /blobs
source Source Fields *required*	type: `array` of `string` minimum number of items (`minItems`): 1
tokenizerModel Tokenizer Model *required*	type: `string` reference: /blobs

14.3.26. Set Property Index Stage

The Set Property Index Stage is used to set a value on a document, or into the Pipeline Context for downstream consumption by specifying a series of simple matching conditions. These conditions match against whether a field exists, and simple substring matches on the field contents. For more complex logic, use a JavaScript Index Stage.

Configuration

When using Fusion's REST-API, the ID of this stage is: `index-set-property`.

Configuration Properties

Property	Description, Type
falseOutputValue False Output Value	The value to set when nothing matches. If null/empty, then nothing will be output (same behavior as in Fusion 2.2). May be a String Template. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation type: `string`
ifEquals If Value Equals	Check whether a value equals another value. Valid entries for this are: another field/context key, the word 'null' or 'not null' (without the quotes) or a literal string (e.g. "match") in quotes (single or double). The value may be a String Template. type: `string`
outputKey Output Key	The name of the key to insert into the output location (document or context). May be a String Template. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation type: `string` default value: 'setPropertyKey'
outputType Output Type *required*	Should we set the flag on the document or in the Pipeline Context? type: `string` default value: 'context' enum: \{ document context }

Property	Description, Type
outputValue Output Value	The value to set. May be a String Template. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation type: `string` default value: ‘true’
regularExpression Regular Expression Match	Apply the regular expression to the source fields and or context keys. If any of them match, than set the property. The implementation returns true if the regular expression matches anywhere in the string. type: `string`
source Source	The source fields and context keys to check conditions against. If just the source fields are set and nothing else, than only set the property if all fields are present and non-null on the document. May be a String Template. type: `array` of `string`
whatMatchedKey What Matched Key	The name of the context key to use to store a space separated list of what conditionals matched. type: `string` default value: ‘whatMatched’

14.3.27. Filter Short Fields Index Stage

The Filter Short Fields Index Stage was renamed for Fusion 3.0; it was called the Short Field Filter stage in previous versions.

A Filter Short Fields index stage removes short field values from a pipeline document according to a set of filters where each filter specifies a field name and a minimum length. All field values less than the specified length will be removed from the document.

For the REST API, this stage type is "filter-short-fields".

Example Stage Specification

Remove entities in the `author_s` field that are less than 3 characters:

```
{
  "type" : "filter-short-fields",
  "id" : "short-stage",
  "filters" : [ {
    "sourceField" : "author_s",
    "length" : 3
  } ],
  "skip" : false
}
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `short-filter`.

Configuration Properties

Property	Description, Type
filters Filters	type: `array` of `object` object attributes: \{ `length` *(required)* : \{ display name: Minimum Length type: `integer` default value: ‘3’ description : Minimum length of permissible entities exclusiveMinimum : false minimum : 1 } `sourceField` *(required)* : \{ display name: Source Field type: `string` } }

14.3.28. Format Signals Index Stage

The Format Signals stage (called the Signal Formatter stage in versions earlier than 3.0) normalizes both the fields and field contents of a PipelineDocument to ensure that certain pre-defined fields for signals are populated.

Date/time parsing and formatting

Timestamp data can be obtained from the following fields, in this order of precedence:

- timestamp
- timestamp_tdt
- timestamp_dt
- epoch - value in this field is treated as a number of milliseconds since epoch, and UTC zone is assumed.

It’s now possible to specify the locale to be used for parsing timestamps by setting the "timestampLocale" property in the stage configuration. If this property is null then the default platform locale will be used.

Output document will carry the following two fields:

- "timestamp" - containing the ISO8601 timestamp
- "tz_timestamp_txt" - containing the "zoned format" of the timestamp with normalized components.

Example Stage Specification

_The Format Signals stage defined as part of the default 'signals_ingest' pipeline included with Fusion.
Note that this stage does not define a list of allowed types. _

```
{
    "type" : "format-signals",
    "id" : "ingest-signals",
    "flatten" : true,
    "undefinedType" : "general",
    "skip" : false,
    "label" : "format-signals",
    "type" : "format-signals"
  }
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `signal-formatter`.

Configuration Properties

Property	Description, Type
allowedTypes Allowed Types	List of allowed signal types. type: `array` of `string`

Property	Description, Type
flatten Flatten	Flatten nested values. type: `boolean` default value: ‘true’
timestampLocale Timestamp locale	Use this locale when parsing timestamp information. Null uses platform default locale. type: `string`
undefinedType Undefined Type	Signal type when undefined. Null discards events with undefined type. type: `string`

14.3.29. Solr Indexer Stage

A Solr Indexer stage transforms a Fusion PipelineDocument object into a Solr document and sends it to Solr for indexing into a collection.

A PipelineDocument object contains fields which take as their values either a string or list of strings. Solr fields have a rich variety of types. The Solr Indexer stage transforms PipelineDocument field values into Solr document fields. The Solr Indexer stage can be configured so that it will try to ensure that all document fields are valid Solr fields. This feature is convenient, but offers very little control over how fields and field values are transformed, especially with respect to dates. A Date Parsing stage offers greater control over date values.

Configuration

When using Fusion's REST-API, the ID of this stage is: `solr-index`.

Configuration Properties

Property	Description, Type
bufferDocsForSolr Buffer documents and send them to Solr in batches	type: `boolean` default value: 'false'
dateFormats Additional date formats	type: `array` of `string`
enforceSchema Map to Solr schema	type: `boolean` default value: 'false'
params Additional update request parameters	type: `array` of `object` object attributes: \{ `key` *(required)* : \{ display name: Parameter Name type: `string` } `value` : \{ display name: Parameter Value type: `string` } }

Property	Description, Type

unmapped
 	
Unmapped fields mapping
	type: `object`
	
	object attributes: \{
	`operation` : \{
	display name: Operation
	type: `string`
	default value: ‘copy’
	description : The type of mapping to perform: move, copy, delete, add, set, or keep
	enum: \{ copy move delete set add keep }
	
	`} +`
	`` `source` : \{ + ``
	`display name: Source Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped. +`
	`} +`
	`` `target` : \{ + ``
	`display name: Target Field +`
	`` type: `string` + ``
	`description : The name of the field to be mapped to. +`
	`} +`
	}

14.3.30. Solr Partial Update Indexer Stage

The Solr Partial Update Indexer Stage updates of one or more fields of an existing Solr document in a collection managed by Fusion. It provides an alternative to the Solr Indexer stage. The Solr Partial Update Indexer stage was introduced in Fusion 2.1.

When a data feed consists of an ongoing flow of messages about known documents in a collection, such as item price, inventory counts, or weather conditions at a location, this stage provides fast indexing throughput and can be configured to enforce data atomicity to guarantee that the index always reflects the most recent update.

This stage is configured with a set of update directives based on Solr's atomic updates. At run time, it creates a Solr update by applying these directive to the data from a Fusion PipelineDocument object and then submits this update to Solr's update handler.

Note	Solr's atomic update functionality requires that the schema for a collection is configured so that all fields have the attribute stored="true", excepting fields which are <copyField/> destinations which must be configured as stored="false".

Example Stage Specification

Configuration for a Partial Updater Stage in JSON:

```
{ "type" : "solr-partial-update-index",
  "enforceSchema" : false,
  "solrDocIdFieldName" : "id",
  "solrDocIdFieldValue" : "<doc.id>",
  "updatedFields" : [
    { "updateType" : "set", "fieldName" : "statusValue", "values" : "<doc.statusValue>" },
    { "updateType" : "set", "fieldName" : "lastCommunicationTime", "values" : "<doc.lastCommunicationTime>" }
  ],
  "concurrencyControlEnabled" : true,
  "skip" : false,
  "label" : "solr-partial-update-index",
  }
```

The expression <doc.X> will evaluate to the contents of the current PipelineDocument's field named "X".

Types of Update Operations

The set of update operations are based on operations supported by Solr. They are:

- 'add' - add a new value or values to an existing Solr document field, or add a new field and value(s).
- 'set' - change the value or values in an existing Solr document field.
- 'remove' - remove all occurrences of the value or values from an existing Solr document field.
- 'removeregex' - remove all occurrences of the values which match the regex or list of regexes from an existing Solr document field.
- 'increment' - increment the the numeric value of existing Solr document field by a specific amount.
- 'decrement' - decrement the the numeric value of existing Solr document field by a specific amount.

In addition, this stage introduces experimental "Positional" operations which can be used to add, set or remove exactly one element of a field which takes a list of values (i.e, a multi-valued field).

- 'positionalUpdates' - used to add or set the value at specific position.
- 'positionalRemoves' - used to delete an element at a specific position.

When a collection contains two or more multi-value fields which are maintained in parallel so that taken together, they act like a table stored column by column, a positional update operation updates several data cells across one row of the table. To maintain this kind of column-oriented table, the positional delete directive must specify all the fields in the document which logically comprise the table.

Document Identifier Field

A Fusion collection is a Solr collection managed by Fusion. Underlyingly, a Solr document is a list of named, typed fields. The Solr unique key field stores a string which is the unique identifier for that document. There is at most one UniqueKey field per document, which is defined in the Solr schema. The UniqueKey field value is required. For collections created via Fusion, the UniqueKey field is named "id". Other document fields may also store string values which can be used as a unique identifier.

Solr uses the UniqueKey field to find the document to be updated. If the data feed information contains a document identifier which is different than the identifier value stored in the UniqueKey field, then this stage must do a Solr lookup to find the UniqueKey value.

Optimistic Concurrency

Solr’s Optimistic Concurrency is a mechanism which checks whether or not a document has changed between the point at which an update request was submitted and the point at which the request is processed. Solr documents have an internal field named "*version*" which is updated whenever there is any change made to any of the other fields in that document. When optimistic concurrency control is on, update requests will be discarded if the current version of the document has changed since that request was made. This guarantees that the document will always reflect the most recent update. However, this require an additional Solr lookup to get the current document version number, which is submitted as part of the update request.

Performance Considerations

In order to send a single update request to Solr, without preliminary lookup requests:

- The document identifier field should match the Solr collection’s UniqueKey identifier field.
- Optimistic Concurrency should be turned off.
- Positional updates are experimental and potentially expensive, since all the values for all fields being updated must be fetched into memory in order to perform positional operations.

Solr Date Formats

```
"yyyy-MM-dd'T'HH:mm:ss'Z'", // Solr format without milliseconds
"yyyy-MM-dd'T'HH:mm:ss.SSS'Z'", // standard Solr format, with literal "Z" at the end
"yyyy-MM-dd'T'HH:mm:ss.SS'Z'", // standard Solr format, with literal "Z" at the end
"yyyy-MM-dd'T'HH:mm:ss.S'Z'" // standard Solr format, with literal "Z" at the end
```

See https://cwiki.apache.org/confluence/display/solr/Working+with+Dates

Configuration

When using Fusion’s REST-API, the ID of this stage is: `solr-partial-update-index`.

Configuration Properties

Property	Description, Type
concurrencyControlEnabled Enable Concurrency Control	Select to enable Optimistic Concurrency Control in Solr, guaranteeing that the document update will not be overridden by another Partial Update to the same document. If disabled, in the case of an edit collision, the last committed update to the document will win. type: `boolean` default value: ‘true’
customRouteFieldName Custom Route Field Name	This option is used when custom shard routing is configured is Solr so the document route is defined by value of Solr document’s field (defined as 'router.field' when created the collection). If set here, the field with this name will be transferred to the partial update Solr document from the pipeline document. type: `string`
dateFormats Additional Date Formats	type: `array` of `string`
deletedFields Deletions	Fields to Delete from Solr Document. type: `array` of `object` object attributes: \{ `fields` *(required)* : \{ display name: Field type: `string` } }
enforceSchema Map to Solr Schema	type: `boolean` default value: ‘true’

Property	Description, Type
params Additional Update Request Parameters	type: `array` of `object` object attributes: \{ `key` *(required)* : \{ display name: Parameter Name type: `string` } `value` : \{ display name: Parameter Value type: `string` } }
positionalRemovals Positional Removals	Update Field or Group of Fields to remove value at a specific position. See documentation for additional information. type: `array` of `object` object attributes: \{ `fields` *(required)* : \{ display name: Fields List type: `string` description : The field [,field ...] list of fields (Solr field names) where removal of a value at specified position should happen. } `position` *(required)* : \{ display name: Position type: `string` description : The position at which the field value will be removed. Could be 'first', 'last' or numeric value (position index). } }

Property	Description, Type

positionalUpdates
 	
Positional Updates
	Update Field or Group of Fields to update (add or set) value at a specific position. See documentation for additional information.
	
	type: `array` of `object`
	
	object attributes: \{
	`fieldsAndValues` *(required)* : \{
	display name: Fields and Values
	type: `string`
	description : The field:value [,field:value ...] list. The values will be changed for specified fields at specified position. The list separator is comma (,) if the comma present in the field value, escape it with a backslash (\). Quotation marks("") can be used to enclose Field value to preserve the white spaces, if needed.
	}
	`position` *(required)* : \{
	display name: Position
	type: `string`
	description : The position at which the new value(s) will be changed. Could be 'first', 'last' or numeric value (position index).
	}
	`positionalUpdateType` : \{
	display name: Update Type
	type: `string`
	default value: ‘set’
	description : The Update Type
	enum: \{ set add }
	
	`} +`
	`}`

rejectUpdatesIfDocNotPresent Reject Update if Solr Document is not Present	Whether to reject the update attempt if the document with given id is not present in Solr. This is not typical situation since the updates usually are performed on existing documents, however you may disable this to attempt update even if the document is not present. If the concurrency control is disabled, enabling this flag will force set the *version* field to 1, or to 0 otherwise. type: `boolean` default value: ‘true’

Property	Description, Type
solrDocIdFieldName Solr Document ID Field Name *required*	type: `string` default value: ‘id’
solrDocIdFieldValue Solr Document ID Field Value *required*	type: `string` default value: '`’
updateAllDocFields Process All Pipeline Doc Fields	If this option is set, the Partial Update Stage will process pipeline document fields even if they are not set by Updates and Deletions instructions here. In this case those fields will be included into the partial update document and will be processed by Solr according to atomic update rules, i.e. non-map field value(s) will be treated as a 'set' update for the field, and Map field values will be processed as an atomic update defined in the Map. The Map structure should comply to Solr atomic update rules. Note that the Partial Update stage does NOT validate consistency of fields that are not Updates or Deletions configured here, it just sends them to Solr 'as is'. type: `boolean` default value: ‘false’

Property	Description, Type
updatedFields Updates	Fields to update (set, add or remove field values) in the Solr Document. type: `array` of `object` object attributes: \{ `fieldName` *(required)* : \{ display name: Field Name type: `string` description : The Solr Document Field to update. } `updateType` : \{ display name: Update Type type: `string` default value: ‘set’ description : The Update Type enum: \{ set add remove remove_regex increment decrement } } + \`values\` _ (required)_ : \{ + display name: Value + type: \`string\` + description : For increment operation only one value (positive or negative int) is allowed. For add, set, remove or remove_regex a single value or list of values can be specified. The list separator is a comma (,) if the comma should be present in the field value, escape it with a backslash (\\). Quotation marks("") can be used to enclose Field value to preserve the white spaces, if needed. + } + }

14.3.31. XML Transformation Index Stage

The XML Transformation stage (previously called the XML Transform Stage) allows you to process an XML document into one or more Solr documents and to specify mappings between elements and document fields. A common use case for an XML Transformation stage in a pipeline is when the XML document is a container-like document which contains a set of inner elements, each of which should be treated as a separate document. A parent ID field can be used to relate these multiple documents back to the containing document.

Pipeline Configuration

The default XML processing provided by the Apache Tika Parser index stage extracts all text from an XML into a single document field called `content`. This not only flattens the document contents, it loses all information about the containing elements in the document. To process XML documents using an XML Transformation stage, the index pipeline must have as its initial processing stage an Apache Tika Parser index stage which is configured to pass the document through to the XML Transformation stage *as raw XML*, via the following configuration:

- UI checkbox "Add original document content" **unchecked** / REST API property "addOriginalContent" set to **false**
- UI checkbox "Return parsed content as XML or HTML" **checked** / REST API property "keepOriginalStructure" set to **true**
- UI checkbox "Return original XML and HTML instead of Tika XML output" **checked** / REST API property "returnXml" set to **true**

With this configuration, the Tika parser stage decodes the raw input stream of bytes into a string containing the entire XML document which is returned in the PipelineDocument field `body`.

The pipeline must have a Field Mapping stage after the XML Transformation stage, before the Solr Indexer stage. The Field Mapping stage is used to remove the following fields from the document:

- *raw-content*
- Content-Type
- Content-Length
- parsing
- parsing_time

XML Transforms

The XML Transformation stage uses a Solr XPathRecordReader which is a streaming XML parser that supports *only a limited subset of XPath selectors*. It provides exact matching on element attributes and it can only extract the element text, not attribute values.

Examples of allowed XPath specifications where "a", "b", "c" are any element tags, likewise "attrName" is any attribute name:

```
/a/b/c
/a/b/c[@attrName='someValue']
/a/b/c[@attrName=]/d
/a/b/c/@attrName
//b//...
```

Note	When specifying the list of mappings, for each mapping, the specification for the xpath attribute must include the full path, i.e., the xpath attribute will include the rootXPath. See the example configuration below.

Example Stage Specification

Definition of an XML-Transformation stage that extracts elements from a MEDLINE/Pubmed article abstract:

```
{ "type" : "xml-transform",
  "id" : "n0j2a9k9",
  "rootXPath" : "/MedlineCitationSet/MedlineCitation",
  "bodyField" : "body",
  "mappings" : [ {
      "xpath" : "/MedlineCitationSet/MedlineCitation/Article/ArticleTitle",
      "field" : "article-title_txt",
      "multivalue" : false
  }, {
      "xpath" : "/MedlineCitationSet/MedlineCitation/Article/Abstract/AbstractText",
      "field" : "article-abstract_txt",
      "multivalue" : true
  }, {
      "xpath" : "/MedlineCitationSet/MedlineCitation/MeshHeadingList/MeshHeading/DescriptorName",
      "field" : "mesh-heading_txt",
      "multivalue" : true
  }, {
      "xpath" : "/MedlineCitationSet/MedlineCitation/PMID",
      "field" : "pmid_txt",
      "multivalue" : false
  } ],
  "keepParent" : false,
  "skip" : false,
  "label" : "medline_xml_transform",
}
```

Template for a minimal index pipeline that includes an XML-Transformation stage. Replace the XPath and field names in the XML-Transformation stage according to your data.

```
{
    "id" : "xml-pipeline-default",
    "stages" : [ {
    "type" : "tika-parser",
    "includeImages" : false,
    "flattenCompound" : false,
    "addFailedDocs" : false,
    "addOriginalContent" : false,
    "contentField" : "_raw_content_",
    "returnXml" : true,
    "keepOriginalStructure" : true,
    "extractHtmlLinks" : false,
    "extractOtherLinks" : false,
    "csvParsing" : false,
    "skip" : false,
    "label" : "tika",
    "sourceField" : "_raw_content_"
    }, {
    "type" : "xml-transformation",
    "rootXPath" : "/ROOTS/ROOT",
    "bodyField" : "body",
    "mappings" : [ {
        "xpath" : "/ROOTS/ROOT/element",
        "field" : "element-field_t",
        "multivalue" : false
    } ],
    "keepParent" : false,
    "skip" : false,
    "label" : "xml"
    }, {
    "type" : "field-mapping",
    "mappings" : [ {
        "source" : "parsing",
        "operation" : "delete"
    }, {
        "source" : "parsing_time",
        "operation" : "delete"
    }, {
        "source" : "Content-Type",
        "operation" : "delete"
    }, {
        "source" : "Content-Length",
        "operation" : "delete"
    } ],
    "skip" : false,
    "label" : "field mapping"
    }, {
    "type" : "solr-index",
    "enforceSchema" : true,
    "bufferDocsForSolr" : false,
    "skip" : false,
    "label" : "solr-index"
    } ]
}
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `xml-transform`.

Configuration Properties

Property	Description, Type
bodyField Body Field Name	type: `string` default value: ‘body’
keepParent Keep parent document	type: `boolean` default value: ‘false’
mappings XPath Mappings	type: `array` of `object` object attributes: \{ `field` *(required)* : \{ display name: Field type: `string` } `multivalue` : \{ display name: Multi Value type: `boolean` default value: ‘false’ } `xpath` *(required)* : \{ display name: XPath Expression type: `string` } }
metadata Additional Metadata	type: `array` of `object` object attributes: \{ `field` *(required)* : \{ display name: Field type: `string` } `value` *(required)* : \{ display name: Value type: `string` } }

Property	Description, Type
parentIdField Parent ID Field Name	type: `string`
rootXPath Root XPath *required*	type: `string`

14.4. Query Pipeline Stages

A query pipeline is made up of a series of query stages that process incoming search queries.

A pipeline stage definition associates a unique ID with a set of properties. These definitions are registered with the Fusion API service and stored in ZooKeeper for re-use across pipelines and search applications.

Fusion includes a number of specialized query stages as well as a JavaScript stage that allows advanced processing via a JavaScript program.

Use the The Query Workbench to configure stages in a query pipeline.

See these reference topics for details about each query pipeline stage:

- Active Directory Security Trimming
- Advanced Boosting
- Auto Facet
- Block Documents
- Boost Documents
- Email
- Facet
- Javascript
- JDBC
- Landing Pages
- Logging
- Logging Message
- Machine Learning
- PagerDuty
- Recommendation Boosting
- Return QueryParams
- Rollup Aggregator
- Search Fields
- Security Trimming
- Set Query Params Stage
- Slack
- Solr Query
- Stored Parameters
- SubQuery
- User Recommendation Boosting

14.4.1. Active Directory Security Trimming Stage

An Active Directory Security Trimming query-pipeline stage retrieves an Active Directory user's security identifiers to build a security filter. This restricts the documents in the query result to only those documents for which a user has access permissions. Security trimming is commonly used in business to authenticate between administrative users and normal users, or to limit the site access of website users according to a login/password.

Example Stage Setup

Active Directory Security Trimming query stage setup:

Input

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d '{"id":"ADSecurity", "type": "active-directory-security-trimming",
"server":"ldap://hostname:port","bindName":"ADuser@example.com","bindPassword":"login1"}'
http://localhost:8764/api/apollo/query-stages/instances
```

Output

```
{
  "type" : "active-directory-security-trimming",
  "id" : "ADSecurity",
  "server" : "ldap://hostname:port",
  "bindName" : "ADuser@example.com",
  "bindPassword" : "login1",
  "enableCache" : true,
  "cacheSize" : 1000,
  "expirationTime" : 3600,
  "skip" : false,
  "label" : "active-directory-security-trimming",
  "type" : "active-directory-security-trimming"
}
```

Configuration

When using Fusion's REST-API, the ID of this stage is: `active-directory-security-trimming`.

Configuration Properties

Property	Description, Type
bindName Bind Name *required*	username in userPrincipalName format (e.g. user@example.com) type: `string` format: @.

Property	Description, Type
bindPassword Bind Password *required*	type: `string`
cacheSize Cache Size	type: `integer` default value: ‘1000’
enableCache Enable Cache	type: `boolean` default value: ‘true’
expirationTime Cache Expiration Time	(in seconds) type: `integer` default value: ‘3600’
filterAttribute Filter Attribute	Active Directory attribute to use as the security-trimming filter criterion type: `string` enum: \{ objectSid sAMAccountName userPrincipalName }
filterDatasources Restrict Filter to Datasource(s)	A list of Fusion datasources to which security-trimming should be restricted, allowing content from other datasources to pass through un-filtered; if empty, all matching content is subject to filtering type: `array` of `string`
overrideUserIdentityHandling Override default user identity handling?	Default handling first attempts to take the user identity from a 'fusion-user-id' http-header, which is the logged-in user ID from the Fusion proxy service. If that value is empty, a 'username' query parameter is tried instead. When this DataSource property is enabled, the specified source and key properties are used explicitly, without any fallback behavior. type: `boolean` default value: ‘false’

Property	Description, Type
server Active Directory Url *required*	E.g. ldap://hostname:port type: `string` format: ldap://(:\d)?
userIdentityKey User ID key	eg. username or userID etc.. type: `string` default value: ‘username’
userIdentitySource User ID source	Specify whether the value comes from an http header or query parameter. type: `string` default value: ‘query_param’ enum: \{ query_param header }

14.4.2. Parameterized Boosting Stage

The Advanced Boosting query-pipeline stage reads the boostValues (in List<DocumentResult> format) from the context variable (added by a Rollup Aggregation stage or a JavaScript stage), and adds boosts to the main query using 'bq' or 'boost' based on the stage configuration. The weights for the boost values can also be scaled.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `adv-boost`.

Configuration Properties

Property	Description, Type
boostFieldName Boost Field *required*	The field name to boost the values on type: `string`
boostingMethod Boost Method *required*	The boost method to use. query-parser should be chosen if defType!=edismax for main query type: `string` default value: ‘query-param’ enum: \{ query-param query-parser }
boostingParam Boost Param *required*	'boost' multiplies scores by the boost values whereas 'bq' adds optional clauses to main query type: `string` default value: ‘boost’ enum: \{ boost bq }
key Context Key *required*	The key name to read from context for boost id and values type: `string`

Property	Description, Type
scaleRange Scale Boosts	Scale the boost values to a [min,max] range type: `object` object attributes: \{ `scaleMax` : \{ display name: Maximum value of the scale range type: `number` } `scaleMin` : \{ display name: Minimum value of the scale range type: `number` } }

14.4.3. Parameterized Faceting Stage

This stage was introduced in Fusion 2.4 as the Auto Facet stage. In Fusion 3.0, it was renamed to the Parameterized Faceting stage.

The Auto Facet query-pipeline stage facets documents in the query pipeline. The stage queries Solr to determine the most frequent value for a given field, and then uses that value to look up stored parameters to automatically select the most appropriate facet for a query.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `auto-facet`.

Configuration Properties

Property	Description, Type
facetFieldConfig Facet Field *required*	type: object object attributes: \{ enumCacheMinDf : \{ display name: Enum Cache Minimum DF type: integer exclusiveMinimum : false minimum : 0 } field : \{ display name: Field type: string description : The field whose values you want to facet on } limit : \{ display name: Limit type: integer description : Maximum number of facets to return } method : \{ display name: Method type: string enum: \{ enum fc fcs } } + `minCount` : \{ + display name: Minimum Count + type: `integer` + default value: '`1`' + description : Lower threshold of term counts to be included + } + `missing` : \{ + display name: Count Missing + type: `boolean` + default value: '`false`' + description : Optionally include a 'missing' facet bucket for documents without the selected field + } + `offset` : \{ + display name: Offset + type: `integer` + description : Offset into list of resulting facets + } + `prefix` : \{ + display name: Prefix + type: `string` + description : Prefix of terms to facet on + } + `sort` : \{ + display name: Sort +

Property	Description, Type
failQueryOnError Fail Query on Error	Fail the query request if an error occurs when processing this stage. type: `boolean` default value: ‘false’
fallbackValue Fallback Value	Optional fallback value to use for the facet field if one cannot be determined by querying Solr. type: `string`
handler Request handler *required*	Solr query request handler to send the auto-facet query request to. type: `string` default value: ‘select’
headers Headers	type: `array` of `object` object attributes: \{ `key` *(required)* : \{ display name: Parameter Name type: `string` } `value` : \{ display name: Parameter Value type: `string` } }
inherit Inherit Parameters From Parent?	Should a child category inherit its parents' stored parameters? type: `boolean` default value: ‘false’
method HTTP method *required*	HTTP method used to send the auto-facet query request. type: `string` default value: ‘GET’ enum: \{ GET POST }

Property	Description, Type
params Additional Query params	type: `array` of `object` object attributes: \{ `key` *(required)* : \{ display name: Parameter Name type: `string` } `value` : \{ display name: Parameter Value type: `string` } }

14.4.4. Block Documents Stage

The Block Documents query-pipeline stage removes documents from the result based on a Block Documents rule which consists of the following elements:

- `field` - the document field to filter on.
- `keywords` - the words, phrases, or regex used as the filter.
- `mode` - filtering logic applied to keywords, one of:
    - `exact` - exact matching on any item in the keywords list.
    - `phrase` - phrase matching on the items in the keywords list.
    - `regex` - treat items in the keywords list as a regex.
    - `match` - requires a match for every item in the keyword list, but doesn't require phrase matching.
- `blocks` - a list of document IDs for documents which are always removed from the query result.

The block documents rule is used to craft a Solr query. The `keywords` are added to the `q` Solr query parameter, by default. The configuration property `queryParam` can be used to specify a different Solr query parameter to use as the keywords filter. The rest of the rule is processed into the `fq` Solr query parameter.

Configuration

When using Fusion's REST-API, the ID of this stage is: `blocks`.

Configuration Properties

Property	Description, Type
queryParam Query param for matching	type: `string` default value: 'q'

Property	Description, Type

rules
Block rules
	type: `array` of `object`
	object attributes: \{
	`blocks` : \{
	display name: Blocks
	type: `array` of `string`
	description : The documents to block
	}
	`field` *(required)* : \{
	display name: Field
	type: `string`
	default value: ‘id’
	description : The name of the Solr Field to use for blocks
	}
	`keyword` *(required)* : \{
	display name: Keyword
	type: `string`
	description : Search keywords to match on
	}
	`mode` *(required)* : \{
	display name: Match Strategy
	type: `string`
	default value: ‘exact’
	description : How to match the keywords
	enum: \{ exact phrase regex match }
	`} +`
	`}`

14.4.5. Boost Documents Stage

The Boost Documents query-pipeline stage adds boosting parameters to matched documents based on specific search terms. Boosts are defined with a term value to boost and the boost factor to add. The boosting parameters are added to the `bq` Solr query parameter.

The Boost Documents rule consists of the following elements:

- `field` - the document field to filter on
- `keywords` - the words, phrases, or regex used as the filter
- `mode` - filtering logic applied to keywords, one of:
  - `exact` - the keyword and the query must match exactly. This is case-sensitive.
  - `phrase` phrase matching on the items in the keywords list
  - `regex` treat items in the keywords list as a regex
  - `match` must match every item in the keyword list, but doesn't require phrase matchin
- `boosts` - consists of a list of pairs
  - `value` - one or more terms used for boosting
  - `boost` - the numeric boost value. Default `100`.

Configuration

When using Fusion's REST-API, the ID of this stage is: `boosts`.

Configuration Properties

Property	Description, Type
queryParam Query param for matching	type: `string` default value: 'q'

Property	Description, Type

rules
Boost rules
	type: `array` of `object`
	object attributes: \{
	`boosts` : \{
	display name: Boosts
	type: `array` of `object`
	}
	`field` *(required)* : \{
	display name: Field
	type: `string`
	default value: ‘id’
	description : The name of the Solr Field to use for boosting
	}
	`keyword` *(required)* : \{
	display name: Keyword
	type: `string`
	description : Search keywords to match on
	}
	`mode` *(required)* : \{
	display name: Match Strategy
	type: `string`
	default value: ‘exact’
	description : How to match the keywords
	enum: \{ exact phrase regex match }
	`} +`
	`}`

14.4.6. Facets Stage

The Facets query-pipeline stage (previously called the Facet stage) is used to add a Solr Field Facet query to the search query pipeline. A Field facet query computes the top values for a field and returns the list of those values along with a count of the subset of documents in the search results which match that term. Field faceting works best over fields which contain a single label or set of labels from a finite, controlled lexicon such as product category. Facet field parameters can be tuned for performance, see: Facet Field Configuration.

It’s possible to specify more than one field facets. For each field facet you must specify the field name plus the following additional parameters:

- Limit – the maximum number of terms to be returned. Default 100.
- Offset – the number of top facet values to skip in the response. Default 0.
- Sort - the order in which to list facet values: `count` ordering is by documents per term, descending, and `index` ordering is sorted on term values themselves.
- Missing - the number of documents in the results set which have no value for the facet field.
- Choice of facet method (advanced) - specify Solr algorithm used to calculate facet counts. (See Facet Method Configuration for details). One of:
    - `enum` - small number of distinct categories
    - `fc` ("field cache") - many different values in the field, each document has low number of values, multi-valued field
    - `fcs` ("single value string fields") - good for rapidly changing indexes

For further details see: Solr Wiki Faceting Overview.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `facet`.

Configuration Properties

Property	Description, Type
fieldFacets Facet Fields	type: `array` of `object` object attributes: \{ `enumCacheMinDf` : \{ display name: Enum Cache Minimum DF type: `integer` exclusiveMinimum : false minimum : 0 } `field` *(required)* : \{ display name: Field type: `string` description : The field whose values you want to facet on } `limit` : \{ display name: Limit type: `integer` description : Maximum number of facets to return } `method` : \{ display name: Method type: `string` enum: \{ enum fc fcs } } + `minCount` : \{ + display name: Minimum Count + type: `integer` + default value: '`1`' + description : Lower threshold of term counts to be included + } + `missing` : \{ + display name: Count Missing + type: `boolean` + default value: '`false`' + description : Optionally include a 'missing' facet bucket for documents without the selected field + } + `offset` : \{ + display name: Offset + type: `integer` + description : Offset into list of resulting facets + } + `prefix` : \{ + display name: Prefix + type: `string` + description : Prefix of terms to facet on + } + `sort` : \{ + display name: Sort +

14.4.7. JavaScript Stage

For a complete description of the JavaScript query-pipeline stage, see: Custom JavaScript Stages For Query Pipelines.

```
    type: `string` +
    description : Ordering of the faceting results
+
    enum: \{ count index     }
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `javascript-query`.

```
    } +
  `threads` : \{ +
```

Configuration Properties

Property	Description, Type
script Script Body *required*	One context variable 'request' is visible to the script type: `string`

14.4.8. JDBC Lookup Stage

The JDBC Query Lookup query-pipeline stage is used to call out to a database as part of a pipeline stage, to inject results into either the context/request or the pipeline document. For example, if you needed to look up a user from a DB and add their profile information onto a request for downstream use in a pipeline, the JDBC Lookup Stage would facilitate this.

Note	You must first upload the JDBC driver to Fusion, see the Connector JDBC API page.

Example

An example of setup for a JDBC Query Lookup query-pipeline stage

Upload stage config via POST to Fusion REST API endpoint `api/apollo/query-stages/instances`

```
curl -u user:pass -X POST -H "Content-Type: application/json" -d '{"id": "jdbc-quer","type":"jdbc-query-
lookup","driver":"postgresql-9.3-1101-
jdbc4.jar","connection":"jdbc:postgresql:database","username":"user","password":"password1","preparedStatement
":"select ID as id from DATABASE;"}' http://localhost:8764/api/apollo/query-stages/instances
```

Response

```
{
  "type" : "jdbc-query-lookup",
  "id" : "jdbc-quer",
  "driver" : "postgresql-9.3-1101-jdbc4.jar",
  "connection" : "jdbc:postgresql:database",
  "username" : "user",
  "password" : "password1",
  "preparedStatement" : "select ID as id from DATABASE;",
  "fetchSize" : -1,
  "join" : true,
  "rows" : 10,
  "skip" : false,
  "label" : "jdbc-query-lookup",
  "type" : "jdbc-query-lookup"
}
```

Configuration

When using Fusion's REST-API, the ID of this stage is: `jdbc-query-lookup`.

Configuration Properties

Property	Description, Type
connection Connection URI *required*	The connection string for the database type: `string`

Property	Description, Type
driver JDBC Driver *required*	The fully qualified class name of the JDBC Driver to use. type: `string`
fetchSize Fetch Size	The JDBC Fetch Size to use. If -1, use the driver default. type: `integer` default value: ‘-1’
join Join with Request	If true, the results will be added on to the request using the prefix key and the row id, else the results will be put in the pipeline context using type: `boolean` default value: ‘true’
password Password *required*	The password to connect to the database. type: `string`
prefix Result Prefix Key	The string to use as a prefix for all values type: `string`
preparedStatement SQL Prepared Statement *required*	The SQL Prepared Statement to execute when bound with values. type: `string`
preparedStatementKeys Prepared Statement Keys	The keys in the Request/Header/Context to use to map request attributes into the prepared statement. These must map to the '?'s in your prepared statement. They must also be able to be resolved as the first parameter of that name in a request. type: `array` of `string`

Property	Description, Type
rows Rows	The number of rows to return type: `integer` default value: ‘10’
username Username *required*	The username to connect to the database. type: `string`

14.4.9. Landing Pages Stage

The Landing Pages query-pipeline stage provides the mechanism by which specific search queries will be pinned to certain URLs. This stage returns one or more URLs which can be used for redirection. It doesn't preform a redirection; this must be done by the calling application. The redirection URLs are returned in a separate section of the Fusion response object, with attribute name `fusion`.

This stage is configured using Landing Page Rules, which consist of the following:

- `keyword` - words, phrases, or a regex
- `mode` - filtering logic applied to the query, one of:
  - `exact` - the keyword and the query must match exactly. This is case-sensitive.
  - `phrase` phrase matching on the items in the keywords list
  - `regex` treat items in the keywords list as a regex
  - `match` must match every item in the keyword list, but doesn't require phrase matchin
- `url` - a list of URLs

Configuration

When using Fusion's REST-API, the ID of this stage is: `landing-pages`.

Configuration Properties

Property	Description, Type
matchCount Maximum matches	type: `integer` default value: '1' exclusiveMinimum: false minimum: 1
queryParam Query param for matching	type: `string` default value: 'q'

Property	Description, Type

rules
 	
Landing page rules
	type: `array` of `object`
	
	object attributes: \{
	`keyword` *(required)* : \{
	display name: Keyword
	type: `string`
	description : Search keywords to match on
	}
	`mode` *(required)* : \{
	display name: Match Strategy
	type: `string`
	default value: ‘exact’
	description : How to match the keywords
	enum: \{ exact phrase regex match }
	
	`} +`
	`` `url` _ (required)_ : \{ + ``
	`display name: Redirect URL +`
	`` type: `string` + ``
	`description : The URL to return for the redirect. For 'regex' match types, the URL can contain %s characters which will get populated by the regex capture groups +`
	`} +`
	`}`

14.4.10. Logging Stage

The Logging query-pipeline stage prints messages to the API log file; the default location is `fusion/var/log/api/api.log`.

The verbosity of this message is controlled by the property `detailed`. If true, then the current Request object will be pretty-printed to the log file. If false, only the basic information about this stage will be logged.

In a production environment logging stages should be configured with property `skip` set to `true`, if possible. Use of detailed logging may impact performance.

Configuration

When using Fusion's REST-API, the ID of this stage is: `query-logging`.

Configuration Properties

Property	Description, Type
detailed Detailed Logging	type: `boolean` default value: 'false'

14.4.11. Write Log Message Stage

This stage was renamed to Write Log Message in Fusion 3.0. In prior Fusion version, it was called the Log a Message stage.

The Write Log Message query-pipeline stage is an extension of Fusion's Logging Query Stage, which logs any message sent to the configured logging system using the Messaging Services.

Configuration

When using Fusion's REST-API, the ID of this stage is: `logging-message`.

Configuration Properties

Property	Description, Type
errorKey Message Response Failure Key	The name of the key to store a boolean if sending a message failed. If set, you can check the MessageResponse errorCode and other attributes for the reason. type: `string` default value: 'messageResponseFailure'
from From *required*	Who the message is from. May be a string template similar to the body and subject. type: `string`
logLevel Log Level	The Log Level. May be: debug, info, warn, error type: `string` enum: \{ debug info warn error }
messageBodyTemplate Message Body Template *required*	A Message Template that is used to create the message body to send. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation for details on the template language type: `string` default value: '`Processing Document '

Property	Description, Type
messageSubjectTemplate Message Subject Template *required*	A Message Template that is used to create the message subject to send. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation for details on the template language type: `string` default value: ‘Hello’
responseKey Message Response Context Key	The name of the key to store the MessageResponse under in the Pipeline Context. type: `string` default value: ‘messageResponse’
storeInContext Add to Pipeline Context	Put the generated Message later in the pipeline. type: `boolean` default value: ‘false’
to To *required*	Who to send the message to. May be a string template similar to the body and subject. type: `array` of `string`

14.4.12. Boost with Signals Stage

The Recommendation Boosting query-pipeline stage uses aggregated signals to selectively boost items in the set of search results. Recommendation boosting implements *collaborative filtering*. Signals contain relevance feedback, that is, information about what users do. For example, they might contain information about clicks or page views. Boosting occurs when the same *query* comes in again.

See Recommendations for more information.

Tip	This stage accesses the signals_aggr collection. Before using it, verify that the following permission is set: `GET:/solr/<collection-name>_signals_aggr/select`

Configuration

When using Fusion's REST-API, the ID of this stage is: `recommendation`.

Configuration Properties

Property	Description, Type
aggrType Aggregation Type	type: `string` default value: '*'
boostId Solr field to boost on	Which Solr field to use when applying recommendation boosts type: `string` default value: 'id'
numRecommendations Number of Recommendations	type: `integer` default value: '10'
numSignals Number of signals	Number of signals to process when getting recommended items type: `integer` default value: '100'

14.4.13. Return Query Parameters Stage

This stage was renamed to Return Query Parameters in Fusion 3.0. In prior Fusion versions, it was called the Return Query Params stage.

The Return Query Parameters query-pipeline stage is used to return query parameters as a pipeline response. The stage takes no additional properties beyond id, type, label, skip, and condition.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `return-queryparams`.

Configuration Properties

Property	Description, Type

14.4.14. Rollup Aggregation Stage

The Rollup Aggregation query-pipeline stage (previously called the Rollup Aggregator stage) is used to rollup Solr results from a context variable. If using the REST API, this stage type is named "rollup-rec-aggr".

This stage reads the Solr results (SolrResponse.class) from the context and rolls up over a single field product a list of unique IDs and also aggregates the weights (any numeric filed in Solr) for those IDs using any of the statistical aggregation functions available. The result from aggregation is saved back in the context and can later be used in the Parameterized Boosting stage.

Configuration

When using Fusion's REST-API, the ID of this stage is: `rollup-rec-aggr`.

Configuration Properties

Property	Description, Type
excludeResultsKey Results to Exclude	The key containing a set of results to exclude from this rollup type: `string`
key Key *required*	The key name to use from context to read Solr results. type: `string`
maxRows Max rows	The maximum number of results to return type: `integer` default value: '10'
resultKey Result Key *required*	The key name to which the results should be saved type: `string`
rollupField Rollup Field *required*	The field to rollup on type: `string`

Property	Description, Type
sort Sort results	If enabled, the output is sorted based on weight field if it is not null type: `boolean` default value: ‘true’
weightField Weight Field	The numerical field to consider as weight type: `string`
weightFunction Weight Arithmetic Function	The arithmetic function to use for weight fields on documents with same rollup field type: `string` default value: ‘sum’ enum: \{ sum mean max min stddev variance geoMean sumOfSquares sumOfLogs }

14.4.15. Query Fields Stage

This stage was renamed to Query Fields in Fusion 3.0. It was called the Search Fields stage in previous Fusion versions.

The Query Fields query-pipeline stage defines common Solr query parameters for the edismax query parser. An alternative to this stage is the Set Query Params stage.

Configuration

When using Fusion's REST-API, the ID of this stage is: `search-fields`.

Configuration Properties

Property	Description, Type
minimumMatch Minimum Should Match	type: `string`
queryFields Search Fields	type: `array` of `object` object attributes: \{ `boost` : \{ display name: Field Boost type: `number` } `field` : \{ display name: Field Name type: `string` } }
returnFields Return Fields	type: `array` of `string`
rows Number of Results	type: `integer` default value: '10'
sort Results Sort Order	type: `string`
start Result Offset	type: `integer` default value: '0'

14.4.16. Security Trimming Stage

The Security Trimming query-pipeline stage restricts query results according to the user ID. While indexing the content, the Fusion connectors service stores security ACL metadata associated with the crawled items and indexes them as fields. The Security Trimming stage matches this information against the ID of the user running the search query.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `security-trimming`.

Configuration Properties

Property	Description, Type
overrideUserIdentityHandling Override default user identity handling?	Default handling first attempts to take the user identity from a 'fusion-user-id' http-header, which is the logged-in user ID from the Fusion proxy service. If that value is empty, a 'username' query parameter is tried instead. When this DataSource property is enabled, the specified source and key properties are used explicitly, without any fallback behavior. type: `boolean` default value: ‘false’
userIdentityKey User ID key	eg. username or userID etc.. type: `string` default value: ‘username’
userIdentitySource User ID source	Specify whether the value comes from an http header or query parameter. type: `string` default value: ‘query_param’ enum: \{ query_param header }

14.4.17. Send SMTP Email Stage

This stage was renamed to Send SMTP Email in Fusion 3.0; it was called the Send email (via SMTP) stage in previous Fusion versions.

This stage sends an SMTP message from Fusion, for alerting, reporting, and more, using Fusion’s Messaging Services.

Enabling Email Messaging

Before you can use the Email pipeline stage, you must enable Email messaging in Fusion:

1. Click **Applications** > **System** > **Messaging Services**.
2. Select **SMTP Message Service** from the drop-down menu.
3. Verify that the default settings are sufficient.
4. Click **Save message service**.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `smtp-message`.

Configuration Properties

Property	Description, Type
errorKey Message Response Failure Key	The name of the key to store a boolean if sending a message failed. If set, you can check the MessageResponse errorCode and other attributes for the reason. type: `string` default value: ‘messageResponseFailure’
from From *required*	Who the message is from. May be a string template similar to the body and subject. type: `string`
messageBodyTemplate Message Body Template *required*	A Message Template that is used to create the message body to send. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation for details on the template language type: `string` default value: '`Processing Document ’

Property	Description, Type
messageSubjectTemplate Message Subject Template *required*	A Message Template that is used to create the message subject to send. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation for details on the template language type: `string` default value: 'Hello'
responseKey Message Response Context Key	The name of the key to store the MessageResponse under in the Pipeline Context. type: `string` default value: 'messageResponse'
smtpPassword SMTP Password *required*	The SMTP password for the user credentials type: `string`
smtpUser SMTP Username *required*	The SMTP user to send the message from type: `string`
storeInContext Add to Pipeline Context	Put the generated Message later in the pipeline. type: `boolean` default value: 'false'
to To *required*	Who to send the message to. May be a string template similar to the body and subject. type: `array` of `string`

14.4.18. Send PagerDuty Message Stage

This stage sends a PagerDuty Message from Fusion, for alerting, monitoring, and more, using Fusion's Messaging Services.

Read more about PagerDuty integration in Fusion on our blog.

Enabling PagerDuty Messaging

Before you can use the PagerDuty pipeline stage, you must enable PagerDuty messaging in Fusion:

1. Click **Applications** > **System** > **Messaging Services**.
2. Select **PagerDuty Message Service** from the drop-down menu.
3. Enter the following information:
   - PagerDuty service key
   - PagerDuty Service API URL; this should be `https://events.pagerduty.com/generic/2010-04-15/create_event.json`.
4. Click **Save message service**.

Configuration

When using Fusion's REST-API, the ID of this stage is: `pagerduty-message`.

Configuration Properties

Property	Description, Type
client Client	The name of the monitoring client that is triggering this event. type: `string` default value: 'Fusion' minLength: 1
clientURL Client URL	The URL of the monitoring client that is triggering this event. type: `string` default value: 'fusion-monitoring.yourdomain.com' minLength: 1

Property	Description, Type
errorKey Message Response Failure Key	The name of the key to store a boolean if sending a message failed. If set, you can check the MessageResponse errorCode and other attributes for the reason. type: `string` default value: ‘messageResponseFailure’
eventType Event Type *required*	The Pager Duty Event Type. One of: trigger, acknowledge, resolve type: `string` default value: ‘trigger’ enum: \{ trigger acknowledge resolve }
incidentContextImages Incident Context Images	type: `array` of `object` object attributes: \{ `alt` : \{ display name: Alternate Text type: `string` description : HTM 'alt' tag for the image } `href` *(required)* : \{ display name: Target Link type: `string` description : URL to open when clicked on the image } `src` *(required)* : \{ display name: Source type: `string` description : HTML 'src' tag for the image. Should be always secure connection (https://) } }

Property	Description, Type
incidentContextLinks Incident Context Links	type: `array` of `object` object attributes: \{ `href` *(required)* : \{ display name: Target Link type: `string` description : URL to open when clicked on the link } `text` : \{ display name: Text type: `string` description : Arbitrary text explaining the URL } }
incidentDescription Description *required*	A short description of the problem that led to this trigger. This field (or a truncated version) will be used when generating phone calls, SMS messages and alert emails. It will also appear on the incidents tables in the PagerDuty UI. The maximum length is 1024 characters. type: `string` default value: ‘Sample Description’ maxLength: 1024 minLength: 1
incidentDetails Incident Details	type: `array` of `object` object attributes: \{ `name` *(required)* : \{ display name: Name type: `string` } `value` *(required)* : \{ display name: Value type: `string` } }

Property	Description, Type
incidentKey Incident Key	Identifies the incident to which this trigger event should be applied. If there’s no open (i.e. unresolved) incident with this key, a new one will be created. If there’s already an open incident with a matching key, this event will be appended to that incident’s log. type: `string` default value: '`Incident ’ minLength: 1
responseKey Message Response Context Key	The name of the key to store the MessageResponse under in the Pipeline Context. type: `string` default value: ‘messageResponse’
storeInContext Add to Pipeline Context	Put the generated Message later in the pipeline. type: `boolean` default value: ‘false’

14.4.19. Send Slack Message Stage

This stage sends a Slack message from Fusion, for alerting, reporting, and more, using Fusion's Messaging Services.

Enabling Slack Messaging

Before you can use the Slack pipeline stage, you must enable Slack messaging in Fusion:

1. Click **Applications** > **System** > **Messaging Services**.
2. Select **Slack Message Service** from the drop-down menu.
3. Enter the following information:
   - Slack auth token
   - Message template

     The default is `<subject> : <body>`, which are configured with `messageSubjectTemplate` and `messageBodyTemplate` below. See Messaging Services Templates for details on the template language.
   - Optionally, you can configure a proxy or the error reporting channel name.
4. Click **Save message service**.

Configuration

When using Fusion's REST-API, the ID of this stage is: `slack-message`.

Configuration Properties

Property	Description, Type
errorKey Message Response Failure Key	The name of the key to store a boolean if sending a message failed. If set, you can check the MessageResponse errorCode and other attributes for the reason. type: `string` default value: 'messageResponseFailure'
from From *required*	Who the message is from. May be a string template similar to the body and subject. type: `string`

Property	Description, Type
messageBodyTemplate Message Body Template *required*	A Message Template that is used to create the message body to send. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation for details on the template language type: `string` default value: '`Processing Document ’
messageSubjectTemplate Message Subject Template *required*	A Message Template that is used to create the message subject to send. See https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation for details on the template language type: `string` default value: ‘Hello’
responseKey Message Response Context Key	The name of the key to store the MessageResponse under in the Pipeline Context. type: `string` default value: ‘messageResponse’
storeInContext Add to Pipeline Context	Put the generated Message later in the pipeline. type: `boolean` default value: ‘false’
to To *required*	Who to send the message to. May be a string template similar to the body and subject. type: `array` of `string`

14.4.20. Additional Query Parameters Stage

This stage was renamed to Additional Query Parameters in Fusion 3.0. It was called the Set Query Params stage in previous versions.

The Additional Query Parameters query-pipeline stage is used to set, append, and remove Solr query parameters. This stage takes a list of query parameters. Each parameter is specified as a triple consisting of parameter name, parameter value, and update policy.

Available update policies are: `replace`, `append`, `remove` and `default`. The policy 'default' means that this parameter will be added only if it has not yet been set via the request or by a default specification in the Solr config file solrconfig.xml.

Example Stage Specification

Set the response format to JSON:

```
{ "type":"set-params",
   "id":"default-params",
   "params": [
      {"key":"wt", "value":"json", "policy":"default"}
     ],
   "skip":false
}
```

Return a field named "id", and 10 rows of results:

```
{  "type":"set-params",
   "id":"default-params",
   "params": [
      {"key":"fl", "value":"id", "policy":"append"},
      {"key":"rows", "value":"10", "policy":"replace"}
    ],
   "skip":false
}
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `set-params`.

Configuration Properties

Property	Description, Type
params Parameters and Values	type: `array` of `object` object attributes: \{ `key` *(required)* : \{ display name: Parameter Name type: `string` } `policy` : \{ display name: Update Policy type: `string` default value: ‘append’ enum: \{ replace append remove default } } + `value` : \{ + display name: Parameter Value + type: `string` + } + }

14.4.21. Solr Query Stage

This stage was renamed to Solr Query in Fusion 3.1. It was called the Query Solr stage in prior Fusion versions.

The Solr Query query-pipeline stage transforms the Fusion query pipeline Request object into a Solr query and sends it to Solr.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `solr-query`.

Configuration Properties

Property	Description, Type
allowFederatedSearch Allow Federated Search	Allow Solr 'collection' and 'shards' parameters. type: `boolean` default value: ‘false’
allowedRequestHandlers Configure request handlers allowed for queries	type: `array` of `string` minimum number of items (`minItems`): 0
httpMethod HTTP Method	HTTP method for querying Solr type: `string` default value: ‘POST’ enum: \{ POST GET }

14.4.22. Retrieve Stored Parameters Stage

This stage was renamed to Retrieve Stored Parameters in Fusion 3.0. It was called the Stored Parameters stage in prior Fusion versions.

The Retrieve Stored Parameters stage is used to add parameters to the downstream Solr query stage in the pipeline. The primary use case for this query stage is in eCommerce applications to organize and enhance search query results, where the parameters added to the Solr query are used for faceting. This stage is used in query pipelines for applications where the documents in the collection are to be classified according to a known category hierarchy that is stored in Fusion as an auxiliary "stored parameters" collection.

The information in a taxonomy is meta-information about the categories used to classify a set of things. For an eCommerce site, the set of things are items in the product catalog, which are stored in a Fusion collection, referred to as the primary collection. The taxonomy itself is specified as a set of categories which are stored in an auxiliary collection and naming conventions relate the primary and auxiliary collections so that for a primary collection named "COLL", the auxiliary taxonomy is stored in a collection named "COLL_stored_parameters".

Configuration

When using Fusion’s REST-API, the ID of this stage is: `stored-parameters`.

Configuration Properties

Property	Description, Type
inherit Inherit Parameters From Parent?	Should a child category inherit its parents' stored parameters? type: `boolean` default value: ‘false’
key Key	The parameter name from the context/query whose value will be used to query the stored parameters collection. type: `string`

14.4.23. Solr SubQuery Stage

This stage was renamed to Solr SubQuery in Fusion 3.0. It was called the Sub Query stage in prior Fusion versions.

The Solr SubQuery query-pipeline stage is used to query external collections. The Solr SubQuery stage runs a query and adds the results to the pipeline Request object. The query must return results in JSON format, so that they can be added to the query pipeline.

Configuration

When using Fusion’s REST-API, the ID of this stage is: `sub-query`.

Configuration Properties

Property	Description, Type
collection Collection *required*	type: `string`
handler Request handler *required*	type: `string` default value: ‘select’
headers Headers	type: `array` of `object` object attributes: \{ `key` *(required)* : \{ display name: Parameter Name type: `string` } `value` : \{ display name: Parameter Value type: `string` } }
key Key *required*	The key used to join these results onto the pipeline context type: `string` default value: ‘subquery-results’

Property	Description, Type
method HTTP method *required*	type: `string` default value: ‘GET’ enum: \{ GET POST }
params Query params *required*	type: `array` of `object` object attributes: \{ `key` *(required)* : \{ display name: Parameter Name type: `string` } `value` : \{ display name: Parameter Value type: `string` } }
parentParams Query parameters from the parent query	type: `array` of `string`
rollupKeys Rollup results to include	type: `array` of `string`

14.4.24. User Recommendation Boosting Stage

This stage was removed in the 3.0.1 release.

The User Recommendation Boosting query-pipeline stage allows user recommendations to boost specified results according to weighted criteria at query time. See Recommendations for more information.

Tip	This stage accesses the signals_aggr collection. Before using it, verify that the following permission is set:

```
GET:/solr/COLL_signals_aggr/select
```

Configuration

When using Fusion’s REST-API, the ID of this stage is: `user-recommendation`.

Configuration Properties

Property	Description, Type
aggrType Aggregation Type	type: `string` default value: ‘*’
collection Aggregation Collection	if left blank, we will use the default aggregation collection for the collection being queried. type: `string`
maxPositionalWeight Maximum weight	The weight of the first item in the recommended item list. Subsequent items will have decreasing weight. type: `number` default value: ‘10.0’
minPositionalWeight Minimum weight	The weight of the last item in the recommended item list. type: `number` default value: ‘1.0’
numRecommendations Number of Recommendations	type: `integer` default value: ‘10’

Property	Description, Type
resultKey Result Key	The key name to use to save the raw recommendation results. This should correspond to the 'key' value of a Rollup Aggregator. type: `string` default value: ‘results’

Chapter 15. REST API Reference

Fusion API service are designed to be accessed via Fusion's authentication proxy module which is part of the Fusion UI service (default port 8764). All applications should use this method to access the API service. However, for development purposes it is possible to access the API service directly:

Development	Production
<code><a href="http://&lt;fusion-host&gt;:8765/api/v1/&lt;endpoint&gt" class="bare">http://&lt;fusion-host&gt;:8765/api/v1/&lt;endpoint&gt</a>;</code> Direct access to the API service, bypassing the authentication proxy. *Access to this port should be restricted.*	<code><a href="http://&lt;fusion-host&gt;:8764/api/apollo/&lt;endpoint&gt" class="bare">http://&lt;fusion-host&gt;:8764/api/apollo/&lt;endpoint&gt</a>;</code> Access via the authentication proxy, using either a username/password pair or a session cookie with the user ID.

Important	In a production environment, do not expose port 8765 to users. Using your firewall software or the Jetty configuration of the API server, make it accessible only to the auth proxy service and the connectors service.

15.1. Listing all Fusion component services

The Fusion `introspect` endpoint lists basic information about endpoints and parameters for all Fusion endpoints, including the Connectors services endpoints:

```
curl -u user:pass http://localhost:8764/api/apollo/introspect
```

15.2. Fusion Components REST API Reference Pages

- Blob Store API
- Collection Features API
- Collections API
- Configurations API
- Connectors APIs
- History API
- Index Pipelines API
- Index Profiles API
- Index Stages API
- Messaging API
- Nodes API
- Parsers API

- Query Pipelines API
- Query Profiles API
- Query Stages API
- Realms API
- Recommendations API
- Reporting API
- Roles API
- Scheduler API
- Search Cluster API
- Sessions API
- Signals API
- Signals Aggregator API
- Solr API
- SolrAdmin API
- Stopwords API
- Synonyms API
- System API
- Usage API
- User API
- ZooKeeper Import/Export API

15.3. Fusion Services

Fusion services are registered with the Apache ZooKeeper.

15.3.1. Summary of Services

Aggregator

The aggregator service handles signal aggregation. Aggregated signal events are indexed into one of two system-level collections: *collection_signals-aggr* and *collection_signals*. The primary function of this service is to provide faster querying for recommendations.

More information about this service is in the Signals section.

Configurations

The configurations service handles system configuration settings, which can be accessed via the UI or a REST API. Some of the information provided by this service comes from Fusion configuration files, and some is read from the operating system.

More information about this service is in the section Configurations API.

Connectors

The connectors service handles crawling of various types of content repositories. Connectors are general-purpose programs used to fetch data, and the connector service contains these programs.

More information about this service is in the section Connectors.

dynamicSchema

The dynamicSchema service enables Solr's managed schema functionality, which allows programmatic access to Solr's schema.xml file, and also enables schemaless mode.

More information about this service is in the section Collection Features.

Features

The features service handles each of the collection features currently supported in Fusion, including dynamicSchema, searchLogs, and signals.

More information about this service is in the section Collection Features.

History

The history services handles history data for each service. Most commonly, it logs when each service stops and starts, but in some cases it records more information. The scheduler history, for example, records the result of each job the scheduler runs.

More information about this service is in the section History API.

Index-pipelines

The index-pipelines service handles the definition and execution of the index pipelines, which is an extract-transform-load system to prepare and process documents into the index.

More information about this service is in the section Index Pipelines.

Index-profiles

The index-profiles stage handles the creation of search profiles to associate index pipelines with specific collections, and it allows a single index pipeline to be shared among multiple collections.

More information about this service is in the section Index Profiles.

Index-stages

The index-stages service handles the definition of the index stages that form index pipelines.

More information about this service is in the section Index Pipelines.

Introspect

The introspect service provides schema-like investigation into the REST API endpoints supported by each service. It shows supported methods, as well as any applicable paths and query parameters.

More information about this service is in the section REST API Reference.

Nodes

The nodes service reports information about cluster nodes configured with Fusion.

More information about this service is in the section Nodes API.

Query-pipelines

The query-pipelines service handles the definition and execution of query pipelines, which transform user queries and processes the responses from Solr.

More information about this service is in the section Query Pipelines.

Query-profiles

The query-profiles stage handles creating search profiles to associate query pipelines with specific collections.

More information about this service is in the section Query Profiles.

Query-stages

The query-stages service handles the definition of the query stages that form query pipelines.

More information about this service is in the section Query Pipelines.

Realm-configs

The realm-configs service handles the definition of authentication realms, or the repositories that contain data about users who authenticate to the system.

More information about this service is in the section Security Realms.

Recommend

The recommend service handles the recommendations provided by Fusion. Recommendations are search results that are derived from previous user behavior, much of which was captured by the aggregator service and stored as signals.

More information about this service is in the section Recommendations.

Reports

The reports service handles analytic reports provided by Fusion, which includes: lessThanN, topQueries, topN, histo, and dateHisto.

More information about this service is in the section Search Query Reporting.

Resources

The resources service provides information to the UI on available and enabled services.

Roles

The roles service handles role definition, which provides a framework to control user access to any Fusion service. This includes access to the services via the UI or with the REST API.

More information about this service is in the section Roles.

Scheduler

The scheduler service handles recurring events in the Fusion system. These events are designated by the user to occur at specific start times and to repeat for defined intervals, and multiple instances of the scheduler service can run accross different nodes.

More information about this service is in the section Schedules.

Schema

The schema service allows editing fields and dynamic fields defined in Solr’s schema.xml file.

searchCluster

The searchCluster service handles information about Solr clusters that are accessible to the Fusion system. This may include an existing SolrCloud implementation or standalone Solr instance.

More information about this service is in the section Search Clusters.

searchLogs

The searchLogs service allows search logs to be indexed for later use by the reporting service.

More information about this service is in Collection Features.

Signals

The signals service handles search events that can later be used for recommendations.

More information about this service is in the section Signals.

Solr

The solr service handles collection-level requests to Solr, providing a way to allow only authorized users direct access to your Solr installations.

More information about this service is in Solr API.

solrAdmin

The solr service handles admin-level requests to Solr, providing a way to allow only authorized users direct access to your Solr installations.

More information about this service is in SolrAdmin API.

Stopwords

The stopwords service allows managing a stop word list for a collection.

More information about this service is in the section Stopwords API.

Synonyms

The synonyms service allows managing a synonym list for a collection.

More information about this service is in the section Synonyms API.

System

The system service provides analytics on the performance of your Solr installation, including threads in use, buffers, and other metrics.

More information about this service is in the section System API.

Usage

The usage service provides data for the System Usage Monitor, a voluntary program to send anonymous data about your system to Fusion.

More information about this service is in the section System Usage Monitor.

Users

The users service handles user records, passwords, and authentication.

More information about this service is in the section Users.

15.4. Authentication and Authorization APIs

The Fusion access control component handles user authentication and authorization. It runs in the same process as the Fusion UI and provides the following sets of endpoints:

- User API: create, update, delete and list user records in Fusion.
- Roles API: create, update, delete and list roles which grant users different levels of permissions in the system.
- Realms API: create, update, delete and list realms.
- Sessions API: create an authenticated session and session cookie.

15.5. Blob Store API

The Blob Store REST API allows storing binary objects in Solr. The primary use case for this is to store entity extraction models, lookup lists or exclusion lists for use in index pipelines. This may include the entity extraction models and lookup lists included with Fusion in the `fusion/data/nlp` directory, or files that you have created on your own.

Blobs uploaded to Solr with this REST API are stored in the 'system_blobs' collection.

15.5.1. Get, Upload, Update or Delete a Blob

The path for this request is:

`/api/apollo/blobs`

- GET - list all blobs.
- PUT - upload a new blob with your defined ID, or update an existing blob. A PUT request will allow you to define the name of the blob while uploading it to Solr, so this is the preferred method for adding blobs.
- POST - uploaded a new blob, defined ID is included in request body.
- DELETE request will remove this blob.

15.5.2. List a Blob Manifest

The path for this request is:

`/api/apollo/blobs/<id>/manifest`

where <*id*> is the name of a specific blob.

- GET - returns the manifest this item in the blob store. Output list the blog name (which is the blob ID), the content-type, size, when it was last modified, the document version in Solr, and an md5 hash for the document.

15.5.3. Examples

Upload a file to the blob store:

REQUEST

```
curl -u user:pass -X PUT --data-binary @airports.lst -H 'Content-type: text/plain'
http://localhost:8764/api/apollo/blobs/airports.lst
```

RESPONSE

```
{
  "name" : "airports.lst",
  "contentType" : "text/plain",
  "size" : 66,
  "modifiedTime" : "2014-12-03T22:26:16.436Z",
  "version" : 0,
  "md5" : "fbe581898cb426f6bdcabc3f52253594"
}
```

Upload an OpenNLP sentence model binary file to the blob store:

REQUEST

```
curl -u user:pass -X PUT --data-binary @data/nlp/models/en-sent.bin -H 'Content-type: application/octet-stream' http://localhost:8764/api/apollo/blobs/sentenceModel.bin
```

Note	In this example that we have changed the name of the blob during upload by giving it a different ID. The file is named 'en-sent.bin' but we have defined the ID of this to 'sentenceModel.bin'. When we use this blob in an index pipeline, we would refer to it by the ID we have given it.

Get the manifest for a sentence OpenNLP model we've previously saved in the blob store:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/blobs/sentenceModel.bin/manifest
```

RESPONSE

```
{
  "name" : "sentenceModel.bin",
  "contentType" : "application/octet-stream",
  "size" : 98533,
  "modifiedTime" : "2014-09-08T18:50:07.559Z",
  "version" : 1478704189996531712,
  "md5" : "3822c5f82cb4ba139284631d2f6b7fde"
}
```

Upload a JDBC driver, using slashes in the blob name:

REQUEST

```
curl -u user:pass -X PUT --data-binary @mysql-connector-java-5.1.42-bin.jar -H 'Content-length: 996444' -H 'Content-Type: application/zip' http://localhost:8764/api/apollo/blobs/good/to/go/mysql-connector-java-5.1.42-bin.jar?resourceType=driver:jdbc
```

RESPONSE

```
{
  "name" : "good/to/go/mysql-connector-java-5.1.42-bin.jar",
  "contentType" : "application/zip",
  "size" : 996444,
  "modifiedTime" : "2017-04-04T15:58:32.856Z",
  "version" : 0,
  "md5" : "b1946ac92492d2347c6235b4d2611184",
  "metadata" : {
    "subtype" : "driver:jdbc",
    "resourceType" : "driver:jdbc"
  }
}
```

Get the JDBC driver that was uploaded:

REQUEST

```
curl -u user:pass -H "Accept: application/zip" http://localhost:8764/api/apollo/blobs/jtds-1.3.1-
src.zip?resourceType=driver:jdbc  -o jtds-1.3.1-src.zip
```

RESPONSE

```
[ {
  "name" : "good/to/go/sentenceModel.bin",
  "contentType" : "application/octet-stream",
  "size" : 6,
  "modifiedTime" : "2017-04-04T06:21:53.465Z",
  "version" : 1563727666574524416,
  "md5" : "b1946ac92492d2347c6235b4d2611184",
  "metadata" : {
    "subtype" : "driver:jdbc",
    "resourceType" : "driver:jdbc"
  }
} ]
```

15.6. Collections API

The Collections API manages Fusion collections. It provides endpoints for creating, updating, and deleting collection, as well as endpoints for getting a collection’s status and usage statistics.

Fusion maintains internal system collections for logs, blobs, and metrics data which operate in conjunction with collections created by users. The Collections API is used to manage all Fusion collections.

15.6.1. Create, List, Update or Delete Collections

The path for this request is:

`/api/apollo/collections/<name>`

where <*name*> is the name of an specific collection.

- GET - returns properties for the specified collection or properties for all collections if no collection name is specified.
- PUT - create or update a collection
- DELETE - remove a collection. A DELETE request takes two parameters:
  - solr - if **true**, the collection will be deleted from Solr while it is deleted from Fusion. The default is **false**.
  - purge - if **true**, then associated signals and/or searchLogs collections will also be deleted. The default is **false**.

Collections properties

The body of a PUT request contains the following properties, specified in JSON:

Property	Description
searchClusterId *Optional*	The name of the search cluster to associate the collection with. If this is not defined, "default" will be used, which is the name of the LucidWorks cluster at installation. See also the section on Search Clusters for more information.
solrParams	A set of name:value pairs corresponding to the properties avaliable on a Solr collection. See Solr’s collection API documentation for a list of valid parameters.
type *Optional*	The type of collection. The type must be one of the following: • DATA • LOGS • METRICS • SIGNALS • SEARCHLOGS If not defined, the type will default to "DATA", which is fine for most cases.

Property	Description
metadata	This is an internal property, and will be removed in the future.

15.6.2. Get Collection Status or Statistics

A GET request to endpoint:

`/api/apollo/collections/<name>/status`

returns a listing of state of each shard and all of its replicas, including the core name, its status as leader, the base URL and the node name.

A GET request to endpoint:

`/api/apollo/collections/<name>/stats`

returns a more detailed listing, including the number of documents in the index, how many requests have been made of the collection, the queries-per-second (qps) and the index size in bytes.

15.6.3. Examples

Create a new collection called 'newCollection', with appropriate SolrCloud environment settings:

REQUEST

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d
'{"solrParams":{"replicationFactor":1,"numShards":1}}'
http://localhost:8764/api/apollo/collections/newCollection
```

RESPONSE

```
{
  "id" : "newCollection",
  "createdAt" : "2014-09-19T18:46:52.954Z",
  "searchClusterId" : "default",
  "solrParams" : {
    "name" : "newCollection",
    "numShards" : 1,
    "replicationFactor" : 1
  },
  "type" : "DATA",
  "metadata" : { }
}
```

Create a collection named 'local-collection1' that refers to 'collection1' in a pre-existing SolrCloud cluster named 'Solr4.10' (see also the section Search Clusters):

REQUEST

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d '{"id":"local-collection1",
"searchClusterId":"Solr4.10", "solrParams":{"name":"collection1"}}'
http://localhost:8764/api/apollo/collections
```

RESPONSE

```
{
  "id" : "local-collection1",
  "createdAt" : "2014-09-19T18:48:45.396Z",
  "searchClusterId" : "Solr4.10",
  "solrParams" : {
    "name" : "collection1"
  },
  "type" : "DATA",
  "metadata" : { }
}
```

Delete a collection, but keep the associated signals and searchLogs collections:

REQUEST

```
curl -u user:pass -X DELETE http://localhost:8764/api/apollo/collections/newCollection?purge=false
```

Get the status of the 'demo' collection:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/collections/demo/status
```

RESPONSE

```
{
  "maxShardsPerNode" : 1,
  "replicationFactor" : 1,
  "shards" : {
    "shard1" : {
      "range" : "80000000-7fffffff",
      "state" : "active",
      "replicas" : {
        "core_node1" : {
          "state" : "active",
          "core" : "demo_shard1_replica1",
          "leader" : true,
          "base_url" : "http://localhost:8983/solr",
          "node_name" : "localhost:8983_solr"
        }
      }
    }
  }
}
```

Get stats for the 'demo' collection:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/collections/demo/stats
```

RESPONSE

```
{
  "collectionId" : "demo",
  "documentCount" : 536,
  "requestCount" : 6,
  "qps" : 28.34716542561849,
  "sizeInBytes" : 7646045,
  "lastModified" : "2014-05-19T19:58:33.545Z"
}
```

15.7. Collection Features API

The Collection Features API allows the following settings to be specified for a collection:

Property	Description
dynamicSchema	Modifies the Solr schema to be "managed", which means it’s possible for Fusion to use Solr’s schema API to manage the schema. It also sets Solr to operate in 'schemaless' mode, which means fields do not need to be pre-defined in the schema for them to be added to Solr’s index. Note that this applies to the Solr included with Fusion, and does not modify an existing Solr cluster, if you have one already.
searchLogs	Creates a parallel collection for the storage of log data which is used to generate search query reports.
signals	Creates a parallel collection for the storage of signals data (such as user clicks, or ratings). Signals will need to be indexed and aggregated in order to be used. See the section on Signals for more information.
partitionByTime	Partition the corresponding Solr collection by time; see Time Series Indexing.

15.7.1. Get a List of Collection Feature Properties

The path for this request is:

`/api/apollo/collections/<collection>/features/<featureName>`

where <*collection*> is the name of an existing collection and <*featureName*> is the name of a specific feature from the table above.

A GET request will return the current properties for that feature. If no feature name is specified, a GET request will return all features for that collection.

Collection Feature Properties

Property	Description
featureName	The name of the feature, from the table above.
collectionId	The collection name.
params	Parameters of the feature.
enabled	Whether the feature is enabled or disabled. In a new collection, all features are disabled.

15.7.2. Update a Specified Feature

A PUT request to endpoint:

`/api/apollo/collections/<collection>/features/<featureName>`

updates the setting for the specified collection feature. The PUT request body consists of either `{ "enabled": true }` or `{ "enabled" : false }`. A PUT request with no body is equivalent to `{ "enabled" : false }`.

15.7.3. Examples

List the status of the features for the "demo" collection:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/collections/demo/features
```

RESPONSE

```
 [ {
  "name" : "dynamicSchema",
  "collectionId" : "demo",
  "params" : { },
  "enabled" : false
}, {
  "name" : "searchLogs",
  "collectionId" : "demo",
  "params" : { },
  "enabled" : false
}, {
  "name" : "signals",
  "collectionId" : "demo",
  "params" : { },
  "enabled" : false
} ]
```

Enable signals for a collection named 'demo':

REQUEST

```
curl -u user:pass -X PUT -H Content-type:application/json -d '{"enabled":true}'
http://localhost:8764/api/apollo/collections/demo/features/signals
```

15.8. Configurations API

The Configurations API allows setting global properties for Fusion. Some settings are not set by any configuration file but are reported as settings from the operating system. Those settings cannot be changed with this API.

15.8.1. Get, Create or Update Configuration Items

The path for this request is:

`/api/apollo/configurations/<key>`

where <*key*> is the name of a configuration item.

A POST will create a new configuration item, and a PUT will update an existing configuration item.

A GET request will list matching configuration items or all configuration items if no key is specified. When no key is specified, a GET request takes the following query parameters:

Parameter	Description
pattern	The pattern allows finding all configuration items that contain the specified string.
prefix	The prefix allows finding all configuration items that start with the specified string.
verbose	If **true**, the response will include what part of the Fusion system (or the server's operating system) has set the configuration item. The default is **false**, which means the output will include only the name of the configuration item and its value.

When adding new configuration items or updating existing items, pass them in a JSON object, meaning they should be enclosed in curly braces (`{}`). The output includes the configuration items that match the query parameters and their values. If verbose is enabled, the "location" property will also be returned.

15.8.2. Examples

Show the configuration items that include the pattern 'zk-connect', with verbose enabled:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/configurations?verbose=true&pattern=zk-connect
```

RESPONSE

```
{
  "com.lucidworks.apollo-admin.config/zk-connect" : [ {
    "value" : "localhost:9983",
    "location" : "system properties"
  } ]
}
```

Get the configuration items from the Connectors JVM that include the term 'connect':

REQUEST

```
curl -u user:pass http://localhost:8984/connectors/api/v1/configurations?pattern=connect
```

Get the configuration items from the Fusion JVM that start with 'com.lucidworks':

REQUEST

```
http://localhost:8764/api/apollo/configurations?prefix=com.lucidworks
```

Change the default allowed recommendation types to include 'itemsForItem'':

REQUEST

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d '{"itemsForQuery", "termsForDocument",
"itemsforItem"}'
http://localhost:8764/api/apollo/configurations/com.lucidworks.apollo.service.recommend.allowed.types
```

RESPONSE

None. Check the setting again to confirm the changes.

15.9. Connector APIs

Connectors run as a standalone module. Requests for connector services are routed through the Lucidworks proxy layer, which manages the authorizations for users. This means that all connector requests must contain authentication credentials, and access to the connector REST APIs can be limited to certain users.

There are several REST APIs for Connectors, listed here:

- Connector Datasources API
- Connector History API
- Connector JDBC API
- Connector Jobs API
- Connector Plugins API
- Connectors Crawl Database API
- Connector Status API

15.10. Connector Datasources API

The connector datasources API is used to create and configure datasources. See Connectors and Datasources for general information, Connectors and Datasource Reference for details on configuring specific datasources.

15.10.1. List a Datasource or All Datasources

The path for this request is:

`/api/apollo/connectors/datasources?collection=<collectionName>`

A GET request will return the definitions for all datasources for that collection. If the collection parameter is omitted, all datasource definitions will be returned.

Input

None.

Output

The output will include all datasources that match the request, with all properties.

Examples

Note	Use port 8765 in local development environments only. In production, use port 8764.

Get datasources assigned to the "demo" collection:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/connectors/datasources?collection=demo
```

RESPONSE

```
[ {
  "id" : "database",
  "created" : "2014-05-04T19:47:22.867Z",
  "modified" : "2014-05-04T19:47:22.867Z",
  "connector" : "lucid.jdbc",
  "type" : "jdbc",
  "description" : null,
  "pipeline" : "conn_solr",
  "properties" : {
    "db" : null,
    "commit_on_finish" : true,
    "verify_access" : true,
    "sql_select_statement" : "select CONTENTID as id from CONTENT;",
    "debug" : false,
    "collection" : "demo",
    "password" : "password",
    "url" : "jdbc:postgresql://localhost:5432/db",
    "nested_queries" : null,
    "clean_in_full_import_mode" : true,
    "username" : "user",
    "delta_sql_query" : null,
    "commit_within" : 900000,
    "primary_key" : null,
    "driver" : "org.postgresql.Driver",
    "max_docs" : -1
  }
} ]
```

15.10.2. Create or Update a Datasource

A POST request will create a new datasource. The path for this request is:

`/api/apollo/connectors/datasources`

A PUT request will update an existing datasource. The path for this request is:

`/api/apollo/connectors/datasources/<id>`

where <*id*> is the name of the datasource to be updated.

Input

Property	Description
id *Required*	A unique identifier for this datasource. When creating a datasource, this property is required.
connector *Required*	The connector that will be used for this datasource.
type *Required*	The type of datasource, which must be supported by the defined connector.
description *Optional*	A description of the datasource.

Property	**Description**
pipeline *Optional*	The index pipeline that should be used for this datasource.
properties *Required*	The properties for the datasource. The available properties will depend on the connector and type chosen. Not all available properties are required, but all datasources have at least one property that must be set to create the datasource.

Output

A successful request returns the datasource definition. A failed request returns a JSON-formatted error message.

Examples

Create a datasource to index Solr-formatted XML files:

REQUEST

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d '{"id":"SolrXML",
"connector":"lucid.solrxml", "type":"solrxml", "properties":{"path":"/Applications/solr-
4.10.2/example/exampledocs", "generate_unique_key":false, "collection":"MyCollection"}}'
http://localhost:8764/api/apollo/connectors/datasources
```

RESPONSE

```
{
  "id" : "SolrXML",
  "created" : "2015-05-18T15:47:51.199Z",
  "modified" : "2015-05-18T15:47:51.199Z",
  "connector" : "lucid.solrxml",
  "type" : "solrxml",
  "properties" : {
    "commit_on_finish" : true,
    "verify_access" : true,
    "generate_unique_key" : false,
    "collection" : "MyCollection",
    "include_datasource_metadata" : true,
    "include_paths" : [ ".*\\.xml" ],
    "initial_mapping" : {
      "id" : "a35c9ff3-dbb6-434b-af40-597722c2986a",
      "skip" : false,
      "label" : "field-mapping",
      "type" : "field-mapping"
    },
    "path" : "/Applications/apache-repos/solr-4.10.2/example/exampledocs",
    "exclude_paths" : [ ],
    "url" : "file:/Applications/apache-repos/solr-4.10.2/example/exampledocs/",
    "max_docs" : -1
  }
}
```

Change the `max_docs` value for the above datasource:

REQUEST

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d '{"id":"SolrXML", "connector":"lucid.solrxml",
"type":"solrxml", "properties":{"path":"/Applications/solr-4.10.2/example/exampledocs", "max_docs":10}}'
http://localhost:8764/api/apollo/connectors/datasources/SolrXML
```

RESPONSE

```
true
```

15.10.3. Delete a Datasource or All Datasources

The path for a DELETE request is:

`/api/apollo/connectors/datasources/<id>`

where *<id>* is the name of the datasource to be deleted. If the id is omitted, all datasources will be deleted. The delete request takes one parameter:

Parameter	Description
force	If **false**, the default, the delete will wait for any running datasource jobs to complete before deleting the datasource. If you want to abort jobs before deleting, you can change this to **true.**

Input

None.

Output

The request succeeds silently. A failed request returns a JSON-formatted error message.

Examples

Delete the datasource named 'database':

REQUEST

```
curl -u user:pass -X DELETE http://localhost:8764/api/apollo/connectors/datasources/database
```

RESPONSE

If successful, no response.

15.11. Connector History API

The connector history REST API provides details of the last 50 runs of a named datasource. It also provides a way to clear the history.

15.11.1. Get or Delete the History for a Datasource

To GET or DELETE the history, the request path is:

`/api/apollo/connectors/history/<id>`

where <*id*> is the name of a datasource.

A GET request takes an optional parameter:

Parameter	Description
cumulative	If false, the default, details of each datasource run will be returned. Set to true if you would like to see a cumulative summary of the datasource runs.

Input

None.

Output

When the cumulative query parameter is set to true, the output will include the total number of documents processed as input to the pipeline, skipped, or failed as well as the total number of documents processed as output. Also shown will be the total number of runs of the datasource.

When the cumulative query parameter is set to false, details of each datasource job run will be shown for up to 50 past runs. The details shown will include any messages returned by the job (such as errors), the start and stop times of the job, the number of documents processed as input and output (including skipped or failed), and data for each stage of the index pipeline.

Examples

Note	Use port 8765 in local development environments only. In production, use port 8764.

Show the cumulative history for a datasource named "Lucid5Docs":

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/connectors/history/Lucid5Docs?cumulative=true
```

RESPONSE

```
{
  "total_time" : 160000,
  "input" : 378,
  "skipped" : 5,
  "failed" : 7,
  "total_runs" : 7,
  "output" : 366
}
```

Show the detailed history for a datasource named "Lucid5Docs":

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/connectors/history/Lucid5Docs
```

RESPONSE

This output has been truncated to only show one successful run.

```
{
  "message" : null,
  "id" : "Lucid5Docs",
  "crawl_stopped" : "2014-06-05T15:16:08+0000",
  "pipeline" : {
    "stats" : {
      "counters" : {
        "stage.field-mapping::myMapping" : {
          "processed" : 58
        },
        "stage.logging::conn_logging" : {
          "info" : 116,
          "processed" : 116
        },
        "stage.solr-index::solr-default" : {
          "processed" : 58
        },
        "stage.tika-parser::tika" : {
          "input" : 58,
          "processed" : 58
        }
      },
      "gauges" : { },
      "histograms" : { },
      "meters" : { },
      "timers" : { }
    },
    "history" : {
      "events" : [ ]
    }
  },
  "crawl_started" : "2014-06-05T15:15:45+0000",
  "crawl_state" : "FINISHED",
  "counters" : {
    "input" : 60,
    "skipped" : 1,
    "failed" : 1,
    "output" : 58
  },
  "job_id" : "Lucid5Docs"
}
```

15.12. Connector JDBC API

This REST API allows you to upload JDBC drivers for use with a JDBC datasource. It can also be used to find the driver class name needed for JDBC datasource configuration.

Drivers are uploaded to a directory named `fusion/data/connectors/lucid.jdbc/jdbc/<collection>`, where <*collection*> is the collection name used during the upload.

Note	When choosing a driver, be sure to use the latest version available that is compatible with your database and the Java version you are using with Fusion.

15.12.1. Upload a Driver

The path to upload a .jar file containing a JDBC driver via a POST request is:

```
/api/apollo/connectors/plugins/lucid.jdbc/resources/jdbc?collection=<collectionName>
```

where <collectionName> is the collection for which the driver will be used.

Input

The .jar file is a binary file. If using curl, use the `--form` option, or if using another client, use the supported method for calling binary files.

Output Content

No output will be returned.

Examples

Note	Use port 8765 in local development environments only. In production, use port 8764.

Upload a Postgres driver to be used by a collection named 'docs':

REQUEST

```
curl -u user:pass -X POST --form file=@/path/postgresql-9.3-1101.jdbc4.jar
http://localhost:8764/api/apollo/connectors/plugins/lucid.jdbc/resources/jdbc?collection=docs
```

RESPONSE

None.

15.12.2. List Available Drivers

A GET request will return all of the loaded drivers for a collection. The request path is:

```
/api/apollo/connectors/plugins/lucid.jdbc/resources/jdbc?collection=<collectionName>
```

where <collectionName> is the name of the collection.

Input

None.

Output

The response will include the name of the .jar file loaded and the available driver class names. The driver class name is the value that should be used when configuring a Database datasource.

Examples

Return drivers for a collection named 'docs':

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/connectors/plugins/lucid.jdbc/resources?collection=docs
```

RESPONSE

```
[ {
  "name" : "postgresql-9.3-1101.jdbc4.jar",
  "properties" : {
    "type" : "jar"
  }
}, {
  "name" : "org.postgresql.Driver",
  "properties" : {
    "type" : "class"
  }
} ]
```

15.12.3. Delete a Driver

The path to remove the .jar file of a JDBC driver from a specific collection via a DELETE request is:

```
/connectors/v1/connectors/plugins/lucid.jdbc/resources/<driver>?collection=<collection>
```

where <driver> is the name of the driver being removed, and <collection> is the name of the collection to which it is attached.

Input

None.

Output

No output will be returned. The removal of the JDBC driver can be confirmed by sending the above request to list available drivers and verifying its absence.

Example

Delete the Postgres driver from the collection named 'docs':

REQUEST:

```
curl -u admin:password -X DELETE http://localhost:8984/con
nectors/v1/connectors/plugins/lucid.jdbc/resources/postgresql-9.3-1101-jdbc4.jar
?collection=docs
```

RESPONSE:

None.

Note that the DELETE request is sent to the connector port (default: 8984), rather than the Fusion UI port (default: 8764).

15.13. Connector Jobs API

The Connector Jobs API provides methods to start, monitor, and stop a datasource. A datasource is specific connector instance that connects to a defined repository, collects content, and sends it through an index pipeline.

The Connector Jobs API can also provide detailed information about the indexing job and each stage of the index pipeline.

To define and launch a job, simply use the datasource id with a POST request to start crawling.

15.13.1. Start, Stop or Check Status of a Job

The request path is:

`/api/apollo/connectors/jobs/<id>`

where <*id*> is the name of the datasource. A GET request will return the status of a job. A POST will create and start a job. PUT will start an existing job. DELETE will stop the job.

DELETE requests have the following parameters which control how running jobs are stopped:

Parameter	Description
abort	When **false**, the default, the job will be allowed to finish processing before stopping. Use **true**, if you need the job to stop immediately.
wait_time	The wait_time allows you to configure the number of milliseconds to wait while stopping a job before the job is aborted.

If the job id is not specified, the DELETE request has the following parameters:

Parameter	Description
connector	The name of a connector. This would allow you to stop all jobs using the 'lucid.dih' connector, for example, to stop all database crawls.
collection	The name of a collection, to stop all jobs for a single named collection.

Input

None.

Output

The output will include the state of the job (RUNNING, FINISHED, etc.), the start time, and documents retrieved so far (in `counters`).

Examples

Note	Use port 8765 in local development environments only. In production, use port 8764.

Start crawling a datasource named "MyDocs":

REQUEST

```
curl -u user:pass -X POST http://localhost:8764/api/apollo/connectors/jobs/MyDocs
```

RESPONSE

```
{
  "id" : "MyDocs",
  "dataSourceId" : "MyDocs",
  "state" : "RUNNING",
  "message" : null,
  "startTime" : 1397840639000,
  "endTime" : -1,
  "finished" : false,
  "counters" : { },
  "exception" : null,
  "running" : true
}
```

15.13.2. Get Detailed Job Statistics

To GET detailed job statistics, the request path is:

```
`/api/apollo/connectors/jobs/<id>/pipeline`
```

where <*id*> is the name of the datasource.

Input

None.

Output

The output will show the results of each stage of the index pipeline, including the number of documents processed at each stage.

For example, the field-mapping stage would just show the number of documents that passed through the stage. The tika-parser stage, however, will show the number of documents input from the connector and the number output to the next stage. In a solr-index stage, the number of documents processed would indicate the number of documents that were added to the index.

If a stage encountered errors for any or all documents, the error would be shown for each document. If all documents failed due to a badly formed index pipeline, the output of this REST API may be quite lengthy.

Examples

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/connectors/jobs/TwitterSearch/pipeline
```

RESPONSE

```
{
  "stats" : {
    "counters" : {
      "stage.field-mapping::twitter-mapping" : {
        "processed" : 101
      },
      "stage.logging::conn_logging" : {
        "info" : 202,
        "processed" : 202
      },
      "stage.solr-index::solr-default" : {
        "command.ok.commit" : 1,
        "processed" : 201
      },
      "stage.tika-parser::tika" : {
        "input" : 101,
        "processed" : 202
      }
    },
    "gauges" : { },
    "histograms" : { },
    "meters" : { },
    "timers" : { }
  },
  "history" : {
    "events" : [ ]
  }
}
```

15.14. Connector Plugins API

The Connector Plugins API allows you to identify the available connectors and plugins as well as listing the datasource configuration properties for a specific connector plugin type. See Connectors and Datasources for discussion of the relationship between the two.

15.14.1. List Connectors

A GET request lists all available connectors. The path for this request is one of:

`/api/apollo/connectors`

`/api/apollo/connectors/plugins`

Both endpoints have the same behavior, so either can be used.

Input

None.

Output

A JSON list of connector names.

Examples

Note	Use port 8765 in local development environments only. In production, use port 8764.

Get all available connectors:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/connectors/plugins
```

RESPONSE

```
[ "lucid.twitter.search", "lucid.push", "lucid.hadoop.mapr", "lucid.azure.table", "lucid.anda",
"lucid.hadoop.cloudera", "lucid.azure.blob", "lucid.hortonworks", "lucid.mongodb", "lucid.hadoop",
"lucid.hadoop.apache2", "lucid.fs", "lucid.hadoop.apache1", "lucid.jdbc", "lucid.hadoop.intel",
"lucid.twitter.stream", "lucid.solrxml" ]
```

15.14.2. List Connector Types

A GET request lists all supported plugin types for a specific connector. The path for this request is one of:

`/api/apollo/connectors/plugins/<name>`

`/api/apollo/connectors/plugins/<name>/types`

where <*name*> is a specific connector. These endpoints act the same, so either can be used.

To see the complete datasource specification for this connector plugin type, the GET request path is:

```
/api/apollo/connectors/plugins/<name>/types/<type>
```

where <*name*> is a specific connector and <*type*> is a supported plugin type for this connector.

Input

None.

Output

The output is a JSON schema that details each available property of each supported type for each connector requested.

The schema will include default values for any property, the data type for the property, if the property is editable, if it has a list of allowed values, and if it is required when creating a datasource for that type.

Examples

List the property specifications for the "file" type of the "lucid.fs" connector (the output has been truncated for space):

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/connectors/plugins/lucid.fs/types/file
```

RESPONSE

```
{
  "props" : [ {
    "description" : "general",
    "name" : "---"
  }, {
    "read_only" : false,
    "default_value" : null,
    "description" : "datasource.collection",
    "hints" : [ "summary" ],
    "allowed_values" : null,
    "name" : "collection",
    "type" : "string",
    "required" : false
  }, {
    "read_only" : false,
    "default_value" : false,
    "description" : "datasource.debug",
    "hints" : [ "advanced" ],
    "allowed_values" : null,
    "name" : "debug",
    "type" : "boolean",
    "required" : false
  }, {
    "read_only" : false,
    "default_value" : null,
    "description" : "datasource.initial_mapping",
    "hints" : [ "advanced" ],
    "allowed_values" : null,
    "name" : "initial_mapping",
    "type" : "map",
    "required" : false
  }, {
    "read_only" : false,
    "default_value" : null,
    "description" : "datasource.db",
    "hints" : [ "advanced" ],
    "allowed_values" : null,
    "name" : "db",
    "type" : "map",
    "required" : false
  }, {
    "description" : "root path",
    "name" : "---"
  }, {
    "read_only" : false,
    "default_value" : null,
    "description" : "datasource.path",
    "hints" : [ "summary" ],
    "allowed_values" : null,
    "name" : "path",
    "type" : "string",
    "required" : true
  }, {
    "description" : "Filesystem URL",
    "name" : "---"
  }, {
    "read_only" : true,
    "default_value" : null,
    "description" : "datasource.url",
    "hints" : [ "summary" ],
    "allowed_values" : null,
    "name" : "url",
    "type" : "string",
    "required" : false
  },
...
] }
```

15.14.3. Drop ConnectorDB State

A DELETE request will drop the tables in the connector database for the named plugin. The request path is:

`DELETE /api/apollo/connectors/plugins/<name>/state?collection=<collectionId>`

where <*name*> is the name of the connector plugin for which to drop the state and <*collectionId*> is the name of a collection associated with this plugin. The collection parameter is optional.

Input

None.

Output Content

None.

Examples

Delete the crawl database for the SolrXML plugin:

REQUEST

```
curl -u user:pass -X DELETE http://localhost:8764/api/apollo/plugins/lucid.solrxml/state
```

RESPONSE

None.

15.15. Connector Status API

The connector registry loads all of the plugin classes and their dependencies. The Connector Status API reports the status of this registry. It can also reload or shut down the registry.

15.15.1. Reload, Quit or Get Status of Connector Registry

The path for the request is:

`/api/apollo/connectors/status`

A GET request will return the connector registry status. A DELETE request will shut down the registry. A PUT will reload the registry. The PUT request takes an optional parameter:

Parameter	Description
initRegistry	If **true**, reinitialize the connectors registry.

Input

None.

Output Content

The output of a GET request will include the status of the registry and any messages reported.

The output of a PUT or a DELETE request will be empty.

Examples

Note	Use port 8765 in local development environments only. In production, use port 8764.

Get the status of the connectors registry:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/connectors/status
```

RESPONSE

```
{
  "status" : "OK",
  "node" : "localhost",
  "messages" : [ {
    "message" : "2 data sources loaded, 2 data sources verified.",
    "time" : "2014-06-10T17:44:10.023Z"
  } ],
  "type" : "Local ConnectorManager"
}
```

15.16. Connectors Crawl Database API

Some of the connectors use a crawl database to track documents that have been seen by prior crawls and are able to use this information to understand which documents are new or have been updated or removed and take appropriate action in the index. The connectors that support this are currently lucid.fs and lucid.solrxml.

This API allows looking into the crawl database and dropping tables or clearing the database.

Note:

This API can be used only with connectors which support it, which at the current time are the 'lucid.fs' and 'lucid.solrxml' connectors.

The 'lucid.anda' connector also uses a crawl database, but it is not the same database, and does not have a REST API or other interface to access it.

15.16.1. Get Statistics for a Datasource Database or Drop Database

The path for this request is:

`/api/apollo/connectors/datasources/<id>/db`

where <*id*> is the name of the datasource.

A GET request will return statistics from the crawl database associated with a specific datasource. DELETE will drop the tables, meaning that the history of any crawl will be removed and all documents found on the next crawl will be treated as brand new documents and will be submitted to the index pipeline.

Input

None.

Output

The output from a GET request will include several sections detailing the database structure:

- counters: the counters section reports the document counts of database activities, such as table inserts.
- ops: the ops section reports on database operations that have occurred, such as initiating tables, retrieving items, processing items and table drops.
- tables: the tables section lists the tables in the database with a count of the number of items in each table. Inspecting the items is described in the next section.

The output from a DELETE request will be empty. When dropping the database, note that no documents will be removed from the index. However, the crawl database will be empty, so on the next datasource run, all documents will be treated as though they were never seen by the connectors.

Examples

Note	Use port 8765 in local development environments only. In production, use port 8764.

Get the crawl database statistics for the datasource named "SolrXML":

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/connectors/datasources/SolrXML/db
```

RESPONSE

```
{
  "counters" : {
    "new" : 14,
    "processed.insert" : 14
  },
  "ops" : {
    "initTable" : 4,
    "dropTable" : 7,
    "flush" : 1,
    "getItem" : 28,
    "renameTable" : 2,
    "commitUpdates" : 1,
    "listTables" : 2,
    "finishProcessing" : 14,
    "beginUpdates" : 1,
    "insertItem" : 14
  },
  "tables" : {
    "deleted" : {
      "count" : 0
    },
    "discarded" : {
      "count" : 0
    },
    "errors" : {
      "count" : 0
    },
    "items" : {
      "count" : 14
    }
  }
}
```

15.16.2. Get Table Statistics or Drop the Table

The path for this request is:

`/api/apollo/connectors/datasources/<id>/db/<table>`

where <*id*> is the name of the datasource and <table> is the name of a database table.

A GET request will return the table statistics. A DELETE request will drop the table and clear its data.

Input

None.

Output

The output from a GET request will be the statistics for the named table. This is usually the item count.

The output from a DELETE request will be empty.

When dropping tables, be aware that the 'items' table does not delete documents from the index, but instead changes the database so database considers them new documents. When dropping other tables, such as the 'errors' table, it will merely clear out old error messages.

Examples

Get the statistics for the 'items' table in the 'SolrXML' datasource's connector database:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/connectors/datasources/SolrXML/db/items
```

RESPONSE

```
{
  "count" : 14
}
```

Back to Top

15.16.3. Get or Delete Table Items

The path for this request is:

`/api/apollo/connectors/datasources/<id>/db/items/<item>`

where *<id>* is the name of the datasource and *<item>* is the name of a specific item in the table. If no item name is specified, the request will get all items.

A GET request retrieves information about an item or items.

A DELETE request removes the information *from the Crawl Database only*. Note that this doesn't affect the Solr Index.

A request takes two optional parameters:

Parameter	Description
start	The starting key, which is the document ID. If empty, response will start at the first row of the table. Used with a GET request only.
rows	The number of rows to return. The default is to return all records. Used with a GET request only.

Input

```
None.
```

Output

The output of a GET request will include information on when the document was fetched, if it contained any links to other documents, and the size of the document.

The output of a DELETE request will be empty. Note that this does not delete a document from the index, it only changes the database so if or when the document is crawled again, the database considers it a new document.

Examples

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/connectors/datasources/SolrXML/db/items/items?rows=5
```

RESPONSE

```
{
  "/Applications/solr-4.8.0/example/exampledocs/gb18030-example.xml" : {
    "timestamp" : "1398117855000",
    "fetchedUri" : null,
    "fetchTime" : 1402503143632,
    "docsCount" : 1,
    "outlinks" : null,
    "discarded" : false,
    "discardMessage" : null,
    "byteSize" : 1331,
    "exception" : null,
    "fetched" : true
  },
  "/Applications/solr-4.8.0/example/exampledocs/hd.xml" : {
    "timestamp" : "1398117855000",
    "fetchedUri" : null,
    "fetchTime" : 1402503143691,
    "docsCount" : 1,
    "outlinks" : null,
    "discarded" : false,
    "discardMessage" : null,
    "byteSize" : 2241,
    "exception" : null,
    "fetched" : true
  }
}
```

15.17. History API

Fusion stores history for each running service within the system. Usually this is used to log start and stop events for a service. However, the scheduler uses the history to store the results of scheduled tasks. For more information on schedule history, see the section on Schedules.

The History API provides information about the services that are running. The list of these services is provided by Introspect API, which is described in the REST API Reference.

15.17.1. List History and History Items

The path for this request is one of:

`/api/apollo/history`

`/api/apollo/history/<service>`

`/api/apollo/history/<service>/items`

`/api/apollo/history/<service>/items/<item>`

where <*service*> is one of the Fusion services and <*item*> is the name of a specific item.

A GET request returns history information at different levels of granularity, depending on the path.

The output will include details of each event stored in the history. These details include the start and end time, the source of the event, the type, and any additional status, details or errors.

15.17.2. Examples

View the history of the index-pipelines service:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/history/index-pipelines::v1
```

RESPONSE

```
{
  "events" : [ {
    "start" : "2014-05-16T14:11:48.849Z",
    "end" : "2014-05-16T14:11:48.849Z",
    "source" : "index-pipelines::v1",
    "type" : "start",
    "status" : "ok",
    "details" : null,
    "error" : null
  }, {
    "start" : "2014-05-16T14:12:48.845Z",
    "end" : "2014-05-16T14:12:48.845Z",
    "source" : "index-pipelines::v1",
    "type" : "start",
    "status" : "ok",
    "details" : null,
    "error" : null
  }
}
```

View items in the scheduler history:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/history/scheduler/items/1
```

RESPONSE

```
{
  "events" : [ {
    "start" : "2014-05-16T15:34:49.008Z",
    "end" : "2014-05-16T15:34:49.435Z",
    "source" : "scheduler",
    "type" : "execute",
    "status" : "ok",
    "details" : {
      "status" : 200,
      "entity" : "{\n  \"id\" : \"TwitterSearch\",\n  \"dataSourceId\" : \"TwitterSearch\",\n  \"state\" :
\"RUNNING\",\n  \"message\" : null,\n  \"startTime\" : 1400254489000,\n  \"endTime\" : -1,\n  \"finished\" :
false,\n  \"counters\" : { },\n  \"exception\" : null,\n  \"running\" : true\n}"
    },
    "error" : null
  }, {
    "start" : "2014-05-16T15:38:32.536Z",
    "end" : "2014-05-16T15:38:32.559Z",
    "source" : "scheduler",
    "type" : "execute",
    "status" : "ok",
    "details" : {
      "status" : 200,
      "entity" : "{\n  \"id\" : \"TwitterSearch\",\n  \"dataSourceId\" : \"TwitterSearch\",\n  \"state\" :
\"RUNNING\",\n  \"message\" : null,\n  \"startTime\" : 1400254712000,\n  \"endTime\" : -1,\n  \"finished\" :
false,\n  \"counters\" : { },\n  \"exception\" : null,\n  \"running\" : true\n}"
    },
    "error" : null
  }
}
```

15.18. Index Pipelines API

The Index Pipelines API provides methods for managing a set of named index pipelines. Every pipeline is made up of one or more stages. Stages can be defined during the creation of a pipeline, or stages can be defined separately and included into one or more pipelines. For details of the REST API for index stages, see Index Stages API.

Document processing proceeds stage by stage in a linear fashion. The order of the stages in a pipeline is the order in which they were defined. At installation, Fusion includes several pre-configured pipelines. See Index Pipelines for details on these default pipelines.

For more information about structuring documents for indexing, see Pushing Documents to a Pipeline.

15.18.1. Manage Index Pipelines: List, Create, Update, Delete

The path for this request is:

`/api/apollo/index-pipelines/<id>`

where <*id*> is the name of an specific pipeline.

- GET - returns the definition of a specific index pipeline, or all defined pipelines, if the pipeline name is omitted.
- POST - creates a new pipeline. The pipeline name is included in the definition in the request body.
- PUT - modifies an existing pipeline. Used to reorder pipeline stages.
- DELETE - removes the pipeline.

Index Pipeline Definition Properties

Property	Description
id *Required*	A unique ID for the pipeline.
stages *Optional*	A JSON map that lists the index pipeline stages that make up this pipeline. Each identified stage should include: • **id**: the ID of the stage to include. • **type**: The stage type. You can define a stage directly on a pipeline, or you can use a pre-existing stage. If you use a pre-existing stage, you must use "ref", as a reference to an existing stage. • **skip**: If the stage should be skipped during pipeline processing. By default, this is 'false'. • Finally, if a stage has not already been defined, the appropriate stage properties should be included. See Index Pipeline Stages for details of the properties for each stage type.

15.18.2. Refresh an Index Pipeline

When changes are made to a pipeline, the pipeline needs to be refreshed (reloaded), via a PUT request to endpoint:

`/api/apollo/index-pipelines/<id>/refresh`

A Fusion restart refreshes all pipelines. If the refresh was successful, there will be no response.

15.18.3. Submit a Set of Documents to an Index Pipeline

The path for this request is:

`/api/apollo/index-pipelines/<id>/collections/<collectionName>/index`

where <*id*> is the name of an specific pipeline and <*collectionName*> is the name of an specific collection.

Documents are indexed through a POST request. The documents to be indexed are sent in the request body.

This request takes the following optional query parameters:

Parameter	Description
simulate	If **true**, documents won’t be sent to solr for indexing, i.e., the solr-index stage is skipped. The default is **false**.
echo	If **true**, the default, the list of documents indexed will be returned. If this is **false**, no output will be returned when the pipeline has finished.
bufferDocsForSolr	If **true**, documents will be buffered before sending to Solr. This is an asynchronous mode and may give faster performance when indexing a large number of documents. The default is **false**.
eventsCollection	Required for event processing: a string containing the name of the target collection that is the index over link events.
eventsPipeline	Required for event processing: a string containing the name of the index pipeline that processes the link events.
eventTypes	Optional: comma-separated list of eventTypes to be processed by the index pipeline specified by eventsPipeline parameter. Currently, only event type is "links".

Input

The index pipelines can take any document format that is supported by Tika (assuming the pipeline contains a tika-parser stage). The document format is indicated by a content type declaration in the header of the REST call.

Documents can be submitted using the PipelineDocument JSON notation. The content type header to use for this format is:

`application/vnd.lucidworks-document`

Parameter	Description
id *Optional*	The id of the document. If the document does not have an ID, one will be automatically generated during processing.

fields
Optional
	The content of the document arranged into fields. The fields are expressed in a JSON array with strings for each field that contain the field "name" property and the field "value" property, as in:
	`"fields":[{"name":"fieldName","value":"fieldValue"}]`
	For multiple values of a field, you would repeat the "name" and "value" string, as in:
	`"fields":[{"name":"fieldName","value":"fieldValue1"},{"name":"fieldName","value":"fieldValue2"}]`

commands *Optional*	Commands can be added to documents to tell the index what to do with the document. Documents can consist entirely of a command in order to issue a commit, or delete documents based on a query. The valid commands are: • add: add a document to the index. If the document already exists, it will be removed and the new document added. • delete: remove a document from the index. • commit: issue a commit to the index. This command would be used as the last document in a set of documents to commit them all to the index. • query: • delete_by:
metadata *Optional*	This property is used by stages to add how field data was retrieved from a document, and for other purposes.

Output

The output by default will be a JSON representation of the document, including all fields of the document after the pipeline has completed processing. This may be the original fields of the document and will also include any fields added by the pipeline stages.

If the query parameter 'echo' has been set to false, no output will be returned.

15.18.4. Debug a Pipeline

The path for this request is:

`/api/apollo/index-pipelines/<id>/collections/<collectionName>/debug`

where <*id*> is the name of an specific pipeline and <*collectionName*> is the name of an specific collection.

Debugging is done via a POST request. The documents to be debugged are sent in the request body. The output shows the state of the document after each indexing stage. A debug request will index documents to the system, but you can prevent it from doing so by setting the query parameter `simulate` to false.

The request takes to optional query parameters:

Parameter	Description
simulate	If **true**, documents won't be sent to solr for indexing, i.e., the solr-index stage is skipped.The default is **false**.
bufferDocsForSolr	If **true**, documents will be buffered before sending to Solr. This is an asynchronous mode and may give faster performance when indexing a large number of documents.

The output will include details of each stage of the pipeline and a JSON representation of each document as it passed through each stage, including all fields of the document (original fields of the document and any fields added by the pipeline stages).

15.18.5. Examples

List the 'default' pipeline: REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/index-pipelines/default
```

RESPONSE

```
{
  "id" : "default",
  "stages" : [ {
    "type" : "solr-index",
    "id" : "solr-default",
    "skip" : false
  } ]
}
```

Create an index pipeline named 'my-index-pipeline' with three stages, one of which does not yet exist:

REQUEST

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d '{"id":"my-index-
pipeline","stages":[{"id":"tika","type":"tika-
parser","includeImages":true},{"id":"conn_mapping","type":"ref"},{"id":"solr-default","type":"ref"}]}'
http://localhost:8764/api/apollo/index-pipelines
```

RESPONSE

```
{
  "id" : "my-index-pipeline",
  "stages" : [ {
    "type" : "tika-parser",
    "id" : "tika",
    "includeImages" : true,
    "flattenCompound" : false,
    "addFailedDocs" : false,
    "addOriginalContent" : true,
    "contentField" : "_raw_content_",
    "skip" : false,
    "label" : "tika-parser"
  }, {
    "type" : "ref",
    "id" : "conn_mapping",
    "skip" : false,
    "label" : "ref"
  }, {
    "type" : "ref",
    "id" : "solr-default",
    "skip" : false,
    "label" : "ref"
  } ]
}
```

Reload the 'my-index-pipeline' pipeline:

INPUT

```
curl -u user:pass -X PUT http://localhost:8764/api/apollo/index-pipelines/my-index-pipeline/refresh
```

Index two JSON documents through a pipeline named 'conn_solr' and a collection named 'my-docs':

INPUT

```
curl -u user:pass -X POST -H "Content-Type: application/vnd.lucidworks-document" -d '[{"id":
"myDoc1","fields": [{"name":"title", "value": "My first document"},{"name":"body", "value": "This is a simple
document."}]}, {"id": "myDoc2","fields": [{"name":"title","value": "My second document"},{"name":"body",
"value": "This is another simple document."}]}]' http://localhost:8764/api/apollo/index-
pipelines/conn_solr/collections/my-docs/index
```

OUTPUT

```
[ {
  "id" : "myDoc1",
  "fields" : [ {
    "name" : "content",
    "value" : "This is a simple document.",
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "title",
    "value" : "My first document",
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "parsing_s",
    "value" : "no_raw_data",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  }, {
    "name" : "parsing_time_l",
    "value" : [ "java.lang.Long", 7 ],
    "metadata" : { },
    "annotations" : [ ]
  } ],
  "metadata" : { },
  "commands" : [ ]
}, {
  "id" : "myDoc2",
  "fields" : [ {
    "name" : "content",
    "value" : "This is another simple document.",
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "title",
    "value" : "My second document",
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "parsing_s",
    "value" : "no_raw_data",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  }, {
    "name" : "parsing_time_l",
    "value" : [ "java.lang.Long", 0 ],
    "metadata" : { },
    "annotations" : [ ]
  } ],
  "metadata" : { },
  "commands" : [ ]
} ]
```

Index a PDF document with the 'conn_solr' pipeline:

INPUT

```
curl -u user:pass -X POST -H "Content-Type: application/pdf" --data-binary @/solr/core/src/test-
files/mailing_lists.pdf http://localhost:8764/api/apollo/index-pipelines/conn_solr/collections/my-docs/index
```

OUTPUT

```
[ {
  "id" : "d6c7757e-33d9-4fbb-aa38-eef84d679ca9",
  "fields" : [ {
    "name" : "fileSize_l",
    "value" : "8582",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  }, {
    "name" : "parsing_s",
    "value" : "no_raw_data",
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "pageCount_i",
    "value" : "2",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  }, {
    "name" : "parsing_time_l",
    "value" : [ "java.lang.Long", 1171 ],
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "parsing_time_l",
    "value" : [ "java.lang.Long", 0 ],
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "attr_pdf:encrypted_",
    "value" : "false",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  }, {
    "name" : "attr_X-Parsed-By_",
    "value" : "org.apache.tika.parser.pdf.PDFParser",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  }, {
    "name" : "attr_pdf:PDFVersion_",
    "value" : "1.3",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  }, {
    "name" : "attr_producer_",
    "value" : "FOP 0.20.5",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  }, {
    "name" : "content",
    "value" : "\nSolr Mailing Lists\n\nTable of contents\n1 ",
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "attr_dc:format_",
    "value" : "application/pdf; version=1.3",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  }, {
    "name" : "mimeType_s",
    "value" : "application/pdf",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  } ],
  "metadata" : { },
  "commands" : [ ]
} ]
```

Index a JSON document though the 'conn_solr' pipeline into a collection called 'docs', using the "command" option:

INPUT

```
curl -u user:pass -X POST -H "Content-Type: application/vnd.lucidworks-document" -d '[{"id":
"myDoc2","commands": [{"name":"delete","value": "myDoc2"}]},{"id": "myDoc1","commands":
[{"name":"delete","value": "myDoc1"},{"name":"commit","value": "true"}]}]'
http://localhost:8764/api/apollo/index-pipelines/conn_solr/collections/docs/index
```

OUTPUT

```
[ {
  "id" : "myDoc2",
  "fields" : [ ],
  "commands" : [ {
    "name" : "delete",
    "params" : { }
  } ]
}, {
  "id" : "myDoc1",
  "fields" : [ ],
  "commands" : [ {
    "name" : "delete",
    "params" : { }
  }, {
    "name" : "commit",
    "params" : { }
  } ]
} ]
```

Index two simple documents through a pipeline named 'conn_solr' and a collection named 'my-docs' and get a detailed output of the pipeline process:

INPUT

```
curl -u user:pass -X POST -H "Content-Type: application/vnd.lucidworks-document" -d '[{"id":
"myDoc1","fields": [{"name":"title", "value": "My first document"},{"name":"body", "value": "This is a simple
document."}]}, {"id": "myDoc2","fields": [{"name":"title","value": "My second document"},{"name":"body",
"value": "This is another simple document."}]}]' http://localhost:8764/api/apollo/index-
pipelines/conn_solr/collections/my-docs/debug
```

OUTPUT

The output will include how each document passed through each stage. (In the example output below, we have truncated the 'field-mapping' stage for space.)

```
 {
  "stages" : [ {
    "type" : "tika-parser",
    "id" : "conn_tika",
    "includeImages" : true,
    "flattenCompound" : false,
    "addFailedDocs" : true,
    "addOriginalContent" : true,
    "contentField" : "_raw_content_",
    "skip" : false
  }, {
    "type" : "field-mapping",
    "id" : "conn_mapping",
    "mappings" : [
...
],
    "unmapped" : {
      "source" : "/(.*)/",
      "target" : "attr_$1_",
      "operation" : "move"
    },
    "skip" : false
  }, {
    "type" : "multivalue-resolver",
    "id" : "conn_multivalue_resolver",
    "typeStrategy" : [ {
      "fieldName" : "string",
      "resolverStrategy" : "pick_last"
    } ],
    "skip" : false
  }, {
    "type" : "solr-index",
    "id" : "conn_solr",
    "enforceSchema" : true,
    "skip" : false
  } ],
  "output" : [ {
    "stageType" : "tika-parser",
    "stageId" : "conn_tika",
    "context" : {
      "simulate" : false,
      "stageIndex" : 0,
      "collection" : "docs",
      "async" : false
    },
    "docs" : [ {
      "id" : "6b5c10f1-d941-41a6-957f-f677f5ad0fd5",
      "fields" : [ {
        "name" : "attr_id_",
        "value" : "myDoc1",
        "metadata" : { },
        "annotations" : [ ]
      }, {
        "name" : "parsing_time_l",
        "value" : [ "java.lang.Long", 0 ],
        "metadata" : { },
        "annotations" : [ ]
      }, {
        "name" : "parsing_s",
        "value" : "no_raw_data",
        "metadata" : {
          "creator" : "tika-parser"
        },
        "annotations" : [ ]
      }, {
        "name" : "attr_fields_",
        "value" : [ "java.util.ArrayList", [ {
          "name" : "title",
          "value" : "My first document"
        }, {
          "name" : "body",
          "value" : "This is a simple document."
        } ] ],
        "metadata" : { },
        "annotations" : [ ]
      } ],
      "metadata" : { },
      "commands" : [ ]
    }, {
      "id" : "4dac3c4e-d7f5-4cbd-96dc-e2eae69711e3",
      "fields" : [ {
        "name" : "attr_id_",
        "value" : "myDoc2",
        "metadata" : { },
        "annotations" : [ ]
      }, {
        "name" : "parsing_time_l",
        "value" : [ "java.lang.Long", 0 ],
        "metadata" : { },
        "annotations" : [ ]
      }, {
        "name" : "parsing_s",
        "value" : "no_raw_data",
        "metadata" : {
          "creator" : "tika-parser"
        },
        "annotations" : [ ]
      }, {
        "name" : "attr_fields_",
        "value" : [ "java.util.ArrayList", [ {
          "name" : "title",
          "value" : "My second document"
        }, {
          "name" : "body",
          "value" : "This is another simple document."
        } ] ],
        "metadata" : { },
        "annotations" : [ ]
      } ],
      "metadata" : { },
      "commands" : [ ]
    } ]
  }, {
    "stageType" : "field-mapping",
    "stageId" : "conn_mapping",
    "context" : {
      "simulate" : false,
      "stageIndex" : 1,
      "collection" : "docs",
      "async" : false
    },
    "docs" : [ {
      "id" : "6b5c10f1-d941-41a6-957f-f677f5ad0fd5",
      "fields" : [ {
        "name" : "attr_id_",
        "value" : "myDoc1",
        "metadata" : { },
        "annotations" : [ ]
      }, {
        "name" : "parsing_s",
        "value" : "no_raw_data",
        "metadata" : {
          "creator" : "tika-parser"
        },
        "annotations" : [ ]
        }, {
          "name" : "parsing_time_l",
          "value" : [ "java.lang.Long", 0 ],
          "metadata" : { },
          "annotations" : [ ]
        }, {
          "name" : "attr_fields_",
          "value" : [ "java.util.ArrayList", [ {
            "name" : "title",
            "value" : "My first document"
          }, {
            "name" : "body",
            "value" : "This is a simple document."
          } ] ],
          "metadata" : { },
          "annotations" : [ ]
        } ],
        "metadata" : { },
        "commands" : [ ]
      }, {
        "id" : "4dac3c4e-d7f5-4cbd-96dc-e2eae69711e3",
        "fields" : [ {
          "name" : "attr_id_",
          "value" : "myDoc2",
          "metadata" : { },
          "annotations" : [ ]
        }, {
          "name" : "parsing_s",
          "value" : "no_raw_data",
          "metadata" : {
            "creator" : "tika-parser"
          },
          "annotations" : [ ]
        }, {
          "name" : "parsing_time_l",
          "value" : [ "java.lang.Long", 0 ],
          "metadata" : { },
          "annotations" : [ ]
        }, {
          "name" : "attr_fields_",
          "value" : [ "java.util.ArrayList", [ {
            "name" : "title",
            "value" : "My second document"
          }, {
            "name" : "body",
            "value" : "This is another simple document."
          } ] ],
          "metadata" : { },
          "annotations" : [ ]
        } ],
        "metadata" : { },
        "commands" : [ ]
      } ]
    }, {
      "stageType" : "multivalue-resolver",
      "stageId" : "conn_multivalue_resolver",
      "context" : {
        "simulate" : false,
        "stageIndex" : 2,
        "collection" : "docs",
        "async" : false
      },
      "docs" : [ {
        "id" : "6b5c10f1-d941-41a6-957f-f677f5ad0fd5",
        "fields" : [ {
          "name" : "attr_id_",
          "value" : "myDoc1",
          "metadata" : { },
          "annotations" : [ ]
        }, {
          "name" : "parsing_s",
          "value" : "no_raw_data",
          "metadata" : {
            "creator" : "tika-parser"
          },
          "annotations" : [ ]
        }, {
          "name" : "parsing_time_l",
          "value" : [ "java.lang.Long", 0 ],
          "metadata" : { },
          "annotations" : [ ]
        }, {
          "name" : "attr_fields_",
          "value" : [ "java.util.ArrayList", [ {
            "name" : "title",
            "value" : "My first document"
          }, {
            "name" : "body",
            "value" : "This is a simple document."
          } ] ],
          "metadata" : { },
          "annotations" : [ ]
        } ],
        "metadata" : { },
        "commands" : [ ]
      }, {
        "id" : "4dac3c4e-d7f5-4cbd-96dc-e2eae69711e3",
        "fields" : [ {
          "name" : "attr_id_",
          "value" : "myDoc2",
          "metadata" : { },
          "annotations" : [ ]
        }, {
          "name" : "parsing_s",
          "value" : "no_raw_data",
          "metadata" : {
            "creator" : "tika-parser"
          },
          "annotations" : [ ]
        }, {
          "name" : "parsing_time_l",
          "value" : [ "java.lang.Long", 0 ],
          "metadata" : { },
          "annotations" : [ ]
        }, {
          "name" : "attr_fields_",
          "value" : [ "java.util.ArrayList", [ {
            "name" : "title",
            "value" : "My second document"
          }, {
            "name" : "body",
            "value" : "This is another simple document."
          } ] ],
          "metadata" : { },
          "annotations" : [ ]
        } ],
        "metadata" : { },
        "commands" : [ ]
      } ]
    }, {
      "stageType" : "solr-index",
      "stageId" : "conn_solr",
      "context" : {
        "simulate" : false,
        "stageIndex" : 3,
        "collection" : "docs",
        "async" : false
      },
      "docs" : [ {
        "id" : "6b5c10f1-d941-41a6-957f-f677f5ad0fd5",
        "fields" : [ {
          "name" : "attr_id_",
          "value" : "myDoc1",
          "metadata" : { },
          "annotations" : [ ]
        }, {
          "name" : "parsing_s",
          "value" : "no_raw_data",
          "metadata" : {
            "creator" : "tika-parser"
          },
          "annotations" : [ ]
        }, {
          "name" : "parsing_time_l",
          "value" : [ "java.lang.Long", 0 ],
          "metadata" : { },
          "annotations" : [ ]
        }, {
          "name" : "attr_fields_",
          "value" : [ "java.util.ArrayList", [ {
            "name" : "title",
            "value" : "My first document"
          }, {
            "name" : "body",
            "value" : "This is a simple document."
          } ] ],
          "metadata" : { },
          "annotations" : [ ]
        } ],
        "metadata" : { },
        "commands" : [ ]
      }, {
        "id" : "4dac3c4e-d7f5-4cbd-96dc-e2eae69711e3",
        "fields" : [ {
          "name" : "attr_id_",
          "value" : "myDoc2",
          "metadata" : { },
          "annotations" : [ ]
        }, {
          "name" : "parsing_s",
          "value" : "no_raw_data",
          "metadata" : {
            "creator" : "tika-parser"
          },
          "annotations" : [ ]
        }, {
          "name" : "parsing_time_l",
          "value" : [ "java.lang.Long", 0 ],
          "metadata" : { },
          "annotations" : [ ]
        }, {
          "name" : "attr_fields_",
          "value" : [ "java.util.ArrayList", [ {
            "name" : "title",
            "value" : "My second document"
          }, {
            "name" : "body",
            "value" : "This is another simple document."
          } ] ],
          "metadata" : { },
          "annotations" : [ ]
        } ],
        "metadata" : { },
        "commands" : [ ]
      } ]
    } ]
  }
```

15.19. Index Stages API

The Index Stages API provides endpoints to: * manage query stage instances. * list query stage configuration properties * test processing on a set of queries

An index pipeline is comprised of index stages. Each index stage has a name and a type. The name identifies the stage instance, and the type identifies its class. Every stage type has a number of properties, which can be configured for a particular index stage instance. See the section Index Pipeline Stages for a taxonomy of index stage types.

15.19.1. List Index Stages or Properties of an Index Stage Type

The path for this request is one of:

`api/apollo/index-stages/schema`

`api/apollo/index-stages/schema/<type>`

When no type name is specified, a GET request returns a listing of all configuration properties for all index pipeline stages. When a type name is specified, a GET request returns the properties for that type.

This provides a type template, listing the name and type information of all properties, if they are required, and what the default value is (if any).

15.19.2. List an Index Stage

A GET request to endpoints:

`api/apollo/index-stages/instances/` `api/apollo/index-stages/instances/<stageId>`

If the pipeline stage instance is specified, it returns the properties for that stage, else it returns the currently configured properties for all index pipeline stage instances.

If no index stage instances have been created, this request returns the empty list "[ ]".

15.19.3. Create, Update or Delete an Index Stage

The path for this request is:

`api/apollo/index-stages/instances/<stageId>`

where <*stageId*> is the name of an index stage instance.

- POST - create a new stage. Returns a listing of the specified properties. No <*stageId*> path parameter is necessary; the information in the POST body to determine the stageId.
- PUT - update an existing stage. Returns a listing of the specified properties.
- DELETE - remove an index pipeline stage.

15.19.4. Send a Test Document through an Index Stage

A POST request with a payload containing a list of PipelineDocument objects to the endpoint:

`api/apollo/index-stages/instances/<stageId>/<collectionName>/test` will send all documents through the index stage and returns a list of the resulting PipelineDocuments after processing.

15.19.5. Examples

View the configuration properties for index stage type "regex-extractor":

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/index-stages/schema/regex-extractor
```

RESPONSE

```
{
  "type" : "object",
  "title" : "Regex Field Extraction",
  "description" : "This stage allows you to extract entities using regular expressions",
  "properties" : {
    "rules" : {
      "type" : "array",
      "title" : "Regex Rules",
      "items" : {
        "type" : "object",
        "required" : [ "pattern" ],
        "properties" : {
          "source" : {
            "type" : "array",
            "title" : "Source Fields",
            "items" : {
              "type" : "string"
            }
          },
          "target" : {
            "type" : "string",
            "title" : "Target Field"
          },
          "pattern" : {
            "type" : "string",
            "title" : "Regex Pattern",
            "format" : "regex"
          },
          "annotateAs" : {
            "type" : "string",
            "title" : "Annotation Name"
          }
        }
      }
    }
  }
}
```

See all defined index pipeline stages, regardless of type:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/index-stages/instances
```

RESPONSE

```
[{
  "type" : "tika-parser",
  "id" : "conn_tika",
  "includeImages" : true,
  "flattenCompound" : false,
  "addFailedDocs" : true,
  "addOriginalContent" : true,
  "skip" : false
},
{
  "type" : "index-logging",
  "id" : "detailed-logging",
  "detailed" : true,
  "skip" : false,
  "label" : "detailed-index-logging",
}]
```

See details of an index-stage named 'conn_tika':

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/index-stages/instances/conn_tika
```

RESPONSE

```
{
  "type" : "tika-parser",
  "id" : "conn_tika",
  "includeImages" : true,
  "flattenCompound" : false,
  "addFailedDocs" : true,
  "addOriginalContent" : true,
  "skip" : false
}
```

Create a an index stage:

REQUEST

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d '{"id": "storagesize-regex-extractor",
"type":"regex-extractor", "rules": [{"source":["name"], "target":"storage_size_ss",
"pattern":"(\\d{1,20}\\s{0,3}(GB|MB|TB|KB|mb|gb|tb|kb))", "annotateAs":"storage_size"}]}'
http://localhost:8764/api/apollo/index-stages/instances
```

RESPONSE

```
{
  "type" : "regex-extractor",
  "id" : "storagesize-regex-extractor",
  "rules" : [ {
    "source" : [ "name" ],
    "target" : "storage_size_ss",
    "pattern" : "(\\d{1,20}\\s{0,3}(GB|MB|TB|KB|mb|gb|tb|kb))",
    "annotateAs" : "storage_size"
  } ],
  "skip" : false
}
```

Delete an index stage:

REQUEST

```
curl -u user:pass -X DELETE http://localhost:8764/api/apollo/index-stages/instances/storagesize-regex-
extractor
```

No response is returned. To check that the stage is no longer defined, list all index stage instances.

Send a document through the index stage named 'conn_tika':

REQUEST

```
curl -u user:pass -X POST -H "Content-Type: application/json" -d '[{"id": "myDoc4","fields": [{"name":"title",
"value": "Another little document document"},{"name":"body", "value": "This is a simple document."}]}]'
http://localhost:8764/api/apollo/index-stages/instances/conn_tika/docs/test
```

RESPONSE

```
[ {
  "id" : "7b8a1d5b-9e42-40eb-8059-5804c4b4fc6b",
  "fields" : [ {
    "name" : "id",
    "value" : "myDoc4",
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "parsing_time",
    "value" : [ "java.lang.Long", 0 ],
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "parsing",
    "value" : "no_raw_data",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  }, {
    "name" : "fields",
    "value" : [ "java.util.ArrayList", [ {
      "name" : "title",
      "value" : "Another little document document"
    }, {
      "name" : "body",
      "value" : "This is a simple document."
    } ] ],
    "metadata" : { },
    "annotations" : [ ]
  } ],
  "metadata" : { },
  "commands" : [ ]
} ]
```

15.20. Index Profiles API

Index profiles are used to send documents to consistent endpoints, and to share pipelines among different collections. For information on using this service through the UI, see Index Profiles.

15.20.1. Create, List, Update or Delete an Index Profile

The path for this request is:

`/api/apollo/collections/<collectionName>/index-profiles/<profileName>`

where <*collectionName*> is the name of an specific collection and <*profileName*> is the name of a new or existing profile.

- GET - returns the details of the profile, or all profiles for the collection is profile name is not specified.
- PUT - create a new profile or update an existing one. No response returned.
- DELETE - remove the profile. No response returned.

15.20.2. Use a Profile to Index Documents

A set of Fusion PipelineDocument objects in JSON can be indexed directly via a POST to endpoints:

`/api/apollo/collections/<collectionName>/index-profiles/<profileName>/index`

`/api/apollo/collections/<collectionName>/index-profiles/<profileName>/debug`

where <*collectionName*> is the name of an specific collection and <*profileName*> is the name of a new or existing profile.

A POST to the `index` endpoint indexes the document. A POST to the `debug` endpoint indexes the document and outputs information about the state of the pipeline document after each stage. See also the Index Pipelines API.

Both endpoints allow the following query parameters:

Parameter	Description
simulate	If **true**, documents will not be sent to solr for indexing, i.e., the solr-index stage will be skipped. The default is **false**.
echo	This parameter is only available with the the `index` endpoint. If **true**, the default, the list of documents indexed will be returned. If this is set to **false**, no output will be returned when the pipeline has finished.
bufferDocsForSolr	If **true**, documents will be buffered before sending to Solr. This is an asynchronous mode and may give faster performance when indexing a large number of documents. The default is **false**.

The output will include a JSON representation of the documents after they have passed through the pipeline, unless the `echo` query parameter has been set to 'false'.

Examples

Create a profile named 'testProfile' for the 'docs' collection and associate it with the pipeline named 'docs-default':

REQUEST

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d '"docs-default"'
http://localhost:8764/api/apollo/collections/docs/index-profiles/testProfile
```

RESPONSE

None.

Index two simple documents with the profile named 'testProfile':

REQUEST

```
curl -u user:pass -X POST -H "Content-Type: application/vnd.lucidworks-document" -d '[{"id":
"myDoc1","fields": [{"name":"title", "value":"My first document"},{"name":"body", "value": "This is a simple
document."}]},{"id": "myDoc2","fields": [{"name":"title","value": "My second document"},{"name":"body",
"value": "This is another simple document."}]}]' http://localhost:8764/api/apollo/collections/docs/index-
profiles/testProfile/index
```

RESPONSE

```
[ {
  "id" : "myDoc1",
  "fields" : [ {
    "name" : "content",
    "value" : "This is a simple document.",
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "title",
    "value" : "My first document",
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "parsing_s",
    "value" : "no_raw_data",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  }, {
    "name" : "parsing_time_l",
    "value" : [ "java.lang.Long", 6 ],
    "metadata" : { },
    "annotations" : [ ]
  } ],
  "metadata" : { },
  "commands" : [ ]
}, {
  "id" : "myDoc2",
  "fields" : [ {
    "name" : "content",
    "value" : "This is another simple document.",
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "title",
    "value" : "My second document",
    "metadata" : { },
    "annotations" : [ ]
  }, {
    "name" : "parsing_s",
    "value" : "no_raw_data",
    "metadata" : {
      "creator" : "tika-parser"
    },
    "annotations" : [ ]
  }, {
    "name" : "parsing_time_l",
    "value" : [ "java.lang.Long", 0 ],
    "metadata" : { },
    "annotations" : [ ]
  } ],
  "metadata" : { },
  "commands" : [ ]
} ]
```

15.21. Messaging API

The Messaging Service provides implementations to support sending messages and alerts under a coherent REST API. Service instances can be added and removed via REST API. Messages can be scheduled or sent immediately.

15.21.1. Attributes

The Messaging API supports the generic concept of a message, and each implementation interprets attributes in context. For example, *"to:"* in the context of a Slack message references the intended channnel, whereas *"to:"* in the context of an email(SMTP) references the intended email address of its recipient.

Message Attribute	Description
`id`	An application specific id for tracking the message. Must be unique.
`type`	The type of message to send, one of the following: * `log` * `smtp` * `slack` * `pagerduty`
`to`	One or more destinations for the message, as a list
`from`	Who/what the message is from
`subject`	The subject of the message
`body`	The main body of the message
`schedule`	Used to pass messages at a later time, or at a recurring basis.
`messageServiceParams`	Passes a map of any service-specific parameters, such as the user name and password

15.21.2. Add or Display a list of messaging service instances

The path for this request is:

```
api/apollo/messaging/service
```

- GET - lists all active instances.
- POST - adds a new MessageService instance.

15.21.3. Send a message

The path for this request is one of:

```
api/apollo/messaging/send
api/apollo/messaging/send/<type>/<id>
```

- POST - sends the message, where *type* specifies the type of message, and *id* specifies which message if there is more than one
- PUT - sends scheduled message.

15.21.4. Schedule messaging

Fusion will automatically fill in the Schedule Call Params. The path for this request is:

```
apollo/messaging/schedule
apollo/messaging/schedule/<id>
```

POST schedule one or more messages. Each Message has an associated Schedule object that is used to schedule that item.
PUT: Update a Message that has been scheduled. This only works for unsent or repeating Messages that have been scheduled.
DELETE: Deletes a Message that has been scheduled. This only works for unsent or repeating Messages that have been scheduled.

15.21.5. Check message status

To **GET** message services status, the path is:

```
api/apollo/messaging/status
```

15.21.6. Examples

Display current messaging services:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/messaging/service
```

RESPONSE

```
[ {
"type": "logging-message-service-config",
"defaultLogLevel" : "info",
"template": "<to>:<from>:<subject>:<body>",
"serviceType": "log"
} ]
```

Send a message using Slack

REQUEST

```
curl -u user:pass -X POST -H 'Content-Type: application/json' -d  '{"id": "myID", "type": "slack", "subject":
"test", "body": "@recipient: this is a test", "to": ["recipient"], "from": "sender"}'
http://localhost:8764/api/apollo/messaging/send
```

RESPONSE

```
[ {   "id": "myID",
    "type": "slack",
    "subject": "test",
    "body": "@recipient: this is a test",
        "to": "recipient",
    "from": "sender"
} ]
```

Get all scheduled messages

```
curl -u user:pass http://localhost:8764/api/apollo/messaging/scheduled
```

RESPONSE

```
[
 {
   "id": "myID",
   "type": "slack",
   "subject": "Updates",
   "body": "@recipient:here is the latest version",
   "to": ["updatechannel"],
   "from": "sender",
   "schedule":{"id":"slack", "creatorType":"human", "creatorId":"admin", "repeatUnit":"DAY", "interval":1,
"startTime":"2015-05-21T06:44:00.000Z", "active":true}
 }
```

Show the current status of messaging services:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/messaging/status
```

RESPONSE

```
[ {
  "status" : "ok",
  "node" : "http://xxx.xx.xx.x:8764/api/apollo/messaging",
  "messages" : [ ]
} ]
```

15.22. Nodes API

The Nodes API allows users to check active services according to their hosts.

15.22.1. Get Hosts, Node Status by Service or Node Status by Host

The path for this request is one of:

`/api/apollo/nodes/up`

`/api/apollo/nodes/upByHost`

`/api/apollo/nodes/hosts`

A GET request to the 'up' endpoint, shows each service that is 'up' and which hosts are serving the service.

A GET request to the 'upByHost' endpoint shows each host and the services served by that host.

A GET to the 'hosts' endpoint shows the active hosts.

The output will vary by endpoint.

When using the 'up' endpoint, the output will list each service active in the system and then the hosts serving the service.

When using the 'upByHost' endpoint, the output will list each host, and then the services active on each host.

When using the 'hosts' endpoint, the output will list the hosts only.

15.22.2. Examples

Show each service:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/nodes/up
```

RESPONSE

```
{
  "dynamicSchema::v1" : [ "http://localhost:8764/api/apollo/dynamicSchema" ],
  "history::v1" : [ "http://localhost:8984/connectors/v1/history", "http://localhost:8764/api/apollo/history"
],
  "query-pipelines::v1" : [ "http://localhost:8764/api/apollo/query-pipelines",
"http://localhost:8764/api/apollo/query-pipelines" ],
  "solrconfig::v1" : [ "http://localhost:8764/api/apollo/collections//solr-config" ],
  "solrAdmin::v1" : [ "http://localhost:8764/api/apollo/solrAdmin",
"http://localhost:8984/connectors/v1/solrAdmin" ],
  "stopwords::v1" : [ "http://localhost:8764/api/apollo/stopwords" ],
  "collections::v1" : [ "http://localhost:8764/api/apollo/collections" ],
  "index-stages::v1" : [ "http://localhost:8984/connectors/v1/index-stages",
"http://localhost:8764/api/apollo/index-stages" ],
  "nodes::v1" : [ "http://localhost:8984/connectors/v1/nodes", "http://localhost:8764/api/apollo/nodes" ],
  "index-profiles::v1" : [ "http://localhost:8764/api/apollo/collections//index-profiles",
"http://localhost:8984/connectors/v1/collections//index-profiles" ],
  "schema::v1" : [ "http://localhost:8764/api/apollo/collections//schema" ],
  "solr::v1" : [ "http://localhost:8984/connectors/v1/solr", "http://localhost:8764/api/apollo/solr" ],
  "query-stages::v1" : [ "http://localhost:8764/api/apollo/query-stages" ],
  "blobs::v1" : [ "http://localhost:8764/api/apollo/blobs" ],
  "connectors::v1" : [ "http://localhost:8984/connectors/v1/connectors" ],
  "recommend::v1" : [ "http://localhost:8764/api/apollo/recommend" ],
  "configurations::v1" : [ "http://localhost:8764/api/apollo/configurations",
"http://localhost:8984/connectors/v1/configurations" ],
  "system::v1" : [ "http://localhost:8764/api/apollo/system", "http://localhost:8984/connectors/v1/system" ],
  "index-pipelines::v1" : [ "http://localhost:8764/api/apollo/index-pipelines",
"http://localhost:8764/api/apollo/index-pipelines", "http://localhost:8764/api/apollo/index-pipelines",
"http://localhost:8764/api/apollo/index-pipelines", "http://localhost:8764/api/apollo/index-pipelines",
"http://localhost:8764/api/apollo/index-pipelines", "http://localhost:8764/api/apollo/index-pipelines",
"http://localhost:8984/connectors/v1/index-pipelines", "http://localhost:8984/connectors/v1/index-pipelines",
"http://localhost:8764/api/apollo/index-pipelines", "http://localhost:8764/api/apollo/index-pipelines" ],
  "query-pipeline-templates::v1" : [ "http://localhost:8764/api/apollo/pipeline-templates/query" ],
  "registration::v1" : [ "http://localhost:8984/connectors/v1/registration",
"http://localhost:8764/api/apollo/registration" ],
  "searchLogs::v1" : [ "http://localhost:8764/api/apollo/searchLogs" ],
  "scheduler::v1" : [ "http://localhost:8764/api/apollo/scheduler" ],
  "aggregator::v1" : [ "http://localhost:8764/api/apollo/aggregator" ],
  "query-profiles::v1" : [ "http://localhost:8764/api/apollo/collections//query-profiles" ],
  "usage::v1" : [ "http://localhost:8764/api/apollo/usage", "http://localhost:8984/connectors/v1/usage" ],
  "introspect::v1" : [ "http://localhost:8764/api/apollo/introspect",
"http://localhost:8984/connectors/v1/introspect" ],
  "searchCluster::v1" : [ "http://localhost:8764/api/apollo/searchCluster",
"http://localhost:8984/connectors/v1/searchCluster" ],
  "features::v1" : [ "http://localhost:8764/api/apollo/features" ],
  "index-pipeline-templates::v1" : [ "http://localhost:8984/connectors/v1/pipeline-templates/index",
"http://localhost:8764/api/apollo/pipeline-templates/index" ],
  "synonyms::v1" : [ "http://localhost:8764/api/apollo/synonyms" ],
  "reports::v1" : [ "http://localhost:8764/api/apollo/reports" ],
  "signals::v1" : [ "http://localhost:8764/api/apollo/signals" ]
}
```

15.23. Query Pipelines API

The Query Pipelines API allows you to define parameters that should be applied to all queries processed by the system. A query pipeline consists of one or more stages. Query pipeline stages can be defined with the Query Stages API. See the Query Language Cheat Sheet for help constructing queries to send through a query pipeline.

15.23.1. Manage Query Pipelines: List, Create, Delete

The path for this request is:

`/api/apollo/query-pipelines/<id>`

where *<id>* is the name of an specific pipeline.

- GET - returns the definition of a specific index pipeline, or all defined pipelines, if the pipeline name is omitted.
- POST - creates a new pipeline. The pipeline name is included in the definition in the request body.
- DELETE - removes the pipeline.

15.23.2. Manage Query Pipelines: Update

A PUT request to endpoint:

`/api/apollo/query-pipelines/<id>`

will update an existing query pipeline. A followup PUT request to endpoint:

`/api/apollo/query-pipelines/<id>/refresh`

is required to make this change take effect.

15.23.3. List a Query Pipeline

The path for this request is:

`/api/apollo/query-pipelines/<id>`

where *<id>* is the name of an specific pipeline.

A GET request returns the definition of a specific query pipeline, or all defined pipelines, if the pipeline name is omitted.

The output is a representation of each pipeline requested, created, or modified, including all properties of each stage.

Output

The output is a JSON object or list of JSON objects for each pipeline.

In the case of DELETE, no output is returned.

Note:

If you update a stage with a PUT request, you must reload the stage to the system with a second PUT request:

```
curl -u user:password-X PUT http://localhost:8764/api/apollo/query-pipelines/id/refresh
```

15.23.4. Send a Query through a Pipeline

The path for this request is:

`/api/apollo/query-pipelines/<id>/collections/<collectionName>/<handler>?<solrQuery>`

where <*id*> is the name of an specific pipeline, <*collectionName*> is the name of an specific collection, and <handler> is a Solr request handler, e.g. "select" and <*solrQuery*> is any valid Solr query.

Either a GET or POST method can be used to submit this request. Use the POST method if the length of your query exceeds normal GET limitations.

The output will be very similar to a standard Solr JSON response to a query. The response header will include the query parameters, and the response will include the documents and any other requested features such as facets, highlighting, etc.

15.23.5. Query Pipeline Definition Properties

Property	Description
id *Required*	Required only when creating a new pipeline.
stages *Required*	A JSON map of the stages to include with the pipeline. The stages can exist already or they can be defined while defining the pipeline. The stage must include the stage definitions: • **id**: the ID of the stage to include. • **stageParams**: Any parameters for the stage. These are only required for a stage that has been pre-defined and you wish to modify the properties. • **type**: The stage type. You can define a stage directly on a pipeline, or you can use a pre-existing stage. If you use a pre-existing stage, you must use "ref", as a reference to an existing stage. • **skip**: If the stage should be skipped during pipeline processing. By default, this is 'false', and does not need to be defined unless you would like to skip a stage in the future. If the stage supports setting other properties, as defined in the section Query Pipeline Stages, then those can be defined while defining the stage within the pipeline.

15.23.6. Examples

Get the definition for the default query pipeline:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/query-pipelines/default
```

RESPONSE

```
{
  "id" : "default",
  "stages" : [ {
    "type" : "recommendation",
    "id" : "recommendation",
    "numRecommendations" : 10,
    "useFq" : true,
    "numSignals" : 100,
    "aggrType" : "click@doc_id_s-query_s-filters_s",
    "skip" : false
  }, {
    "type" : "solr-query",
    "id" : "solr",
    "skip" : false
  } ]
}
```

Create a new query pipeline, specifying a facet stage that is new and a solr-query stage that was previously defined:

REQUEST

```
curl -u user:pass -X POST -H 'Content-Type: application/json' -d '{"id":"my-pipeline",
"stages":[{"type":"facet", "id":"myFacets", "fieldFacets":[{"field":"data_source_s"},{"field":"author_s",
"minCount":5, "missing":true},{"field":"isRetweet_b"}], "skip":false}, {"id":"mySolr", "type":"ref",
"skip":false}]}' http://localhost:8764/api/apollo/query-pipelines
```

RESPONSE

```
{
  "id" : "my-pipeline",
  "stages" : [ {
    "type" : "facet",
    "id" : null,
    "fieldFacets" : [ {
      "field" : "data_source_s",
      "prefix" : null,
      "sort" : null,
      "limit" : null,
      "offset" : null,
      "missing" : null,
      "method" : null,
      "threads" : null,
      "minCount" : null,
      "enumCacheMinDf" : null
    }, {
      "field" : "author_s",
      "prefix" : null,
      "sort" : null,
      "limit" : null,
      "offset" : null,
      "missing" : true,
      "method" : null,
      "threads" : null,
      "minCount" : 5,
      "enumCacheMinDf" : null
    }, {
      "field" : "isRetweet_b",
      "prefix" : null,
      "sort" : null,
      "limit" : null,
      "offset" : null,
      "missing" : null,
      "method" : null,
      "threads" : null,
      "minCount" : null,
      "enumCacheMinDf" : null
    } ],
    "skip" : false
  }, {
    "type" : "solr-query",
    "id" : null,
    "skip" : false
  } ]
}
```

Query a pipeline named 'default' and a collection named "docs" for the term "solr". Also limit the response to only document 'titles' and return it in JSON format:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/query-
pipelines/default/collections/docs/select?q=solr&fl=title&wt=json
```

RESPONSE

```
{
    "response": {
        "numFound": 106,
        "start": 0,
        "docs": [
            {
                "title": [
                    "Solr and SolrAdmin APIs - Fusion Documentation - Lucidworks"
                ]
            },
            {
                "title": [
                    "Solr and SolrAdmin APIs - Fusion Documentation - Lucidworks"
                ]
            },
            {
                "title": [
                    "Search Clusters - Fusion Documentation - Lucidworks"
                ]
            },
            {
                "title": [
                    "Search Clusters - Fusion Documentation - Lucidworks"
                ]
            },
            {
                "title": [
                    "Searching - Fusion Documentation - Lucidworks"
                ]
            }
        ]
    },
    "responseHeader": {
        "status": 0,
        "QTime": 1,
        "params": {
            "fl": "title",
            "lw.pipelineId": "default",
            "q": "solr",
            "wt": "json",
            "rows": "5",
            "defType": "edismax"
        }
    }
}
```

15.24. Query Profiles API

The Query Profiles API allows users to manage query profiles. Query profiles allow users to change query pipelines during search time while pointing the search toward a static endpoint. This gives flexibility to test different stage combinations without reconfiguration. For information, see Query Profiles.

15.24.1. Create, List, Update or Delete a Query Profile

The path for this request is:

`/api/apollo/collections/<collectionName>/query-profiles/<profileName>`

where <*collectionName*> is the name of an specific collection and<*profileName*> is the name of a new or existing profile.

- GET - returns the details of the profile, or all profiles for the collection is profile name is not specified.
- PUT - create a new profile or update an existing one. No response returned.
- DELETE - remove the profile. No response returned.

15.24.2. Use a Profile to Run a Query

`GET /api/apollo/collections/collection/query-profiles/alias/handler?key=value `

`POST /api/apollo/collections/collection/query-profiles/alias/handler?key=value `

where <*collectionName*> is the name of an specific collection, <*profileName*> is the name of a query profile, and <handler> is a Solr request handler, e.g. "select" and<*solrQuery*> is any valid Solr query.

If the length of your query exceeds normal GET limitations, then the query can be sent as the body of the POST request.

This returns a standard Solr response containing the documents requested in the format requested. The output will vary depending on how the query pipeline has been configured. For example, if a Facets stage has been added to the query pipeline, the response will include facet information.

15.24.3. Examples

Create a profile named 'testProfile' for the 'docs' collection and associate it with the pipeline named 'docs-default':

REQUEST

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d '"docs-default"'
http://localhost:8764/api/apollo/collections/docs/query-profiles/testProfile
```

RESPONSE

None.

Query for the term 'title:fusion' in the 'docs' collection, using the profile named 'newProfile':

INPUT

```
curl -u user:pass http://localhost:8764/api/apollo/collections/docs/query-
profiles/newProfile/select?q=title:fusion&wt=json&fl=title
```

Note that we have also added a few other query parameters, such as 'wt=json' to get the results in JSON format, and 'fl=title' to only retrieve document titles.

OUTPUT

```
{
    "response": {
        "numFound": 85,
        "start": 0,
        "docs": [
            {
                "title": [
                    "Fusion Services - Fusion Documentation - Lucidworks"
                ]
            },
            {
                "title": [
                    "Lucidworks Fusion Documentation - Fusion Documentation - Lucidworks"
                ]
            },
            {
                "title": [
                    "Installation - Fusion Documentation - Lucidworks"
                ]
            },
            {
                "title": [
                    "Searching - Fusion Documentation - Lucidworks"
                ]
            },
            {
                "title": [
                    "Collections - Fusion Documentation - Lucidworks"
                ]
            },
            {
                "title": [
                    "Recommendations - Fusion Documentation - Lucidworks"
                ]
            },
            {
                "title": [
                    "Connectors - Fusion Documentation - Lucidworks"
                ]
            },
            {
                "title": [
                    "Reporting - Fusion Documentation - Lucidworks"
                ]
            },
            {
                "title": [
                    "Schedules - Fusion Documentation - Lucidworks"
                ]
            },
            {
                "title": [
                    "Profiles - Fusion Documentation - Lucidworks"
                ]
            }
        ]
    },
    "responseHeader": {
        "status": 0,
        "QTime": 1,
        "params": {
            "facet": "true",
            "fl": "title",
            "lw.pipelineId": "docs-default",
            "q": "title:fusion",
            "wt": "json",
            "defType": "edismax",
            "rows": "10"
        }
    }
}
```

15.25. Query Stages API

The Query Stages API provides endpoints to: * manage query stage instances. * list query stage configuration properties * test processing on a set of queries

15.25.1. List Query Stages or Properties of an Query Stage Type

The path for this request is one of:

`api/apollo/query-stages/schema`

`api/apollo/query-stages/schema/<type>`

When no type name is specified, a GET request returns a listing of all configuration properties for all query pipeline stages. When a type name is specified, a GET request returns the properties for that type.

This provides a type template, listing the name and type information of all properties, if they are required, and what the default value is (if any).

15.25.2. List a Query Stage

The path for this request is:

`api/apollo/query-stages/instances/<stageId>`

where <*stageId*> is the name of an query stage instance.

If the pipeline stage instance is specified, it returns the properties for that stage, else it returns the currently configured properties for all query pipeline stage instances.

If no query stage instances have been created, this request returns the empty list "[ ]".

15.25.3. Create, Update, or Delete a Query Stage

The path for this request is:

`api/apollo/query-stages/instances/<stageId>`

where <*stageId*> is the name of a query stage instance.

- POST - create a new stage. Returns a listing of the specified properties. No <*stageId*> path parameter is necessary; the information in the POST body to determine the stageId.
- PUT - update an existing stage. Returns a listing of the specified properties.
- DELETE - remove an query pipeline stage.

15.25.4. Send a Test Query through a Query Stage

`GET /api/apollo/query-stages/instances/stageId/test/?key=value `

`POST /api/apollo/query-stages/test?` `*key*=*value*`

The path for this request is:

`api/apollo/query-stages/instances/<stageId>/test?<key=value>`

where <*stageId*> is the name of a query stage instance and <*collectionName*> is the name of an existing collection and <*key*=*value*> is a valid Solr query.

A GET request will send the defined key=value pair through the defined stage.

A POST request allows you to define a stage and test it on a query to see the possible response.

Input

A POST request takes a query stage specification as a JSON object.

Note that when using the POST request to test a stage that does not yet exist, the stage is not created, even if the response is successful.

Output

The output will be a query response in JSON format, or an error if the stage was not defined correctly.

15.25.5. Examples

Get the properties for the "apply-defaults" type:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/query-stages/schema/set-params
```

RESPONSE

```
{
  "type" : "object",
  "title" : "Set Query Params",
  "description" : "This stage allows you to set, append, and remove query params",
  "properties" : {
    "params" : {
      "type" : "array",
      "title" : "Parameters and Values",
      "items" : {
        "type" : "object",
        "required" : [ "key" ],
        "properties" : {
          "key" : {
            "type" : "string",
            "title" : "Parameter Name"
          },
          "value" : {
            "type" : "string",
            "title" : "Parameter Value"
          },
          "policy" : {
            "type" : "string",
            "title" : "Update Policy",
            "enum" : [ "replace", "append", "remove" ],
            "default" : "append"
          }
        }
      }
    }
  }
}
```

See all defined query pipeline stages, regardless of type:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/query-stages/instances
```

RESPONSE

```
[{
  "type" : "query-logging",
  "id" : "detailed-logging",
  "detailed" : true,
  "skip" : false,
  "label" : "detailed-query-logging",
}]
```

Add a new query stage:

REQUEST

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d'{ "type" : "query-logging", "id" : "detailed-
logging", "detailed" : true }' http://localhost:8764/api/apollo/query-stages/instances
```

RESPONSE

```
 {
  "type" : "query-logging",
  "id" : "detailed-logging",
  "detailed" : true,
  "skip" : false,
  "label" : "query-logging"
}
```

Update a query stage:

REQUEST

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d'{ "type" : "query-logging", "id" : "detailed-
logging", "detailed" : true, "label" : "detailed-query-logging" }' http://localhost:8764/api/apollo/query-
stages/instances/detailed-logging
```

RESPONSE

```
{
  "type" : "query-logging",
  "id" : "detailed-logging",
  "detailed" : true,
  "skip" : false,
  "label" : "detailed-query-logging",
}
```

Note that all required elements must be included in the update.

Delete a query stage:

REQUEST

```
curl -u user:pass -X DELETE http://localhost:8764/api/apollo/query-stages/instances/detailed-logging
```

No response is returned. To check that the stage is no longer defined, list all query stage instances.

Test that a set-params stage defines properties correctly:

REQUEST

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d '{"type":"set-params",
"params":[{"key":"rows", "value":"2", "policy":"append"}]}' http://localhost:8764/api/apollo/query-
stages/solr/test?q=*:*
```

RESPONSE

```
{
  "request" : {
    "headers" : {
      "User-Agent" : [ "curl/7.30.0" ],
      "Content-Type" : [ "application/json" ],
      "Accept" : [ "*/*" ],
      "Host" : [ "localhost:8764" ],
      "Content-Length" : [ "80" ]
    },
    "params" : {
      "q" : [ "*:*" ],
      "rows" : [ "2" ]
    }
  },
  "response" : null,
  "totalTime" : 0
}
```

15.26. Realms API

Realms are used to authenticate users across several different user access control systems.

Fusion supports these types of security realms.

15.26.1. Create, Update, Delete or List Realms

The path for this request is:

`/api/realm-configs/<id>`

where <*id*> is the ID of a realm. The ID is optional for a GET request and omitted from a POST request.

A GET request returns the configured realms. If ID is omitted, all realms will be returned.

A POST request creates a new realm. If the request is successful, a new ID will be generated.

A PUT request updates a realm.

A DELETE request removes the realm.

Input

Parameter	Description
name *Required*	The name of the realm. This name will appear on the login screen of the UI, and will appear in user records to identify the realm they belong to.
enabled *Required*	If **true**, the realm is available for users to use with system authentication.
realmType *Required*	String value for realm type. Supported realm types are `native`, `ldap`, `kerberos`, `saml`, and `trusted-http`.

Native realms have users whose usernames and passwords are stored in the Fusion database. Authenticating users with an LDAP system creates a user record in Fusion, which includes a property for the realm the user belongs to. This Fusion user record is used by administrators to grant users access permissions for the UI or REST API services. LDAP realms connect to an LDAP server to verify the user’s ID and password.

Configuration for an LDAP security realm requires the following additional properties:

Parameter	Description
host	The hostname of the LDAP server.
port	The port to use when connecting to the LDAP server.
ssl	If **true**, SSL will be used when connecting to the LDAP server.
bindDN	A string consisting of the LDAP server DN (Distinguished Name) and a single pair of curly braces (`{}`) which is a placeholder for the username.

Output

When creating a new realm, the output will include the properties for the realm just created, or an error to indicate a problem with the entry.

For a GET request, the output will include all defined properties of the realm.

For a DELETE or a PUT request, no output will be returned.

Examples

Note	Use port 8765 in local development environments only. In production, use port 8764.

Get details of the default 'native' realm:

REQUEST

```
curl -u user:pass http://localhost:8764/api/realm-configs/86df9b5b-4a1c-4b0b-bc10-25aee55fef63
```

RESPONSE

```
{
    "enabled": true,
    "id": "86df9b5b-4a1c-4b0b-bc10-25aee55fef63",
    "name": "native",
    "realmType": "native"
}
```

Create a realm to support LDAP authentication:

REQUEST

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d '{"realmType":"ldap", "name":"dev-ldap",
"enabled":true, "config":{"host":"localhost", "port":10636 , "ssl":true,
"bindDn":"uid={},ou=users,dc=security,dc=example,dc=com"} }' http://localhost:8764/api/realm-configs
```

RESPONSE

```
{
   "realmType":"ldap",
   "name":"dev-ldap",
   "enabled":true,
   "config":{
      "bindDn":"uid={},ou=users,dc=security,dc=example,dc=com",
      "ssl":true,
      "port":10636,
      "host":"localhost"
   }
}
```

15.27. Recommendations API

The Recommendations REST API uses signals and aggregated signals to return a list of items that can be used for recommendations.

To use the REST API Recommendations service to get recommendations for items in some collection, that collection must have associated signals and aggregated-signals system collections. How good the recommendations are depends on how well the information in the signals and aggregated signals collections, which is derived from observed user behavior, matches user behavior going forward.

In addition to these endpoints, is also possible to get recommendations as part of a query request.

See Recommendations for a discussion of when to use the API and when to use recommender query stages.

15.27.1. Recommendation types

The API includes separate endpoints for retrieving different types of recommendations:

`itemsForQuery`	Get items for a defined query. The response is a list of *document IDs* and their weights.
`queriesForItem`	Get queries for a defined item (a document ID). This finds the top queries that led users to the defined item. The response is a list of *query terms* and their weights.
`itemsForItems`	This type retrieves items that are related to a defined item (a document ID). This can be thought of as "People also viewed…". The response is a list of *document IDs* and their weights. This type combines `itemsForQuery` and `queriesForItem`. It gets the top items using `queriesForItems`, creates shingles using the queries, then joins the shingles with an OR clause to use `itemsForQuery` to get the top items.

15.27.2. Request Parameters

`itemsForQuery` parameters

Parameter	Description
q *Required*	The query terms to search for.
rows *Optional*	The number of items to return. The default is **10**.

Parameter	Description
fq *Optional*	A filter query to limit results. If used, the field name must be included along with the value.
aggrRows *Optional*	The amount of rows per aggregation. The default is **100**.
aggrType *Optional*	The type of aggregation. The default is aggregated click.
debug *Optional*	If **true**, debug information will be returned with the results. The default is **false**.

There are also several parameters used to boost possible matches, which can be modified with the Configurations API. By default the following boosts and defaults are applied:

- A query filter limiting the query to the 'query_t' field (`qf=query_t`). This is defined with the configuration property `com.lucidworks.apollo.service.recommend.solr.query.qf`
- Three phrase field boost on the 'query_t' field (`pf=query_t3,query_t27,query_t0^1`). This boosts the 'query_t' field by 3 when terms are next to each other, a boost of 7 when they are within 2 terms of each other, and an additional boost of 1. This is defined with the configuration property`com.lucidworks.apollo.service.recommend.solr.query.pf`.
- A minimum match of 50% (`mm=50%`), which means at least 50% of the terms used must be present. This is defined with the configuration property `com.lucidworks.apollo.service.recommend.solr.query.mm`.
- The query parser is 'edismax (`defType=edismax`). This is defined with the configuration property `com.lucidworks.apollo.service.recommend.solr.query.defType`.
- The sort is by descending score, then descending weight (`score desc, weight_d desc`). This is defined with the configuration property `com.lucidworks.apollo.service.recommend.solr.query.sort`.

queriesForItem parameters

Parameter	Description
docId *Required*	The document ID to search for.
rows *Optional*	The number of items to return. The default is **10**.
aggrType *Optional*	The type of aggregation. The default is aggregated click.

itemsForItems parameters

Parameter	Description
docId *Required*	The document ID that is the basis for the itemForItem query.
rows *Optional*	The number of items to return. The default is **10**.

Parameter	Description
shinglesLimit *Optional*	The maximum number of shingles (nGrams) to use. The shingles will be joined with a boolean OR statement to find the related documents for the named item. The default is **20**.
aggrRows *Optional*	The amount of rows per aggregation. The default is **100**.
aggrType *Optional*	The type of aggregation. The default is click.
termsLimit *Optional*	A limit to the number of terms from queries to use in the itemForItems calculation. The default is **30**.

15.27.3. Output

The output includes the following sections:

`header`	The query parameters (in a section named `queryParams`) and the total time it took to process the query.
`items`	Depending on the recommendation type: - `itemsForQuery` The document IDs and the weights of aggregated events that match the query. This type supports a `debug` option that adds a `debug` section to the output. - `queriesForItem` The queries and the weights of aggregated events that match the document ID. - `itemsForItems` The document IDs and the weights of aggregated events that match the query.

15.27.4. Examples

Below are examples for each recommendation type.

itemsForQuery

Get the top items for the query 'ipod':

INPUT

```
curl -u user:pass
http://localhost:8764/api/apollo/recommend/lucidworks102/itemsForQuery?q=ipod&fq=count_d:4&debug=true
```

OUTPUT

```
{
  "header" : {
    "queryParams" : {
      "aggrType" : "*",
      "rows" : 10,
      "collection" : "lucidworks102",
      "aggrRows" : 100,
      "debug" : true,
      "q" : "ipod",
      "fq" : [ "count_d:4" ]
    },
    "totalTime" : 5
  },
  "items" : [ {
    "weight" : 1.0726584E-11,
    "docId" : "8771929"
  }, {
    "weight" : 3.865899E-12,
    "docId" : "9225439"
  }, {
    "weight" : 9.230597E-12,
    "docId" : "3109302"
  } ],
  "debugInfo" : {
    "aggrTime" : 1,
    "queryTime" : 4,
    "solrParams" : {
      "mm" : [ "50%" ],
      "pf" : [ "query_t^3", "query_t~2^7", "query_t~0^1" ],
      "fl" : [ "id", "doc_id_s", "weight_d" ],
      "sort" : [ "score desc,weight_d desc" ],
      "q" : [ "ipod" ],
      "qf" : [ "query_t" ],
      "collection" : [ "lucidworks102_signals_aggr" ],
      "fq" : [ "aggr_type_s:*", "count_d:4" ],
      "rows" : [ "100" ],
      "defType" : [ "edismax" ]
    }
  }
}
```

queriesForItem

INPUT

```
curl -u user:pass http://localhost:8764/api/apollo/recommend/lucidworks102/queriesForItem?docId=9225439
```

OUTPUT

```
{
  "header" : {
    "queryParams" : {
      "aggrType" : "*",
      "rows" : 10,
      "collection" : "lucidworks102",
      "docId" : "9225439"
    },
    "totalTime" : 8
  },
  "items" : [ {
    "query" : "ipod",
    "weight" : 3.865899E-12
  }, {
    "query" : "columbusday ipod mp3 20111009",
    "weight" : 3.5141304E-12
  }, {
    "query" : "apple itouch",
    "weight" : 2.3619889E-12
  }, {
    "query" : "ipod 4th generation",
    "weight" : 1.6436526E-12
  }, {
    "query" : "ipod touch 4th generation",
    "weight" : 9.674966E-13
  }, {
    "query" : "onlinemidnightsale ipod mp3players",
    "weight" : 9.568035E-13
  }, {
    "query" : "ipod touch",
    "weight" : 7.774231E-13
  }, {
    "query" : "itouch",
    "weight" : 7.707221E-13
  } ]
}
```

itemsForItems

INPUT

```
curl -u user:pass http://localhost:8764/api/apollo/recommend/demo/itemsForItem?docId=9225439
```

OUTPUT

```
{
  "header" : {
    "queryParams" : {
      "aggrType" : "*",
      "rows" : 10,
      "collection" : "lucidworks102",
      "aggrRows" : 100,
      "docId" : "9225439",
      "shinglesLimit" : 20,
      "termsLimit" : 30,
      "q" : "ipod OR columbusday ipod mp3 OR ipod mp3 20111009 OR columbusday ipod OR ipod mp3 OR mp3 20111009
OR 4th generation OR apple itouch OR itouch OR ipod touch OR ipod 4th OR ipod 4th generation OR ipod touch 4th
OR touch 4th generation OR touch 4th OR onlinemidnightsale ipod mp3players OR onlinemidnightsale ipod OR ipod
mp3players"
    },
    "totalTime" : 63
  },
  "items" : [ {
    "weight" : 1.02983795E-10,
    "docId" : "9225377"
  }, {
    "weight" : 1.1725405E-11,
    "docId" : "9225439"
  }, {
    "weight" : 7.803512E-13,
    "docId" : "1207345"
  }, {
    "weight" : 7.803291E-13,
    "docId" : "2193848"
  }, {
    "weight" : 7.803053E-13,
    "docId" : "2128719"
  }, {
    "weight" : 1.4804163E-12,
    "docId" : "1237142"
  }, {
    "weight" : 1.480315E-12,
    "docId" : "2136062"
  }, {
    "weight" : 3.4738492E-12,
    "docId" : "1207414"
  }, {
    "weight" : 3.4676099E-12,
    "docId" : "2382259"
  }, {
    "weight" : 1.7393786E-12,
    "docId" : "1200933"
  } ]
}
```

15.28. Reporting API

The reporting API provides a set of reports for key metrics about user searches over a primary Fusion collection by running reports against the auxiliary searchLog and signals collections. Reports include:

- 'topClicked' - documents which have received the most clicks.
- 'topQueries' - queries ordered by frequency.
- 'lessThanN' - queries which return less than N documents.
- 'topN' - query terms ordered by frequency.
- 'histo' - a histogram over all queries binned by length of query execution time.
- 'dateHisto' - shows relative rate of queries over time.

The reports available for a particular collection depends on whether or not the auxiliary searchLogs and signals collection have been created. Collections created with the Fusion UI will have both searchLogs and signals by default. The 'topClicked' report requires both searchLogs and signals auxiliary collections. All other reports require searchLogs auxiliary collections.

15.28.1. Report Configuration Information

POST requests should always send a JSON object containing the Report configuration. If no special configuration is required, this is the empty object "\{}".

The report configuration object contains the following attributes, depending on the report:

- "n" value is a positive integer, used for both "topN" and "lessThanN" reports. E.g.: `"n" : 1`
- "num" value is a positive integer, used for both "topClicked" report. E.g.: `"num" : 5`
- "field" : required for report "topN", specifies the field where the search terms are stored. Default field is "q_txt". E.g.: `"field" : "q_txt"` See Search Query Reporting for details.
- "rangeStart", "rangeEnd", "interval" : attributes used to restrict histogram report range and set bin size accordingly. E.g.: `"rangeStart": 0, "rangeEnd": 1000, "interval" : 1000`
- "dateRangeStart", "dateRangeEnd", "timeInterval" : attributes required for date range histogram report. E.g.: `"dateRangeStart": "NOW/DAY-1DAY", "dateRangeEnd": "NOW/DAY+1DAY", "timeInterval": "+1DAY"`

15.28.2. Reports request syntax

Allowed requests are of the following form:

- GET `/api/apollo/reports/<collection_name>/` - lists reports availabe a Fusion primary collection specified by path parameter <collection_name>.
- POST `/api/apollo/reports/<collection_name>/<report_name>/` - runs a report for collection specified by path parameter <collection_name>. Report configuration is sent in the body of the POST request as JSON.
- GET `/api/apollo/reports/<collection_name>/<report_name>/<item_ID>` - gets a detailed information for a specific item returned in a report. Used to drill down on specific queries or query terms.

15.28.3. Format of Report Results

All reports return a JSON list of objects or the empty list.

Contents of the object vary according to the report. The following attribute names are used across all reports:

- "key" : matching query terms
- "count" : raw count for this object
- "percentage" : normalized count expressed as the proportion of the total items in the current report represented by this item, a real number between 0.0 and 1.0.
- "token" or "item" : ID of token or item.

15.28.4. Examples

See reports available for collection "bb_catalog" which has searchLogs and signals enabled:

```
> curl -u user:pass http://localhost:8764/api/apollo/reports/bb_catalog

[ "histo", "topClicked", "topQueries", "lessThanN", "dateHisto", "topN" ]
```

See reports available for system collection "system_blobs" which doesn’t have any auxiliary collections:

```
> curl -u user:pass http://localhost:8764/api/apollo/reports/system_blobs

[ ]
```

Identify queries over collection "bb_catalog" for which no matching documents are found, i.e., queries which return less than 1 result:

```
> curl -u user:pass -X POST -H 'Content-type: application/json' -d @- \
> http://localhost:8764/api/apollo/reports/bb_catalog/lessThanN \
> <<EOF
> {"n":1}
> EOF

[ {
  "key" : "ipad",
  "count" : 3,
  "percentage" : 0.375,
  "token" : "eyJmaWx0ZXJzIjpbIm51bWRvY3NfbDpbKiBUTyAxXSIsInFfczppcGFkIl19"
}, {
  "key" : "id:2125233",
  "count" : 2,
  "percentage" : 0.25,
  "token" : "eyJmaWx0ZXJzIjpbIm51bWRvY3NfbDpbKiBUTyAxXSIsInFfczppZFxcOjIxMjUyMzMiXX0="
}, {
  "key" : "ipod",
  "count" : 1,
  "percentage" : 0.125,
  "token" : "eyJmaWx0ZXJzIjpbIm51bWRvY3NfbDpbKiBUTyAxXSIsInFfczppcG9kIl19"
}, {
  "key" : "typewriter",
  "count" : 1,
  "percentage" : 0.125,
  "token" : "eyJmaWx0ZXJzIjpbIm51bWRvY3NfbDpbKiBUTyAxXSIsInFfczp0eXBld3JpdGVyIl19"
}, {
  "key" : "unicorn",
  "count" : 1,
  "percentage" : 0.125,
  "token" : "eyJmaWx0ZXJzIjpbIm51bWRvY3NfbDpbKiBUTyAxXSIsInFfczp1bmljb3JuIl19"
} ]
```

Drill down on "lessThanN" report to examine information for "key" : "ipad" by token ID:

```
> curl -u user:pass \
>
http://localhost:8764/api/apollo/reports/bb_catalog/lessThanN/eyJmaWx0ZXJzIjpbIm51bWRvY3NfbDpbKiBUTyAxXSIsInFf
czppcGFkIl19

{
  "numFound" : 3,
  "start" : 0,
  "maxScore" : 0.0,
  "docs" : [ {
    "id" : "cdcdd42c-66f2-499e-a940-33d980596d36",
    "collection_s" : "bb_catalog",
    "q_txt" : [ "ipad" ],
    "q_s" : "ipad",
    "qtime_l" : 1,
    "totaltime_l" : 2,
    "numdocs_l" : 0,
    "timestamp_dt" : "2015-08-31T20:40:48.096Z",
    "httpmethod_s" : "POST",
    "req_q_ss" : [ "ipad" ],
    "req_debug_ss" : [ "true" ],
    "req_json.nl_ss" : [ "arrarr" ],
    "req_echoParams_ss" : [ "all" ],
    "req_lw.pipelineId_ss" : [ "bb_catalog-default" ],
    "req_fl_ss" : [ "*,score" ],
    "req_start_ss" : [ "0" ],
    "req_isFusionQuery_ss" : [ "true" ],
    "req_sort_ss" : [ "score desc" ],
    "req_rows_ss" : [ "10" ],
    "req_wt_ss" : [ "json" ],
    "_version_" : 1511054270105911296
  }, {
    "id" : "d4e22f4e-ae27-4662-82ed-17c68111f0d5",
    "collection_s" : "bb_catalog",
    "q_txt" : [ "ipad" ],
    "q_s" : "ipad",
    "qtime_l" : 3,
    "totaltime_l" : 4,
    "numdocs_l" : 0,
    "timestamp_dt" : "2015-09-01T13:45:28.008Z",
    "httpmethod_s" : "POST",
    "req_debug_ss" : [ "true" ],
    "req_json.nl_ss" : [ "arrarr" ],
    "req_echoParams_ss" : [ "all" ],
    "req_lw.pipelineId_ss" : [ "bb_catalog-default" ],
    "req_fl_ss" : [ "*,score" ],
    "req_start_ss" : [ "0" ],
    "req_isFusionQuery_ss" : [ "true" ],
    "req_rows_ss" : [ "10" ],
    "req_bq_ss" : [ "id:1945531^4.090439397841692", "id:2339322^1.5108471289277077",
"id:1945595^1.0636971555650234", "id:1945674^0.40656840801239014", "id:2842056^0.33429211378097534",
"id:2408224^0.43880610167980194", "id:2339386^0.39254774153232574", "id:2319133^0.32736557722091675",
"id:9924603^0.19560790061950684", "id:1432551^0.18906432390213013" ],
    "req_q_ss" : [ "ipad" ],
    "req_defType_ss" : [ "edismax" ],
    "req_wt_ss" : [ "json" ],
    "req_facet_ss" : [ "true" ],
    "_version_" : 1511118736467165184
  }, {
    "id" : "a249d93c-9232-4ea7-a99a-fcf01b6c2c2f",
    "collection_s" : "bb_catalog",
    "q_txt" : [ "ipad" ],
    "q_s" : "ipad",
    "qtime_l" : 0,
    "totaltime_l" : 2,
    "numdocs_l" : 0,
    "timestamp_dt" : "2015-09-01T13:46:41.309Z",
    "httpmethod_s" : "POST",
    "req_q_ss" : [ "ipad" ],
    "req_debug_ss" : [ "true" ],
    "req_json.nl_ss" : [ "arrarr" ],
    "req_echoParams_ss" : [ "all" ],
    "req_lw.pipelineId_ss" : [ "bb_catalog-default" ],
    "req_fl_ss" : [ "*,score" ],
    "req_start_ss" : [ "0" ],
    "req_isFusionQuery_ss" : [ "true" ],
    "req_rows_ss" : [ "10" ],
    "req_wt_ss" : [ "json" ],
    "req_facet_ss" : [ "true" ],
    "req_bq_ss" : [ "id:1945531^4.090439397841692", "id:2339322^1.5108471289277077",
"id:1945595^1.0636971555650234", "id:1945674^0.40656840801239014", "id:2842056^0.33429211378097534",
"id:2408224^0.43880610167980194", "id:2339386^0.39254774153232574", "id:2319133^0.32736557722091675",
"id:9924603^0.19560790061950684", "id:1432551^0.18906432390213013" ],
    "_version_" : 1511118813327785984
  } ]
}
```

Get all of the top queries, regardless of date, pass in empty date range specification:

```
> curl -u user:pass -X POST -H 'Content-type: application/json' -d '{}' \
> http://localhost:8764/api/apollo/reports/bb_catalog/topQueries

[ {
  "key" : "ipad",
  "count" : 42,
  "percentage" : 0.7118644,
  "token" : "eyJmaWx0ZXJzIjpbInFfczppcGFkIl19"
}, {
  "key" : "*:*",
  "count" : 10,
  "percentage" : 0.16949153,
  "token" : "eyJmaWx0ZXJzIjpbInFfczpcXCpcXDpcXCoiXX0="
}, {
  "key" : "id:2125233",
  "count" : 2,
  "percentage" : 0.033898305,
  "token" : "eyJmaWx0ZXJzIjpbInFfczppZFxcOjIxMjUyMzMiXX0="
}, {
  "key" : "typewriter",
  "count" : 2,
  "percentage" : 0.033898305,
  "token" : "eyJmaWx0ZXJzIjpbInFfczp0eXBld3JpdGVyIl19"
}, {
  "key" : "unicorn",
  "count" : 2,
  "percentage" : 0.033898305,
  "token" : "eyJmaWx0ZXJzIjpbInFfczp1bmljb3JuIl19"
}, {
  "key" : "ipod",
  "count" : 1,
  "percentage" : 0.016949153,
  "token" : "eyJmaWx0ZXJzIjpbInFfczppcG9kIl19"
} ]
```

_Drill down on topQueries report for item with "key" : "ipod", "token": "eyJmaWx0ZXJzIjpbInFfczppcG9kIl19"

```
> curl -u user:pass \
> http://localhost:8764/api/apollo/reports/bb_catalog/topQueries/eyJmaWx0ZXJzIjpbInFfczppcG9kIl19

{
  "numFound" : 1,
  "start" : 0,
  "maxScore" : 0.0,
  "docs" : [ {
    "id" : "4a6f7f5e-3d13-4f20-b59e-6188ce4c5783",
    "collection_s" : "bb_catalog",
    "q_txt" : [ "ipod" ],
    "q_s" : "ipod",
    "qtime_l" : 1,
    "totaltime_l" : 2,
    "numdocs_l" : 0,
    "timestamp_dt" : "2016-04-05T17:51:56.197Z",
    "httpmethod_s" : "POST",
    "req_debug_ss" : [ "true" ],
    "req_json.nl_ss" : [ "arrarr" ],
    "req_echoParams_ss" : [ "all" ],
    "req_lw.pipelineId_ss" : [ "default" ],
    "req_fl_ss" : [ "*,score" ],
    "req_start_ss" : [ "0" ],
    "req_isFusionQuery_ss" : [ "true" ],
    "req_rows_ss" : [ "10" ],
    "req_q_ss" : [ "ipod" ],
    "req_defType_ss" : [ "edismax" ],
    "req_qf_ss" : [ "doc_id_s" ],
    "req_wt_ss" : [ "json" ],
    "req_facet_ss" : [ "true" ],
    "_version_" : 1530793784714985472
  } ]
}
```

Run "topN" report over collection "bb_catalog", return top-ranking query, search field "q_txt":

```
> curl -u user:pass -X POST -H 'Content-type: application/json' -d @- \
> http://localhost:8764/api/apollo/reports/bb_catalog/topN \
> <<EOF
> { "num" : 1, "field" : "q_txt" }
> EOF

[ {
  "key" : "ipad",
  "count" : 42,
  "percentage" : 0.7118644,
  "token" : "eyJmaWx0ZXJzIjpbInFfdHh0OmlwYWQiXX0="
} ]
```

Run "topClicked" report, return 5 most-clicked documents:

```
> curl -u user:pass -X POST -H 'Content-type: application/json' -d @- \
> http://localhost:8764/api/apollo/reports/bb_catalog/topClicked \
> <<EOF
> {"num":5}
> EOF

[ {
  "key" : "2842056",
  "count" : 42636,
  "percentage" : 0.0107869385,
  "token" : "eyJmaWx0ZXJzIjpbInR5cGVfczpjbGljayIsImRvY19pZF9zOjI4NDIwNTYiXX0="
}, {
  "key" : "1945531",
  "count" : 23510,
  "percentage" : 0.0059480467,
  "token" : "eyJmaWx0ZXJzIjpbInR5cGVfczpjbGljayIsImRvY19pZF9zOjE5NDU1MzEiXX0="
}, {
  "key" : "2842092",
  "count" : 22683,
  "percentage" : 0.0057388153,
  "token" : "eyJmaWx0ZXJzIjpbInR5cGVfczpjbGljayIsImRvY19pZF9zOjI4NDIwOTIiXX0="
}, {
  "key" : "9225377",
  "count" : 21603,
  "percentage" : 0.0054655746,
  "token" : "eyJmaWx0ZXJzIjpbInR5cGVfczpjbGljayIsImRvY19pZF9zOjkyMjUzNzciXX0="
}, {
  "key" : "9755322",
  "count" : 20993,
  "percentage" : 0.005311244,
  "token" : "eyJmaWx0ZXJzIjpbInR5cGVfczpjbGljayIsImRvY19pZF9zOjk3NTUzMjIiXX0="
} ]
```

Get a histogram of the number of documents returned for queries over range 0 to 2000, interval 500 (4 bins):

```
> curl -u user:pass -X POST -H 'Content-type: application/json' -d @- \
> http://localhost:8764/api/apollo/reports/bb_catalog/histo \
> <<EOF
> {"field": "numdocs_l", "rangeStart": 0, "rangeEnd": 100, "interval": "25"}
> EOF

-X POST -H 'Content-type: application/json' -d @- http://localhost:8764/api/apollo/reports/bb_catalog/histo
<<EOF
> {"field": "numdocs_l", "rangeStart": 0, "rangeEnd": 2000, "interval": 500 }
> EOF
[ {
  "key" : "0",
  "count" : 10,
  "percentage" : 0.16949153,
  "token" : "eyJmaWx0ZXJzIjpbIm51bWRvY3NfbDpbMCBUTyA1MDB9Il19"
}, {
  "key" : "500",
  "count" : 0,
  "percentage" : 0.0,
  "token" : "eyJmaWx0ZXJzIjpbIm51bWRvY3NfbDpbNTAwIFRPIDEwMDB9Il19"
}, {
  "key" : "1000",
  "count" : 39,
  "percentage" : 0.66101694,
  "token" : "eyJmaWx0ZXJzIjpbIm51bWRvY3NfbDpbMTAwMCBUTyAxNTAwfSJdfQ=="
}, {
  "key" : "1500",
  "count" : 0,
  "percentage" : 0.0,
  "token" : "eyJmaWx0ZXJzIjpbIm51bWRvY3NfbDpbMTUwMCBUTyAyMDAwfSJdfQ=="
} ]
```

Get a date histogram for the last two days, with an interval of 1 day:

```
> curl -u user:pass -X POST -H Content-type:application/json -d @- \
> http://localhost:8764/api/apollo/reports/bb_catalog/dateHisto \
> <<EOF
> {"dateRangeStart": "NOW/DAY-1DAY", "dateRangeEnd": "NOW/DAY+1DAY", "timeInterval": "+1DAY"}
> EOF

[ {
  "key" : "2016-04-04T00:00:00Z",
  "count" : 0,
  "percentage" : 0.0,
  "token" :
"eyJmaWx0ZXJzIjpbInRpbWVzdGFtcF9kdDpbTk9XL0RBWS0xREFZIFRPIE5PVy9EQVkrMURBWV0iLCJ0aW1lc3RhbXBfZHQ6WzIwMTZcXC0wN
FxcLTA0VDAwXFw6MDBcXDowMFogVE8gMjAxNi0wNC0wNFQwMDowMDowMForMURBWX0iXX0="
}, {
  "key" : "2016-04-05T00:00:00Z",
  "count" : 7,
  "percentage" : 1.0,
  "token" :
"eyJmaWx0ZXJzIjpbInRpbWVzdGFtcF9kdDpbTk9XL0RBWS0xREFZIFRPIE5PVy9EQVkrMURBWV0iLCJ0aW1lc3RhbXBfZHQ6WzIwMTZcXC0wN
FxcLTA1VDAwXFw6MDBcXDowMFogVE8gMjAxNi0wNC0wNVQwMDowMDowMForMURBWX0iXX0="
} ]
```

15.29. Roles API

Roles are groups of permissions that allow access to the UI and the REST APIs. See Roles for details.

Security Realms can be configured to use LDAP group membership to assign Roles to users. See the LDAP configuration instructions for details.

15.29.1. Create, Update or Delete Roles

The endpoint for this request can take the role ID as a request parameter:

`/api/roles/<id>`

The role ID string is generated by Fusion when the role is created.

A GET request returns the configured roles for a specific ID. If the ID is omitted from the path, all roles will be returned.

A POST request creates a new role. When creating a new role, the request path is `/api/roles`. If the role is created, the request returns the role ID.

A PUT request updates an existing role.

A DELETE request will remove the role configuration.

15.29.2. Role Specification

To create or update a Role via a POST or PUT request, the request body is a JSON object with the following attributes:

Property	Description
name *Required*	A string containing the role name.
desc *Optional*	A string containing a brief text description, for display on the Access Control "ROLES" panel.
permissions *Optional*	A list of permissions, specified in JSON notation. See section Permissions for details.
uiPermissions *Optional*	A list of names of UI components.

The following example describes a role with permissions to access Fusion Dashboards for collection "mdb1":

```
{
  "name":"view-dashboard-mdb1",
  "desc":"can access/use analytics dashboard \"mdb1\" but not allowed to change dashboard controls.",
  "permissions":[
    {"methods":["GET"],"path":"/solr/system_banana/*"},
    {"methods":["GET"],"path":"/solr/{id}/*","params":{"id":["mdb1"]}},
    {"methods":["GET"],"path":"/solr/{id}/admin/luke","params":{"id":["mdb1"]}},
    {"methods":["GET"],"path":"/collections/system_banana"}
  ],
  "uiPermissions":[
    "dashboards",
    "fields"
  ]
}
```

15.29.3. Examples

Get the details for the role with id '3416c03a-31df-4103-b446-358f6790af3e':

REQUEST

```
curl -u user:pass http://localhost:8764/api/roles/3416c03a-31df-4103-b446-358f6790af3e
```

RESPONSE

```
{
  "id":"3416c03a-31df-4103-b446-358f6790af3e",
  "name":"search",
  "createdAt":"2016-03-09T20:01:48Z",
  "permissions":[
    {"methods":["GET"],"path":"/query-pipelines/*/collections/*/select"},
    {"methods":["GET"],"path":"/query-pipelines"},
    {"methods":["GET"],"path":"/solr/*/schema"},
    {"methods":["GET"],"path":"/prefs/apps/search/*"},
    {"methods":["GET"],"path":"/collections/**"},
    {"methods":["GET"],"path":"/solr/*/admin/luke"}
  ],
  "uiPermissions":[
    "search",
    "collections"
  ],
  "desc":"Provides read-only/required permissions for the Fusion Search UI."
}
```

15.30. Scheduler API

The scheduler REST API is used to define a schedule for system activities and manage the jobs that result from the schedule.

All of the Fusion services are available for scheduling, as are any Solr activities and any other HTTP-based URI.

See Schedules for more information about all of the options available for scheduling, and instructions for configuring scheduler jobs using the Fusion UI.

15.30.1. Manage Schedules: List, Create, Update, Delete

The path for this request is:

`/api/apollo/scheduler/schedules/<id>`

where <*id*> is the name of a specific schedule.

- GET - returns the definition of a specific schedule, or all defined schedules, if the schedule name is omitted.
- POST - creates a new schedule. The schedule name is included in the definition in the request body.
- PUT - modifies an existing schedule.
- DELETE - removes the schedule.

15.30.2. Stop or List Running Scheduled Jobs

The path for this request is:

`/api/apollo/scheduler/jobs/<id>`

where <*id*> is the name of a specific schedule.

A GET request will list the specified job, or all jobs if the ID is omitted.

A DELETE will stop the defined job, or all jobs if the ID is omitted.

15.30.3. Get Job History

This is a request to the History API. The path for this request is:

`/api/apollo/history/scheduler/items/<id>`

where <*id*> is the name of a scheduled job.

A GET request will list the history of 50 runs of the specified job. If the job is omitted, this will return a list of the IDs for all scheduled jobs.

To see a history of past jobs, you can use the History API.

15.30.4. Schedule Definition Properties

Parameter	Description
id *Required*	The ID of the schedule. This is required when creating a schedule with a POST request.
creatorType *Optional*	The type of user that created the schedule. If you have resources creating schedules programmatically, you could define a type that helps distinguish those schedules from others created by people.
creatorId *Optional*	An ID for the user that created the schedule.
startTime *Optional*	A time when the schedule should be started. If this is not set, it will be set to the date/time when the schedule was created.
endTime *Optional*	A time when the schedule should be terminated, i.e., when it will not run any more in the future. If not set, the schedule will run until it is disabled (i.e., "active" is set to false) or deleted.
repeatUnit *Optional*	A unit of time that when combined with the interval property defines how often the schedule will run. Allowed values are: • "millisecond" or "ms" • "second" or "sec" • "minute" or "min" • "hour" or "hr" • "day" • "week" • "month" These time units are not case sensitive.
interval *Optional*	An integer that when combined with the repeatUnit property defines how often the schedule will run. If this is not set, or set to a number lower than '1' (i.e., '0'), the schedule will only be run once.
active *Optional*	If **true**, the schedule will be executed according to the defined interval. If **false**, the schedule will be disabled and will not run at the defined time.

Parameter	Description
callParams *Required*	The callParams define the API call to execute at the defined time intervals. It allows several properties: • **uri**: a fully-qualified service URI. This can be an HTTP call, a Solr request, or a Fusion service call. Supported URI schemas are: ◦ `http://` or `https://` ◦ `solr://{collection}/…` A SolrCloud request. ◦ `service://{serviceName}/{path}` A load-balanced Fusion service request. • **method**: The method to use, as in GET, POST, PUT, or DELETE. • **queryParams**: query parameters to be passed with the request. For Solr calls, this could be parameters such as "q", "fq", "commit", etc. • **headers**: any additional protocol headers, such as "Content-type". • **entity**: an optional payload to be sent with the request.

15.30.5. Examples

Run a crawl of the datasource "LocalDocs2" every day:

REQUEST

```
curl -u user:pass -X POST -H 'Content-Type: application/json' -d '{"id":"1", "creatorType":"human",
"creatorId":"admin1", "repeatUnit":"DAY", "interval":1, "active":true,
"callParams":{"uri":"service://connectors/jobs/LocalDocs2", "method":"POST"}}'
http://localhost:8764/api/apollo/scheduler/schedules
```

RESPONSE

```
{
  "id" : "1",
  "trigger" : "1_1400007488512_-9223372036854775808",
  "schedule" : {
    "id" : "1",
    "creatorType" : "human",
    "creatorId" : "admin1",
    "createTime" : "2014-05-13T18:58:08.512Z",
    "startTime" : "2014-05-13T18:58:08.512Z",
    "endTime" : null,
    "repeatUnit" : "DAY",
    "interval" : 1,
    "active" : true,
    "callParams" : {
      "uri" : "service://connectors/jobs/LocalDocs2",
      "method" : "POST",
      "queryParams" : { },
      "headers" : { },
      "entity" : null
    }
  }
}
```

Issue a commit to the SolrCloud collection "demo" every hour:

REQUEST

```
curl-u user:pass -X POST -H 'Content-Type: application/json' -d '{"id":"2", "creatorType":"human",
"creatorId":"admin1", "repeatUnit":"HOUR", "interval":1, "active":true,
"callParams":{"uri":"solr://demo/update", "method":"GET", "queryParams":{"stream.body":"<commit/>"}}}'
http://localhost:8764/api/apollo/scheduler/schedules
```

RESPONSE

```
{
  "id" : "2",
  "trigger" : "2_1400011854443_-9223372036854775807",
  "schedule" : {
    "id" : "2",
    "creatorType" : "human",
    "creatorId" : "admin1",
    "createTime" : "2014-05-13T20:10:54.443Z",
    "startTime" : "2014-05-13T20:10:54.443Z",
    "endTime" : null,
    "repeatUnit" : "HOUR",
    "interval" : 1,
    "active" : true,
    "callParams" : {
      "uri" : "solr://demo/update",
      "method" : "GET",
      "queryParams" : {
        "stream.body" : "<commit/>"
      },
      "headers" : { },
      "entity" : null
    }
  }
}
```

Set schedule "2" to inactive:

REQUEST

```
curl -u user:pass -X PUT -H 'Content-Type: application/json' -d '{"creatorType":"human", "creatorId":"admin1",
"repeatUnit":"HOUR", "interval":1, "active":false, "callParams":{"uri":"solr://demo/update", "method":"GET",
"queryParams":{"stream.body":"<commit/>"}}}' http://localhost:8764/api/apollo/scheduler/schedules/2
```

There will be no response if the PUT request was successful.

List all scheduled jobs:

REQUEST

```
 curl -u user:pass http://localhost:8764/api/apollo/scheduler/jobs
```

RESPONSE

```
[ {
  "id" : "2",
  "trigger" : "2_1400011854443_-9223372036854775807",
  "schedule" : {
    "id" : "2",
    "creatorType" : "human",
    "creatorId" : "admin1",
    "createTime" : "2014-05-13T20:10:54.443Z",
    "startTime" : "2014-05-13T20:10:54.443Z",
    "endTime" : null,
    "repeatUnit" : "HOUR",
    "interval" : 1,
    "active" : true,
    "callParams" : {
      "uri" : "solr://demo/update",
      "method" : "GET",
      "queryParams" : {
        "stream.body" : "<commit/>"
      },
      "headers" : { },
      "entity" : null
    }
  }
}, {
  "id" : "1",
  "trigger" : "1_1400007488512_-9223372036854775808",
  "schedule" : {
    "id" : "1",
    "creatorType" : "human",
    "creatorId" : "admin1",
    "createTime" : "2014-05-13T18:58:08.512Z",
    "startTime" : "2014-05-13T18:58:08.512Z",
    "endTime" : null,
    "repeatUnit" : "DAY",
    "interval" : 1,
    "active" : true,
    "callParams" : {
      "uri" : "service://connectors/jobs/LocalDocs2",
      "method" : "POST",
      "queryParams" : { },
      "headers" : { },
      "entity" : null
    }
  }
} ]
```

List the history of job ID '1':

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/history/scheduler/items/1
```

RESPONSE

```
{
 "events" : [ {
 "start" : "2014-05-16T15:34:49.008Z",
 "end" : "2014-05-16T15:34:49.435Z",
 "source" : "scheduler",
 "type" : "execute",
 "status" : "ok",
 "details" : {
 "status" : 200,
 "entity" : "{\n \"id\" : \"TwitterSearch\",\n \"dataSourceId\" : \"TwitterSearch\",\n \"state\" :
\"RUNNING\",\n \"message\" : null,\n \"startTime\" : 1400254489000,\n \"endTime\" : -1,\n \"finished\" :
false,\n \"counters\" : { },\n \"exception\" : null,\n \"running\" : true\n}"
 },
 "error" : null
 }, {
 "start" : "2014-05-16T15:38:32.536Z",
 "end" : "2014-05-16T15:38:32.559Z",
 "source" : "scheduler",
 "type" : "execute",
 "status" : "ok",
 "details" : {
 "status" : 200,
 "entity" : "{\n \"id\" : \"TwitterSearch\",\n \"dataSourceId\" : \"TwitterSearch\",\n \"state\" :
\"RUNNING\",\n \"message\" : null,\n \"startTime\" : 1400254712000,\n \"endTime\" : -1,\n \"finished\" :
false,\n \"counters\" : { },\n \"exception\" : null,\n \"running\" : true\n}"
 } ]
}
```

15.31. Search Cluster API

The search cluster API allows users to connect Fusion with any existing Solr instances that are already present.

Once the Solr cluster is registered with Fusion, requests can be proxied through Fusion to it. The possible requests include search requests, but they can also be content indexing requests, such as the content crawled with a connector.

Once the searchCluster has been configured, the user can create Fusion collections that refer to the Solr collections that have been previously defined.

15.31.1. Manage Search Cluster Definitions: List, Create, Update, Delete

The path for this request is:

`/api/apollo/searchCluster/<id>`

where <*id*> is the name of a specific search cluster.

- GET - returns the definition of a specific cluster, or all defined clusters, if the cluster name is omitted.
- POST - creates a new search cluster definition.
- PUT - modifies an existing serach cluster definition.
- DELETE - removes the search cluster definition.

15.31.2. Get System Information and Node Status for a Specific Search Cluster

The path for this request is one of:

`/api/apollo/searchCluster/<id>/systemInfo`

`/api/apollo/searchCluster/<id>/nodes`

where <*id*> is the name of a specific search cluster.

A GET request provides information about a specific search cluster. The systemInfo request returns details such as the JVM version, the OS, free space available, etc. A nodes request returns the status of each node of the cluster.

A systemInfo request takes the following parameters:

Parameter	Description
nodeName	When defined with the name of a specific node, such as "nodeName=10.0.1.11:7574_solr", system information will be returned for the named node only.
all	When defined as "all=true", system information for every node of the cluster will be returned.

For the 'nodes' endpoint, the output will include the node name, the baseURL for the node and it’s current state as known by ZooKeeper.

For the 'systemInfo' endpoint, the output will include detailed system information such as the Lucene and Solr versions, the JVM version and memory settings, the OS and version, the available server memory and disk space, the system load and other similar settings.

15.31.3. Search Cluster Definition Properties

Property	Description
id *Required*	The ID of the search cluster. This is only required when creating a new cluster definition with a POST request.
connectString *Required*	The string to use to connect to the existing Solr cluster or standalone instance. If the existing Solr is running in SolrCloud mode, use the connect string for the ZooKeeper ensemble. If the existing Solr is running as a standalone instance, use the full URL for the Solr instance.
cloud *Required*	Defines if the "cluster" being defined is a SolrCloud cluster (**true**) or a standalone Solr instance (**false**).
bufferCommitWithin *Optional*	Defines a commitWithin property for the buffer when writing to this cluster. If not defined, the system will default to 10,000 milliseconds.
bufferFlushInterval *Optional*	Defines how often to flush the buffer when writing to this cluster. If not defined, the system will default to 1000 milliseconds.
bufferSize *Optional*	Defines the size of the buffer. If not defined, the system will default to 100 items in the buffer.
concurrency *Optional*	Defines the maximum number of concurrent /parallel requests to Solr servers when Fusion index pipeline Solr Indexer stage has property bufferDocsForSolr set to true.
zkClientTimeout *Optional*	The maximum amount of time to wait when communicating with the ZooKeeper ensemble for a SolrCloud instance.
zkConnectTimeout *Optional*	The maximum amount of time to wait when trying to connect to the ZooKeeper ensemble for a SolrCloud instance.

15.31.4. Examples

Create a new search cluster that is an existing SolrCloud cluster:

REQUEST

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d '{"id":"mySolrCluster",
"connectString":"10.0.1.6:5001,10.0.1.6:5002,10.0.1.6:5003", "cloud":true}'
http://localhost:8764/api/apollo/searchCluster
```

RESPONSE

```
{
  "id" : "mySolrCluster",
  "connectString" : "10.0.1.6:5001,10.0.1.6:5002,10.0.1.6:5003",
  "cloud" : true,
}
```

Create a 'cluster' that is a standalone Solr instance:

REQUEST

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d '{"id":"myOtherSolrCluster",
"connectString":"http://localhost:8983/solr", "cloud":false}' http://localhost:8764/api/apollo/searchCluster
```

RESPONSE

```
{
  "id" : "myOtherSolrCluster",
  "connectString" : "http://localhost:8983/solr",
  "cloud" : false,
}
```

Show the status of each node of 'mySolrCluster':

REQUEST

```
curl http://localhost:8764/api/apollo/searchCluster/mySolrCluster/nodes
```

RESPONSE

```
[ {
  "name" : "10.0.1.11:7574_solr",
  "baseUrl" : "http://10.0.1.11:7574/solr",
  "state" : "active"
}, {
  "name" : "10.0.1.8:7574_solr",
  "baseUrl" : "http://10.0.1.8:7574/solr",
  "state" : "active"
} ]
```

Show the system information for one named node:

REQUEST

```
curl http://10.0.1.8:8764/api/apollo/searchCluster/mySolrCluster/systemInfo?nodeName=10.0.1.8:7574_solr
```

RESPONSE

```
{
  "10.0.1.8:7574_solr" : {
    "mode" : "solrcloud",
    "lucene" : {
      "solr-spec-version" : "4.8.0",
      "lucene-spec-version" : "4.8.0"
    },
    "jvm" : {
      "version" : "1.8.0_121 25.121-b13",
      "name" : "Oracle Corporation Java HotSpot(TM) 64-Bit Server VM",
      "processors" : 4,
      "memory" : {
        "raw" : {
          "free" : 66736272,
          "total" : 204800000,
          "max" : 204800000,
          "used" : 138063728,
          "used%" : 67.4139296875
        }
      }
    },
    "system" : {
      "name" : "Mac OS X",
      "version" : "10.9.3",
      "arch" : "x86_64",
      "systemLoadAverage" : 2.130859375,
      "committedVirtualMemorySize" : 2963378176,
      "freePhysicalMemorySize" : 9321914368,
      "freeSwapSpaceSize" : 1073741824,
      "processCpuTime" : 313176000000,
      "totalPhysicalMemorySize" : 17179869184,
      "totalSwapSpaceSize" : 1073741824,
      "openFileDescriptorCount" : 208,
      "maxFileDescriptorCount" : 10240,
      "uname" : "Darwin MacMini.local 13.2.0 Darwin Kernel Version 13.2.0: Thu Apr 17 23:03:13 PDT 2014;
root:xnu-2422.100.13~1/RELEASE_X86_64 x86_64\n",
      "uptime" : "15:48  up 3 days,  7:08, 7 users, load averages: 2.13 2.01 1.91\n"
    }
  }
}
```

15.32. Sessions API

The session API is used to create sessions using defined realms, such as LDAP.

A session can be saved into a cookies file that can be re-used for subsequent requests. Sessions time out after 10 minutes of no activity, or after 8 hours.

15.32.1. Create a Session

The path for this request is:

`/api/session?realmName=<realmName>`

where the query parameter *realmName* takes as its value the name of a realm to authenticate against.

Input

Parameter	Description
username *Required*	The username to use in authentication.
password *Required*	The password to use in authentication.

Output

The output will include a cookie ID in the HTTP response header. This can be saved to a file and re-used with subsequent REST API requests.

Examples

Note	Use port 8765 in local development environments only. In production, use port 8764.

Create a session against an LDAP server and store it in a file named 'cookies':

REQUEST

```
curl -c cookies -i -H "content-type:application/json" -X POST -d '{"username":"myUser",
"password":"myPassword"}' http://localhost:8764/api/session?realmName=myLDAP
```

RESPONSE

```
HTTP/1.1 201 Created
Set-Cookie: id=840a33d4-b650-49f2-87a4-85412e99b37c;HttpOnly;Path=/api
Content-Length: 0
Server: Jetty(9.1.4.v20140401)
```

Note	In this case, we got a response because we set `curl` to include the HTTP in the output. Otherwise, we would not know for sure the session was created.

Use the cookie in another cURL request to see all collections:

```
curl -b cookies http://localhost:8764/api/apollo/collections
```

15.33. Signals Aggregator API

The Signals Aggregator API is used to aggregate signal events, which allows faster querying for recommendations. To use recommendations, signals need to be recorded and then aggregated.

When signals are enabled for a collection, two system-level collections are created. The first is named `collection_signals`, where `collection` is the sibling collection name, and signal events are indexed to this collection. The second is named `collection_signals_aggr`, and is the default location for aggregated signal events. See Signals API for more information on how to index signal events.

The aggregation process creates tuples for the fields selected when creating the aggregator job. A default tuple is applied if none is specified.

The aggregation process can remove the raw signals if desired, or keep them for other aggregation jobs.

15.33.1. Create, Update, Delete or List an Aggregator Job

The path for this request is:

`/api/apollo/aggregator/aggregations/<id>`

where <*id*> is the ID of an aggregation job.

A GET request will list the properties for an aggregator job with a provided ID or all defined aggregation jobs by ID if the path doesn't include a specific job ID.

POST requests with parameters define jobs, and PUT allows replacing existing job properties. Note that the PUT request will replace the existing job definitions with the new properties submitted with the request. Any properties not included with the PUT request will be replaced (possibly as 'null' if omitted.)

DELETE will remove the aggregator job.

Aggregator Job Definition Properties

Parameter	Description
time	A timestamp for signals to aggregate, used when starting the aggregation job.
rows	Defines the size of event batches to retrieve.
sync	If set to **true**, the aggregation job will run in the foreground and will restrict any other aggregation jobs from running until it is complete. The default is **false**, which also requires you to poll the job for status.

The response to either a POST or a GET request includes the properties of the job and the current status. Whan a job is started, the output includes the output_collection, the dates that will be used, and the collection being used for the data.

15.33.2. Start or Check an Aggregator Job

The path for this request is:

`/api/apollo/aggregator/jobs/<collectionName>_signals/<id>`

where *<id>* is the ID of an aggregation job and *<collectionName>_signals* is the signals collection that contains the events to be aggregated.

A POST request will start the specified job, while a GET request will check the job status.

15.33.3. Get Aggregator Job History

The path for this request is one of:

`/api/apollo/history/aggregator/items`

`/api/apollo/history/aggregator/items/<id>`

where *<id>* is the ID of an aggregation job.

If the ID is specified, a GET request will return the history for the defined aggregator job. If the ID is omitted, a GET request will return a list of all aggregation job IDs.

Input

None.

Output Content

The output will include information about the job including when it started and ended, how many signals were processed, and the details of the job properties.

15.33.4. Signals Aggregator Definitions Properties

Parameter	Description
id *Optional*	A unique identifier for this aggregator job.
groupingFields *Optional*	The fields that define unique tuples. The fields list is defined as a JSON array, with commas between each field name. If a set of fields is not defined, then a default tuple `'doc_id_s','query_s','filters_s'` will be used.
signalTypes *Optional*	The types of signals to aggregate. The type list is defined as a JSON array, with commas between each type. The types must be existing types used for events in your signals collection.

Parameter	Description
aggregator *Optional*	The name of the aggregator implementation. If it is not defined, this will default to **click** , which is an implementation optimized for aggregating signals based on user clicks. Aggregated records from this implementation will include a 'weight_d' field which can be used in boosting clicked documents. If you are not aggregating user click events, you can choose **simple**. This implementation does not add a 'weight_d' field to each record. A third option is **special** is described in more detail in page Aggregator Scripting.
selectQuery *Optional*	Any query to identify signal events.
timeRange *Optional*	A valid range query to select events to aggregate.
sort *Optional*	Specifies ordering of raw signal events within an aggregation. The default ordering is by event id ("id asc"). It can be set to use other fields using the standard Solr sort expressions, e.g. "timestamp_dt asc", also multiple criteria separate by comma, e.g. "type_s asc,timestamp_dt desc". Note: the sorting by "id asc" is always appended as the last sort criteria in order to break ties.
outputPipeline *Optional*	The name of a pipeline to use for processing aggregating events.
outputCollection *Optional*	The collection in which to store the aggregated events.
rollupPipeline *Optional*	The pipeline to use for rollups.
rollupAggregator *Optional*	The name of the aggregator implementation to use for rollups.
sourceRemove *Optional*	If **true**, then signal events that have been aggregated will be removed from the index. The default is **false**.

Parameter	Description
sourceCatchup *Optional*	If **true**, the original time range of the aggregation will be modified to span only the period since the last successful aggregation. The default is **false**.
outputRollup *Optional*	If **true**, the default, after performing the source data aggregation an additional aggregation step will be executed to roll-up the new aggregates with old aggregates that exist in the output collection for the same aggregation type.
aggregates *Optional*	A list of aggregation functions. Since it's possible to pass side-effects from one function to a later function in the list, the functions should be declared in the desired order of execution. The available aggregator functions are described in more detail in the section Aggregator Functions.
params *Optional*	The params allows defining aggregation job parameters. The most common use of this property is to define JavaScript scripts to customize the aggregator behavior. See the section Aggregator Scripting for more details.

Note that for large aggregation definitions, you could create a .json formatted file with the desired properties and upload it with cURL's `-d` parameter.

No output is returned when creating or updating an aggregator job.

When a job is listed, the properties returned are the same as the possible properties when defining a job.

15.33.5. Examples

Create an aggregator job for the click type of signals, with an aggregate function to provides counts by the id field:

REQUEST

```
curl -u user:pass -X POST -H 'Content-Type: application/json' -d '{"id":"1", "signalTypes":["click"],
"aggregates":[{"type":"count", "sourceFields":["id"], "targetField": "count_d"}]}'
http://localhost:8764/api/apollo/aggregator/aggregations
```

RESPONSE

None.

Update the properties for aggregator job '1', including all the original properties plus the ones we want to add or change:

REQUEST

```
curl -u user:pass -X PUT -H 'Content-Type: application/json' -d '{"signalTypes":["click"], "timeRange":"[NOW/-
1 TO NOW]", "aggregates":[{"type":"count", "sourceFields":["id"], "targetField": "count_d"}]}'
http://localhost:8764/api/apollo/aggregator/aggregations/1
```

RESPONSE

None.

List the properties for aggregator job '1':

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/aggregator/aggregations/1
```

RESPONSE

```
{
  "id" : "1",
  "groupingFields" : [ ],
  "signalTypes" : [ "click" ],
  "timeRange" : "[NOW/-1 TO NOW]",
  "sourceRemove" : false,
  "sourceCatchup" : false,
  "outputRollup" : false,
  "aggregates" : [ {
    "type" : "count",
    "sourceFields" : [ "id" ],
    "targetField" : "count_d",
    "params" : { }
  } ],
  "params" : { }
}
```

Start job '1' on the 'demo_signals' collection:

REQUEST

```
curl -u user:pass -X POST http://localhost:8764/api/apollo/aggregator/jobs/demo_signals/1
```

RESPONSE

The following output has been truncated to omit the aggregation job definition and only shows the other job properties that are returned on start.

```
{
  "signals" : {
    "types" : [ "click" ],
    "stats" : { }
  },
  "state" : "running",
  "job_id" : "4d69ec73358b41d38caf1eb3b378809e",
  "aggregation_time_date" : "2014-09-11T16:39:58.347Z",
  "aggregation" : {
    "id" : "r1",
    "groupingFields" : [ "doc_id_s", "query_s", "filters_s" ],
    "signalTypes" : [ "click" ],
    "selectQuery" : "*:*",
...
  "output_collection" : "bestbuy_signals_aggr",
  "NOW" : 1410453598347,
  "NOW_date" : "2014-09-11T16:39:58.347Z",
  "collection" : "bestbuy_signals",
  "aggregation_time" : 1410453598347,
  "compound_id" : "bestbuy_signals:r1"
}
```

See the list of aggregator job items:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/history/aggregator/items
```

RESPONSE

```
[ "demo_signals:1" ]
```

Get the history of job "demo_signals:1":

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/history/aggregator/items/demo_signals:1
```

RESPONSE

```
{
  "events" : [ {
    "start" : "2014-04-16T20:45:16.582Z",
    "end" : "2014-04-16T20:45:16.781Z",
    "source" : "demo_signals:1",
    "type" : "run",
    "status" : "ok",
    "details" : {
      "signals" : {
        "click" : {
          "state" : "finished",
          "raw" : 2,
          "aggr_type_s" : "click",
          "aggr_class" : "com.lucidworks.apollo.service.aggregation.ClickSignalAggregator",
          "aggregated" : 2
        }
      },
      "state" : "finished",
      "job_id" : "467bc0db-a9c9-4b48-8080-439958818907",
      "aggregation_time_date" : "2014-04-16T20:45:16.556Z",
      "aggregation" : {
        "id" : "1",
        "fields" : [ "doc_id_s", "query_s", "filters_s" ],
        "types" : [ "click" ],
        "select" : "*:*",
        "range" : "[* TO NOW]",
        "remove" : false,
        "rolling" : false,
        "params" : { },
        "anyAggr" : false
      },
      "NOW" : 1397681116556,
      "commit" : "done",
      "NOW_date" : "2014-04-16T20:45:16.556Z",
      "collection" : "demo_signals",
      "aggregation_time" : 1397681116556,
      "compound_id" : "demo_signals:1"
    },
    "error" : null
  } ]
}
```

15.34. Signals API

The Signals API accepts a set of signals, encoded as JSON objects, for indexing into a signals collection.

Normally, signals are indexed just like ordinary documents, through a configured datasource and index pipeline. This API is provided for cases where it is more convenient to index signals directly.

To aggregate the signals, see the Signals Aggregator API.

15.34.1. Index a Signal

The REST API endpoint which ingests raw signal data in JSON format is:

`/api/apollo/signals/<collectionName>`

where <*collectionName*> is the name of an specific collection.

A POST request sends one or more signals to be indexed.

This request takes several optional query parameters:

Parameter	Description
pipeline Optional	An optional parameter to define a specific index pipeline to be used while indexing these signals. If none is defined, the pre-configured 'signals_ingest' pipeline will be used. Note that any index pipeline used must include a solr-index stage at the end (to index the documents to Solr), and in this stage the 'enforce_schema' property must be set to true.
commit	If **true**, then a Solr commit will occur at the end of indexing.
async	If **true**, then signals will be indexed in asynchronous mode, which issues an autoCommit and does not report failures. The default is **false**.

15.34.2. Signal document structure

A raw signal is stored as a Solr document with the following fields, which are derived from the raw signal as follows:

Field	Description
`id` *Optional*	The signal ID. If no ID is supplied, one will be automatically generated.

Field	Description
`type` *Required*	The signal type that is being sent. This value is used during aggregation to filter events of the same type. Types can be mixed in aggregation jobs, if needed. The type can consist of any string you choose. For consistency, always send events of the same type with the same type value. During indexing, type values will be moved to a field named `type_s`.
`params` *Optional*	The params allow flexible definition of the fields you care about and will use later for signal aggregation: • `docId` – A unique document ID This is stored in the Solr raw signal document as field `doc_id_s`. • `userId` – A unique user ID This is stored in the Solr raw signal document as field `user_id_s`. • `query` – A query string; for example, a user's search This is copied to the Solr raw signal document as both fields `query_s` and `query_t`. Some cleanup occurs to convert the string to lowercase, decode URL encoding, and replace white space with single space characters. The original query is saved in field `query_orig_s`. • `filterQueries` – A list of strings, such as filters on the search query This is copied to the Solr raw signal document as both `filters_s` and `filters_orig_ss`. • `collection` – The primary collection name • `weight` – A float value representing the relative weight of this signal This is saved in the field `weight_d`. • `count` – A positive integer value representing the incremented count of signals This is saved in the field `count_i`.

Field	Description
timestamp	The timestamp of the signal event. • When using the Signals API, this property is optional; it defaults to the current server time. • When using the Signal Formatter index stage, one of the following fields must be present: timestamp, timestamp_tdt, timestamp_dt, or epoch.

Here is the JSON representation of one click signal, taken from an example dataset of synthetic clickstream data:

```
{ "params": {
      "docId": "2125233",
      "filterQueries": ["cat00000","abcat0100000", "abcat0101000", "abcat0101001"],
      "query": "Televisiones Panasonic  50 pulgadas" }
 "type":"click",
 "timestamp": "2011-09-01T23:44:52.533000Z",
}
```

15.34.3. Examples

Send two signal events to record user clicks:

REQUEST

```
curl \
 -u user:pass -X POST -H 'Content-type:application/json' -d @- \
http://localhost:8764/api/apollo/signals/docs?commit=true \
<<EOF
[
 {"params": {
      "query": "Televisiones Panasonic  50 pulgadas",
      "filterQueries": ["cat00000","abcat0100000", "abcat0101000", "abcat0101001"],
      "docId": "2125233" },
 "type":"click",
 "timestamp": "2011-09-01T23:44:52.533000Z"
 },
 {"params": {
       "query": "Sharp",
       "filterQueries": ["cat00000", "abcat0100000", "abcat0101000", "abcat0101001"],
       "docId": "2009324" },
  "type":"click",
  "timestamp": "2011-09-05T12:25:37.420000Z"
 }
]
EOF
```

A successful request results in events being added to the signals collection. For the above example, the events will be represented as follows:

```
{
 "responseHeader":{
   "status":0,
   "QTime":1,
   "params":{
     "indent":"true",
     "q":"doc_id_s: 2125233",
     "wt":"json"}
},
 "response":{"numFound":1198,"start":0,"docs":[
     {
       "id": "7aee7b1f-5cde-4957-b73c-c15881f559ec",
       "filters_s": "abcat0100000 $ abcat0101000 $ abcat0101001 $ cat00000",
       "query_orig_s": "Televisiones Panasonic  50 pulgadas",
       "params.user_s": "000000df17cd56a5df4a94074e133c9d4739fae3",
       "doc_id_s": "2125233",
       "params.query_time__s": "2011-09-01T23:43:59.752Z",
       "query_t": "televisiones panasonic 50 pulgadas",
       "query_s": "televisiones panasonic 50 pulgadas",
       "filters_orig_ss": [
         "abcat0100000",
         "abcat0101000",
         "abcat0101001",
         "cat00000"
       ],
       "flag_s": "EVENT",
       "type_s": "click",
       "attr_params.filterQueries_": [
         "cat00000",
         "abcat0100000",
         "abcat0101000",
         "abcat0101001"
       ],
       "timestamp_dt": "2011-09-01T23:44:52.533Z",
       "_version_": 1478892846857584600
     },
     {
       "id": "6789a209-f5b5-457e-9df6-8033b8f7f317",
       "filters_s": "abcat0100000 $ abcat0101000 $ abcat0101001 $ cat00000",
       "query_orig_s": "Sharp",
       "params.user_s": "00000192816224 7ffaf63185cd8b2a244c78e7c6",
       "doc_id_s": "2009324",
       "params.query_time__s": "2011-09-05T12:25:01.187Z",
       "query_t": "sharp",
       "query_s": "sharp",
       "filters_orig_ss": [
         "abcat0100000",
         "abcat0101000",
         "abcat0101001",
         "cat00000"
       ],
       "flag_s": "EVENT",
       "type_s": "click",
       "attr_params.filterQueries_": [
         "cat00000",
         "abcat0100000",
         "abcat0101000",
         "abcat0101001"
```

```
        ],
        "timestamp_dt": "2011-09-05T12:25:37.42Z",
        "_version_": 1478892846859681800
      }
    ]
  }
}
```

15.35. Solr API

The Solr API is used to manage collection-level configurations.

The path for this request is:

`/api/apollo/solr/<collectionName>/<solrRequest>`

where <*collectionName*> is the name of an specific collection and <solrRequest> is the Solr command you wish to run.

Since this API proxies requests to Solr, each available method corresponds to the method in Solr. So, a GET request to Solr would use the GET method of this endpoint; a POST would use the POST method, etc.

Depending on the request, the response may consist of records that match a query or output from a Schema API request.

15.35.1. Examples

Query Solr for documents in the 'docs' collection containing the term 'solr', limiting the results to only 2 records, returning only the title, and in JSON format:

REQUEST

```
http://localhost:8764/api/apollo/solr/docs/select?q=solr&rows=2&fl=title&wt=json
```

RESPONSE

```
{
    "responseHeader": {
        "status": 0,
        "QTime": 2,
        "params": {
            "fl": "title",
            "q": "solr",
            "wt": "json",
            "rows": "2"
        }
    },
    "response": {
        "numFound": 52,
        "start": 0,
        "docs": [
            {
                "title": [
                    "Solr and SolrAdmin APIs - Fusion Documentation - Lucidworks"
                ]
            },
            {
                "title": [
                    "Search Clusters - Fusion Documentation - Lucidworks"
                ]
            }
        ]
    }
}
```

15.36. SolrAdmin API

The Solr Admin API lets you send commands to Solr through Fusion’s proxy service. This allows you to protect your Solr instances from outside connections, and apply roles and user permissions from Fusion when running Solr commands. Requests sent to this API are subject to access restrictions above the collection level.

Note that because one searchCluster may host several collections, it’s not recommended to use this with a collection-level command (such as a query, or document update).

The path for this request is:

`/api/apollo/solrAdmin/<searchCluster>/<solrRequest>`

where <*searchCluster*> is the ID of the searchCluster and <solrRequest> is the Solr command you wish to run, e.g., check status of the nodes.

15.36.1. Example

Request all CORE MBeans for the 'default' searchCluster, formatted in JSON:

REQUEST

```
http://localhost:8764/api/apollo/solrAdmin/default/admin/mbeans?cat=CORE&wt=json
```

RESPONSE

```
{
    "responseHeader": {
        "status": 0,
        "QTime": 1
    },
    "solr-mbeans": [
        "CORE",
        {
            "searcher": {
                "class": "org.apache.solr.search.SolrIndexSearcher",
                "version": "1.0",
                "description": "index searcher",
                "src": null
            },
            "core": {
                "class": "collection1",
                "version": "1.0",
                "description": "SolrCore",
                "src": null
            },
            "Searcher@435cf502[collection1] main": {
                "class": "org.apache.solr.search.SolrIndexSearcher",
                "version": "1.0",
                "description": "index searcher",
                "src": null
            }
        }
    ]
}
```

15.37. Solr Configuration API

The Solr Configuration REST API is used to access and manage the Solr configuration files stored in ZooKeeper.

To manage Solr synonym lists, use the Synonyms API.

To manage Solr lists of stopwords, use the Stopwords API.

15.37.1. List and View Configuration Files

The path for this request is:

`/api/apollo/collections/<collectionName>/solr-config`

`/api/apollo/collections/<collectionName>/solr-config/<path>`

where <collectionName> is the name of an specific collection and <path> is the name of a new or existing configuration file.

A GET request will retrieve either a list of all configuration files from ZooKeeper, if no path is specified. If a path or part of a path is specified, a GET request will return information on a single file or a directory.

If the "recursive" query parameter is used (below), this request will recurse through all subdirectories and files.

15.37.2. Create or Update a File in ZooKeeper

The path for this request is:

`/api/apollo/collections/<collectionName>/solr-config/<path>`

where <collectionName> is the name of an specific collection and <path> is the name of a new or existing configuration file.

A POST request will add a configuration file.

A PUT request will replace the configuration file.

15.37.3. Solr Configuration Definition Properties

Path Parameters

Parameter	Description
collection	The collection that contains the configuration files to list or view.
path	The path to a specific file or nested child nodes. If the file is not nested, the filename can be entered without any path information.

Query Parameters

Parameter	Description
expand	If true, the binary content of a file will be returned base64 encoded. If this is not included, only the metadata about each node will be returned. If you would like to see the content of the file in plain text, you can add 'Accept: text/plain' to the request header. Alternately, you can get the raw bytes by adding 'Accept: application/octet-stream' to the header.
recursive	If true, children of nested ZooKeeper nodes will be returned. This can be used in conjunction with the path to show only children of a specific node.

The output will include a list of each file with the following information:

- **name:** The name of the file or node.
- **version:** The file or node version from ZooKeeper.
- **isDir:** If the node has children, this value will be true.
- **value:** Only returned if "expand=true" is added as a query parameter. This will be returned as the base64 encoding of the contents of the file.

If the header includes 'Accept: text/plain' or 'Accept: application/octet-stream', metadata about the file will not be returned and only the content as either plain text or raw bytes.

If recursive=true and expand=true are both used in the same request, the request may be a little slow depending on how much data is requested.

15.37.4. ZooKeeper Collection Configuration Definition Properties

Path Parameters

Parameter	Description
collection	The collection that contains the configuration files to list or view.
path	The path to a specific file (including nested child nodes).

Query Parameters

Parameter	Description
reload	If true, the collection will be reloaded to make the changes available to Solr immediately.

Input Content

The REST API does not restrict the type of content that can be sent to ZooKeeper, but care should be taken to make sure the the format of the file remains compatible with Solr’s requirements.

As a best practice, all requests should include the Content-type in the request header.

15.37.5. Examples

Show _rest_managed.json for the 'docs' collection, with the base64 encoded content:

INPUT

```
curl -u user:pass http://localhost:8764/api/apollo/collections/docs/solr-config/_rest_managed.json?expand=true
```

OUTPUT

```
{
  "name" : "_rest_managed.json",
  "version" : 0,
  "isDir" : false,
  "value" : "eyJpbml0QXJncyI6e30sIm1hbmFnZWRMaXN0IjpbXX0NCg=="
}
```

Get the plain text version of 'synonyms.txt':

INPUT

```
curl -u user:pass -H 'Accept: text/plain' http://localhost:8764/api/apollo/collections/docs/solr-
config/synonyms.txt
```

OUTPUT

```
GB,gib,gigabyte,gigabytes
MB,mib,megabyte,megabytes
Television, Televisions, TV, TVs
pixima => pixma
```

Show the child nodes of 'clustering':

INPUT

```
curl -u user:pass http://localhost:8764/api/apollo/collections/docs/solr-config/clustering?recursive=true
```

OUTPUT

```
{
  "name" : "clustering",
  "version" : 0,
  "isDir" : true,
  "children" : [ {
    "name" : "carrot2",
    "parent" : "clustering",
    "version" : 0,
    "isDir" : true,
    "children" : [ {
      "name" : "kmeans-attributes.xml",
      "parent" : "clustering/carrot2",
      "version" : 0,
      "isDir" : false
    }, {
      "name" : "lingo-attributes.xml",
      "parent" : "clustering/carrot2",
      "version" : 0,
      "isDir" : false
    }, {
      "name" : "stc-attributes.xml",
      "parent" : "clustering/carrot2",
      "version" : 0,
      "isDir" : false
    } ]
  } ]
}
```

Replace the contents of the existing synonyms.txt file with the contents of a file named "updated-synonyms.txt" and reload the collection:

INPUT

```
curl -u user:pass -X PUT -H 'Content-type: text/plain' -d '@updated-synonyms.txt'
http://localhost:8764/api/apollo/collections/docs/solr-config/synonyms.txt?reload=true
```

OUTPUT

None.

15.38. Spark Jobs API

This is a set of endpoints for configuring and running Spark jobs.

15.38.1. Spark Cluster Information: `/spark/master`

Path	Method	Description
`/spark/master`	GET	Returns a string containing the Spark master URL
`/spark/master/status`	GET	Returns the detailed status of the Spark cluster started by Fusion.

15.38.2. Configure Spark Jobs: `/spark/configurations`

This endpoint is used to view or manage Spark job configuration objects. A configuration always consists of the following properties:

- id - a unique identifier for this object, used by Fusion to manage this object. If this property is not specified at object creation time, Fusion will supply one.
- Type - one of a set of defined job types: \{`aggregation`, `script`, `wordcount`, `item-similarity`, `quality`}.

Depending on the job type, further configuration properties are required, described in section Job Types and Configurations below.

Configurations are defined and stored using JSON. Here is an example of a configuration for a Spark job of type "wordcount":

```
{
  "id": "wordcount-1",
  "type": "wordcount",
  "file": "hdfs://localhost:9000/data/aol.json.gz"
}
```

This endpoint take an optional "id" parameter, used to restrict an operation to a specific configuration object. The body of a POST or PUT request is the JSON for the Spark job configuration.

Path	Method	Args	Description
`/spark/configurations`	GET	`?type=`	Returns a JSON list of defined job configurations. The value of the optional request parameter `type=` is a comma-separated list of one or more job type names. If present, the result is restricted to the specified job types.

Path	Method	Args	Description
`/spark/configurations`	POST	job config JSON	Create and return a new Spark job configuration in JSON. When creating a new job configuration if a unique job id is missing one will be created by Fusion and returned to the caller.
`/spark/configurations/{id}`	GET		Returns a defined Spark job configuration in JSON.
`/spark/configurations/{id}`	PUT	job config JSON	Update a defined Spark job configuration.
`/spark/configurations/{id}`	DELETE		Delete a defined Spark job configuration.

15.38.3. Manage Spark Jobs: `/spark/jobs`

The `/spark/jobs` endpoint manages all Spark jobs, including aggregation jobs over signals data, however these can be more efficiently managed using the more specific `/aggregator/jobs` API.

Path	Method	Args	Description
Path	Method	Args	Description
`/spark/jobs`	GET	`?type=`	Returns a listing of all active and recently finished jobs. Result is a JSON map from jobs to an object which lists both the job type and execution information. The value of the optional request parameter `type=` is a comma-separated list of one or more job type names. If present, the result is restricted to the specified job types.
`/spark/jobs`	POST	job JSON	Start an arbitrary job using the provided configuration. A unique id will be automatically assigned and returned to the caller. Returns current status of that job.
`/spark/jobs/{id}`	GET		Returns the detailed job status for a running or recently completed job.

Path	Method	Args	Description
`/spark/jobs/{id}`	POST	`?sync=true`	Start a job using an existing job configuration. Optionally execute it synchronously (blocking until the job is complete). Returns current status of that job.
`/spark/jobs/{id}`	DELETE		Stops a running job.

15.38.4. Job Types and Configurations

Job configurations are defined and stored using JSON. The required configuration information depends on the job type.

`aggregation`

This job type is supported by `/spark/jobs` API, and can be defined and managed using `/spark/configurations` API, but the dedicated `/aggregator/jobs` and `/aggregator/aggregations` API should be used in most cases because it provides additional support for managing jobs per collection, passing specific job parameters from requests, stricter parameter validation, etc.

`script`

This job type supports the execution of arbitrary Scala scripts. Results of computation can be returned to the caller as a part of job status, by assigning the result in the script to a variable called `result`.

Example configuration:

```
{
  "id": "scripted-1",
  "type": "script",
  "maxRows": 100,
  "script": "var rdd = sc.textFile(\"hdfs://localhost:9000/data/aol.json.gz\")\nrdd.count\nval result = rdd\n"
}
```

Example output:

```
{
  "state": "finished",
  "jobId": "d42260a6T153ebaed126",
  "jobConfig": {
    "id": "scripted-1",
    "script": "var rdd = sc.textFile(\"hdfs://localhost:9000/data/aol.json.gz\")\nrdd.count\nval result =
rdd\n",
    "maxRows": 100,
    "type": "script",
    "sync": "true",
    "submitted": 1459948015910
  },
  "result": {
    "result": [
      {
        "_": "{\"user_id_s\":\"142\",\"query_s\":\"westchester.gov\",\"timestamp_dt\":\"2006-03-
20T03:55:57.000Z\",\"params.position_s\":\"1\",\"doc_id_s\":\"http://www.westchestergov.com\"}"
      },
      {
        "_": "{\"user_id_s\":\"142\",\"query_s\":\"207 ad2d 530\",\"timestamp_dt\":\"2006-04-
08T01:31:14.000Z\",\"params.position_s\":\"1\",\"doc_id_s\":\"http://www.courts.state.ny.us\"}"
      },
        ...
    ],
    "state": "finished",
    "resultCode": "SUCCESS",
    "output": "rdd: org.apache.spark.rdd.RDD[String] = hdfs://localhost:9000/data/aol.json.gz
MapPartitionsRDD[1] at textFile at <console>:17\nres0: Long = 99999\nresult: org.apache.spark.rdd.RDD[String]
= hdfs://localhost:9000/data/aol.json.gz MapPartitionsRDD[1] at textFile at <console>:17\n"
  },
    ...
}
```

(Note: the entries in the "result" part of the job output are simply the content of RDD[String] where lines are flat Strings and not a real JSON. Users can use `sqlContext.read.json(…)` to read this file as a `DataFrame`).

`wordcount`

This is a simple version of the Unix `wc` utility implemented using Spark. It can open a text file defined as any data source supported by Spark `DataFrameReader`, and returns three numbers: line count, word count, and character count. Words are delimited by whitespace, and character count doesn't include the end-of-line characters.

Please note that this implementation uses Spark accumulators. When accumulators are used in transformations - as opposed to actions - they may count multiple times if some tasks should fail. As a consequence the result is not as reliable, but it's very inexpensive to calculate.

Example configuration:

```
{
  "id": "wordcount-1",
  "type": "wordcount",
  "file": "hdfs://localhost:9000/data/aol.json.gz"
}
```

Example output:

```
{
  "state": "finished",
  "jobId": "aadddc12T153e6b822d8",
  "jobConfig": {
    "file": "hdfs://localhost:9000/data/aol.json.gz",
    "id": "wordcount-1",
    "type": "wordcount",
    "sync": "true",
    "submitted": 1459864740568
  },
  "result": {
  "state": "finished",
  "data": [
    99999,
    282865,
    15888690
  ]
  },
    ...
}
```

`item-similarity`

Example configuration:

```
{
  "id": "item-similarity-1",
  "inputProperties": [
    {
      "rowFieldName": "user_id_s",
      "columnFieldName": "item_id_s",
      "collection": "test",
      "query": "type_s:purchase",
      "queryParams": [],
      "idField": "id"
    },
    {
      "rowFieldName": "user_id_s",
      "columnFieldName": "item_id_s",
      "collection": "test",
      "query": "type_s:view",
      "queryParams": [],
      "idField": "id"
    }
  ],
  "maxPrefs": 100,
  "maxSimilaritiesPerItem": 100,
  "outputProperties": {
    "rowFieldName": "item_id_s",
    "columnFieldName": "item_id_ss",
    "columnStrengthFieldName": "weight_ds",
    "collection": "test_signals_aggr",
    "pipelineName": "test_signals_aggr-default"
  }
}
```

Example output (trimmed):

```
{
  "state": "finished",
  "jobId": "8a4c0000T153eb71cba0",
  "jobConfig": {
  "inputProperties": [
    {
      "rowFieldName": "user_id_s",
      "columnFieldName": "item_id_s",
      "collection": "test",
      "query": "type_s:purchase",
      "queryParams": [],
      "idField": "id"
    },
    {
      "rowFieldName": "user_id_s",
      "columnFieldName": "item_id_s",
      "collection": "test",
      "query": "type_s:view",
      "queryParams": [],
      "idField": "id"
    }
  ],
  "maxPrefs": 100,
  "maxSimilaritiesPerItem": 100,
  "outputProperties": {
    "rowFieldName": "item_id_s",
    "columnFieldName": "item_id_ss",
    "columnStrengthFieldName": "weight_ds",
    "collection": "test_signals_aggr",
    "pipelineName": "test_signals_aggr-default"
  },
  "id": "item-similarity-1",
  "type": "item-similarity",
  "sync": "true",
  "submitted": 1459944016778
  },
  "result": {
  "state": "finished",
  "data": 101023
  },
...
}
```

Results of this job are sent to the output collection, and the returned data is the number of found co-occurrences.

`quality`

This job type calculates common search quality metrics, such as precision, recall, MRR, MAP and NDCG, for a collection or rankings. Ranking data may be represented as individual tuples of `(query, doc_id, position`, which naturally fits the format of data received as click-through events.

Example configuration:

```
{
  "id": "quality-1",
  "type": "quality",
  "inputA": {
    "format": "solr",
    "options": {
      "collection": "test"
    }
  },
  "nameField": "query_s",
  "itemField": "doc_id_s",
  "rankField": "position_i",
  "tagA": "id",
  "n": 10
}
```

Example output (trimmed):

```
{
  "state": "finished",
  "jobId": "be00034dT153eb7ee74c",
  "jobConfig": {
  "inputA": {
    "format": "solr",
    "options": {
      "collection": "test"
    }
  },
  "nameField": "query_s",
  "itemField": "doc_id_s",
  "rankField": "position_i",
  "tagA": "id",
  "n": 10,
  "id": "quality-1",
  "type": "quality",
  "sync": "true",
  "useCursorMark": true,
  "referenceTime": 1459944875828,
  "submitted": 1459944875846,
  "aggregationTime": 1459944875828,
  "inputCollection": "test",
  "rows": 10000
  },
  "result": {
  "summaryStats": {
    "map": {
      "count": 4943,
      "mean": 0.14667206150111256,
      "stdev": 0.1054289576185325,
      "variance": 0.011115265104530325,
      "max": 1,
      "min": 0.1,
      "sum": 725.0000000000091,
      "sumOfLogs": -10157.487088839782,
      "sumOfSquares": 161.28000000000247,
      "geoMean": 0.1281026291366323,
      "skewness": 3.7730477042414576,
      "kurtosis": 19.623389512667725
    },
    "mrr": { ... },
    "ndcg": { ... },
    "precision": { ... },
    "recall": { ... }
  },
  "state": "finished",
  "n": 10
  },
...
}
```

15.38.5. Spark job subtypes

For the Spark job type, the available subtypes are listed below.

ALS Recommender	Train a collaborative filtering matrix decomposition recommender using SparkML’s Alternating Least Squares (ALS) to batch-compute user recommendations and item similarities.
Aggregation	Define an aggregation job to be executed by Fusion Spark.
Co-occurrence Similarity	Compute a mutual-information item similarity model.
Random Forest Classifier Training	Train a random forest classifier for text classification.
Script	Run a custom Scala script as a Fusion Job.
Matrix Decomposition-Based Query-Query Similarity Job	Train a collaborative filtering matrix decomposition recommender using SparkML’s Alternating Least Squares (ALS) to batch-compute query-query similarities.
Bisecting KMeans Clustering Job	Train a bisecting KMeans clustering model.
Logistic Regression Classifier Training Job	Train a regularized logistic regression model for text classification.
Item Similarity Recommender	Compute user recommendations based on pre-computed item similarity model.
Levenshtein	Compare the items in a collection and produces possible spelling mistakes based on the Levenshtein edit distance.
Collection Analysis	Produce statistics about the types of documents in a collection and their lengths.
Statistically Interesting Phrases (SIP)	Output statistically interesting phrases in a collection, that is, phrases that occur more frequently or less frequently than expected.
Doc Clustering	An end-to-end document clustering job that preprocesses documents, separates out extreme-length documents and other outliers, automatically selects the number of clusters, and extracts keyword labels for clusters. You can choose between Bisecting KMeans and KMeans clustering methods, and between TFIDF and word2vec vectorization methods.
Outlier Detection	Find groups of outliers for the entire set of documents in the collection.
Cluster Labeling	Attach keyword labels to documents that have already been assigned to groups.

15.38.6. Spark Configuration Properties

Fusion passes all configuration properties with prefix "spark." to the Spark master, Spark worker and each Spark application, both for aggregation jobs and custom-scripted processing.

These properties are stored in Fusion’s ZooKeeper and can be updated via requests to Fusion endpoint `api/apollo/configurations` which will update the stored value without restarting the service, therefore existing jobs and SparkContexts will not be affected. The Fusion endpoint `api/apollo/configurations` returns all configured properties for that installation. You can examine spark default configurations in a Unix shell using the utilities `curl` and `grep`. Here is an example which checks a local Fusion installation running on port 8764:

```
curl -u username:password http://localhost:8764/api/apollo/configurations | grep '"spark.'

  "spark.executor.memory" : "2g",
  "spark.task.maxFailures" : "10",
  "spark.worker.cleanup.appDataTtl" : "7200",
  "spark.worker.cleanup.enabled" : "true",
  "spark.worker.memory" : "2g",
```

The default SparkContext that Fusion uses for aggregation jobs can be assigned a fraction of cluster resources (executor memory and/or available CPU cores). This allows other applications (such as scripted jobs, or shell sessions) to use the remaining cluster resources even when some aggregation jobs are running. Fusion 2.3 also permits dynamic allocation for all applications. This can be overridden per application. In practice, this means that even when there's an already running SparkContext with a relatively long idle time (eg. 10 minutes) but there are no active jobs that use it, its resources (CPU cores and executor memory) will be released for use by other applications.

For scripted Spark jobs, users can specify per-job configuration overrides as a set of key / value pairs in a "sparkConfig" property element of a script job configuration, which takes precedence over values stored in ZooKeeper. The following is an example of a scripted job with a "sparkConfig" section:

```
{
  "id": "scripted_job_example",
  "script": "val rdd = sc.textFile(\"/foo.txt\")\nrdd.count\n",
  "sparkConfig": {
    "spark.cores.max": 2,
    "spark.executor.memory": "1g"
  }
}
```

The following table lists those Spark configuration properties that Fusion overrides or uses in order to determine applications' resource allocations.

Property	Description
`spark.master.url`	By default, left unset. This property is only specified when using an external Spark cluster; when Fusion is using its own standalone Spark cluster, this property isn't set.
`spark.cores.max`	The maximum number of cores across the cluster assigned to the application. If not specified, there is no limit. The default is unset, i.e., an unlimited number of cores.
`spark.executor.memory`	Amount of memory assigned to each application's executor. The default is 2G.
`spark.scheduler.mode`	Controls how tasks are assigned to available resources. Can be either 'FIFO' or 'FAIR'. Default value is 'FAIR'.
`spark.dynamicAllocation.enabled`	Boolean - whether or not to enable dynamic allocation of executors. Default value is 'TRUE'.
`spark.shuffle.service.enabled`	Boolean - whether or not to enable internal shuffle service for standalone Spark cluster. Default value is 'TRUE'.

Property	Description
`spark.dynamicAllocation.executorIdleTimeout`	Number of seconds after which idle executors are removed. Default value is '60s'.
`spark.dynamicAllocation.minExecutors`	Number of executors to leave running even when idle. Default value is 0.
`spark.eventLog.enabled`	Boolean - whether or not event log is enabled. Event log stores job details and can be accessed after application finishes. Default value is 'TRUE'.
`spark.eventLog.dir`	Directory which stores event logs. Default location is `fusion/var/spark-eventlog`.
`spark.eventLog.compress`	Boolean - whether or not to compress event log data. Default value is 'TRUE'.
`spark.logConf`	Boolean - whether or not to log effective SparkConf of new SparkContext-s. Default value is 'TRUE'.
`spark.deploy.recoveryMode`	Default value is 'ZOOKEEPER'
`spark.deploy.zookeeper.url`	ZooKeeper connect string. Default value is `$FUSION_ZK`
`spark.deploy.zookeeper.dir`	ZooKeeper path, default value is `/lucid/spark`
`spark.worker.cleanup.enabled`	Boolean - whether or not to periodically cleanup worker data. Default value is 'TRUE'.
`spark.worker.cleanup.appDataTtl`	Time-to-live in seconds. Default value is 86400 (24h).
`spark.deploy.retainedApplications`	The maximum number of applications to show in the UI. Default value is 50.
`spark.deploy.retainedDrivers`	The maximum number of drivers. Default value is 50.
`spark.worker.timeout`	The maximum timeout in seconds allowed before a worker is considered lost. The default value is 30.
`spark.worker.memory`	The maximum total heap allocated to all executors running on this worker. Defaults to value of the executor memory heap.

15.38.7. Fusion Configuration Properties

Property	Description
`fusion.spark.master.port`	Spark master job submission port. Default value is 8766.
`fusion.spark.master.ui.port`	Spark master UI port. Default value is 8767.
`fusion.spark.idleTime`	Maximum idle time in seconds, after which the application (ie. SparkContext) is shut down. Default value is 300.
`fusion.spark.executor.memory.min`	Minimum executor memory in MB. Default value 450Mb, which is sufficient to let Fusion components in application task’s to initialize themselves

Property	Description
`fusion.spark.executor.memory.fraction`	A float number in range `(0.0, 1.0]` indicating what portion of `spark.executor.memory` to allocate to this application. Default value is 1.0.
`fusion.spark.cores.fraction`	A float number in range `(0.0, 1.0]` indicating what portion of `spark.cores.max` to allocate to this application. Default value is 1.0.

15.39. Stopwords API

Note	These endpoints have been deprecated since Fusion 2.4.

The Stopwords REST API manages the stop words file for a collection. Only a one general set of stop words can be defined using this REST API. If you need different sets of stop words for different field types (perhaps for different languages), you will need to edit the schema.xml and manually manage the stop word files.

When updating the stop words, please note that only PUT requests are supported and any new data sent will overwrite the previous stop words. As such, PUT requests can be seen as replacement requests.

15.39.1. List or Update Stop Words

The path for this request is:

`/api/apollo/stopwords/<collectionName>`

where <*collectionName*> is the name of an specific collection.

A GET request will return the stop words in JSON format. A PUT request will replace the existing stop words with the new list.

These endpoints also allow for import and export of the stop words list. To download the list, replace the Content-Type in the GET request to "`Accept: application/octet-stream"`. To upload a file, you can emulate an HTTP form with cURL's "-form" option, as in `-form file=@stopwords.txt`.

Input

A JSON format list of stop words.

Output

A GET request will return the current stop words list, in JSON format.

No data will be returned for a PUT request.

15.39.2. Examples

Note	Use port 8765 in local development environments only. In production, use port 8764.

Update the stop words list with a new list of stop words:

REQUEST

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d '["a", "and", "of", "the"]'
http://localhost:8764/api/apollo/stopwords/docs
```

RESPONSE

None.

List the current stop words list:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/stopwords/docs
```

RESPONSE

```
[ "a", "and", "of", "the" ]
```

Upload a stop words file:

REQUEST

```
curl -u user:pass -X PUT --form file=@stopwords.txt http://localhost:8764/api/apollo/stopwords/docs
```

RESPONSE

None.

Download a stop words list & save it as a file named "stopwords.txt":

REQUEST

```
curl -H "Accept: application/octet-stream" http://localhost:8764/api/apollo/stopwords/docs > stopwords.txt
```

RESPONSE

None.

15.40. Synonyms API

Note	These endpoints have been deprecated since Fusion 2.4. Use the Synonyms Editor API instead.

The Synonyms API manages the set of synonyms defined in Solr for a collection:

- a string of terms that will expand on the terms the user entered, like `Television, TV`.
- a term that should be mapped to another term, like `i-pod⇒ipod`.

When updating the synonyms, note that only PUT requests are supported, and any new data sent will overwrite the previous synonyms. As such, PUT requests can be seen as replacement requests.

It is only possible to have a single set of stop words when using this REST API. If you need different sets of stop words for different field types (perhaps for different languages), you will need to edit the schema.xml and manually manage the stop word files.

15.40.1. List or Update Synonyms

The path for this request is:

`/api/apollo/synonyms/<collectionName>`

where <*collectionName*> is the name of an specific collection.

A GET request will return the synonym definitions in JSON format. A PUT request will replace the existing synonyms with the new list.

These endpoints also allow for import and export of the synonyms list. To download the list, replace the Content-Type in the GET request to "`Accept: application/octet-stream"`. To upload a file, you can emulate an HTTP form with cURL's "-form" option, as in `-form file=@synonyms.txt`.

Input

Property	**Description**
match *Required*	The terms that should be matched from the user input. When defining synonyms that should expand on the user's input, simply enter all of the terms as values of this property separated by commas. When defining synonyms where one or more terms should map to another term, only enter the term(s) to be mapped as the values for this property.
replace *Optional*	The terms that should replace the user input. This property is only used when mapping terms entered by the user to another term.

Output

A GET request will return the current synonym list, in JSON format.

No data will be returned for a PUT request.

15.40.2. Examples

Note	Use port 8765 in local development environments only. In production, use port 8764.

Update the synonym list with a new list of synonyms for a collection named 'docs':

REQUEST

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d '[{"match":[ "GB","gib","gigabyte","gigabytes"
]}, {"match": ["MB","mib","megabyte","megabytes"]}, {"match" :
["Television","Televisions","TV","TVs"]},{"match":["foo"],"replace":["bar"]}]'
http://localhost:8764/api/apollo/synonyms/docs
```

RESPONSE

None.

List the current synonyms list:

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/synonyms/docs
```

RESPONSE

```
[ {
  "match" : [ "GB", "gib", "gigabyte", "gigabytes" ]
}, {
  "match" : [ "MB", "mib", "megabyte", "megabytes" ]
}, {
  "match" : [ "Television", "Televisions", "TV", "TVs" ]
}, {
  "match" : [ "foo" ],
  "replace" : [ "bar" ]
}, {
  "match" : [ "i-pod" ],
  "replace" : [ "ipod" ]
} ]
```

Upload a synonyms list:

REQUEST

```
curl -u user:pass -X PUT --form file=@synonyms.txt http://localhost:8764/api/apollo/synonyms/docs
```

RESPONSE

None.

Download a synonyms list, and save it as a file named "synonyms.txt":

REQUEST

```
curl -u user:pass -H "Accept: application/octet-stream" http://localhost:8764/api/apollo/synonyms/docs >
synonyms.txt
```

RESPONSE

None.

15.41. System Admin APIs

The System Admin APIs are REST API endpoints that can be used by a system administrator to configure various Fusion processes. The REST APIs are:

- Configurations API, to check and modify global settings in Fusion.
- History API, to retrieve historic events of each service in the system.
- Nodes API, to check the status of each node in the search cluster.
- System API, to retrieve system metrics, ping the system for life, flush buffers, or view active threads.
- Usage API, to retrieve system information that is shared with Lucidworks about the size of your system.

15.42. System API

The System REST API allows you to monitor the system performance.

The 'metrics' endpoint provides statistics on the performance of each service, with several available counters, gauges, histograms, etc. More information about the available metrics is available in the section System Metrics.

The 'ping' endpoint allows you to ping the system for a response, to be sure it is up.

The 'threads' endpoint reports on the running threads.

The 'buffers' endpoint allows flushing buffers.

The 'version' endpoint shows the versions of included components.

15.42.1. Get Metric Names

The request path is:

`/api/apollo/system/metricNames`

A GET request shows available metric names, for use with the 'metric' endpoint (described in the following section). This request takes the following optional parameters:

Parameter	Description
pattern	The pattern allows using a regular expression to find all configuration items that contain the specified string.
prefix	The prefix allows finding all configuration items that start with the specified string.

The response will include all matching metrics, separated by type of metric:

- Gauges: single values, valid for the point in time at which the metrics are collected.
- Counters: values that are incremented or decremented over time.
- Histograms: the distribution of values. They will report the minimum, maximum, mean, and the values at the 50th, 75th, 95th, 98th, 99th, and 99.9th percentiles.
- Meters: rate of events over time. They include a mean rate, as well as a 1-, 5- and 15-minute moving average. Most of these moving averages are exponentially weighted, so that more recent values contribute more heavily than older values; exceptions to this rule have the word "unweighted" in their name.
- Timers: a meter combined with a histogram; it measures the length of time that a particular operation takes (both mean duration and moving averages) as well as the distribution of those durations.

When filtering on metric names, matching names will be shown in the applicable section.

15.42.2. Get System Metrics

The request path is:

`/api/apollo/system/metrics`

A GET request will return the system metrics.

Because there are nearly 600 metrics, the metricNames endpoint can help you identify a specific metric to return. This request takes the following optional query parameters which can also be used to filter the outputs:

Parameter	Description
pattern	The pattern allows using a regular expression to find all configuration items that contain the specified string.
prefix	The prefix allows finding all configuration items that start with the specified string.
pretty	If **true**, the response will be formatted for easier reading.
rateUnit	The time unit (i.e., "seconds", "minutes", etc.) to use to show rates.
durationUnit	The time unit (i.e., "seconds", "minutes", etc.) to use to show durations.
showSamples	If **true**, the data values used in calculations will be shown.

The output will include matching metrics, separated by type of metric, and any data for the metrics.

Many of the metrics include percentile and mean rates. Some are simply counts.

See the section System Metrics for more details about metrics that may be of interest.

15.42.3. Threads

The request path is:

`/api/apollo/system/threads`

A GET request shows all active system threads, as Java thread dump in JSON. The output is a standard Java thread dump, formatted for JSON.

15.42.4. Buffers

The request path is:

`/api/apollo/system/buffers`

A PUT request with this endpoint will flush the currently active buffers, if needed.

15.42.5. Ping

The request path is:

`/api/apollo/system/ping`

A GET request queries the system to see if it is up. If the system is up, the response will be 'pong'.

15.42.6. Examples

Metric names

Get metric names that start with 'mem.heap':

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/system/metricNames?prefix=mem.heap
```

RESPONSE

```
{
  "gauges" : [ "mem.heap.committed", "mem.heap.init", "mem.heap.max", "mem.heap.usage", "mem.heap.used" ],
  "counters" : [ ],
  "histograms" : [ ],
  "meters" : [ ],
  "timers" : [ ]
}
```

Metrics

*Find all metrics that match the regular expression pattern 'com.lucidworks.apollo.**pipeline.index.**', format the response so it's readable, and show the sample data, if any:*

REQUEST

```
curl -u user:pass
http://localhost:8764/api/apollo/system/metrics?pattern=com.lucidworks.apollo.*pipeline.index.*&pretty=true&sh
owSamples=true
```

RESPONSE

```
{
  "version" : "3.0.0",
  "gauges" : { },
  "counters" : { },
  "histograms" : { },
  "meters" : {
    "com.lucidworks.apollo.pipeline.index.IndexPipelineCache.cacheHit" : {
      "count" : 4775,
      "m15_rate" : 0.03604340402401043,
      "m1_rate" : 0.04985610410800882,
      "m5_rate" : 0.04753263154077047,
      "mean_rate" : 0.05028487069705915,
      "units" : "events/second"
    }
  },
  "timers" : {
    "com.lucidworks.apollo.pipeline.index.IndexPipelineCache.deserialize" : {
      "count" : 7,
      "max" : 0.078783,
      "mean" : 0.028265285714285715,
      "min" : 9.800000000000001E-5,
      "p50" : 1.94E-4,
      "p75" : 0.06996100000000001,
      "p95" : 0.078783,
      "p98" : 0.078783,
      "p99" : 0.078783,
      "p999" : 0.078783,
      "values" : [ 9.800000000000001E-5, 1.09E-4, 1.62E-4, 1.94E-4, 0.04855, 0.06996100000000001, 0.078783 ],
      "stddev" : 0.03620774742010466,
      "m15_rate" : 2.964393875E-314,
      "m1_rate" : 2.128434034679706E-46,
      "m5_rate" : 4.9195401948202935E-138,
      "mean_rate" : 7.371603924510614E-5,
      "duration_units" : "seconds",
      "rate_units" : "calls/second"
    },
    "com.lucidworks.apollo.pipeline.index.IndexStageConfigCache.deserialize" : {
      "count" : 7,
      "max" : 0.002377,
      "mean" : 4.6642857142857147E-4,
      "min" : 8.7E-5,
      "p50" : 1.16E-4,
      "p75" : 3.46E-4,
      "p95" : 0.002377,
      "p98" : 0.002377,
      "p99" : 0.002377,
      "p999" : 0.002377,
      "values" : [ 8.7E-5, 9.300000000000001E-5, 1.11E-4, 1.16E-4, 1.35E-4, 3.46E-4, 0.002377 ],
      "stddev" : 8.47267737523163E-4,
      "m15_rate" : 2.964393875E-314,
      "m1_rate" : 2.1522151745137967E-46,
      "m5_rate" : 5.086288568158318E-138,
      "mean_rate" : 7.372680363461613E-5,
      "duration_units" : "seconds",
      "rate_units" : "calls/second"
    },
    "com.lucidworks.apollo.pipeline.index.IndexStageConfigStore.deserialize" : {
      "count" : 6,
      "max" : 0.0019760000000000003,
      "mean" : 5.606666666666667E-4,
      "min" : 1.0800000000000001E-4,
      "p50" : 1.4250000000000002E-4,
      "p75" : 0.00114875,
      "p95" : 0.0019760000000000003,
      "p98" : 0.0019760000000000003,
      "p99" : 0.0019760000000000003,
      "p999" : 0.0019760000000000003,
      "values" : [ 1.0800000000000001E-4, 1.2200000000000001E-4, 1.35E-4, 1.5000000000000001E-4,
8.730000000000001E-4, 0.0019760000000000003 ],
      "stddev" : 7.54704622131511E-4,
      "m15_rate" : 2.964393875E-314,
      "m1_rate" : 1.0220227879692082E-48,
      "m5_rate" : 7.20591046931865E-140,
      "mean_rate" : 6.318494243486903E-5,
      "duration_units" : "seconds",
      "rate_units" : "calls/second"
    },
    "com.lucidworks.apollo.pipeline.index.IndexStageConfigStore.getItem" : {
      "count" : 6,
      "max" : 0.003128,
      "mean" : 0.002295,
      "min" : 0.0017770000000000002,
      "p50" : 0.0020715,
      "p75" : 0.002774,
      "p95" : 0.003128,
      "p98" : 0.003128,
      "p99" : 0.003128,
      "p999" : 0.003128,
      "values" : [ 0.0017770000000000002, 0.002066, 0.002066, 0.0020770000000000003, 0.0026560000000000004,
0.003128 ],
      "stddev" : 4.989869737778733E-4,
      "m15_rate" : 2.964393875E-314,
      "m1_rate" : 1.0220227879692082E-48,
      "m5_rate" : 7.20591046931865E-140,
      "mean_rate" : 6.318494244418448E-5,
      "duration_units" : "seconds",
      "rate_units" : "calls/second"
    }
  }
}
```

The above output has been truncated for space to remove metrics with no data or with very long value lists.

Threads

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/system/threads
```

RESPONSE (truncated to a single thread)

```
[ {
  "id" : 2,
  "native" : false,
  "name" : "Reference Handler",
  "locks" : {
    "waiting" : {
      "identity" : "0x7257d934",
      "class" : "java.lang.ref.Reference$Lock"
    },
    "locking" : {
      "identity" : "0x7257d934",
      "class" : "java.lang.ref.Reference$Lock"
    }
  },
  "state" : "WAITING",
  "suspended" : false,
  "stackTrace" : [ {
    "methodName" : "wait",
    "fileName" : "Object.java",
    "lineNumber" : -2,
    "className" : "java.lang.Object",
    "nativeMethod" : true
  }, {
    "methodName" : "wait",
    "fileName" : "Object.java",
    "lineNumber" : 503,
    "className" : "java.lang.Object",
    "nativeMethod" : false
  }, {
    "methodName" : "run",
    "fileName" : "Reference.java",
    "lineNumber" : 133,
    "className" : "java.lang.ref.Reference$ReferenceHandler",
    "nativeMethod" : false
  } ]
} ]
```

Buffers

REQUEST

```
curl -u user:pass -X PUT http://localhost:8764/api/apollo/system/buffers
```

Ping

REQUEST

```
curl -u user:pass http://localhost:8764/api/apollo/system/ping
```

RESPONSE

```
pong
```

15.43. Taxonomy API

The Taxonomy API is used to manage taxonomies, such as a hierarchy of product categories used for an eCommerce site.

The information in a taxonomy is meta-information about the categories used to classify a set of things. For an eCommerce site, the set of things are items in the product catalog. Fusion uses this meta-information to enhance search. In Fusion, these products are stored in a Fusion collection, (the primary collection), and the taxonomy is stored in an auxiliary collection. Naming conventions are used to relate the primary and auxiliary collections: given a primary collection named "COLL", the auxiliary taxonomy is stored in a collection named "COLL_stored_parameters".

The Taxonomy API It has the following set of endpoints to manage taxonomies and their constituent elements. All endpoints specify the name of the primary collection in the request path.

- Methods and endpoints to manage an entire taxonomy:
  - Create a taxonomy:
    POST `api/apollo/collections/<primary-collection-name>/taxonomy`
  - Retrieve a taxonomy:
    GET `api/apollo/collections/<primary-collection-name>/taxonomy`
  - Delete a taxonomy:
    DELETE `api/apollo/collections/<primary-collection-name>/taxonomy`
- Methods and endpoints to manage the categories in the taxonomy:
  - Create a new child category of an existing category:
    PUT `api/apollo/collections/<primary-collection-name>/taxonomy/<parent-category-id>`
  - Create a new top-level category in a taxonomy:
    POST `api/apollo/collections/<primary-collection-name>/taxonomy/`
  - Retrieve a category:
    GET `api/apollo/collections/<primary-collection-name>/taxonomy/<category-id>`
  - Update an existing category
    PUT `api/apollo/collections/<primary-collection-name>/taxonomy/<parent-category-id>`
    note: modification requires use of the "version" property.
  - Delete a category
    DELETE `api/apollo/collections/<primary-collection-name>/taxonomy/<parent-category-id>`

15.43.1. Taxonomy structure

In information-theoretic terms, the structure of a taxonomy is either a tree (or trees), or at least a directed acyclic graph (DAG) or set of DAGS. Each category object is a vertex in the tree or DAG, and the "children" property of each category object lists the set of outgoing edges.

Each category object has a unique ID. In addition, category objects have an optional property called a "parameter", which is a list of named attributes associated with that category. These parameters values from the taxonomy are added to search queries over the primary collection. In addition, every category object has a property "version" which is managed by Fusion and used to ensure that concurrent updates to the same category do not conflict with each other. The value of this property corresponds to the system clock time in milliseconds. This value is set and updated by Fusion and need only be specified overtly when sending category update requests.

Given the above specification, the set of property: values for a category object is:

- id: a unique string identifier.
- children: a list of category objects.
- parameters: a list of named attributes associated with that category where each attribute is specified as a pair of strings key:value.
- version: a string corresponding to the integer value of some system clock in milliseconds.

15.43.2. JSON example of a taxonomy

The following example is part of a taxonomy for an online pet supply store. In this example, we have the following hierarchy of category id, label pairs:

- 1 : "Pet Supplies"
  - 11 "Cat Supplies"
    - 111 "Cat Food"
    - 112 "Cat Toys"
  - 12 "Dog Supplies"
    - 121 "Dog Food"
    - 122 "Dog Toys"

We annotate these category objects with a set of parameters, which will be used organize and enrich user searches over the product catalog:

```
{ "id": "1",
  "label": "Pet Supplies",
  "children": [
      { "id": "11",
        "label": "Cat Supplies",
        "children": [
            { "id": "111",
              "label": "Cat Food",
              "parameters": [
                  { "key": "facet.field", "value": "brand" },
                  { "key": "facet.field", "value": "health_benefit" },
                  { "key": "facet.field", "value": "size" }
              ]},
            { "id": "112",
              "label": "Cat Toys",
              "parameters": [
                  { "key": "facet.field", "value": "price" }
              ] }
        ] },
      { "id": "12",
        "label": "Dog Supplies",
        "children": [
            { "id": "121",
              "label": "Dog Food",
              "parameters": [
                  { "key": "facet.field", "value": "life_stage" },
                  { "key": "facet.field", "value": "health_benefit" },
                  { "key": "facet.field", "value": "flavor" }
              ] },
            { "id": "122",
              "label": "Dog Toys",
              "parameters": [
                  { "key": "facet.field", "value": "price" },
                  { "key": "facet.field", "value": "dog_size" },
                  { "key": "facet.field", "value": "activity_type" }
              ] } ]
      } ]
}
```

The version property on these category objects isn't overtly specified here as it is managed by Fusion.

15.43.3. Manage a taxonomy

Create a taxonomy

POST `api/apollo/collections/<primary-collection-name>/taxonomy`

Retrieve a taxonomy

GET `api/apollo/collections/<primary-collection-name>/taxonomy`

Delete a taxonomy

DELETE `api/apollo/collections/<primary-collection-name>/taxonomy`

15.43.4. Manage the categories in the taxonomy

Create a new child category of an existing category

PUT `api/apollo/collections/<primary-collection-name>/taxonomy/<parent-category-id>`

Create a new top-level category in a taxonomy

POST `api/apollo/collections/<primary-collection-name>/taxonomy/`

Retrieve a category

GET `api/apollo/collections/<primary-collection-name>/taxonomy/<category-id>`

Update an existing category

PUT `api/apollo/collections/<primary-collection-name>/taxonomy/<parent-category-id>`

note: modification requires use of the "version" property.

Delete a category

DELETE `api/apollo/collections/<primary-collection-name>/taxonomy/<parent-category-id>`

15.44. Usage API

The Usage API sends system usage data to Fusion. See the section System Usage Monitor for details.

15.44.1. Get Usage Stats or the System ID

The path for this request is one of:

`/api/apollo/usage`

`/api/apollo/usage/<id>`

where <*id*> is the UUID of a specific system.

A GET request with the 'usage' endpoint retrieves the data that is sent to Fusion. A GET request with the 'id' endpoint retrieves the UUID of the system that is sent to Fusion with the usage data.

15.44.2. Examples

Get the latest data that will be sent to Fusion on the next run:

INPUT

```
curl -u user:pass http://localhost:8764/api/apollo/usage
```

OUTPUT

```
[ {
  "type" : "GAUGE",
  "name" : "solr.clusters",
  "value" : 3
}, {
  "type" : "GAUGE",
  "name" : "solr.nodes",
  "value" : 2
}, {
  "type" : "RATE",
  "name" : "recommender.itemsForQuery.timer.rate",
  "value" : 0
}, {
  "type" : "RATE",
  "name" : "recommender.queriesForItem.timer.rate",
  "value" : 0
}, {
  "type" : "GAUGE",
  "name" : "nodes",
  "value" : 2
}, {
  "type" : "RATE",
  "name" : "recommender.queriesForItem.counter.rate",
  "value" : 0
}, {
  "type" : "RATE",
  "name" : "recommender.itemsForItem.timer.rate",
  "value" : 0
}, {
  "type" : "RATE",
  "name" : "recommender.itemsForQuery.counter.rate",
  "value" : 0
}, {
  "type" : "RATE",
  "name" : "recommender.itemsForItem.counter.rate",
  "value" : 0
} ]
```

15.45. User API

The User API allows you to create, update, and remove user accounts. This API should only be called to manage users in the native security realm. Users from other security realms are managed directly by Fusion's auth proxy.

15.45.1. Create, Update, Delete or List Users

The path for this request is:

`/api/users/<id>`

where <*id*> is the user ID.

A GET request lists information about the user. The ID can be omitted in a GET request to get all users.

A POST request creates a new user, while a PUT updates a user record.

DELETE will remove the user.

Input

Parameter	Description
username *Required*	The username. This is distinct from their ID, which is assigned by the system as a unique identifier.
password *Required*	The user's password. Required when creating a new user. The user's password is not returned in the output of any request.
passwordConfirm *Required*	When creating a user or updating a user's password, you must confirm the defined password.
realmName *Required*	The realm the user belongs to, which defines how they authenticate against the system.
permissions *Optional*	The permissions that have been defined for this user that are not inherited from their assigned role.
inheritedPermissions *Optional*	The user's specific permissions that are inherited from their role assignment.
roleNames *Optional*	The list of user's roles, which define some or all of the permissions they have.

Output

When creating a user with a POST request or listing users with GET, the user properties will be returned.

When updating or removing a user with a PUT or DELETE, no output will be returned.

Examples

Note	Use port 8765 in local development environments only. In production, use port 8764.

Get all the configured users of the system:

REQUEST

```
curl -u user:pass http://localhost:8764/api/users
```

RESPONSE

```
[
    {
    "realmName":"native",
    "username":"admin",
    "id":"2856ba33-80bd-400d-99dc-3d181bc68d9a",
    "roleNames":["admin"],
    "permissions":[],
    "createdAt":"2015-07-01T03:18:06Z"},
        {"realmName":"native",
        "username":"collection-admin",
        "id":"9780a33c-c49d-48e3-a869-bd65951aea8f",
        "roleNames":["ui-user","collection-admin"],
        "permissions":[],
        "createdAt":"2015-07-01T03:18:06Z"
    }
]
```

Add a new user named 'guest':

REQUEST

```
curl -u user:pass -X POST -H 'Content-type: application/json' -d '{"username":"guest",
"password":"password456", "passwordConfirm":"password456", "realmName": "native"}'
http://localhost:8764/api/users
```

RESPONSE

```
{
    "realmName":"native",
    "username":"guest",
    "id":"2f5b52a7-550d-407d-b592-32ab42afe3ca",
    "roleNames":[],
    "permissions":[],
    "createdAt":"2015-08-06T11:42:15Z"
}
```

Update a user to include the role named "admin":

REQUEST

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d '{"name":"joe.smith", "realmName":"myLDAP",
"roleNames":["admin"]}' http://localhost:8764/api/users/aefa7ffc-23f1-45ac-b326-f7bb007d3b9d
```

RESPONSE

None.

15.46. ZooKeeper Import/Export API

The ZooKeeper Import/Export API provides methods to upload or download information from Fusion's ZooKeeper service. This service provides an alternative to the ZooKeeper clients `zkCli.sh` and `zk-shell` which are part of the Apache Zookeeper distribution included as part of the Fusion distribution. It was introduced as part of the Fusion 2.0 release.

The ZKImportExport service may be used to export ZooKeeper data for any Fusion release. It can be used to import configuration data into the ZooKeeper service for a new or existing Fusion deployment. Note that since the Fusion 3.0 release all ZooKeeper paths vary according to the version of Fusion that you are running.

- For details on using this script during the Fusion upgrade procedure, see Upgrading Fusion.
- For details on using this script to migrate Fusion configurations from one deployment to another, see Migrating Fusion data.

The REST API only supports requests to export ZooKeeper configurations. The Fusion distribution includes a utility script `zkImportExport.sh` which can be used to import ZooKeeper configuration as well as to export it from arbitrary Fusion instances.

15.46.1. ZooKeeper

Apache ZooKeeper is a distributed configuration service, synchronization service, and naming registry. Fusion uses ZooKeeper to configure and manage all Fusion components in a single Fusion deployment.

- **znode**: ZooKeeper data is organized into a hierarchal name space of data nodes called znodes. A znode can have data associated with it as well as child znodes. The data in a znode is stored in a binary format, but it is possible to import, export, and view this information as JSON data. Paths to znodes are always expressed as canonical, absolute, slash-separated paths; there are no relative reference.
- **ephemeral nodes**: An ephemeral node is a znode which exists only for the duration of an active session. When the session ends the znode is deleted. An ephemeral znode cannot have children.
- **server**: A ZooKeeper service consists of one or more machines; each machine is a server which runs in its own JVM and listens on its own set of ports. For testing, you can run several ZooKeeper servers at once on a single workstation by configuring the ports for each server.
- **quorum**: A quorum is a set of ZooKeeper servers. It must be an odd number. For most deployments, only 3 servers are required.
- **client**: A client is any host or process which uses a ZooKeeper service.

See the official ZooKeeper documentation for details about using and managing a ZooKeeper service.

15.46.2. Utility script `zkImportExport.sh`

This script is located in the top-level Fusion `scripts` directory. The script takes the following command-line arguments:

```
-c,--cmd <arg>        Command, one of: 'export', 'import', 'update', 'delete'.

-e,--encode <arg>     Type of encoding for znodes. Valid options:
                       'none', 'utf-8', 'base64', default is 'base64'.
                       Option 'none' will not return any data from the znodes.

-ep,--exclude <arg>   Exclude znode paths, followed by list of paths.
                      Can only be used to exclude nodes one level below the root node.

-eph,--ephemeral      Include ephemeral nodes while exporting znodes, boolean, default false.

-f,--filename <arg>   Name of file containing import/export data.

-h,--help             Display help page.

-ip,--include <arg>   Include znode paths to include, followed by a list of paths.
                      Can only be used to include nodes one level below the root node.

-nr,--non-recursive   Do not perform recursive operations on znodes.

-o,--overwrite        Overwrite data for existing znodes. Valid only with 'update' command.

-p,--path <arg>       Path from ZooKeeper root node, e.g. '/lucid/query-pipelines'.

-r,--recursive        Perform recursive operations on znodes.

-z,--zkhost <arg>     ZooKeeper Connect string, required.
```

Required arguments are: * -c, --cmd : operation to perform. * -z, --zkhost : the ZooKeeper Connect string.

Examples

Export all data from a local single-node ZooKeeper service, save data to a file:

```
zkImportExport.sh -zkhost localhost:9983 -cmd export -path / -filename znode_dump.json
```

Export all Fusion configurations from a local single-node ZooKeeper service, save data to a file:

```
zkImportExport.sh -zkhost localhost:9983 -cmd export -path /lucid -filename znode_lucid_dump.json
```

Export Fusion user databases, groups, roles, and realms configurations from a local single-node ZooKeeper service, save data to a file:

```
zkImportExport.sh -zkhost localhost:9983 -cmd export -path /lucid-apollo-admin -filename
znode_lucid_admin_dump.json
```

Initial import of saved Fusion configuration into a new ZooKeeper:

```
zkImportExport.sh -zkhost localhost:9983 -cmd import -filename znode_lucid_dump.json
```

Note that the above command will fail if there is conflict between existing znode structures or contents between the ZooKeeper service and the dump file.

Update information for Fusion’s ZooKeeper service:

```
zkImportExport.sh -zkhost localhost:9983 -cmd update -filename znode_lucid_dump.json
```

Remove a znode from Fusion’s ZooKeeper service:

```
zkImportExport.sh -zkhost localhost:9983 -cmd delete -path /lucid/test
```

15.46.3. Fusion REST API service ZKImportExport

The Fusion REST API can only be used to download information from ZooKeeper, via the 'GET' method with the following configuration:

```
  "zk-import-export::v1" : {
    "name" : "com.lucidworks.apollo.resources.ZKImportExportResource",
    "uri" : "/zk/export",
    "methods" : [ {
      "uri" : "/zk/export/{path:.*}",
      "name" : "getNodeInfo",
      "verb" : "GET",
      "pathParams" : [ {
        "name" : "path",
        "type" : "String"
      } ],
      "queryParams" : [ {
        "name" : "recursive",
        "type" : "Boolean"
      }, {
        "name" : "excludePaths",
        "type" : "List"
      }, {
        "name" : "includePaths",
        "type" : "List"
      }, {
        "name" : "encodeValues",
        "type" : "String"
      } ],
      "hasEntity" : false
    } ]
  }
```

GET data from path `/lucid/query-pipelines`

```
> curl -u user:pass -X GET http://localhost:8764/api/apollo/zk/export/lucid/query-pipelines

{
  "path" : "/lucid/query-pipelines",
  "children" : [ {
    "path" : "/lucid/query-pipelines/default",
    "data" :
"ewogICJpZCIgOiAiZGVmYXVsdCIsCiAgInN0YWdlcyIgOiBbIHsKICAgICJ0eXBlIiA6ICJzZWFyY2gtZmllbGRzIiwKICAgICJpZCIgOiAiM
zc1NmI1ZDctY2MwMC00MDAyLWJiOWQtNTQzNjQ4NzVjMjgyIiwKICAgICJyb3dzIiA6IDEwLAogICAgInN0YXJ0IiA6IDAsCiAgICAic2tpcCI
gOiBmYWxzZSwKICAgICJsYWJlbCIgOiAic2VhcmNoLWZpZWxkcyIsCiAgICAidHlwZSIgOiAic2VhcmNoLWZpZWxkcyIKICB9LCB7CiAgICAid
HlwZSIgOiAiZmFjZXQiLAogICAgImlkIiA6ICI3MTFmZWM1Ni03MzRmLTRiYTAtOWY1NS0wZTQ4YmU2NTllM2UiLAogICAgInNraXAiIDogZmF
sc2UsCiAgICAibGFiZWwiIDogImZhY2V0IiwKICAgICJ0eXBlIiA6ICJmYWNldCIKICB9LCB7CiAgICAidHlwZSIgOiAic29sci1xdWVyeSIsC
iAgICAiaWQiIDogIjYwNDU1YjU2LTdkN2MtNDZjYi1iN2QzLTIxOWU1N2U3MWNjMyIsCiAgICAiaHR0cE1ldGhvZCIgOiAiUE9TVCIsCiAgICA
ic2tpcCIgOiBmYWxzZSwKICAgICJsYWJlbCIgOiAic29sci1xdWVyeSIsCiAgICAidHlwZSIgOiAic29sci1xdWVyeSIKICB9IF0KfQ=="
  } ],
  "data" : ""
}
```

Get info for node `/lucid/query-pipelines`*. Do not expand the znodes*

```
curl -u user:pass -X GET http://localhost:8764/api/apollo/zk/export/lucid/query-
pipelines?recursive=true&encodeValues=none
{
  "path" : "/lucid/query-pipelines",
  "children" : [ {
    "path" : "/lucid/query-pipelines/default"
  } ]
}
```

Get info for node `/lucid/query-pipelines`*. encode in `utf-8`*

```
curl -u user:pass -X GET http://localhost:8764/api/apollo/zk/export/lucid/query-
pipelines?recursive=true&encodeValues=utf-8
{
  "path" : "/lucid/query-pipelines",
  "children" : [ {
    "path" : "/lucid/query-pipelines",
    "data" : "{\n  \"id\" : \"default\",\n  \"stages\" : [ {\n    \"type\" : \"search-fields\",\n    \"id\" :
\"3756b5d7-cc00-4002-bb9d-54364875c282\",\n    \"rows\" : 10,\n    \"start\" : 0,\n    \"skip\" : false,\n
\"label\" : \"search-fields\",\n    \"type\" : \"search-fields\"\n  }, {\n    \"type\" : \"facet\",\n
\"id\" : \"711fec56-734f-4ba0-9f55-0e48be659e3e\",\n    \"skip\" : false,\n    \"label\" : \"facet\",\n
\"type\" : \"facet\"\n  }, {\n    \"type\" : \"solr-query\",\n    \"id\" : \"60455b56-7d7c-46cb-b7d3-
219e57e71cc3\",\n    \"httpMethod\" : \"POST\",\n    \"skip\" : false,\n    \"label\" : \"solr-query\",\n
\"type\" : \"solr-query\"\n  } ]\n}"
  } ],
  "data" : ""
}
```

Get info for node `/lucid/query-pipelines`*. Exclude path `/lucid/query-pipelines/default`*

```
curl -u user:pass -X GET http://localhost:8764/api/apollo/zk/export/lucid/query-
pipelines?recursive=true&encodeValues=utf-8&excludePaths=/lucid/query-pipelines/default
{
  "path" : "/lucid/query-pipelines",
  "data" : ""
}
```