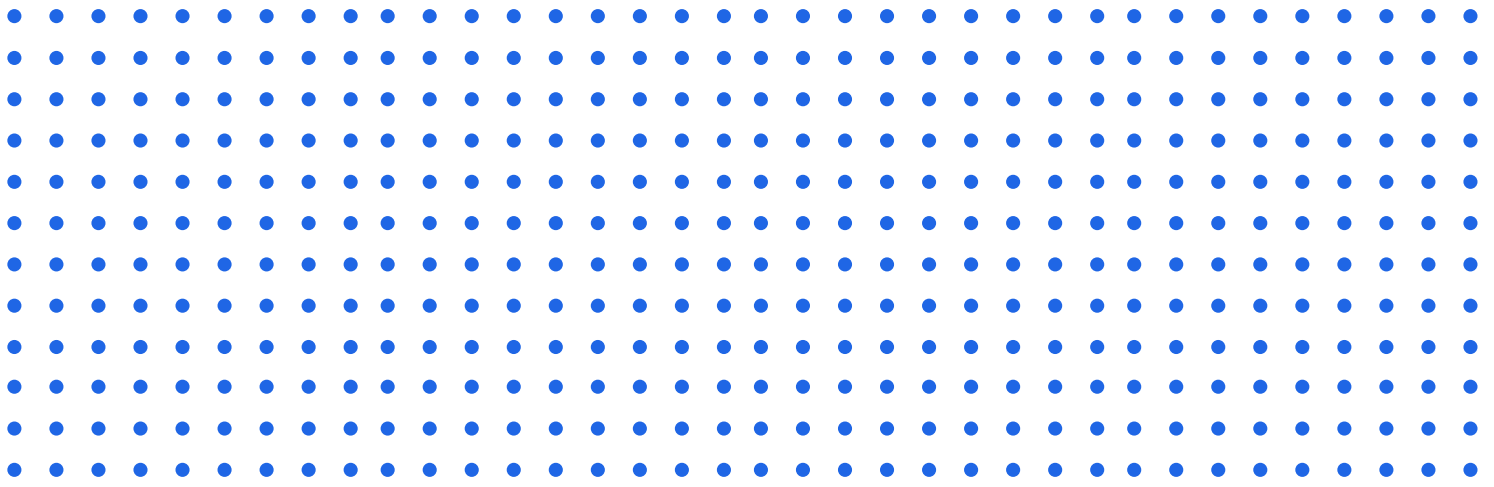




Fusion Server

Version 4.1

Table of Contents

- *About Legacy Docs*
- *Fusion Concepts and Components*
 - *Fusion Server Deployment*
 - *Deployment Types*
 - *On-Prem Deployment*
 - *Fusion UI*
 - *Querying*
 - *Building Applications*
 - *Autocomplete*
 - *Faceting*
 - *Search Applications*
 - *Stopwords Files*
 - *Synonyms Files*
 - *Query Pipelines and Profiles*
 - *Custom JavaScript Stages for Query Pipelines*
 - *Query Pipeline Stages*
 - *Query Pipelines*
 - *Query Profiles*
 - *Stored Parameters Collection*
 - *Query Workbench*
 - *Security*
 - *Access Control*
 - *Permissions*
 - *Roles*
 - *Security Realms*
 - *Users*
 - *Ingest and Indexing*
 - *Blob Storage*
 - *Collections*
 - *Time-Based Partitioning*
 - *Connectors*
 - *Datasources*
 - *Index Workbench*
 - *Index Pipelines*
 - *Index Pipeline Stages*
 - *Index Profiles*
 - *Custom JavaScript Stages For Index Pipelines*
 - *The Parallel Bulk Loader*
 - *Dashboards*
 - *Built-in Dashboards*
 - *Display Panels*
 - *Input Panels*
 - *System Administration*
 - *Monitoring*
 - *Cluster Monitoring and Troubleshooting*
 - *Messaging and Alerting*
 - *Troubleshooting Performance Issues*
 - *System Usage Monitor*
 - *Jobs*
 - *Datasource Jobs*
 - *Fusion Schedules*
 - *Job History*
 - *Spark Jobs*
 - *Tasks*
 - *Fusion SQL Query and Analytics*
 - *Fusion SQL Administration*
 - *Licensing*
 - *Object Explorer*
 - *Spark Operations*
 - *Spark Components*
 - *Spark Job Drivers*
- *Reference Guides*
 - *System Requirements*

- *Fusion Server 4.x File Download Links*
- *Additional Resources*
- *Pipeline Stages*
 - *Query Pipeline Stages*
 - *Active Directory Security Trimming Stage*
 - *Additional Query Parameters Stage*
 - *Analytics Catalog Stage*
 - *Block Documents Stage*
 - *Boost Documents Stage*
 - *Boost with Signals Stage*
 - *Custom JavaScript Query Stage Examples*
 - *Experiment Stage*
 - *Field Facet Stage*
 - *JDBC Lookup Stage*
 - *JavaScript Query Stage*
 - *Landing Pages Stage*
 - *Logging Stage*
 - *Managed JavaScript Query Stage*
 - *Parameterized Boosting Stage*
 - *Parameterized Faceting Stage*
 - *Query Fields Stage*
 - *REST Query Stage*
 - *Recommend Items for Item Stage*
 - *Recommend Items for User Stage*
 - *Retrieve Stored Parameters Stage*
 - *Return Query Parameters Stage*
 - *Rollup Aggregation Stage*
 - *Run Query Pipeline Stage*
 - *Security Trimming Stage*
 - *Send PagerDuty Message Stage*
 - *Send SMTP Email Stage*
 - *Send Slack Message Stage*
 - *Solr MoreLikeThis Stage*
 - *Solr Query Stage*
 - *Solr Subquery Stage*
 - *Write Log Message Stage*
 - *Index Pipeline Stages*
 - *Apache Tika Parser Index Stage*
 - *Call Pipeline Index Stage*
 - *Date Parsing Index Stage*
 - *Detect Language Index Stage*
 - *Detect Sentences Index Stage*
 - *Exclude Documents Index Stage*
 - *Exclusion Filter Index Stage*
 - *Field Mapping Index Stage*
 - *Field Parser Index Stage*
 - *Filter Short Fields Index Stage*
 - *Find and Replace Index Stage*
 - *Format Signals Index Stage*
 - *Gazetteer Lookup Extraction Index Stage*
 - *GeoIP Lookup Stage*
 - *Include Documents Index Stage*
 - *JDBC Index Stage*
 - *JavaScript Index Stage*
 - *Logging Index Stage*
 - *Machine Learning Index Stage*
 - *Managed JavaScript Index Stage*
 - *OpenNLP NER Extraction Index Stage*
 - *REST Query Index Stage*
 - *Regex Field Extraction Index Stage*
 - *Regex Field Filter Index Stage*
 - *Regex Field Replacement Index Stage*
 - *Resolve Multivalued Fields Index Stage*
 - *Send PagerDuty Message Index Stage*
 - *Send SMTP Email Index Stage*
 - *Send Slack Message Index Stage*
 - *Set Property Index Stage*

- *Solr Dynamic Field Name Mapping*
 - *Solr Indexer Stage*
 - *Solr Partial Update Indexer Stage*
 - *Tag Part-of-Speech Index Stage*
 - *Update Related Document Stage*
 - *Write Log Message Index Stage*
 - *XML Transformation Index Stage*
- *Jobs Configuration*
 - *Aggregation Jobs*
 - *Custom Spark Jobs*
 - *Download Blob*
 - *Log Cleanup Jobs*
 - *REST Call Jobs*
 - *Script Jobs*
- *Parsers*
 - *Apache Tika Parser Stage*
 - *Archive Parser Stage*
 - *CSV Parser Stage*
 - *Enable Tika Asynchronous Parsing*
 - *Fallback Parser Stage*
 - *Grok Parser Stage*
 - *Grok Patterns*
 - *HTML Parser Stage*
 - *JSON Parser Stage*
 - *Text Parser Stage*
 - *XML Parser Stage*
- *REST APIs*
 - *Apps API*
 - *Authentication and Authorization APIs*
 - *Realms API*
 - *Roles API*
 - *Sessions API*
 - *User API*
 - *Blob Store API*
 - *Catalog API*
 - *Collection Features API*
 - *Collections API*
 - *System Admin APIs*
 - *Configurations API*
 - *History API*
 - *Nodes API*
 - *System API*
 - *Usage API*
 - *Connector APIs*
 - *Connector Datasources API*
 - *Connector History API*
 - *Connector Jobs API*
 - *Connector Status API*
 - *Experiments API*
 - *Groups API*
 - *Index APIs*
 - *Index API*
 - *Index Pipelines API*
 - *Index Profiles API*
 - *Index Stages API*
 - *Jobs API*
 - *License API*
 - *Links API*
 - *Messaging API*
 - *Objects API*
 - *Parsers API*
 - *Query APIs*
 - *Query API*
 - *Query Pipelines API*
 - *Query Profiles API*
 - *Query Stages API*
 - *Recommendations API*

- *Scheduler API*
 - *Search Cluster API*
 - *Signals API*
 - *Solr API*
 - *Solr Configuration API*
 - *SolrAdmin API*
 - *Spark Jobs API*
 - *Synonyms Editor API*
 - *Tasks API*
 - *Taxonomy API*
 - *Webapps API*
 - *Webapps Admin API*
 - *ZooKeeper Export API*
 - *Configure Fusion Logging in Fusion 4.0*
 - *Query Operations*
 - *Configure Fusion for JWT*
 - *Custom JavaScript Stages Variables and Parameters*
 - *Solr Query Language Cheat Sheet*
 - *Querying Fusion SQL*
 - *Cumulative Probability (cumulative_probability)*
 - *Mann-Whitney (mann_whitney)*
 - *Probability (probability)*
 - *QQPlot*
 - *Time Series (date_format)*
 - *click_pos*
 - *concat_text*
 - *significantTerms*
 - *timediff*
 - *ttest*
 - *weighted_sum*
 - *Fusion Server Directories, Files, and Ports*
 - *Template Expressions*
 - *Index Pipeline Stage Template Expression Variables*
 - *Messaging Services Template Expression Variables*
 - *Query Pipeline Stage Template Expression Variables*
 - *Resources and Support*
 - *Language Support in Fusion*
 - *Monitor Fusion 4.x.x*
 - *Solr Query Language Cheat Sheet*
 - *Solr Reference Guide*
 - *Spark Reference Guide*
 - *System Metrics*
 - *Time-based Partitioning*
 - *Release Notes*
 - *Fusion Server 4.1.3 Release Notes*
 - *Fusion Server 4.1.2 Release Notes*
 - *Fusion Server 4.1.1 Release Notes*
 - *Fusion Server 4.1.0 Release Notes*
 - *Fusion Server 4.0.2 Release Notes*
 - *Fusion Server 4.0.1 Release Notes*
 - *Fusion Server 4.0.0 Release Notes*
 - *Deprecations and Removals*
-

About Legacy Docs

This site includes legacy documentation for product versions that are no longer supported, according to our Lucidworks Fusion Product Lifecycle (<https://doc.lucidworks.com/policies/7shln5/lucidworks-version-support-lifecycle>).

Legacy documentation is not updated or maintained. As a result, you may discover mistakes or inaccuracies within the documentation.

Recent migrations

Product	End of support	Migrated to legacy
Fusion 5.6.x	February 10, 2023	October 21, 2024
Fusion 5.7.x	June 13, 2023	October 21, 2024
Fusion 5.8.x	September 22, 2023	October 21, 2024
Managed Fusion 5.6.x	February 10, 2023	Managed Fusion documentation is not migrated.
Managed Fusion 5.7.x	June 13, 2023	Managed Fusion documentation is not migrated.
Managed Fusion 5.8.x	September 22, 2023	Managed Fusion documentation is not migrated.

Frequently asked questions

Why did you migrate legacy docs to a new site?

A new site is the best way to provide a high quality user experience. The legacy documentation site is based on this site and will continue to receive feature updates, as appropriate.

What about my saved pages?

We understand that you may have saved pages in bookmarks, shared documents, and other places. To help minimize the impact of this migration, we added page redirects from the current documentation site. However, we recommend that you update your saved pages as soon as possible.

Will more products be affected in the future?

Yes, we plan to migrate documentation for product versions that reach their end of life support to the legacy documentation site in the future. This will take place on or after the date stated in the Lucidworks Fusion Product Lifecycle (<https://doc.lucidworks.com/policies/7shln5/lucidworks-version-support-lifecycle>) policy.

Fusion Concepts and Components

Basic Fusion concepts are explained below. Since the core of Fusion Server is Solr, you may also find it useful to familiarize yourself with Solr terminology (<https://lucene.apache.org/solr/guide/solr-glossary.html>).

Apps

Fusion apps provide tailored search functionality to specific groups of users.

An app is a named set of linked *objects*, including collections, datasources, index and query pipelines, index and query profiles, parsers, and more. Using roles and security realms, you can define security on a per-app basis.

Collections

Collections consist of stored data and the datasources that determine how the data is ingested and indexed. Collections are a way to logically group your data sets. Fusion's concept of collections is the same as Solr collections. See *Collection Management*.

Datasources

Datasources are the configurations that determine how data is ingested and indexed. Each datasource includes a connector configuration, a parser configuration, and an index pipeline configuration. See *Datasource Configuration*.

Connectors

Connectors are the conduit between Fusion and your external data sources. Connectors retrieve your data and import it into Fusion Server.

See the *Connectors Reference Guide* for a complete list of available connectors.

Parsers

Parsers interpret incoming data in order to determine its format and fields. A parser consists of a sequence of parsing stages, each designed to parse a different data format, sometimes recursively. See the *Parser Stages Reference Guide* for complete details about all available parsing stages.

Index pipelines

Index pipelines format the incoming raw data into fielded documents that it can be indexed and searched by the Solr core. A pipeline consists of a sequence of stages, and each stage performs a different kind of processing based on user-configured logic. See the *Index Pipeline Stages Reference Guide* for a complete list of available index pipeline stages.

Query Pipelines

Query pipelines manipulate incoming queries and return an ordered list of matching results from Solr. Individual search results are called documents. See *Query Pipeline Configuration*.

Fusion Components

Apache 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.

Solr configuration

Fusion requires that Solr run with SolrCloud (<https://wiki.apache.org/confluence/display/solr/SolrCloud>) enabled.

Configuration for Solr's Web service is in `https://FUSION_HOST:FUSION_PORT/apps/jetty/solr`.

Solr logs

Solr log files are in `https://FUSION_HOST:FUSION_PORT/var/log/solr`.

Accessing the Solr UI

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

Solr documentation

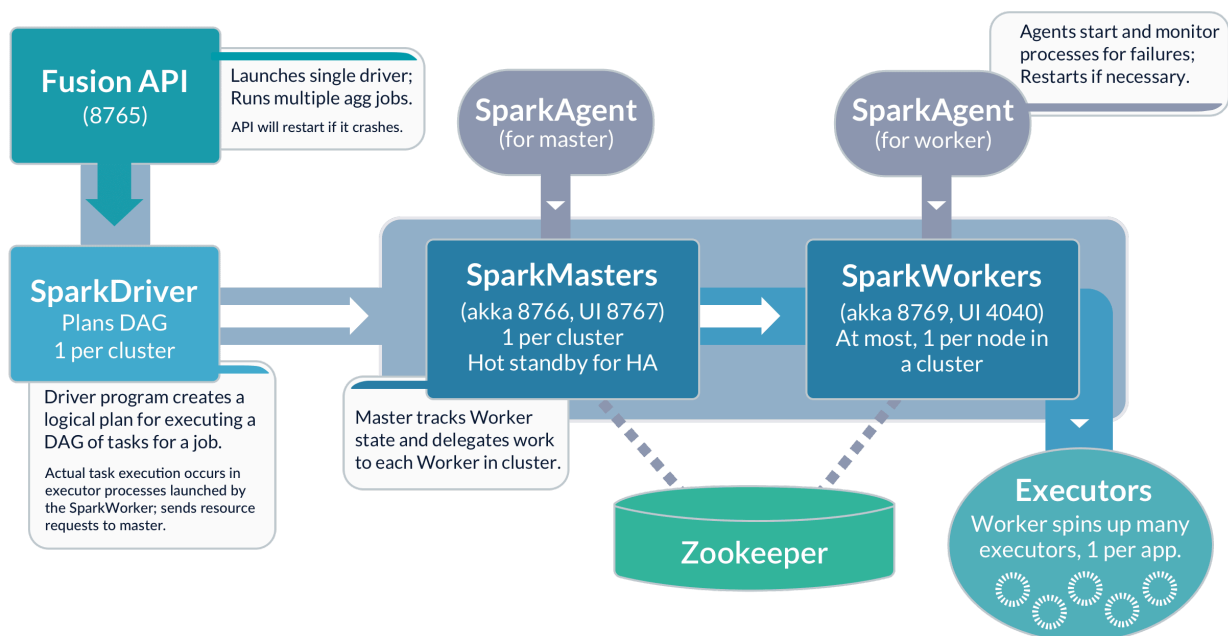
Solr documentation and additional resources are available at <http://lucene.apache.org/solr/resources.html> (<http://lucene.apache.org/solr/resources.html>).

You can also find plenty of Solr tips and technical discussions in our blog (<https://lucidworks.com/?s=blogs/s=solr>) and webinars (<https://lucidworks.com/?s=webinars/s=solr>).

Apache Spark

Apache Spark (<http://spark.apache.org/>) 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 Fusion:



See *Spark Administration* for more information.

Apache ZooKeeper

Apache ZooKeeper (<http://zookeeper.apache.org/>) 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 (http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#ch_deployment).

You can find ZooKeeper's logs at `https://FUSION_HOST:FUSION_PORT/var/log/zookeeper`.

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 (<http://zookeeper.apache.org/doc/current/>) for details about using and managing a ZooKeeper service.

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:

Field	Value
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
pxxid	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:

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 at 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:

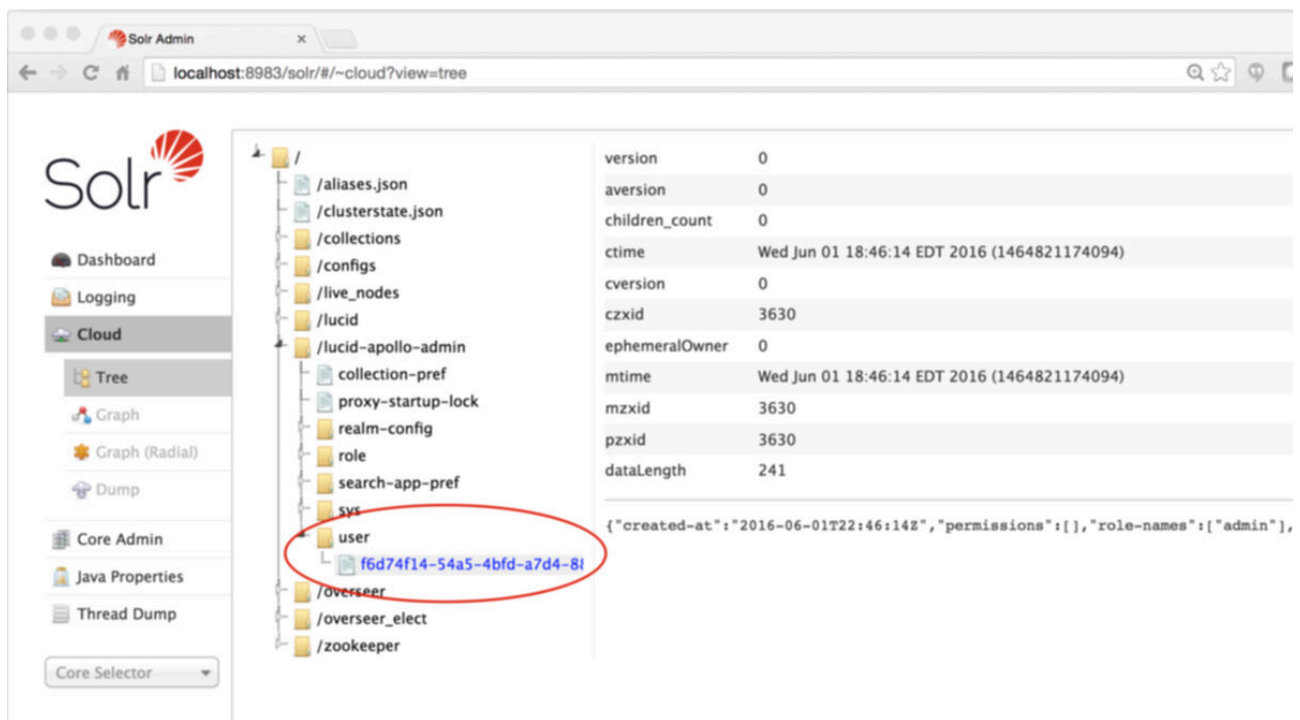
The screenshot shows the Solr Admin interface with the ZooKeeper node browser. The left sidebar contains navigation links: Dashboard, Logging, Cloud, Tree, Graph, Graph (Radial), Dump, Core Admin, Java Properties, Thread Dump, and Core Selector. The main area displays a tree view of the ZooKeeper node structure. The path `/lucid/connectors/datasources/ds1` is highlighted with a red circle. The right pane shows the configuration details for this node, including version, aversion, children_count, ctime, cversion, czxid, ephemeralOwner, mtime, mxid, pzxid, and dataLength. Below this, a JSON configuration is shown.

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)
mxid	4173
pxid	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"`:



Jetty

Jetty (<https://eclipse.org/jetty>) provides Web services for Fusion's UI, APIs, and Connectors, plus Solr. Each of those components runs inside its own instance of Jetty, using a separate configuration. Configurations for each component are located in https://FUSION_HOST:FUSION_PORT/apps/jetty.

Securing Fusion using SSL requires configuring Jetty to use SSL. For example, to secure the UI you need to modify the configuration in https://FUSION_HOST:FUSION_PORT/apps/jetty/admin-ui. See *SSL Security (Unix)* or *SSL Security (Windows)*.

Log messages about Jetty are written to the log files for the components that use it:

- https://FUSION_HOST:FUSION_PORT/var/log/ui
- https://FUSION_HOST:FUSION_PORT/var/log/api
- https://FUSION_HOST:FUSION_PORT/var/log/connectors
- https://FUSION_HOST:FUSION_PORT/var/log/solr

Fusion Server Deployment

These topics explain how to plan for, install, and upgrade your Fusion Server deployment.

On-premises deployment

Architecture and requirements

- **Review system requirements.** Start here to learn about required software and hardware.
- **Review deployment types.** Learn about deployment types for demos, trials, development, and production.
- **Start and stop Fusion Server.** Start and stop everything or individual services.
- **Learn about directories, files, and ports.**
- **Review licensing information.**

Deploy Fusion Server on-prem

- **Install Fusion Server on a single node (Unix).** Install Fusion Server on a single Unix node. This is an easy way to try Fusion Server. Out-of-the-box, Fusion Server uses bundled instances of Solr, ZooKeeper, and Spark.
- **Install Fusion Server on a single node (Windows).** Install Fusion Server on a single Unix node. This is an easy way to try Fusion Server. Out-of-the-box, Fusion Server uses bundled instances of Solr, ZooKeeper, and Spark.
- **Learn about installing a Fusion cluster.** You can cluster Fusion Server, Solr, and ZooKeeper nodes in a number of ways.
- **Install a Fusion cluster (Unix).** Install multiple instances of Fusion Server, and of Solr and ZooKeeper, in a cluster.
- **Install a Fusion cluster (Windows).** Install multiple instances of Fusion Server, and of Solr and ZooKeeper, in a cluster.
- **Integrate Fusion with an existing Solr deployment.** If you have an existing Solr deployment, you can integrate Fusion Server with that deployment (if it meets Fusion Server's version requirement for Solr).
- **Troubleshoot installation.** Troubleshoot difficulties that you might encounter when installing Fusion Server.

Upgrade and maintain Fusion Server

- **Upgrade Fusion.** Run a migrator to upgrade Fusion Server.
- **Migrate Fusion objects by hand.** Migrate Fusion objects by hand. You might need to do this for objects that the migrator does not migrate automatically. You can also use this manual approach to migrate Fusion objects from development environments into testing and production environments, or to back up data and restore it after an incident of data loss.
- **Add or move Fusion Server nodes.** Add a new node to an existing Fusion cluster, or move Fusion from one node to another.

Deployment Types

The Fusion platform is designed to support enterprise search applications at any scale. You can deploy Fusion across multiple nodes in order to store large amounts of data or to achieve high processing throughput or both.

Fusion consists of a number of Java processes that run in JVMs, including the `api`, `connectors-classic`, `connectors-rpc`, and `admin-ui` processes, and possibly others such as `spark-master` and `spark-worker`. When you start Fusion, the processes that start are listed. You might also see `zookeeper` and `solr` processes, depending on the *cluster arrangement*.

For more information about Fusion components, see *Fusion Components*. For a complete list of Fusion services, see *Start and stop Fusion*.

Deployment goals

- **Demo, trial, and development deployments.** The simplest possible architecture is the one you get out of the box, by unpacking the tar/zip file and running `https://FUSION_HOST:FUSION_PORT/bin/fusion start`, so that all components (including the bundled Solr and ZooKeeper instances) run on a single host in their default configurations. You can quickly install and run Fusion on a computer (even on your laptop) to explore its features and work with sample data. See *Quickstart* for instructions. This diagram illustrates a single-node Fusion deployment:

Fusion node



* Depends on features used

- **Onsite late-stage development and test deployments.** Ideally, an onsite deployment for late-stage development and testing should have the same architecture as the production deployment, though it does not need to be scaled to provide the same level of service.
- **Production deployments.** Fusion is designed for flexible, distributed deployment. Any of its components can be distributed across your network, and some can be clustered. A production deployment requires multiple Fusion nodes, each of which runs some or all Fusion services (including Solr and ZooKeeper).

Cluster Arrangements

You can deploy Fusion across multiple nodes in a *Fusion cluster* and use a ZooKeeper cluster as the centralized, synchronized store for both application configurations and user data.

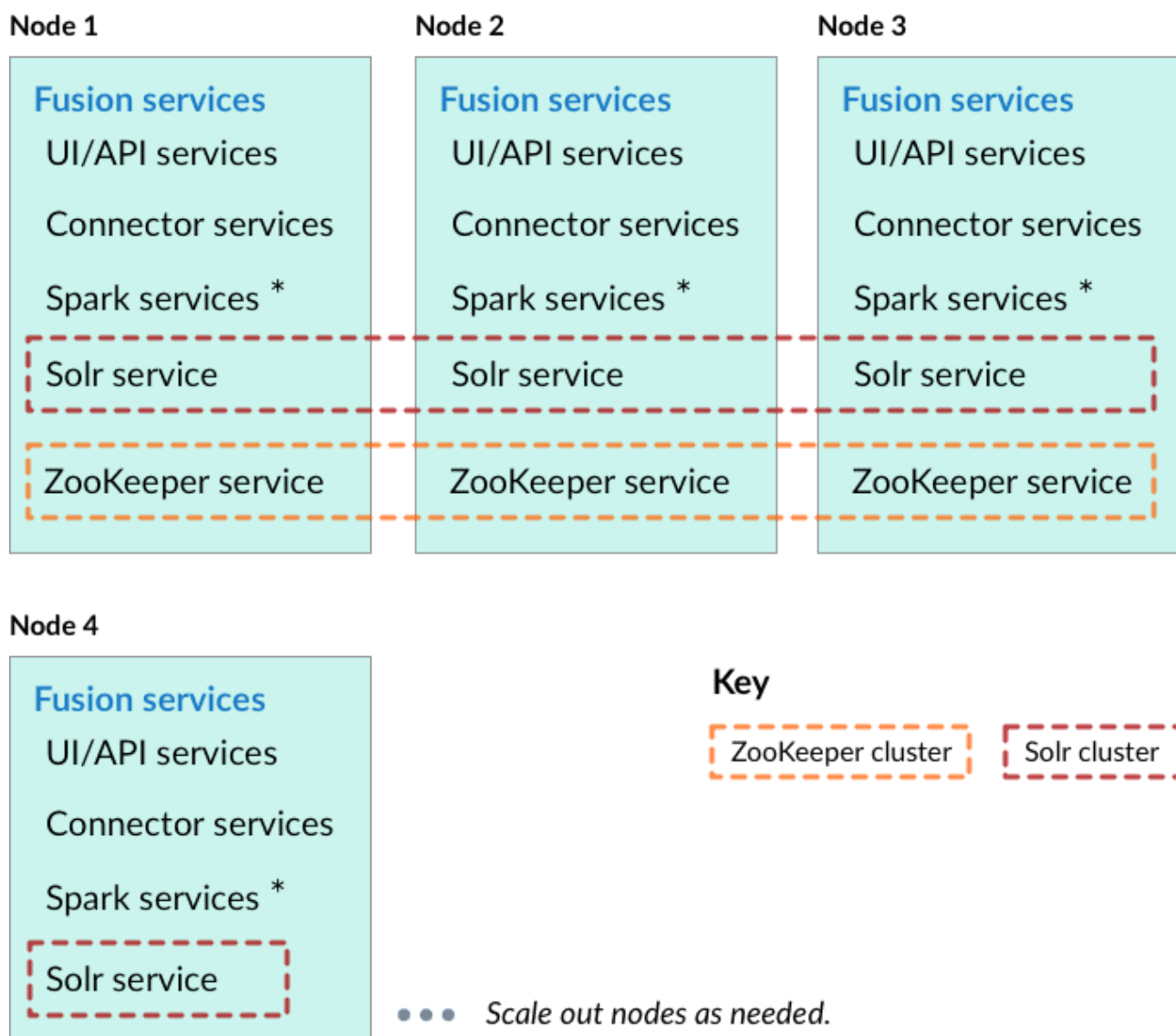
Regarding Solr, if you already have SolrCloud clusters managing your data, you can *integrate them into a Fusion deployment*.

To satisfy processing requirements, install Fusion, ZooKeeper, and Solr on specific nodes. These are the possibilities:

Nodes running core Fusion services and Solr also run ZooKeeper

In this cluster arrangement, a ZooKeeper cluster runs on the same nodes that run core Fusion services and Solr. This arrangement works well for a small cluster with low usage and can reduce cost when compared with placing ZooKeeper on separate nodes.

Fusion cluster



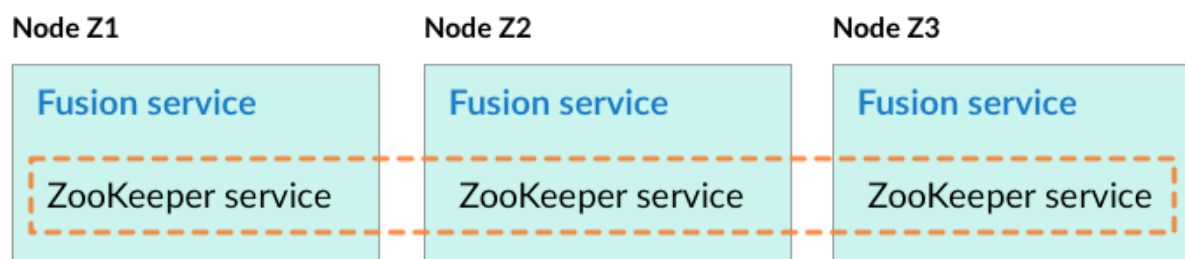
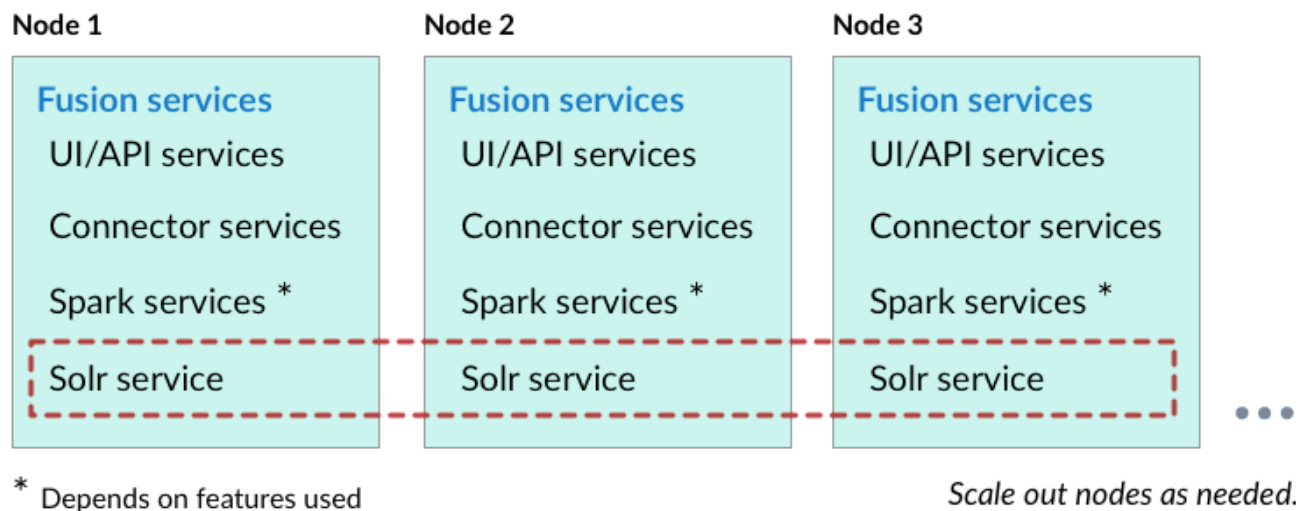
* Depends on features used

Here are installation instructions for *Unix* and *Windows*.

Nodes running ZooKeeper are not running core Fusion services or Solr

In this cluster arrangement, the ZooKeeper cluster runs on nodes in the Fusion cluster on which core Fusion services and Solr are not running. This arrangement is good for larger or more active clusters and can help reduce write latency. This also lets Fusion nodes scale horizontally without impacting the ZooKeeper nodes.

Fusion cluster



Key



Here are installation instructions for *Unix* and *Windows*.

On-Prem Deployment

These topics explain how to plan for, install, and upgrade your Fusion Server deployment on-premises:

Introductory and reference information

- **Review system requirements** - Start here to learn about required software and hardware.
- **Review deployment types** - Learn about deployment types for demos, trials, development, and production.
- **Start and stop Fusion Server** - Start and stop everything or individual services.
- **Learn about directories, files, and ports**
- **Review licensing information**

Install Fusion Server

- **Install Fusion Server on a single node (Unix)** - Install Fusion Server on a single Unix node. This is an easy way to try Fusion Server. Out-of-the-box, Fusion Server uses bundled instances of Solr, ZooKeeper, and Spark.
- **Install Fusion Server on a single node (Windows)** - Install Fusion Server on a single Unix node. This is an easy way to try Fusion Server. Out-of-the-box, Fusion Server uses bundled instances of Solr, ZooKeeper, and Spark.
- **Learn about installing a Fusion cluster** - You can cluster Fusion Server, Solr, and ZooKeeper nodes in a number of ways.
- **Install a Fusion cluster (Unix)** - Install multiple instances of Fusion Server, and of Solr and ZooKeeper, in a cluster.
- **Install a Fusion cluster (Windows)** - Install multiple instances of Fusion Server, and of Solr and ZooKeeper, in a cluster.
- **Integrate Fusion with an existing Solr deployment** - If you have an existing Solr deployment, you can integrate Fusion Server with that deployment (if it meets Fusion Server's version requirement for Solr).
- **Troubleshoot installation** - Troubleshoot difficulties that you might encounter when installing Fusion Server.

Upgrade and maintain Fusion Server

- **Upgrade Fusion** - Run a migrator to upgrade Fusion Server.
- **Migrate Fusion objects by hand** - Migrate Fusion objects by hand. You might need to do this for objects that the migrator does not migrate automatically. You can also use this manual approach to migrate Fusion objects from development environments into testing and production environments, or to back up data and restore it after an incident of data loss.
- **Add or move Fusion Server nodes** - Add a new node to an existing Fusion cluster, or move Fusion from one node to another.

Fusion UI

The Fusion UI gives you easy access to all of Fusion Server's features.

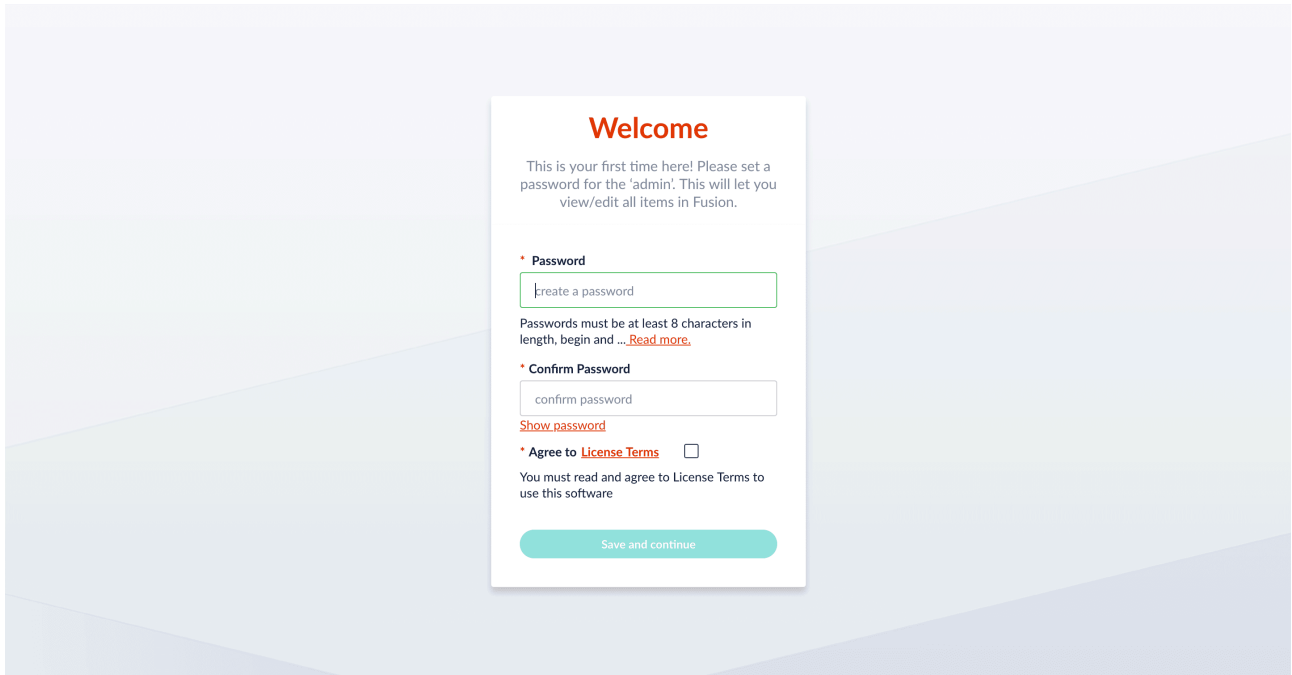
The UI service utilizes the API service, so each UI feature has an API equivalent. For programmatic access to the features you see in the UI, see the *API Reference Guide*.

Logging in

The Proxy service runs on port 8764 by default. Access the Fusion UI through the Proxy service. To log in, point your browser to this port on the Fusion host, for example, <http://localhost:8764/>.

When you are the *first user* to log in:

Fusion Server prompts you to set the password for the "admin" account and agree to the license terms. All other accounts must be created by an admin.

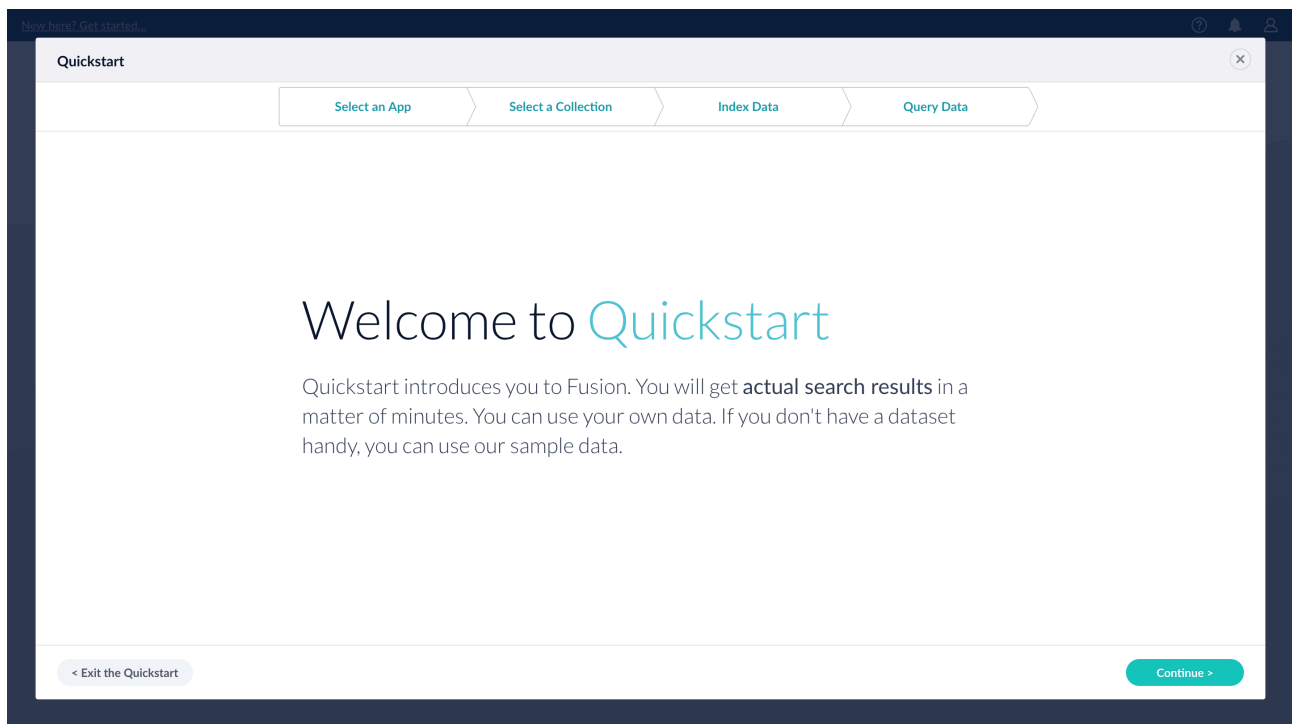
The image shows a 'Welcome' dialog box in the Fusion UI. The title 'Welcome' is in red. Below it, a message says: 'This is your first time here! Please set a password for the 'admin'. This will let you view/edit all items in Fusion.' There are two input fields: 'Password' with a placeholder 'create a password' and 'Confirm Password' with a placeholder 'confirm password'. A note states: 'Passwords must be at least 8 characters in length, begin and ...[Read more](#).' Below the confirm password field is a link '[Show password](#)'. At the bottom, there is a checkbox for 'Agree to [License Terms](#)' with the text 'You must read and agree to License Terms to use this software' underneath. A green 'Save and continue' button is at the very bottom.

Fusion Server also prompts you to register to send metrics to Lucidworks. You can click **Skip** to send no metrics; you can also register later.

The first time you log in:

Fusion Server opens the *Quickstart*, to give you a quick demonstration of Fusion's most basic features. You can click **Exit the Quickstart** to close it and go to the Fusion launcher.

You can open the Quickstart at any time by navigating to the Fusion launcher and clicking **New here? Get started...**



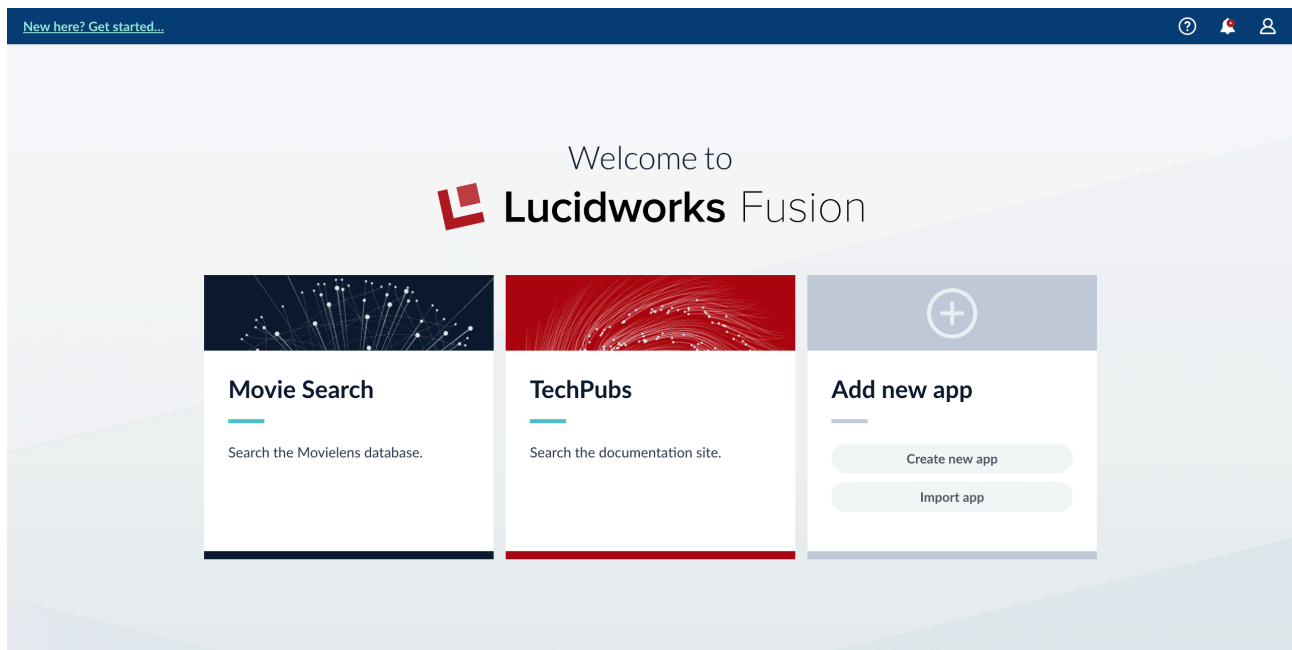
The brief tour below shows you how to find your way around the interface. For a more detailed tour, try the *Get Started*.

A brief tour

- The *launcher* is where you can view all of the apps in your Fusion environment and create new ones.
- The *Fusion workspace* is where you configure how an app ingests, indexes, queries, and analyzes data.

The Fusion launcher

When you log in to Fusion Server, the first page is the Fusion launcher. This is where you create and launch apps:

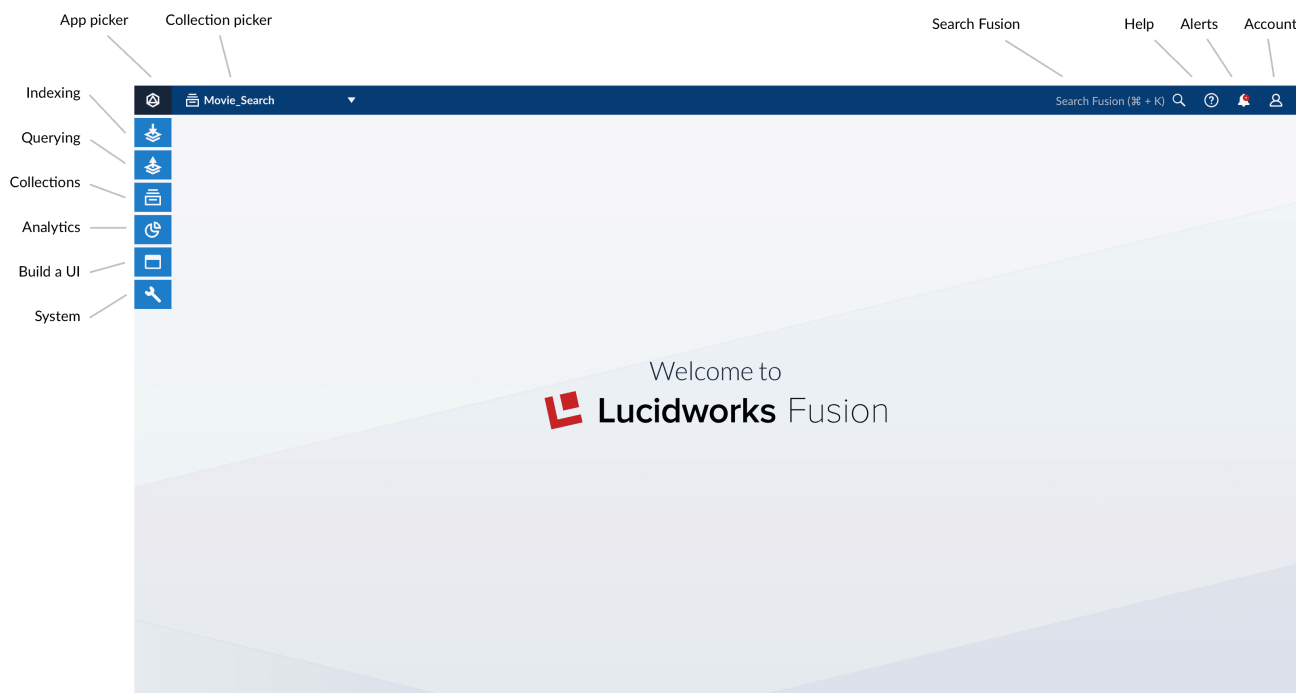


To return to the Fusion workspace, hover over , and then click **Return to launcher**.

See details about *creating*.

The Fusion workspace

You can reach the Fusion workspace by clicking on any app in the launcher. When you enter the Fusion workspace, you find all the tools you need to configure the currently-selected app:



- On the left side are the *app tool menus*, where you can find tools for working with your app.
- At the top, you can use *Object Explorer* to search for and browse objects that are in the current app, in any app, in no apps, and in any app or no apps.
- *System-wide tools* are in the upper right corner.

App tool menus

On the left side of the screen are five menus where you can find tools for working with your app:

-  Indexing (getting data in)
The *Index Workbench* is the primary tool for getting data into your app through the Fusion UI. It provides an all-in-one view of the components of a datasource. You can also access those components individually through this menu.
-  Querying (getting data out)
For getting data out, the *Query Workbench* is the main tool. It shows you a live preview of search results while you configure and fine tune your query pipelines.
See also *Getting Data Out*.
-  Collection management
Manage your data collections and configure their features.
See also *Collection Management*.
-  System administration
System administration for your apps.
See the *System Administration with the Fusion UI*.

Finding objects in Object Explorer

At the top of the Fusion workspace is a Find Objects search field. When you click **Find Objects**, *Object Explorer* opens.



In the **Filter** search box, you can search by name or type for all *types of Fusion objects*, such as collections, apps, pipelines, and more. Browse objects by clicking them. You can use Object Explorer to view and configure the links between objects. For example, you can use it to *share an object between multiple apps*.

System-wide tools

On the right are menus for system-wide actions:

-  Help
Click here to access the documentation from the Fusion UI.
-  Alerts
New alerts are briefly displayed in the foreground. Open this menu to see past alerts, or to clear the alert history.
-  Account
Here you can edit your profile, log out, or manage your Fusion license.

Querying

Querying

Query Pipelines

Query pipelines process search requests before sending them to Fusion Server's Solr core. A query pipeline can perform transformations on a search request in order to customize the search results that Solr returns. See *Query Pipeline Configuration*.

Search Applications

Search applications are the front-end interfaces that you build on top of Fusion. Your application makes calls to Fusion's REST API in order to retrieve search results or perform other actions. Certain features, like autocomplete and synonyms, require some configuration on the Fusion back end. See *Application Development*.

Query Language

Whenever you are getting data out of Fusion Server, you may find it handy to consult the *Query Language Cheat Sheet*.

If you are using Fusion AI with Fusion Server, you can generate and retrieve additional data for analysis or to enhance the end-user experience. For example, Fusion AI can produce sophisticated recommendations to guide end users to the best available results, including results that do not exactly match the original user-submitted query. It can also perform machine learning functions that automatically improve search results based on the past activities of users. See the *Fusion AI documentation*.

Building Applications

Ultimately, Fusion Server is the back end for your search applications.

Fusion App Studio is a tool for developing a search interface from within the Fusion workspace, using a simple configuration wizard and an integrated code editor. You can set up a simple search interface in just minutes, and then customize it to suit your organization.

If you develop your own search application from scratch, it must make calls to Fusion's *REST API* in order to retrieve search results or perform other actions, such as uploading blobs, modifying user roles, or sending messages.

Certain features require configuration on the Fusion back end before you can surface them in your application:

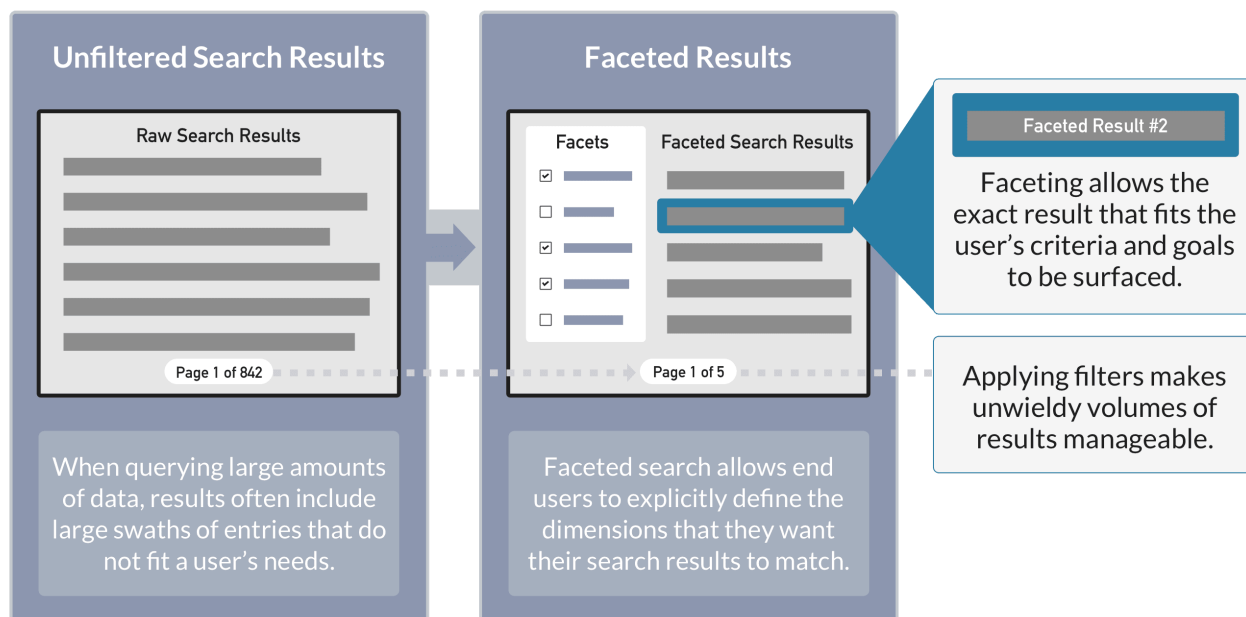
- *autocomplete*
- *faceting*
- *synonyms*
- *stopwords*

If you are using *Fusion AI*, then your application can also send *signals* to Fusion and retrieve *recommendations* to personalize each user's search experience.

Autocomplete

Autocomplete, also known as auto-suggest, look-ahead, or type-ahead, is a feature that displays a list of common search terms that begin with, or are contained by, the query string. For example, the query "search technology" might result in an autocomplete list with results "search technology rocks", "i love search technology", and/or "this is a search technology company", depending on how the autocomplete has been configured. Like most other search components, Fusion leverages Solr's autosuggest (<https://solr.apache.org/guide/suggester.html>) component. Assuming auto-suggest is configured in the Solr config, it is sufficient to enable it in Fusion in order to use the component.

Faceting



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 (<https://cwiki.apache.org/confluence/display/solr/Faceting>) search components.

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.

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 (<https://cwiki.apache.org/confluence/display/solr/Faceting#Faceting-RangeFaceting>).

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 (<https://wiki.apache.org/confluence/display/solr/Faceting#Faceting-Thefacet.methodParameter>) 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.

Further Reading

<https://lucidworks.com/post/pivot-facets-inside-and-out/> (<https://lucidworks.com/post/pivot-facets-inside-and-out/>)

<https://lucidworks.com/post/apache-solr-statistics/> (<https://lucidworks.com/post/apache-solr-statistics/>)

<https://lucidworks.com/post/pivoting-to-the-query-using-pivot-facets-to-build-a-multi-field-suggester/>
(<https://lucidworks.com/post/pivoting-to-the-query-using-pivot-facets-to-build-a-multi-field-suggester/>)

Search Applications

Ultimately, Fusion is the back end for your own search applications.

- Your application uses Fusion's REST API to interact with the Fusion system. The REST API supports all the features available in the Fusion UI. At a minimum, your application will employ the `/query-pipelines/{id}/collections/{collection}/{handler}` endpoint to query Fusion collections.
- *Recommendations* are a way to use aggregations to enhance the search experience. Based on the current search, or signals collected previously, Fusion can return results that are relevant in the end user's current context.
- Certain front-end features require some Fusion configuration:
 - *Autocomplete*
 - *Faceting*
 - *Stopwords*
 - *Synonyms*

Stopwords Files

Fusion collections are Solr collections that are managed by Fusion. Solr itself manages a set of resources (<https://wiki.apache.org/confluence/display/solr/Managed+Resources>) for a collection.

Stopword lists are one such resource.

Stopwords are words that are filtered out of a user-entered search request. Examples of common English stopwords are *a*, *an*, and *the*.

The Fusion UI provides a stopwords manager tool that is reached from the **Stopwords** page in the **Collections** menu. This tool lets you view the contents of all configured stopwords files in an editable browser.

Synonyms Files

Synonyms

Synonyms are:

- Words that mean the same thing, within the context where they are used.
- Used in searches. And synonym expansion allows Fusion to return results that match the meaning of the query terms, but not the words themselves.
- Important for mapping query terms such as:
 - Acronyms to their names
 - Jargon to public terms
 - Misspellings to correct spellings
 - Old to new personal or corporate names
 - Bridging the gap between the user vocabulary and terms in the original text

Fusion uses the Solr `synonyms.txt` and Solr collections, which are managed by Fusion.

Solr itself manages a set of resources (<https://cwiki.apache.org/confluence/display/solr/Managed+Resources>) to apply synonym expansion, with configuration through the Fusion API and the Fusion UI. However, Fusion synonyms are not interchangeable with Solr synonyms files.

Synonym types

There are three kinds of search synonyms, depending on the requirements of the search for each specific term.

Replacement synonyms

Replacements are used to change the query, to replace it with a more standard term or terms. For example:

```
lucid => lucidworks
```

In this case, "lucid" by itself not an approved term, so there should be no instances where the company name is a partial word.

One-way expansion synonyms

One-way expansions expand original terms with more standard terms while retaining the original term; they do not do the opposite, expand standard terms to the original non-standard terms

```
monitor => monitor, display
```

In this case, "display" is the standard term, but "monitor" is used in some older user-generated content.

Multi-way expansion synonyms

Where each term is considered equally standard, multiway synonyms expand the query so any items with any of the terms is retrieved.

```
login, logon, signin, signon
```

This example shows terms that are used interchangeably by authors. For this search engine, there is no need to distinguish among them, and considerable value in increasing recall to find all items discussing this topic. However, other content stores may use them

differently. Note that "logging" and "signing" have specific meanings in many contexts, so they might not be candidates for synonyms.

Example of synonym expansion:

Results before synonyms	Results after synonyms
No Search Results No results found for the provided query/filters. Try indexing some data or enabling a Solr Query pipeline stage. Search executed in 3 ms	Configuring Fusion for Kerberos https://doc.lucidworks.com/fusion/3.1/Security/Access_Control/Configuring-Fusion-for-Kerberos.html Body t: request was sent from a browser, Fusion doesn't display the initial sign-on panel; instead on login, the Score: 6.196554 show fields Getting Started with Fusion https://doc.lucidworks.com/fusion/3.1/Tutorials/Getting_Started/Part-4_UX.html Body t: : /view.sh start When you start View, usually your browser opens to the login page automatically. If not Score: 4.575011 show fields Hadoop Connector and Datasource Configuration https://doc.lucidworks.com/fusion/3.1/Connectors_and_Datasources_Reference/Hadoop-Connector-and-Datasource-Configuration.html Body t: ' fusion_login_app_name Config App Name Login Config App Name FusionClient by default. type: string default value Score: 4.5661583 show fields 1 2 Next 1-10 of 18 docs (45 ms, max-score 6.196554)

Viewing the query using the debug=true parameter shows how it is expanded:

```
"querystring": "login",
"parsedquery": "(+DisjunctionMaxQuery((Synonym(_text_:login _text_:login _text_:signin _text_:signon)))/no_coord",
"parsedquery_toString": "+(Synonym(_text_:login _text_:login _text_:signin _text_:signon))",
```

Multiword synonyms

Lucene/Solr supports multiword synonyms in version 6.6 and later, and Fusion in version 3.1 and later. There are significant technical complexities in performing graphed phrased expansion that had to be overcome.

To enable multiword synonyms in Fusion, create an Additional Parameter stage for disabling the split on whitespace tokenization process (which applies to synonyms only):

```
sow=false
```

Using EDismax, this allows the new Solr SynonymGraphFilter to create the graph representations of token streams containing overlapping synonyms of varying word counts, and expand the queries with additional terms. If using a less-than sign (<) with EDismax, it must be escaped using a backslash.

Examples:

```
appstudio => app studio
signup => signup, sign up
login, log in, logon, log on, signin, sign in, signon, sign on
```

Multiword synonyms work just like single-word synonyms, expanding the parsed query with additional query terms. For Solr details, see: [Multi-Word Synonyms: Solr Adds Query-Time Support \(https://lucidworks.com/post/multi-word-synonyms-solr-adds-query-time-support/\)](https://lucidworks.com/post/multi-word-synonyms-solr-adds-query-time-support/).

Synonyms Editor API

The *Synonyms Editor API* provides complete REST access to all aspects of the synonyms.

Query Pipelines and Profiles

- *Query pipelines* filter, transform, and augment Solr queries and responses in order to return all and only the most relevant search results.
- *Query pipeline stages* are the components of a query pipeline, configured and ordered to produce the desired search results.
- *Query profiles* provide aliases for query pipelines, enabling your application to point to the same endpoint regardless of the query pipeline behind it. A query profile can also specify the collection and/or configuration parameters for the query pipeline, if abstraction is desired for those aspects.
- *Custom Javascript query stages* are a versatile way to handle a wide range of special requirements in your pipeline.
- The *stored parameters collection* stores Solr facet parameters for a particular document or product category.

Custom JavaScript Stages for Query Pipelines

The JavaScript Query stage allows you to write 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 objects and Response objects, respectively.

JavaScript Stages Variables and Parameters

JavaScript (<http://en.wikipedia.org/wiki/JavaScript>) is a lightweight scripting language. The JavaScript in a JavaScript stage is standard ECMAScript (<http://en.wikipedia.org/wiki/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
<code>request</code>	<i>Request</i>	The Solr query information.
<code>response</code>	<i>Response</i>	The Solr response information.
<code>ctx</code>	<i>Context</i>	A reference to the container which holds a map over the pipeline properties. Used to update or modify this information for downstream pipeline stages.
<code>collection</code>	String	The name of the Fusion collection being indexed or queried.
<code>solrServer</code>	<i>BufferingSolrServer</i>	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 <i>SolrClient</i> (https://lucene.apache.org/solr/5_2_1/solr-solrj/org/apache/solr/client/solrj/SolrClient.html) for details.
<code>solrServerFactory</code>	<i>SolrClientFactory</i>	The <i>SolrCluster</i> server used for lookups by collection name which returns a <i>Solr</i> server instance for a that collection. For example: <code>var productsSolr = solrServerFactory.getSolrServer("products");</code>
<code>QueryRequestAndResponse</code>	<i>QueryRequestAndResponse</i>	Returns list of info as a string.

Syntax Variants

JavaScript stages are written using *function syntax*, which passes variables as function parameters.

Function Syntax

```
function(request,response) {  
    request.addParam("foo", "bar");  
}
```

Function syntax is used for the examples in this document.

Global variable `logger`

The global variable named `logger` writes messages to the log file of the server running the pipeline. This variable is truly global and does not need to be declared as part of the function parameter list.

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".

See also

- *Fusion Pipeline Javadocs*
- *JavaScript Query Stage*
- *Custom JavaScript Query Stage Examples*
- *Custom JavaScript Stages Variables and Parameters*
- *Managed JavaScript Query Stage*

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.

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 to create, list, update, or delete query stages using JSON. See also the *Query Pipelines API*.

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*.

Default Query Pipelines

When you create a new app with a default collection, the collection includes a default query pipeline. When you create a new collection in an existing app, Fusion also creates a default query pipeline for the new collection. The pipeline name is the same as the collection name.

The default query pipeline has the following pre-configured stages:

- *Boost with Signals*
The Boost with Signals query pipeline stage uses *aggregated signals* to selectively boost items in the set of search results.
- *Query Fields*
The Query Fields query pipeline stage defines common Solr query parameters for the edismax query parser (<https://cwiki.apache.org/confluence/display/solr/The+Extended+DisMax+Query+Parser>). If using a less-than sign (<) with DisMax, it must be escaped using a backslash. An alternative to this stage is the *Additional Query Parameters stage*.
- *Field Facet*
The Field Facet query pipeline stage is used to add a Solr Field Facet query (<https://cwiki.apache.org/confluence/display/solr/Faceting#Faceting-Field-ValueFacetingParameters>) to the search query pipeline.
- *A Solr Query*
The Solr Query stage transforms the Fusion query pipeline *Request* object into a Solr query and sends it to Solr.

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.

Query Profiles

Query profiles let you consistently point your search application at a static endpoint, but give you the flexibility to change the actual query pipeline being used (and optionally, the collection and/or configuration parameters for the query pipeline).

For example, an e-commerce site might want to create a query pipeline to support a month-long promotion. After the query pipeline is configured, it can be easily enabled by changing the query profile in use by the front-end application to use the new pipeline.

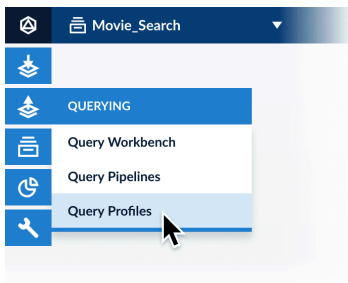
To run a query through a query profile, use the REST API as described below. For CRUD operations, you can use either the REST API or the Fusion UI.

Query profiles in the REST API

- *Query Profiles API* (`/query-profiles`)
Create, read, update, and delete query profiles.
- *Query API* (`/query`)
Run a query through a query profile by specifying the profile ID and appending the request with a *Solr query string*, as in `/api/query/<id>?<solrQuery>`.

Query profiles in the UI

Query profiles are configured at **Querying > Query Profiles**.



Stored Parameters Collection

A Stored Parameters collection contains lists of parameters, where each list is has a unique identifier called a "key". These lists of parameters can be plugged into Fusion query pipelines. The Stored Parameters facility provides a mapping layer between the keys and specific actions carried during search and is somewhat similar to database triggers or stored procedures. The use for this collection is to store Solr facet parameters for a particular document or product category, where the facet parameters are stored as a JSON object. An example of what the input JSON is shown in a section below.

A Stored Parameters collection is an auxiliary Fusion collection which is associated with a primary collection via naming conventions: for a primary collection, the name of the associated Stored Parameters collection consists of the name of the primary collection plus the suffix "_stored_parameters". This Stored Parameter collection is used by a *Retrieve Stored Parameters query stage*.

In the Fusion UI, Stored Parameter collections are not displayed in the list of collections in the Collections Admin panel, nor are there controls on the UI for creating or populating these collections; all collection administration is done via calls to the REST API.

Stored Parameters for Facets

- An existing facets configuration can be converted into the required format via a once-off custom script (e.g. written by Lucidworks solution engineers or by a customer's technical team).
- Note that if a caller wants particular behavior e.g. specifying that all results should be from a given category, then it is up to them to specify that in their original request (e.g. using a filter query "fq" setting).
- A new Stored Parameter Query Stage can then be used in a query pipeline to add the required parameters for a given key (e.g. a category ID).

Stored Parameters Example

The following example shows how faceting can be carried out over a hierarchy of categories and subcategories where there are many known subcategory-specific attributes to query on. The example data is taken from the Best Buy product catalog data.

In this list of stored parameter objects, the set of fields used are:

- "id" : record ID (or "key" in the sense of a "Key-Value" store) of the category to facet on.
- "label_s" : human-readable label for this record, i.e., the category name
- "in_ids_ss" : parent categories, if an intermediate category in a category hierarchy. Intended use is in traversing a directed graph of categories.
- "out_ids_ss" : child categories, if an intermediate category in a category hierarchy. Intended use is in traversing a directed graph of categories.
- "parameters_ss" : list of arbitrary Solr parameters that can be used to construct a query that is relevant for the entity described by this record. In this example, it is the list of facet parameters to use for this specific category.

```
[ { "id": "abcat0600000", "label_s": "Movies & Music", "in_ids_ss": ["cat00000"],  
  "out_ids_ss": ["cat02001", "cat02015"] }, { "id": "cat02001", "label_s": "Music",  
  "in_ids_ss": ["abcat0600000"], "out_ids_ss": ["cat02673", "cat02012", "cat02004"]  
}, { "id": "cat02010", "label_s": "Rock", "in_ids_ss": ["cat02001"],  
  "out_ids_ss": ["cat02673", "cat02687", "cat02661", "cat02662", "cat02675",  
  "cat02678", "cat02680"], "parameters_ss": ["facet.field=attr_format_",  
  "facet.field=attr_albumLabel_", "facet.field=attr_genre_"] }, { "id": "cat02673",  
  "label_s": "Hard Rock", "in_ids_ss": ["cat02010"], "parameters_ss":  
  ["facet.field=attr_format_", "facet.field=attr_albumLabel_",  
  "facet.field=attr_genre_"] }, { "id": "abcat0208001", "label_s": "Batteries" }, {  
  "id": "abcat0208002", "label_s": "Alkaline Batteries", "in_ids_ss":  
  ["abcat0208001"], "parameters_ss": ["facet.field=attr_hardGoodType_"] } ]
```

Direct Access and Navigation

A caller can query the `<collection>_stored_parameters` system collection directly.

For example, a Solr request to get the human-readable label stored in field "label_s" for a specific category ID is:

```
`http://127.0.0.1:8983/solr/<collection>_stored_parameters/query?q=id:
<category_id>&fl=label_s`
```

Direct access allows for navigation up and down a hierarchy of categories by listing incoming and outgoing links that specify a category's location. For example, the Solr requests to find the parent(s) and children of a given category, respectively, are:

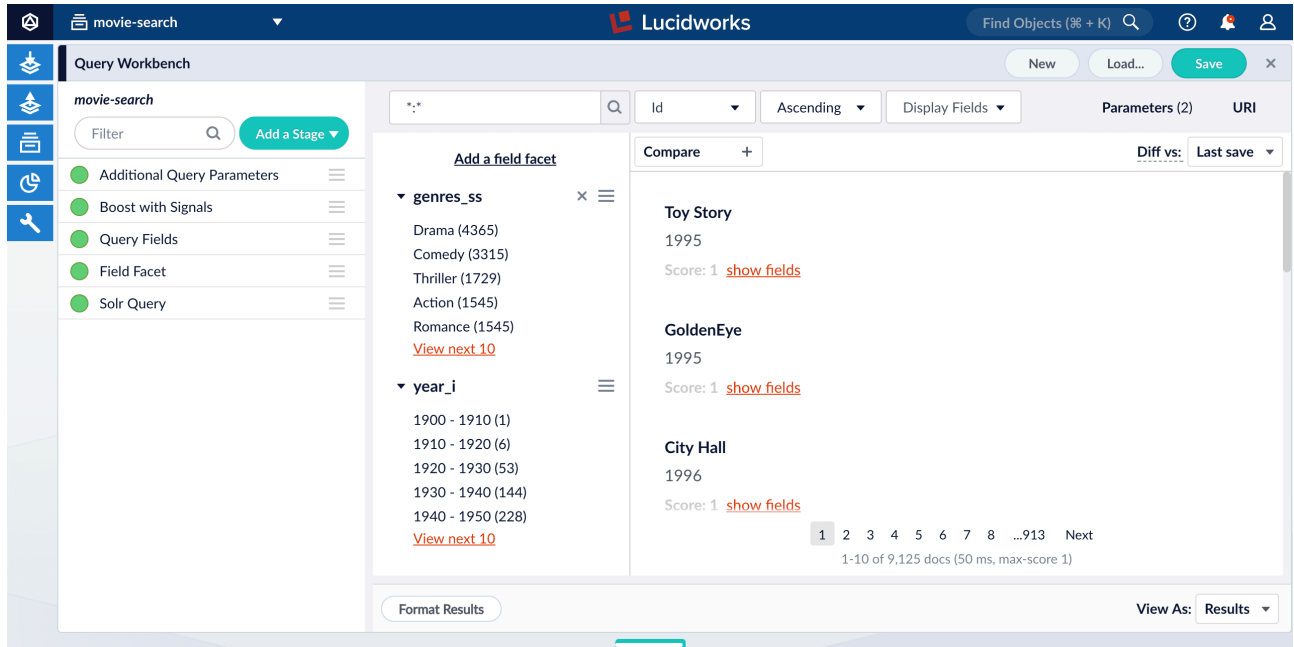
```
`http://127.0.0.1:8983/solr/<collection>_stored_parameters/query?q=id:
<category_id>&fl=in_ids_ss`
`http://127.0.0.1:8983/solr/<collection>_stored_parameters/query?q=id:
<category_id>&fl=out_ids_ss`
```

Note that this kind of "graph navigation" can become inefficient if it requires submitting a large number of queries to navigate up and down the hierarchy.

Query Workbench

The Query Workbench is where you edit query pipeline stages, results parameters, and more, then preview the impacts of the changes in real time before saving them. Relevancy tools such as boosting and blocking can be accessed and altered through the Query Workbench. The Query Workbench streamlines the process of querying indexed results and fine-tuning Query Pipelines to surface the results that best satisfy the end user's needs.

See the *Query Language Cheat Sheet* for help constructing queries.

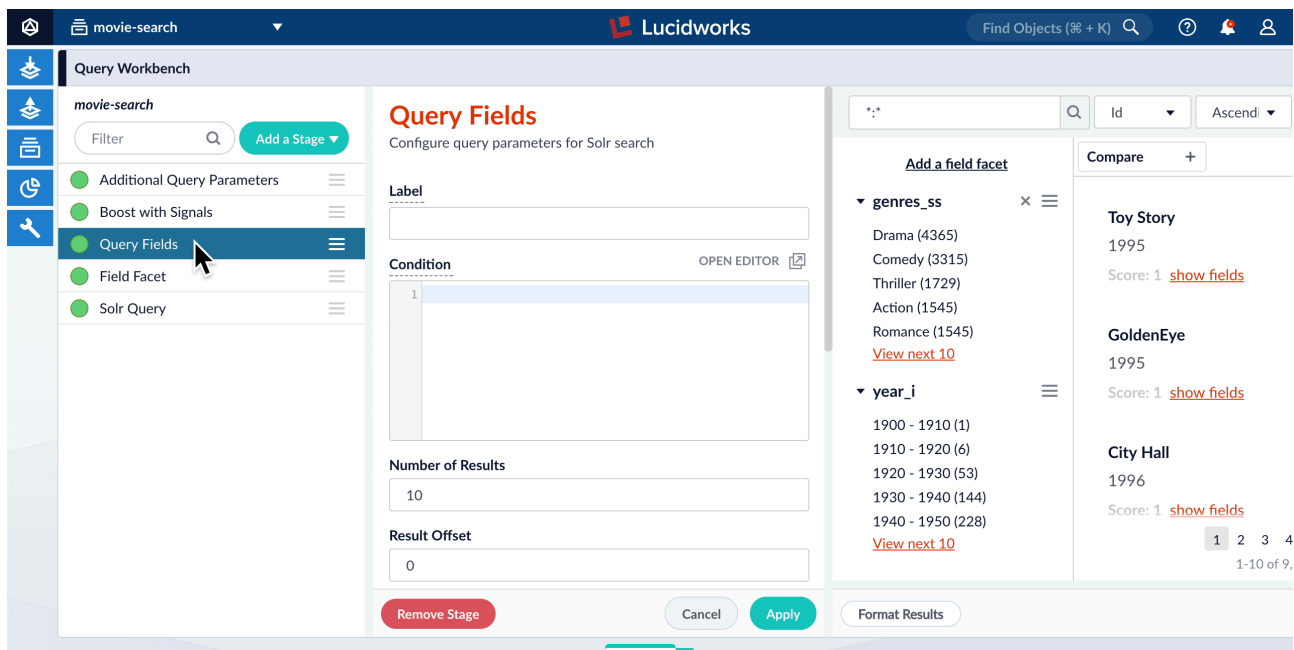


With this sandbox paradigm, you can experiment with query pipelines without consequence because Fusion uses a copy of a pipeline to simulate the new results, and it is not permanently modified in the collection until you click **Save**.

Query pipelines and pipeline stages

Query pipelines work by processing search requests and returning an ordered list of matching documents. Each pipeline consists of a series of query stages that can be added, ordered, and configured using the Query Workbench.

Select any stage in the pipeline to open its configuration panel:



Boosting and blocking

As you search your data and inspect the results, you can manipulate the rankings of individual documents. Boosting a document raises its ranking, while blocking a document removes it from search results.

Boosting and blocking affects the results for specific queries. For example, when you search for "citizen" and boost the document for "Citizen Kane", your end users will see that title boosted to the top when they also search for "citizen", but not necessarily when they search for "kane".

The Query Workbench provides convenient buttons for instantly blocking or boosting the documents that you see in the preview panel. Clicking **Boost** automatically adds the *Boost Documents stage* to your query pipeline, with a boost rule that matches your query and the document you clicked. Likewise, clicking **Block** adds and configures the *Block Documents stage*.

Compare mode

Compare mode allows you to compare search results using two different query pipelines, or compare search results from the original query pipeline and the copy being modified in the Query Workbench. On the right is the working pipeline, which you can edit. On the left is one of your saved pipelines.

Security

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.

Fusion uses a number of security measures:

- **Authenticating UI users** – Fusion authenticates users when they log in. Logging in creates a new Fusion session. Fusion also authenticates users when the *Sessions* REST API creates a session.
- **Authorizing UI users** – Fusion authorizes users to use specific parts of the Fusion UI.

UI users must also be authorized to make API requests, because the UI makes API requests.
- **Audit trail (log) with USERID generated for all successful and failed login attempts.** Audit logs are sent to a Solr collection *system logs*. These logs are distinguished from other system logs in the collection by the following attribute:
 - `logger_class_s:"com.lucidworks.cloud.gateway.audit"`
- **Authenticating and authorizing users who make API requests**
- **Password Encryption**– Fusion uses 128-bit AES keys to encrypt passwords and "AES/CBC/PKCS7Padding" for the cipher. The ciphertext is also signed.
- **Using session cookies**
- **Using an external authentication provider** (optional) – A security realm can specify use of an external authentication provider, such as LDAP, JWT, or SAML.
- **Using SSL/TLS to ensure that data in flight between your application and Fusion is not observable** (optional) – See *SSL Security (Unix)* or *SSL Security (Windows)*.
- **Constraining the documents that are indexed** (optional)
- **Trimming the documents that are returned by queries based on authorization** (optional)

Fusion user login

When logging into the Fusion UI, a user provides a username and password, as well as their assigned security realm. An administrator must specify these in Fusion (using the native security realm) or configure Fusion to use an external authentication provider (for example, LDAP or SAML). See *Access control*.

Fusion uses *roles* defined by *permissions* to authorize Fusion UI access and perform tasks in Fusion, including searching. The recommended method to delegate permissions is as follows:

- Assign each user to a role and create custom roles as needed.
- Assign permissions on a per-app basis.

Manage users with security realms

Fusion uses *security realms* to authenticate users of the Fusion UI. Each user has an assigned security realm, which the user must select when logging in. If the user selects a different realm, authentication fails.

A security realm also provides a list of roles as follows:

- The list always includes the role(s) that are specified in the security realm.
- (Optional) If an external directory service (such as LDAP) is used for authentication, the list can also contain roles that are mapped from the names of the directory-service groups. That is, you can configure a security realm to return group information for users from *that same* directory service.
- (Optional) The security realm can reference one or more Fusion roles or, when using an external directory service provider, use group membership information from the provider to determine roles for users. Fusion maps the group names to role names and adds these roles to the user's list of roles.

Fusion does not use permissions from LDAP to authorize UI access or API requests. It only obtains group names (optionally), which are used as role names or 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.

Per-Request Authentication

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* and its layering is at request time.

You can define multiple *security realms* for a Fusion instance. A Fusion instance can manage multiple security realms, which allows users from different domains to have (different levels of) access to specific Fusion collections.

Access Control

These additional topics explain how to configure the supported authentication methods:

- *Configure Fusion for JWT*
- *Configure Fusion for Kerberos in Unix and Windows*
- *Configure Fusion for LDAP*
- *Configure Fusion for SAML*
- *Configure Fusion for SSO*

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*.

User Account Administration

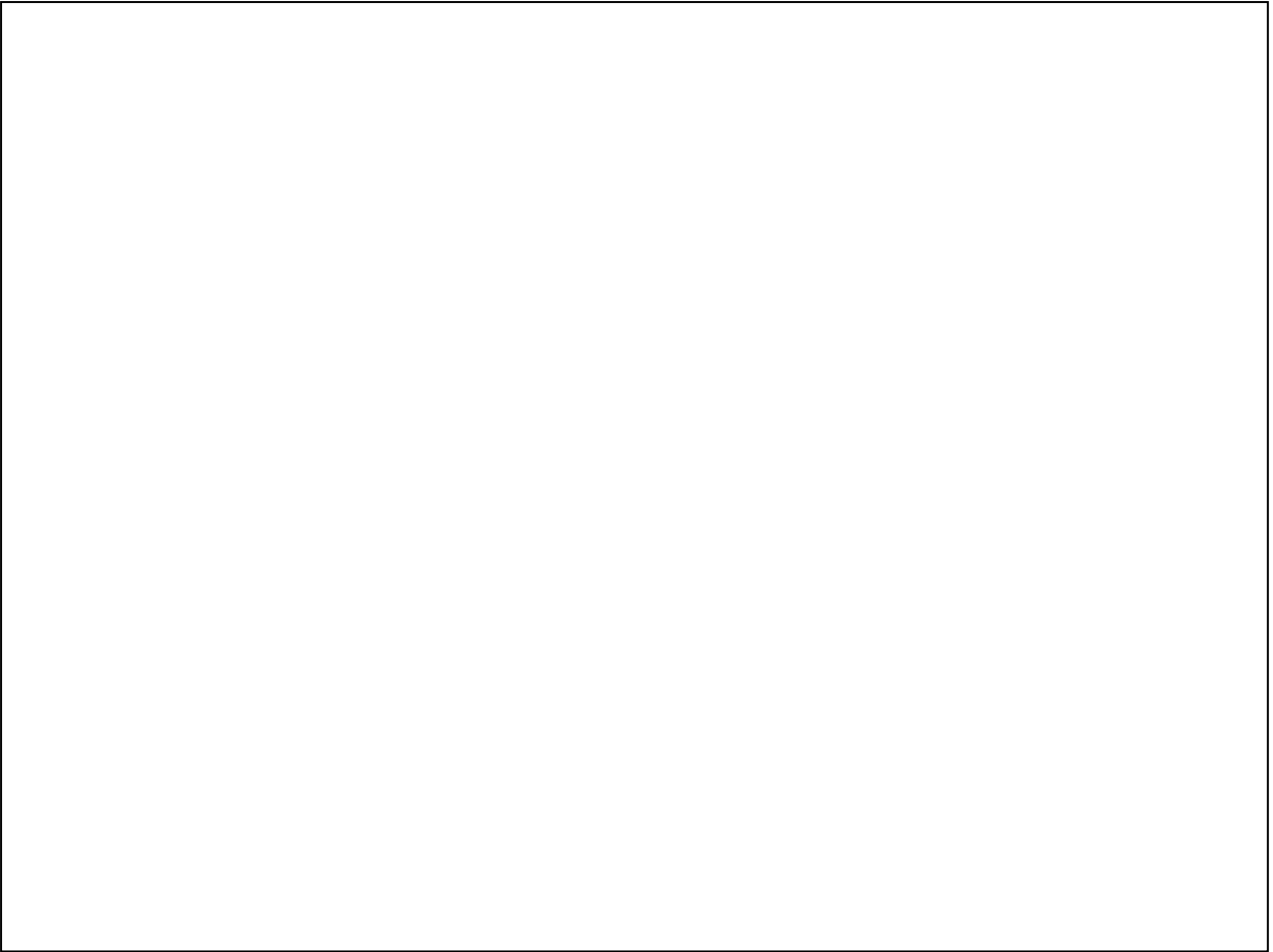
A Fusion *Security Realm* encapsulates a user database together with specific authentication and authorization mechanisms. This information is stored in ZooKeeper (<http://zookeeper.apache.org/>) so that it is always available to all Fusion components across the deployment.

Fusion's native security realm manages both authentication and authorization directly. All user information is stored in ZooKeeper: usernames, hashes of passwords, roles, and permissions. Passwords are hashed using bcrypt (<https://en.wikipedia.org/wiki/Bcrypt>). Authentication compares a hash of the entered login password with the stored password hash. 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 (<https://en.wikipedia.org/wiki/SPNEGO>) is used for authentication via Kerberos.
- Kerberos authentication, LDAP authorization - Fusion stores a local user record in ZooKeeper. SPNEGO (<https://en.wikipedia.org/wiki/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.
- JWT. JSON Web token.

Video tutorial



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.
- **API permissions** – Control which requests a user can submit to which REST API endpoints.

Permissions can be defined by either a role or a user, or both. Fusion combines 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 a role *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).
For example, say role A allows GET and POST access to a specific endpoint. User X is a member of role A and also has a user definition that allows only GET access to that endpoint. In this case, user X has only GET access to that specific endpoint. Alternatively, you could define less permissive roles.

Specify UI Permissions

Specify UI permissions in *roles*.

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/", followed by the service name and any methods and parameters. The permissions specification includes everything following `api/`. 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. Two wildcards match all possible values for any number of path segments. Granting access to a subset of Fusion functionality can require a list of narrowly defined permissions. Carefully defining each Fusion app to address a specific use case can simplify permission sets, by letting you grant access liberally within a single app.

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 separated by commas. If the endpoint specification has multiple variable, the semi-colon character ";" is used as the separator between parameter specifications.

A permission specification has three elements: methods, path, and parameters. A permissions specification is written as a string using the colon character ":" as the separator between the elements. The lists of permission specifications are stored in the ZooKeeper User and Roles entries in JSON format.

Example Permission Sets

Admin permissions

The permissions for the admin user can be written in a single line:

```
GET, POST, PUT, DELETE, PATCH, HEAD: /**
```

This permission specification uses two wildcards. One wildcard specifies any path segment, and the other specifies any value.

Best practice: Delegate access to a Fusion app

When possible, create a Fusion app for a specific use case and assign permissions on a per-app basis. For example, to configure a Fusion account to be able to run queries against only a certain collection:

1. Create a Fusion app.
2. *Create a query profile* by defining a collection. This lets you change the underlying collection without having to redo permissions.
3. Assign permissions as follows:

```
GET:/apps/APP_NAME/query/QUERY_PROFILE/*
```

The above permission set allows a user to run queries against the collection specified by `QUERY_PROFILE` under the Fusion app `APP_NAME`.

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.

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 do not override the other roles. They are added to the other roles.

Default Roles

At initial startup, Fusion creates a set of default roles for common types of users.

admin

The admin role is the equivalent to the Unix `root` or superuser. It allows full access to all Fusion services:

```
GET,POST,PUT,DELETE,PATCH,HEAD:/**
```

developer

The developer role has all the read/write permissions required for building and running applications.

```

GET,POST,PUT:/system/**
GET,POST,PUT,DELETE,HEAD:/stopwords/**
GET,POST,PUT:/usage/**
GET:/features/**
GET,POST,PUT,DELETE,HEAD:/blobs/**
GET,POST,PUT,DELETE,HEAD:/scheduler/**
GET,POST,PUT,DELETE,HEAD:/experiments
GET:/introspect/**
PUT:/usage/**
GET,POST,PUT,DELETE,HEAD:/index-stages/**
GET,POST,PUT,DELETE,HEAD:/messaging/**
GET,POST,PUT,DELETE,HEAD:/catalog
GET,POST,PUT,DELETE,HEAD:/parsers/**
GET,POST,PUT:/appkit/**
GET,POST,PUT,DELETE,HEAD:/index-profiles/**
GET,POST,PUT:/recommend/**
GET,POST,PUT,DELETE,HEAD:/history/**
GET,POST,PUT,DELETE,HEAD:/apps/**
GET,POST,PUT,DELETE,HEAD:/solr/**
GET,POST:/query/**
GET,POST,PUT:/signals/**
GET,POST,PUT:/searchLogs/**
GET,POST,PUT,DELETE,HEAD:/query-pipelines/**
GET,POST,PUT:/configurations/**
GET:/suggestions/**
GET,POST,PUT,DELETE,HEAD:/searchCluster/**
GET,POST,PUT,DELETE,HEAD:/index-pipelines/**
GET:/license
GET,POST,PUT,DELETE,HEAD:/spark/**
GET,POST,PUT,DELETE,HEAD:/query-stages/**
GET,POST,PUT,DELETE,HEAD:/prefs/apps/search/*
GET:/nodes/**
GET,POST,PUT,DELETE,HEAD:/solrAdmin/**
GET,POST,PUT:/synonyms/**
GET,POST,PUT,DELETE,HEAD:/jobs/**
GET,POST,PUT,DELETE,HEAD,OPTIONS:/collections/**
GET,POST,PUT,DELETE,HEAD:/connectors/**
GET,POST,PUT,DELETE,HEAD:/groups/**
GET,POST,PUT,DELETE,HEAD:/query-profiles/**
GET,POST,PUT:/templates/**
GET,POST,PUT,DELETE,HEAD:/tasks/**
GET,POST,PUT,DELETE,HEAD:/links/**
PATCH:/users/{id}:id=#ID
GET,POST,PUT:/registration/**
POST:/index/**
GET,POST,PUT:/objects/**

```

The permission <code>PATCH:/users/{id}:id=#ID</code> uses the variable value <code>#ID</code> 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.

search

The search role has read-only query and write-only signal API access to the Fusion "default" collection. These permissions are required for search applications, for example, for App Studio.

```
POST:/apps/*/signals/**
GET,POST:/query/**
POST:/signals/**
PATCH:/users/{id}:id=#ID
GET,POST:/apps/*/query/**
```

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.

webapps-role

The webapps role can list and download Fusion apps.

```
GET,HEAD:/webapps/**
GET,HEAD:/license
```

Role Information

Fusion stores role information in Apache ZooKeeper (<http://zookeeper.apache.org/>). 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-permissions`– A list of names of Fusion UI components
- `created-at`– Timestamp; created by Fusion
- `updated-at`– Timestamp for last edit; created by Fusion

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 is 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 the Fusion UI. Click **System > Access Control > Roles**.

To create a new role from the Fusion UI, first 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 that 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*.

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.

Fusion does not use permissions from the LDAP to authorize UI access or API requests. It only obtains group names (optionally), which are used as role names or 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.

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.

The directory service that you use to return group information must be the same one that is used for authentication.

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. 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*.

Passwords are not stored, but are hashed using bcrypt (<https://en.wikipedia.org/wiki/Bcrypt>).

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 *Configure Fusion for SSO*.

JWT

The JSON Web Token (JWT) realm uses an "Authorization" header in the request to authenticate the user and the data inside the JWT token for the authorization. You can configure Fusion to use a shared secret key to encrypt the JWT payload.

For Fusion's JWT realm, first create a JWT token, with the tool you use to validate users. Then create a Fusion realm.

See *Configure Fusion for JWT*.

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 (https://docs.oracle.com/cd/E21455_01/common/tutorials/kerberos_principal.html).

SPNEGO (<https://en.wikipedia.org/wiki/SPNEGO>) is used for authentication via Kerberos.

See how to configure Fusion for Kerberos in *Unix* and *Windows*.

LDAP

You can use 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 (https://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol#Directory_structure) (DN). Fusion permissions can be assigned automatically based on LDAP group membership.

See *Configure 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 *Configure Fusion for SAML*.

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 **System > Access Control > Security Realms**.

Using the Realms API

Use the `http://localhost:8764/api/realms-configs/` endpoint to manage security realms. See the *Realms API* reference for details. In production environments, use port 8765.

Users

All Fusion requests must come from a registered user.

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 does not 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 will not 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.

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 Fusion UI. Click **System > Access Control > Users**.

Manage Users via HTTP Requests to the Users API

See page *Users API*.

User Information

Fusion stores user information in Apache ZooKeeper (<http://zookeeper.apache.org/>).

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 **System > Access Control**)
- `role-names` – List of roles assigned to the user in the Fusion UI (in **System > 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$3I82umlXLPShQIW6nqj.0r06D0VqDLGohGmCB9GC0yRtvy5Nfkn6", "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", }
```

Ingest and Indexing

Data Ingestion

Data ingestion gets your data into Fusion Server, and data indexing stores it in a format that is optimized for searching. These topics explain how to get your data into Fusion Server in a search-optimized format.

Collections

Collections are a way of grouping data sets so that related data sets can be managed together. Every data set that you ingest belongs to a collection. Any app can contain one or more collections. See *Collection Management*.

Datasources

Datasources are configurations that determine how your data is handled during ingest by Fusion Server's connectors, parsers, and index pipelines. When you run a fully-configured datasource, the result is an indexed data set that is optimized for search, depending on the shape of your data and how you want to search it. See *Datasource Configuration*.

Connectors

Connectors are Fusion components that ingest and parse specific kinds of data. There is a Fusion *connector* for just about any data type.

Blob storage

Blob storage is a way to upload binary data to Fusion Server. This can be your own data, such as images or executables, or it can be plugins for Fusion Server, such as connectors, JDBC drivers, and so on.

Other methods

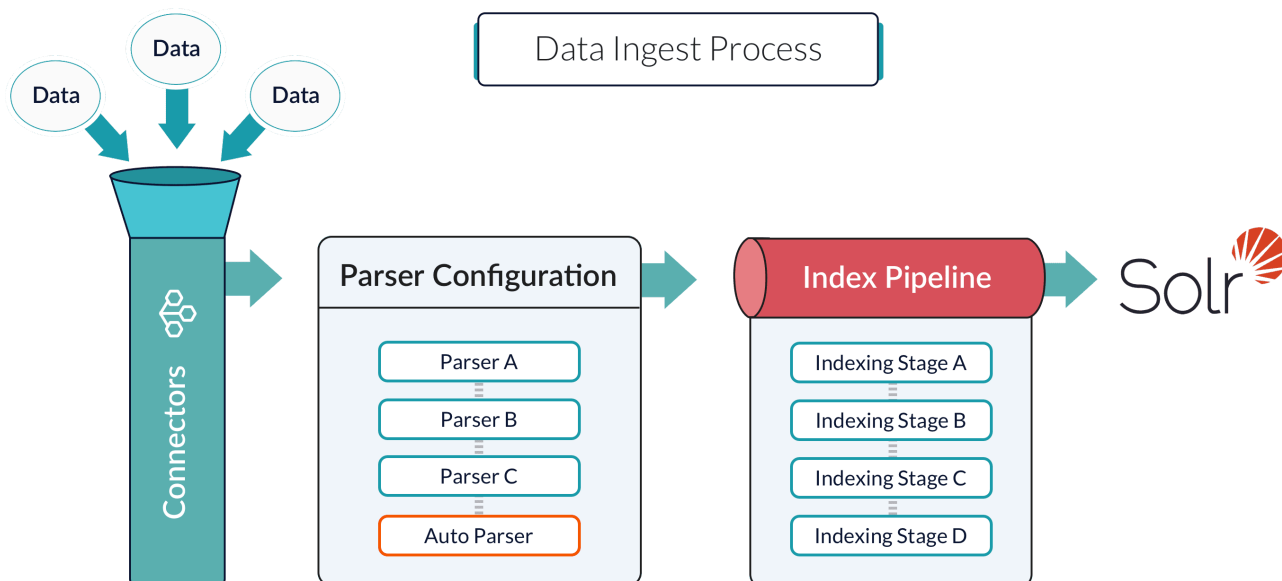
In some cases, you might find that it is best to use other ingestion methods, such as the *Parallel Bulk Loader*, *Hive*, *Pig*, or pushing data to a *REST API* endpoint.

Batch ingestion of signals is also available with a *Fusion AI license*.

Indexing

Indexing converts your data into an indexed collection in Fusion's Solr core. It is critical for ensuring that your data is stored in a format that is 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 Workbench* is Fusion's index pipeline development tool.



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.

Blob Types

A `resourceType` query parameter can be used to specify the a blob type. For example, specify `plugin:connector` when uploading a connector, like this:

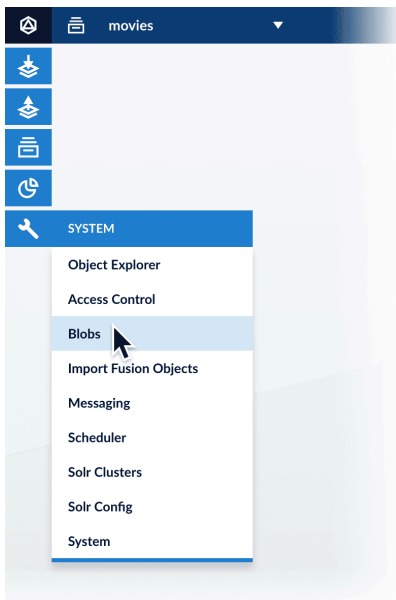
```
curl -H 'content-type:application/zip' -X PUT 'fusion-host:{api-port}/api/blobs/myplugin?resourceType=plugin:connector' --data-binary @myplugin.zip
```

The complete list of valid values for `resourceType` is below:

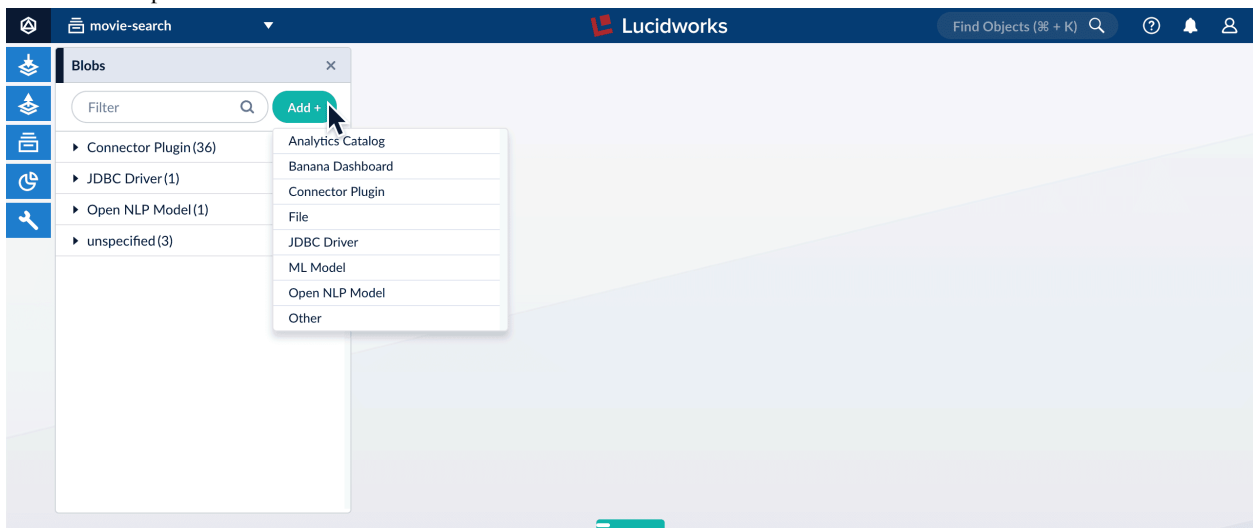
Type	Description
<code>banana</code>	A Banana <i>dashboard</i>
<code>catalog</code>	An <i>analytics catalog</i>
<code>driver:jdbc</code>	A <i>JDBC driver</i>
<code>file:js-index</code>	A JavaScript file for use with a <i>Managed Javascript index stage</i> .
<code>file:js-query</code>	A JavaScript file for use with a <i>Managed Javascript query stage</i> .
<code>file</code>	Any uploaded file, such as from the <i>Quickstart</i> or the <i>Index Workbench</i> .
<code>model:ml-model</code>	A <i>machine learning model</i> (Fusion AI only)
<code>model:open-nlp</code>	An <i>OpenNLP model</i> (Fusion AI only)
<code>other</code>	A blob of unknown type If no <code>resourceType</code> is specified on upload, "other" is assigned by default.
<code>plugin:connector</code>	A <i>connector plugin</i>

The Blob manager

In addition to the *Blob Store API*, the Fusion UI provides an interface to the blob store in the *Fusion workspace* at **System > Blobs**:

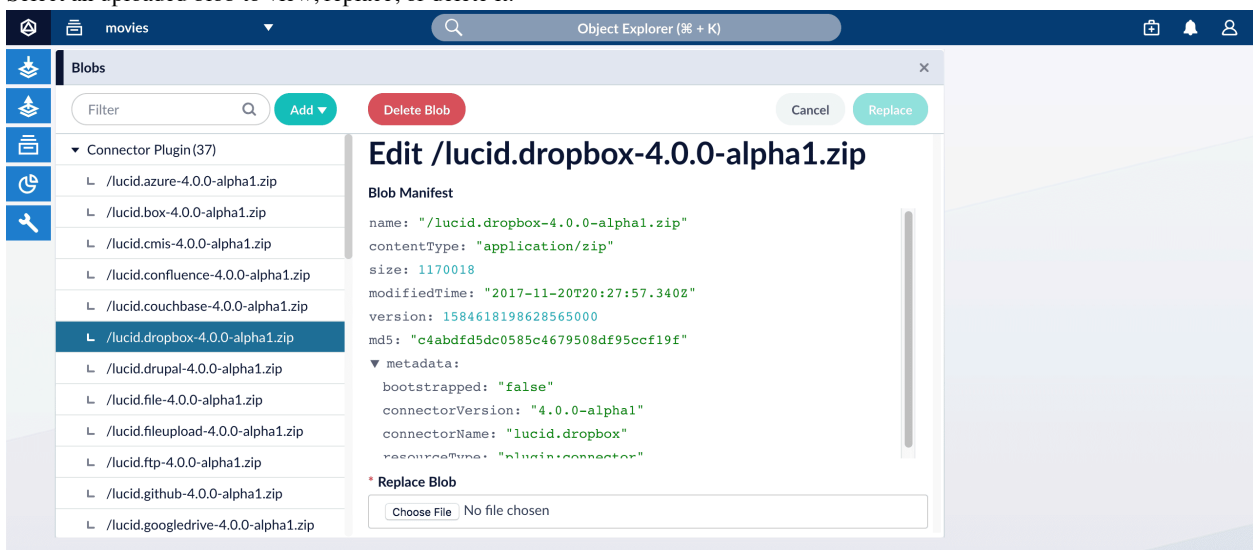


- Click **Add** to upload a new blob:



In Fusion 4.1, it is not possible to upload a blob of type `file:js-index` or `file:js-query` using the Fusion UI. Instead, use the *Blob Store API*. These blob types become visible in the blob manager after upload.

- Select an uploaded blob to view, replace, or delete it:



Collections

Your data is organized into collections. When you create an app, Fusion automatically creates a collection with the same name. You can create additional collections in any app.

A primary collection contains the data that your users will search. Every primary collection is associated with a set of auxiliary collections that contain related data, such as signals, aggregations, and more.

Under the hood, a Fusion collection is a distributed index in Solr, 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.

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

Collection names are case-insensitive, but Fusion preserves case when displaying collection names.

Auxiliary Collections

Every primary collection is associated with a set of auxiliary collections that contain related data, such as signals, aggregations, and more.

Some auxiliary collections are created for every primary collection. Others are created only for the app's default collection, one per app.

Auxiliary collections are described below:

<code>COLLECTION_NAME_signals</code>	A search query logs and signals collection.	1 per collection
<code>COLLECTION_NAME_signals_aggr</code>	A collection for aggregated signals.	1 per collection

Do not create primary collections with names that end in the suffixes above; these are reserved for Fusion auxiliary collections, which are created and managed by Fusion directly.

Fusion maintains a set of Solr collections that store Fusion's own log files and other internal information. These are called *System Collections*, described below.

Do not create primary collections named "logs" or beginning 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

System Collections

Fusion automatically creates some collections that are used for internal purposes and shared across all apps:

- **system_autocomplete** store the content that the Fusion UI displays when you use the search bar.
- **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_history** keeps a record of configuration changes, start and stop times for services and experiments, and more.
- **system_jobs_history** keeps a record of Fusion *jobs*, including start/stop times and status.

- **system_logs** stores parsed Java logs from the REST API, connectors-classic component, and other parts of Fusion, like proxy, connectors-rpc, and appkit app insights. It also includes http logs and optional gc logs (off by default in Fusion 4.1). Prior to Fusion version 4.1, Java logs were stored in the `logs` collection and HTTP requests were stored in the `audit_logs` collection.
- **system_messages** is used by Fusion's *messaging services*.
- **system_metrics** stores metrics about Fusion hosts and services, when enabled. See *System Metrics*. The data is polled at regular intervals according to the internal configuration variable: `com.lucidworks.apollo.metrics.poll.seconds`. This collection does not appear until after the first set of metrics are collected.

Collection Configuration Properties

Collections have three properties that you can configure only when you are creating a collection using the *Collections API*.

Property	Description	Default behavior
signals *	The <code>signals</code> property determines whether to create auxiliary collections with suffixes <code>_signals</code> and <code>_signals_aggr</code> .	When you create a collection in the Fusion UI, <code>signals</code> defaults to true . When you create a collection using the Fusion API, this property defaults to false .
search Logs	The <code>searchLogs</code> property determines whether to create an auxiliary search query logs collection with suffix <code>_logs</code> .	When you create a collection in the Fusion UI, this property defaults to true . When you create a collection using the Fusion API, this property defaults to false .

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

**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 can be convenient during preliminary application development, but it is rarely appropriate in a production environment.

Using profiles to associate collections with pipelines

Index pipelines and query pipelines are not connected to a specific collection by default. Index profiles and query profiles are configurations that create consistent endpoints for indexing and querying, each with a specific pipeline and collection.

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

Field Editor UI

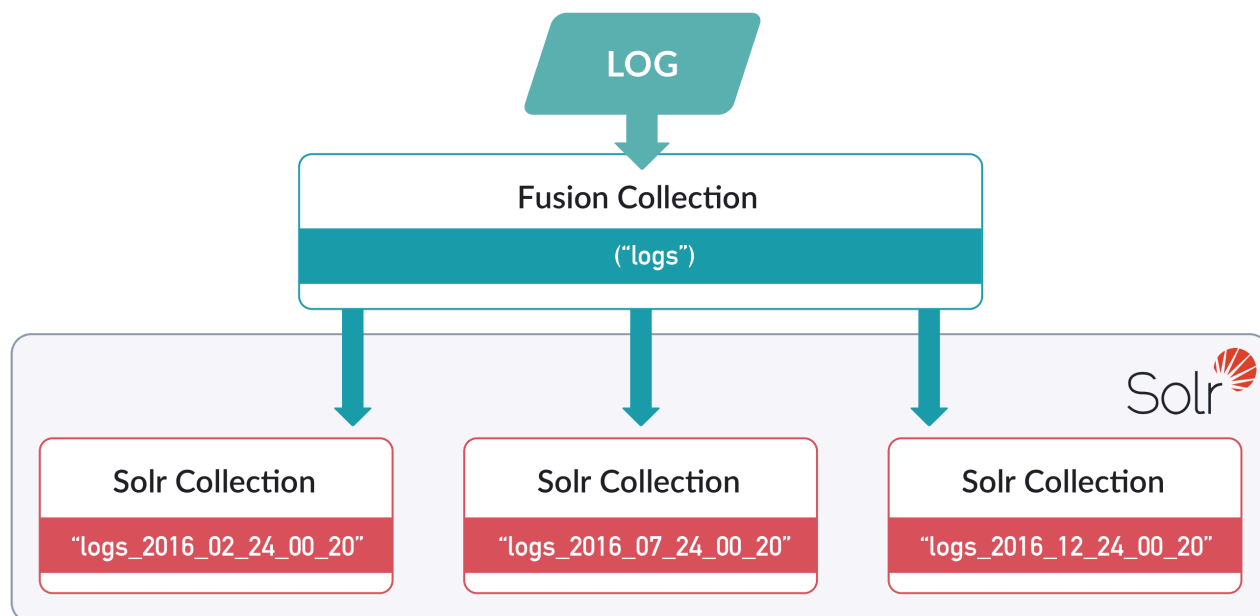
The Fusion UI includes a space under Collections to edit Fields. Descriptions for these fields can be found in the Field Type Definitions section of the *Solr Reference Guide* associated with your Fusion release.

Field options displayed in the UI include:

- **Dynamic** checkbox (cannot change via UI)
- **Field Name** (cannot change via UI)
- **Field Type** (a preset value is shown that can be changed using edit mode)
- Checkboxes for **Indexed**, **Stored**, **Multivalued**, **Required**
- Text field to enter a **Default Value**
- **Copy Fields** uses the plus sign to add rows (**static** can copy to `raw_content` or `text`; **dynamic** can copy to any `raw_content/text` or any other dynamic field)
- **Advanced** toggles checkboxes for **Doc Values**, **Omit Norms**, **Omit Positions**, **Omit term freq and positions**, **Term Vectors**, **Term Positions**, **Term Offsets**

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*.

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

Connectors

Connectors are the out-of-the-box components for pulling your data into Fusion. Lucidworks provides 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.

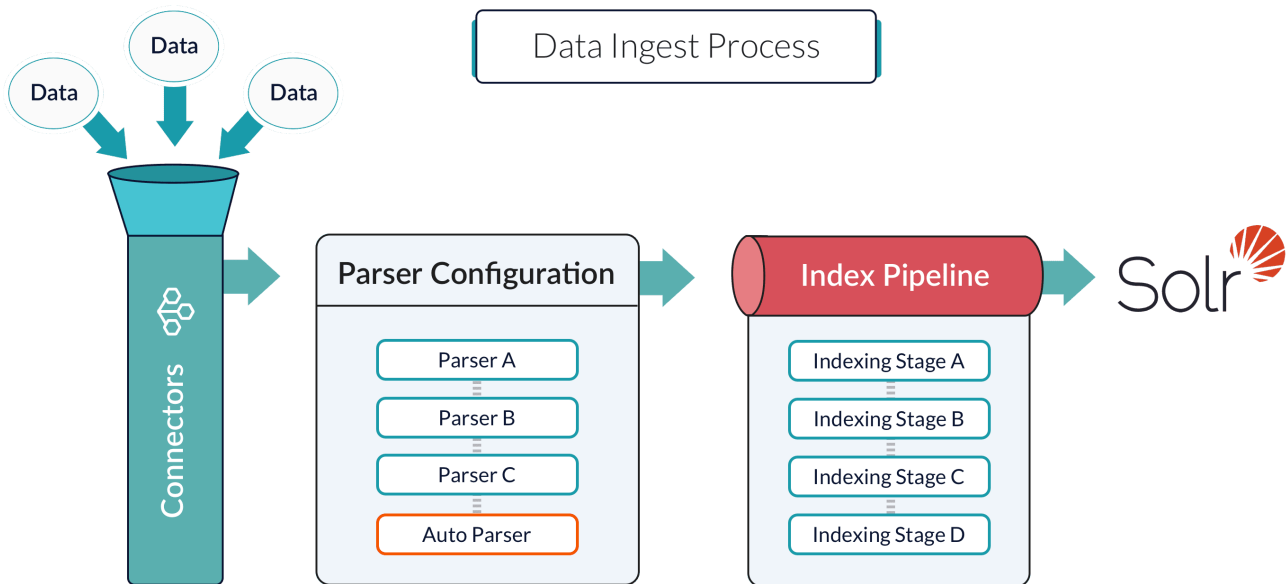
Connectors are distributed separately from Fusion Server. For complete information, see *Fusion Connectors*.

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:

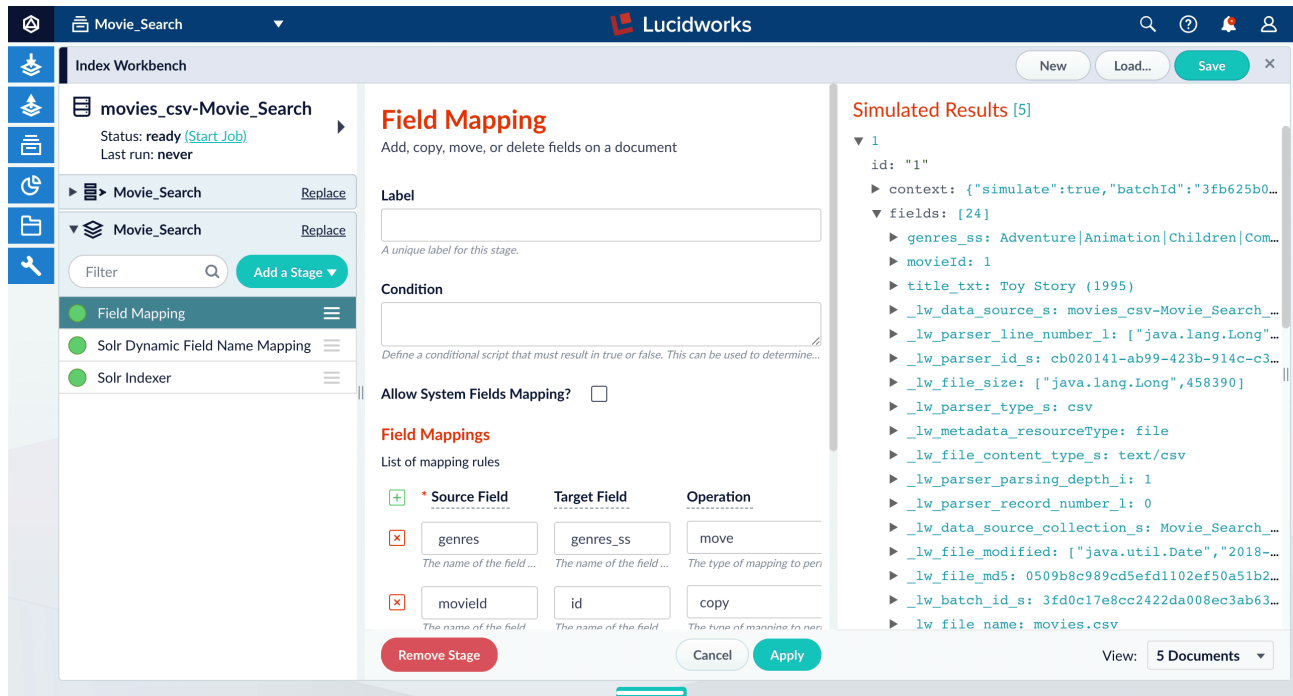
- Connector configuration, specifying the source and format of the incoming data.
- Parser configuration, describing a series of conditional parsing stages to transform the incoming data into PipelineDocument objects.
- Index pipeline configuration, consisting of stages that transform PipelineDocument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



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

Index Workbench

The Index Workbench is a powerful tool that combines key aspects of the data indexing configuration process into one user-friendly panel. It guides the user through the workflow for configuring datasources, parsers and index pipelines, and then running a datasource job to index your data.



Datasources

Add A New Datasource

Point to a URL:

☐

Or, upload a file:

☐ No file chosen

Or, if you know the location of your data:

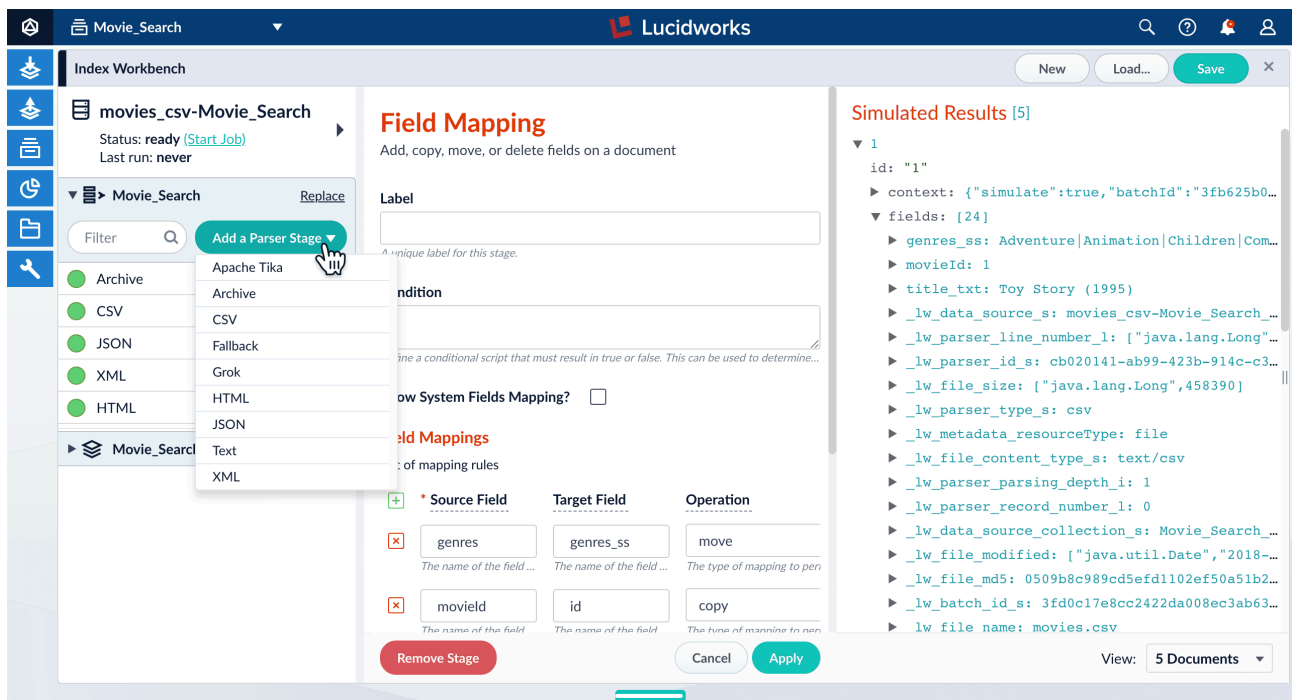
☐

To set up a new datasource in the Index Workbench, click **New**. The **Add A New Datasource** panel appears, offering three simplified methods of configuring how Fusion fetches your data:

- By URL
 - By local file upload
 - By selecting an installed connector to configure
- See *Connectors Configuration Reference* for a complete list of available connectors. Some are installed by default, and others require separate download and installation.

Parsers

Parsers are a configurable component of the indexing workflow, for flexibility and specificity when parsing inbound data.

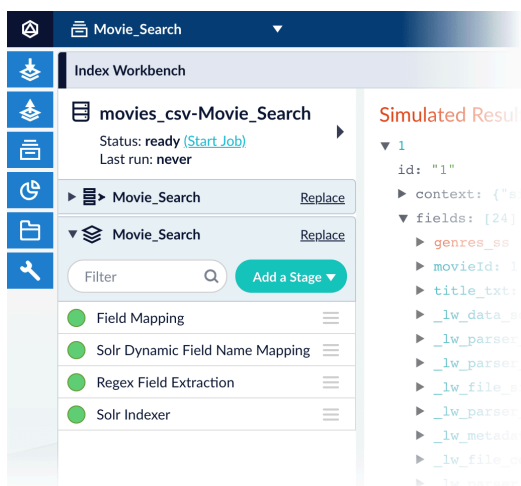


A parser consists of an ordered list of parser stages that is completely customizable. The same parser stage can be added to a given configuration multiple times if the different specified settings within those stages best suits the parsing of the data. There is no limit to the number of parser stages that can be included in a parser, and the order in which they run is also completely flexible.

In a parser, after all of the doctype-specific parser stages have run, the Tika and Fallback stages are useful catch-all stages that can attempt to parse anything that has not yet been matched. Tika is used for parsing many types of unstructured documents like PDFs, DOCX, and many more. If all of the other stages in the parser fail to completely parse the data, the Fallback stage can copy the raw bytes directly to Solr.

Index pipelines

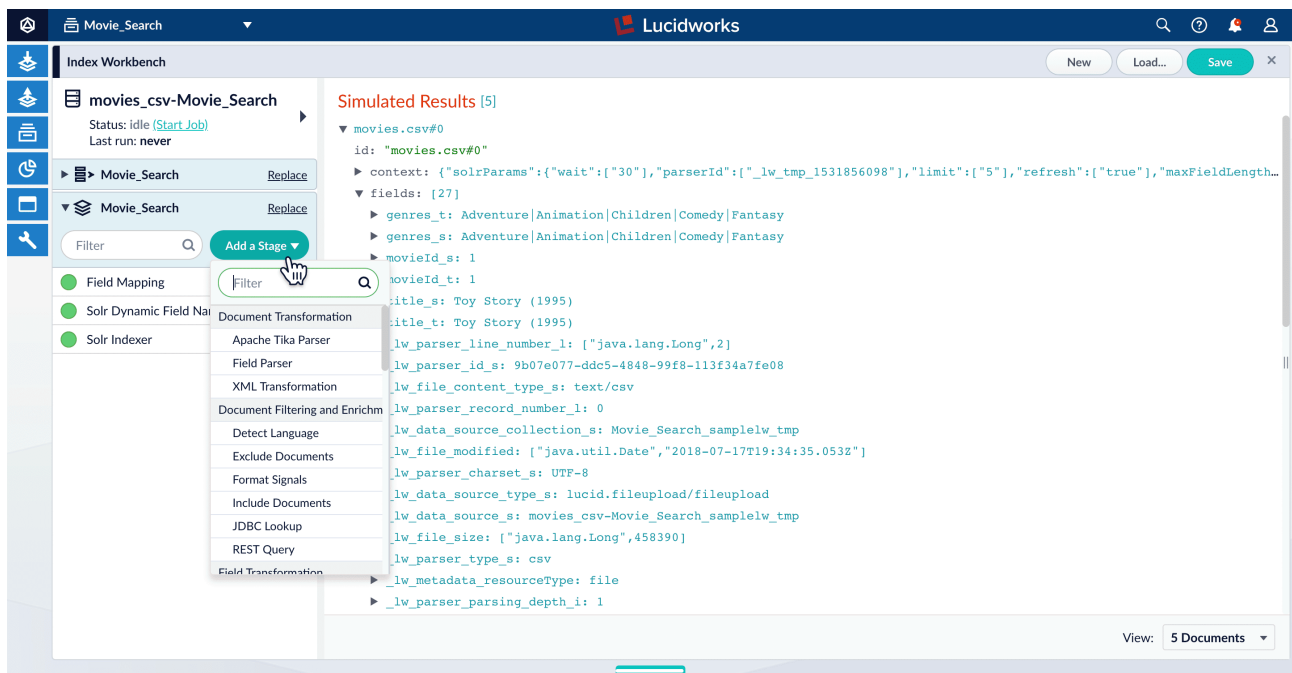
An *Index Pipeline* transforms incoming data into a document suitable for indexing by Solr via a series of modularized operations called stages. Fusion provides a variety of specialized index stages to index data effectively. Stages can be selected, configured, and enabled or disabled in the Index Pipeline section of the Index Workbench.



Once you finish configuring a datasource using the Index Workbench, you can move on to setting up queries using the *Query Workbench*, which provides a similar workflow for configuring and previewing search results.

Adding a pipeline stage

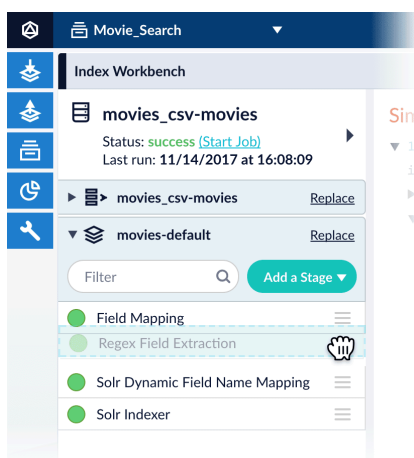
Click **Add a Stage** to add index pipeline stages that can perform data filtering, transformations, and more.



The *Solr Indexer* stage is the only pipeline stage that is required to complete the indexing process, and therefore exists in every index pipeline. It is always the last stage in a series.

Re-ordering pipeline stages

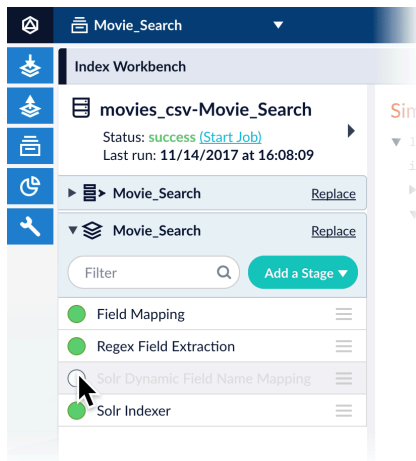
The order of the pipeline stages matters, because the output from one stage becomes the input to the next stage. For example, the *Solr Indexer* stage must always come last in the sequence, so that data is indexed only after it has been processed by all other stages. Putting this stage first in the sequence means that subsequent stages have no effect on the indexed data.



Drag any stage in the pipeline to move it up or down in the sequence of stages. The preview panel automatically updates the results to reflect the output of the new sequence.

Enabling and disabling pipeline stages

By default, every stage in an index pipeline is enabled. While working with a pipeline, it can be helpful to disable stages without removing them completely. This allows you to preserve a stage's configuration while observing how the search results change in its absence. You can re-enable the stage at any time. When you save a query pipeline, the enabled/disabled state of each stage is also saved.

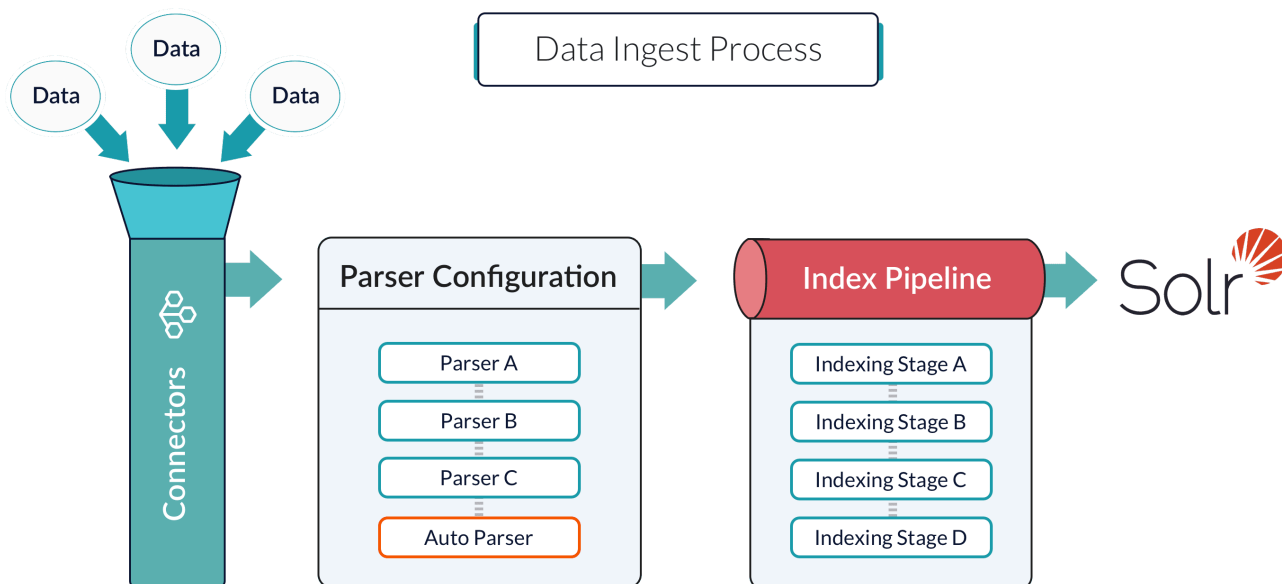


Click the circle next to any stage in order to enable or disable it.

Index Pipelines

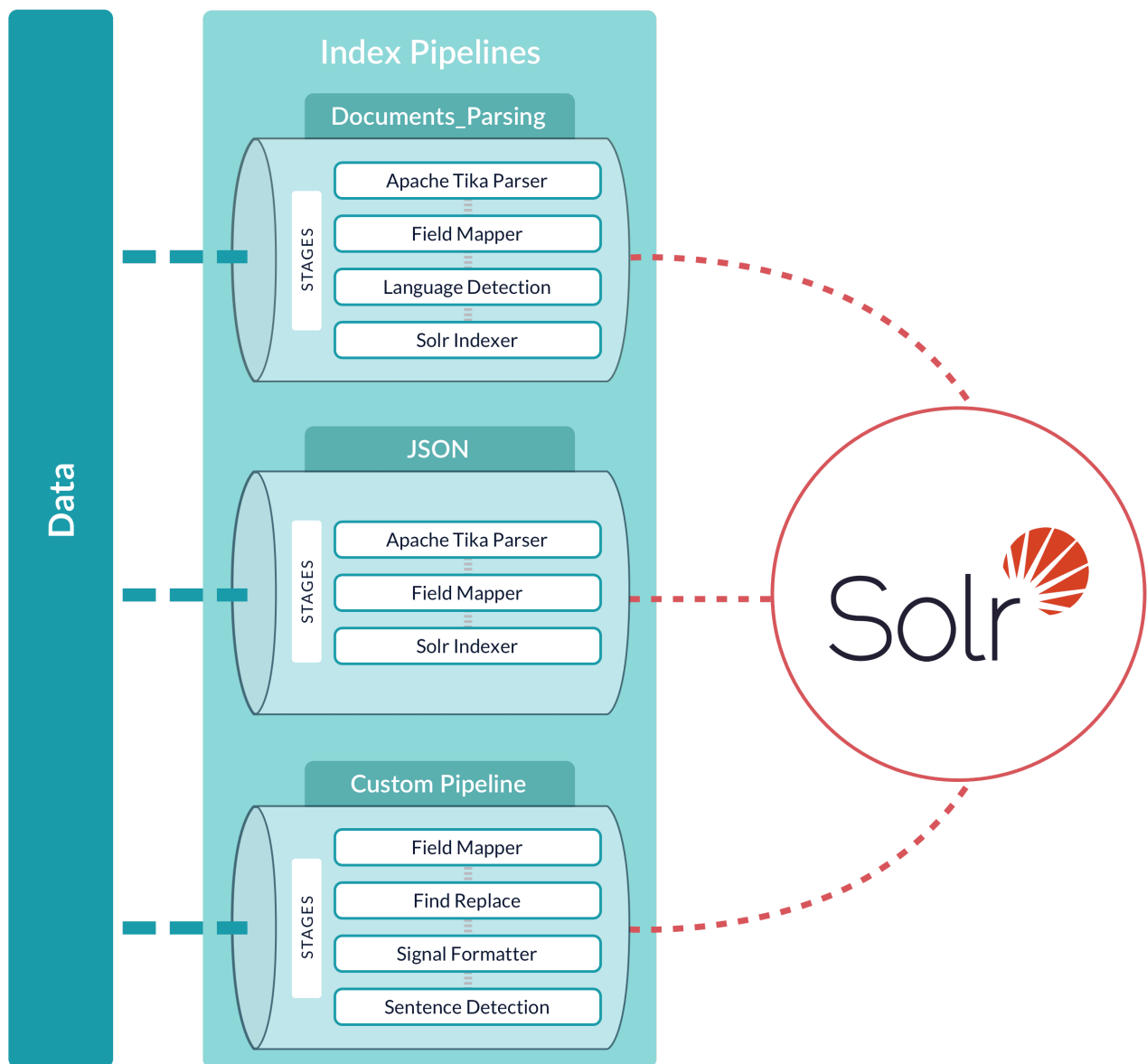
Index pipelines transform incoming data into `PipelineDocument` objects for indexing by Fusion-managed Solr service. An index pipeline consists of a series of configurable *index pipeline stages*, each performing a different transformation on the data before passing the result to the next stage in the 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 or profile with the REST API; see *Importing Data with the REST API*.

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



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 Mapping index stage*.

Although default pipelines are created when a Fusion collection is created, they are not deleted when the collection is deleted. This is because pipelines can be used across collections, so a named pipeline, although originally associated with a collection, can be used by several collections.

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 the Extract/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 that 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.

Additional Resources

Available index pipeline stages are listed below:

Document transformation

- *XML Transformation*

Document filtering and enrichment

- *Detect Language*
- *Exclude Documents*
- *Format Signals*
- *Include Documents*
- *JDBC Lookup*
- *REST Query*

Field transformation

- *Date Parsing*
- *Field Mapping*
- *Filter Short Fields*
- *Find and Replace*
- *GeoIP Lookup*
- *Regex Field Extraction*
- *Regex Field Filter*
- *Regex Field Replacement*
- *Resolve Multivalued Fields*
- *Solr Dynamic Field Name Mapping*

Natural language processing

- *Detect Sentences*
- *Gazetteer Lookup Extraction*
- *OpenNLP NER Extraction*
- *Tag Part-of-Speech*

Indexing

- *Solr Indexer*
- *Solr Partial Update Indexer*
- *Update Related Document*

Troubleshooting

- *Logging*

- *Send PagerDuty Message*
- *Send SMTP Email*
- *Send Slack Message*
- *Write Log Message*

Advanced

- *Call Pipeline*
- *Exclusion Filter*
- *Javascript*
- *Machine Learning*
- *Managed Javascript*
- *Set Property*

Index Profiles

Index profiles let your applications *send documents for indexing to a consistent endpoint* (the index profile endpoint) and change the underlying index pipeline as needed.

An index profile is also a simple way to use one pipeline for multiple collections, without any one collection "owning" the pipeline. Associating a profile with an index pipeline, or an index pipeline and a collection, is simply a mapping.

Finally, you can also specify configuration parameters for an index pipeline in the index profile.

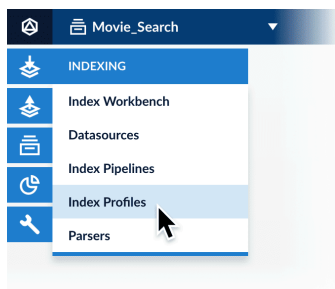
You can use the API or UI to manage index profiles.

Index Profiles in the API

- *Index Profiles API* (`/index-profiles`)
Create, read, update, and delete index profiles.
- *Index API* (`/index`)
Index data through an index profile by specifying the profile ID.

Index Profiles in the UI

Index profiles are configured at **Indexing > Index Profiles**.



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. For example, Solr commits have a `null` value and are dropped. For information about how to skip stages when Solr commits are sent, see the *Skip JavaScript stages during Solr commits*.

JavaScript Index Stage Global Variables

JavaScript (<http://en.wikipedia.org/wiki/JavaScript>) is a lightweight scripting language. The JavaScript in a JavaScript stage is standard ECMAScript (<http://en.wikipedia.org/wiki/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
<code>doc</code>	<i>Pipeline Document</i>	The contents of each document submitted to the pipeline.
<code>ctx</code>	<i>Context</i>	A reference to the container that holds a map over the pipeline properties. Used to update or modify this information for downstream pipeline stages.
<code>collection</code>	String	The name of the Fusion collection being indexed or queried.
<code>solrServer</code>	<i>Buffering SolrServer</i>	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 <code>SolrClient</code> (https://lucene.apache.org/solr/5_2_1/solr-solrj/org/apache/solr/client/solrj/SolrClient.html) for details.
<code>solrServerFactory</code>	<i>SolrClientFactory</i>	The <code>SolrCluster</code> server used for lookups by collection name which returns a Solr server instance for a that collection, e.g. <pre>var productsSolr = solrServerFactory.getSolrServer("products");</pre>

The now-deprecated global variable `_context` refers to the same object as `ctx`.

Syntax Variants

JavaScript stages can be written using **legacy syntax** or **function syntax**. The key difference between these syntax variants is how the "global variables" are used. While using legacy syntax, these variables are used as global variables. With function syntax, however, these variables are passed as function parameters.

Legacy Syntax

```
//do some work...
return doc;
```

Function Syntax

```
function (doc) {  
  // do some work ...  
  return doc;  
}
```

`Function syntax` is used for the examples in this document.

JavaScript Use

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 that returns a document or an array of documents

All pipeline variables referenced in the body of the JavaScript function are passed in as arguments to the function. E.g., in order to access the PipelineDocument in global variable 'doc', the JavaScript function is written as follows:

```
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 log file of the server running the pipeline. This variable is truly global and does not need to be declared as part of the function parameter list.

Since Fusion's API service does the index pipeline processing, these log messages go into the log file:

`https://FUSION_HOST:FUSION_PORT/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".

JavaScript Index Stage Examples

Skip JavaScript stages during Solr commits

When JavaScript stages contain data manipulation such as values in the `conditions` field, errors may be generated and Solr commits are dropped because the value is `null`. To skip those stages when a Solr commit is sent through the pipeline, add the following condition to those stages: `doc.getCommands().size()==0`.

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 is 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 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 org.apache.solr.client.solrj.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;
}
```

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 log file).
- Use the *Index Workbench*.

The JavaScript Engine Used by Fusion

The JavaScript engine used by Fusion is the Nashorn engine from Oracle. See The Nashorn Java API (<https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/api.html>) 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_INSTALL_PATH`
 - `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 (https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/prog_guide/javascript.html#A1147207).

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;
```

The Parallel Bulk Loader

Fusion Parallel Bulk Loader jobs enable bulk ingestion of structured and semi-structured data from big data systems, NoSQL databases, and common file formats like Parquet and Avro.

The Parallel Bulk Loader leverages the popularity of Spark as a prominent distributed computing platform for big data. A number of companies invest heavily in building production-ready Spark SQL data source implementations for big data and NoSQL systems, much as Lucidworks has done for Solr. The Parallel Bulk Loader uses connectors provided by the experts who develop these complex systems.

This feature is available with a Fusion AI license. See the *Fusion AI Licensing* for details.

Dashboards

Dashboards are Fusion's built-in analytics component. Most of the components of dashboards, called *panels*, are for data visualization. Some panels display textual information; for example, records in a table or documents.

Additional analytics functionality is available in *App Insights*. A Fusion AI license is needed to enable App Insights.

Dashboards permit near-real-time views of data. For some types of panels, you can configure how frequently the panel gets new data.

Dashboards also permit users to interact with data. For example, users can submit queries, select facets, and zoom in and out on histograms.

A Fusion dashboard contains one or more controls for search query inputs and one or more quantitative displays over the results for that query.

Dashboards run as a client-side application in a web browser, using JavaScript components for HTML5. Client-side JavaScript provides a dynamic, responsive browsing experience. Solr facets provide the quantifications required for visualizations, for example, charts, graphs, tables, and maps (for geospatial data). Dashboards can also have tabular displays for drilling down to the individual documents in a results set.

The underlying browser application is Banana, an open-source visualization tool based on Kibana 3 (<http://www.elastic.co/guide/en/kibana/3.0/index.html>). Where Kibana communicates with Elasticsearch, Banana communicates with Solr. Banana can be downloaded from: <https://github.com/LucidWorks/banana/> (<https://github.com/LucidWorks/banana/>).

Fusion dashboards were first developed for log file analysis. Log file 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.

See these additional topics for details:

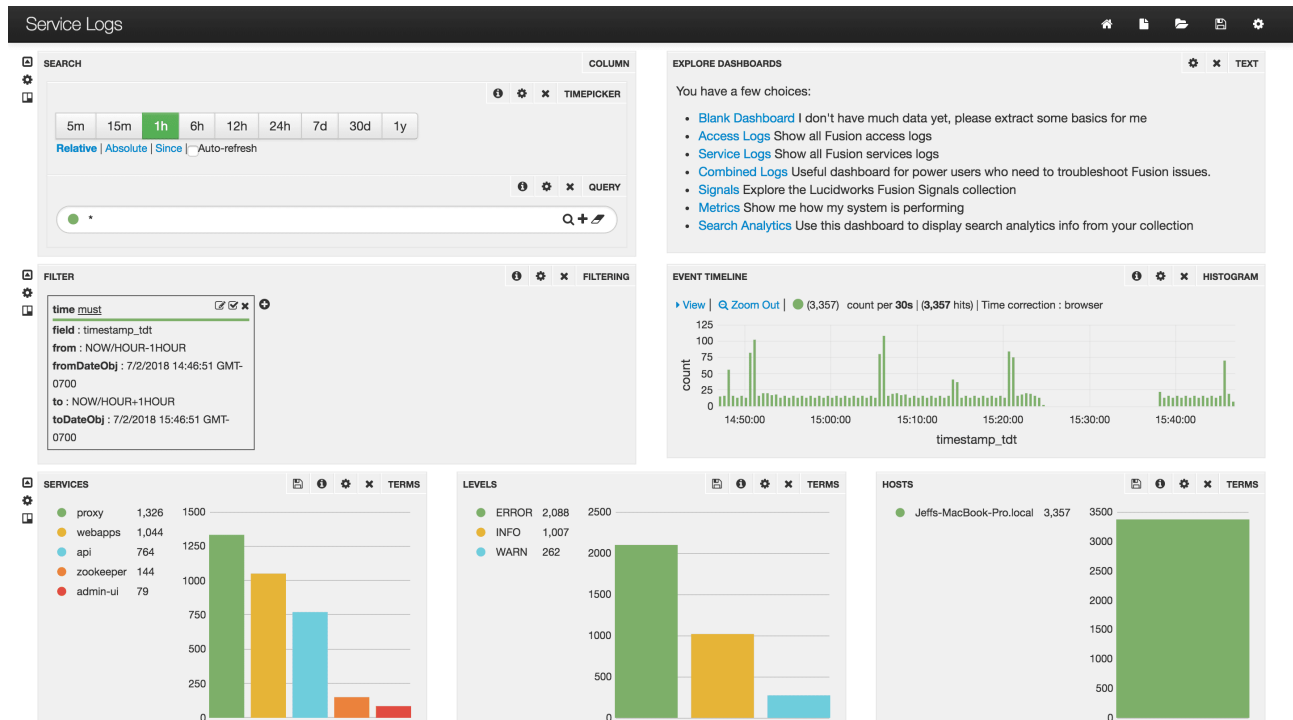
- *Create Dashboards*
- *Use Dashboards*
- *Manage Dashboards*
- *Input Panels*
- *Display Panels*

Built-in Dashboards

Fusion ships with built-in dashboards. To access these dashboards from the Fusion UI, click **System > Log Viewer**.

Service Logs dashboard

The Service Logs dashboard (`lucidworks-service-logs.json`) is also the default dashboard (`default.json`).



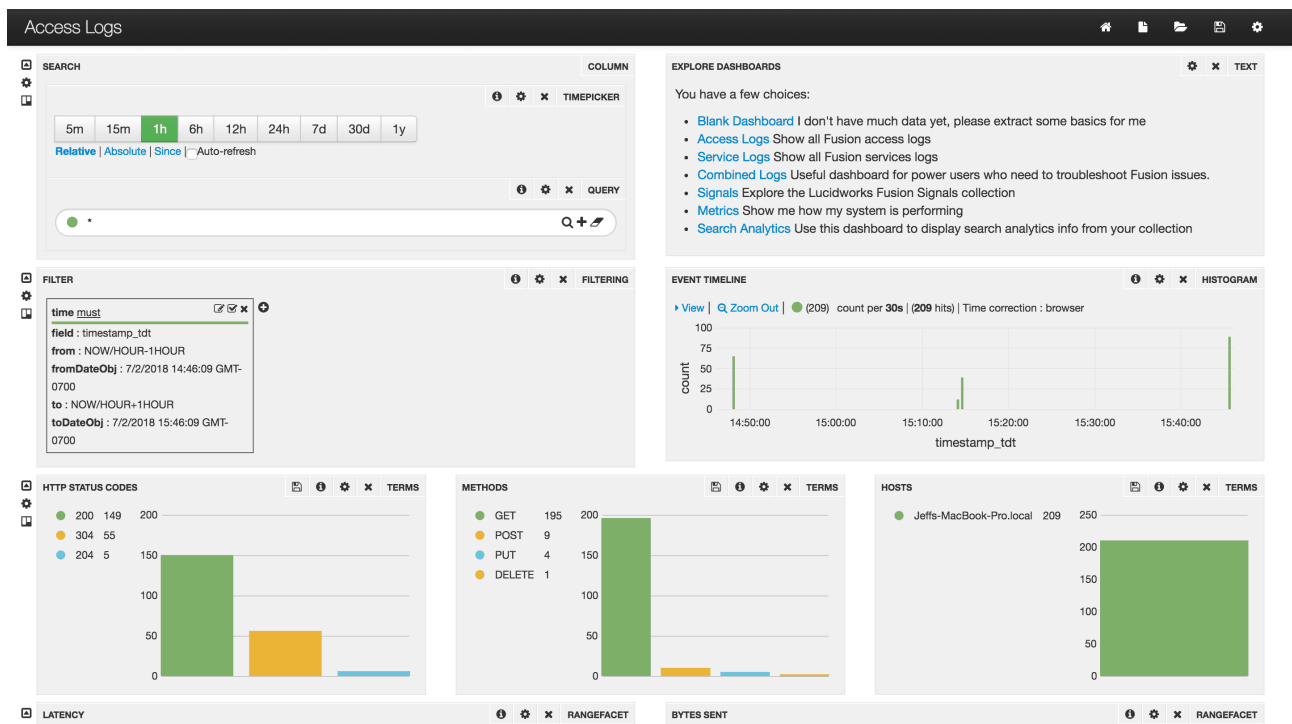
You can modify the default dashboard to suit your needs. To set a different dashboard as the browser default, click the dashboard you want to set as the default, then click **Save > Set as Browser Default**.

To open the default dashboard from the Fusion workspace, click Analytics  > **Dashboards** or System  > **Log Viewer**.

Alternatively, on any dashboard, click **Goto saved default** .

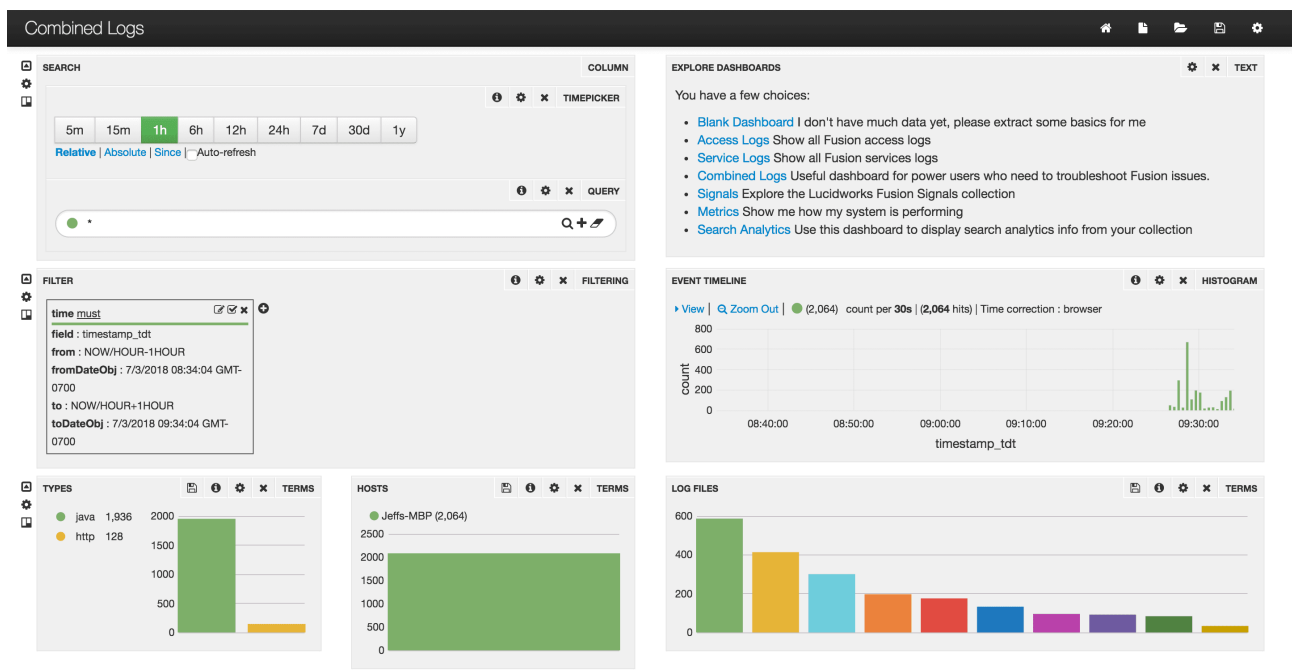
Access Logs dashboard

The Access logs dashboard (`lucidworks-access-logs.json`) lets you analyze logs from the `system_logs` collection where `type_s:http`, that is Fusion access over the Web, for example, from the Fusion UI and web apps that submit Fusion requests.



Combined Logs dashboard

The Combined Logs dashboard (`lucidworks-combined-logs.json`) lets you analyze all logs from the `system_logs` collection.

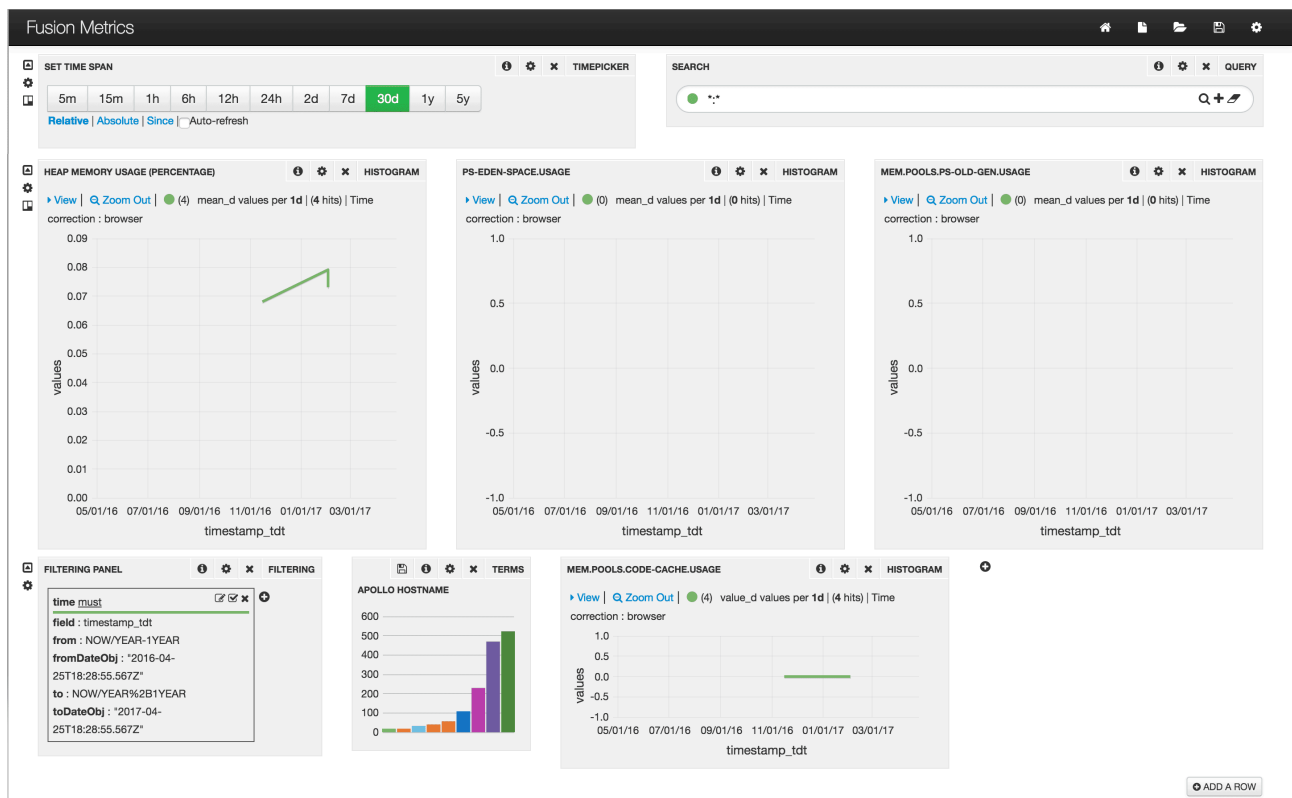


Blank dashboard

An entirely blank dashboard (`blank.json`) is available. To open this dashboard from the default dashboard, click **Blank Dashboard** at the top right.

Fusion Metrics dashboard

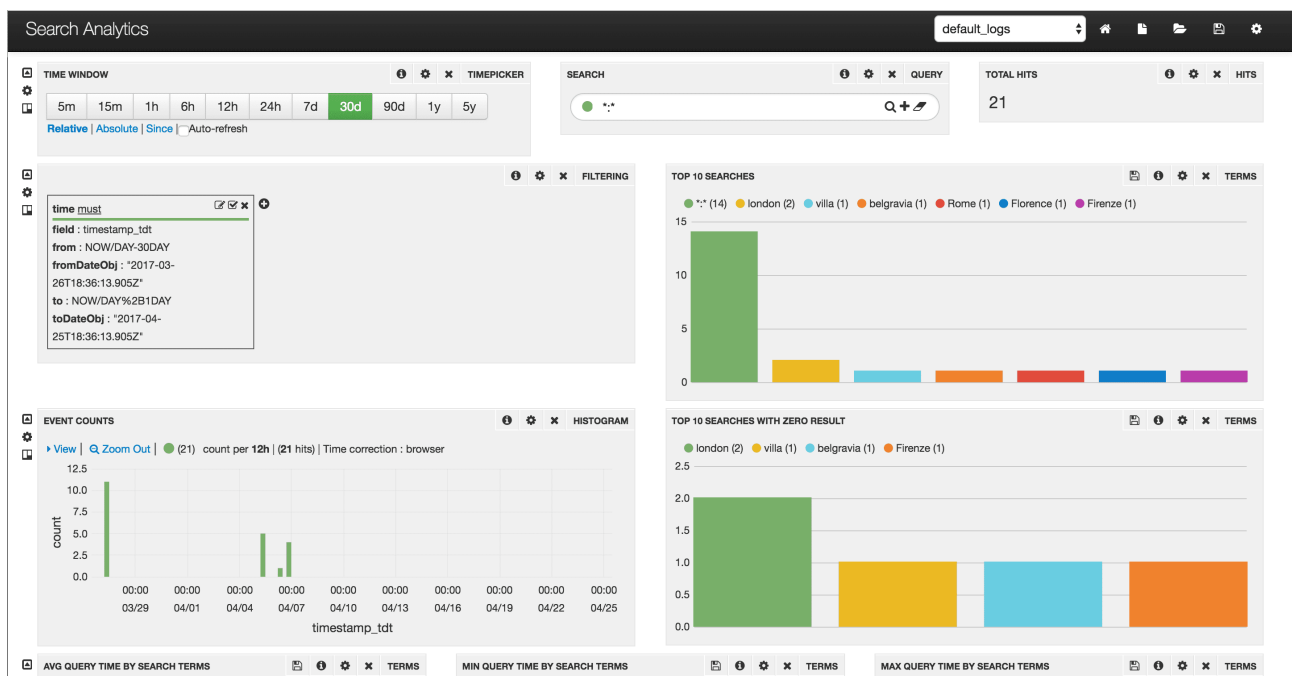
The Fusion Metrics dashboard (`lucidworks-metrics.json`) displays key Fusion metrics from the Fusion collection `system_metrics`, if this type of metrics collection is enabled. See *System Metrics*.



To open the Fusion Metrics dashboard from the default dashboard, click **Metrics Dashboard** at the top right.

Search Analytics dashboard

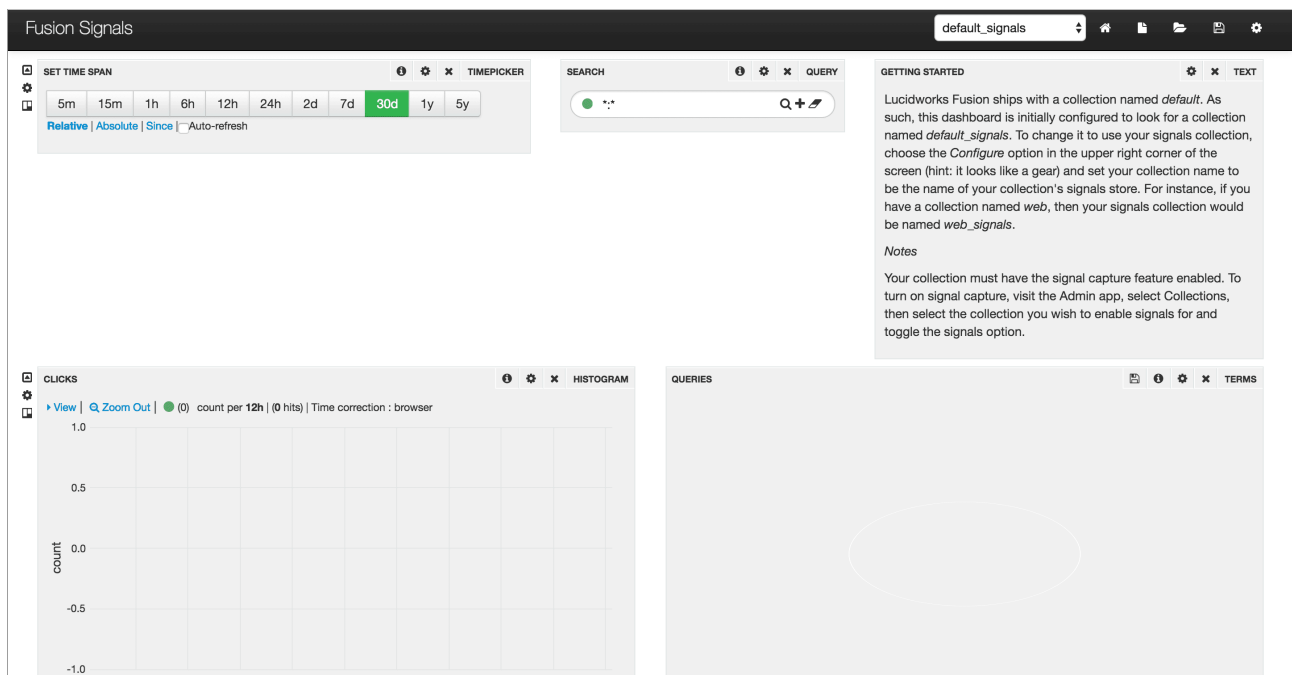
The Fusion Search Analytics dashboard (`lucidworks-searchanalytics.json`) displays search analytics derived from collection logs and from queries to Solr (`/api/solr`).



To open the Search Analytics dashboard from the default dashboard, click **Search Analytics** at the top right.

Fusion Signals dashboard

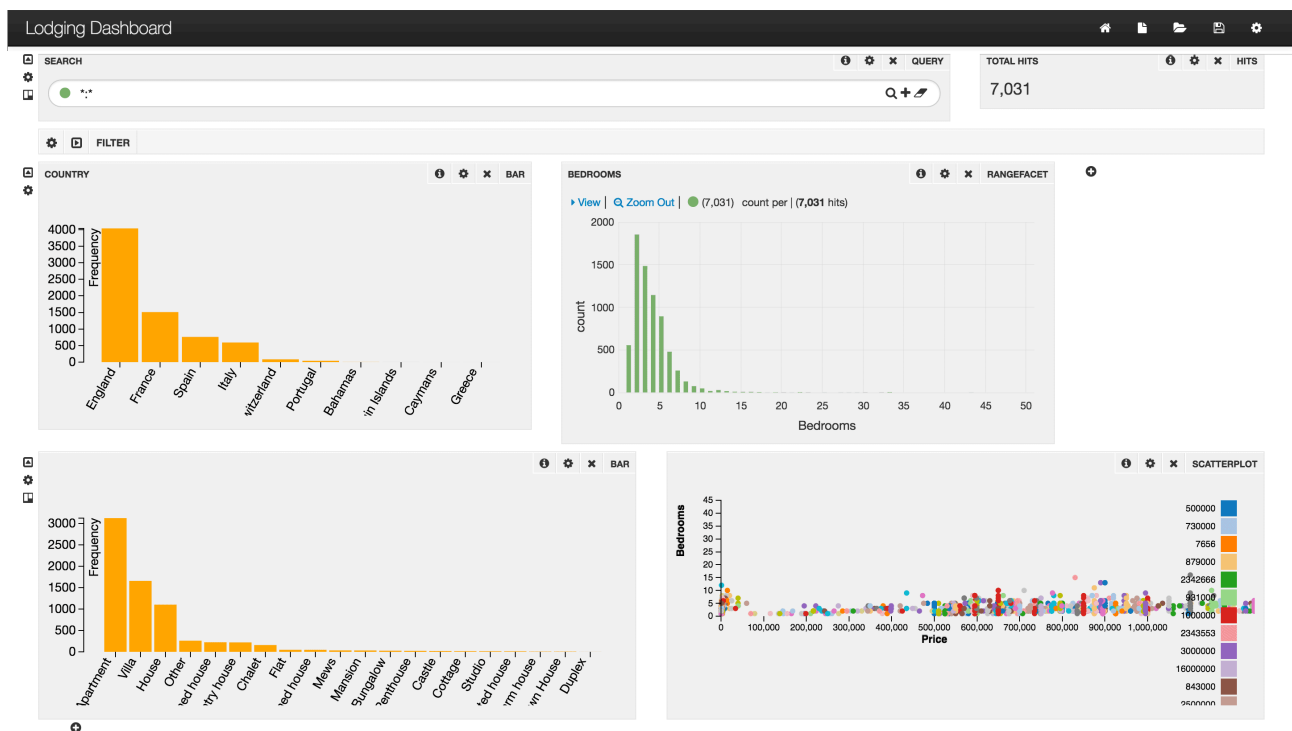
The Fusion Signals dashboard (`lucidworks-signals.json`) is a time-series dashboard that you can use to monitor signals collections. It uses the signal timestamp as the time field.



To open the Fusion Signals dashboard from the default dashboard, click **Lucidworks Fusion Signals** at the top right.

Default Non-Time-Series dashboard

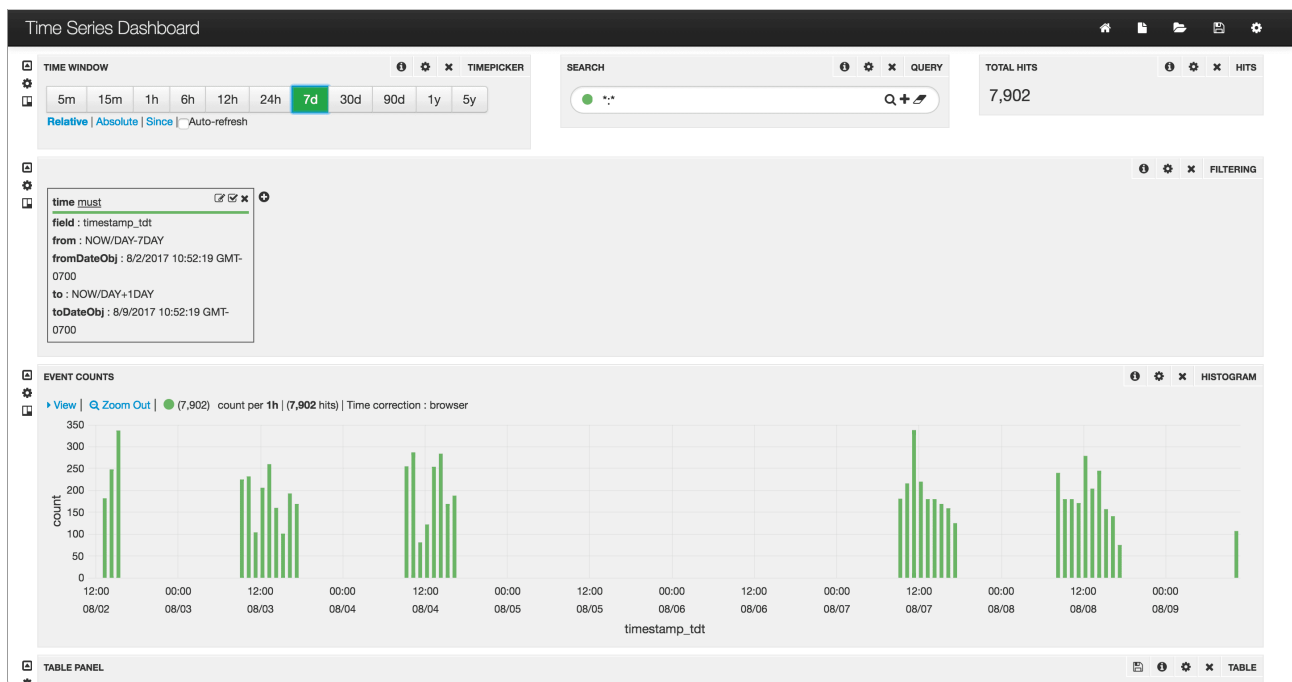
Fusion includes a default non-time-series dashboard (`default-nts.json`). This is an example:



To open the default non-time-series dashboard, at the top right of any dashboard, click **New > Non-time-series dashboard**.

Default Time-Series dashboard

Fusion includes a default time-series dashboard (`default-ts.json`). This is an example:



To open the default time-series dashboard, at the top right of any dashboard, click **New > Time-series dashboard**.

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.

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

Layout Panels

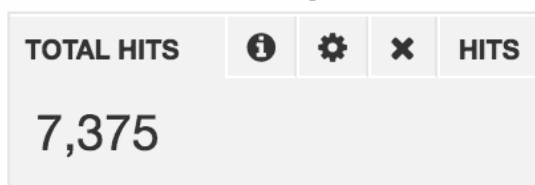
These layout panels can help you organize a dashboard:

- **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.

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 does not 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.

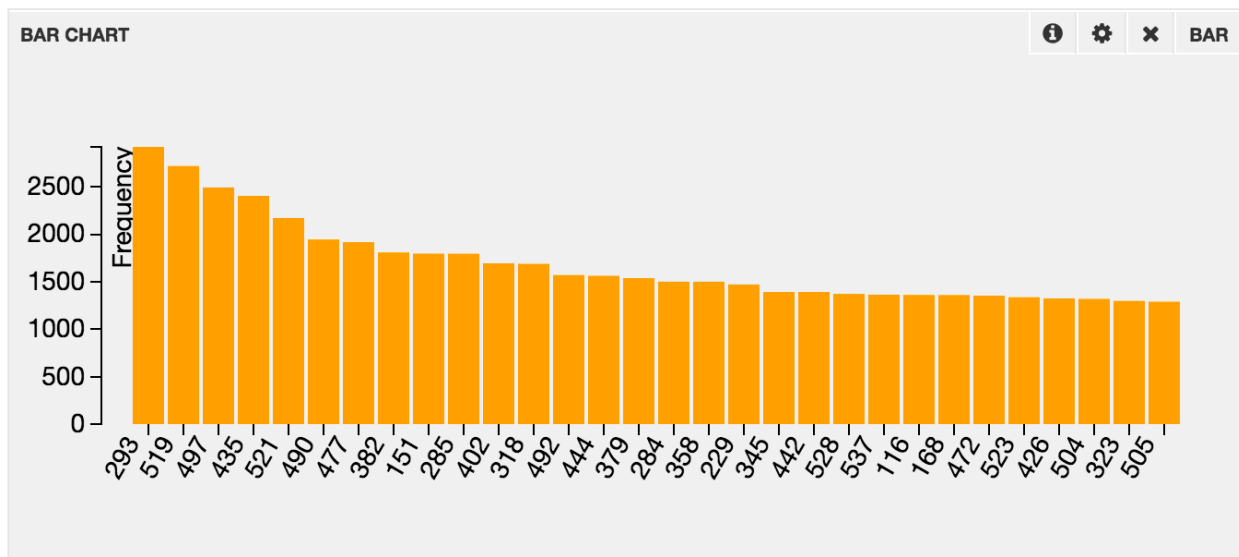


- **Table.** Display data in a table.
Note: Make a Table panel wide. Start with Span = 12. Depending on the content, you might be able to make it narrower. If you make a Table panel too narrow, content and controls can be inaccessible.

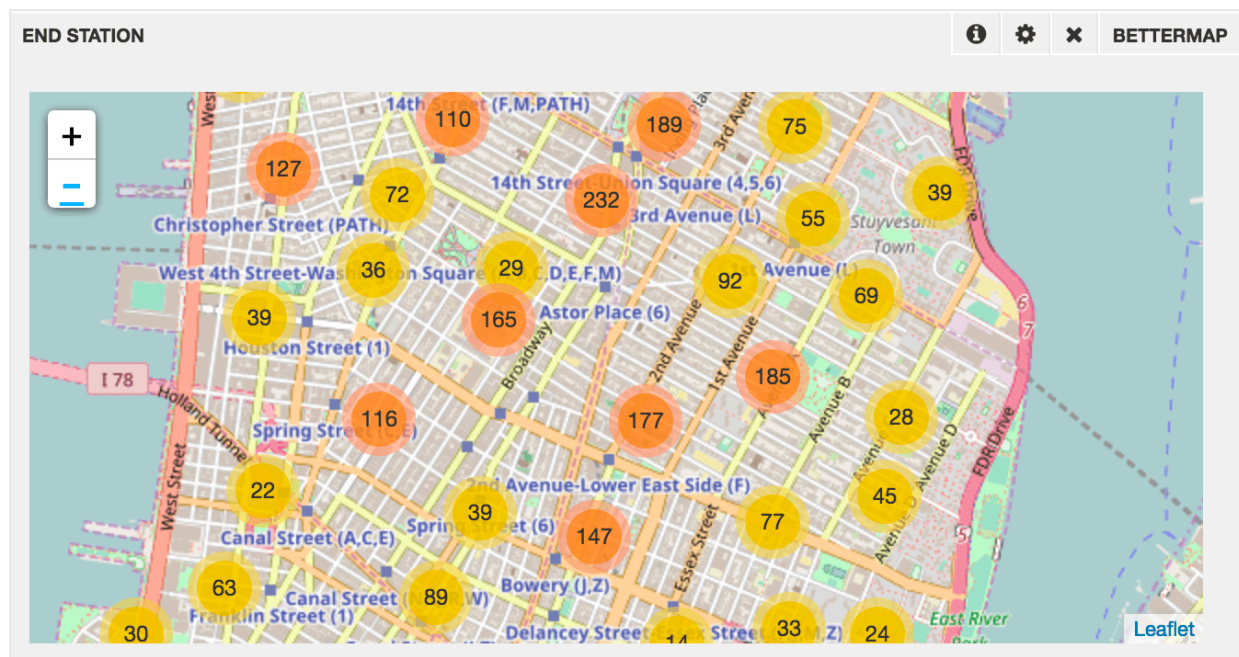
Graphical Visualization Panels

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:

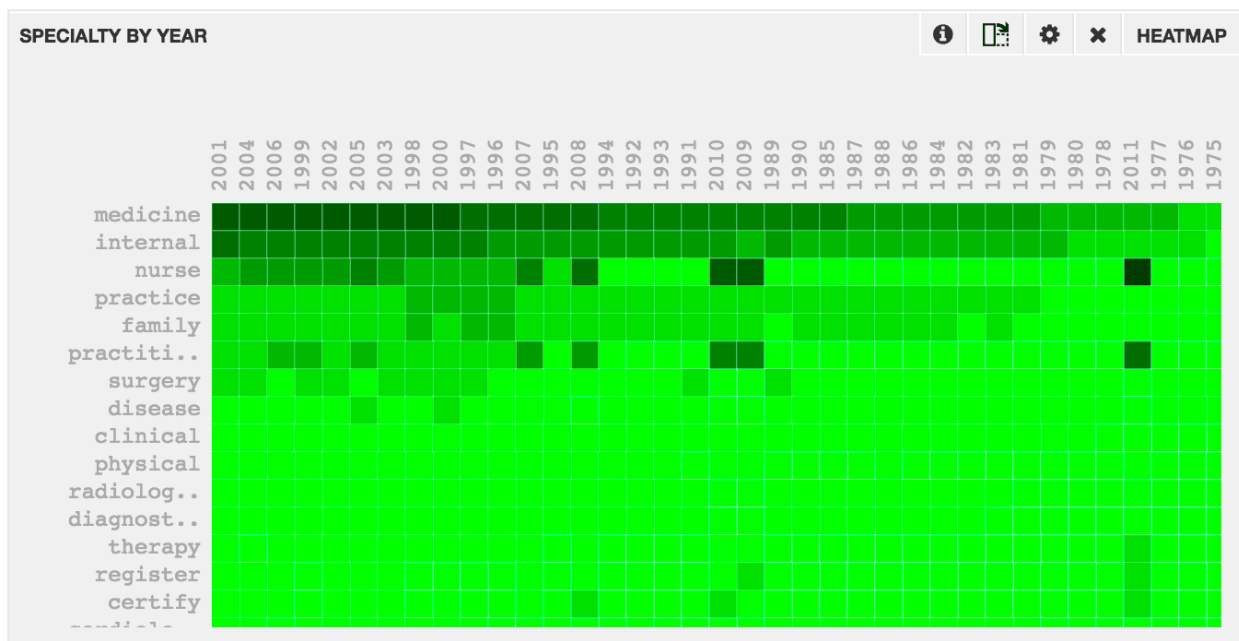
- **Bar.** Graph data as frequencies in a bar chart.



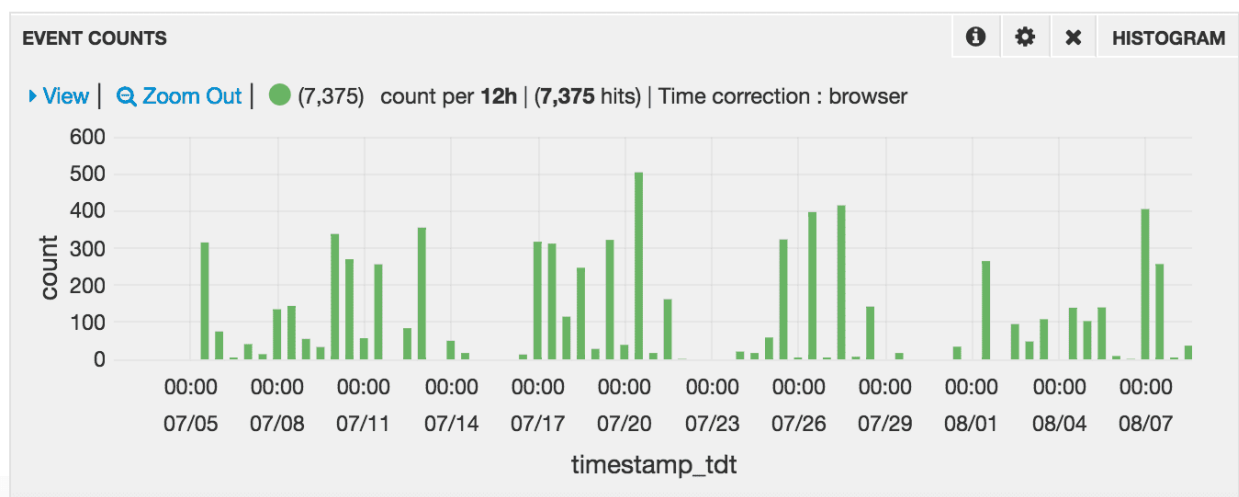
- **Better Map.** Display geolocated points in clustered groups on a map. As you zoom out, points are clustered into fewer groups. As you zoom in, points are clustered into fewer groups (and at some point are not clustered). The Better Map panel *does not* use the geospatial search capabilities of Solr. 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.



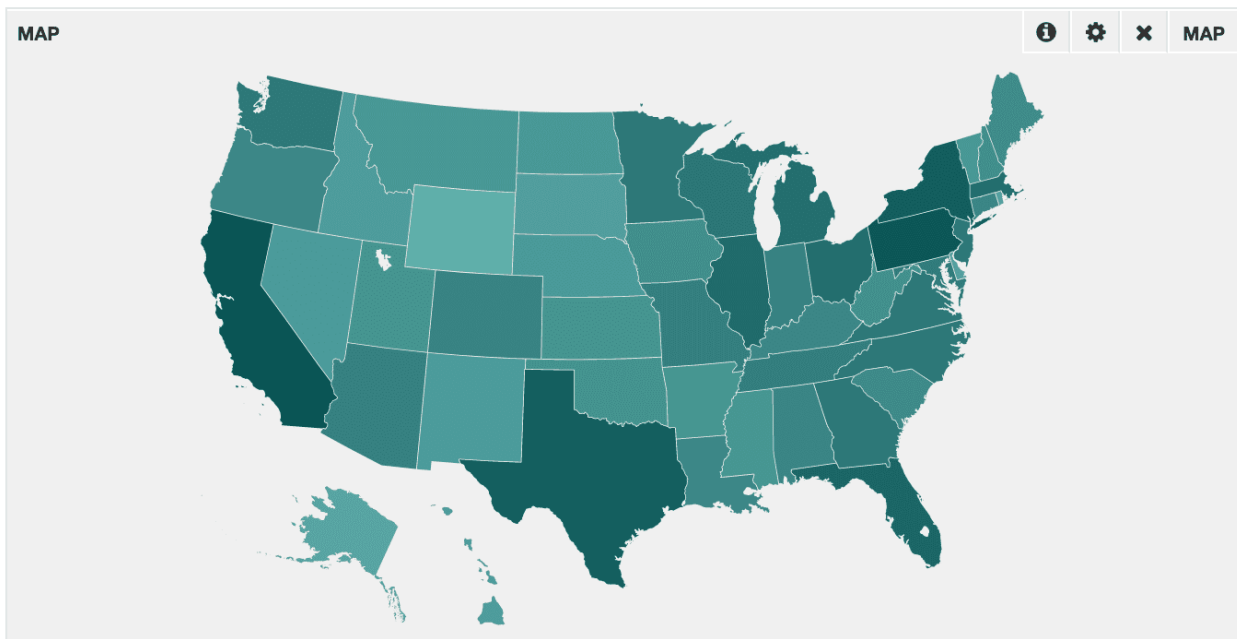
- **Heat Map.** Display a heat map, that is, a graphical representation of data along two facet axes.



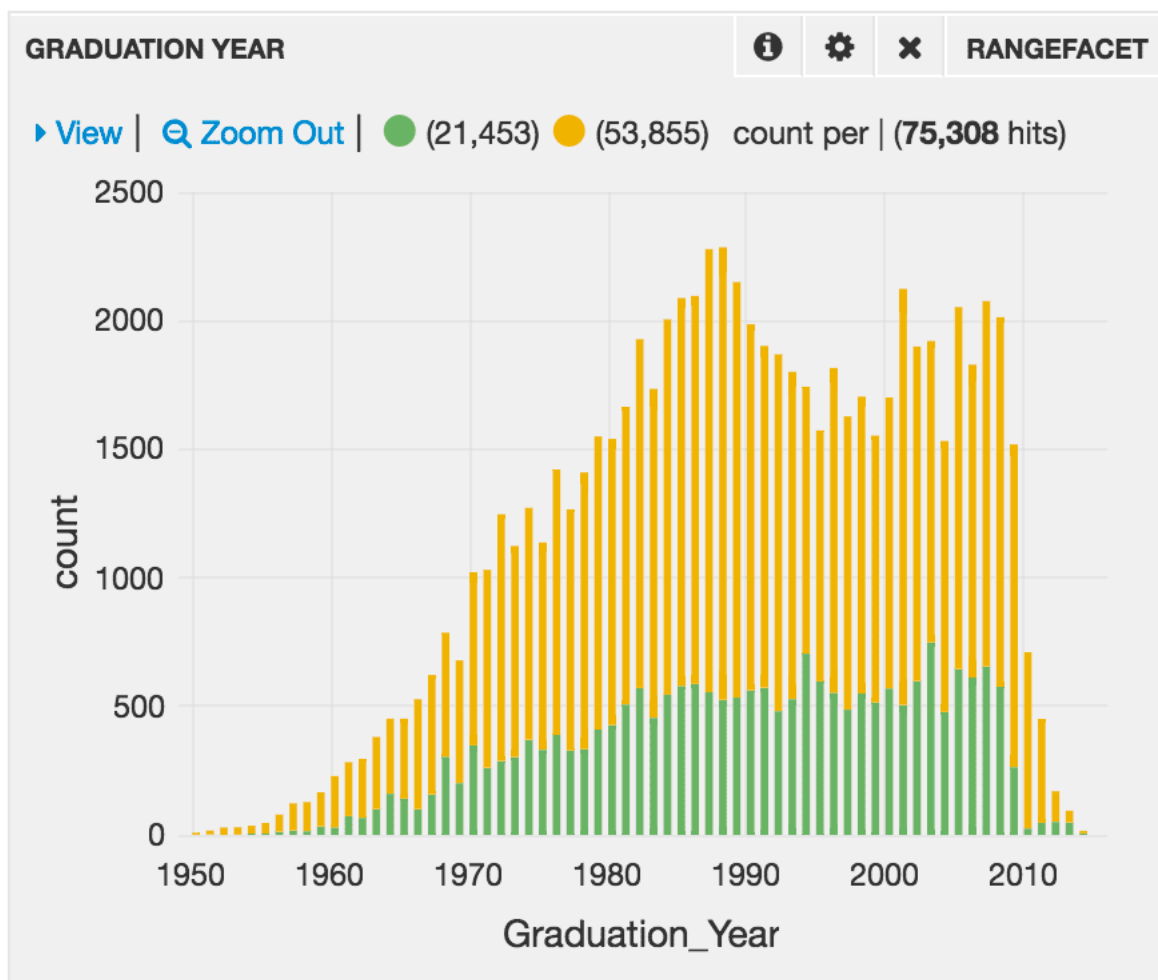
- **Histogram.** Display a histogram. When used in `count` mode, this is a bucketed chart of the current query, including all applied time and non-time filters. When used in `values` mode, the histogram plots the value of a specific field over time, and lets the user group the values based on the values of a second field.



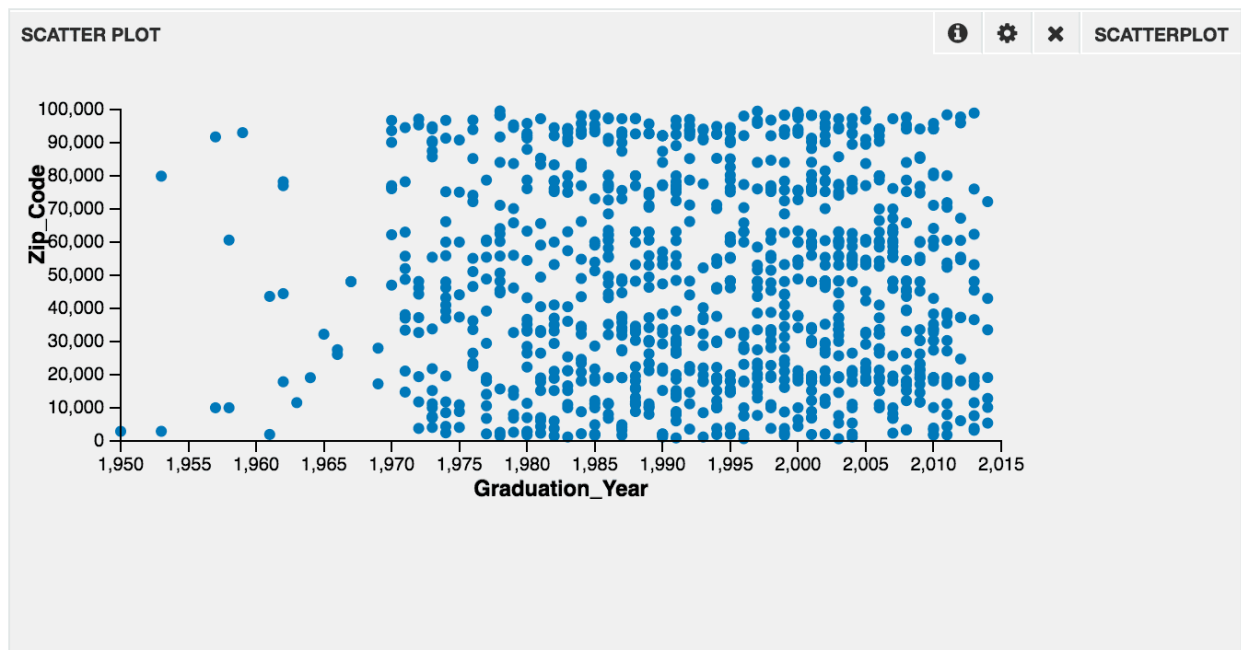
- **Map.** Display a map of shaded regions using a field that contains a 2-letter country code or U.S. state code. Regions with more hits are shaded darker. Instead of a count, you can choose to shade regions based on the minimum, maximum, mean, or sum. The map panel uses facets, so it is important that you set field values to the appropriate 2-letter codes at index time.



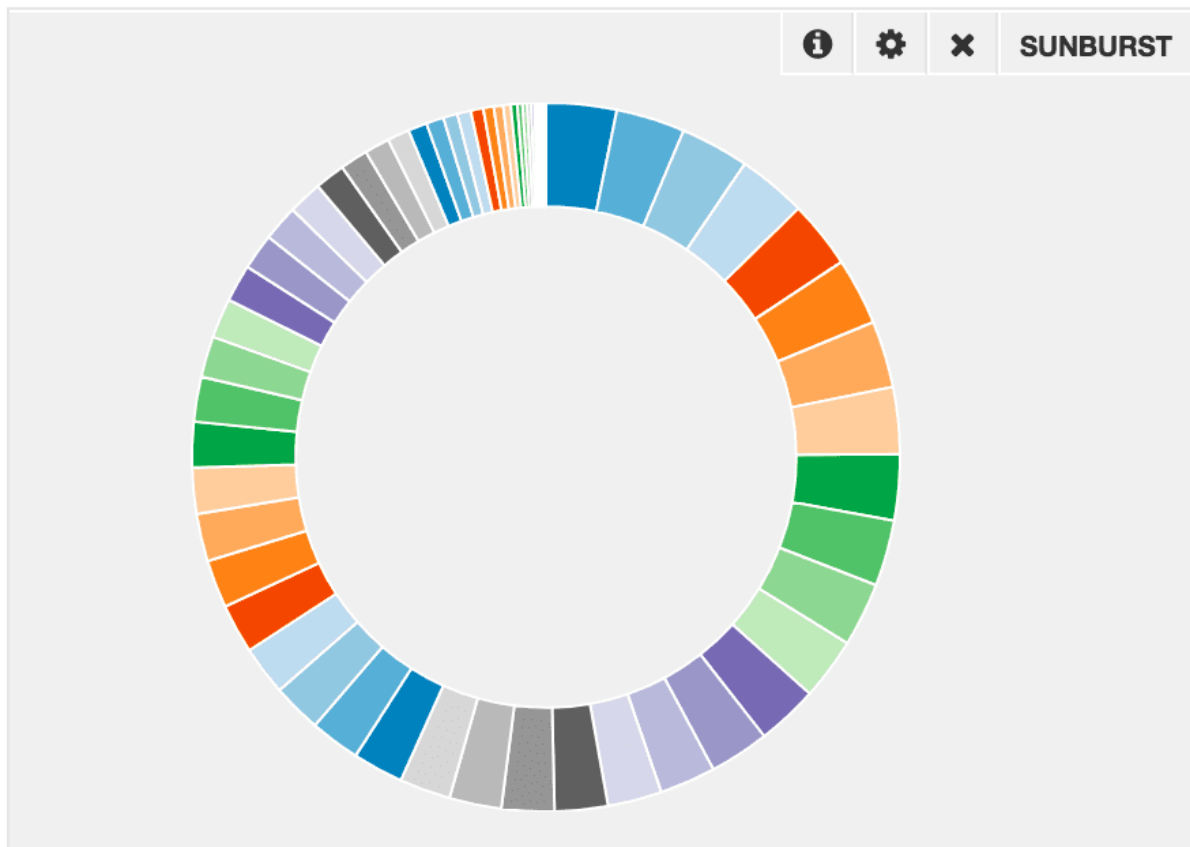
- **Multi-series.** Graph multiple fields of the same type that vary with time in the same graph. The X-axis is used for the time series field. Values in the other fields are graphed along the Y-axis.
- **Range Facet.** Display a histogram of a numeric field. The Range Facet panel is similar to the time 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:



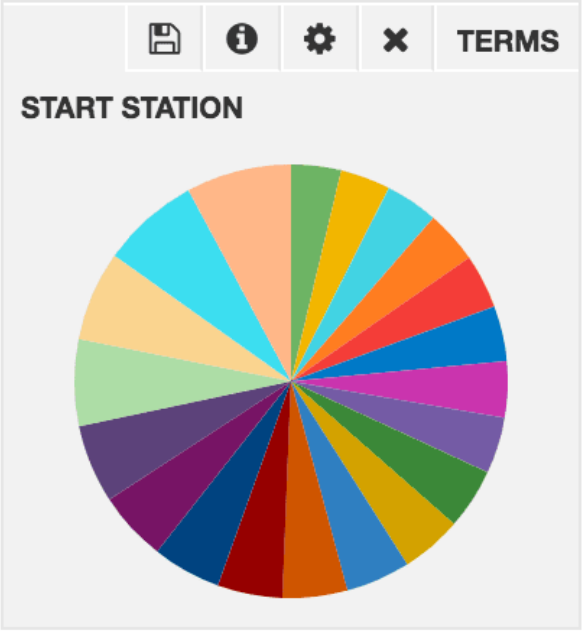
- **Scatter Plot.** Display a scatter plot between two variables or four variables.



- **Sunburst.** Display a sunburst plot based on facets.

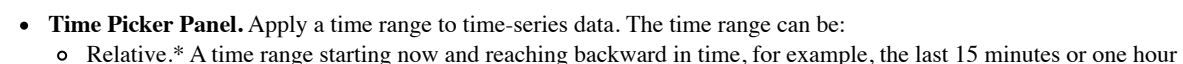


- **Tag Cloud.** Display a tag cloud of the top N words in a field.



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.

- 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:



- o Absolute.* A specific time-and-date range, for example, from 06/01/2017 00:00:00 to 06/30/17 23:59:59

TIME WINDOW

06/29/2017

19:27:46

to

08/08/2017

14:16:21

✓

[Relative](#) | [Absolute](#) | [Since](#)

ⓘ

⚙

✕

TIMEPICKER

- 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.

ADDITIONAL FILTERS

ⓘ

⚙

✕

FACET

Filter

City

▼

Medical_School_Name

▼

Gender

▲

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

You must use a Filtering panel somewhere on your dashboard, so that all panels work correctly when you interact with data.

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 (<https://lucene.apache.org/solr/guide/the-standard-query-parser.html#the-standard-query-parser>). 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 cannot specify `*:value` or `*:"value"` to search over all fields; that syntax does not 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	<code>term</code> (matches any case)
Single term in any field; blanks in term	<code>"term"</code> (matches any case)
Multiple terms, each in any field; with OR logic; no blanks in terms	<code>term1 term2 ...</code>
Multiple terms, each in any field; with OR logic; blanks in terms	<code>"term1" "term2" ...</code> (matches any case)
Multiple terms, each in any field; with AND logic (in the same record); no blanks in terms	<code>term1 AND term2 ...</code>
Multiple terms, each in any field; with AND logic (in the same record); blanks in terms	<code>"term1" AND "term2" ...</code> (matches any case)
Single term in a specific field; no blanks in term	<code>field:term</code> (matches any case)
Single term in a specific field; blanks in term	<code>"field:term"</code> (matches <i>exact</i> case)
Multiple terms, each in a specific field; with OR logic; no blanks in terms	<code>field1:term1 field2:term2 ...</code>
Multiple terms, each in a specific field; with OR logic; blanks in terms	<code>"field1:term1" "field2:term2" ...</code> (matches <i>exact</i> case)
Multiple terms, each in each in a specific field; with AND logic (in the same record); no blanks in terms	<code>field1:term1 AND field2:term2 ...</code>
Multiple terms, each in a specific field; with AND logic (in the same record); blanks in terms	<code>"field1:term1" AND "field2:term2" ...</code> (matches <i>exact</i> case)

For more information about the query syntax, see Standard Query Parser Parameters (<https://lucene.apache.org/solr/guide/the-standard-query-parser.html#the-standard-query-parser>).

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. Do not 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.

Inspect a Panel Query

You cannot 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&w  
t=json&rows=0&fq=start_time:[2014-01-28T20:16:59.000Z%20T0%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
```


System Administration

The Fusion UI provides information for system administration and lets you configure system components.

The REST API provides equivalent functionality. See *System Admin APIs*.

Object Explorer

Object Explorer lets you easily find and manage Fusion objects within a specific app and across multiple apps in an entire Fusion deployment.

See *Object Explorer*.

Log Viewer

Fusion includes some dashboards to view and analyze events in Fusion logs. You can also customize existing dashboards or create new ones. To access these dashboards, click **System > Log Viewer**. See *Dashboards*.

Access Control

Fusion provides application security by restricting access to known users. Users are assigned named sets of permissions which provide access to a specific function called roles. A Security Realm encapsulates a user database together with specific authentication and authorization mechanisms.

The Access Control panel allows you to view and edit *users*, *roles*, and *security realms*.

Blobs

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 addition to the Blob Store API, the Fusion UI provides an interface to the blob store in the Fusion workspace.

See *Blob Storage*.

Import Fusion Objects

You can use the Fusion UI and the Objects API to migrate collections and related objects, including your searchable data, configuration data, pipelines, aggregations, and other objects on which your collections depend. You can also migrate entire apps.

See *Migrate Fusion Objects*.

Messaging Services

The Messaging Services menu lets you configure system notifications, log messages, and alerts to external applications. Several messaging services are available.

See *Messaging Services*.

Scheduler

A job is a runnable Fusion object that performs a specific activity. Datasource jobs, Spark jobs, and tasks are all types of jobs.

The Scheduler provides an interface for scheduling jobs.

See *Jobs and Schedules*.

Solr Clusters

The Solr Clusters menu provides controls for both reporting on and configuring Solr servers. When using Fusion with an existing Solr cluster, this provides a single point of access for both systems.

Solr Config

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

This Solr Config 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 (<https://cwiki.apache.org/confluence/display/solr/Configuring+solrconfig.xml>) for more information on configuring Solr.

System

To access the system dashboard, click  **System > System**. You need to be logged in as `admin` to access the full feature set.

System Overview

The System Overview tab shows an overview of the collections and their states in the cluster. Click a specific collection to view the core name, collection name, Solr memory, system memory, and swap space usage.

API Insight

The API Insight tab shows the available REST API endpoints in the system.

Configurations

The Configurations tab provides read access to all of the global property settings of the system. The UI allows you to filter the list of configuration items by terms and choose the node of the cluster to examine. If you need to change any configuration item, you should use the *Configurations API*.

History

The History tab shows the start and stop times of all services.

Heartbeat Data

The Heartbeat Data tab shows you the information that would be sent to Lucidworks as part of the *System Usage Monitor*, an optional program to report information about your system. You can enable or disable sending the data in this screen, by checking the box next to **Report Heartbeat**.

Metrics

The Metrics tab provides access to the metrics indexed in the `system_metrics` collection, if enabled.

For details, see *System Metrics*.

Threads

The Threads tab shows the details of each active thread in the system. This is the same information returned by the threads endpoint of the *System API*.

Monitoring

Fusion Server has functions built in for observability and monitoring with Fusion's tools or third-party monitoring tools.

Fusion system metrics

Fusion system metrics are logged and available in the UI, in log files, and through the REST API.

- For an overview of noteworthy metrics that are logged and indexed, see *System Metrics*.
- See *System Administration with the Fusion UI*.
- Read about *System Admin APIs*, including APIs for metrics and cluster status, in the REST API Reference Guide.

Additional tools for monitoring Fusion

- Read about other necessary areas to monitor, such as Java Virtual Machines, Unix file systems, networks, and applications like Solr in *Monitor Fusion 4.x.x*.
- Read about health checks for components of your Fusion deployment in *Checking System State*.
- Read about using notifications or alerts in *Messaging Service*.
- Read about monitoring techniques and guidelines in *Cluster Monitoring and Troubleshooting*.

For more information

In the Lucidworks Knowledge Base:

- General guidelines (<https://support.lucidworks.com/article/What-should-you-monitor>) about monitoring Solr.

From our blogs:

- An example of using a third-party tool, Pager Duty, to monitor Fusion (<https://lucidworks.com/post/pagerduty-integration-in-lucidworks-fusion/>).

Cluster Monitoring and Troubleshooting

Fusion structure and components

Fusion consists of a collection of services running on one or more nodes in a network, as summarized below.

The default ports listed under Runs on below can be altered in <code>fusion.properties</code> file located in <code>/fusion/4.1.x/conf/</code> .

Fusion services

Service	Runs on	Description
Fusion UI	All nodes in Fusion cluster. Port 8763.	Fusion Admin Web App.
API	All nodes in Fusion cluster. Port 8765.	The API service fulfills REST API requests, processes pipeline stages, and starts, stop, and sometimes monitors scheduled jobs.
Connectors	Fusion nodes. Port 8984.	The connectors service runs connector jobs and processes pipeline stages.
Proxy	All nodes in Fusion cluster. Port 8764.	The proxy service authenticates API requests sent to port 8764 and dispatches to an API or connectors service over port 8765 or 8984.
Apache Solr	All Signals nodes. Port 8983.	Solr for various Fusion collections. <code>signals</code> , <code>aggr</code> , <code>logs</code> , <code>various system* collections</code> such as models, blobs, history, messages, metrics, etc. On larger installations, Solr is often installed and run separately from Fusion.
Apache Spark	Signals nodes. Various ports.	The Apache Spark service provides large-scale data processing jobs for Spark SQL, aggregation, machine learning, etc.
Web Apps	All nodes in Fusion cluster. Port 8780.	The web apps service runs installed App Studio and/or Rules Editor web applications (war).
Apache Zookeeper	Port 9983.	Apache ZooKeeper configures and manages all Fusion components in a single Fusion deployment. ZooKeeper is often installed and run separately from Fusion. In this case, it typically runs on port 2181.

Typical directory structure

```
├── bin    # scripts for start|stop|status of Fusion services
├── conf   # contains config for Fusion processes and logging
├── data   # data directories (often linked or mounted to larger disk)
│   ├── connectors
│   ├── nlp
│   ├── solr
│   └── zookeeper
├── var    # default location for log files.
│   └── log
│       ├── admin-ui
│       ├── agent    # agent used to start/stop services
│       ├── api      # API service calls, jobs, and pipeline stage output
│       ├── connectors
│       │   ├── connectors-classic # Connector and pipeline output
│       │   └── connectors-rpc
│       ├── log-shipper
│       ├── proxy
│       ├── solr
│       ├── spark-master
│       ├── spark-worker
│       ├── sql
│       ├── webapps
│       └── zookeeper
```

In general, each log directory contains a log named after the directory followed by a `.log` suffix. garbage collection (GC) logs are also located in these directories. The GC logs show internal timings for Java garbage collection cycles, as well as logs showing Jetty requests and errors for the given service. Each Fusion service has a `log4j2.xml` settings file located in `/fusion/{fusion}.x/conf` which controls the particulars of log retention and contents.

By default, file rotation is enabled in the `log4j2.xml` settings file. File rotation ensures a new log file is created under the following conditions: * A 24-hour period has elapsed * A log file has exceeded 100MB

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 (<https://lucidworks.com/post/alerts-lucidworks-fusion/>).

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 (<http://www.slack.com>) 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*.

Messaging Service Configuration

The Message Services as a whole can be configured via the *Configurations API* with these attributes:

Attribute	Description
<code>rateLimit</code>	The time, in milliseconds, to wait between sending messages on a per-second basis. This does not synchronize throttling between requests.
<code>storeAllMessages</code>	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 messaging services must be explicitly enabled.

You can do this through the UI at **System > System > Messaging Services**, or through the *Messaging API*.

How to see which messaging services are currently enabled:

```
curl -u USERNAME:PASSWORD http://localhost:8764/api/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.

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.

By default, only scheduled messages are stored. To configure Fusion to store all messages, see <i>Messaging Service Configuration</i> above. The default collection for message storage is <code>system_messages</code> , 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.

Troubleshooting Performance Issues

Troubleshooting

High CPU load

Heavy CPU load across all nodes in the cluster might indicate the need for more nodes. More typically, however, a single node or service is being overloaded.

- Use `top` or another monitoring utility to find out which processes are using high CPU.
- All Fusion services, including Solr and Zookeeper, run as Java processes. Most services indicate the name of the service via the `-DserviceName` command-line argument.

Common reasons for busy services:

- Connectors-classic: document parsing and ingestion
- Connectors-classic: document-heavy index and/or query pipeline use
- API: document-heavy index and/or query pipeline use
- Solr: use across multiple nodes holding the same collection
- Solr: use across multiple nodes holding shard leaders
- Heavy query load involving facets, stats, and/or sorting
- Heavy document writing to a given collection

What to do:

- Find out which services or nodes are being overloaded.
- Find out if there is known activity to explain the excessive CPU load.
- If no explanation is found, Contact Lucidworks Support (<https://support.lucidworks.com/hc/en-us/requests/new>).

Low memory

Low memory problems are typically the result of two issues: low system memory and a lack of heap space for an individual service.

Lack of free memory

You can detect a lack of free memory on a server using the `free -h` command. On servers running the Solr service, large amounts of free memory is ideal because this memory is used for filesystem caching. Take note of the amount of free memory and the disk cache in comparison to the overall index size. Ideally, the disk cache should be able to hold all of the index, although this is not as important when using newer SSD technologies.

There are several ways to increase the available memory on a server:

- Change the command-line options configured in `conf/fusion.properties` to reduce the heap sizes of individual services. However, this might result in a *lack of heap space*, as described below.
- Run fewer services on a node and/or reallocate services that need more memory to nodes with extra capacity.
- Add nodes to the cluster.
- Add memory to the nodes.

Lack of heap space

Finding processes that have exited with `OutOfMemoryError` in the logs, or finding a dump file called `java_pidXXX.hprof` in the log directory, indicates that a service failed due to lack of heap space.

Heap space is configured on a per-service basis in the `conf/fusion.properties` file via the `-Xmx` and `-Xms` command-line parameters. Avoid allocating heap sizes that are known to be larger than needed for a service, because these can lead to long GC pauses.

Two common ways to detect long GC pauses include:

- Examine the `gc_*.log` files in the log directories. For deep analysis, upload files to <http://gceasy.io> (<http://gceasy.io>).
- Using `top`, look for periods when all cores are busy followed by a spike in one core (with most or all other cores dropping to near zero). This pattern is typically found when one service is busy and is encountering long GC pauses.

The need for GC analysis varies from application to application.

High response time at proxy and api layer

Under regular load with some spike, the response time might go very high.

The default values for both of these properties that control on the number of connections available in the pool is `50`:

- `com.lucidworks.apollo.admin.proxy.max.conn.per.route`: controls the number of connections per route
- `com.lucidworks.apollo.admin.proxy.max.conn.total`: controls the max number of connections

Check the proxy logs:

```
019-08-26T18:20:26,377 - INFO [main:clojure.tools.logging$eval11$fn__16@0] - {} -  
max-per-route: 50 , max-conn-total: 50
```

In the `conf/fusion.properties` file, modify the property values. If the number of connections is a bottleneck, configure the connections for the proxy JVM parameters. For example, configure the pool to have 150 connections:

```
proxy.jvmOptions = -Xmx${PROXY_MEM:-512m} -  
Dcom.lucidworks.apollo.admin.proxy.max.conn.per.route=150 -  
Dcom.lucidworks.apollo.admin.proxy.max.conn.total=150
```

The proxy logs should now show the max connections set to 150:

```
019-08-26T18:25:26,377 - INFO [main:clojure.tools.logging$eval11$fn__16@0] - {} -  
max-per-route: 150 , max-conn-total: 150
```

System Usage Monitor

The System Usage Monitor is a voluntary program to allow users to anonymously send metrics 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 (<https://www.lucidworks.com/privacy-policy>).

Information Collected by the Usage Monitor

The System Usage Monitor collects the following types of information:

- The uuid, a randomly generated identifier per Fusion cluster.
- System information, including the operating system, version, and Java version.
- Fusion information, including the version, number of Fusion nodes, and number of CPU cores in use.
- Solr statistics including the number of collections, number, types, and indexing rates of documents, and number, type, and rate of search requests.
- Aggregation information, including number, types, and rates of aggregation runs, and number, types, and rates of signals requests.
- Recommendations information, including number, types, and rates of recommendation requests.

You can see the data that will be sent to Lucidworks with the *Usage API* and also in the UI by navigating to **System > System**, then clicking the **Heartbeat Data** tab.

The UI and the REST API report only the data currently scheduled to be sent, so they are not a complete picture of all data collected.

How Data is Sent

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

When Fusion is started, the System Usage Monitor transmits data about your system to a server hosted by Lucidworks with two HTTP requests. The first request contains system-level information. If the first request is successful, the second request sends system-specific information.

The information is sent via an encrypted POST request to the heartbeat service. Each request includes a unique identifier, which is anonymous and cannot be used to identify the sender. The IP that sent the request is not stored with the request.

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:

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

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

Jobs

A *job* is a runnable Fusion object that performs a specific activity. Datasource jobs, Spark jobs, and tasks are all types of jobs. Examples of the activities include:

- Aggregating signal data
- Training a shallow neural model and projecting each document onto the resulting vector embedding space
- Loading data into a data source

Job types

Each job type is a type of Fusion object that you can run or schedule to be run.

<code>datasource</code>	A job to ingest data according to the specified datasource configuration, such as <code>datasource:movie-db</code> . Datasources are created using the <i>Connector Datasources API</i> or the Fusion UI. See <i>Datasource Jobs</i> .
<code>spark</code>	A Spark job to process data, such as <code>spark:dailyMetricsRollup-counters</code> . Spark jobs are created using the <i>Spark Jobs API</i> or the Fusion UI. See <i>Spark Jobs</i> .
<code>task</code>	A job to perform an HTTP call or log cleanup, such as <code>task:delete-old-system-logs</code> . Tasks are created using the <i>Tasks API</i> or the Fusion UI. See <i>Tasks</i> .

The Jobs manager

The Jobs manager, available in the Fusion workspace at **Collections > Jobs**, provides a simple interface for viewing and scheduling jobs. You can also create tasks and Spark jobs here.

You cannot create, run, or schedule datasource jobs in the Jobs manager. You must create them on the Datasources page, in the Index Workbench, or in the Quickstart. To run a datasource job, use the *scheduler*.

Datasource Jobs

A datasource is a configuration that manages the import and indexing of data into the collection.

Datasources are created using the *Index Workbench*, or at **Indexing > Datasources**, or using the *Connector Datasources API*.

A datasource is also a job that can run on demand or on a schedule.

Fusion Schedules

A *schedule* defines when Fusion will perform a job.

You can schedule a job using `CRON` notation, ISO-8601 interval definitions, or triggers that depend on the results of other jobs.

You can also run jobs on demand.

The *Jobs API* lets you view jobs and their run histories, configure their schedules, and control them directly.

The Scheduler

The Scheduler, available in the Fusion workspace at **System > Scheduler**, provides an interface for scheduling jobs.

You cannot create jobs in the Scheduler. To create new jobs, use the *Jobs manager* or the REST API for each *job type*.

Job History

Checking a job status

With the API

You can see the current job states of all datasources for the app `APP_NAME` by calling the Jobs API:

```
curl -u USERNAME:PASSWORD http://proxy-url:8764/api/apollo/apps/APP_NAME/jobs?
type=datasource
```

If a job has never run, a Jobs API call returns the following:

```
[{ "resource": "datasource:myDatasource", "enabled": true, "startedBy":
"09765f1d-2f1c-4c3b-ba22-887e3886d8aa", "status": "ready", "extra": {} }]
```

The value `datasource:myDatasource` indicates a datasource, `myDatasource`, is present. The line `"status": "ready"` indicates the current state of the job, which has never run.

After a job successfully completes, the following is returned:

```
[{ "resource": "datasource:myDatasource", "enabled": true, "startedBy":
"09765f1d-2f1c-4c3b-ba22-887e3886d8aa", "status": "success", "extra": {
  "counter.failed": 0, "counter.other.pipeline.in": 58242, "counter.stage.field-
mapping::d3ebd921-6065-479e-9d45-0bfed819346b.processed": 58241,
  "counter.deleted": 0, "counter.stage.solr-dynamic-field-name-mapping::e6aa9c51-
2867-40a0-97fd-34d909098a20.processed": 58241, "counter.new": 2268,
  "counter.output": 58241, "counter.other.pipeline.complete": 58242,
  "datasourceId": "myDatasource", "counter.input": 60435, "startTime":
1586298855843, "endTime": 1586299313433, "counter.skipped": 2194,
  "counter.stage.solr-index::01e1c561-0440-44b2-afb4-d6ff471f646c.processed":
58241, "counter.other.pipeline.out": 58242 }, "lastStartTime": "2020-04-
07T22:34:15.852Z", "lastEndTime": "2020-04-07T22:41:59.271Z" }]
```

To see the complete job history, use the following Jobs API call:

```
curl -u USERNAME:PASSWORD http://proxy-
url:8764/api/apollo/datasource:myDatasource/history
```

The following is returned:

```
[ { "resource" : "datasource:myDatasource", "runId" : "fXfKTQNSKM", "startTime" :
"2020-04-07T22:34:15.852Z", "endTime" : "2020-04-07T22:41:59.271Z", "status" :
"success", "extra" : { "counter.failed" : 0, "counter.other.pipeline.in" : 58242,
"counter.stage.field-mapping::d3ebd921-6065-479e-9d45-0bfed819346b.processed" :
58241, "counter.deleted" : 0, "counter.stage.solr-dynamic-field-name-
mapping::e6aa9c51-2867-40a0-97fd-34d909098a20.processed" : 58241, "counter.new" :
2268, "counter.output" : 58241, "counter.other.pipeline.complete" : 58242,
"datasourceId" : "myDatasource", "counter.input" : 60435, "startTime" :
1586298855843, "endTime" : 1586299313433, "counter.skipped" : 2194,
"counter.stage.solr-index::01e1c561-0440-44b2-afb4-d6ff471f646c.processed" :
58241, "counter.other.pipeline.out" : 58242 }, "startedBy" : "09765f1d-2f1c-4c3b-
ba22-887e3886d8aa" }, { "resource" : "datasource:myDatasource", "runId" :
"XrxochV0B", "startTime" : "2020-04-07T20:06:25.819Z", "endTime" : "2020-04-
07T20:14:01.686Z", "status" : "success", "extra" : { "counter.failed" : 0,
"counter.other.pipeline.in" : 59090, "counter.stage.field-mapping::d3ebd921-6065-
479e-9d45-0bfed819346b.processed" : 59089, "counter.deleted" : 0,
"counter.stage.solr-dynamic-field-name-mapping::e6aa9c51-2867-40a0-97fd-
34d909098a20.processed" : 59089, "counter.new" : 2268, "counter.output" : 59089,
"counter.other.pipeline.complete" : 59090, "datasourceId" : "myDatasource",
"counter.input" : 61283, "startTime" : 1586289985809, "endTime" : 1586290440433,
"counter.skipped" : 2194, "counter.stage.solr-index::01e1c561-0440-44b2-afb4-
d6ff471f646c.processed" : 59089, "counter.other.pipeline.out" : 59090 },
"startedBy" : "09765f1d-2f1c-4c3b-ba22-887e3886d8aa" }, { "resource" :
"datasource:myDatasource", "runId" : "2jy3UemLTp", "startTime" : "2020-04-
07T19:23:39.576Z", "endTime" : "2020-04-07T19:31:26.694Z", "status" : "success",
"extra" : { "counter.failed" : 0, "counter.other.pipeline.in" : 58906,
"counter.stage.field-mapping::d3ebd921-6065-479e-9d45-0bfed819346b.processed" :
58905, "counter.deleted" : 0, "counter.stage.solr-dynamic-field-name-
mapping::e6aa9c51-2867-40a0-97fd-34d909098a20.processed" : 58905, "counter.new" :
2268, "counter.output" : 58905, "counter.other.pipeline.complete" : 58906,
"datasourceId" : "myDatasource", "counter.input" : 61099, "startTime" :
1586287419556, "endTime" : 1586287883825, "counter.skipped" : 2194,
"counter.stage.solr-index::01e1c561-0440-44b2-afb4-d6ff471f646c.processed" :
58905, "counter.other.pipeline.out" : 58906 }, "startedBy" : "09765f1d-2f1c-4c3b-
ba22-887e3886d8aa" } ]
```

The section *How does the Jobs API work?* below gives additional details on how the Jobs API retrieves information from Fusion. See the *Jobs API reference* for more information.

Job history results field definitions

Field	Definition
counter.deleted	Total number of documents deleted or removed from the Solr index.
counter.failed	Total number of documents not indexed due to failure. The errors are written to the log.
counter.input	Total number of fetched documents passed to the index pipeline.
counter.new	Total number of documents fetched for the first time.
counter.output	Total number of documents processed in the run (sum of failed + deleted + indexed documents).
counter.skipped	Total number of documents skipped due to rules such as max, min, and file size.
endTime	Time the job concluded, aborted, or was manually stopped.
resource: datasource	Name of the datasource.
startTime	Time the job started.
status	Job status such as Success, Running, or Aborted.

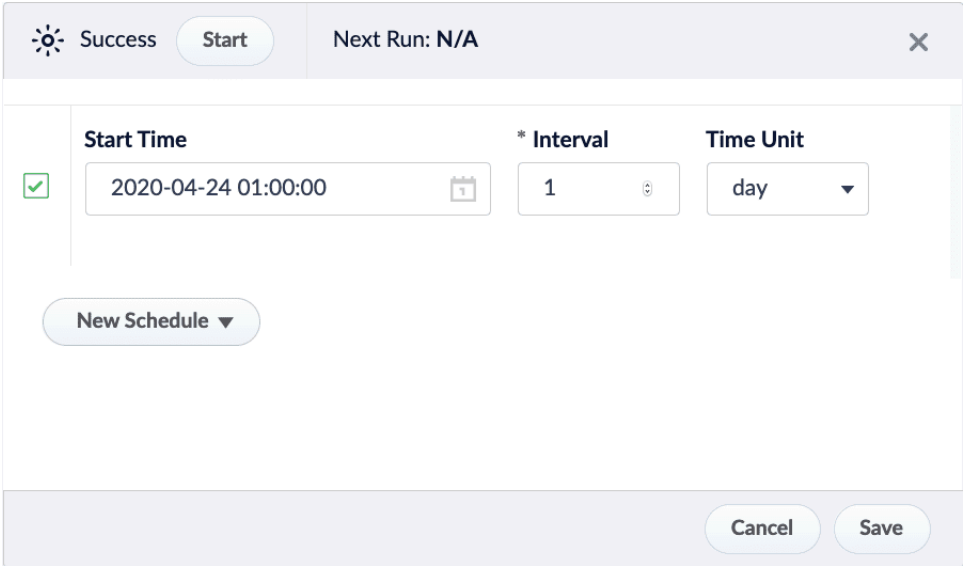
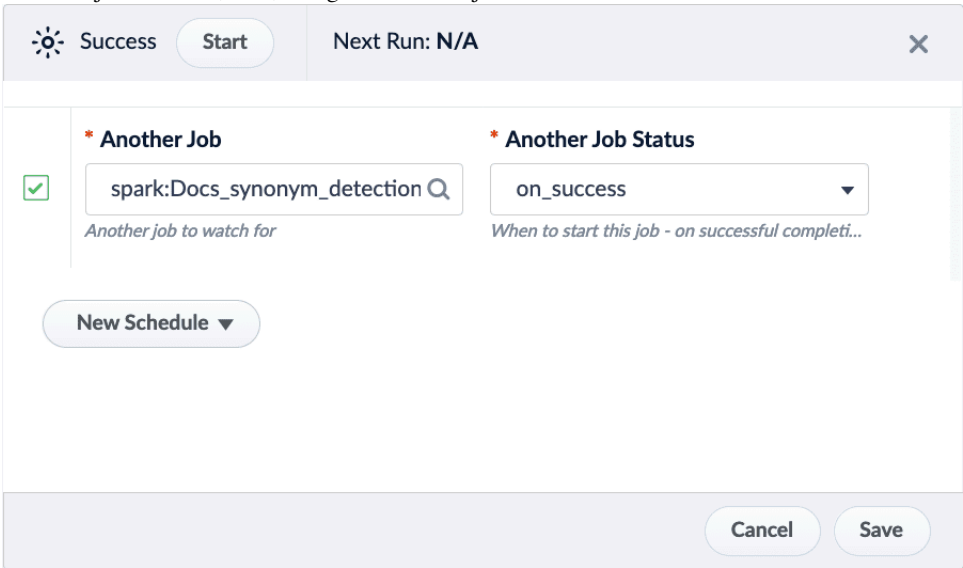
With the UI

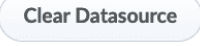
In the Fusion UI, navigate to **Indexing > Datasources**. Click a datasource to open the datasource panel.

The screenshot shows the 'myDatasource' configuration panel in the Lucidworks Fusion UI. The panel includes a left sidebar with navigation icons and a main configuration area. The configuration area contains the following fields and controls:

- Run** button (with a calendar icon) and **Job History** link.
- Last run start:** Thursday, 4/23/2020 at 15:24:56
- Last run stop:** Thursday, 4/23/2020 at 15:25:01
- myDatasource (Fileupload)** description: A connector that can crawl a single file that has been uploaded through Fusion's Blob Store.
- Datasource ID:** myDatasource
- Pipeline ID:** Docs
- Parser:** Docs
- Description:** Optional description for this datasource.
- File Upload:** Browse... No file selected. Upload File button.
- File ID:** myDatasource.zip
- File Media Type:** application/octet-stream
- Delete datasource** link.

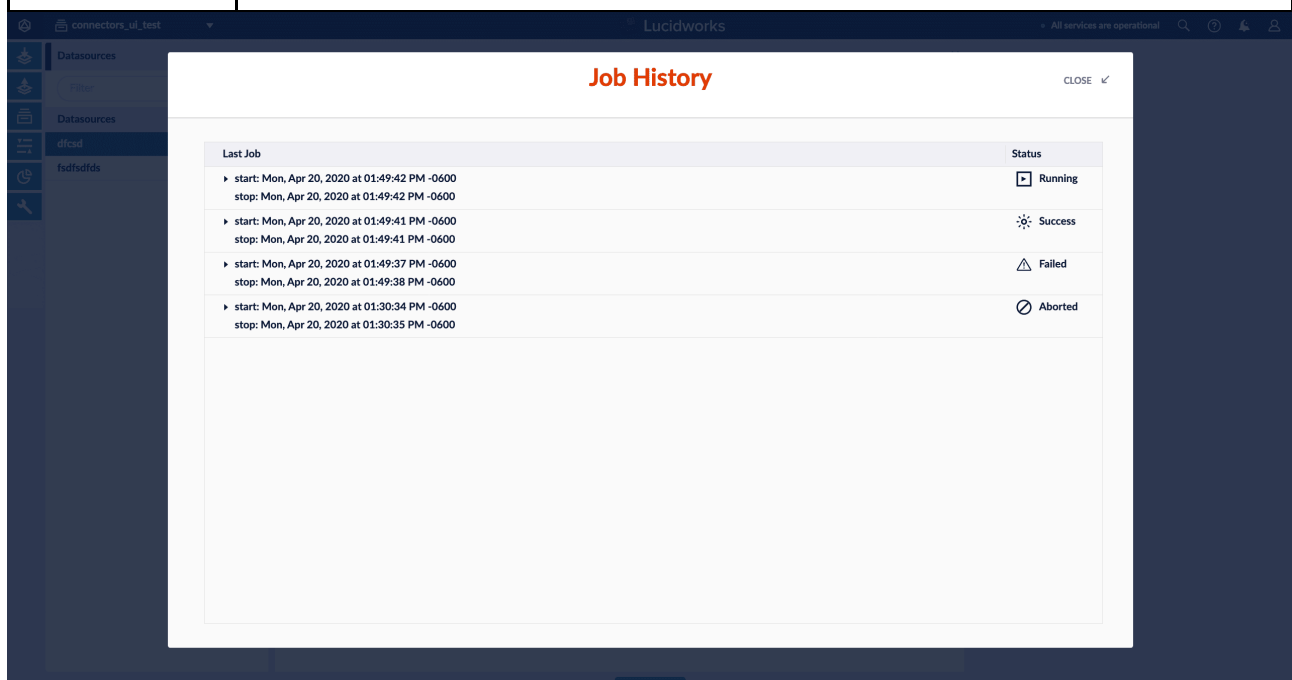
Note that the **Last run start** and **Last run stop** values are equivalent to the `lastStartTime` and `lastEndTime` values from the Jobs API call. (These values are only present if a job has been run at least once.)

Icon	Description
	The Run button opens a panel containing various run options and information.
	The Start button starts a job.
	The New Schedule button displays a dropdown with several scheduling options: • cron. Schedule a job based on a Crontab expression. See Cron Trigger documentation (http://www.quartz-scheduler.org) for more info. • interval. Schedule a job to run on an interval after a set starting point. For example, you can schedule a job to run every day at 1:00AM.  • job_completion. Schedule a job to run after another job has completed. You can schedule the job to run if the job succeeds, fails, or regardless of the job's success. 

Icon	Description
	The Stop button stops a running job. This results in a status of <code>"status" : "aborted"</code> .
	The Job History button opens the <i>Job History panel</i> . Click one of the job entries to see additional details. Expand the job details for more information. Click Open Logs in Dashboard for complete information.
	The Clear Datasource removes documents from your collection that are indexed from that datasource. The job history will <i>not</i> be deleted.

In the job history panel, there is an icon on each datasource indicating its latest run history. The possible statuses are:

Icon	Description
 Never Run	The job has been created but has not been run.
 Running	The job is currently running. Depending on the number of documents being processed, the job can take a long time to complete. Stop the job with the Stop button.
 Success	The job was completed successfully.
 Failed	The job was not able to successfully run. Expand the job details for more information. Click Open Logs in Dashboard for complete information.
 Aborted	The job was manually stopped before it could finish.



The screenshot shows the Lucidworks interface with the 'Job History' panel open. The panel displays a table of job entries. The table has two columns: 'Last Job' and 'Status'. The 'Last Job' column contains details about the job, including start and stop times. The 'Status' column shows the job's status with a corresponding icon.

Last Job	Status
- start: Mon, Apr 20, 2020 at 01:49:42 PM -0600 - stop: Mon, Apr 20, 2020 at 01:49:42 PM -0600	 Running
- start: Mon, Apr 20, 2020 at 01:49:41 PM -0600 - stop: Mon, Apr 20, 2020 at 01:49:41 PM -0600	 Success
- start: Mon, Apr 20, 2020 at 01:49:37 PM -0600 - stop: Mon, Apr 20, 2020 at 01:49:38 PM -0600	 Failed
- start: Mon, Apr 20, 2020 at 01:30:34 PM -0600 - stop: Mon, Apr 20, 2020 at 01:30:35 PM -0600	 Aborted

The image above shows all possible job statuses.

How does the Jobs API work?

In *Checking a job status with the API* above, two Jobs API endpoints were used to call the datasource `myDatasource`:

- `APP_NAME/jobs?type=datasource` - Retrieves the current datasource job states for *all* datasources of a specified app. In this case, the app is specified as `APP_NAME`.
You can also use `type=spark` to retrieve all Spark jobs and `type=task` for task jobs.
- `/jobs/datasource:myDatasource/history` - Retrieves a specific object's job history. In this case, the object type is a datasource, as selected with `datasource:`.

Datasources are not the only object type for jobs. There are also Spark jobs (`spark:`) and task jobs (`task:`).

Both of the API calls above send a request to Fusion's API application, which serves the Jobs API. The API application contains two main components related to job history data, `JobController` and `SolrJobHistoryStore`.

The Job Controller

The Job Controller is responsible for getting any active job's run status, if an active job is running.

In the case of V1 connectors classic datasource jobs, it will reach out to Fusion's connectors-classic application's connectors API.

When the API application calls the connectors-classic Jobs API, it will first use Zookeeper to identify the IP address of the connector-classic node that is assigned to this datasource, if one exists.

When the connectors-classic node IP assigned to the job is identified, the API application will call out to the connectors-classic node via `http://<connectors-node-ip>:8984/connectors/v1/connectors/jobs/myDatasource` (`http://<connectors-node-ip>:8984/connectors/v1/connectors/jobs/myDatasource`). The call reaches the component `ConnectorsManagerController` to query the job status of currently running jobs with the ID `myDatasource`.

The Solr Job History Store

`SolrJobHistoryStore` reads and writes a job's history data to and from Solr. Fusion's API application creates a special system Solr collection, `system_jobs_history`, to store the history.

You can also query the job history from Solr directly, if you want:

The screenshot displays the Solr Admin interface. On the left, the 'Query' tab is selected under the 'system_jobs_history' collection. The 'Request-Handler (qt)' is set to '/select'. The 'q' field contains the query: `resource_s:"datasource:sharepoint"`. The 'fq' field is empty. The 'sort' field is set to 'timestamp_dt desc'. The 'start, rows' are set to '0' and '10' respectively. The 'Raw Query Parameters' section shows 'key1=val1&key2=val2'. The 'wt' dropdown is set to 'json'. The 'indent' checkbox is checked. The 'debugQuery' checkbox is unchecked. The 'Execute Query' button is visible at the bottom of the query form.

The right pane shows the JSON response from the query. The response includes a 'responseHeader' with 'zkConnected': true, 'status': 0, 'QTime': 0, and 'params'. The 'response' object contains 'numFound': 4, 'start': 0, and 'docs'. The first document in the 'docs' array is the one shown in the screenshot, representing a job with status 'SUCCESS' and timestamp '2020-04-07T22:41:59.271Z'.

```
{
  "responseHeader": {
    "zkConnected": true,
    "status": 0,
    "QTime": 0,
    "params": {
      "q": "resource_s:\\\"datasource:sharepoint\\\"*",
      "sort": "timestamp dt desc",
      "start": "0",
      "end": "10"
    }
  },
  "response": {
    "numFound": 4,
    "start": 0,
    "docs": [
      {
        "id": "17156ceb3e77df7b1f9df2",
        "jobRunId_s": "FXFKTONSKM",
        "resource_s": "datasource:sharepoint",
        "startedBy_s": "09765f1d-2f1c-4c3b-ba22-887e3886d8aa",
        "startTime_dt": "2020-04-07T22:34:15.852Z",
        "endTime_dt": "2020-04-07T22:41:59.271Z",
        "timestamp_dt": "2020-04-07T22:41:59.271Z",
        "status_s": "SUCCESS",
        "type_s": "history",
        "extra_counter.failed": [0],
        "extra_counter.other.pipeline.in": [58242],
        "extra_counter.deleted": [0],
        "extra_counter.new": [2268],
        "extra_counter.output": [58241],
        "extra_counter.other.pipeline.complete": [58242],
        "extra_datasourceId": "sharepoint",
        "extra_counter.input": [60435],
        "extra_startTime": [1586298855843],
        "extra_endTime": [1586299313433],
        "extra_counter.skipped": [2194],
        "extra_counter.other.pipeline.out": [58242],
        "raw_content": "ewogICJyZXNvdXJjZS9pOAIzZGF0YXNvdXJjZTpwagFyZX8vaW50IiwKICAicnVusWQIIDogImZyZktUUUS",
        "extra_counter.stage.field-mapping_d3ebd921-6065-479e-9d45-0bfd819346b.processed": [58241],
        "extra_counter.stage.solr-dynamic-field-name-mapping_e6aa9c51-2867-40a0-97fd-34d909098a20.processed": [58241],
        "extra_counter.stage.solr-index_01elc561-0440-44b2-afb4-d6ff471f646c.processed": [58241],
        "_version_": 1663355395060605248
      }
    ]
  }
}
```

Spark Jobs

Apache Spark can power a wide variety of data analysis jobs. In Fusion, Spark jobs are especially useful for generating *recommendations*.

Spark job subtypes

For the Spark *job type*, the available subtypes are listed below.

- *Aggregation*
Define an aggregation job.
- *Script*
Run a custom Scala script as a Fusion job.

Additional Spark jobs are available with a <i>Fusion AI</i> license.

Spark job configuration

Spark jobs can be created and modified using the *Fusion UI* or the *Spark Jobs API*. They can be scheduled using the Fusion UI or the *Jobs API*.

For the complete list of configuration parameters for all Spark job subtypes, see the *Jobs Configuration Reference*.

Tasks

Tasks are a flexible job type that can be used to clean up old logs or run any REST call.

Task subtypes

The "task" job type has the following subtypes:

- *Log Cleanup*
Delete old log messages from system logs collection.
- *REST Call*
A versatile job type that runs an arbitrary REST/HTTP/Solr command.

Task configuration

Tasks can be created and modified using the *Fusion UI* or the *Tasks API*. They can be scheduled using the Fusion UI or the *Jobs API*.

See also *Jobs Configuration Reference*.

Fusion SQL Query and Analytics

Most organizations that deploy Fusion also have SQL-compliant business intelligence (BI) or dashboarding tools to facilitate self-service analytics.

The Fusion SQL service:

- Lets organizations leverage their investments in BI tools by using JDBC and SQL to analyze data managed by Fusion. For example, Tableau is popular data analytics tool that connects to Fusion SQL using JDBC to enable self-service analytics.
- Helps business users access important data sets in Fusion without having to know how to query Solr.

See *Fusion SQL Administration* to learn about architecture and configuration.

See the reference topics below to learn how to query Fusion using the Fusion SQL engine:

In addition to the specified *System Requirements*, Fusion on Windows requires Visual C++ Redistributable for Visual Studio 2015 (<https://www.microsoft.com/en-us/download/details.aspx?id=48145>) to start the SQL service successfully.

Approaches for optimizing the performance of queries

In general terms, the Fusion SQL service supports three approaches for optimizing the performance of queries:

- **Join or aggregate in Spark.** Read a set of raw rows from Solr and join or aggregate the rows in Spark. For queries that rely on Spark performing joins and aggregations on raw rows read from Solr, your goal is to minimize the number of rows read from Solr and achieve the best read performance of those rows.
- **Push down into Solr.** Push down aggregation queries into Solr, returning a smaller set of aggregated rows to Spark.
- **Use views.** Use views that send queries directly to Solr, using options supported by the spark-solr library.

Examples of queries written to optimize performance

Here we provide examples of how to get the best performance out of the Fusion SQL service.

The examples show how to write queries to optimize performance, by ensuring that Fusion Spark is not performing unnecessary computations. This lets you do complex operations in Spark.

Pulling back rows is a rate-limiting step when executing aggregations. These examples ensure that the Fusion Spark query planner pulls back the smallest number of rows.

For general SQL information, see the Spark SQL Language manual (<https://spark.apache.org/docs/latest/sql-programming-guide.html>).

Example 1: Only request fields that have DocValues enabled

Optimal read performance is achieved by only requesting fields that have `DocValues` enabled, because these can be pulled through the `/export` handler.

It goes without saying that you should only request the fields you need for each query. Spark's query planner pushes the field list down into the Fusion SQL service, which translates it into an `fl` parameter to Solr.

For example, if you need `movie_id` and `title` from the `movies` table, do this:

```
select movie_id, title from movies
```

Do not do this:

```
select * from movies
```

Example 2: Use WHERE clause filtering to reduce the number of rows

Use WHERE clause criteria, including full Solr queries, to do as much filtering in Solr as possible to reduce the number of rows.

Spark's SQL query planner pushes down simple filter criteria into the Fusion SQL service, which translates SQL filters into Solr filter query (`fq`) parameters. For example, if you want to query using SQL:

```
select user_id, movie_id, rating from ratings where rating = 4
```

Then behind the scenes, the Fusion SQL service transforms this query into the following Solr query:

```
q=*&qt=/export&sort=id+asc&collection=ratings&fl=user_id,movie_id,rating&fq=rating:4
```

Notice that the WHERE clause was translated into an `fq` parameter and the specific fields needed for the query are sent along in the `fl` parameter. Also notice that the Fusion SQL service will use the `/export` handler if all of the fields requested have `DocValues` enabled. This makes a big difference in performance.

You can also perform full-text queries using the WHERE clause. For example, the following SQL performs a full-text search for the term "love" in the `plot_txt_en` field of the `movies` table.

```
select movie_id,title from movies where plot_txt_en='love'
```

This works because the Fusion SQL service uses the Solr schema to determine that the `plot_txt_en` field is a text field, and so Solr assumes the user wants to perform a full-text query. Keep in mind that this full-text search feature does not work with cached tables, because those are held in Spark memory and WHERE criteria are not sent to Solr when tables are cached.

The Fusion SQL service also provides the `_query` user-defined function (UDF) to execute any valid Solr query; for example:

```
select movie_id,title from movies where _query_("+plot_txt_en:dogs - plot_txt_en:cats")
```

You should use the `_query` UDF approach when you need to pass a complex query to Solr.

Here is another example where we push down a subquery into Solr directly to apply a complex full-text search (in this case, a `geofilt` geospatial query) and then join the resulting rows with a different table:

```
SELECT geo.place_name, count(*) as cnt FROM users u INNER JOIN (select place_name,zip_code from zipcodes where _query_('{!geofilt sfield=geo_location pt=44.9609,-93.2642 d=50}')) as geo ON geo.zip_code = u.zip_code WHERE u.gender='F' GROUP BY geo.place_name
```

Example 3: Apply LIMIT clauses on pushdown queries

Let us say you have a table of movies and ratings and want to join the `title` with the `ratings` table to get the top 100 movies with the most ratings, using something like this:

```
select m.title, count(*) as num_ratings from movies m, ratings r where m.movie_id = r.movie_id group by m.title order by num_ratings desc limit 100
```

Given that LIMIT clause, you might think this query will be very fast because you are only asking for 100 rows. However, if the ratings table is big (as is typically the case), then Spark has to read all of the ratings from Solr before joining and aggregating. The better approach is to push the LIMIT down into Solr, and then join from the smaller result set.

```
select m.title, solr.num_ratings from movies m inner join (select movie_id, count(*) as num_ratings from ratings group by movie_id order by num_ratings desc limit 100) as popular_movies on m.movie_id = popular_movies.movie_id order by num_ratings desc
```

Notice that the LIMIT is now on the subquery that gets run directly by Solr, using the Fusion SQL service pushdown. You should use this strategy whether you are aggregating in Solr or just retrieving raw rows. For example:

```
SELECT e.id, e.name, solr.* FROM ecommerce e INNER JOIN (select timestamp_tdt, query_s, filters_s, type_s, user_id_s, doc_id_s from ecommerce_signals order by timestamp_tdt desc limit 50000) as signals ON signals.doc_id_s = e.id
```

The subquery pulls the last 50,000 signals from Solr before joining the signals with the `ecommerce` table.

Example 4: Tune read options of the underlying data asset in Fusion

When you need to return fields that do not support `DocValues` from Solr, consider tuning the read options of the underlying data asset in Fusion.

Behind the scenes, the Fusion SQL service uses parallel queries to each shard and cursorMark to page through all documents in each shard. This approach, while efficient, is not as fast as reading from the `/export` handler. For example, our `ecommerce` table contains text fields that cannot be exported using `DocValues`, so we can tune the read performance using the Catalog API:

```
curl -X PUT -H "Content-type:application/json" --data-binary '{ "name": "ecommerce", "assetType": "table", "projectId": "fusion", "description": "ecommerce demo data", "tags": ["fusion"], "format": "solr", "cacheOnLoad": false, "options" : [ "collection -> ecommerce", "splits_per_shard -> 4", "solr.params -> sort=id asc", "exclude_fields -> _lw*,_raw_content_", "rows -> 10000" ] }' https://FUSION_HOST:6764/api/catalog/fusion/assets/ecommerce
```

Notice that, in this case, we are reading all fields except those matching the patterns in the `exclude_fields` option. We have also increased the number of rows read per paging request to 10000 and we want 4 `splits_per_shard`, which means that we will use 4 tasks per shard to read data across all replicas of that shard.

For more information, read about tuning Spark-Solr read performance (<https://github.com/lucidworks/spark-solr/blob/master/README.adoc#configuration-and-tuning>).

The key take-away here is that, if your queries need fields that cannot be exported, then you will need to do some manual tuning of the options on the data asset to get optimal performance. Here is another example where we want a geospatial field that does not support `DocValues` from Solr:

```
curl -u USERNAME:PASSWORD -XPOST -H "Content-type:application/json"
"https://FUSION_HOST:6764/api/catalog/fusion/assets" --data-binary @<(cat <<EOF {
"name": "minn_zipcodes", "assetType": "table", "projectId": "fusion", "format":
"solr", "description": "zips around minn", "options": [ "collection -> zipcodes",
"fields ->
zip_code,place_name,state,county,geo_point,geo_location,geo_location_rpt", "query
-> {!geofilt sfield=geo_location pt=44.9609,-93.2642 d=50}", "solr.params ->
sort=id asc" ] } EOF )
```

In this example, we want to return the `geo_location_rpt` field, for example to generate a heatmap, so we need to define a custom data asset in the catalog.

Example 5: Aggregate in Solr using the underlying faceting engine

Solr provides a number of basic aggregation capabilities, such as `count`, `min`, `max`, `avg`, and `sum`. To reduce the number of raw rows that are returned from Solr to Spark, the Fusion SQL service leverages Solr aggregation when possible. The smaller the number of rows returned from Solr lets Spark perform better query optimization, such as doing a broadcast of a small table across all partitions of a large table (a hash join).

Here is an example where we push a group count operation (a facet basically) down into Solr using a subquery.

```
SELECT m.title as title, topRatedMovies.aggCount as aggCount FROM movies m
INNER JOIN (SELECT movie_id, COUNT(*) as aggCount FROM ratings WHERE rating >= 4
GROUP BY movie_id ORDER BY aggCount desc LIMIT 10) as topRatedMovies ON
topRatedMovies.movie_id = m.movie_id ORDER BY aggCount DESC
```

Solr returns aggregated rows by `movie_id`, and then we leverage Spark to perform the join between `movies` and the aggregated results of the subquery, which it can do quickly using a hash join with a broadcast.

Example 6: Aggregate in Solr using streaming expressions

To determine whether an aggregation can be pushed down into Solr for better performance, the Fusion SQL service analyzes the logical plan in Spark.

Alternatively, you can write a streaming expression and then expose that as a view in the Fusion SQL service. For example, the following streaming expression joins ecommerce products and signals:

```
select( hashJoin( search(ecommerce, q="*:*", fl="id,name", sort="id asc",
qt="/export", partitionKeys="id"), hashed=facet(ecommerce_signals, q="*:*",
buckets="doc_id_s", bucketSizeLimit=10000, bucketSorts="count(*) desc",
count(*)), on="id=doc_id_s"), name as product_name, count(*) as click_count, id
as product_id )
```

This streaming expression performs a hash join between the `ecommerce` table and the results of a facet expression on the signals collection. We also use the `SELECT` expression decorator to return human-friendly field names.

For more information, read about *how to write streaming expressions*.

After creating and testing the streaming expression in Solr, you must JSON encode the expression and then create a data asset in the Fusion catalog. For example:

```
curl -XPOST -H "Content-Type:application/json" --data-binary '{ "name":
"ecomm_popular_docs", "assetType": "table", "projectId": "fusion", "description":
"Join product name with facet counts of docs in signals", "tags": ["ecommerce"],
"format": "solr", "cacheOnLoad": true, "options": ["collection -> ecommerce",
"expr -> select(hashJoin(search(ecommerce,q=\"*:*\",fl=\"id,name\",sort=\"id
asc\",qt=\"\"/export\",partitionKeys=\"id\"),hashed=facet(ecommerce_signals,q=\"*:
*\",buckets=\"doc_id_s\",bucketSizeLimit=10000,bucketSorts=\"count(*)
desc\",count(*)),on=\"id=doc_id_s\"),name as product_name,count(*) as
click_count,id as product_id)\"}]}'
https://FUSION_HOST:6764/api/catalog/fusion/assets
```

Here is another example of a streaming expression that leverages the underlying faceting engine's support for computing aggregations beyond just a count:

```
select( facet( ratings, q="*:*", buckets="rating", bucketSorts="count(*) desc",
bucketSizeLimit=100, count(*), sum(rating), min(rating), max(rating), avg(rating)
), rating, count(*) as the_count, sum(rating) as the_sum, min(rating) as the_min,
max(rating) as the_max, avg(rating) as the_avg )
```

Example 7: Use sampling

When doing exploratory analysis on a large table, use the `rand` function to bring back a small random sample of data from Solr, for example:

```
select user_id as user, age as _age_ from users where rand() < 0.1 AND gender='F'
```

The `rand` function returns a random number between 0 and 1, so in the previous example, we are requesting a sample that is roughly 10% of the total number of rows. Behind the scenes, Fusion SQL translates the `rand` UDF into a random streaming expression to draw a sample.

Example 8: Cache results in Spark

If you plan to perform additional queries against results, cache results in Spark.

Catalog assets support the `cacheOnLoad` attribute, which caches the results of the query in Spark (memory with spill to disk). You can also request the results for any query sent to the Catalog API to be cached using the `cacheResultsAs` parameter:

```
curl -XPOST -H "Content-Type:application/json" -d '{ "sql":"SELECT u.user_id as
user_id, age, gender, occupation, place_name, county, state, zip_code,
geo_location_rpt, title, movie_id, rating, rating_timestamp FROM minn_users u
INNER JOIN movie_ratings m ON u.user_id = m.user_id", "cacheResultsAs":
"ratings_by_minnc_users" }' "https://FUSION_HOST:6764/api/catalog/fusion/query"
```

Be careful! If cached, updates to the underlying data source, most likely Solr, will no longer be visible. To trigger Spark to recompute a cached view by going back to the underlying store, you can use the following SQL command:

```
curl -XPOST -H "Content-Type:application/json" -d '{"sql":"refresh table table-name"}' \ "https://FUSION_HOST:6764/api/catalog/fusion/query"
```

If a table is not cached, you can cache it using:

```
curl -XPOST -H "Content-Type:application/json" -d '{"sql":"cache table table-name"}' \ "https://FUSION_HOST:6764/api/catalog/fusion/query"
```

Or uncache it:

```
curl -XPOST -H "Content-Type:application/json" -d '{"sql":"uncache table table-name"}' \ "https://FUSION_HOST:6764/api/catalog/fusion/query"
```

User defined functions (UDFs)

You can use SQL and Spark User Defined Functions (UDF) to clean and/or transform data from Solr.

For example, Lucidworks Search Hub signals use complex field names generated by Snowplow. The following data asset definition uses SQL to make the data a bit more user friendly as it comes out of Solr:

```
{ "name": "shub_signals", "assetType": "table", "projectId": "fusion",
"description": "SearchHub signals", "tags": ["shub"], "format": "solr",
"cacheOnLoad": false, "options": ["collection -> shub_signals", "solr.params ->
sort=id asc", "fields ->
timestamp_tdt,type_s,params.useragent_family_s,params.useragent_os_family_s,param
s.tz_s,params.totalResults_s,params.lang_s,params.useragent_type_name_s,params.te
rms_s,params.query_unique_id,params.useragent_v,params.doc_0,params.doc_1,params.
doc_2,params.facet_ranges_publishedOnDate_before_d,params.uid_s,params.refr_s,par
ams.useragent_category_s,params.sid_s,ip_sha_s,params.vid_s,params.page_s,params.
fp_s"], "sql": "SELECT timestamp_tdt as timestamp, type_s as signal_type,
`params.useragent_family_s` as ua_family, `params.useragent_os_family_s` as
ua_os, `params.tz_s` as tz, cast(`params.totalResults_s` as int) as num_found,
`params.lang_s` as lang, `params.useragent_type_name_s` as ua_type,
`params.terms_s` as query_terms, `params.query_unique_id` as query_id,
`params.useragent_v` as ua_vers, `params.doc_0` as doc0, `params.doc_1` as doc1,
`params.doc_2` as doc2, `params.facet_ranges_publishedOnDate_before_d` as
pubdate_range, `params.uid_s` as user_id, `params.refr_s` as referrer,
`params.useragent_category_s` as ua_category, `params.sid_s` as session_id,
`ip_sha_s` as ip, cast(`params.vid_s` as int) as num_visits, `params.page_s` as
page_name, `params.fp_s` as fingerprint FROM shub_signals" }
```

Given how Spark uses lazy evaluation of DataFrame transformations, there is little additional overhead beyond the cost of reading from Solr to execute a SQL statement on top of the raw results from Solr.

Beyond simple field renaming, you can also leverage 100s of built-in UDFs to enrich and/or transform fields. See:
[https://spark.apache.org/docs/1.6.3/api/scala/index.html#org.apache.spark.sql.functions\\$](https://spark.apache.org/docs/1.6.3/api/scala/index.html#org.apache.spark.sql.functions$)
([https://spark.apache.org/docs/1.6.3/api/scala/index.html#org.apache.spark.sql.functions\\$](https://spark.apache.org/docs/1.6.3/api/scala/index.html#org.apache.spark.sql.functions$))

In the example above, the `cast` function casts a string field coming from Solr as an `int`:

```
cast(`params.vid_s` as int)
```

Keep in mind that Solr is a much more flexible data engine than is typically handled by BI visualization tools like Tableau. Consequently, the Fusion Catalog API lets you apply more structure to less structured data in Solr. Here is an example of using a UDF to aggregate by day (similar to Solr's round down operator):

```
select count(*) as num_per_day, date_format(rating_timestamp,"yyyy-MM-dd") as  
date_fmt from ratings group by date_format(rating_timestamp,"yyyy-MM-dd") order  
by num_per_day desc LIMIT 10
```

Fusion SQL Administration

Most organizations that deploy Fusion also have SQL-compliant business intelligence (BI) or dashboarding tools to facilitate self-service analytics.

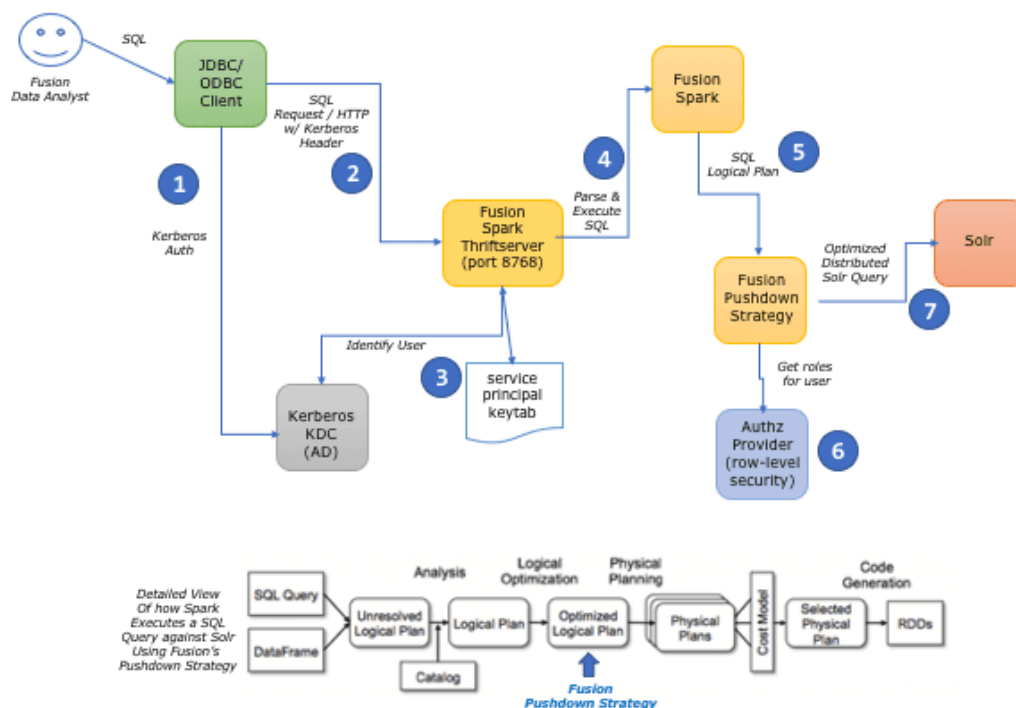
The Fusion SQL service:

- Lets organizations leverage their investments in BI tools by using JDBC and SQL to analyze data managed by Fusion. For example, Tableau is popular data analytics tool that connects to Fusion SQL using JDBC to enable self-service analytics.
- Helps business users access important data sets in Fusion without having to know how to query Solr.

In addition to the specified *System Requirements*, Fusion on Windows requires Visual C++ Redistributable for Visual Studio 2015 (<https://www.microsoft.com/en-us/download/details.aspx?id=48145>) to start the SQL service successfully.

Fusion SQL architecture

The following diagram depicts a common Fusion SQL service deployment scenario using the Kerberos network authentication protocol for single sign-on. Integration with Kerberos is optional. By default, the Fusion SQL service uses Fusion security for authentication and authorization.



The numbered steps in the diagram are:

1. The JDBC/ODBC client application (for example, TIBCO Spotfire or Tableau) uses Kerberos to authenticate a Fusion data analyst.
2. After authentication, the JDBC/ODBC client application sends the user's SQL query to the Fusion SQL Thrift Server over HTTP.
3. The SQL Thrift Server uses the keytab of the Kerberos service principal to validate the incoming user identity.
The Fusion SQL Thrift Server is a Spark application with a specific number of CPU cores and memory allocated from the pool of Spark resources. You can scale out the number of Spark worker nodes to increase available memory and CPU resources to the Fusion SQL service.
4. The Thrift Server sends the query to Spark to be parsed into a logical plan.
5. During the query planning stage, Spark sends the logical plan to *Fusion's pushdown strategy* component.
6. During pushdown analysis, Fusion calls out to the registered AuthZ FilterProvider implementation to get a filter query to perform row-level filtering for the Kerberos-authenticated user.
By default, there is no row-level security provider but users can install their own implementation using the Fusion SQL service API.
7. Spark executes a distributed Solr query to return documents that satisfy the SQL query criteria and row-level security filter. To leverage the distributed nature of Spark and Solr, Fusion SQL sends a query to all replicas for each shard in a Solr collection.
Consequently, you can scale out SQL query performance by adding more Spark and/or Solr resources to your cluster.

Fusion pushdown strategy

The pushdown strategy analyzes the query plan to determine if there is an optimal Solr query or streaming expression that can push down aggregations into Solr to improve performance and scalability. For example, the following SQL query can be translated into a Solr facet query by the Fusion pushdown strategy:

```
select count(1) as the_count, movie_id from ratings group by movie_id
```

The basic idea behind Fusion's pushdown strategy is it is much faster to let Solr facets perform basic aggregations than it is to export raw documents from Solr and have Spark perform the aggregation. If an optimal pushdown query is not possible, then Spark pulls raw documents from Solr, and then performs any joins or aggregations needed in Spark. Put simply, the Fusion SQL service tries to translate SQL queries into optimized Solr queries. But failing that, the service simply reads all matching documents for a query into Spark, and then performs the SQL execution logic across the Spark cluster.

Which collections are registered

By default, all Fusion collections except system collections are registered in the Fusion SQL service so you can query them without any additional setup. However, empty collections cannot be queried, or even described from SQL, so empty collections will not show up in the Fusion SQL service until they have data. In addition, any fields with a dot in the name are ignored when tables are auto-registered. You can use the Catalog API to alias fields with dots in the names to include these fields.

If you add data to a previously empty collection, then you can execute either of the following SQL commands to ensure that the data gets added as a table:

```
show tables show tables in `default`
```

The Fusion SQL service checks previously empty collections every minute and automatically registers recently populated collections as a table.

You can describe any table using:

```
describe table-name
```

See the `movielens` lab in the Fusion Spark Bootcamp (<https://github.com/lucidworks/fusion-spark-bootcamp>) for a complete example of working with the Fusion Catalog API and Fusion SQL service. Also read about the *Catalog API*.

Hive configuration

Behind the scenes, the Fusion SQL service is based on Hive. Use the `hive-site.xml` file in `/opt/fusion/4.1.x/conf/` (on Unix) or `C:\lucidworks\fusion\4.1.x\conf\` (on Windows) to configure Hive settings.

If you change `hive-site.xml`, you must restart the Fusion SQL service with `./sql restart` (on Unix) or `sql.cmd restart` (on Windows).

Approaches for optimizing the performance of queries

In general terms, the Fusion SQL service supports three approaches for optimizing the performance of queries:

- **Join or aggregate in Spark.** Read a set of raw rows from Solr and join or aggregate the rows in Spark.
For queries that rely on Spark performing joins and aggregations on raw rows read from Solr, your goal is to minimize the number of rows read from Solr and achieve the best read performance of those rows.
- **Push down into Solr.** Push down aggregation queries into Solr, returning a smaller set of aggregated rows to Spark.

- **Use views.** Use views that send queries directly to Solr, using options supported by the spark-solr library.

Examples of queries written to optimize performance

Here we provide examples of how to get the best performance out of the Fusion SQL service.

The examples show how to write queries to optimize performance, by ensuring that Fusion Spark is not performing unnecessary computations. This lets you do complex operations in Spark.

Pulling back rows is a rate-limiting step when executing aggregations. These examples ensure that the Fusion Spark query planner pulls back the smallest number of rows.

For general SQL information, see the Spark SQL Language manual (<https://spark.apache.org/docs/latest/sql-programming-guide.html>).

Example 1: Only request fields that have DocValues enabled

Optimal read performance is achieved by only requesting fields that have `DocValues` enabled, because these can be pulled through the `/export` handler.

It goes without saying that you should only request the fields you need for each query. Spark's query planner pushes the field list down into the Fusion SQL service, which translates it into an `fl` parameter to Solr.

For example, if you need `movie_id` and `title` from the `movies` table, do this:

```
select movie_id, title from movies
```

Do not do this:

```
select * from movies
```

Example 2: Use WHERE clause filtering to reduce the number of rows

Use WHERE clause criteria, including full Solr queries, to do as much filtering in Solr as possible to reduce the number of rows.

Spark's SQL query planner pushes down simple filter criteria into the Fusion SQL service, which translates SQL filters into Solr filter query (`fq`) parameters. For example, if you want to query using SQL:

```
select user_id, movie_id, rating from ratings where rating = 4
```

Then behind the scenes, the Fusion SQL service transforms this query into the following Solr query:

```
q=*:*&qt=/export&sort=id+asc&collection=ratings&fl=user_id,movie_id,rating&fq=rating:4
```

Notice that the WHERE clause was translated into an `fq` parameter and the specific fields needed for the query are sent along in the `fl` parameter. Also notice that the Fusion SQL service will use the `/export` handler if all of the fields requested have `DocValues` enabled. This makes a big difference in performance.

You can also perform full-text queries using the WHERE clause. For example, the following SQL performs a full-text search for the term "love" in the `plot_txt_en` field of the `movies` table.

```
select movie_id,title from movies where plot_txt_en='love'
```

This works because the Fusion SQL service uses the Solr schema to determine that the `plot_txt_en` field is a text field, and so Solr assumes the user wants to perform a full-text query. Keep in mind that this full-text search feature does not work with cached tables, because those are held in Spark memory and WHERE criteria are not sent to Solr when tables are cached.

The Fusion SQL service also provides the `_query_` user-defined function (UDF) to execute any valid Solr query; for example:

```
select movie_id,title from movies where _query_("+plot_txt_en:dogs -
plot_txt_en:cats")
```

You should use the `_query_` UDF approach when you need to pass a complex query to Solr.

Here is another example where we push down a subquery into Solr directly to apply a complex full-text search (in this case, a `geofilt` geospatial query) and then join the resulting rows with a different table:

```
SELECT geo.place_name, count(*) as cnt FROM users u INNER JOIN (select
place_name,zip_code from zipcodes where _query_('!geofilt sfield=geo_location
pt=44.9609,-93.2642 d=50}')) as geo ON geo.zip_code = u.zip_code WHERE
u.gender='F' GROUP BY geo.place_name
```

Example 3: Apply LIMIT clauses on pushdown queries

Let us say you have a table of movies and ratings and want to join the `title` with the `ratings` table to get the top 100 movies with the most ratings, using something like this:

```
select m.title, count(*) as num_ratings from movies m, ratings r where m.movie_id
= r.movie_id group by m.title order by num_ratings desc limit 100
```

Given that LIMIT clause, you might think this query will be very fast because you are only asking for 100 rows. However, if the ratings table is big (as is typically the case), then Spark has to read all of the ratings from Solr before joining and aggregating. The better approach is to push the LIMIT down into Solr, and then join from the smaller result set.

```
select m.title, solr.num_ratings from movies m inner join (select movie_id,
count(*) as num_ratings from ratings group by movie_id order by num_ratings desc
limit 100) as popular_movies on m.movie_id = popular_movies.movie_id order by
num_ratings desc
```

Notice that the LIMIT is now on the subquery that gets run directly by Solr, using the Fusion SQL service pushdown. You should use this strategy whether you are aggregating in Solr or just retrieving raw rows. For example:

```
SELECT e.id, e.name, solr.* FROM ecommerce e INNER JOIN (select timestamp_tdt,
query_s, filters_s, type_s, user_id_s, doc_id_s from ecommerce_signals order by
timestamp_tdt desc limit 50000) as signals ON signals.doc_id_s = e.id
```

The subquery pulls the last 50,000 signals from Solr before joining the signals with the `ecommerce` table.

Example 4: Tune read options of the underlying data asset in Fusion

When you need to return fields that do not support `DocValues` from Solr, consider tuning the read options of the underlying data asset in Fusion.

Behind the scenes, the Fusion SQL service uses parallel queries to each shard and cursorMark to page through all documents in each shard. This approach, while efficient, is not as fast as reading from the `/export` handler. For example, our `ecommerce` table contains text fields that cannot be exported using `DocValues`, so we can tune the read performance using the Catalog API:

```
curl -X PUT -H "Content-type:application/json" --data-binary '{ "name":
"ecommerce", "assetType": "table", "projectId": "fusion", "description":
"ecommerce demo data", "tags": ["fusion"], "format": "solr", "cacheOnLoad":
false, "options" : [ "collection -> ecommerce", "splits_per_shard -> 4",
"solr.params -> sort=id asc", "exclude_fields -> _lw*,_raw_content_", "rows ->
10000" ] }' https://FUSION_HOST:6764/api/catalog/fusion/assets/ecommerce
```

Notice that, in this case, we are reading all fields except those matching the patterns in the `exclude_fields` option. We have also increased the number of rows read per paging request to 10000 and we want 4 `splits_per_shard`, which means that we will use 4 tasks per shard to read data across all replicas of that shard.

For more information, read about tuning Spark-Solr read performance (<https://github.com/lucidworks/spark-solr/blob/master/README.adoc#configuration-and-tuning>).

The key take-away here is that, if your queries need fields that cannot be exported, then you will need to do some manual tuning of the options on the data asset to get optimal performance. Here is another example where we want a geospatial field that does not support `DocValues` from Solr:

```
curl -u USERNAME:PASSWORD -XPOST -H "Content-type:application/json"
"https://FUSION_HOST:6764/api/catalog/fusion/assets" --data-binary @<(cat <<EOF {
"name": "minn_zipcodes", "assetType": "table", "projectId": "fusion", "format":
"solr", "description": "zips around minn", "options": [ "collection -> zipcodes",
"fields ->
zip_code,place_name,state,county,geo_point,geo_location,geo_location_rpt", "query
-> {!geofilt sfield=geo_location pt=44.9609,-93.2642 d=50}", "solr.params ->
sort=id asc" ] } EOF )
```

In this example, we want to return the `geo_location_rpt` field, for example to generate a heatmap, so we need to define a custom data asset in the catalog.

Example 5: Aggregate in Solr using the underlying faceting engine

Solr provides a number of basic aggregation capabilities, such as `count`, `min`, `max`, `avg`, and `sum`. To reduce the number of raw rows that are returned from Solr to Spark, the Fusion SQL service leverages Solr aggregation when possible. The smaller the number of rows returned from Solr lets Spark perform better query optimization, such as doing a broadcast of a small table across all partitions of a large table (a hash join).

Here is an example where we push a group count operation (a facet basically) down into Solr using a subquery.

```
SELECT m.title as title, topRatedMovies.aggCount as aggCount FROM movies m
INNER JOIN (SELECT movie_id, COUNT(*) as aggCount FROM ratings WHERE rating >= 4
GROUP BY movie_id ORDER BY aggCount desc LIMIT 10) as topRatedMovies ON
topRatedMovies.movie_id = m.movie_id ORDER BY aggCount DESC
```

Solr returns aggregated rows by `movie_id`, and then we leverage Spark to perform the join between `movies` and the aggregated results of the subquery, which it can do quickly using a hash join with a broadcast.

Example 6: Aggregate in Solr using streaming expressions

To determine whether an aggregation can be pushed down into Solr for better performance, the Fusion SQL service analyzes the logical plan in Spark.

Alternatively, you can write a streaming expression and then expose that as a view in the Fusion SQL service. For example, the following streaming expression joins ecommerce products and signals:

```
select( hashJoin( search(ecommerce, q="*:*", fl="id,name", sort="id asc",
qt="/export", partitionKeys="id"), hashed=facet(ecommerce_signals, q="*:*",
buckets="doc_id_s", bucketSizeLimit=10000, bucketSorts="count(*) desc",
count(*)), on="id=doc_id_s"), name as product_name, count(*) as click_count, id
as product_id )
```

This streaming expression performs a hash join between the `ecommerce` table and the results of a facet expression on the signals collection. We also use the `SELECT` expression decorator to return human-friendly field names.

For more information, read about *how to write streaming expressions*.

After creating and testing the streaming expression in Solr, you must JSON encode the expression and then create a data asset in the Fusion catalog. For example:

```
curl -XPOST -H "Content-Type:application/json" --data-binary '{ "name":
"ecomm_popular_docs", "assetType": "table", "projectId": "fusion", "description":
"Join product name with facet counts of docs in signals", "tags": ["ecommerce"],
"format": "solr", "cacheOnLoad": true, "options": ["collection -> ecommerce",
"expr -> select(hashJoin(search(ecommerce,q=\"*:*\",fl=\"id,name\",sort=\"id
asc\",qt=\"\"/export\",partitionKeys=\"id\"),hashed=facet(ecommerce_signals,q=\"*:*\",
buckets=\"doc_id_s\",bucketSizeLimit=10000,bucketSorts=\"count(*)
desc\",count(*)),on=\"id=doc_id_s\"),name as product_name,count(*) as
click_count,id as product_id)"]}'
https://FUSION_HOST:6764/api/catalog/fusion/assets
```

Here is another example of a streaming expression that leverages the underlying faceting engine's support for computing aggregations beyond just a count:

```
select( facet( ratings, q="*:*", buckets="rating", bucketSorts="count(*) desc",
bucketSizeLimit=100, count(*), sum(rating), min(rating), max(rating), avg(rating)
), rating, count(*) as the_count, sum(rating) as the_sum, min(rating) as the_min,
max(rating) as the_max, avg(rating) as the_avg )
```

Example 7: Use sampling

When doing exploratory analysis on a large table, use the `rand` function to bring back a small random sample of data from Solr, for example:

```
select user_id as user, age as _age_ from users where rand() < 0.1 AND gender='F'
```

The `rand` function returns a random number between 0 and 1, so in the previous example, we are requesting a sample that is roughly 10% of the total number of rows. Behind the scenes, Fusion SQL translates the `rand` UDF into a random streaming expression to draw a sample.

Example 8: Cache results in Spark

If you plan to perform additional queries against results, cache results in Spark.

Catalog assets support the `cacheOnLoad` attribute, which caches the results of the query in Spark (memory with spill to disk). You can also request the results for any query sent to the Catalog API to be cached using the `cacheResultsAs` parameter:

```
curl -XPOST -H "Content-Type:application/json" -d '{ "sql":"SELECT u.user_id as
user_id, age, gender, occupation, place_name, county, state, zip_code,
geo_location_rpt, title, movie_id, rating, rating_timestamp FROM minn_users u
INNER JOIN movie_ratings m ON u.user_id = m.user_id", "cacheResultsAs":
"ratings_by_minn_users" }' "https://FUSION_HOST:6764/api/catalog/fusion/query"
```

Be careful! If cached, updates to the underlying data source, most likely Solr, will no longer be visible. To trigger Spark to recompute a cached view by going back to the underlying store, you can use the following SQL command:

```
curl -XPOST -H "Content-Type:application/json" -d '{"sql":"refresh table table-
name"}' \ "https://FUSION_HOST:6764/api/catalog/fusion/query"
```

If a table is not cached, you can cache it using:

```
curl -XPOST -H "Content-Type:application/json" -d '{"sql":"cache table table-
name"}' \ "https://FUSION_HOST:6764/api/catalog/fusion/query"
```

Or uncache it:


```
curl -XPOST -H "Content-Type:application/json" -d '{"sql":"uncache table table-name"}' \ "https://FUSION_HOST:6764/api/catalog/fusion/query"
```

User defined functions (UDFs)

You can use SQL and Spark User Defined Functions (UDF) to clean and/or transform data from Solr.

For example, Lucidworks Search Hub signals use complex field names generated by Snowplow. The following data asset definition uses SQL to make the data a bit more user friendly as it comes out of Solr:

```
{ "name": "shub_signals", "assetType": "table", "projectId": "fusion",
  "description": "SearchHub signals", "tags": ["shub"], "format": "solr",
  "cacheOnLoad": false, "options": ["collection -> shub_signals", "solr.params ->
  sort=id asc", "fields ->
  timestamp_tdt,type_s,params.useragent_family_s,params.useragent_os_family_s,param
  s.tz_s,params.totalResults_s,params.lang_s,params.useragent_type_name_s,params.te
  rms_s,params.query_unique_id,params.useragent_v,params.doc_0,params.doc_1,params.
  doc_2,params.facet_ranges_publishedOnDate_before_d,params.uid_s,params.refr_s,par
  ams.useragent_category_s,params.sid_s,ip_sha_s,params.vid_s,params.page_s,params.
  fp_s"], "sql": "SELECT timestamp_tdt as timestamp, type_s as signal_type,
  `params.useragent_family_s` as ua_family, `params.useragent_os_family_s` as
  ua_os, `params.tz_s` as tz, cast(`params.totalResults_s` as int) as num_found,
  `params.lang_s` as lang, `params.useragent_type_name_s` as ua_type,
  `params.terms_s` as query_terms, `params.query_unique_id` as query_id,
  `params.useragent_v` as ua_vers, `params.doc_0` as doc0, `params.doc_1` as doc1,
  `params.doc_2` as doc2, `params.facet_ranges_publishedOnDate_before_d` as
  pubdate_range, `params.uid_s` as user_id, `params.refr_s` as referrer,
  `params.useragent_category_s` as ua_category, `params.sid_s` as session_id,
  `ip_sha_s` as ip, cast(`params.vid_s` as int) as num_visits, `params.page_s` as
  page_name, `params.fp_s` as fingerprint FROM shub_signals" }
```

Given how Spark uses lazy evaluation of DataFrame transformations, there is little additional overhead beyond the cost of reading from Solr to execute a SQL statement on top of the raw results from Solr.

Beyond simple field renaming, you can also leverage 100s of built-in UDFs to enrich and/or transform fields. See:

[https://spark.apache.org/docs/1.6.3/api/scala/index.html#org.apache.spark.sql.functions\\$](https://spark.apache.org/docs/1.6.3/api/scala/index.html#org.apache.spark.sql.functions$)

([https://spark.apache.org/docs/1.6.3/api/scala/index.html#org.apache.spark.sql.functions\\$](https://spark.apache.org/docs/1.6.3/api/scala/index.html#org.apache.spark.sql.functions$))

In the example above, the `cast` function casts a string field coming from Solr as an `int`:

```
cast(`params.vid_s` as int)
```

Keep in mind that Solr is a much more flexible data engine than is typically handled by BI visualization tools like Tableau. Consequently, the Fusion Catalog API lets you apply more structure to less structured data in Solr. Here is an example of using a UDF to aggregate by day (similar to Solr's round down operator):

```
select count(*) as num_per_day, date_format(rating_timestamp,"yyyy-MM-dd") as  
date_fmt from ratings group by date_format(rating_timestamp,"yyyy-MM-dd") order  
by num_per_day desc LIMIT 10
```

Licensing

Fusion Server requires a valid license. Depending on the details of your contract, your license may also enable optional *connectors* or *Fusion AI*.

Fusion Server provides a license management UI and a license API for installing and managing licenses. When you upload a license, Fusion Server stores it in ZooKeeper, so you only need to upload it to one node per cluster.

The Fusion UI notifies you when your trial license is about to expire. When your license has expired, Fusion Server accepts no configuration changes until you upload a valid license.

Visit our company page to try Fusion (<https://lucidworks.com/try>).

Object Explorer

Object Explorer lets you easily find and manage Fusion *objects* within a specific app and across multiple apps in an entire Fusion deployment.

Fusion apps provide tailored search functionality to specific groups of users.

An app is a named set of linked *objects*, including collections, datasources, index and query pipelines, index and query profiles, parsers, and more. Using roles and security realms, you can define security on a per-app basis.

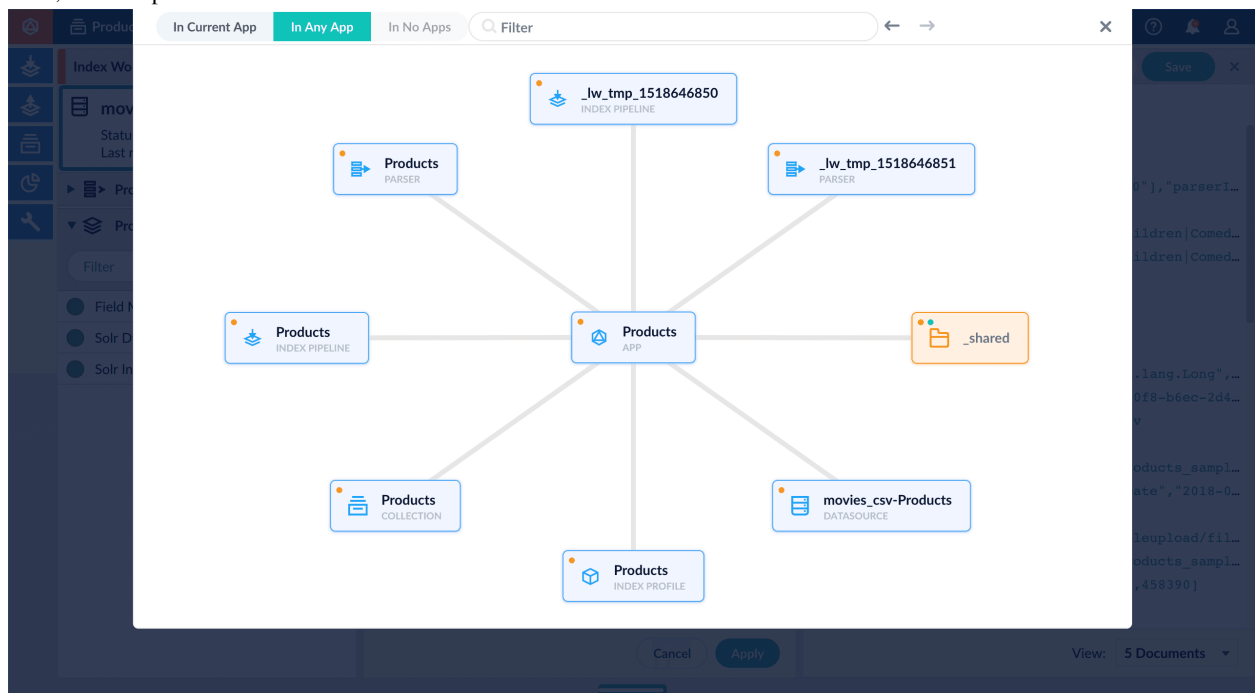
An object is something that has a specific function in a Fusion app, such as an index pipeline or Spark job. It is often something that you configure. An app is an object. Several object types are *groups* that group other objects. For more information about object types, see *Object types*.

What you can do in Object Explorer

In Object Explorer, you can explore objects.

Which objects you can explore depends on the *permissions* that your security realm and user definition grant you. As the user `admin`, you can explore all objects supported by Object Explorer.

- **Determine which objects are in which apps.** You can determine which objects are in the current app, in any app, in no apps, and in any app or no apps. When you navigate to an object, you can display the apps that the object is in.
- **View and navigate between related objects.** In Object Explorer, related objects are connected by lines. Clicking an object moves it to the center of the Object Explorer window and displays related objects around it.
- **View related objects.** When you select an object, Object Explorer centers and displays related objects around it, connected by lines, for example:



- **Perform some management tasks.** You can perform some management tasks inside Object Explorer. For example, you can schedule jobs and tasks from Object Explorer. For other management tasks, Object Explorer takes you to where you need to be.
- **Add and remove objects (Advanced feature)** - When you create an app, it consists of objects. When you create additional apps, Fusion can *share* some objects across apps. Shared objects are in the `_shared` system group, and they have multiple dots on them, one for each app they are in.

In some cases, you might want to remove an object from an app (undo sharing) or add an object to an app. For example, after configuring a parser in one app, you could share the parser with other apps, *without having to share things you do not want to share*, for example, the collection.

Object types

You can explore these types of objects with Object Explorer:

 App	 Spark job or task	 Collection
 Datasource	 Index pipeline	 Parser
 Query pipeline	 A generic icon that is currently used for query profiles, index profiles, blobs, and experiments	 Recommendations group
 System group	 User-defined group or Fusion-defined group that is not a system group	

Spark Operations

Apache Spark (<http://spark.apache.org/>) is an open-source cluster-computing framework that serves as 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 improves on previous MapReduce implementations by using resilient distributed datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner.

Spark in Fusion On-Prem

These topics provide information about Spark administration in Fusion Server:

- **Spark Components.** Spark integration in Fusion, including a diagram
- **Spark Getting Started.** Starting Spark processes and working with the shell and the Spark UI
- **Spark Driver Processes.** Fusion jobs run on Spark use a driver process started by the API service
- **Spark Configuration.** How to configure Spark for maximum performance. The article also provides information about ports, directories, and configuring connections for an SSL-enabled Solr cluster.
- **Scaling Spark Aggregations.** How to configure Spark so that aggregations scale
- **Spark Troubleshooting.** How to troubleshoot Spark

Additionally, you can configure and run *Spark jobs* in Fusion, using the *Spark Jobs API* or the *Fusion UI*.

Spark with Fusion AI

With a Fusion AI license, you can also use the Spark cluster to *train and compile machine learning models*, as well as to run experiments via the *Fusion UI* or the *Spark Jobs API*.

Related concepts

- *Jobs and Schedules*
- *Spark Jobs*

Related reference topics

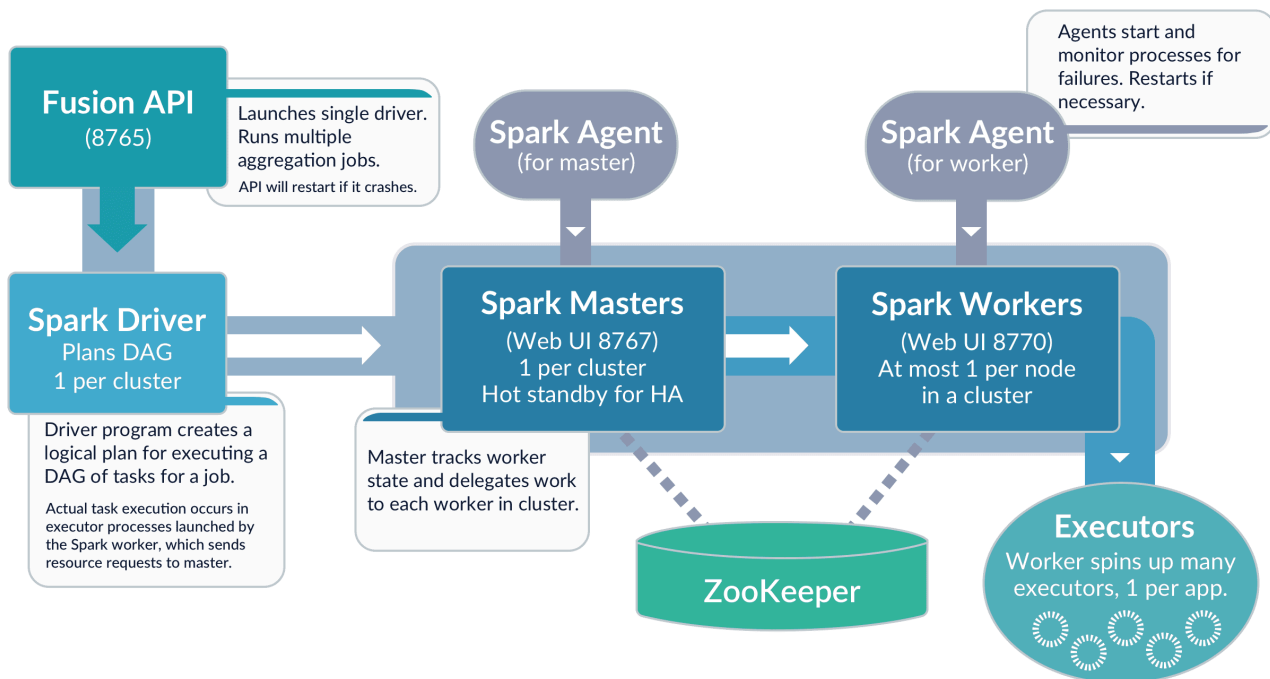
- *Spark jobs for Fusion AI*

Further Reading

- Apache Spark Key Terms, Explained (<https://databricks.com/blog/2016/06/22/apache-spark-key-terms-explained.html>)
- Apache Spark on Wikipedia (https://en.wikipedia.org/wiki/Apache_Spark)

Spark Components

This diagram shows the Spark components available from Fusion:

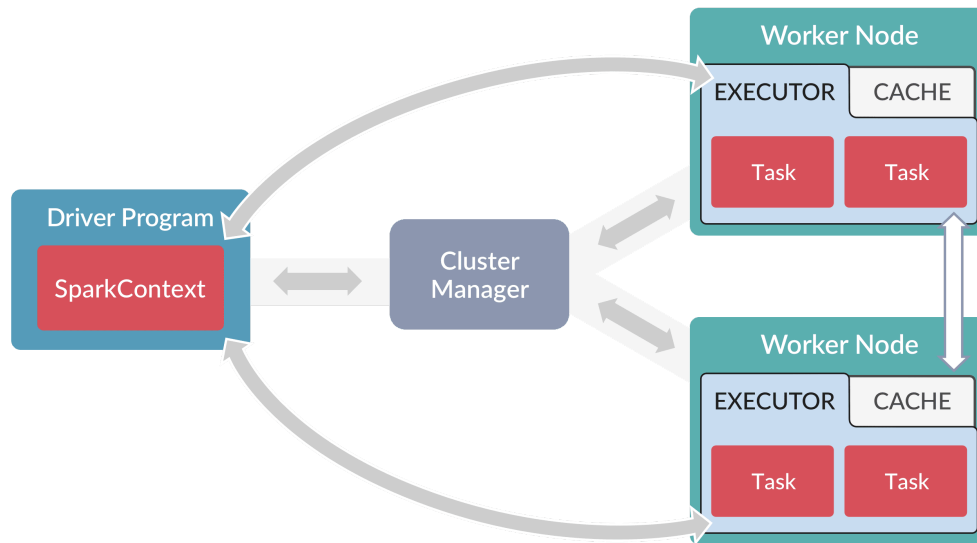


Spark components in Fusion

- **Application.** An active SparkContext in the Spark Master web UI, which consists of a classpath and a configuration. Jobs submitted to the cluster always run as classes in a specific application, that is, using the application's classpath and configuration.
- **SparkDriver.** The Spark driver program, a 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 is no longer needed.
- **Spark master (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 (spark-worker):** Agent-managed Fusion service that launches executors for Spark applications. Spark-workers communicate with the master to launch executors for an application.
- **SQL service (sql):** Agent-managed Fusion service that runs Spark's thrift-based SQL engine. It provides JDBC access to a Spark cluster.
- **Spark shell (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 a shaded Fusion JAR added.
- **Custom script job.** A Fusion job that executes a custom Scala script using the Spark shell.
- **Spark Job Workbench.** A toolkit provided by Lucidworks to help build custom Spark jobs using Scala, Java, or Python. See Spark Job Workbench (<https://github.com/lucidworks/fusion-spark-job-workbench>).
- **CoarseGrainedExecutorBackend.** Executor process(es) launched by a spark-worker to execute the tasks for a specific application, such as the spark-shell.
- **Shaded JAR.** The Fusion API service creates an assembly jar (also call an uber jar) that contains all of the dependencies needed to use spark-solr and Fusion classes within a Spark job. Classes that conflict with classes on Spark's classpath are shaded to ensure that Fusion classes use the correct version.

Spark Job Drivers

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 to 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.

Custom jobs

Fusion supports custom Spark jobs that are written in Scala, Java, or Python jobs and that are built using the Spark Job Workbench (<https://github.com/Lucidworks/fusion-spark-job-workbench>), a toolkit provided by Lucidworks. See the examples in the repository for details.

The repository examples may rely on third-party information or require prerequisites. Verify all configuration and usage before using the examples.

Drivers

Fusion has four types of job drivers:

- **Default driver.** Executes built-in Fusion jobs, such as a signal aggregation job or a metrics rollup job.
- **Script-job driver.** Executes custom script jobs; a separate driver is needed to isolate the classpath for custom Scala scripts.
- **Spark-shell driver.** Wrapper script provided with Fusion to launch the Spark Scala REPL shell with the correct master URL (pulled from Fusion's API) and a shaded Fusion JAR added. Launched using `{fusion-parent-unix}/fusion/4.1{sub-version}/bin/spark-shell` (on Unix) or `{fusion-parent-windows}\var\fusion\4.1{sub-version}\bin\spark-shell.cmd` (on Windows).
- **Custom-job driver.** Executes custom jobs built using the Spark Job Workbench (<https://github.com/Lucidworks/fusion-spark-job-workbench>), a toolkit provided by Lucidworks to help build custom Spark jobs using Scala, Java, or Python.

Reference Guides

If you also have a Fusion AI license, be sure to see the *Fusion AI Reference* for reference information about features that are exclusive to Fusion AI.

- *Language Support*
- *API Reference Guide*
- *Connectors Reference Guide*
- *Parser Stages Reference Guide*
- *Index Pipeline Stages Reference Guide*
- *Query Pipeline Stages Reference Guide*

System Requirements

Requirements for Fusion installation are detailed below.

Lucidworks recommends <i>not</i> virus scanning the <code>fusion/data</code> folder. Virus scanning can cause slow performance, and it can cause downtime if it quarantines an index file identified as a possible virus.
Fusion does not support installation under a directory whose name contains a space.

Supported Operating Systems

Certain systems may place limits on open files or processes that may be used by a process or user (for example, Unix systems may use <code>ulimit</code> and the <code>limits.conf</code> file). Fusion may require increasing these limits. We recommend setting these to "unlimited" or the maximum allowed by the system. If "unlimited" is not an option, we suggest a minimum limit of 65535 on the number of open files, and a minimum limit of 16384 on the number of processes for the Fusion user.

Supported for production use of Fusion services

Operating system:

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

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

RedHat based Linux, including CentOS, must be 6.6.x or later. See http://www.infoq.com/news/2015/05/redhat-futex (http://www.infoq.com/news/2015/05/redhat-futex) for more information.

JVM:

See *Java requirements* below.

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 and later

For information about the embedded versions of Solr, ZooKeeper, and so forth, see the *Release Notes*.

JVM:

See *Java requirements* below.

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.) Even if you have only a small amount of content, we recommend provisioning an 8 core machine as having only 4 cores runs into a possible issue with allocating enough Java heap memory.

** Large production environments will have 3+ nodes with these specs.

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 its 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

Java requirements

Java 8 is recommended for Lucidworks Fusion 4.1. In some cases, Java 9 can be used without issue. Java 10 and later is not supported by Lucidworks Fusion 4.1.

Fusion is a Java-based application, and some features require a pre-installed JDK. The JRE alone may not be sufficient.

- Oracle JDK 1.8 (<https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>) or JRE 1.8 (<https://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>), 64-bit (x64 only)
- OpenJDK JDK or JRE 1.8 (<https://openjdk.java.net/install/>), 64-bit (x64/x86_64 only)
- AdoptOpenJDK 1.8 (<https://adoptopenjdk.net/releases.html?variant=openjdk8&jvmVariant=hotspot>), 64-bit (x64 only)

For Windows users, the Fusion start and stop logging output found in `$Fusion\var\logs\metrics\metric.log` tends to be more reliable when using AdoptOpenJDK.

Checking your Java version

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
```

JavaScript pipeline stages

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.

Cluster Requirements

Supported Solr Versions

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

Fusion Server 4.1 is certified with Solr 7.4.0 (included), 7.2.1, and 6.6.2.

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 (<http://www.ntp.org/>).

Supported ZooKeeper Versions

Fusion includes an embedded instance of ZooKeeper; see the *release history* to find out which version of ZooKeeper is included in each Fusion release. For demo, trial, and some testing deployments, you can use the embedded ZooKeeper. *For production deployments, we recommend using an external ZooKeeper cluster.*

ZooKeeper 3.4.11 is supported for external ZooKeeper instances.

A ZooKeeper cluster must have 3+ nodes configured as an ensemble, and it must have an odd number of nodes. We recommend three ZooKeeper nodes in a ZooKeeper cluster/ensemble. Only for very large SolrCloud clusters (50+ nodes) might you need more ZooKeeper nodes.

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 Extended Support Release (ESR)
- Internet Explorer 11

Fusion Server 4.x File Download Links

This section contains download links for the Fusion Server 4.x.x files other than connectors.

See <i>Fusion 4.x VI Connector Downloads</i> to access links to download connectors for the Fusion 4.x releases.

Fusion application files

Select the link to download the Fusion release:

Release	Zip
Fusion 4.0.0	fusion-4.0.0.zip (https://download.lucidworks.com/fusion-4.0.0/fusion-4.0.0.zip)
Fusion 4.0.1	fusion-4.0.1.zip (https://download.lucidworks.com/fusion-4.0.1/fusion-4.0.1.zip)
Fusion 4.0.2	fusion-4.0.2.zip (https://download.lucidworks.com/fusion-4.0.2/fusion-4.0.2.zip)
Fusion 4.1.0	fusion-4.1.0.zip (https://download.lucidworks.com/fusion-4.1.0/fusion-4.1.0.zip)
Fusion 4.1.1	fusion-4.1.1.zip (https://download.lucidworks.com/fusion-4.1.1/fusion-4.1.1.zip)
Fusion 4.1.3	fusion-4.1.3.zip (https://download.lucidworks.com/fusion-4.1.3/fusion-4.1.3.zip)
Fusion 4.2.0	fusion-4.2.0.zip (https://download.lucidworks.com/fusion-4.2.0/fusion-4.2.0.zip)
Fusion 4.2.1	fusion-4.2.1.zip (https://download.lucidworks.com/fusion-4.2.1/fusion-4.2.1.zip)
Fusion 4.2.2	fusion-4.2.2.zip (https://download.lucidworks.com/fusion-4.2.2/fusion-4.2.2.zip)
Fusion 4.2.3	fusion-4.2.3.zip (https://download.lucidworks.com/fusion-4.2.3/fusion-4.2.3.zip)
Fusion 4.2.4	fusion-4.2.4.zip (https://download.lucidworks.com/fusion-4.2.4/fusion-4.2.4.zip)
Fusion 4.2.5	fusion-4.2.5.zip (https://download.lucidworks.com/fusion-4.2.5/fusion-4.2.5.zip)
Fusion 4.2.6	fusion-4.2.6.zip (https://download.lucidworks.com/fusion-4.2.6/fusion-4.2.6.zip), Linux: fusion-4.2.6.tar.gz (https://download.lucidworks.com/fusion-4.2.6/fusion-4.2.6.tar.gz)

Fusion 4.2.0 drivers

Select the link to download the Fusion driver:

Driver	Zip
chrome-linux	chrome-linux.zip (https://download.lucidworks.com/fusion-4.2.0/drivers/chrome-linux.zip)
chrome-remote-interface-js-eval-tool-alpine-linux	chrome-remote-interface-js-eval-tool-alpine-linux.zip (https://download.lucidworks.com/fusion-4.2.0/drivers/chrome-remote-interface-js-eval-tool-alpine-linux.zip)
chrome-remote-interface-js-eval-tool-linux	chrome-remote-interface-js-eval-tool-linux.zip (https://download.lucidworks.com/fusion-4.2.0/drivers/chrome-remote-interface-js-eval-tool-linux.zip)
chrome-remote-interface-js-eval-tool-macos	chrome-remote-interface-js-eval-tool-macos.zip (https://download.lucidworks.com/fusion-4.2.0/drivers/chrome-remote-interface-js-eval-tool-macos.zip)
chrome-remote-interface-js-eval-tool-win	chrome-remote-interface-js-eval-tool-win.zip (https://download.lucidworks.com/fusion-4.2.0/drivers/chrome-remote-interface-js-eval-tool-win.zip)
chromedriver_linux64	chromedriver_linux64.zip (https://download.lucidworks.com/fusion-4.2.0/drivers/chromedriver_linux64.zip)
chromedriver_mac64	chromedriver_mac64.zip (https://download.lucidworks.com/fusion-4.2.0/drivers/chromedriver_mac64.zip)
chromedriver_win32	chromedriver_win32.zip (https://download.lucidworks.com/fusion-4.2.0/drivers/chromedriver_win32.zip)
chromium-nosync	chromium-nosync.zip (https://download.lucidworks.com/fusion-4.2.0/drivers/chromium-nosync.zip)

Fusion 4.2.6 Service Pack application files

Select the link to download the Fusion 4.2.6 Service Pack for your operating system. Before applying service packs, review [instructions on upgrading Fusion Server 4.2.6 to Latest SP](#).

Operating system(s)	Package
Windows Zip archive for new Fusion 4.2.6 SP installations and patch updates.	fusion-4.2.6-sp3.zip (https://storage.googleapis.com/docs-downloads/fusion-4.2.6-sp3.zip)
Linux archive for new Fusion 4.2.6 SP installations and patch updates	fusion-4.2.6-sp3.tar.gz (https://storage.googleapis.com/docs-downloads/fusion-4.2.6-sp3.tar.gz)
Linux/Windows patch program	fusion-4.2.6-service-pack-patch.jar (https://storage.googleapis.com/docs-downloads/fusion-4.2.6-service-pack-patch.jar)

Additional Resources

In addition to this documentation, these resources are available to assist you:

- **Blog** (<https://lucidworks.com/blog/>)
Learn from the experts in these in-depth topics.
- **Videos and webinars** (<https://lucidworks.com/videos/>)
View demos, webinars, and tutorials.
- **Answers** (<https://support.lucidworks.com/>)
Browse questions and answers from the Lucidworks community, or ask a new question.
- **Knowledge base** (<https://support.lucidworks.com/>)
Find solutions in these articles from the Lucidworks support team.
- **Fusion Server 4.x File Download Links**
Download Fusion products and drivers.
- **Fusion 4.x V1 Connector Downloads**
Download Fusion connectors.
- **Open a support case** (<https://support.lucidworks.com/hc/en-us/requests/new>)
Get personalized assistance from Lucidworks support engineers. A customer login is required.

Pipeline Stages

Index Pipeline stages are used to create and modify PipelineDocument objects.

Query Pipeline stages are used to modify Request objects and Response objects.

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**.

Pipeline Condition Expression

The JavaScript expression specified in the condition property of a pipeline stage has access the pipeline objects.

Index Pipeline Stage Condition Example

An index PipelineDocument has two available variables: "doc" and "ctx".

Check whether pipeline document contains a named field:

```
doc.hasField("acl_ss")
```

Query pipeline stage Condition Example

Process query based the query Request object:

```
request.hasParam("fusion-user-name") && request.getFirstParam("fusion-user-name").equals("SuperUser"); !request.hasParam("isFusionPluginQuery")
```

The first condition checks that the request parameter "fusion-user-name" is present and has the value "SuperUser". The second condition checks that the request parameter "isFusionPluginQuery" is not present.

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:

Setup

- *Active Directory Security Trimming*
- *Field Facet*
- *Query Fields*
- *Security Trimming*

Relevancy

- *Block Documents*
- *Landing Pages*
- *Parameterized Boosting*
- *Recommend More Like This*
- *Recommend Items for User*
- *Recommend Items for Item*

Fetch data

- *JDBC Lookup*
- *REST Query Stage*
- *Solr Query*
- *Solr Subquery*

Troubleshooting

- *Logging*
- *Send PagerDuty Message*
- *Send Slack Message*
- *Send SMTP Email*
- *Write Log Message*

Advanced

- *Additional Query Parameters*
- *JavaScript Query Stage*
- *Managed Javascript*
- *Retrieve Stored Parameters*

Other

- *Analytics Catalog Query*
- *Call Pipeline*
- *Experiment Query*
- *Parameterized Faceting*
- *Return Query Parameters*
- *Rollup Aggregation*

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 USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d
'{"id":"ADSecurity", "type": "active-directory-security-trimming",
"server":"ldap://hostname:port", "bindName":"ADuser@example.com", "bindPassword":"l
ogin1"}' http://localhost:8764/api/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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Additional Query Parameters Stage

The Additional Query Parameters query pipeline stage is used to set, append, and remove Solr query parameters (<https://wiki.apache.org/solr/QueryParametersIndex>). 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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Analytics Catalog Stage

The Analytics Catalog query pipeline stage lets you define views of data stored in Fusion collections. You can query the objects in the Analytics Catalog using SQL, Solr Streaming Expressions (<https://cwiki.apache.org/confluence/display/solr/Streaming+Expressions>), or regular Solr queries. See also the *Catalog API*.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

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 does not 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 entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Boost Documents Stage

The Boost Documents query pipeline stage adds boosting parameters to matched documents based on user-defined rules. Boosts are defined with a term value to boost and the boost factor to add. The boosting parameters are added to the `boost` 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`. The 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 does not require phrase matching.
- `boosts`. Consists of a list of pairs.
 - `value`. One or more terms used for boosting.
 - `boost`. The numeric boost value. Default `100`.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Boost with Signals Stage

The Boost with Signals query pipeline stage uses *aggregated signals* to selectively boost items in the set of search results.

Using the main query and the stage configuration parameters, this stage performs a secondary query to the `_signals_aggr` collection and returns updated boost weights for the items in the main query's search results. Items that have received more user interaction also receive higher boost weights.

See *Recommendation Methods* for more information.

This stage accesses the <code>signals_aggr</code> collection. Before using it, verify that the following <i>permission</i> is set: <code>GET:/solr/<collection-name>_signals_aggr/select</code>

Configuration overview

The fields below are especially useful to understand when configuring this stage.

Number of Recommendations numRecommendations	Sets the <code>rows</code> query param in the main query as the maximum number of query results which will be boosted by this pipeline stage.
Number of Signals numSignals	Sets the <code>rows</code> query param in the query that searches the <code>_signals_aggr</code> collection, so only the specified number of aggregated signals are retrieved and used for boosting. When signals boosting is applied to a query, aggregated signals records are queried from the appropriate <code>_signals_aggr</code> collection to find out the popularity or boost weight for documents which have signals. <code>numSignals</code> limits the number of records to be queried from a <code>{{_signals_aggr}}</code> collection and used to calculate this boost.
Aggregation Type aggrType	A filter to retrieve aggregated signals in the <code>_signals_aggr</code> collection per each aggregated signal's <code>aggr_type_s</code> field value.
Solr Field to Boost On boostId	The document field in the main collection on which to perform boosting. Typically it should use default field, which is <code>id</code>. This field corresponds to the Rollup Field/<code>rollupField</code> field. Together, these two fields act like a <code><field>:<value></code> pair in the query modification for boosting.
Boost Method boostingMethod	This adds a query parameter to the original query, either “query-param” or “query-parser”. The result is (“query-param” or “query-parser”) + Boost Param(“boost” or <code>bq</code>), as in the examples below: “query-param”+“boost”, result boost query param <pre>boost="map(query({!field f='id' v='6239046,13026192'}), 0, 0, 1, 27.1705)"</pre> “query-parser”+“boost”, result boost query param <pre>bp_xxx_bbqx="map(query({!field f='id' v='6239046,13026192'}), 0, 0, 1, 27.1705)"</pre> When Boost Param uses <code>bq</code>, similar logic applies. When Boost Param/<code>boostingParam</code> uses “boost”, it works with both “query-param” and “query-parser”.
Rollup Field rollupField	Indicates which aggregated signal document field the boost parameter will use for the final boosting. It works in combination with the Solr Field to Boost On/<code>boostId</code> field. This should be set to the field in the aggregated signal collection that stores the doc list that is aggregated as one record. By default it's set to <code>doc_id_s</code>, because by default this is used in the <i>click_signal_aggr SQL job</i>.

Rollup Weight Field <code>rollupWeightField</code>	Indicates the final boost weight used to calculate the new score for docs retrieved by the main query. Similar to Rollup Field/<code>rollupField</code> above, this should be set to the field in the aggregated signal collection that stores the final weight that was calculated. By default it's <code>weight_d</code>, because by default this is used in the <code>click_signal_aggr SQL job</code>.
Final Boost Weight Expression <code>weightExpression</code>	Calculates the final weight using the weight and score retrieved from the <code>_signals_aggr</code> collection. The default value is <code>math:log(weight_d + 1) + 10 * math:log(score+1)</code>.

Solr query parameters

These parameters are used in the **Solr Query parameters**/`queryParams` field for retrieving signal aggregation docs from the `_signals_aggr` collection. These Solr query params will affect which aggregated signals are used for producing the boosting parameter on the main query.

<code>qf=query_t</code>	Defines which field to query. In the default case, the query searches on the <code>query_t</code> field of aggregated signal docs.
<code>pf=query_t^50</code>	Boosts docs within the set of retrieved docs using phrase matching.
<code>pf2=query_t^20</code>	<code>pf2</code> is similar to <code>pf</code> ; the difference is that <code>pf2</code> works on bigram phrases.
<code>pf3=query_t^10</code>	<code>pf3</code> is similar to <code>pf</code> ; the difference is that <code>pf3</code> works on trigram phrases.

FAQs

If there is `fq` in the main query, how is it matched with the correct aggregated signal?

In this case, you need to use the `lw.rec.fq` query parameter in the main query. `lw.rec.fq` can be parsed by the Boost with Signals stage, and therefore the filters specified in it can be added to the Solr query that is retrieving the aggregated signals.

For example, if we have filter query param `fq=format:CD&fq=name:Latin`, this needs to be translated into `lw.rec.fq=filters_s:"format:cd $ name:latin"`. Values must be lowercase. The final main query should be:

```
http://localhost:8764/api/apollo/apps/demo_app/query-
pipelines/demo_app/collections/demo_app/select?
echoParams=all&wt=json&json.nl=arrarr&sort&start=0&q=apple&debug=true&rows=10&lw.
rec.fq=filters_s:"format:cd $ name:latin"
```

Now the Boost with Signals stage will only retrieve aggregated signals that have the same filter query.

If there are multiple `fq` values (for example, `format:cd` and `name:latin`), they are ordered alphabetically as strings and joined with " \$ " (a \$ with a space on each side). In the example, `"format:cd $ name:latin"`.

What if my aggregated signals are in a different collection?

You can point the Boost with Signals stage to a different signal collection by adding a `collection` parameter in the `Solr Query Parameters` section.

Solr Query parameters

Parameters for querying Signal aggregation collection

 * Parameter Name	Parameter Value
 qf	query_t
 pf	query_t^50
 pf	query_t~3^20
 pf2	query_t^20
 pf2	query_t~3^10
 pf3	query_t^10
 pf3	query_t~3^5
 boost	map(query({!field f=qu
 mm	50%
 defType	edismax
 sort	score desc, weight_d d
 collection	signal_aggre2

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Custom JavaScript Query Stage Examples

The following is a list of examples for reference when creating *custom JavaScript Query stages*.

Add a parameter to the query request

```
function(request,response , ctx, collection, solrServer, solrServerFactory) {  
  request.addParam("foo", "bar"); }  
}
```

Add a parameter to the query response

This example contains a simple JavaScript function which copies information from the `ctx` (Context) object into the query response. Requirements:

- The `response` parameter cannot be used in query pipeline stages prior to the Solr query stage. This example assumes use in a later stage.
- The `response.initialEntity.appendStringList()` function only works if the query's `wt` (writer type) parameter is set to json or xml, for example: `wt=json`.

```
(function () { "use strict"; var List = Java.type('java.util.List'); function  
add_to_response(key, list) { if (list.length > 0) {  
  response.initialEntity.appendStringList(key, Java.to(list, List)); } } return  
function(request,response , ctx, collection, solrServer, solrServerFactory) {  
  add_to_response('banners', ctx.getProperty('banners')); add_to_response('landing-  
pages', ctx.getProperty('redirects')); }; }());
```

Java object interactions

The JavaScript engine used by Fusion is the Nashorn engine from Oracle. See The Nashorn Java API (<https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/api.html>) for details.

The Nashorn engine exposes the full Java Classpath to JavaScript. This means that Java objects can be instantiated and called from JavaScript. Objects returned by a call to a Java function will not be JavaScript objects. For example, calling `typeof(object)` will not give meaningful results. Instead, you can use JavaScript to discover the Java type of an object. Once you know the Java type of an object, you can refer to the *Fusion Pipeline Javadocs* to learn more about the Java type and how to use Java Objects.


```

/* globals Java, logger*/ (function () { "use strict"; var Collection =
Java.type("java.util.Collection") var isDebug = false function logIfDebug(m) { if
(isDebug && m) logger.info(m, Array.prototype.slice.call(arguments).slice(1)); }
/** * @return a boolean indicating whether or not a given object looks like a
Java object */ function isJavaType(obj) { return (obj && typeof obj.getClass ===
'function' && typeof obj.notify === 'function' && typeof obj.hashCode ===
'function'); }; function isJavaArray(obj) { return (obj && isJavaType(obj) &&
getTypeOf(obj).endsWith('[]')); }; function isJavaCollection(collection) { return
isJavaType(collection) && Collection['class'].isInstance(collection); }; /** *
Nashorn allows obj.length and obj['property'] and obj.property functionality *
for some object types. This helps discover that fact at runtime */ function
isArrayLike(obj) { return isJavaCollection(obj) || Array.isArray(obj) ||
isJavaArray(obj); }; /** * For Java objects, return the simplified version of
their classname e.g. 'Format' for a java.text.Format * because Java naming
conventions call for upper case in the first character. * * For JavaScript
objects, the type will be a lower case string e.g. 'date' for a JavaScript Date.
* Note: Nashorn uses java.lang.String objects instead of JavaScript string
objects */ function getTypeOf(obj, verbose) { 'use strict'; var typ = 'unknown';
var isJava = isJavaType(obj); logIfDebug("UTIL getTypeOf:called with verbose:
",verbose); //test for java objects if (isJava && verbose) { typ =
obj.getClass().getName(); logIfDebug("UTIL getTypeOf:inverbose block, type::
",typ); } else if (isJava) { typ = obj.getClass().getSimpleName(); } else if (obj
=== null) { typ = 'null'; } else if (typeof (obj) === typeof (undefined)) { typ =
'undefined'; } else if (typeof (obj) === typeof (String())) { typ = 'string'; }
else if (Array.isArray(obj)) { typ = 'array'; } else if
(Object.prototype.toString.call(obj) === '[object Date]') { typ = 'date'; } else
{ typ = obj ? typeof (obj) : typ; } return typ; }; return function main(request,
response, ctx, collection, solrServer, solrServerFactory) { //output info about
the objects passed in logger.info("request object has type
'{}'",getTypeOf(request,true)) //note, response is only valued after the solr
query stage logger.info("response object has type '{}'",getTypeOf(response,true))
logger.info("ctx object has type '{}'",getTypeOf(ctx,true))
logger.info("collection object has type '{}'",getTypeOf(collection,true))
logger.info("solrServer object has type '{}'",getTypeOf(solrServer,true))
logger.info("solrServerFactory object has type
'{}'",getTypeOf(solrServerFactory,true)) }; })();

```

Manually add dependencies

To install dependencies manually, jar files must be placed in the `./apps/libs` folder. The jar file path must be included in `./apps/jetty/api/webapps/api-extra-classpath.txt` and `apps/jetty/connectors-classic/webapps/connectors-extra-classpath.txt`.

For example, to create script objects that access and reference Java types from Javascript use the `Java.type()` function:

```

var TwitterFactory = Java.type("twitter4j.TwitterFactory"); var twitter =
TwitterFactory.getSingleton();

```

For more information, see *Custom JavaScript Stages Variables and Parameters for Fusion 4.x.x*.

Experiment Stage

The Experiment query pipeline stage is part of Fusion's *Machine Learning* framework.

The Machine Learning framework provides tools to evaluate alternative methods of computing and displaying search results. For example, an *experiment* may consist of a system which has two or more different query pipelines running, and users are randomly served search results using some variant of the system via instrumented pages that capture user response. In such a system, the Experiment query stage selects a variant from a running experiment and injects its properties into the current *Context*.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Field Facet Stage

The Field Facet query pipeline stage is used to add a Solr Field Facet query (<https://cwiki.apache.org/confluence/display/solr/Faceting#Faceting-Field-ValueFacetingParameters>) 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 (<https://cwiki.apache.org/confluence/display/solr/Faceting#Faceting-Field-ValueFacetingParameters>).

It is 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 (<https://cwiki.apache.org/confluence/display/solr/Faceting#Faceting-Thefacet.methodParameter>) 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 Faceting Overview (<https://lucene.apache.org/solr/guide/faceting.html>).

Configuration

JDBC Lookup Stage

The JDBC 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.

You must first *upload the JDBC driver to Fusion*.

Example

An example of setup for a JDBC Lookup query pipeline stage.

Upload stage config via POST to Fusion REST API endpoint `/api/query-stages/instances``

```
curl -u USERNAME:PASSWORD -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/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 entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

JavaScript Query Stage

The JavaScript Query stage allows you to write 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 objects and Response objects, respectively.

JavaScript Stages Variables and Parameters

JavaScript (<http://en.wikipedia.org/wiki/JavaScript>) is a lightweight scripting language. The JavaScript in a JavaScript stage is standard ECMAScript (<http://en.wikipedia.org/wiki/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
<code>request</code>	<i>Request</i>	The Solr query information.
<code>response</code>	<i>Response</i>	The Solr response information.
<code>ctx</code>	<i>Context</i>	A reference to the container which holds a map over the pipeline properties. Used to update or modify this information for downstream pipeline stages.
<code>collection</code>	String	The name of the Fusion collection being indexed or queried.
<code>solrServer</code>	<i>BufferingSolrServer</i>	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 (https://lucene.apache.org/solr/5_2_1/solr-solrj/org/apache/solr/client/solrj/SolrClient.html) for details.
<code>solrServerFactory</code>	<i>SolrClientFactory</i>	The SolrCluster server used for lookups by collection name which returns a Solr server instance for a that collection. For example: <code>var productsSolr = solrServerFactory.getSolrServer("products");</code>
<code>QueryRequestAndResponse</code>	<i>QueryRequestAndResponse</i>	Returns list of info as a string.

Syntax Variants

JavaScript stages are written using *function syntax*, which passes variables as function parameters.

Function Syntax

```
function(request,response) {  
    request.addParam("foo", "bar");  
}
```

Function syntax is used for the examples in this document.

Global variable `logger`

The global variable named `logger` writes messages to the log file of the server running the pipeline. This variable is truly global and does not need to be declared as part of the function parameter list.

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".

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

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 does not 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`. The 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 does not require phrase matching.
- `url`. A list of URLs.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Logging Stage

The Logging query pipeline stage prints messages to the API log file; the default location is `https://FUSION_HOST:FUSION_PORT/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 entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Managed JavaScript Query Stage

This stage manipulates a query using a script from the *blob store*. The script must have the blob type `file:js-query`.

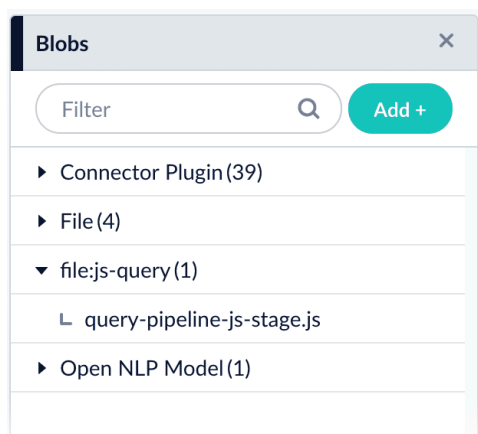
See also: *Custom JavaScript Stages For Query Pipelines*.

Uploading a JavaScript blob

In Fusion 4.1, JavaScript blobs must be uploaded using the *Blob Store API*, like this:

```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-Type: text/javascript' --data-binary @query-pipeline-js-stage.js http://localhost:8764/api/apps/APP_NAME/blobs/query-pipeline-js-stage.js?resourceType=file:js-query
```

Open the *blob manager* to verify that the new blob is present:



Referencing a JavaScript blob

Use any of the following formats for the `ref`/Script Reference configuration field:

```
query-pipeline-js-stage.js
```

```
blob:query-pipeline-js-stage.js
```

```
urn:x-fusion:blob:query-pipeline-js-stage.js
```

The examples above use `query-pipeline-js-stage.js` as the blob ID.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Parameterized Boosting Stage

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Parameterized Faceting Stage

The Parameterized Faceting query pipeline stage determines the most frequent value for a given field, then uses that value to look up stored parameters and auto-select the most appropriate facet for a query.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Query Fields Stage

The Query Fields query pipeline stage defines common Solr query parameters for the edismax query parser (<https://cwiki.apache.org/confluence/display/solr/The+Extended+DisMax+Query+Parser>). If using a less-than sign (<) with DisMax, it must be escaped using a backslash. An alternative to this stage is the *Additional Query Parameters stage*.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

REST Query Stage

The REST Query pipeline stage is available in Fusion versions 3.0 and later.

In Fusion version 3.0 and earlier, this pipeline stage was called the Query RPC Stage. RPC is the acronym for Remote Procedure Calls.

The notation to send text as payload in **Request entity (as string)** field is `<request.q>`.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Recommend Items for Item Stage

The Recommend Items for Item query pipeline stage uses signals about users' item choices to recommend related items based on a specific item. Relationships between items can be based on different criteria, such as click patterns, people who bought this also bought that, percentage match of document tags, and so on.

Given an item ID, this stage performs a secondary query to the `_items_for_item_recommendations` collection to find related items, then retrieves those items from the main collection.

This pipeline stage uses items-for-item recommendations that have been precomputed by an *ALS Recommender*.

See also *Items-for-item Recommendations* to learn how to configure this recommender type and fetch recommendations.

Prerequisites

Enable recommendations:

Before creating a Recommend Items for Item stage, enable recommendations.

- **Fusion UI** – In Collections Manager, click the settings icon next to the collection > **Enable Recommendations**.
- **Using the REST API** – Use this command to enable recommendations:

```
curl -u admin:<password> -X PUT \http://<hostname>:  
<port>/api/v1/collections/<collection-name>/features/recommendations -H  
'content-type: application/json' -d '{"enabled":true}'
```

When you enable recommendations, Fusion creates a query pipeline that already contains this stage, and that is configured for boosting. The query pipeline is `COLLECTION_NAME_items_for_item_recommendations`.

Using live signals

The **Estimate Recent Results** option uses live signals to augment items-for-user recommendations with real-time recommendations.

When this is enabled, Fusion first looks up items (from previously-generated recommendations) that are similar to the new items. If there are none then it looks up similar users (who also interacted with that item) to get a list of recommendations based on the new items. It then combines those new recommendations with the job-based recommendations already generated for that user (if any), to generate a final list of recommendations.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Recommend Items for User Stage

The Recommend Items for User query pipeline stage uses signals about item choices to recommend other similar items for a specific user. Personalization for the user can be based on the user's search history, browsing history, or purchase history, and so on.

This pipeline stage uses items-for-user recommendations that have been precomputed by an *ALS Recommender* job.

See also *Items-for-user Recommendations* to learn how to configure this recommender type and fetch recommendations.

Prerequisites

Enable recommendations:

Before creating a Recommend Items for Item stage, enable recommendations.

- **In the Fusion UI** – With Query Workbench open, click **Settings > Enable Recommendations**.
- **Using the REST API** – Use this command to enable recommendations:

```
`curl -u admin:<password> -X PUT http://<hostname>:<port>/api/v1/collections/<collection-name>/features/recommendations -H 'content-type: application/json' -d '{"enabled":true}'`
```

When you enable recommendations, Fusion creates a query pipeline that already contains this stage, and that is configured for boosting. The query pipeline is <code>COLLECTION_NAME_items_for_user_recommendations</code> .

Using live signals

The **Estimate Recent Results** option uses live signals to augment items-for-user recommendations with real-time recommendations.

When this is enabled, Fusion first looks up items (from previously-generated recommendations) that are similar to the new items. If there are none then it looks up similar users (who also interacted with that item) to get a list of recommendations based on the new items. It then combines those new recommendations with the job-based recommendations already generated for that user (if any), to generate a final list of recommendations.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Retrieve Stored Parameters Stage

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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Return Query Parameters 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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Rollup Aggregation Stage

The Rollup Aggregation query pipeline stage (previously called the Rollup Aggregator stage) is used to roll up Solr results from a context variable. In 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 field 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 be used later in a Parameterized Boosting stage.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Run Query Pipeline Stage

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Send PagerDuty Message Stage

This stage sends a PagerDuty (<https://www.pagerduty.com/>) Message from Fusion, for alerting, monitoring, and more, using Fusion's *Messaging Services*.

Read more about PagerDuty integration in Fusion on our blog (<https://lucidworks.com/post/pagerduty-integration-in-lucidworks-fusion/>).

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:
 - o PagerDuty service key (<https://developer.pagerduty.com/documentation/integration/events/>)
 - o 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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Send SMTP Email Stage

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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Send Slack Message Stage

This stage sends a Slack (<https://slack.com/>) 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. From any app, click **System > Messaging Services**.
2. Select **Slack Message Service** from the drop-down menu.
3. Enter the following information:
 - Slack auth token (<https://api.slack.com/docs/oauth>)
 - 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 entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Solr MoreLikeThis Stage

This stage uses the content of the current document to query for similar documents, using *Solr's MoreLikeThis component*.

This stage provides content-based recommendations. For collaborative recommendations, use the *Recommend Items for Item stage*.

See *Recommendations and Boosting* for more information.

Tips

- The incoming query must include an `id` field in order to get recommendations from this stage. The stage returns documents similar to the one specified by this field.
- Since these secondary queries tend to be large, this stage can impact search performance. You can improve performance by first *clustering your documents*, then configuring this stage to query a specific document cluster instead of all documents.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Solr Query Stage

The Solr Query stage transforms the Fusion query pipeline *Request* object into a Solr query and sends it to Solr.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Solr Subquery Stage

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 Context object, which is part of the Request object. The query must return results in JSON format, so that they can be added to the query pipeline.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Write Log 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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

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:

Document transformation

- *XML Transformation*

Document filtering and enrichment

- *Detect Language*
- *Exclude Documents*
- *Format Signals*
- *Include Documents*
- *JDBC Lookup*
- *REST Query*

Field transformation

- *Date Parsing*
- *Field Mapping*
- *Filter Short Fields*
- *Find and Replace*
- *GeoIP Lookup*
- *Regex Field Extraction*
- *Regex Field Filter*
- *Regex Field Replacement*
- *Resolve Multivalued Fields*
- *Solr Dynamic Field Name Mapping*

Natural language processing

- *Detect Sentences*
- *Gazetteer Lookup Extraction*
- *OpenNLP NER Extraction*
- *Tag Part-of-Speech*

Indexing

- *Solr Indexer*
- *Solr Partial Update Indexer*
- *Update Related Document*

Troubleshooting

- *Logging*
- *Send PagerDuty Message*
- *Send SMTP Email*
- *Send Slack Message*
- *Write Log Message*

Advanced

- *Call Pipeline*
- *Exclusion Filter*
- *Javascript*
- *Machine Learning*
- *Managed Javascript*
- *Set Property*

Apache Tika Parser Index Stage

The Apache Tika Parser index stage type includes rules for parsing documents with Apache Tika (<http://tika.apache.org/>). Fusion uses Tika v1.13. (Note that components of the Solr distribution included with Fusion contain their own Tika jar files; these are not used by Fusion.)

Compatibility Issues

Raw streams create new docs which Tika then tries to parse again. For this reason, avoid using the SDK connector or any other client that streams to the index pipeline.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Call Pipeline Index Stage

The Call Pipeline index stage provides a means of composing index pipelines.

Be aware that the processing does not return to the calling pipeline automatically.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

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 a 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 is 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
<code>tz.test</code>	Europe/Warsaw
<code>tz_offset.test</code>	+01:00
<code>epoch.test</code>	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
sec.utc.test	0
sec.local.test	0

Field name	value
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 is 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 does not matter whether it is Wednesday or Thursday.

Configuration

Adding a New Format

If no new format is added, a default list of formats will be used. The default formats are listed in the table below:

Pattern	Notes
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	
EEEE, dd-MMM-yy HH:mm:ss zzz	
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

Pattern	Notes
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	
dd-MM-yyyy HH:mm:ss zzz	

The list of accepted symbols for constructing a new format are listed below:

Symbol	Meaning	Example
<code>G</code>	era	<code>AD</code>
<code>C</code>	century of era (≥ 0)	<code>20</code>
<code>Y</code>	year of era (≥ 0)	<code>1996</code>
<code>x</code>	weekyear	<code>1996</code>
<code>w</code>	week of weekyear	<code>27</code>
<code>e</code>	day of week	<code>2</code>
<code>E</code>	day of week	<code>Tuesday Tue</code>
<code>y</code>	year	<code>1996</code>
<code>D</code>	day of year	<code>189</code>
<code>M</code>	month of year	<code>July Jul 07</code>
<code>d</code>	day of month	<code>10</code>
<code>a</code>	halfday of day	<code>PM</code>
<code>K</code>	hour of halfday (0~11)	<code>0</code>
<code>h</code>	clockhour of halfday (1~12)	<code>12</code>
<code>H</code>	hour of day (0~23)	<code>0</code>
<code>k</code>	clockhour of day (1~24)	<code>24</code>
<code>m</code>	minute of hour	<code>30</code>
<code>S</code>	second of minute	<code>55</code>
<code>S</code>	fraction of second	<code>978</code>
<code>Z</code>	time zone	<code>Pacific Standard Time; PST</code>
<code>Z</code>	time zone offset/id	<code>-0800 -08:00 America/Los_Angeles</code>
<code>'</code>	escape for text	
<code>' '</code>	single quote	<code>'</code>

The number of symbols used determine the format of the output.

For text output, if 4 or more symbols are used, the full form is used. `EEEE` would output `Tuesday`, while `E` outputs `Mon`.

For numeric outputs, the amount of symbols used is the same as the minimum digits used for representation.

If the number is smaller, then it will be padded with zeroes. `HH` would be `09`, while `H` is `9`.

For the year, the representation for year and weekyear is different. For example, if the count of `y` is two, the year would be the two digit, zero-based year of the century.

For the month, if there are three or more symbols, then the text is used. For example, `MM` would output `03`, `MMM` outputs `Mar`, and `MMMM` outputs `March`.

For the zone, `Z` outputs the offset without a colon, `ZZ` outputs the offset with a colon, and `ZZZ` or more outputs the zone id.

For the zone name, time zone names `'Z'` cannot be parsed.

Any characters in the input that are not in the range of `['a'...'z']` and `['A'...'Z']` will be treated like quoted text. So, characters like `' , '` or `' - '` will appear in the output time text even if they are not wrapped in single quotes.

A new configuration can be made using a combination of the symbols and regular characters. For example, the pattern `E MM/dd/yyyy HH:mm:ss.SSS` would output the formatted date `Mon 01/16/2017 10:53:32.939`.

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Detect Language Index Stage

The Detect Language index stage (called the Language Detection stage in versions earlier than 3.0) operates over one or more fields in the Pipeline Document. The contents of each field are analyzed using the Language Detection Library for Java (<https://github.com/optimaize/language-detector>), 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".

There is also an option to allow detection of multiple languages. This can be achieved by setting "Return all detected languages and their confidence scores." to true. In this case, the detected languages will be either set as document fields in a form of "Field Name_Document Postfix.Language:Confidence", or as a field with name "Output Key" in the Context having a dictionary of following form "{ "language": "probability" }" as a value. Example Document fields could look like this: "plot_summary_txt_lang.pl_: [0.99]", "plot_summary_txt_lang.en_: [0.99]" when languages pl and en would be detected.

Languages

The Language Detection Library for Java has build-in profiles for many languages (<https://github.com/optimaize/language-detector/blob/master/README.md#70-built-in-language-profiles>). 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 (<https://github.com/optimaize/language-detector/blob/master/README.md#challenges>):

- This software does not work as well when the input text to analyze is short, or unclear. 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.
- Detection of unwanted languages (for example the stage might detect some language that is not even used in the input data because of some language similarities). By default, the stage uses a full array of available languages for detection (List here (<https://github.com/optimaize/language-detector>)). If one wants to only use selected languages, this can be configured

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Detect Sentences Index Stage

The Detect Sentences index stage (called the Sentence Detection stage in versions earlier than 3.0) operates over one or 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 (<http://opennlp.apache.org/>) project's Sentence Detection (<http://opennlp.apache.org/docs/#tools.sentdetect>) 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. _

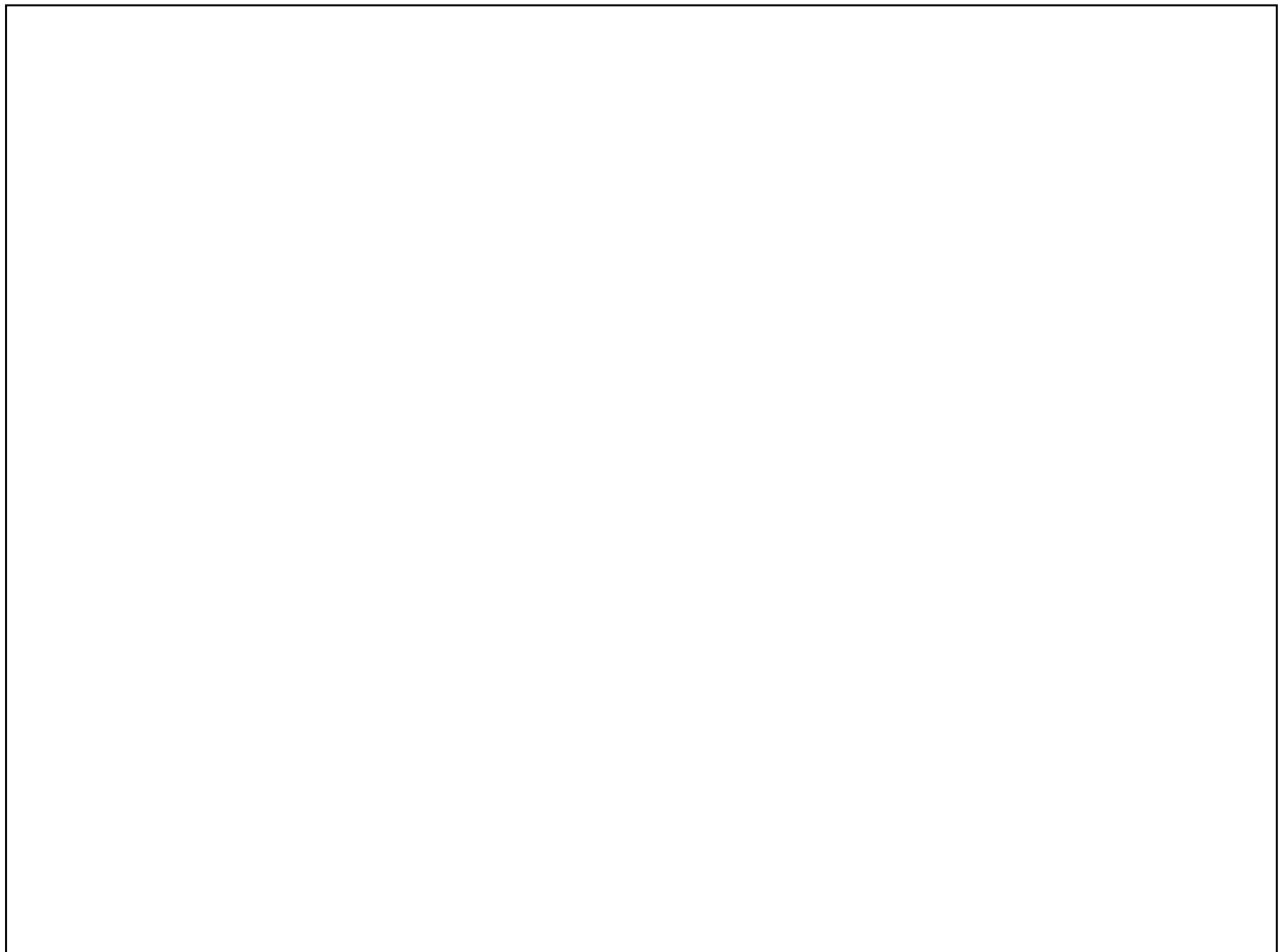
Fusion comes with a set of OpenNLP language models for english. These data files are found in the directory: `https://FUSION_HOST:FUSION_PORT/data/nlp/models`. The included sentence model is `en-sent.bin`.

Models are available from the OpenNLP models SourceForge repository (<http://opennlp.sourceforge.net/models-1.5/>).

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 USERNAME:PASSWORD -X PUT --data-binary @en-pos-maxent.bin -H 'Content-type: text/plain' http://localhost:8764/api/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 USERNAME:PASSWORD -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/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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Exclude Documents Index Stage

The Exclude Documents stage drops all documents that match all of the specified rules (Boolean AND). If some field has multiple values then at least one value must match against the specified pattern. No further processing is done on any matching documents, thus they will not be indexed into a Fusion collection. All non-matching documents are passed to the next stage in the pipeline. Rules are defined using regular expression field matching.

Examples

Give the "simple-exclude" pipeline a stage that excludes certain document types:

```
curl -u USERNAME:PASSWORD -X POST -H "Content-type: application/json"
'http://localhost:8764/api/index-pipelines' -d '{ "id" : "simple-exclude",
"stages" : [ { "type" : "exclude-doc", "matchRules" : [ { "field" :
"document_type", "pattern" : "(xls|xlsx|xlst|doc|docx)" } ] } ] }'
```

Send a text document through the "simple-exclude" pipeline:

```
curl -u USERNAME:PASSWORD 'http://localhost:8764/api/index-pipelines/simple-
exclude/collections/logs/index?simulate=true&echo=true' -H 'Content-type:
application/json' -d '{ "document_type": "txt" }'
```

The response is document metadata, indicating the document passed the stage:

```
[ { "id" : "93da43ff-4218-4f24-a690-23b530926104", "fields" : [ { "name" :
"document_type", "value" : "txt", "metadata" : { }, "annotations" : [ ] } ] } ]
```

Send an XLS document through the "simple-exclude" pipeline:

```
curl -u USERNAME:PASSWORD 'http://localhost:8764/api/index-pipelines/simple-
exclude/collections/logs/index?simulate=true&echo=true' -H 'Content-type:
application/json' -d '{ "document_type": "xls" }'
```

The empty response indicates the document was dropped:

```
[ ]
```

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

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

`https://FUSION_HOST:FUSION_PORT/data/nlp/excludes/excludes.txt`. Here is an example of how to upload this file using `curl`, where `USERNAME:PASSWORD` are the credentials for an admin-level user:

```
curl -u USERNAME:PASSWORD -X PUT --data-binary @data/nlp/excludes/excludes.txt -H 'Content-type: text/plain' http://localhost:8764/api/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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Field Mapping Index Stage

A Field Mapping stage is used to do customized mapping of the fields in an Index Pipeline document to fields in the Solr schema.

For examples of how to use this stage in the Fusion UI, see *Part 2* of the *Getting Started* tutorial.

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 does not 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, <code>/(.*)text(.*)/</code> 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 <code>source</code> 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 <code>Matcher.replaceAll</code> (http://docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html#replaceAll-java.lang.String-)). Otherwise, the source field name will be simply substituted by the value of target according to the operation rules described below.
operation	What to do with the field during mapping. Several options are available: • copy. Content contained in fields matching <code>source</code> will be copied to <code>target</code>. • move. Content contained in fields matching <code>source</code> will be moved to <code>target</code> (it may also help to think of this as the field name being replaced by the value of <code>target</code>). • delete. Content contained in fields matching <code>source</code> will be dropped from the document and not indexed. In this case, the <code>target</code> can be null or not defined at all. • add. The literal value of <code>target</code> will be added to the <code>source</code> if <code>source</code> is a regular expression. If <code>source</code> is not a regular expression, <code>target</code> will be added as a new field. • set. The literal value of <code>target</code> will be set as the new value of <code>source</code> if <code>source</code> is a regular expression. If <code>source</code> is not a regular expression, <code>target</code> will be set as a new field. • keep. Content contained in fields matching <code>source</code> 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 <code>renameUnknown</code> 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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Field Parser Index Stage

Parse the contents of a field using a pre-defined parser.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Filter Short Fields Index Stage

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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Find and Replace Index Stage

The Find and Replace stage uses exact string matching to standardize or remove a set of specific passages, phrases, or words from a document field.

This stage is configured with one or more rules. Each rule specifies:

- One or more text strings to search for matching text (phrases or words)
- A *single* text string that is the replacement value

When the replacement string is empty, the system deletes the matched text from the document field.

Configure the Find and Replace stage in one of the following ways:

- *Fusion blob store setup*
- *Fusion UI or REST API setup*

Fusion blob store setup

Setting up the stage using Fusion's blob store is referred to as **Find - List - Replace**.

The general procedure for *each* find/list/replace action is:

- Upload a file that contains a list of text passages to be standardized into the *Fusion Blob Store*.
- Each file:
 - Contains one line for each text passage
 - Has a `.lst` suffix

The required rule properties to standardize the text are the:

- Document field to scan
- Name of the `.lst` file uploaded to the blob store

The `.lst` suffix must be included in the file name designated in the rule.

- Value of the replacement text

Multiple `.lst` files can be uploaded and used to perform find/replace actions.

Example

One example could be to remove the country information and replace it with the phrase: **country not displayed for security purposes** to make the data anonymous for reporting.

- The document field to scan is **country**.
- The name of the `.lst` file in this example is `country.lst`. An excerpt is:

```
Afghanistan Afrique Albania Albanie Alderney Algeria Algérie Allemagne America  
Amérique Amériques American Samoa Andorra Andorre Angleterre Anglo-Normandes  
Angola Anguilla Antigua and Barbuda Antigua et Barbuda Antilles Antilles  
Néerlandaises Arabie Saoudite Argentina Argentine Arménie Aruba
```

- The value of the replacement text is: **country not displayed for security purposes**.

If the text of the passages scanned includes any of the items in the `country.lst` file, it is replaced with the phrase: **country not displayed for security purposes**.

Fusion UI or REST API setup

Setting up the stage using the Fusion UI or REST API is referred to as **Find - Replace**.

The Fusion UI access path is:

- Sign in to Fusion.
- Select the application.
- Select **Indexing > Index Workbench**.
- Select **Add a Stage**.

- Select **Find and Replace** in the Field Transformation section.
- Enter values in the fields as described in the *Configuration section*.

The REST API uses the parameters specified below.

The required rule properties to standardize the text are the:

- Document field to scan
- List of find/replace pairs

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Format Signals Index Stage

The Format Signals stage 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 is 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.

This stage does not define a list of allowed types.

Example Stage Specification

The Format Signals stage defined as part of the default 'signals_ingest' pipeline included with Fusion:

```
{ "type" : "format-signals", "id" : "ingest-signals", "flatten" : true,
  "undefinedType" : "general", "skip" : false, "label" : "format-signals", "type" :
  "format-signals" }
```

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

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 `https://FUSION_HOST:FUSION_PORT/data/nlp/gazetteer` (`https://FUSION_HOST:FUSION_PORT/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/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 USERNAME:PASSWORD -X PUT --data-binary @data/nlp/gazetteer/colors.lst -H
'Content-type: text/plain' http://localhost:8764/api/blobs/colors.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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

GeoIP Lookup Stage

This index pipeline stage looks up the geographical location of an IP address.

If the **IP Address Field** (`ipAddrField`) is specified, it will be used first to perform the geolocation lookup. If the lookup is successful, then the **Hostname Field** (`hostnameField`) will be skipped. However, if the lookup failed or the **IP Address Field** is blank, and the **Hostname Field** is specified, then the stage will try to perform the lookup using the hostname.

The default geo database is GeoLite2-City (<https://dev.maxmind.com/geoip/geoip2/geolite2/>) from MaxMind. To use a different database (for which you have a commercial license), you can *upload it to the blob store* (in binary format), then specify its blob ID in the **Geoip Lookup Database Blob ID** (`blobId`) parameter.

Fields added in this stage

When this stage is run, specific fields are added to the document. If the value is not present, the value added is `undefined`.

Field	Definition
<code>geo_loc_coord</code>	The latitude and longitude coordinates of the location. For example, 37.751,-97.822.
<code>geo_postal_code</code>	The postal code of the location.
<code>geo_country_name</code>	The country of the geographic location. For example, United States.
<code>geo_subdivision_iso</code>	The subdivision of the country in an ISO code format. For example, the ISO code for the state or province of the geographic location.
<code>geo_lon</code>	The longitude of the location. For example, -97.822.
<code>geo_country_iso</code>	The two-letter ISO code for the country of the location. For example, US.
<code>geo_subdivision_name</code>	The subdivision name of the location in the country. For example, the name of the state or province of the geographic location.
<code>geo_lat</code>	The latitude of the location. For example, 37.751.
<code>geo_city_name</code>	The city name of the location. For example, Chicago.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Include Documents Index Stage

This stage passes documents to the next stage in the pipeline if they match one or more of the specified rules (Boolean OR). If some field has multiple values then at least one value must match against specified pattern. All non-matching documents are dropped. Rules are defined using regular expression field matching.

Examples

Give the "simple-include" pipeline a stage that includes only certain document types:

```
curl -u USERNAME:PASSWORD -X POST -H "Content-type: application/json"
'http://localhost:8764/api/index-pipelines' -d '{ "id" : "simple-include",
"stages" : [ { "type" : "include-doc", "matchRules" : [ { "field" :
"document_type", "pattern" : "(xls|xlsx|xlst|doc|docx)" } ] } ] }'
```

Response:

```
{ "id" : "simple-include", "stages" : [ { "type" : "include-doc", "id" :
"f701f96b-780e-4355-9dd3-6e53a89afe3e", "matchRules" : [ { "field" :
"document_type", "pattern" : "(xls|xlsx|xlst|doc|docx)" } ], "type" : "include-
doc", "skip" : false, "label" : "include-doc" } ], "properties" : { } }
```

Send a text document through the "simple-include" pipeline:

```
curl -u USERNAME:PASSWORD 'http://localhost:8764/api/index-pipelines/simple-
include/collections/logs/index?simulate=true&echo=true' -H 'Content-type:
application/json' -d '{ "document_type": "txt" }'
```

The empty response indicates the document was dropped:

```
[ ]
```

Send an XLS document through the pipeline:

```
curl -u USERNAME:PASSWORD 'http://localhost:8764/api/index-pipelines/simple-
include/collections/logs/index?simulate=true&echo=true' -H 'Content-type:
application/json' -d '{ "document_type": "xls" }'
```

The response is document metadata, indicating the document passed the stage:

```
{ "id" : "9e7d1c2e-343a-49de-bc6a-1d1fc25fa93f", "fields" : [ { "name" :  
"document_type", "value" : "xls", "metadata" : { }, "annotations" : [ ] } ] } ]
```

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

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.

You must first *upload the JDBC driver to Fusion*.

Example

An example of a JDBC Index Stage setup

Upload stage config via POST to Fusion REST API endpoint ``api/index-stages/instances``

```
curl -u USERNAME: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/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 entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

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 entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Logging Index Stage

The Logging Index stage prints messages to the Connectors log file. The default location is:
https://FUSION_HOST:FUSION_PORT/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 entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Machine Learning Index Stage

The Fusion machine learning indexing stage uses a trained machine learning model to analyze a field or fields of a *PipelineDocument* and stores the results of analysis in a new field of either the PipelineDocument or *Context* object.

In order to use the Machine Learning Stage, you must train a machine learning model. There are two different ways to train a model:

- Use a Fusion AI job that trains a model, like *Logistic Regression* or *Random Forest*.
- Train a model using Spark's MLlib API (<https://spark.apache.org/docs/latest/mllib-guide.html>) outside of Fusion, and upload this model into Fusion's *blob store*. Complete details are available in *Machine Learning Models in Fusion*. TIP: When specifying field names, multiple field names are supported, in this format: `field1:weight,field2:weight,field3:weight`

Although this stage is available without a Fusion AI license, it is only effective after running the Fusion AI jobs mentioned above.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Managed JavaScript Index Stage

This stage runs a JavaScript transformation on documents using a script from the *blob store*. The script must have the blob type `file:js-index`.

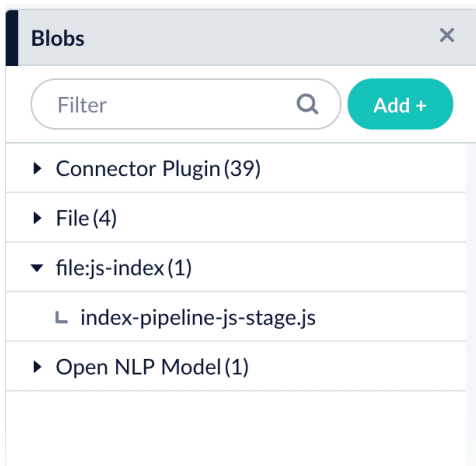
See also: *Custom JavaScript Stages For Index Pipelines*.

Uploading a JavaScript blob

In Fusion 4.1, JavaScript blobs must be uploaded using the *Blob Store API*, like this:

```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-Type: text/javascript' --data-binary @index-pipeline-js-stage.js http://localhost:8764/api/apps/APP_NAME/blobs/index-pipeline-js-stage.js?resourceType=file:js-index
```

Open the *blob manager* to verify that the new blob is present:



Referencing a JavaScript blob

Use any of the following formats for the `ref`/Script Reference configuration field:

```
index-pipeline-js-stage.js
```

```
blob:index-pipeline-js-stage.js
```

```
urn:x-fusion:blob:index-pipeline-js-stage.js
```

The examples above use `index-pipeline-js-stage.js` as the blob ID.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

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 field (the "target") with these entities.

This stage uses Apache OpenNLP (<http://opennlp.apache.org/>) project's Named Entity Recognition tool (<http://opennlp.apache.org/docs/#tools.namefind.recognition>) (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.

Fusion 4.x.x contains a common set of NER models for English that include sentence, token, and part-of-speech models. These models are:

Model	Purpose
<code>nlp/models/en-sent.bin</code>	Sentence model to detect sentences
<code>nlp/models/en-token.bin</code>	Tokenizer model for tokenization of sentences
<code>nlp/models/en-ner-date.bin</code>	Date name finder model
<code>nlp/models/en-ner-location.bin</code>	Location name finder model
<code>nlp/models/en-ner-money.bin</code>	Money name finder model
<code>nlp/models/en-ner-organization.bin</code>	Organization name finder model
<code>nlp/models/en-ner-percentage.bin</code>	Percentage name finder model
<code>nlp/models/en-ner-person.bin</code>	Person name finder model
<code>nlp/models/en-ner-time.bin</code>	Time name finder model

See OpenNLP 1.5 series (<http://opennlp.sourceforge.net/models-1.5/>) for additional pre-trained OpenNLP models.

To use these models, upload to Fusion using the *Fusion Blob Store service*. Here is an example of how to upload the sentence model file using the `curl` command-line utility, where "admin" is the name of a user with admin privileges, and "pass" is the password:

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

Example Specification

Specification of a stage which extracts names of people and places from field named 'body':

```
{ "type":"nlp-extractor", "id":"iqtr", "rules":[ { "source":[ "body_t" ],
"target":"organizations", "writeMode":"append",
"sentenceModelLocation":"nlp/models/en-sent.bin",
"tokenizerModelLocation":"nlp/models/en-token.bin", "entityTypes":[ {
"name":"organization", "definition":"nlp/models/en-ner-organization.bin" } ] }, {
"source":[ "body_t" ], "target":"persons", "writeMode":"append",
"sentenceModelLocation":"nlp/models/en-sent.bin",
"tokenizerModelLocation":"nlp/models/en-token.bin", "entityTypes":[ {
"name":"person", "definition":"nlp/models/en-ner-person.bin" } ] } ] },
"type":"nlp-extractor", "skip":false, "label":"Extract Entities",
"licensed":true, "secretSourceStageId":"iqtr" }
```

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

REST Query Index Stage

The REST Query index stage performs remote procedure calls to external services, merges the response from that service with a document being processed by the pipeline, and makes calls to the external system for each document. In Fusion version 3.0 and earlier, the stage is called the Query REST Remote Procedure Calls (RPC) Client stage.

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: `extservice://<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 / <code>uri</code>	A fully-qualified service URI. This can be an HTTP call, a Solr request, or a Fusion service call * any HTTP or HTTPS request: <code><http-protocol>://<path></code> * a Solr request: <code>solr://<collection>/...</code> * a Fusion service request: <code>extservice://<serviceName>/<path></code>
Call Method / <code>method</code>	The method to use for the RPC call. Supported methods are <code>GET</code> , <code>POST</code> , <code>PUT</code> , and <code>DELETE</code> .
Query parameters / <code>queryParams</code>	Query parameters to be passed with the request.
Request protocol headers / <code>headers</code>	Request protocol headers, such as <code>"Content-Type" : "application/json"</code> .
Request entity (as string) / <code>entity</code>	The request body. This parameter can only be configured via the REST API; <i>see the example below</i> .

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 (http://www.w3schools.com/xml/xpath_syntax.asp). It is 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 stopwords list:

```
{ "type" : "indexing-rpc", "id" : "demo", "debug" : true, "params" : { "uri" :
"http://localhost:8764/api/stopwords/movies", "method" : "PUT", "headers" : {
"Content-Type" : "application/json" }, "entity" : "New stopwords list" }, "skip" :
false, "label" : "Indexing RPC Demo", "type" : "indexing-rpc" }
```

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

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".

For examples of how to use this stage in the Fusion UI, see *Part 2* of the *Getting Started* tutorial.

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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

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 (<http://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html>).

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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Regex Field Replacement Index Stage

The Regex Field Replacement index stage lets you match against a source field and append or overwrite that field or another one with a replacement string. It uses the `java.util.regex` (<https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html>) library.

One notable feature is support for flags in the prefix of the pattern, such as `(?iu)\w+` where the `i` flag means case-insensitive and `u` means Unicode-aware case folding. You can also configure the desired behavior when there is no match. Additionally, you can use capture groups in your replacement configuration using `$1` to denote the first capture group.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

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" metadata, choose the name of 'creator' as defined in the creatorStrategy property for the field name. Only the last matching value will be retained.
- **CONCATENATE_UNIQUE**: choose to concatenate unique string values with a single space separator.
- **PICK_MIN**: choose the minimum value and discard all others. If the values are non-numeric, then all non-numeric values will be discarded.
- **PICK_MAX**: choose the maximum value and discard all others. If the values are non-numeric, then all non-numeric values will be discarded.

Configuration

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Send PagerDuty Message Index Stage

This stage sends a PagerDuty (<https://www.pagerduty.com/>) Message from Fusion, for alerting, monitoring, and more, using Fusion's *Messaging Services*.

Read more about PagerDuty integration in Fusion on our blog (<https://lucidworks.com/post/pagerduty-integration-in-lucidworks-fusion/>).

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 (<https://developer.pagerduty.com/documentation/integration/events/>)
 - 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 entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Send SMTP Email Index Stage

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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Send Slack Message Index Stage

This stage sends a Slack (<https://slack.com/>) 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 (<https://api.slack.com/docs/oauth>)
 - 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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Solr Dynamic Field Name Mapping

This stage can be used to re-name existing fields to match Solr dynamic field names, so that they will be typed correctly by Solr. It also can be used to specify fields which should be exempt from name changes as well as to set the dynamic type suffixes to a specified field. Note that the field that is processed by a Field Mapping stage prior to this stage will not be renamed, even if the stage config matches the field name.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Solr Indexer Stage

A Solr Indexer stage transforms a Fusion PipelineDocument 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 Index stage* offers greater control over date values.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

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.

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

(<https://cwiki.apache.org/confluence/display/solr/Updating+Parts+of+Documents>). 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.

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 (<https://wiki.apache.org/solr/UniqueKey>) 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 (<http://yonik.com/solr/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 requires 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>
(<https://cwiki.apache.org/confluence/display/solr/Working+with+Dates>)

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

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 (<http://opennlp.apache.org/>) project's Part of Speech Tagger (<http://opennlp.apache.org/docs/#tools.postagger>) to mark tokens with their corresponding word type based on the token itself and the context of the token. The OpenNLP documentation states:

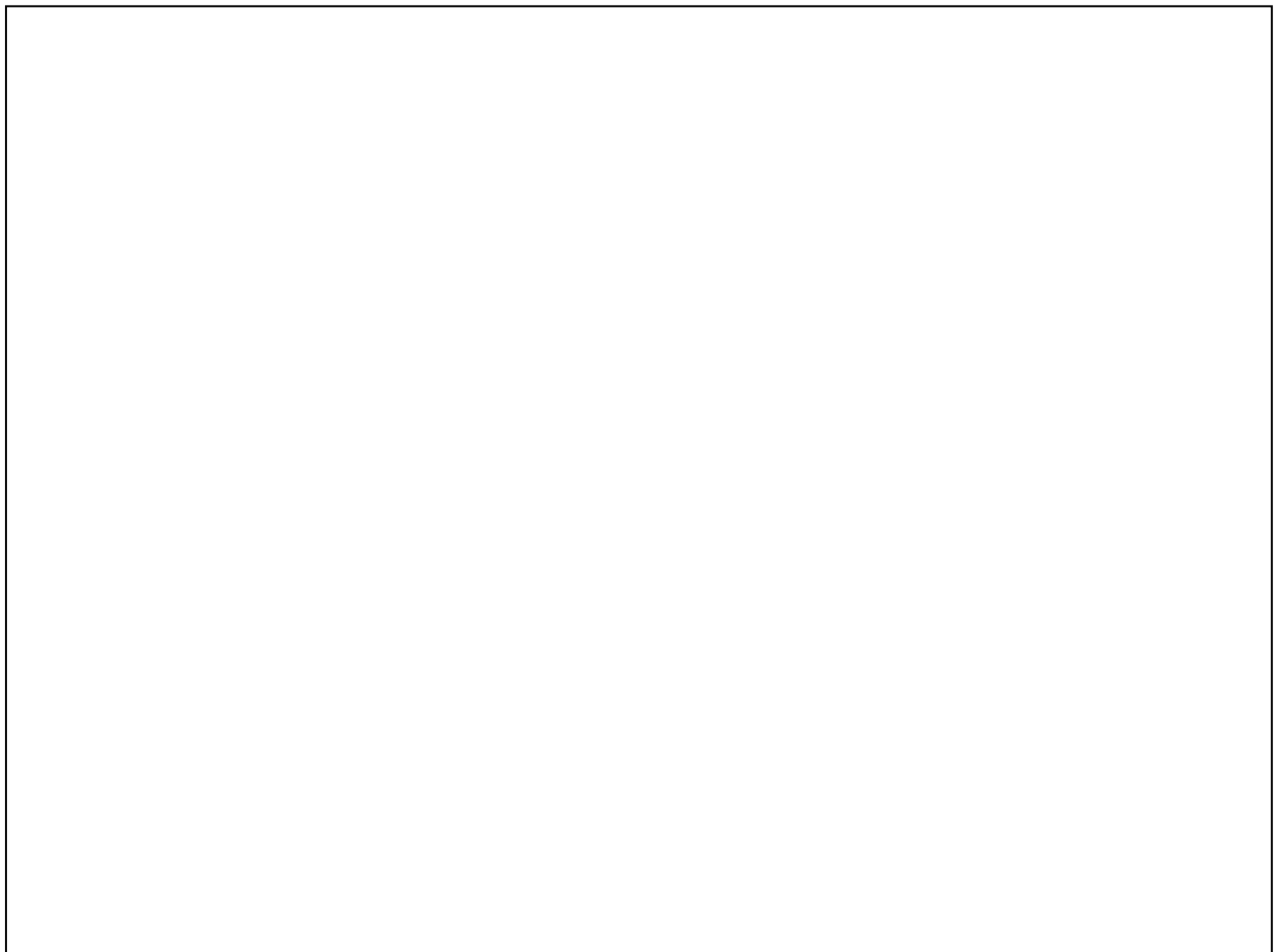
"A token might have multiple pos tags depending on the token and the context. The OpenNLP POS Tagger uses a probability model to predict the correct pos tag out of the tag set. To limit the possible tags for a token a tag dictionary can be used which increases the tagging and runtime performance of the tagger."

Fusion comes with a set of OpenNLP language models for english. These data files are found in the directory: https://FUSION_HOST:FUSION_PORT/data/nlp/models.

Models are available from the OpenNLP models SourceForge repository (<http://opennlp.sourceforge.net/models-1.5/>). 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 USERNAME:PASSWORD -X PUT --data-binary @en-pos-maxent.bin -H 'Content-type: text/plain' http://localhost:8764/api/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 USERNAME:PASSWORD -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/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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Update Related Document Stage

This index pipeline stage performs a Solr atomic update on an existing document using data pulled from the current document being processed by the pipeline.

Configuration

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

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 (http://lucene.apache.org/solr/6_1_0/solr-dataimport/org/apache/solr/handler/dataimport/XPathRecordReader.html) 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//...
```

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 entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Jobs Configuration

These reference topics provide complete information about the configuration properties of jobs for which the subtype is "task" or "spark".

Jobs with the subtype "datasource" have configuration schemas that depend on the connector type; see *Connectors Configuration Reference*.

For conceptual information and instructions for configuring and scheduling jobs, see *Jobs and Schedules*.

Additional Spark jobs are available with a <i>Fusion AI</i> license.

Tasks

- *Download Blob*
- *Log Cleanup*
Delete old log messages from system logs collection.
- *REST Call*
A versatile job type that runs an arbitrary REST/HTTP/Solr command.

Spark jobs

- *Aggregation*
Define an aggregation job.
- *Script*
Run a custom Scala script as a Fusion job.

Aggregation Jobs

Define an aggregation job.

Custom Spark Jobs

Run a custom Spark job.

Download Blob

Download and update a blob.

Log Cleanup Jobs

Delete old log messages from system logs collection.

REST Call Jobs

A versatile job type that runs an arbitrary REST/HTTP/Solr command.

REST jobs can call external systems using:

- any HTTP or HTTPS request: `<http-protocol>://<path>`; requires JWT header.
- a Solr request: `solr://COLLECTION_NAME/...`; authentication is handled.
- a Fusion service request: `extservice://SERVICE_NAME/<path>`; authentication is handled.

For example, you can configure this type of job to delete your old signals data:

1. Navigate to **Collections > Jobs**.
2. Click **Add** and select **REST Call**.
The REST Call job configuration panel appears.
3. Enter an arbitrary ID for this job, such as "Delete-old-signals".
4. Enter the following endpoint URI, substituting the name of your signals collection for `signalsCollectionName`:

```
solr://signalsCollectionName/update
```

5. In the **Call Method** field, select "post".
6. Under **Query Parameters**, enter the property name "wt" with the property value "json".
7. In the **Request entity (as string)** field, enter the following:

```
<root><delete><query>timestamp_tdt:[* TO NOW-2WEEKS]</query></delete><commit /></root>
```

See [Working with Dates \(https://solr.apache.org/guide/8_6/working-with-dates.html\)](https://solr.apache.org/guide/8_6/working-with-dates.html) for details about date formatting.

Your job configuration should look similar to this:

Run  Job History 

Cancel Save Explore 

Delete-old-signals
Run arbitrary REST/HTTP/Solr command

ID *
Delete-old-signals

ENDPOINT URI *
solr://Movie_Search_signals/update

CALL METHOD *
post
One of GET, POST, PUT, or DELETE

QUERY PARAMETERS

 Property Name

Property Value

 wt

json

REQUEST PROTOCOL HEADERS

 Property Name

Property Value

REQUEST ENTITY (AS STRING)

```
<root><delete><query>timestamp_tdt:[* TO NOW-6MONTHS]</query></delete><commit /></root>
```

You can configure a schedule for this job at **System > Scheduler**.

Script Jobs

Run a custom Scala script as a Fusion job.

Parsers

Parsers provide fine-grained configuration for inbound data. You configure parsers with stages, much like index pipelines and query pipelines. Parsers can include conditional parsing and nested parsing. You can configure them through the *Fusion UI* or the *Parsers API*.

Connectors receive the inbound data, convert it into a byte stream, and send the byte stream to a parser's configured parsing stages. The parser selects a parsing stage to handle the stream, which parses the data and produces documents that are sent to the index pipeline.

Each parsing stage evaluates whether the inbound stream matches the stage's default media types or filename extensions. The first stage that finds a match processes the data and can output one or both of the following:

- Zero or more pipeline documents for consumption by the index pipeline
- Zero or more new input streams for re-parsing
This recursive approach is useful for containers (for example, `zip` and `tar` files). The output of the container parsing can be another container or a stream of uncompressed content that requires its own parsing.

Stages that might match the stream beyond the first match will not be used.

A few static fields impact the overall parser configuration. They are accessible when you select the parser in the Index Workbench:

Field	Description
Document ID Source Field	Field in the source file that contains the document ID
Maximum Parser Recursion Depth	Maximum number of times the parser may recurse over the file, before proceeding to the next parser. This is useful for files with hierarchical structures (for example, <code>zip</code> and <code>tar</code> files).
Enable automatic media type detection	Whether to automatically detect the media type of the source files. If disabled, the parser uses the media type <code>application/octet-stream</code> .

Built-in parser stages

These parser stages are available for configuration:

- *Apache Tika parser stage*
- *Archive parser stage*
- *CSV parser stage*
- *Fallback parser stage*
- *Grok parser stage*
- *HTML parser stage*
- *JSON parser stage*
- *Text parser stage*
- *XML parser stage*

Datasources that use connectors that retrieve fixed-structure content (such as those for Twitter and Jira) have hard-coded parsers and do not expose any configurable parser options.

Configure parsers

When you configure a datasource, you can use the *Index Workbench* or the *Parsers API* to create a parser. A parser consists of an ordered list of parser stages, some global parser parameters, and the stage-specific parameters. You can re-order the stages list by dragging them up or down in the Index Workbench.

Any parser stage can be added to the same parser multiple times if different configuration options are needed for different stages. Datasources with fixed-structure data will also be parsed by Fusion, but with default settings that do not need to be customized.

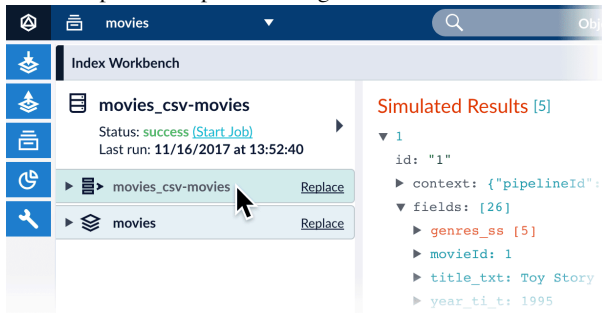
There is no limit to the number of stages that can be included in a parser. The priority-order of the stages is completely flexible. In a default parser configuration, a fallback parser is provided at the end of the parsing stage list to handle streams no other stage matches. If present, this stage is selected and attempts to parse anything that has not yet been matched.

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

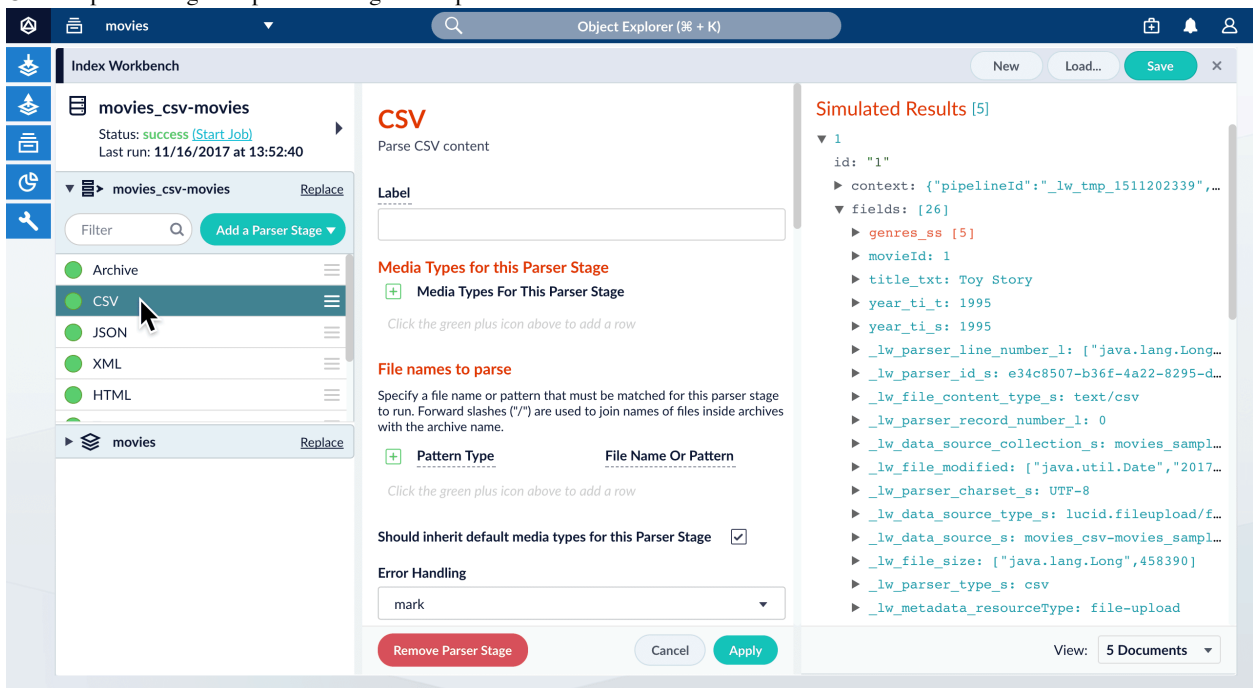
Configure a parser in the Fusion UI

To configure parser stages using the Fusion UI:

1. In the *Fusion workspace*, navigate to the *Index Workbench*.
2. At the upper right of the Index Workbench panel, click **Load**.
3. Under **Load**, click the name of the index pipeline.
4. Click the parser to open its configuration:



5. Click a specific stage to open its configuration panel:



Configure a parser in the REST API

The *Parsers API* provides a programmatic interface for viewing, creating, and modifying parsers, as well as sending documents directly to a parser.

- To get all currently-defined parsers: <http://localhost:8764/api/parsers/>
- To get the parser schema: http://localhost:8764/api/parsers/_schema

Here is a very simple parser example, for parsing JSON input:

```
{ "id": "simple-json", "type": "json", arbitrary parser-specific options here.
  "pretty": false }
```

The example below shows a parser that can parse JSON input, as well as JSON that is inside zip, tar, or gzip containers, or any combination (such as .tar.gz). The order of the stages begins with the outermost containers and ends with the innermost content.

```
{ "id": "default-json", "type": "composite", "parsers": [ { "id" : "zip-parser",  
"type" : "zip" }, { "type" : "gz" }, { "type" : "tar" }, { "id": "json-parser",  
"type": "json", "prettify": false } ] }
```

ID is optional, just as in pipeline stages. Many parser stages require no configuration other than `type`.

Field parser index pipeline stage

The parsers themselves only parse whole documents. Parsing of content embedded in fields is performed separately by the Field Parser index pipeline stage. This stage identifies the field that requires parsing and the parser to use.

Apache Tika Parser Stage

Apache Tika is a versatile parser that supports many types of unstructured document formats, such as HTML, PDF, Microsoft Office documents, OpenOffice, RTF, audio, video, images, and more. A complete list of supported formats is available at <http://tika.apache.org/> (<http://tika.apache.org/>).

To perform image text extraction when **Include images** is enabled, Tesseract (<https://github.com/tesseract-ocr/tesseract>) should be installed in the server hosting Fusion.

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Archive Parser Stage

The Archive parser stage can parse the majority of common archive and compressed file formats. They are parsed into their constituent documents, which can then be parsed further or sent straight to the index pipeline. The following archive formats are supported:

- tar
- zip
- jar
- 7z
- ar
- arj
- Unix dump
- cpio

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

CSV Parser Stage

This parser breaks down incoming CSV files into the most efficient components for Fusion to index. It produces one new document per row from the CSV input, excluding comment rows and header rows.

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Enable Tika Asynchronous Parsing

When you use the Tika Asynchronous Parser, the connector fetches and indexes a document containing only metadata. It does not index the Tika parsed metadata and Tika parsed body.

A separate Tika Asynchronous parsing job will fetch the raw file content, parse it with Tika and do a partial update on the document with both the Tika parsed metadata and Tika parsed body.

See Fusion 4.x V1 Connector Downloads (<https://doc.lucidworks.com/fusion-connectors/4.2/5e5d7o/fusion-4-x-v-1-connector-downloads>) to access the latest versions of the connectors.

Configure the index pipelines

When you use asynchronous parsing, you will need to establish two index pipelines.

Pre-parse pipeline

The pre-parse pipeline will only fetch and index metadata since it does not have the Tika parsed body or Tika parsed metadata. Within this index pipeline, you will want to make sure you are buffering requests to Solr:

The screenshot shows the configuration interface for the 'Solr Indexer' stage. The left sidebar lists the stages: Field Mapping, Solr Dynamic Field Name Mapping, js test, and Solr Indexer (selected). The main panel shows the configuration for the 'Solr Indexer' stage. The 'Condition' field is empty. The 'Map to Solr Schema' checkbox is checked. The 'Fields' section has an 'OPEN FIELDS PANEL' link. The 'Additional Date Formats' section has a green plus icon and the text 'Click the green plus icon above to add a row'. The 'Additional Update Request Parameters' section has a green plus icon and the text 'Click the green plus icon above to add a row'. The 'Buffer Documents and Send Them to Solr in Batches' checkbox is checked. The 'Unmapped Fields Mapping' checkbox is unchecked.

This is the default setting. If this option is not selected, select the box. Doing so will increase the indexing speed.

Post-parse pipeline

The Tika Asynchronous job will download, parse, and send the parsed document to a second Fusion index pipeline. This pipeline is responsible for updating the Tika body and metadata to the document created in the pre-parse pipeline.

The index pipeline **MUST** contain a Solr Partial Update stage with the following selected parameters:

Pipeline ID

partial-update

Stages

Add a new pipeline stage
Remove stage

Field Mapping

Solr Partial Update Indexer

Solr Partial Update Indexer

This stage indexes partial (atomic) updates to Solr documents

Label

A unique label for this stage.

Condition

Define a conditional script that must result in true or false. This can be used to determine if the stage should process or ..

Map to Solr Schema ☐

Enable Concurrency Control ☒

Reject Update if Solr Document is not Present ☐

Buffer Documents and Send Them to Solr in Batches ☒

Process All Pipeline Doc Fields ☒

*** Solr Document ID Field Name**

id

The Solr Partial Update Indexer pipeline will add the Tika fields into the existing document that was added by the connector.

If you do not select the **Process All Pipeline Doc Fields** option, you will receive results similar to the following:

```
"body_t":[{"field\":"I'm the content from JSON!\"}]"
```

Configure the Asynchronous Parser

Next, we will create the new parser. All settings can be left on the default selections.

Parser

* Parser ID

tika-async

Document ID Source Field

A document field to use as the document ID

Maximum Parser Recursion Depth

16

Maximum number of times a parser may recurse over any...

Enable automatic media type detection



Filter



Add a parser stage ▼

Remove



Apache Tika Async



Apache Tika Async

A parser that is used for parsing content using the Tika Async Parser. The Tika Async parser runs in the background of the tika-server services and will

Label

A label for this Parser Stage

Media Types to match



Media Types To Match

Click the green plus icon above to add a row

Match default media types in this Parser Stage



File names to parse

Specify a file name or pattern that must be matched for this parser stage to run. Forward slashes ("/") are used to join names of files inside archives with the archive name.

Configure the datasource

Configure the datasource to use the asynchronous parser.

Run



Job History



Clear Datasource

Cancel

Save

dsid (Sharepoint-optimized)

A crawler for SharePoint- using SharePoint Rest.

Advanced ☐

DATASOURCE ID *

dsid

Unique name for this datasource.

PIPELINE ID *

test

Name of an existing index pipeline for processin...



PARSER

tika-async-parser

Parser used when parsing raw content. Retry pa...



Description

Optional description for this datasource.

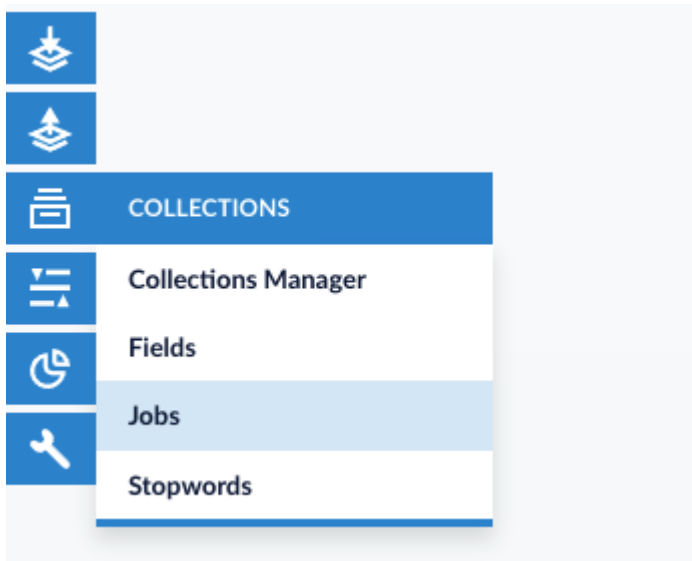
When you run the datasource job, it will display the documents indexed and will have a special async parsing metadata.

- `_lw_async_parsing_id_i`
- `_lw_async_parsing_fail_count_i`

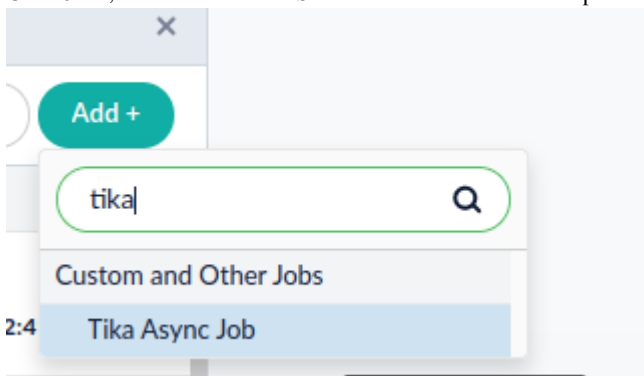
If the connectors have an async parser, they will not download or parse the documents when running the connector job. This will be performed asynchronously by the Tika Asynchronous Parsing job.

Configure the Tika Async Parsing job

1. Select **Jobs** under the Collections header.



2. Click Jobs, and select **Add +**. Search for Tika to select the option for Tika Async Job.



3. Configure fields according to your desired specifications.

Run 

Job History 

Cancel

Save

Explore 

async (Tika Async Job)
Task that runs the Tika Asynchronous job

TIKA ENDPOINTS

Add

Auto updated list of detected tika nodes. Do not require to be modified manually.

http://127.0.1.1:8889	

PIPELINE ID *

partial

Name of an existing index pipeline for processing documents.

FINDER FILTER QUERIES

Add

Optional solr filter queries to use to limit the docs we pull in to parse asynchronously.

FINDER BATCH SIZE 500	MAX BYTES RETURNED 100000000	MAX METADATA FIELD BYTES 30000
Sets the maximum amount of bytes that can be ret...	Sets the maximum amount of bytes that can be sto...	
MAX EMBEDDED DOCUMENTS -1	PARSE TIMEOUT (MS) 15000	MAX PARSE FAILURES 2
The maximum number of embedded documents to...	The maximum amount of time in milliseconds that ...	If a document fails to be parsed, the subsequent tik...
INSERT BATCH MAX DOCS 100	INSERT BATCH MAX BYTES 50000000	

Field options for the Tika Async parsing job include:

Field	Notes
Tika Endpoints	List of Tika server endpoints that are in the cluster. The endpoints will be automatically updated. You do not need to specify the endpoints.
Pipeline ID	The Pipeline ID is determined by the partial-update index pipeline you created earlier.
Find Filter Queries	This is an optional filter to limit the documents fetched by the Tika Async parser.
Finder batch size	This is the Solr <code>rows</code> batch size of the requests.
Max bytes returned	The maximum number of characters indexed.
Max embedded documents	This determines the amount of embedded documents that should be parsed until additional embedded documents are ignored. Use -1 for unlimited documents.
Parse timeout	This configures the length of time for parsing in milliseconds.
Max Parse Failure	If a document fails to parse, you can configure the reattempts up to your designated amount of times.
Insert Batch Max Docs	This controls the batch size in number of documents before submitting to the index pipeline.
Insert Batch Max Bytes	This controls the batch size number of bytes contained in the documents before submitting to the index pipeline.

4. Finally, configure a schedule for the job to run according to your designated specifications.

Run 

Job History 

Cancel

 Success

Start

Next Run: Tue, Nov 30, 2021 at 01:21:00 AM -0600





Start Time

2021-11-23 00:00:00 



Interval

3 

Time Unit

minute 

New Schedule 

Cancel

Save

The logs for the job will be in `$FUSION_HOME/var/log/tika-server/tika-async.log`

Fallback Parser Stage

The Fallback parser stage is useful for processing data that Fusion does not have a specified parsing process for. Fallback does not technically parse data, since it does not know what to do with it, it simply copies the raw bytes into a Solr document. If your Fusion parser stage configuration encounters data it does not know how to parse, such as someone's proprietary data file format, it will copy it as-is, whereas if it encounters recognizable data in more common file types, such as PDFs, Fusion will parse the text and metadata using Tika.

The Fallback parser acts as the final stage that attempts to parse any documents that haven't been parsed already. More commonly, the parser selects a stage specifically built to handle the data and runs that stage.

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Grok Parser Stage

The Grok parser stage uses Java Grok (<https://github.com/thekraken/java-grok>) and Grok patterns (a specific kind of regex matching) to parse log files and similar text files that have line-oriented, semi-structured data. Parsing a text file with the Grok parser lets you give more structure to semi-structured data and extract more information.

Whether the Grok stage parses a file

Before a Grok parser stage parses a file, the file must meet criteria regarding the media type and file name.

Media type

The Grok parser stage parses files that have media types that match either the default media types *or* media types that you specify.

Select or unselect **Use default media types for this parser stage**:

- Selected.* The Grok parser stage parses files that have one of the default media types (`text/plain` or `text/x-log`), as well as files that have media types that you specify under **Media Types for this Parser Stage**.
- Unselected.* The Grok parser stage *only* parses files that have one of the media types that you specify under **Media Types for this Parser Stage**.

File name

Optionally, you can specify a file name or file name pattern that a file must match for the Grok parser stage to parse the file.

Field	Description
Pattern Type	<code>glob</code> . Use <code>bash</code> shell wildcards. Examples include <code>z.txt</code> , <code>*.md</code> , and <code>/a/*/b/f.txt</code> . <code>regex</code> . Use Java regular expressions (PCRE; Perl-compatible regular expressions). Examples include <code>z.txt\$</code> , <code>.*\.txt\$</code> , and <code>^/a/[^\/*]/b/f.txt\$</code> .
File Name or Pattern	Name of the file or a pattern for the file name. The parser parses matching files.

Grok patterns

Grok patterns are regular expressions written in the language of the Oniguruma regular expression library (<https://github.com/kkos/oniguruma>), which has this syntax (<https://github.com/kkos/oniguruma/blob/master/doc/RE>).

You configure a Grok parsing stage to use predefined *Grok patterns* (about 300 patterns are available) and/or Grok pattern definitions that you write yourself.

- Use predefined patterns.** Under the **Grok Pattern** part of the Grok parser stage configuration, specify a single top-level Grok pattern by name, for example, `REDISLOG`.
- Write your own Grok pattern definition(s).** (optional) Write one or more Grok pattern definitions, and then enter them in the **Grok Definition** part of the Grok parser stage configuration.

Parsing rules

These are rules that affect the results of parsing:

- Precedence in the event of identical names.** If the name of a custom Grok pattern definition that you provide is identical to the name of a predefined pattern definition, then your definition is used.
- Invalid patterns.** If a pattern is not syntactically valid, then the full text of the row being parsed is treated as a single field.
- Pattern does not match any data.** If a pattern does not match any data, then the full text of the row being parsed is treated as a single field.
- Line by line.** Parsing is line by line. If data has a multiline structure, the parser does not capture the relationship between lines.

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Grok Patterns

Below we list predefined Grok patterns that you can use with the Grok parser. The patterns are grouped by the kinds of files in which they occur.

Solr

```
HANDLER %{WORD}.*{WORD}.*{WORD}.*{WORD} SOLR_QUERY \[collection%
{NUMBER:collection}\]\s+webapp=/{WORD:webapp}\s+path=/{WORD:path}\s+params=\{%
{GREEDYDATA:params}\}\s+hits={NUMBER:hits}\s+status={NUMBER:status}\s+QTime=%
{NUMBER:QTime} SOLR_LOG %{LOGLEVEL}\s+-\s+{TIMESTAMP_ISO8601};\s+{JAVACLASS};\s+%
{MESSAGE:message} SOLR_6_LOG4J %{TIMESTAMP_ISO8601:logdate} %{LOGLEVEL:level_s} \
(%{DATA:thread_s}\) \[(?:%{DATA:mdc_s}| )\] %{DATA:category_s} %
{JAVALOGMESSAGE:logmessage} SOLR_6_REQUEST \[%{DATA:core_node_name_s}\]
webapp=/{WORD:webapp_s} path=/{DATA:path_s} params=\{%{GREEDYDATA:params_s}\}
hits={NUMBER:hits_i} status={NUMBER:status_i} QTime={NUMBER:qtime_i}
```

Ruby

```
RUBY_LOGLEVEL (?:DEBUG|FATAL|ERROR|WARN|INFO) RUBY_LOGGER [DFEWI], \[%
{TIMESTAMP_ISO8601:timestamp} #{POSINT:pid}\] *%{RUBY_LOGLEVEL:loglevel} -- +%
{DATA:programe}: %{GREEDYDATA:message} REDISTIMESTAMP %{MONTHDAY} %{MONTH} %
{TIME} REDISLOG \[%{POSINT:pid}\] %{REDISTIMESTAMP:timestamp} \*
```

Rails

```
RUUID \h{32} # rails controller with action RCONTROLLER (?<controller>[^#]+)#(?
<action>\w+) # this will often be the only line: RAILS3HEAD (?m)Started %
{WORD:verb} "%{URIPATHPARAM:request}" for %{IPORHOST:clientip} at (?<timestamp>%
{YEAR}-%{MONTHNUM}-%{MONTHDAY} %{HOUR}:%{MINUTE}:%{SECOND} %{ISO8601_TIMEZONE}) #
for some a strange reason, params are stripped of {} – not sure that is a good
idea. RPROCESSING \W*Processing by %{RCONTROLLER} as (?<format>\S+)
(?:\W*Parameters: %{DATA:params}\W*)? RAILS3FOOT Completed %{NUMBER:response}%
{DATA} in %{NUMBER:totalms}ms %{RAILS3PROFILE}%{GREEDYDATA} RAILS3PROFILE (?:\
Views: %{NUMBER:viewms}ms \| ActiveRecord: %{NUMBER:activerecordms}ms\|
(ActiveRecord: %{NUMBER:activerecordms}ms)? # putting it all together RAILS3 %
{RAILS3HEAD}(?:%{RPROCESSING})?(?<context>(?:%{DATA}\n)*) (?:%{RAILS3FOOT})?
```

Postgresql

```
# Default postgresql pg_log format pattern POSTGRESQL %{DATESTAMP:timestamp} %
{TZ} %{DATA:user_id} %{GREEDYDATA:connection_id} %{POSINT:pid}
```

```
#####
#
#####
# # Chop Nagios log files to smithereens! # # A set of GROK filters to process
logfiles generated by Nagios. # While it does not, this set intends to cover all
possible Nagios logs. # # Some more work needs to be done to cover all External
Commands: # http://old.nagios.org/developerinfo/externalcommands/commandlist.php
# # If you need some support on these rules please contact: # Jelle Smet
http://smetj.net #
#####
#####
NAGIOSTIME \[%{NUMBER:nagios_epoch}\]
##### Begin nagios log types
##### NAGIOS_TYPE_CURRENT_SERVICE_STATE
CURRENT SERVICE STATE NAGIOS_TYPE_CURRENT_HOST_STATE CURRENT HOST STATE
NAGIOS_TYPE_SERVICE_NOTIFICATION SERVICE NOTIFICATION
NAGIOS_TYPE_HOST_NOTIFICATION HOST NOTIFICATION NAGIOS_TYPE_SERVICE_ALERT SERVICE
ALERT NAGIOS_TYPE_HOST_ALERT HOST ALERT NAGIOS_TYPE_SERVICE_FLAPPING_ALERT
SERVICE FLAPPING ALERT NAGIOS_TYPE_HOST_FLAPPING_ALERT HOST FLAPPING ALERT
NAGIOS_TYPE_SERVICE_DOWNTIME_ALERT SERVICE DOWNTIME ALERT
NAGIOS_TYPE_HOST_DOWNTIME_ALERT HOST DOWNTIME ALERT
NAGIOS_TYPE_PASSIVE_SERVICE_CHECK PASSIVE SERVICE CHECK
NAGIOS_TYPE_PASSIVE_HOST_CHECK PASSIVE HOST CHECK
NAGIOS_TYPE_SERVICE_EVENT_HANDLER SERVICE EVENT HANDLER
NAGIOS_TYPE_HOST_EVENT_HANDLER HOST EVENT HANDLER NAGIOS_TYPE_EXTERNAL_COMMAND
EXTERNAL COMMAND NAGIOS_TYPE_TIMEPERIOD_TRANSITION TIMEPERIOD TRANSITION
##### End nagios log types
#####
##### Begin external check
types ##### NAGIOS_EC_DISABLE_SVC_CHECK
DISABLE_SVC_CHECK NAGIOS_EC_ENABLE_SVC_CHECK ENABLE_SVC_CHECK
NAGIOS_EC_DISABLE_HOST_CHECK DISABLE_HOST_CHECK NAGIOS_EC_ENABLE_HOST_CHECK
ENABLE_HOST_CHECK NAGIOS_EC_PROCESS_SERVICE_CHECK_RESULT
PROCESS_SERVICE_CHECK_RESULT NAGIOS_EC_PROCESS_HOST_CHECK_RESULT
PROCESS_HOST_CHECK_RESULT NAGIOS_EC_SCHEDULE_SERVICE_DOWNTIME
SCHEDULE_SERVICE_DOWNTIME NAGIOS_EC_SCHEDULE_HOST_DOWNTIME SCHEDULE_HOST_DOWNTIME
NAGIOS_EC_DISABLE_HOST_SVC_NOTIFICATIONS DISABLE_HOST_SVC_NOTIFICATIONS
NAGIOS_EC_ENABLE_HOST_SVC_NOTIFICATIONS ENABLE_HOST_SVC_NOTIFICATIONS
NAGIOS_EC_DISABLE_HOST_NOTIFICATIONS DISABLE_HOST_NOTIFICATIONS
NAGIOS_EC_ENABLE_HOST_NOTIFICATIONS ENABLE_HOST_NOTIFICATIONS
NAGIOS_EC_DISABLE_SVC_NOTIFICATIONS DISABLE_SVC_NOTIFICATIONS
NAGIOS_EC_ENABLE_SVC_NOTIFICATIONS ENABLE_SVC_NOTIFICATIONS
##### End external check types
##### NAGIOS_WARNING Warning:%{SPACE}%
{GREEDYDATA:nagios_message} NAGIOS_CURRENT_SERVICE_STATE %
{NAGIOS_TYPE_CURRENT_SERVICE_STATE:nagios_type}: %{DATA:nagios_hostname};%
{DATA:nagios_service};%{DATA:nagios_state};%{DATA:nagios_statetype};%
{DATA:nagios_statecode};%{GREEDYDATA:nagios_message} NAGIOS_CURRENT_HOST_STATE %
{NAGIOS_TYPE_CURRENT_HOST_STATE:nagios_type}: %{DATA:nagios_hostname};%
{DATA:nagios_state};%{DATA:nagios_statetype};%{DATA:nagios_statecode};%
{GREEDYDATA:nagios_message} NAGIOS_SERVICE_NOTIFICATION %
{NAGIOS_TYPE_SERVICE_NOTIFICATION:nagios_type}: %{DATA:nagios_notifname};%
{DATA:nagios_hostname};%{DATA:nagios_service};%{DATA:nagios_state};%
{DATA:nagios_contact};%{GREEDYDATA:nagios_message} NAGIOS_HOST_NOTIFICATION %
{NAGIOS_TYPE_HOST_NOTIFICATION:nagios_type}: %{DATA:nagios_notifname};%
{DATA:nagios_hostname};%{DATA:nagios_state};%{DATA:nagios_contact};%
{GREEDYDATA:nagios_message} NAGIOS_SERVICE_ALERT %
{NAGIOS_TYPE_SERVICE_ALERT:nagios_type}: %{DATA:nagios_hostname};%
{DATA:nagios_service};%{DATA:nagios_state};%{DATA:nagios_statelevel};%
{NUMBER:nagios_attempt};%{GREEDYDATA:nagios_message} NAGIOS_HOST_ALERT %
{NAGIOS_TYPE_HOST_ALERT:nagios_type}: %{DATA:nagios_hostname};%
{DATA:nagios_state};%{DATA:nagios_statelevel};%{NUMBER:nagios_attempt};%
```



```

{GREEDYDATA:nagios_message} NAGIOS_SERVICE_FLAPPING_ALERT %
{NAGIOS_TYPE_SERVICE_FLAPPING_ALERT:nagios_type}: %{DATA:nagios_hostname};%
{DATA:nagios_service};%{DATA:nagios_state};%{GREEDYDATA:nagios_message}
NAGIOS_HOST_FLAPPING_ALERT %{NAGIOS_TYPE_HOST_FLAPPING_ALERT:nagios_type}: %
{DATA:nagios_hostname};%{DATA:nagios_state};%{GREEDYDATA:nagios_message}
NAGIOS_SERVICE_DOWNTIME_ALERT %{NAGIOS_TYPE_SERVICE_DOWNTIME_ALERT:nagios_type}:
%{DATA:nagios_hostname};%{DATA:nagios_service};%{DATA:nagios_state};%
{GREEDYDATA:nagios_comment} NAGIOS_HOST_DOWNTIME_ALERT %
{NAGIOS_TYPE_HOST_DOWNTIME_ALERT:nagios_type}: %{DATA:nagios_hostname};%
{DATA:nagios_state};%{GREEDYDATA:nagios_comment} NAGIOS_PASSIVE_SERVICE_CHECK %
{NAGIOS_TYPE_PASSIVE_SERVICE_CHECK:nagios_type}: %{DATA:nagios_hostname};%
{DATA:nagios_service};%{DATA:nagios_state};%{GREEDYDATA:nagios_comment}
NAGIOS_PASSIVE_HOST_CHECK %{NAGIOS_TYPE_PASSIVE_HOST_CHECK:nagios_type}: %
{DATA:nagios_hostname};%{DATA:nagios_state};%{GREEDYDATA:nagios_comment}
NAGIOS_SERVICE_EVENT_HANDLER %{NAGIOS_TYPE_SERVICE_EVENT_HANDLER:nagios_type}: %
{DATA:nagios_hostname};%{DATA:nagios_service};%{DATA:nagios_state};%
{DATA:nagios_statelevel};%{DATA:nagios_event_handler_name}
NAGIOS_HOST_EVENT_HANDLER %{NAGIOS_TYPE_HOST_EVENT_HANDLER:nagios_type}: %
{DATA:nagios_hostname};%{DATA:nagios_state};%{DATA:nagios_statelevel};%
{DATA:nagios_event_handler_name} NAGIOS_TIMEPERIOD_TRANSITION %
{NAGIOS_TYPE_TIMEPERIOD_TRANSITION:nagios_type}: %{DATA:nagios_service};%
{DATA:nagios_unknown1};%{DATA:nagios_unknown2} ##### External
checks ##### #Disable host & service check
NAGIOS_EC_LINE_DISABLE_SVC_CHECK %{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_DISABLE_SVC_CHECK:nagios_command};%{DATA:nagios_hostname};%
{DATA:nagios_service} NAGIOS_EC_LINE_DISABLE_HOST_CHECK %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_DISABLE_HOST_CHECK:nagios_command};%{DATA:nagios_hostname} #Enable
host & service check NAGIOS_EC_LINE_ENABLE_SVC_CHECK %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_ENABLE_SVC_CHECK:nagios_command};%{DATA:nagios_hostname};%
{DATA:nagios_service} NAGIOS_EC_LINE_ENABLE_HOST_CHECK %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_ENABLE_HOST_CHECK:nagios_command};%{DATA:nagios_hostname} #Process
host & service check NAGIOS_EC_LINE_PROCESS_SERVICE_CHECK_RESULT %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_PROCESS_SERVICE_CHECK_RESULT:nagios_command};%{DATA:nagios_hostname};%
{DATA:nagios_service};%{DATA:nagios_state};%{GREEDYDATA:nagios_check_result}
NAGIOS_EC_LINE_PROCESS_HOST_CHECK_RESULT %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_PROCESS_HOST_CHECK_RESULT:nagios_command};%{DATA:nagios_hostname};%
{DATA:nagios_state};%{GREEDYDATA:nagios_check_result} #Disable host & service
notifications NAGIOS_EC_LINE_DISABLE_HOST_SVC_NOTIFICATIONS %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_DISABLE_HOST_SVC_NOTIFICATIONS:nagios_command};%
{GREEDYDATA:nagios_hostname} NAGIOS_EC_LINE_DISABLE_HOST_NOTIFICATIONS %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_DISABLE_HOST_NOTIFICATIONS:nagios_command};%
{GREEDYDATA:nagios_hostname} NAGIOS_EC_LINE_DISABLE_SVC_NOTIFICATIONS %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_DISABLE_SVC_NOTIFICATIONS:nagios_command};%{DATA:nagios_hostname};%
{GREEDYDATA:nagios_service} #Enable host & service notifications
NAGIOS_EC_LINE_ENABLE_HOST_SVC_NOTIFICATIONS %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_ENABLE_HOST_SVC_NOTIFICATIONS:nagios_command};%
{GREEDYDATA:nagios_hostname} NAGIOS_EC_LINE_ENABLE_HOST_NOTIFICATIONS %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_ENABLE_HOST_NOTIFICATIONS:nagios_command};%
{GREEDYDATA:nagios_hostname} NAGIOS_EC_LINE_ENABLE_SVC_NOTIFICATIONS %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_ENABLE_SVC_NOTIFICATIONS:nagios_command};%{DATA:nagios_hostname};%
{GREEDYDATA:nagios_service} #Schedule host & service downtime
NAGIOS_EC_LINE_SCHEDULE_HOST_DOWNTIME %
{NAGIOS_TYPE_EXTERNAL_COMMAND:nagios_type}: %
{NAGIOS_EC_SCHEDULE_HOST_DOWNTIME:nagios_command};%{DATA:nagios_hostname};%
{NUMBER:nagios_start_time};%{NUMBER:nagios_end_time};%{NUMBER:nagios_fixed};%
{NUMBER:nagios_trigger_id};%{NUMBER:nagios_duration};%{DATA:author};%

```

```
{DATA:comment} #End matching line NAGIOSLOGLINE %{NAGIOSTIME} (?:%
{NAGIOS_WARNING}|%{NAGIOS_CURRENT_SERVICE_STATE}|%{NAGIOS_CURRENT_HOST_STATE}|%
{NAGIOS_SERVICE_NOTIFICATION}|%{NAGIOS_HOST_NOTIFICATION}|%
{NAGIOS_SERVICE_ALERT}|%{NAGIOS_HOST_ALERT}|%{NAGIOS_SERVICE_FLAPPING_ALERT}|%
{NAGIOS_HOST_FLAPPING_ALERT}|%{NAGIOS_SERVICE_DOWNTIME_ALERT}|%
{NAGIOS_HOST_DOWNTIME_ALERT}|%{NAGIOS_PASSIVE_SERVICE_CHECK}|%
{NAGIOS_PASSIVE_HOST_CHECK}|%{NAGIOS_SERVICE_EVENT_HANDLER}|%
{NAGIOS_HOST_EVENT_HANDLER}|%{NAGIOS_TIMEPERIOD_TRANSITION}|%
{NAGIOS_EC_LINE_DISABLE_SVC_CHECK}|%{NAGIOS_EC_LINE_ENABLE_SVC_CHECK}|%
{NAGIOS_EC_LINE_DISABLE_HOST_CHECK}|%{NAGIOS_EC_LINE_ENABLE_HOST_CHECK}|%
{NAGIOS_EC_LINE_PROCESS_HOST_CHECK_RESULT}|%
{NAGIOS_EC_LINE_PROCESS_SERVICE_CHECK_RESULT}|%
{NAGIOS_EC_LINE_SCHEDULE_HOST_DOWNTIME}|%
{NAGIOS_EC_LINE_DISABLE_HOST_SVC_NOTIFICATIONS}|%
{NAGIOS_EC_LINE_ENABLE_HOST_SVC_NOTIFICATIONS}|%
{NAGIOS_EC_LINE_DISABLE_HOST_NOTIFICATIONS}|%
{NAGIOS_EC_LINE_ENABLE_HOST_NOTIFICATIONS}|%
{NAGIOS_EC_LINE_DISABLE_SVC_NOTIFICATIONS}|%
{NAGIOS_EC_LINE_ENABLE_SVC_NOTIFICATIONS}|%{GREEDYDATA:greedy_message})
```

Mongodb

```
MONGO_LOG %{SYSLOGTIMESTAMP:timestamp} \[%{WORD:component}\] %
{GREEDYDATA:message} MONGO_QUERY \{ (?<= { ).*(?= } nreturn:) \} MONGO_SLOWQUERY
%{WORD} %{MONGO_WORDDASH:database}.\%{MONGO_WORDDASH:collection} %{WORD}: %
{MONGO_QUERY:query} %{WORD}:%{NONNEGINT:nreturn} %{WORD}:%{NONNEGINT:ntoskip} %
{WORD}:%{NONNEGINT:nscanned}.*nreturned:%{NONNEGINT:nreturned}..+ (?<duration>[0-
9])+ms MONGO_WORDDASH \b[\w-]+\b MONGO3_SEVERITY \w MONGO3_COMPONENT %{WORD}|-
MONGO3_LOG %{TIMESTAMP_ISO8601:timestamp} %{MONGO3_SEVERITY:severity} %
{MONGO3_COMPONENT:component}%{SPACE}(?:\[%{DATA:context}\])? %
{GREEDYDATA:message}
```

Mcollective-patterns

```
# Remember, these can be multi-line events. MCOLLECTIVE ., \[%
{TIMESTAMP_ISO8601:timestamp} %#{POSINT:pid}\|%{SPACE}%{LOGLEVEL:event_level}
MCOLLECTIVEAUDIT %{TIMESTAMP_ISO8601:timestamp}:
```

Mcollective

```
MCOLLECTIVEAUDIT %{TIMESTAMP_ISO8601:timestamp}:
```

Lucidworks-enterprise

```
# Parses the Lucidworks Enterprise logs. Requires grok-patterns OLDLUCIDFIND %
{IPORHOST:clientip}(?:%{SPACE})(?:%{USER:ident}|-)(?:%{SPACE})(?:%{USER:auth}|-)
(?:%{SPACE})\[!%{HTTPDATE:timestamp}\](?:%{SPACE})"%{WORD:verb}(?:%{SPACE})%
{DATA:rawrequest}(?: HTTP/%{NUMBER:httpversion})"(?:%{SPACE})%{NUMBER:response}
(?:%{SPACE})(?:%{NUMBER:bytes}|-)(?:%{SPACE})(?:%{NUMBER:timing}|-)
```

Linux-syslog

```
SYSLOG5424PRINTASCII [!~]+ SYSLOGBASE2 (?:%{SYSLOGTIMESTAMP:timestamp})|%
{TIMESTAMP_ISO8601:timestamp8601}) (?:%{SYSLOGFACILITY})?%
{SYSLOGHOST:logsource}+(?: %{{SYSLOGPROG}:|) SYSLOGPAMSESSION %{SYSLOGBASE} (?:=%
{GREEDYDATA:message})%{WORD:pam_module}\(%{DATA:pam_caller}\): session %
{WORD:pam_session_state} for user %{USERNAME:username}(?: by %
{GREEDYDATA:pam_by})? CRON_ACTION [A-Z ]+ CRONLOG %{SYSLOGBASE} \(%{USER:user}\)
%{CRON_ACTION:action} \(%{DATA:message}\) SYSLOGLINE %{SYSLOGBASE2} %
{GREEDYDATA:message} # IETF 5424 syslog(8) format (see http://www.rfc-
editor.org/info/rfc5424) SYSLOG5424PRI <%{NONNEGINT:syslog5424_pri}> SYSLOG5424SD
\[!%{DATA}\]+ SYSLOG5424BASE %{SYSLOG5424PRI}%{NONNEGINT:syslog5424_ver} +(?:%
{TIMESTAMP_ISO8601:syslog5424_ts}|-) +(?:%{HOSTNAME:syslog5424_host}|-) +(-|%
{SYSLOG5424PRINTASCII:syslog5424_app}) +(-|%
{SYSLOG5424PRINTASCII:syslog5424_proc}) +(-|%
{SYSLOG5424PRINTASCII:syslog5424_msgid}) +(?:%{SYSLOG5424SD:syslog5424_sd}|-) )
SYSLOG5424LINE %{SYSLOG5424BASE} +%{GREEDYDATA:syslog5424_msg}
```

Junos

```
# JUNOS 11.4 RT_FLOW patterns RT_FLOW_EVENT
(RT_FLOW_SESSION_CREATE|RT_FLOW_SESSION_CLOSE|RT_FLOW_SESSION_DENY) RT_FLOW1 %
{RT_FLOW_EVENT:event}: %{GREEDYDATA:close-reason}: %{IP:src-ip}/%{INT:src-port}-
>%{IP:dst-ip}/%{INT:dst-port} %{DATA:service} %{IP:nat-src-ip}/%{INT:nat-src-
port}->%{IP:nat-dst-ip}/%{INT:nat-dst-port} %{DATA:src-nat-rule-name} %{DATA:dst-
nat-rule-name} %{INT:protocol-id} %{DATA:policy-name} %{DATA:from-zone} %
{DATA:to-zone} %{INT:session-id} \d+\(%{DATA:sent}\) \d+\(%{DATA:received}\) %
{INT:elapsed-time} .* RT_FLOW2 %{RT_FLOW_EVENT:event}: session created %{IP:src-
ip}/%{INT:src-port}->%{IP:dst-ip}/%{INT:dst-port} %{DATA:service} %{IP:nat-src-
ip}/%{INT:nat-src-port}->%{IP:nat-dst-ip}/%{INT:nat-dst-port} %{DATA:src-nat-
rule-name} %{DATA:dst-nat-rule-name} %{INT:protocol-id} %{DATA:policy-name} %
{DATA:from-zone} %{DATA:to-zone} %{INT:session-id} .* RT_FLOW3 %
{RT_FLOW_EVENT:event}: session denied %{IP:src-ip}/%{INT:src-port}->%{IP:dst-
ip}/%{INT:dst-port} %{DATA:service} %{INT:protocol-id}\(\d\) %{DATA:policy-name}
%{DATA:from-zone} %{DATA:to-zone} .*
```

```
JAVACLASS (?:[a-zA-Z$_][a-zA-Z$_0-9]*\.)*[a-zA-Z$_][a-zA-Z$_0-9]* #Space is an
allowed character to match special cases like 'Native Method' or 'Unknown Source'
JAVAFILE (?:[A-Za-z0-9_.-]+) #Allow special <init>, <clinit> methods JAVAMETHOD
(?:(<(?<cl)?init>)|[a-zA-Z$_][a-zA-Z$_0-9]*) #Line number is optional in special
cases 'Native method' or 'Unknown source' JAVASTACKTRACEPART %{SPACE}at %
{JAVACLASS:class}\.%{JAVAMETHOD:method}\(%{JAVAFILE:file}(?:%{NUMBER:line})?\) #
Java Logs JAVATHREAD (?:[A-Z]{2}-Processor[\\d]+) JAVACLASS (?:[a-zA-Z0-9-]+\.)+
[A-Za-z0-9$]+ JAVAFILE (?:[A-Za-z0-9_.-]+) JAVASTACKTRACEPART at %
{JAVACLASS:class}\.%{WORD:method}\(%{JAVAFILE:file}:%{NUMBER:line}\)
JAVALOGMESSAGE (.*) # MMM dd, yyyy HH:mm:ss eg: Jan 9, 2014 7:13:13 AM
CATALINA_DATESTAMP %{MONTH} %{MONTHDAY}, 20%{YEAR} %{HOUR}:?%{MINUTE}(?:?%
{SECOND}) (?:AM|PM) # yyyy-MM-dd HH:mm:ss,SSS ZZZ eg: 2014-01-09 17:32:25,527
-0800 TOMCAT_DATESTAMP 20%{YEAR}-%{MONTHNUM}-%{MONTHDAY} %{HOUR}:?%{MINUTE}(?:?%
{SECOND}) %{ISO8601_TIMEZONE} CATALINALOG %{CATALINA_DATESTAMP:timestamp} %
{JAVACLASS:class} %{JAVALOGMESSAGE:logmessage} # 2014-01-09 20:03:28,269 -0800 |
ERROR | com.example.service.ExampleService - something completely unexpected
happened... TOMCATLOG %{TOMCAT_DATESTAMP:timestamp} \ | %{LOGLEVEL:level} \ | %
{JAVACLASS:class} - %{JAVALOGMESSAGE:logmessage}
```

```
# These patterns were tested w/ haproxy 1.4.15 ## Documentation of the haproxy log formats can be found at the following links: ##  
http://code.google.com/p/haproxy-docs/wiki/HTTPLogFormat ##  
http://code.google.com/p/haproxy-docs/wiki/TCPLogFormat  
HAPROXYTIME {?!<[0-9]}%  
{HOUR:haproxy_hour}:{MINUTE:haproxy_minute}({?:%{SECOND:haproxy_second}}){?!<[0-9]} HAPROXYDATE %{MONTHDAY:haproxy_mday}/{MONTH:haproxy_month}/%  
{YEAR:haproxy_year}:{HAPROXYTIME:haproxy_time}.%{INT:haproxy_milliseconds} #  
Override these default patterns to parse out what is captured in your haproxy.cfg  
HAPROXYCAPTUREDREQUESTHEADERS %{DATA:captured_request_headers}  
HAPROXYCAPTUREDRESPONSEHEADERS %{DATA:captured_response_headers} # Example: #  
These haproxy config lines will add data to the logs that are captured # by the  
patterns below. Place them in your custom patterns directory to # override the  
defaults. # # capture request header Host len 40 # capture request header X-  
Forwarded-For len 50 # capture request header Accept-Language len 50 # capture  
request header referrer len 200 # capture request header User-Agent len 200 # #  
capture response header Content-Type len 30 # capture response header Content-  
Encoding len 10 # capture response header Cache-Control len 200 # capture  
response header Last-Modified len 200 # # HAPROXYCAPTUREDREQUESTHEADERS %  
{DATA:request_header_host}\|{%{DATA:request_header_x_forwarded_for}\|}%  
{DATA:request_header_accept_language}\|{%{DATA:request_header_referrer}\|}%  
{DATA:request_header_user_agent} # HAPROXYCAPTUREDRESPONSEHEADERS %  
{DATA:response_header_content_type}\|{%{DATA:response_header_content_encoding}\|}%  
{DATA:response_header_cache_control}\|{%{DATA:response_header_last_modified} #  
parse a haproxy 'httplog' line HAPROXYHTTPBASE {%IP:client_ip}:{INT:client_port}  
\|{%HAPROXYDATE:accept_date}\| }{%NOTSPACE:frontend_name} %  
{NOTSPACE:backend_name}/{%NOTSPACE:server_name} {%INT:time_queue}/%  
{INT:time_queue}/%{INT:time_backend_connect}/{%INT:time_backend_response}/%  
{NOTSPACE:time_duration} {%INT:http_status_code} {%NOTSPACE:bytes_read} %  
{DATA:captured_request_cookie} {%DATA:captured_response_cookie} %  
{NOTSPACE:termination_state} {%INT:actconn}/{%INT:feconn}/{%INT:beconn}/{%  
{INT:srvconn}/{%NOTSPACE:retries} {%INT:srv_queue}/{%INT:backend_queue} (\|%  
{HAPROXYCAPTUREDREQUESTHEADERS}\})?(\ )?(\|{%HAPROXYCAPTUREDRESPONSEHEADERS}\})?(  
)"(<BADREQ>|(%{WORD:http_verb} (%{URIPROTO:http_proto}://)?(?:%{USER:http_user}  
(?:\^[^@]*)?)?(?:%{URHOST:http_host})?(?:%{URIPATHPARAM:http_request})?( HTTP/%  
{NUMBER:http_version}))?" HAPROXYHTTP (?:%{SYSLOGTIMESTAMP:syslog_timestamp}|%  
{TIMESTAMP_IS08601:timestamp8601}) {%IPORHOST:syslog_server} {%SYSLOGPROG}: %  
{HAPROXYHTTPBASE} # parse a haproxy 'tcplog' line HAPROXYTCP (?:%  
{SYSLOGTIMESTAMP:syslog_timestamp}|%{TIMESTAMP_IS08601:timestamp8601}) %  
{IPORHOST:syslog_server} {%SYSLOGPROG}: {%IP:client_ip}:{INT:client_port} \|%  
{HAPROXYDATE:accept_date}\| }{%NOTSPACE:frontend_name} {%NOTSPACE:backend_name}/%  
{NOTSPACE:server_name} {%INT:time_queue}/{%INT:time_backend_connect}/%  
{NOTSPACE:time_duration} {%NOTSPACE:bytes_read} {%NOTSPACE:termination_state} %  
{INT:actconn}/{%INT:feconn}/{%INT:beconn}/{%INT:srvconn}/{%NOTSPACE:retries} %  
{INT:srv_queue}/{%INT:backend_queue}
```

```

USERNAME [a-zA-Z0-9._-]+ USER %{USERNAME} EMAILLOCALPART [a-zA-Z][a-zA-Z0-9_+~
=:]+ EMAILADDRESS %{EMAILLOCALPART}@%{HOSTNAME} HTTPDUSER %{EMAILADDRESS}|%{USER}
INT (?:[+-]?(?:[0-9]+)) BASE10NUM (?<![0-9.-])(?>[+-]?(?:[0-9]+(?:\.[0-
9]+)?|(?:\.[0-9]+))) NUMBER (?:%{BASE10NUM}) BASE16NUM (?<![0-9A-Fa-f])(?:[+-]?
(?:0x)?(?:[0-9A-Fa-f]+)) BASE16FLOAT \b(?<![0-9A-Fa-f.])(?:[+-]?(?:0x)?(?:[0-
9A-Fa-f]+(?:\.[0-9A-Fa-f]*)?)|(?:\.[0-9A-Fa-f]+)))\b POSINT \b(?:[1-9][0-9]*)\b
NONNEGINT \b(?:[0-9]+)\b WORD \b\w+\b NOTSPACE \S+ SPACE \s* DATA .*? GREEDYDATA
.* QUOTEDSTRING (?>(?!\\)(?>"(?>\\.|^[^"]+)"+"|'(?>\\.|^[^']+)'+'|'(|(?
>`(?>\\.|^[^`]+)`|`))) UUID [A-Fa-f0-9]{8}-(?:[A-Fa-f0-9]{4}-){3}[A-Fa-f0-9]
{12} # URN, allowing use of RFC 2141 section 2.3 reserved characters URN urn:[0-
9A-Za-z][0-9A-Za-z-]{0,31}:(?:%[0-9a-fA-F]{2}|[0-9A-Za-z()+,.;=@;$_!*'/?#-])+

```



```
{WORD:module}:%{LOGLEVEL:loglevel}\\ \[pid %{POSINT:pid}:tid %{NUMBER:tid}]( \(%
{POSINT:proxy_errorcode}\\)%{DATA:proxy_errormessage}:)?( \[client %
{IPORHOST:client}:%{POSINT:clientport}\\})? %{DATA:errorcode}: %
{GREEDYDATA:message} HTTPD_ERRORLOG %{HTTPD20_ERRORLOG}|%{HTTPD24_ERRORLOG} # Log
Levels LOGLEVEL ([Aa]lert|ALERT|[Tt]race|TRACE|[Dd]ebug|DEBUG|[Nn]otice|NOTICE|
[Ii]nfo|INFO|[Ww]arn?(?:ing)?|WARN?(?:ING)?|[Ee]rr?(?:or)?|ERR?(?:OR)?|[Cc]rit?
(?:ical)?|CRIT?(?:ICAL)?|[Ff]atal|FATAL|[Ss]ever|Severe|EMERG(?:ENCY)?|
[Ee]merg(?:ency)?)
```



```
# NetScreen firewall logs NETSCREENSESSIONLOG %{SYSLOGTIMESTAMP:date} %
{IPORHOST:device} %{IPORHOST}: NetScreen device_id=%{WORD:device_id}%{DATA}:
start_time=%{QUOTEDSTRING:start_time} duration=%{INT:duration} policy_id=%
{INT:policy_id} service=%{DATA:service} proto=%{INT:proto} src zone=%
{WORD:src_zone} dst zone=%{WORD:dst_zone} action=%{WORD:action} sent=%{INT:sent}
rcvd=%{INT:rcvd} src=%{IPORHOST:src_ip} dst=%{IPORHOST:dst_ip} src_port=%
{INT:src_port} dst_port=%{INT:dst_port} src-xlated ip=%{IPORHOST:src_xlated_ip}
port=%{INT:src_xlated_port} dst-xlated ip=%{IPORHOST:dst_xlated_ip} port=%
{INT:dst_xlated_port} session_id=%{INT:session_id} reason=%{GREEDYDATA:reason}
#== Cisco ASA == CISCO_TAGGED_SYSLOG ^<{%POSINT:syslog_pri}>%
{CISCOTIMESTAMP:timestamp}(%{SYSLOGHOST:sysloghost})? ?: %%(CISCOTAG:ciscotag):
CISCOTIMESTAMP %{MONTH} +%{MONTHDAY}(?: %{YEAR})? %{TIME} CISCOTAG [A-Z0-9]+-%
{INT}-(?:[A-Z0-9_]+) # Common Particles CISCO_ACTION
Built|Teardown|Deny|Denied|denied|requested|permitted|denied by
ACL|discarded|est-allowed|Dropping|created|deleted CISCO_REASON Duplicate TCP
SYN|Failed to locate egress interface|Invalid transport field|No matching
connection|DNS Response|DNS Query|(?:%{WORD}\s)* CISCO_DIRECTION
Inbound|inbound|Outbound|outbound CISCO_INTERVAL first hit|{%INT}-second interval
CISCO_XLATE_TYPE static|dynamic # ASA-1-104001 CISCOFW104001 \
(?:Primary|Secondary)\) Switching to ACTIVE - %{GREEDYDATA:switch_reason} # ASA-
1-104002 CISCOFW104002 \((?:Primary|Secondary)\) Switching to STANDBY - %
{GREEDYDATA:switch_reason} # ASA-1-104003 CISCOFW104003 \((?:Primary|Secondary)\)
Switching to FAILED\. # ASA-1-104004 CISCOFW104004 \((?:Primary|Secondary)\)
Switching to OK\. # ASA-1-105003 CISCOFW105003 \((?:Primary|Secondary)\)
Monitoring on [Ii]nterface %{GREEDYDATA:interface_name} waiting # ASA-1-105004
CISCOFW105004 \((?:Primary|Secondary)\) Monitoring on [Ii]nterface %
{GREEDYDATA:interface_name} normal # ASA-1-105005 CISCOFW105005 \
(?:Primary|Secondary)\) Lost Failover communications with mate on [Ii]nterface %
{GREEDYDATA:interface_name} # ASA-1-105008 CISCOFW105008 \
(?:Primary|Secondary)\) Testing [Ii]nterface %{GREEDYDATA:interface_name} # ASA-
1-105009 CISCOFW105009 \((?:Primary|Secondary)\) Testing on [Ii]nterface %
{GREEDYDATA:interface_name} (?:Passed|Failed) # ASA-2-106001 CISCOFW106001 %
{CISCO_DIRECTION:direction} %{WORD:protocol} connection %{CISCO_ACTION:action}
from %{IP:src_ip}/%{INT:src_port} to %{IP:dst_ip}/%{INT:dst_port} flags %
{GREEDYDATA:tcp_flags} on interface %{GREEDYDATA:interface} # ASA-2-106006, ASA-
2-106007, ASA-2-106010 CISCOFW106006 106007 106010 %{CISCO_ACTION:action} %
{CISCO_DIRECTION:direction} %{WORD:protocol} (?:from|src) %{IP:src_ip}/%
{INT:src_port}(\(%{DATA:src_fwuser}\))?(?:to|dst) %{IP:dst_ip}/%{INT:dst_port}(\
(%{DATA:dst_fwuser}\))?(?:on interface %{DATA:interface})|due to %
{CISCO_REASON:reason}) # ASA-3-106014 CISCOFW106014 %{CISCO_ACTION:action} %
{CISCO_DIRECTION:direction} %{WORD:protocol} src %{DATA:src_interface}:%
{IP:src_ip}(\(%{DATA:src_fwuser}\))? dst %{DATA:dst_interface}:%{IP:dst_ip}(\(%
{DATA:dst_fwuser}\))? \((type %{INT:icmp_type}, code %{INT:icmp_code}\) # ASA-6-
106015 CISCOFW106015 %{CISCO_ACTION:action} %{WORD:protocol} \(%
{DATA:policy_id}\) from %{IP:src_ip}/%{INT:src_port} to %{IP:dst_ip}/%
{INT:dst_port} flags %{DATA:tcp_flags} on interface %{GREEDYDATA:interface} #
ASA-1-106021 CISCOFW106021 %{CISCO_ACTION:action} %{WORD:protocol} reverse path
check from %{IP:src_ip} to %{IP:dst_ip} on interface %{GREEDYDATA:interface} #
ASA-4-106023 CISCOFW106023 %{CISCO_ACTION:action}( protocol)? %{WORD:protocol}
src %{DATA:src_interface}:%{DATA:src_ip}(/%{INT:src_port})?(\(%
{DATA:src_fwuser}\))? dst %{DATA:dst_interface}:%{DATA:dst_ip}(/%{INT:dst_port})?
(\(%{DATA:dst_fwuser}\))?( \((type %{INT:icmp_type}, code %{INT:icmp_code}\))? by
access-group "%{DATA:policy_id}"? \[%{DATA:hashcode1}, %{DATA:hashcode2}\] #
ASA-4-106100, ASA-4-106102, ASA-4-106103 CISCOFW106100_2_3 access-list %
{NOTSPACE:policy_id} %{CISCO_ACTION:action} %{WORD:protocol} for user %
{DATA:src_fwuser}' %{DATA:src_interface}/%{IP:src_ip}(\(%{INT:src_port}\) -> %
{DATA:dst_interface}/%{IP:dst_ip}(\(%{INT:dst_port}\) hit-cnt %{INT:hit_count} %
{CISCO_INTERVAL:interval} \[%{DATA:hashcode1}, %{DATA:hashcode2}\] # ASA-5-106100
CISCOFW106100 access-list %{NOTSPACE:policy_id} %{CISCO_ACTION:action} %
{WORD:protocol} %{DATA:src_interface}/%{IP:src_ip}(\(%{INT:src_port}\) \(%
{DATA:src_fwuser}\))? -> %{DATA:dst_interface}/%{IP:dst_ip}(\(%{INT:dst_port}\) \
(%{DATA:src_fwuser}\))? hit-cnt %{INT:hit_count} %{CISCO_INTERVAL:interval} \[%
```

```

{DATA:hashcode1}, %{DATA:hashcode2}\] # ASA-6-110002 CISCOFW110002 %
{CISCO_REASON:reason} for %{WORD:protocol} from %{DATA:src_interface}:%
{IP:src_ip}/{INT:src_port} to %{IP:dst_ip}/{INT:dst_port} # ASA-6-302010
CISCOFW302010 %{INT:connection_count} in use, %{INT:connection_count_max} most
used # ASA-6-302013, ASA-6-302014, ASA-6-302015, ASA-6-302016
CISCOFW302013_302014_302015_302016 %{CISCO_ACTION:action}(:? %
{CISCO_DIRECTION:direction})? %{WORD:protocol} connection %{INT:connection_id}
for %{DATA:src_interface}:%{IP:src_ip}/{INT:src_port}(\(%{IP:src_mapped_ip}/%
{INT:src_mapped_port}\))?( \(%{DATA:src_fwuser}\))? to %{DATA:dst_interface}:%
{IP:dst_ip}/{INT:dst_port}(\(%{IP:dst_mapped_ip}/%{INT:dst_mapped_port}\))?( \(%
{DATA:dst_fwuser}\))?( duration %{TIME:duration} bytes %{INT:bytes})?(?: %
{CISCO_REASON:reason})?( \(%{DATA:user}\))? # ASA-6-302020, ASA-6-302021
CISCOFW302020_302021 %{CISCO_ACTION:action}(:? %{CISCO_DIRECTION:direction})? %
{WORD:protocol} connection for faddr %{IP:dst_ip}/{INT:icmp_seq_num}(:? \(%
{DATA:fwuser}\))? gaddr %{IP:src_xlated_ip}/{INT:icmp_code_xlated} laddr %
{IP:src_ip}/{INT:icmp_code}(\(%{DATA:user}\))? # ASA-6-305011 CISCOFW305011 %
{CISCO_ACTION:action} %{CISCO_XLATE_TYPE:xlate_type} %{WORD:protocol} translation
from %{DATA:src_interface}:%{IP:src_ip}(/%{INT:src_port})?( \(%
{DATA:src_fwuser}\))? to %{DATA:src_xlated_interface}:%{IP:src_xlated_ip}/%
{DATA:src_xlated_port} # ASA-3-313001, ASA-3-313004, ASA-3-313008
CISCOFW313001_313004_313008 %{CISCO_ACTION:action} %{WORD:protocol} type=%
{INT:icmp_type}, code=%{INT:icmp_code} from %{IP:src_ip} on interface %
{DATA:interface}( to %{IP:dst_ip})? # ASA-4-313005 CISCOFW313005 %
{CISCO_REASON:reason} for %{WORD:protocol} error message: %{WORD:error_protocol}
src %{DATA:err_src_interface}:%{IP:err_src_ip}(\(%{DATA:err_src_fwuser}\))? dst %
{DATA:err_dst_interface}:%{IP:err_dst_ip}(\(%{DATA:err_dst_fwuser}\))? \((type %
{INT:err_icmp_type}, code %{INT:err_icmp_code}\) on %{DATA:interface} interface\.
Original IP payload: %{WORD:protocol} src %{IP:orig_src_ip}/{INT:orig_src_port}
(\(%{DATA:orig_src_fwuser}\))? dst %{IP:orig_dst_ip}/{INT:orig_dst_port}(\(%
{DATA:orig_dst_fwuser}\))? # ASA-5-321001 CISCOFW321001 Resource '%
{WORD:resource_name}' limit of %{POSINT:resource_limit} reached for system # ASA-
4-402117 CISCOFW402117 %{WORD:protocol}: Received a non-IPSec packet \((protocol=
%{WORD:orig_protocol}\) from %{IP:src_ip} to %{IP:dst_ip} # ASA-4-402119
CISCOFW402119 %{WORD:protocol}: Received an %{WORD:orig_protocol} packet \((SPI= %
{DATA:spi}, sequence number= %{DATA:seq_num}\) from %{IP:src_ip} \((user= %
{DATA:user}\) to %{IP:dst_ip} that failed anti-replay checking # ASA-4-419001
CISCOFW419001 %{CISCO_ACTION:action} %{WORD:protocol} packet from %
{DATA:src_interface}:%{IP:src_ip}/{INT:src_port} to %{DATA:dst_interface}:%
{IP:dst_ip}/{INT:dst_port}, reason: %{GREEDYDATA:reason} # ASA-4-419002
CISCOFW419002 %{CISCO_REASON:reason} from %{DATA:src_interface}:%{IP:src_ip}/%
{INT:src_port} to %{DATA:dst_interface}:%{IP:dst_ip}/{INT:dst_port} with
different initial sequence number # ASA-4-500004 CISCOFW500004 %
{CISCO_REASON:reason} for protocol=%{WORD:protocol}, from %{IP:src_ip}/%
{INT:src_port} to %{IP:dst_ip}/{INT:dst_port} # ASA-6-602303, ASA-6-602304
CISCOFW602303_602304 %{WORD:protocol}: An %{CISCO_DIRECTION:direction} %
{GREEDYDATA:tunnel_type} SA \((SPI= %{DATA:spi}\) between %{IP:src_ip} and %
{IP:dst_ip} \((user= %{DATA:user}\) has been %{CISCO_ACTION:action} # ASA-7-
710001, ASA-7-710002, ASA-7-710003, ASA-7-710005, ASA-7-710006
CISCOFW710001_710002_710003_710005_710006 %{WORD:protocol} (:?request|access) %
{CISCO_ACTION:action} from %{IP:src_ip}/{INT:src_port} to %
{DATA:dst_interface}:%{IP:dst_ip}/{INT:dst_port} # ASA-6-713172 CISCOFW713172
Group = %{GREEDYDATA:group}, IP = %{IP:src_ip}, Automatic NAT Detection
Status:\s+Remote end\s*%{DATA:is_remote_natted}\s*behind a NAT
device\s+This\s+end\s*%{DATA:is_local_natted}\s*behind a NAT device # ASA-4-
733100 CISCOFW733100 \[\s*%{DATA:drop_type}\s*\] drop %{DATA:drop_rate_id}
exceeded. Current burst rate is %{INT:drop_rate_current_burst} per second, max
configured rate is %{INT:drop_rate_max_burst}; Current average rate is %
{INT:drop_rate_current_avg} per second, max configured rate is %
{INT:drop_rate_max_avg}; Cumulative total count is %{INT:drop_total_count} #==
End Cisco ASA == # Shorewall firewall logs SHOREWALL (%
{SYSLOGTIMESTAMP:timestamp}) (%{WORD:nf_host}) kernel:.*Shorewall:(%
{WORD:nf_action1})?:(%{WORD:nf_action2})?.*IN=(%{USERNAME:nf_in_interface})?.*
(OUT= *MAC=(%{COMMONMAC:nf_dst_mac}):(%{COMMONMAC:nf_src_mac})?)|OUT=%
{USERNAME:nf_out_interface}).*SRC=(%{IPV4:nf_src_ip}).*DST=(%
{IPV4:nf_dst_ip}).*LEN=(%{WORD:nf_len}).*TOS=(%{WORD:nf_tos}).*PREC=(%
{WORD:nf_prec}).*TTL=(%{INT:nf_ttl}).*ID=(%{INT:nf_id}).*PROTO=(%

```

```
{WORD:nf_protocol}).?*SPT=(%{INT:nf_src_port}?.*DPT=%{INT:nf_dst_port}?.*) #==  
End Shorewall
```

Exim

```
EXIM_MSGID [0-9A-Za-z]{6}-[0-9A-Za-z]{6}-[0-9A-Za-z]{2} EXIM_FLAGS (<|[-=>*>|  
[*]{2}|==) EXIM_DATE %{YEAR:exim_year}-%{MONTHNUM:exim_month}-%  
{MONTHDAY:exim_day} %{TIME:exim_time} EXIM_PID \[%{POSINT}\] EXIM_QT ((\d+y)?  
(\d+w)?(\d+d)?(\d+h)?(\d+m)?(\d+s)?) EXIM_EXCLUDE_TERMS (Message is frozen|  
(Start|End) queue run| Warning: | retry time not reached | no (IP address|host  
name) found for (IP address|host) | unexpected disconnection while reading SMTP  
command | no immediate delivery: |another process is handling this message)  
EXIM_REMOTE_HOST (H=(%{NOTSPACE:remote_hostname} )?(\(%  
{NOTSPACE:remote_helname}\) )? \[%{IP:remote_host}\]) EXIM_INTERFACE (I=\[%  
{IP:exim_interface}\](:%{NUMBER:exim_interface_port})) EXIM_PROTOCOL (P=%  
{NOTSPACE:protocol}) EXIM_MSG_SIZE (S=%{NUMBER:exim_msg_size}) EXIM_HEADER_ID  
(id=%{NOTSPACE:exim_header_id}) EXIM_SUBJECT (T=%{QS:exim_subject})
```

Bro

```
# https://www.bro.org/sphinx/script-reference/log-files.html # http.log BRO_HTTP  
%{NUMBER:ts}\t%{NOTSPACE:uid}\t%{IP:orig_h}\t%{INT:orig_p}\t%{IP:resp_h}\t%  
{INT:resp_p}\t%{INT:trans_depth}\t%{GREEDYDATA:method}\t%{GREEDYDATA:domain}\t%  
{GREEDYDATA:uri}\t%{GREEDYDATA:referrer}\t%{GREEDYDATA:user_agent}\t%  
{NUMBER:request_body_len}\t%{NUMBER:response_body_len}\t%  
{GREEDYDATA:status_code}\t%{GREEDYDATA:status_msg}\t%{GREEDYDATA:info_code}\t%  
{GREEDYDATA:info_msg}\t%{GREEDYDATA:filename}\t%{GREEDYDATA:bro_tags}\t%  
{GREEDYDATA:username}\t%{GREEDYDATA:password}\t%{GREEDYDATA:proxied}\t%  
{GREEDYDATA:orig_fuids}\t%{GREEDYDATA:orig_mime_types}\t%  
{GREEDYDATA:resp_fuids}\t%{GREEDYDATA:resp_mime_types} # dns.log BRO_DNS %  
{NUMBER:ts}\t%{NOTSPACE:uid}\t%{IP:orig_h}\t%{INT:orig_p}\t%{IP:resp_h}\t%  
{INT:resp_p}\t%{WORD:proto}\t%{INT:trans_id}\t%{GREEDYDATA:query}\t%  
{GREEDYDATA:qclass}\t%{GREEDYDATA:qclass_name}\t%{GREEDYDATA:qtype}\t%  
{GREEDYDATA:qtype_name}\t%{GREEDYDATA:rcode}\t%{GREEDYDATA:rcode_name}\t%  
{GREEDYDATA:AA}\t%{GREEDYDATA:TC}\t%{GREEDYDATA:RD}\t%{GREEDYDATA:RA}\t%  
{GREEDYDATA:Z}\t%{GREEDYDATA:answers}\t%{GREEDYDATA:TTLs}\t%{GREEDYDATA:rejected}  
# conn.log BRO_CONN %{NUMBER:ts}\t%{NOTSPACE:uid}\t%{IP:orig_h}\t%{INT:orig_p}\t%  
{IP:resp_h}\t%{INT:resp_p}\t%{WORD:proto}\t%{GREEDYDATA:service}\t%  
{NUMBER:duration}\t%{NUMBER:orig_bytes}\t%{NUMBER:resp_bytes}\t%  
{GREEDYDATA:conn_state}\t%{GREEDYDATA:local_orig}\t%{GREEDYDATA:missed_bytes}\t%  
{GREEDYDATA:history}\t%{GREEDYDATA:orig_pkts}\t%{GREEDYDATA:orig_ip_bytes}\t%  
{GREEDYDATA:resp_pkts}\t%{GREEDYDATA:resp_ip_bytes}\t%{GREEDYDATA:tunnel_parents}  
# files.log BRO_FILES %{NUMBER:ts}\t%{NOTSPACE:fuid}\t%{IP:tx_hosts}\t%  
{IP:rx_hosts}\t%{NOTSPACE:conn_uids}\t%{GREEDYDATA:source}\t%  
{GREEDYDATA:depth}\t%{GREEDYDATA:analyzers}\t%{GREEDYDATA:mime_type}\t%  
{GREEDYDATA:filename}\t%{GREEDYDATA:duration}\t%{GREEDYDATA:local_orig}\t%  
{GREEDYDATA:is_orig}\t%{GREEDYDATA:seen_bytes}\t%{GREEDYDATA:total_bytes}\t%  
{GREEDYDATA:missing_bytes}\t%{GREEDYDATA:overflow_bytes}\t%  
{GREEDYDATA:timedout}\t%{GREEDYDATA:parent_fuid}\t%{GREEDYDATA:md5}\t%  
{GREEDYDATA:sha1}\t%{GREEDYDATA:sha256}\t%{GREEDYDATA:extracted}
```

```

BACULA_TIMESTAMP %{MONTHDAY}-%{MONTH} %{HOUR}:%{MINUTE} BACULA_HOST [a-zA-Z0-9-]+
BACULA_VOLUME %{USER} BACULA_DEVICE %{USER} BACULA_DEVICEPATH %{UNIPATH}
BACULA_CAPACITY %{INT}{1,3}(,%{INT}{3})* BACULA_VERSION %{USER} BACULA_JOB %
{USER} BACULA_LOG_MAX_CAPACITY User defined maximum volume capacity %
{BACULA_CAPACITY} exceeded on device \"%{BACULA_DEVICE:device}\" \(%
{BACULA_DEVICEPATH}\) BACULA_LOG_END_VOLUME End of medium on Volume \"%
{BACULA_VOLUME:volume}\" Bytes=%{BACULA_CAPACITY} Blocks=%{BACULA_CAPACITY} at %
{MONTHDAY}-%{MONTH}-%{YEAR} %{HOUR}:%{MINUTE}. BACULA_LOG_NEW_VOLUME Created new
Volume \"%{BACULA_VOLUME:volume}\" in catalog. BACULA_LOG_NEW_LABEL Labeled new
Volume \"%{BACULA_VOLUME:volume}\" on device \"%{BACULA_DEVICE:device}\" \(%
{BACULA_DEVICEPATH}\). BACULA_LOG_WROTE_LABEL Wrote label to prelabeled Volume
\"%{BACULA_VOLUME:volume}\" on device \"%{BACULA_DEVICE:device}\" \(%
{BACULA_DEVICEPATH}\) BACULA_LOG_NEW_MOUNT New volume \"%{BACULA_VOLUME:volume}\"
mounted on device \"%{BACULA_DEVICE:device}\" \(%{BACULA_DEVICEPATH}\) at %
{MONTHDAY}-%{MONTH}-%{YEAR} %{HOUR}:%{MINUTE}. BACULA_LOG_NOOPEN \s+Cannot open %
{DATA}: ERR=%{GREEDYDATA:berror} BACULA_LOG_NOOPENDIR \s+Could not open directory
%{DATA}: ERR=%{GREEDYDATA:berror} BACULA_LOG_NOSTAT \s+Could not stat %{DATA}:
ERR=%{GREEDYDATA:berror} BACULA_LOG_NOJOBS There are no more Jobs associated with
Volume \"%{BACULA_VOLUME:volume}\". Marking it purged.
BACULA_LOG_ALL_RECORDS_PRUNED All records pruned from Volume \"%
{BACULA_VOLUME:volume}\"; marking it \"Purged\" BACULA_LOG_BEGIN_PRUNE_JOBS Begin
pruning Jobs older than %{INT} month %{INT} days . BACULA_LOG_BEGIN_PRUNE_FILES
Begin pruning Files. BACULA_LOG_PRUNED_JOBS Pruned %{INT} Jobs* for client %
{BACULA_HOST:client} from catalog. BACULA_LOG_PRUNED_FILES Pruned Files from %
{INT} Jobs* for client %{BACULA_HOST:client} from catalog. BACULA_LOG_ENDPRUNE
End auto prune. BACULA_LOG_STARTJOB Start Backup JobId %{INT}, Job=%
{BACULA_JOB:job} BACULA_LOG_STARTRESTORE Start Restore Job %{BACULA_JOB:job}
BACULA_LOG_USEDEVICE Using Device \"%{BACULA_DEVICE:device}\" BACULA_LOG_DIFF_FS
\s+%{UNIPATH} is a different filesystem. Will not descend from %{UNIPATH} into
it. BACULA_LOG_JOBEND Job write elapsed time = %{DATA:elapsed}, Transfer rate = %
{NUMBER} (K|M|G)? Bytes/second BACULA_LOG_NOPRUNE_JOBS No Jobs found to prune.
BACULA_LOG_NOPRUNE_FILES No Files found to prune. BACULA_LOG_VOLUME_PREVWRITTEN
Volume \"%{BACULA_VOLUME:volume}\" previously written, moving to end of data.
BACULA_LOG_READYAPPEND Ready to append to end of Volume \"%
{BACULA_VOLUME:volume}\" size=%{INT} BACULA_LOG_CANCELLING Cancelling duplicate
JobId=%{INT}. BACULA_LOG_MARKCANCEL JobId %{INT}, Job %{BACULA_JOB:job} marked to
be canceled. BACULA_LOG_CLIENT_RBJ shell command: run ClientRunBeforeJob \"%
{GREEDYDATA:runjob}\" BACULA_LOG_VSS (Generate )?VSS (Writer)?
BACULA_LOG_MAXSTART Fatal error: Job canceled because max start delay time
exceeded. BACULA_LOG_DUPLICATE Fatal error: JobId %{INT:duplicate} already
running. Duplicate job not allowed. BACULA_LOG_NOJOBSTAT Fatal error: No Job
status returned from FD. BACULA_LOG_FATAL_CONN Fatal error: bsock.c:133 Unable to
connect to (Client: %{BACULA_HOST:client}|Storage daemon) on %{HOSTNAME}:%
{POSINT}. ERR=(?<berror>%{GREEDYDATA}) BACULA_LOG_NO_CONNECT Warning: bsock.c:127
Could not connect to (Client: %{BACULA_HOST:client}|Storage daemon) on %
{HOSTNAME}:%{POSINT}. ERR=(?<berror>%{GREEDYDATA}) BACULA_LOG_NO_AUTH Fatal
error: Unable to authenticate with File daemon at %{HOSTNAME}. Possible causes:
BACULA_LOG_NOSUIT No prior or suitable Full backup found in catalog. Doing FULL
backup. BACULA_LOG_NOPRIOR No prior Full backup Job record found. BACULA_LOG_JOB
(Error: )?Bacula %{BACULA_HOST} %{BACULA_VERSION} \(%{BACULA_VERSION}\):
BACULA_LOGLINE %{BACULA_TIMESTAMP:bts} %{BACULA_HOST:hostname} JobId %
{INT:jobid}: (%{BACULA_LOG_MAX_CAPACITY}|%{BACULA_LOG_END_VOLUME}|%
{BACULA_LOG_NEW_VOLUME}|%{BACULA_LOG_NEW_LABEL}|%{BACULA_LOG_WROTE_LABEL}|%
{BACULA_LOG_NEW_MOUNT}|%{BACULA_LOG_NOOPEN}|%{BACULA_LOG_NOOPENDIR}|%
{BACULA_LOG_NOSTAT}|%{BACULA_LOG_NOJOBS}|%{BACULA_LOG_ALL_RECORDS_PRUNED}|%
{BACULA_LOG_BEGIN_PRUNE_JOBS}|%{BACULA_LOG_BEGIN_PRUNE_FILES}|%
{BACULA_LOG_PRUNED_JOBS}|%{BACULA_LOG_PRUNED_FILES}|%{BACULA_LOG_ENDPRUNE}|%
{BACULA_LOG_STARTJOB}|%{BACULA_LOG_STARTRESTORE}|%{BACULA_LOG_USEDEVICE}|%
{BACULA_LOG_DIFF_FS}|%{BACULA_LOG_JOBEND}|%{BACULA_LOG_NOPRUNE_JOBS}|%
{BACULA_LOG_NOPRUNE_FILES}|%{BACULA_LOG_VOLUME_PREVWRITTEN}|%
{BACULA_LOG_READYAPPEND}|%{BACULA_LOG_CANCELLING}|%{BACULA_LOG_MARKCANCEL}|%
{BACULA_LOG_CLIENT_RBJ}|%{BACULA_LOG_VSS}|%{BACULA_LOG_MAXSTART}|%

```

```
{BACULA_LOG_DUPLICATE}|%{BACULA_LOG_NOJOBSTAT}|%{BACULA_LOG_FATAL_CONN}|%
{BACULA_LOG_NO_CONNECT}|%{BACULA_LOG_NO_AUTH}|%{BACULA_LOG_NOSUIT}|%
{BACULA_LOG_JOB}|%{BACULA_LOG_NOPRIOR})
```

Aws

```
S3_REQUEST_LINE (?:%{WORD:verb} %{NOTSPACE:request}(?: HTTP/%
{NUMBER:httpversion})?|%{DATA:rawrequest}) S3_ACCESS_LOG %{WORD:owner} %
{NOTSPACE:bucket} \[%{HTTPDATE:timestamp}\] %{IP:clientip} %{NOTSPACE:requester}
%{NOTSPACE:request_id} %{NOTSPACE:operation} %{NOTSPACE:key} (?:"%
{S3_REQUEST_LINE}"|-) (?:%{INT:response:int}|-) (?:-%{NOTSPACE:error_code}) (?:%
{INT:bytes:int}|-) (?:%{INT:object_size:int}|-) (?:%{INT:request_time_ms:int}|-)
(?:%{INT:turnaround_time_ms:int}|-) (?:%{QS:referrer}|-) (?:"%{QS:agent}"?|-)
(?:-%{NOTSPACE:version_id}) ELB_URI_PATHPARAM %{URIPATH:path}(?:%
{URIPARAM:params})? ELB_URI %{URIPROTO:proto}://(?:%{USER}(?:[:|^@]*)?@)?(?:%
{URIHOST:urihost})?(?:%{ELB_URI_PATHPARAM})? ELB_REQUEST_LINE (?:%{WORD:verb} %
{ELB_URI:request}(?: HTTP/%{NUMBER:httpversion})?|%{DATA:rawrequest})
ELB_ACCESS_LOG %{TIMESTAMP_ISO8601:timestamp} %{NOTSPACE:elb} %{IP:clientip}:%
{INT:clientport:int} (?:(%{IP:backendip}:?%{INT:backendport:int})|-) %
{NUMBER:request_processing_time:float} %{NUMBER:backend_processing_time:float} %
{NUMBER:response_processing_time:float} %{INT:response:int} %
{INT:backend_response:int} %{INT:received_bytes:int} %{INT:bytes:int} "%
{ELB_REQUEST_LINE}" See https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns
```

HTML Parser Stage

This parser stage processes the following HTML elements:

- `<title>`
- `<body>` (with tags removed)
- `<meta>`
- `<a>` and `<link>`

Additionally, you can configure JSoup selectors (<https://jsoup.org/cookbook/extracting-data/>) to extract specific HTML and CSS elements from a document and map them to PipelineDocument fields. For example, you could use this to process navigational `DIV` elements one way, then process content-ful `DIV` elements another way.

The HTML Transformation index pipeline stage is deprecated in favor of this parser stage.
HTML and CSS elements can be selected for extraction into new documents or fields:

- To create new documents from selected elements, configure `recordSelector`.
- To create new fields from selected elements, configure `mappings`.

Title, body, metadata, and links are only populated in the parent document. Both of these parameters support JSoup selectors (<https://jsoup.org/cookbook/extracting-data/>), which provides a rich syntax for selecting HTML and CSS elements.

When entering configuration values in the UI, use *unescaped* characters, such as `\t` for the tab character. When entering configuration values in the API, use *escaped* characters, such as `\\t` for the tab character.

Example using HTML elements

The following example retrieves all URL values for links with the CSS class `resources`.

```
<p>For more information, consult the following resources:</p><br> <ul> <li><a  
class="resources" href="https://example.com/books">Books on plants and  
animals</a></li> <li><a class="resources"  
href="https://example.com/illustrations">Botanical illustrations</a></li> <li><a  
href="https://example.com/mammals">Small mammal identification</a></li> </ul>
```

The following table provides an explanation of the example:

Parameter	Example value	Description
Select rule	<code>a.resources</code>	The JSoup parser finds <code><a></code> link elements that are assigned the class <code>resources</code> .
Attribute to map	<code>href</code>	The URL of each <code><a></code> link element, contained within the <code>href</code> attribute, is parsed.
Target field	<code>resourceLinks_ss</code>	The Fusion field to which values are saved.
Multi-valued	<code>false</code>	If <code>true</code> , the parser will save multiple values for the same element. In this case, the <code><a></code> link element can only contain one <code>href</code> value, so the example configuration is <code>false</code> .
In this example, the last link URL, which is for "Small mammal identification," lacks the <code>resources</code> class. As a result, it is not captured by this configuration.		

HTML Content Extraction

The Context Extraction setting in the HTML Parser has a special meaning.

By default, when unchecked or false, the HTML parser will essentially attempt to extract the text of the entire HTML page as the text that will be used in the Solr document.

However, when Content Extract is checked, or true, a set of heuristic rules are applied to attempt to automatically determine what node in the entire page is most likely to be the page content, based on the tree of nodes inside it and the text component of all the sub-nodes.

This may be helpful when you have a variety of different page formats, and also significant text on the pages which is not useful to add to the documents, such as significant text in the page header or footer.

However, since the algorithm is based on heuristics, it is possible that the results could change as the site is altered. When you need a high degree of certainty, we recommend that you use explicit rules of which nodes to extract to match your local configuration.

JSON Parser Stage

JSON parsing converts JSON content from a single document field into one or more new documents. This parser uses Solr's `JsonRecordReader` (https://lucene.apache.org/solr/5_1_0/solr-solrj/org/apache/solr/common/util/JsonRecordReader.html) to split JSON into sub-documents.

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

Text Parser Stage

The Plain Text parser can split a text file by lines or consume it into a single document.

Options for treatment of this filetype include:

- Plain Text Parser Fields
- Number of header rows to skip
- Split on line end or not
- Comment character
- Skip empty lines
- Charset

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

XML Parser Stage

The XML parser stage parses whole XML documents by default, but it can also be configured to parse only specific nodes without loading the entire document into memory. It can also split an XML document into multiple documents. XPATH-like expressions are used to select specific nodes to parse, such as `/posts/row` or `/posts/record`. Nested XML elements are flattened.

To create new documents from selected elements, configure `rootPaths`.

The XML Transformation index pipeline stage is deprecated in favor of this parser stage.

When entering configuration values in the UI, use <i>unescaped</i> characters, such as <code>\t</code> for the tab character. When entering configuration values in the API, use <i>escaped</i> characters, such as <code>\\t</code> for the tab character.

REST APIs

Fusion API services 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:

```
https://FUSION_HOST:8764/api/<endpoint>
```

List 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 USERNAME:PASSWORD https://FUSION_HOST:8764/api/introspect
```

Assign Fusion request IDs

Fusion automatically assigns each API request a Fusion request ID. Optionally, you can assign your own Fusion request IDs.

In this example of a search, the request does not set the Fusion request ID. Fusion assigns one, in this case, `PdNfxc1V2o`:

```
curl -v -u USERNAME:PASSWORD
http://127.0.0.1:8764/api/apps/Movie_Search/query/Movie_Profile?q=movieId_s:202 *
Trying 127.0.0.1... * TCP_NODELAY set * Connected to 127.0.0.1 (127.0.0.1) port
8764 (#0) * Server auth using Basic with user 'admin' > GET
/api/apps/Movie_Search/query/Movie_Profile?q=movieId_s:202 HTTP/1.1 > Host:
127.0.0.1:8764 > Authorization: Basic YWRtaW46UGFzc3dvcmQxMjM= > User-Agent:
curl/7.54.0 > Accept: */* > fusion-request-id: Movie202 > < HTTP/1.1 200 OK <
Server: Jetty(9.3.8.v20160314) < fusion-request-id: PdNfxc1V2o < Content-Type:
application/json; charset=UTF-8 < x-fusion-query-id: f5KGn9KPvv < Set-Cookie:
id=c114b816-df6f-4b17-97e0-bb87ef815bf0; Path=/; HttpOnly < Content-Length: 1707 <
Connection: close < * Closing connection 0 (search result omitted)
```

Here, we assign the Fusion request ID `Movie202`:

```
curl -v -H 'fusion-request-id: Movie202' -u USERNAME:PASSWORD
http://127.0.0.1:8764/api/apps/Movie_Search/query/Movie_Profile?q=movieId_s:202 *
Trying 127.0.0.1... * TCP_NODELAY set * Connected to 127.0.0.1 (127.0.0.1) port
8764 (#0) * Server auth using Basic with user 'admin' > GET
/api/apps/Movie_Search/query/Movie_Profile?q=movieId_s:202 HTTP/1.1 > Host:
127.0.0.1:8764 > Authorization: Basic YWRtaW46UGFzc3dvcmQxMjM= > User-Agent:
curl/7.54.0 > Accept: */* > fusion-request-id: Movie202 > < HTTP/1.1 200 OK <
Server: Jetty(9.3.8.v20160314) < fusion-request-id: Movie202 < Content-Type:
application/json; charset=UTF-8 < x-fusion-query-id: f5KGn9KPvv < Set-Cookie:
id=c114b816-df6f-4b17-97e0-bb87ef815bf0; Path=/; HttpOnly < Content-Length: 1707 <
Connection: close < * Closing connection 0 (search result omitted)
```

With a Fusion request ID, you can trace requests in log files and *filter log file dashboards by the request ID*

Apps API

The Apps API manages your Fusion apps. To export or import an app, see the *Objects API*. You can also *import an app using the Fusion UI*.

Examples

List all configured apps

Input

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/apps
```

Output

```
[ {
  "id" : "1032724d-848a-4588-9503-c20702aa3a7b",
  "name" : "Test App",
  "properties" : {
    "headerImageName" : "headerImage2",
    "tileColor" : "apps-red"
  }
}, {
  "id" : "7fc4d4e5-d3bd-4b21-b85e-9aa29c47b281",
  "name" : "Movie Search",
  "properties" : {
    "headerImageName" : "headerImage3",
    "tileColor" : "apps-dark"
  }
} ]
```

Get the configuration for a specific app

Input

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/apps/8fc4d4e5-d3bd-4b21-b85e-9aa29c47b281
```

Output

```
{
  "id" : "7fc4d4e5-d3bd-4b21-b85e-9aa29c47b281",
  "name" : "Movie Search",
  "properties" : {
    "headerImageName" : "headerImage3",
    "tileColor" : "apps-dark"
  }
}
```

Input

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d '{"id":  
"MyNewApp", "name":"My new app", "description":"A really great new app"}'  
https://FUSION_HOST:8764/api/apps
```

Output

```
{  
  "id" : "MyNewApp",  
  "name" : "My new app",  
  "description" : "A really great new app"  
}
```

API specification

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.

Realms API

Realms are used to authenticate users across several different user access control systems.

There are two types of realms currently supported: native, which uses Fusion itself to manage users and passwords, and LDAP, which uses an LDAP server as the source of truth for usernames and passwords.

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.

Create, Update, Delete or List Realms

The path for this request is:

`/api/realms-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.

In order to see this object within the *Fusion UI*, it **must** be associated with an app. To do this, create the object using the `/apps` endpoint.

A PUT request updates a realm.

A DELETE request removes the realm.

Input

Parameter	Description
name <i>Required</i>	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 <i>Required</i>	If true , the realm is available for users to use with system authentication.
realmType <i>Required</i>	String value for realm type.

Native realms store users in the Fusion database.

LDAP realms connect to an LDAP server to verify the user's ID and password. Configuration 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 (<code>{}</code>) 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

Get details of the default 'native' realm:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST: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 USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d '{"realmType":"ldap", "name":"dev-ldap", "enabled":true, "config":{"host":"fusion-host", "port":10636 , "ssl":true, "bindDn":"uid={},ou=users,dc=security,dc=example,dc=com"} }' https://FUSION_HOST: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":"fusion-host" } }
```

API specification

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.

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.

In order to see this object within the *Fusion UI*, it **must** be associated with an app. To do this, create the object using the `/apps` endpoint.

A PUT request updates an existing role.

A DELETE request will remove the role configuration.

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 <i>Required</i>	A string containing the role name.
desc <i>Optional</i>	A string containing a brief text description, for display on the Access Control "ROLES" panel.
permissions <i>Optional</i>	A list of permissions, specified in JSON notation. See section <i>Permissions</i> for details.
uiPermissions <i>Optional</i>	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\n\"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" ] }
```

Examples

Get the details for the role with id '3416c03a-31df-4103-b446-358f6790af3e':

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST: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." }
```

API specification

Sessions API

The Sessions API provides a way for clients to manage Fusion sessions and cookies. When a client successfully authenticates, Fusion will create a unique identifier for the session. This identifier becomes the session key and is mapped to the client. The authentication response contains an HTTP cookie, which is typically stored by the client for later use. By sending this cookie back to Fusion, a client can securely access Fusion without having to re-authenticate.

By default, sessions time out after 1 hour of no activity. A default maximum limit of 8 hours ensures that sessions are cleared, regardless of activity. Once a session times out, clients must re-authenticate to receive a new session cookie.

The following properties control session timeouts *in seconds*:

- `com.lucidworks.apollo.admin.session.timeout` - Maximum amount of time before a session, active or otherwise, expires.
- `com.lucidworks.apollo.admin.session.idle.timeout` - Maximum amount of time before an idle session expires.

To set these values for the Fusion proxy, please edit `fusion.properties` and add them to the `proxy.jvmOptions` variable. Each value must be added with a `-D` prefix. For example, to set the session timeout to 4 hours and the idle timeout to 10 minutes, use the following:

To set these values for the Fusion proxy, please edit `fusion.properties` and add them to the `proxy.jvmOptions` variable. Each value must be added with a `-D` prefix. For example, to set the session timeout to 4 hours and the idle timeout to 10 minutes, use the following:

```
proxy.jvmOptions = -Xmx512m -Dcom.lucidworks.apollo.admin.session.timeout=14400 -Dcom.lucidworks.apollo.admin.session.idle.timeout=600
```

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.

In order to see this object within the *Fusion UI*, it **must** be associated with an app. To do this, create the object using the `/apps` endpoint.

Input

Parameter	Description
username <i>Required</i>	The username to use in authentication.
password <i>Required</i>	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

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":"USERNAME", "password":"PASSWORD"}'
https://FUSION_HOST:8764/api/session?realmName=REALM_NAME
```

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)
```

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 https://FUSION_HOST:8764/api/collections
```

View Session Details

To view session details, including the current user record, all roles directly assigned to that user, and all roles inherited from the realm by that user, use the following path for the request:

`/api/session`

Examples

REQUEST

```
curl 'https://FUSION_HOST:8764/api/session' -H 'Cookie: id=416925d6-6d26-4afd-
b31d-ced61714d287'
```

RESPONSE

```
{ "user": { "id": "abc123", "username": "foo" }, "roles": [ {"name": "bar"} ] }
```

End a Session

The path for this request is:

`/api/session`

Examples

Sessions can be ended automatically in the Fusion UI by clicking **Account Settings**  **> Log out**.

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.

Create, Update, Delete or List Users

The path for this request is:

`/api/users/<id>`

where *<id>* is the user ID.

In order to see this object within the *Fusion UI*, it **must** be associated with an app. To do this, create the object using the `/apps` endpoint.

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 <i>Required</i>	The username. This is distinct from their ID, which is assigned by the system as a unique identifier.
password <i>Required</i>	The user's password. Required when creating a new user. The user's password is not returned in the output of any request.
passwordConfirm <i>Required</i>	When creating a user or updating a user's password, you must confirm the defined password.
realmName <i>Required</i>	The realm the user belongs to, which defines how they authenticate against the system.
permissions <i>Optional</i>	The permissions that have been defined for this user that are not inherited from their assigned role.
inheritedPermissions <i>Optional</i>	The user's specific permissions that are inherited from their role assignment.
roleNames <i>Optional</i>	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

Get all the configured users of the system:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST: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 USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d '{"username":"guest", "password":"password456", "passwordConfirm":"password456", "realmName": "native"}' https://FUSION_HOST: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 USERNAME:PASSWORD -X PUT -H 'Content-type: application/json' -d '{"name":"joe.smith", "realmName":"myLDAP", "roleNames":["admin"]}' https://FUSION_HOST:8764/api/users/aeafa7ffc-23f1-45ac-b326-f7bb007d3b9d
```

RESPONSE

None.

API specification

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 `https://FUSION_HOST:FUSION_PORT/data/nlp` directory, or files that you have created on your own.

You can also work with blobs in the Fusion UI using the *blob manager*.

Blobs uploaded to Solr with this REST API are stored in the `system_blobs` collection.

Blob Types

A `resourceType` query parameter can be used to specify the a blob type. For example, specify `plugin:connector` when uploading a connector, like this:

```
curl -H 'content-type:application/zip' -X PUT 'fusion-  
host:8764/api/blobs/myplugin?resourceType=plugin:connector' --data-binary  
@myplugin.zip
```

The complete list of valid values for `resourceType` is below:

Type	Description
<code>banana</code>	A Banana <i>dashboard</i>
<code>catalog</code>	An <i>analytics catalog</i>
<code>driver:jdbc</code>	A <i>JDBC driver</i>
<code>file:js-index</code>	A JavaScript file for use with a <i>Managed Javascript index stage</i> .
<code>file:js-query</code>	A JavaScript file for use with a <i>Managed Javascript query stage</i> .
<code>file</code>	Any uploaded file, such as from the <i>Quickstart</i> or the <i>Index Workbench</i> .
<code>model:ml-model</code>	A <i>machine learning model</i> (Fusion AI only)
<code>model:open-nlp</code>	An <i>OpenNLP model</i> (Fusion AI only)
<code>other</code>	A blob of unknown type If no <code>resourceType</code> is specified on upload, "other" is assigned by default.
<code>plugin:connector</code>	A <i>connector plugin</i>

Examples

Upload a JavaScript file to the blob store

REQUEST

```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-Type: text/javascript' --data-binary @query-pipeline-js-stage.js https://FUSION_HOST:8764/api/blobs/query-pipeline-js-stage.js?resourceType=file:js-query
```

The response is empty.

Verify that the JavaScript blob was uploaded

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/blobs?resourceType=file:js-query
```

RESPONSE

```
[{"id": "query-pipeline-js-stage.js", "path": "/query-pipeline-js-stage.js", "dir": "/", "filename": "query-pipeline-js-stage.js", "contentType": "text/javascript", "size": 0, "modifiedTime": "2018-09-11T16:23:17.743Z", "version": 1611328911094841344, "md5": "d41d8cd98f00b204e9800998ecf8427e", "metadata": { "resourceType": "file:js-query" } }]
```

Upload an OpenNLP sentence model binary file to the blob store

REQUEST

```
curl -u USERNAME:PASSWORD -X PUT --data-binary @data/nlp/models/en-sent.bin -H 'Content-type: application/octet-stream' https://FUSION_HOST:8764/api/blobs/sentenceModel.bin
```

In this example, 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 have previously saved in the blob store

REQUEST

```
curl -u USERNAME:PASSWORD -X HEAD https://FUSION_HOST:8764/api/blobs/sentenceModel.bin
```

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 USERNAME:PASSWORD -X PUT --data-binary @postgresql-42.0.0.jar -H 'Content-length: 6' -H 'Content-type: application/octet-stream' https://FUSION_HOST:8764/api/blobs/good/to/go/jdbcDriverFile?resourceType=driver:jdbc
```

RESPONSE

```
{ "name": "good/to/go/jdbcDriverFile", "contentType": "application/octet-stream", "size": 6, "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 USERNAME:PASSWORD https://FUSION_HOST:8764/api/blobs?resourceType=driver:jdbc
```

RESPONSE

```
[{ "name": "good/to/go/jdbcDriverFile", "contentType": "application/octet-stream", "size": 6, "modifiedTime": "2017-04-04T06:21:53.465Z", "version": 156372766574524416, "md5": "b1946ac92492d2347c6235b4d2611184", "metadata": { "subtype": "driver:jdbc", "resourceType": "driver:jdbc" } }]
```

API specification

Catalog API

The Fusion Catalog is a collection of one or more analytics projects, and each project is a collection of data assets, such as tables or relations. Fusion comes with a built-in project called "fusion".

The Fusion Catalog API provides access to assets by data analysis applications that can perform SQL or Solr queries. It includes endpoints for finding, retrieving, and manipulating projects and assets using basic keyword and metadata-driven search.

By default, non-admin Fusion users do not have access to Catalog objects. However, the Catalog API itself does not enforce any permissions, so a user who bypasses the auth proxy has full access to all projects and assets. An admin can grant permissions to Catalog endpoints for users; see *Access Control*.

Intra-shard splits

If your Spark cluster has more available executor slots than the number of shards, then you can increase parallelism when reading from Solr by splitting each shard into sub-ranges using a split field. The sub range splitting enables faster fetching from Solr by increasing the number of tasks in Solr. This should only be used if there are enough computing resources in the Spark cluster.

Shard splitting is enabled by default, with two sub-ranges per shard. See *Configuration options* below for shard splitting parameters.

Body attributes

For PUT and POST requests, these are valid JSON body attributes:

Name	Type	Description
<code>projectId</code>	String	The project name
<code>name</code>	String	The asset name
<code>assetType</code>	DataAssetType	One of: + * project * table * relation * field * udf * metric
<code>description</code>	String	A string describing this asset
<code>sourceUri</code>	String	A URI to the data source
<code>owner</code>	String	The user that owns the asset
<code>ownerEmail</code>	String	The owner's email address
<code>tags</code>	Set<String>	A set of arbitrary category strings
<code>format</code>	String	The format of the underlying data source
<code>options</code>	List<String>	A list of options for the underlying data source. See <i>Configuration options</i> below for valid options.
<code>filters</code>	List<String>	A set of Solr query parameters to filter the request
<code>sql</code>	String	A SQL statement to execute
<code>cacheOnLoad</code>	boolean	'True' to cache the dataset in Spark on catalog project initialization
<code>dependsOn</code>	List<String>	A list of other assets to load before initializing this data asset
<code>createdOn</code>	Date	The asset's creation date, in ISO-8601 format; otherwise the current timestamp is used

Configuration options

Name	Description	Default
collection	The Solr collection name.	None
zkhost	A ZooKeeper connect string is the list of all servers and ports for the current ZooKeeper cluster. For example, if running a single-node Fusion developer deployment with embedded ZooKeeper, the connect string is <code>fusion-host:9983/lwfusion/3.1.0/solr</code> . If you have an external 3-node ZooKeeper cluster running on servers <code>"zk1.acme.com"</code> , <code>"zk2.acme.com"</code> , <code>"zk3.acme.com"</code> , all listening on port 2181, then the connect string is <code>zk1.acme.com:2181,zk2.acme.com:2181,zk3.acme.com:2181</code>	The <code>connecting</code> of the default <i>search cluster</i>
query	A Solr query that limits the rows to load into Spark. For example, to only load documents that mention "solr": <pre>options("query","body_t:solr")</pre>	<code>**:</code>
fields	A subset of fields to retrieve for each document in the results, such as: <pre>options("fields","id,author_s,favorited_b,...")</pre> You can also specify an alias for a field using Solr's field alias syntax, such as <code>author:author_s</code> . If you want to invoke a function query, such as <code>rord()</code> , then you'll need to provide an alias, such as <code>ord_user:ord(user_id)</code> . If the return type of the function query is something other than <code>int</code> or <code>long</code> , then you'll need to specify the return type after the function query, such as: <pre>foo:div(sum(x,100),max(y,1)):double</pre> If you request Solr function queries, then the library must use the <code>/select</code> Solr handler to make the request as exporting function queries through <code>/export</code> is not supported by Solr.	By default, all stored fields for each document are pulled back from Solr.
rows	The number of rows to retrieve from Solr per request; do not confuse this with <code>max_rows</code> (see below). This is not the maximum number of rows to read from Solr. All matching rows on the backend are read. The rows parameter is the page size. Behind the scenes, the implementation uses either deep paging cursors or Streaming API and response streaming, so it is usually safe to specify a large number of rows. By default, the implementation uses 1000 rows but if your documents are smaller, you can increase this to 10000. Using too large a value can put pressure on the Solr JVM's garbage collector. Example: <code>options("rows","10000")</code>	<code>1000</code>

Name	Description	Default
max_rows	The maximum number of rows; only applies when using the <code>/select</code> handler. The library will issue the query from a single task and let Solr do the distributed query processing. No paging is performed, that is, the <code>rows</code> param is set to <code>max_rows</code> when querying. Consequently, this option should not be used for large <code>max_rows</code> values, rather you should just retrieve all rows using multiple Spark tasks and then re-sort with Spark if needed. Example: <code>options("max_rows", "100")</code>	None
request_handler	Set the Solr request handler for queries. This option can be used to export results from Solr via <code>/export</code> handler which streams data out of Solr. See Exporting Result Sets for more information. The <code>/export</code> handler needs fields to be explicitly specified. Please use the <code>fields</code> option or specify the fields in the query. Example: <code>options("request_handler", "/export")</code>	<code>/select</code>
splits	Enable shard splitting on default field <i>version</i>. Example: <code>options("splits", "true")</code> The above option is equivalent to <code>options("split_field", "version")</code>	False
split_field	The field to split on can be changed using <code>split_field</code> option. Example: <code>options("split_field", "id")</code>	<i>version</i>
splits_per_shard	Split the shard into evenly-sized splits using filter queries. You can also split on a string-based keyword field but it should have sufficient variance in the values to allow for creating enough splits to be useful. In other words, if your Spark cluster can handle 10 splits per shard, but there are only 3 unique values in a keyword field, then you will only get 3 splits. Keep in mind that this is only a hint to the split calculator and you may end up with a slightly different number of splits than what was requested. Example: <code>options("splits_per_shard", "30")</code>	20
flatten_multivalued	Flatten multi-valued fields from Solr. Example: <code>options("flatten_multivalued", "false")</code>	true
dv	Fetch the docValues that are indexed but not stored by using function queries. Should be used for Solr versions lower than 5.5.0. Example: <code>options("dv", "true")</code>	false

Name	Description	Default
<code>sample_seed</code>	Read a random sample of documents from Solr using the specified seed. This option can be useful if you just need to explore the data before performing operations on the full result set. By default, if this option is provided, a 10% sample size is read from Solr, but you can use the <code>sample_pct</code> option to control the sample size. Example: <code>options("sample_seed", "5150")</code>	None
<code>sample_pct</code>	The size of a random sample of documents from Solr; use a value between 0 and 1. Example: <code>options("sample_pct", "0.05")</code>	0.1
<code>skip_non_dv</code>	Skip all fields that are not docValues. Example: <code>options("skip_non_dv", "true")</code>	false

Examples

Define a "movielens" project:

```
curl -u USERNAME:PASSWORD -X POST -H "Content-type:application/json" \
-d '{
  "name": "movielens",
  "assetType": "project",
  "description": "tables and views for the movielens project",
  "tags": ["movies","users"],
  "cacheOnLoad": false
}' "https://FUSION_HOST:6764/api/catalog"
```

Add a "ratings" table to the "movielens" project:

```
curl -u USERNAME:PASSWORD -X POST -H "Content-type:application/json" -d '{
  "name": "ratings",
  "assetType": "table",
  "projectId": "movielens",
  "description": "movie ratings data",
  "tags": ["movies"],
  "format": "solr",
  "cacheOnLoad": true,
  "options": ["collection -> movielens_ratings", "fields ->
user_id,movie_id,rating,rating_timestamp"]
}' "https://FUSION_HOST:6764/api/catalog/movielens/assets"
```

Issue a SQL statement against the "ratings" table:

```
curl -u USERNAME:PASSWORD -X POST -H "Content-type:application/json" -d '{
  "name": "ratings",
  "assetType": "table",
  "projectId": "movielens",
  "description": "movie ratings data",
  "tags": ["movies"],
  "format": "solr",
  "cacheOnLoad": true,
  "options": ["collection -> movielens_ratings", "fields ->
user_id,movie_id,rating,rating_timestamp"]
}' "https://FUSION_HOST:6764/api/catalog/movielens/query"
```

Issue a SQL query against the "movielens" project:

```
curl -u USERNAME:PASSWORD -X POST -H "Content-Type:application/json" -d '{
"sql":"SELECT m.title as title, solr.aggCount as aggCount FROM movies m INNER
JOIN (SELECT movie_id, COUNT(*) as aggCount FROM ratings WHERE rating >= 4 GROUP
BY movie_id ORDER BY aggCount desc LIMIT 10) as solr ON solr.movie_id =
m.movie_id ORDER BY aggCount DESC"
}' https://FUSION_HOST:6764/api/catalog/movielens/query
```

Load a catalog table from a Postgres database:

```
curl -u USERNAME:PASSWORD -X POST -H "Content-type:application/json" -d '{
  "projectId": "nyc_taxi",
  "assetType": "table",
  "name": "trips",
  "sourceUri": "http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml",
  "owner": "Joe Example",
  "ownerEmail": "examplejoe@gmail.com",
  "description": "The NYC taxi trip data stored in Postgres using tools provided
by https://github.com/toddwschneider/nyc-taxi-data",
  "tags": ["nyc", "taxi", "postgres", "trips"],
  "format": "jdbc",
  "cacheOnLoad": true,
  "options": ["url -> ${nyc_taxi_jdbc_url}", "dbtable -> trips", "partitionColumn ->
id", "numPartitions -> 4", "lowerBound -> 0", "upperBound -> $MAX(id)", "fetchSize
-> 1000"],
  "filters": ["pickup_latitude >= -90 AND pickup_latitude <= 90 AND
pickup_longitude >= -180 AND pickup_longitude <= 180", "dropoff_latitude >= -90
AND dropoff_latitude <= 90 AND dropoff_longitude >= -180 AND dropoff_longitude <=
180"],
  "sql": "SELECT
id,cab_type_id,vendor_id,pickup_datetime,dropoff_datetime,store_and_fwd_flag,rate
_code_id,passenger_count,trip_distance,fare_amount,extra,mta_tax,tip_amount,tolls
_amount,ehail_fee,improvement_surcharge,total_amount,payment_type,trip_type,
concat_ws(',',pickup_latitude,pickup_longitude) as pickup,
concat_ws(',',dropoff_latitude,dropoff_longitude) as dropoff FROM trips"
}' "https://FUSION_HOST:6764/api/catalog/nyc_taxi/assets"
```

Create a data asset using a streaming expression:

```
curl -u USERNAME:PASSWORD -X POST -H "Content-type:application/json" -d '{
  "name": "movie_ratings",
  "assetType": "table",
  "projectId": "movielens",
  "description": "movie ratings data",
  "tags": ["movies"],
  "format": "solr",
  "cacheOnLoad": true, "options": ["collection -> movielens_ratings", "expr ->
hashJoin(search(movielens_ratings,q=\"*:*\",fl=\"movie_id,user_id,rating\",sort=\"
movie_id
asc\",qt=\"\"/export\",partitionKeys=\"movie_id\"),hashed=search(movielens_movies,
q=\"*:*\",fl=\"movie_id,title\",sort=\"movie_id
asc\",qt=\"\"/export\",partitionKeys=\"movie_id\"),on=\"movie_id\")"]
}' 'https://FUSION_HOST:6764/api/catalog/movielens/assets'
```

Send a Solr query:

```
curl -u USERNAME:PASSWORD -X POST -H "Content-Type:application/json" -d '{
  "solr": "*:*",
  "requestHandler": "/select",
  "collection": "movielens_movies",
  "params": {
    "facet": "on",
    "facet.field": "genre",
    "rows": 0
  }
}' https://FUSION_HOST:6764/api/catalog/movielens/query
```

Send a Solr query using a streaming expression:

```
curl -u USERNAME:PASSWORD -X POST -H "Content-Type:application/json" --data-
binary @streaming_join.json https://FUSION_HOST:6764/api/catalog/movielens/query
{
  "solr": "hashJoin(search(movielens_ratings, q=*:*, qt=\"\"/export\",
fl=\"user_id,movie_id,rating\", sort=\"movie_id asc\",
partitionKeys=\"movie_id\"), hashed=search(movielens_movies, q=*:*,
fl=\"movie_id,title\", qt=\"\"/export\", sort=\"movie_id
asc\",partitionKeys=\"movie_id\"),on=\"movie_id\"),
  "collection": "movielens_ratings",
  "requestHandler": "/stream"
}
```

API specification

Collection Features API

The Collection Features API lets you observe the status of features across collections. To manipulate features for a specific collection, use the *Collections API*.

There are four available features:

Property	Description
signals	Create 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 <i>Signals</i> for more information.
partitionByTime	Partition the corresponding Solr collection by time; see <i>Time-Based Partitioning</i> .
recommendations	Get configuration information about recommendations. Parameters (<code>params</code>) are: <code>idField</code> . The document field that contains the document ID. <code>itemsForUser</code> . Whether items-for-user recommendations are enabled (<code>true</code>) or disabled (<code>false</code>). <code>itemsForQuery</code> . Whether items-for-query recommendations are enabled (<code>true</code>) or disabled (<code>false</code>). <code>itemsForItem</code> . Whether items-for-query recommendations are enabled (<code>true</code>) or disabled (<code>false</code>). <code>queriesForQuery</code> . Whether queries-for-query recommendations are enabled (<code>true</code>) or disabled (<code>false</code>).

Examples

List the status of all features:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/features/status
```

RESPONSE

```
[ { "name" : "searchLogs", "collectionId" : "demo", "params" : { }, "enabled" : false }, { "name" : "signals", "collectionId" : "demo", "params" : { }, "enabled" : false } ]
```

API specification

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.

Examples

In order to see this object within the *Fusion UI*, it **must** be associated with an app. To do this, create the object using the `/apps` endpoint.

Create a new collection called 'newCollection', with appropriate SolrCloud environment settings:

REQUEST

```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-type: application/json' -d
'{"solrParams":{"replicationFactor":1,"numShards":1}}'
https://FUSION_HOST:8764/api/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':

REQUEST

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d
'{"id":"local-collection1", "searchClusterId":"Solr4.10", "solrParams":
{"name":"collection1"}}' https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD -X DELETE
https://FUSION_HOST:8764/api/collections/newCollection?purge=false
```

Get the status of the 'demo' collection:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/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" : "https://FUSION_HOST:8983/solr", "node_name" : "fusion-
host:8983_solr" } } } } }
```

Get stats for the 'demo' collection:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/collections/demo/stats
```

RESPONSE

```
{ "collectionId" : "demo", "documentCount" : 536, "requestCount" : 6, "qps" :
28.34716542561849, "sizeInBytes" : 7646045, "lastModified" : "2014-05-
19T19:58:33.545Z" }
```

Add the recommendations feature for the 'tempy' collection:

[For more information on collection features and feature management, see *Collection Features API* **]**

Use this structure to send the request:

```
https://FUSION_HOST:{api-port}/api/collections/{collection}/features/{feature}
```



```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-type: application/json' -d @enable-recommendations.json https://FUSION_HOST:8764/api/apps/Recommender-App/collections/tempy/features/recommendations -v
```

Make sure to change the `collection-id` in the JSON and save as a file or send directly in the string.

Recommendations JSON:

```
{ "name": "recommendations", "collectionId": "tempy", "params": { "idField": "id", "itemsForUser": true, "itemsForQuery": false, "itemsForItem": true, "queriesForQuery": false }, "enabled": true }
```

Signals JSON:

```
{ "name": "signals", "collectionId": "tempy", "params": {}, "enabled": false },
```

Search Logs JSON:

```
{ "name": "searchLogs", "collectionId": "tempy", "params": {}, "enabled": false }
```

Partition by time JSON:

```
{ "name": "partitionByTime", "collectionId": "tempy", "params": {}, "enabled": false }
```

API specification

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.

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.

Examples

Show the configuration items that include the pattern 'zk-connect', with verbose enabled:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/configurations?verbose=true&pattern=zk-connect
```

RESPONSE

```
{
  "com.lucidworks.apollo-admin.config/zk-connect" : [{
    "value" : "fusion-host:9983",
    "location" : "system properties"
  } ]
}
```

Get the configuration items from the Connectors JVM that include the term 'connect':

REQUEST

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:8984/connectors/api/v1/configurations?pattern=connect
```

Get the configuration items from the Fusion JVM that start with 'com.lucidworks':

REQUEST

```
https://FUSION_HOST:8764/api/configurations?prefix=com.lucidworks
```

Change the default allowed recommendation types to include 'itemsForItem':

REQUEST

```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-type: application/json' -d  
'{"itemsForQuery", "termsForDocument", "itemsforItem"}'  
https://FUSION_HOST:8764/api/configurations/com.lucidworks.apollo.service.recomme  
nd.allowed.types
```

RESPONSE

None. Check the setting again to confirm the changes.

API specification

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 *Jobs and 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*.

Examples

View the history of the index-pipelines service:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD https://FUSION_HOST:8764/api/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
  }
]
}

```

API specification

Nodes API

The Nodes API allows users to check active services according to their hosts.

Examples

Show each service:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/nodes/up
```

RESPONSE

```
{ "history::v1" : [ "https://FUSION_HOST:8984/connectors/v1/history",
"https://FUSION_HOST:8764/api/history" ], "query-pipelines::v1" : [
"https://FUSION_HOST:8764/api/query-pipelines",
"https://FUSION_HOST:8764/api/query-pipelines" ], "solrconfig::v1" : [
"https://FUSION_HOST:8764/api/collections//solr-config" ], "solrAdmin::v1" : [
"https://FUSION_HOST:8764/api/solrAdmin",
"https://FUSION_HOST:8984/connectors/v1/solrAdmin" ], "stopwords::v1" : [
"https://FUSION_HOST:8764/api/stopwords" ], "collections::v1" : [
"https://FUSION_HOST:8764/api/collections" ], "index-stages::v1" : [
"https://FUSION_HOST:8984/connectors/v1/index-stages",
"https://FUSION_HOST:8764/api/index-stages" ], "nodes::v1" : [
"https://FUSION_HOST:8984/connectors/v1/nodes",
"https://FUSION_HOST:8764/api/nodes" ], "index-profiles::v1" : [
"https://FUSION_HOST:8764/api/collections//index-profiles",
"https://FUSION_HOST:8984/connectors/v1/collections//index-profiles" ],
"schema::v1" : [ "https://FUSION_HOST:8764/api/collections//schema" ], "solr::v1"
: [ "https://FUSION_HOST:8984/connectors/v1/solr",
"https://FUSION_HOST:8764/api/solr" ], "query-stages::v1" : [
"https://FUSION_HOST:8764/api/query-stages" ], "blobs::v1" : [
"https://FUSION_HOST:8764/api/blobs" ], "connectors::v1" : [
"https://FUSION_HOST:8984/connectors/v1/connectors" ], "recommend::v1" : [
"https://FUSION_HOST:8764/api/recommend" ], "configurations::v1" : [
"https://FUSION_HOST:8764/api/configurations",
"https://FUSION_HOST:8984/connectors/v1/configurations" ], "system::v1" : [
"https://FUSION_HOST:8764/api/system",
"https://FUSION_HOST:8984/connectors/v1/system" ], "index-pipelines::v1" : [
"https://FUSION_HOST:8764/api/index-pipelines",
"https://FUSION_HOST:8764/api/index-pipelines",
"https://FUSION_HOST:8764/api/index-pipelines",
"https://FUSION_HOST:8764/api/index-pipelines",
"https://FUSION_HOST:8764/api/index-pipelines",
"https://FUSION_HOST:8764/api/index-pipelines",
"https://FUSION_HOST:8764/api/index-pipelines",
"https://FUSION_HOST:8984/connectors/v1/index-pipelines",
"https://FUSION_HOST:8984/connectors/v1/index-pipelines",
"https://FUSION_HOST:8764/api/index-pipelines",
"https://FUSION_HOST:8764/api/index-pipelines" ], "query-pipeline-templates::v1"
: [ "https://FUSION_HOST:8764/api/pipeline-templates/query" ], "registration::v1"
: [ "https://FUSION_HOST:8984/connectors/v1/registration",
"https://FUSION_HOST:8764/api/registration" ], "searchLogs::v1" : [
"https://FUSION_HOST:8764/api/searchLogs" ], "scheduler::v1" : [
"https://FUSION_HOST:8764/api/scheduler" ], "aggregator::v1" : [
"https://FUSION_HOST:8764/api/aggregator" ], "query-profiles::v1" : [
"https://FUSION_HOST:8764/api/collections//query-profiles" ], "usage::v1" : [
"https://FUSION_HOST:8764/api/usage",
"https://FUSION_HOST:8984/connectors/v1/usage" ], "introspect::v1" : [
"https://FUSION_HOST:8764/api/introspect",
"https://FUSION_HOST:8984/connectors/v1/introspect" ], "searchCluster::v1" : [
"https://FUSION_HOST:8764/api/searchCluster",
"https://FUSION_HOST:8984/connectors/v1/searchCluster" ], "features::v1" : [
"https://FUSION_HOST:8764/api/features" ], "index-pipeline-templates::v1" : [
"https://FUSION_HOST:8984/connectors/v1/pipeline-templates/index",
"https://FUSION_HOST:8764/api/pipeline-templates/index" ], "synonyms::v1" : [
"https://FUSION_HOST:8764/api/synonyms" ], "reports::v1" : [
"https://FUSION_HOST:8764/api/reports" ], "signals::v1" : [
"https://FUSION_HOST:8764/api/signals" ] }
```

System API

The System REST API allows you to monitor the system performance.

Examples

Metric names

Get metric names that start with 'mem.heap':

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/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 is readable, and show the sample data, if any:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/system/metrics?
pattern=com.lucidworks.apollo.*pipeline.index.*&pretty=true&showSamples=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, 0.001976000000000003 ],
    "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 USERNAME:PASSWORD https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD -X PUT https://FUSION_HOST:8764/api/system/buffers
```

Ping

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/system/ping
```

RESPONSE

pong

API specification

Usage API

The Usage API sends system usage data to Lucidworks. See *System Usage Monitor* for details.

The Usage API is removed in Fusion 5.x

Examples

Get the latest data that will be sent to Lucidworks on the next run:

INPUT

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/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
} ]
```

API specification

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 Jobs API*
- *Connector Plugins API*
- *Connector Status API*

Connector Datasources API

The connector datasources API is used to create and configure datasources, look at the crawl database, or clear items and tables from the crawl database.

See *Connectors* and *Datasources* for related information.

Working with the crawl database

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 `/api/connectors/datasources/{id}/db` endpoints allow looking into the crawl database and dropping tables or clearing the database.

The connectors that support the crawl database are currently `lucid.fs` and `lucid.solrxml`. 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.

Examining a crawlDB

The output from a GET request to `/api/connectors/datasources/<id>/db` 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.

Drop tables from a crawlDB

The output from a DELETE request to `/api/connectors/datasources/<id>/db/<table>` 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.

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.

Get or delete items from a crawlDB

A GET request to `/api/connectors/datasources/<id>/db/items/<item>` retrieves information about an item or items.

A DELETE request removes the information *from the Crawl Database only*. Note that this does not affect the Solr Index.

Examples

Get datasources assigned to the "demo" collection:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/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://FUSION_HOST: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 } } ]
```

Create a datasource to index a file in the Blob Store using the MyApp pipeline, parser, and collection for an app called "MyApp"
REQUEST

```
curl -u USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d '{ "id":
"fileupload-example", "connector": "lucid.fileupload", "type": "fileupload",
"description": "Metadata and abstracts for a selection of documents in
arXiv.org", "pipeline": "MyApp", "parserId": "MyApp", "properties": {
"collection": "MyApp", "fileId": "quickstart/arxiv-fusion.json", "mediaType":
"application/octet-stream" } }'
https://FUSION_HOST:8764/api/apps/MyApp/connectors/datasources
```

RESPONSE

```
{ "parserId": "MyApp", "created": "2021-06-29T20:00:57.534Z", "coreProperties":
{}, "modified": "2021-06-29T20:00:57.534Z", "id": "fileupload-example", "type":
"fileupload", "properties": { "collection" : "MyApp", "fileId" : "IMDB
Dataset.csv", "mediaType" : "application/octet-stream" } }
```

Clear a datasource

You can use both of these APIs in order to fully clear the data:

REQUEST

```
curl -X POST 'http://localhost:8764/api/apollo/solr/COLLECTION_NAME/update?
commit=true' -H 'Content-Type: application/json' --data-binary '{"delete":
{"query": "_lw_data_source_s:DATASOURCENAME"}}' `curl -X DELETE -u
USERNAME:PASSWORD
'http://localhost:8764/api/apollo/connectors/datasources/DATASOURCENAME/db' `
```

The first clears the data from the datasource but does not clear the crawlDB. So if you attempt to index the same document set again, indexing will skip the documents because they are still in the crawl DB. If you send the command to delete the crawlDB afterward, you can then reload the docs.

The `validate=true` element in the create datasource command only validates the datasource. It does *not* automatically save the datasource. An example using this element is:

```
[https://FUSION_HOST:8764/api/apps/  
(https://FUSION_HOST:8764/api/apps/)]APP_NAME/datasources?validate=true
```

The `POST /datasources` section of the API specification allows you to set the `validate` element for testing and use.

API specification

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.

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

Show the cumulative history for a datasource named "Lucid5Docs":

REQUEST

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD
https://FUSION_HOST:8764/api/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" }
```

API specification

Connector Jobs API

This API is deprecated. Use the *Jobs API* instead.

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.

API specification

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.

The output of a GET request will include the status of the registry and any messages reported.

API specification

Experiments API

Use the Experiments API to compare different configuration variants and determine which ones are most successful. For example, configure different variants that use different query pipelines, and then analyze and compare search activity to see which variant best meets your goals.

Experiments let you evaluate multiple *variants* which can differ from each other by pipeline, collection, search handler, request parameters, or some combination of those. An experiment uses one or more metrics to measure the performance of each variant, so they can be compared quantitatively.

Experiments are also available through the *Experiments UI* in the Fusion UI.

This API requires a *Fusion AI license*.

Experiment types

The Experiments API supports A/B (or A/B/n) tests.

A/B testing

The Experiments API in Fusion lets you set up straightforward A/B (or A/B/n) tests. An A/B test is a two-sample hypothesis test with two variants. In A/B/n testing, variant A (the first variant that is defined) is compared with B, and with each of the other variants. For example, with an A/B/C/D test, the comparisons are A/B, A/C, and A/D.

The simplest setup is to create just two variants (A and B). Variant A is customarily considered the “control” configuration, and variant B is the test configuration or “treatment” to compare with the control configuration.

It is also common to perform A/A testing, that is, to create a separate variant with exactly the same configuration as the control. A/A testing is useful for detecting any systemic errors.

You can perform both A/B and A/A testing as a part of the same experiment. To do this, create three variants: one variant that is the control (A1), a second variant with the same configuration as the control (A2), and a third variant with the treatment configuration (B). The comparisons are A1/A2 and A1/B.

For example, the following experiment definition creates an experiment with 3 variants. The first two are identical, while the third uses a different query pipeline, called `with-recommendations`:

```
{ "id": "sample-experiment", "uniqueIdParameter": "userId",  
  "baseSignalsCollection": "bestbuy", "variants": [ { "name": "control-a1", }, {  
    "name": "control-a2", }, { "name": "b", "queryPipeline": "with-recommendations",  
  } ], "enabled": true, "metrics": [ { "type": "ctr", "name": "CTR", "primary":  
    true }, { "type": "conversion-rate", "name": "purchase rate", "signalType":  
    "purchase", } ] }
```

As users interact with this experiment, we expect the results of metrics for the first two variants to be quite similar, since they are configured identically. With a small sample size, metrics could vary somewhat based on random chance but, as more users interact with the experiment, we expect the metrics to converge on identical results. The third variant, however, will likely perform differently, and its performance will tell us whether we should be using the `with-recommendations` pipeline for all search traffic.

API specification

Groups API

The Groups API allows you to view and define groups of Fusion objects.

API specification

Index APIs

Indexing converts your data into an indexed collection in Fusion's Solr core. It is critical for ensuring that your data is stored in a format that is ideal for your search application.

There are several REST APIs related to indexing operations:

- *Index API*
- *Index Pipelines API*
- *Index Profiles API*
- *Index Stages API*

Index API

These endpoints can be used to send data directly to an *index profile*. For index profile CRUD operations, see the *Index Profiles API*.

Index APIs include:

- ***Index Pipelines API***. Index Pipelines are workflows to format data being ingested so it is prepared for indexing. This API manages those pipelines.
- *Index Profiles API* - Profiles match pipelines with collections.
- ***Index Stages API***. Index stages are the individual steps within index pipeline workflows. This API manages those steps.

For more information about structuring documents for indexing, see *Importing Data with the REST API*.

API specification

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 *Importing Data with the REST API*.

Examples

List the 'default' pipeline: REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/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:

In order to see this object within the *Fusion UI*, it **must** be associated with an app. To do this, create the object using the `/apps` endpoint.

REQUEST

```
curl -u USERNAME:PASSWORD -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"}]}' https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD -X PUT https://FUSION_HOST:8764/api/index-pipelines/my-index-pipeline/refresh
```

API specification

Index Profiles API

These endpoints are used to manage *index profiles*. To send a document to an index profile, see the *Index API*.

API specification

Index Stages API

API Objective: Each step of a pipeline.

The Index Stages API provides endpoints to:

- List index stage configuration properties
- Manage index stage instances
- 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.

Examples

See all defined index pipeline stages, regardless of type:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD https://FUSION_HOST:8764/api/index-
stages/instances/conn_tika
```

RESPONSE

```
{ "type" : "tika-parser", "id" : "conn_tika", "includeImages" : true,
  "flattenCompound" : false, "addFailedDocs" : true, "addOriginalContent" : true,
  "skip" : false }
```

Create an index stage:

REQUEST


```
curl -u USERNAME:PASSWORD -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"}]}' https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD -X DELETE https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD -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."}]]' https://FUSION_HOST:8764/api/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" : [ ] } ]
```

View the configuration properties for index stage type "regex-extractor":

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/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" } } } } } }
```

API specification

Jobs API

The Jobs API allows you to define schedules for objects that perform work. You can schedule datasources, tasks, and Spark jobs.

This topic explains how to format requests to the Jobs API. To learn how to work with jobs and schedules in the Fusion UI, see *Jobs and Schedules*.

Job runtime configuration

In order to see this object within the *Fusion UI*, it **must** be associated with an app. To do this, create the object using the `/apps` endpoint.

Job configuration requests have these parameters:

- `<type>:<id>` path parameter (required)
The resource identifier takes the form `<type>:<id>`, such as `datasource:movie-db`, `spark:dailyMetricsRollup-counters`, or `task:delete-old-system-logs`.
See *Job types* below.
- `enabled` body parameter
"True" or "false" to enable or disable the job.
- `triggers` body parameter
This parameter defines one or more conditions that trigger the job. See *Job triggers* below.

Below is an example request that configures a job:

```
PUT /jobs/datasource:movie-db/schedule
{
  "enabled": true,
  "triggers": [
    {"type": "cron", "enabled": true, "value": "0 */10 * * * *"}
  ]
}
```

Job types

The job type must be specified as part of the resource identifier in `PUT`, `POST`, and `DELETE` calls. These are the possible job types:

<code>datasource</code>	A job to ingest data according to the specified datasource configuration, such as <code>datasource:movie-db</code> . Datasources are created using the <i>Connector Datasources API</i> or the Fusion UI. See <i>Datasource Jobs</i> .
<code>spark</code>	A Spark job to process data, such as <code>spark:dailyMetricsRollup-counters</code> . Spark jobs are created using the <i>Spark Jobs API</i> or the Fusion UI. See <i>Spark Jobs</i> .
<code>task</code>	A job to perform an HTTP call or log cleanup, such as <code>task:delete-old-system-logs</code> . Tasks are created using the <i>Tasks API</i> or the Fusion UI. See <i>Tasks</i> .

Job triggers

Each job can have multiple triggers, and each trigger configuration requires `enabled` and `type` parameters.

Additional parameters are required by each trigger type, described below:

Trigger type	Parameters
interval Run the job at a regular interval over some time range.	startTime The date and time at which the job will first run, as an ISO-8601 date-time string (https://www.w3.org/TR/NOTE-datetime). endTime The date and time at which the job will run last, as an ISO-8601 date-time string. interval The interval at which the job will run, as an integer whose unit is specified by repeatUnit below. repeatUnit One of: minute, hour, or day.
cron Run the job on a cron-style schedule.	value A cron string.
job Run the job after another job succeeds or fails.	jobID The ID of the job that triggers this one. runOnSuccess "True" to run when the specified job succeeds. runOnFailure "True" to run when the specified job fails. runOnAbort "True" to run when the specified job is aborted.

Examples

Get all datasource jobs

Input

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/jobs?type=datasource
```

Output

```
[ {
  "resource" : "datasource:ratings-db",
  "enabled" : true,
  "startedBy" : "acb38917-611a-46e1-a3cf-4ed46a30fd24"
}, {
  "resource" : "datasource:movies-db",
  "enabled" : true,
  "startedBy" : "acb38917-611a-46e1-a3cf-4ed46a30fd24",
  "status" : "success",
  "lastStartTime" : "2017-05-17T20:58:10.503Z",
  "lastEndTime" : "2017-05-17T20:58:42.020Z"
} ]
```

Start a datasource job now

Input

```
curl -u USERNAME:PASSWORD -X POST -H "Content-Type: application/json"
https://FUSION_HOST:8764/api/jobs/datasource:movies-db/actions -d '{"action":
"start"}'
```

Output

```
{
  "resource" : "datasource:movies-db",
  "action" : "start",
  "accepted" : true,
  "message" : "Started job datasource:movies-db"
}
```

Get the job history for a datasource

Input

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/jobs/datasource:movies-
db/history
```

Output

```
[ {
  "resource" : "datasource:movies-db",
  "startTime" : "2017-05-17T21:01:46.743Z",
  "endTime" : "2017-05-17T21:02:05.720Z",
  "status" : "success"
}, {
  "resource" : "datasource:movies-db",
  "startTime" : "2017-05-17T20:58:10.503Z",
  "endTime" : "2017-05-17T20:58:42.020Z",
  "status" : "success"
} ]
```

Define a trigger for a datasource job

Input

```
curl -u USERNAME:PASSWORD -X PUT -H "Content-Type: application/json"
https://FUSION_HOST:8764/api/jobs/datasource:ratings-db/schedule -d '{"triggers":
[{"type": "cron", "enabled": "true", "expression": "0 15 10 ? * MON-FRI"}]}'
```

Output

```
{
  "resource" : "datasource:ratings-db",
  "triggers" : [ {
    "type" : "cron",
    "enabled" : true,
    "expression" : "0 15 10 ? * MON-FRI",
    "type" : "cron"
  } ]
}
```

API specification

License API

This API manages *license files*.

API specification

Links API

The Links API manages the links that represent the relationships between Fusion objects. You can see these links in the *Object Explorer* by pressing CMD-K or CTRL-K from any screen in the Fusion UI. You can also export and import links using the *Objects API*.

Links are structured as a tuple of `(subject, object, linkType)`. An example would be `(foo, bar, dependsOn)`, which would read as "foo dependsOn bar".

Link types and link reversibility

The possible values of `LinkType` are as follows:

inContextOf	The subject exists in the context of the object, which is an app. In this example, the "Movie_Search" app is the context for a parser with the same name: <pre>{ "subject" : "parser:Movie_Search", "object" : "app:Movie_Search", "linkType" : "inContextOf", "originator" : "unspecified" }</pre>
hasContext	The subject is an app that contains the object. In this example, an index profile exists in the context of the "Movie_Search" app: <pre>{ "subject" : "app:Movie_Search", "object" : "index-profile:Movie_Search", "linkType" : "hasContext", "originator" : "unspecified" }</pre>
dependsOn	The subject depends on the object. In this example, a datasource called "web_pokeapi_co-default" depends on a parser by the same name: <pre>{ "subject" : "datasource:web_pokeapi_co-default", "object" : "parser:web_pokeapi_co-default", "linkType" : "dependsOn" }</pre>
supports	The subject is depended on by the object (the reverse of <code>dependsOn</code>). In this example, a parser called "web_pokeapi_co-default" supports a datasource by the same name: <pre>{ "subject" : "parser:web_pokeapi_co-default", "object" : "datasource:web_pokeapi_co-default", "linkType" : "supports" }</pre>
isPartOf	The subject is part of the object, where the object is a group. In this example, an aggregation called "click-signals-default" is part of a group called "signals-object-group-default": <pre>{ "subject" : "aggregation:click-signals-default", "object" : "group:signals-object-group-default", "linkType" : "isPartOf" }</pre>

hasPart	The subject is a group, of which the object is a member (the reverse of <code>isPartOf</code>). In this example, an aggregation called "click-signals-default" is part of a group called "signals-object-group-default": <pre>{ "subject" : "group:signals-object-group-default", "object" : "aggregation:click-signals-default", "linkType" : "hasPart" }</pre>
relatesTo	The subject relates to the object. This is a uni-directional relationship that associates two objects that have none of the relationships described above. For example, when recommendations are enabled for the default collection, Fusion creates a group of objects which relates to the default collection: <pre>{ "subject" : "group:recommendations-default", "object" : "collection:default", "linkType" : "relatesTo" }</pre>

With the exception of the `relatesTo` link type, links are reversible:

- `inContextOf` is the reverse of `hasContext` (and vice versa).
- `dependsOn` is the reverse of `supports` (and vice versa).
- `isPartOf` is the reverse of `hasPart` (and vice versa).

For example, here is the group of links for the `signals-default` collection:

```
[ { "subject": "group:4e6ee9a6-d4ab-413a-ac70-a27c1445bb60", "object":
"collection:default_signals_aggr", "linkType": "hasPart" }, { "subject":
"group:4e6ee9a6-d4ab-413a-ac70-a27c1445bb60", "object":
"spark:default_click_signals_aggregation", "linkType": "hasPart" }, { "subject":
"group:4e6ee9a6-d4ab-413a-ac70-a27c1445bb60", "object": "collection:default",
"linkType": "relatesTo" }, { "subject": "group:4e6ee9a6-d4ab-413a-ac70-
a27c1445bb60", "object": "collection:default_signals", "linkType": "hasPart" } ]
```

When we request the links for one of the `hasPart` objects shown above, we get the reverse representation:

```
GET https://FUSION_HOST:8764/api/links?subject=collection:default_signals_aggr [
{ "subject": "collection:default_signals_aggr", "object": "group:4e6ee9a6-d4ab-
413a-ac70-a27c1445bb60", "linkType": "isPartOf" } ]
```

Examples

Get links to the default collection:

```
curl https://FUSION_HOST:8764/api/links?subject=collection:default
```

Output:

```
[ { "subject": "collection:default", "object": "index-pipeline:default",  
  "linkType": "supports" } ]
```

Create a new link:

```
curl -X PUT https://FUSION_HOST:8764/api/links -H 'content-type:  
application/json' -d '{"linkType": "supports", "object": "index-  
pipeline:default", "subject": "collection:default"}'
```

Output:

```
{ "subject" : "collection:default", "object" : "index-pipeline:default",  
  "linkType" : "supports" }
```

Delete a link:

```
curl -X DELETE https://FUSION_HOST:8764/api/links?subject=collection:default
```

API specification

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.

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 channel, whereas "to:" in the context of an email(SMTP) references the intended email address of its recipient.

Message Attribute	Description
<code>id</code>	An application specific id for tracking the message. Must be unique.
<code>type</code>	The type of message to send, one of the following: * <code>log</code> * <code>smtp</code> * <code>slack</code> * <code>pagerduty</code>
<code>to</code>	One or more destinations for the message, as a list
<code>from</code>	Who/what the message is from
<code>subject</code>	The subject of the message
<code>body</code>	The main body of the message
<code>schedule</code>	Used to pass messages at a later time, or at a recurring basis.
<code>messageServiceParams</code>	Passes a map of any service-specific parameters, such as the user name and password

Examples

Display current messaging services:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD -X POST -H 'Content-Type: application/json' -d '{"id": "myID", "type": "slack", "subject": "test", "body": "@recipient: this is a test", "to": ["recipient"], "from": "sender"}' https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD https://FUSION_HOST:8764/api/messaging/status
```

RESPONSE

```
[ { "status" : "ok", "node" : "http://xxx.xx.xx.x:8764/api/messaging", "messages" : [ ] } ]
```

API specification

Objects API

The Objects API lets you migrate objects between Fusion instances by exporting and importing. Fusion objects include all your searchable data, plus pipelines, aggregations, and other configurations on which your collections depend.

As an alternative in some cases, you can *use the Fusion UI to migrate objects*.

For both the API and UI, migrating objects is intended for use <i>within a specific Fusion version</i> . A <i>migrator tool</i> is available for migrating objects from one Fusion version to a different one.

You can select all objects, or limit the operation to specific object types or IDs. In addition to export/import endpoints, a validation endpoint is provided for troubleshooting.

By default, system-created collections are not exported. See below for details.

Object export and import

Collections and *encrypted values* are treated specially; details are provided below. During import, conflicts are resolved according to the specified *import policy*.

For objects other than collections, no implicit filtering is performed; all objects are included by default. However, on export you can *filter by type and ID*.

Supported objects

Fusion lets you export and import the object types below. To filter by type, use the values below with the `type` query string parameter; see the examples below.

Object type	Value	Output
apps	<pre>app</pre> The <code>deep</code> parameter is not needed for this object type. In the case of apps, all linked objects are included by default.	.zip file
Appkit apps	<pre>appkit</pre>	.zip file
collections (with their dependent objects) See below for collection-specific details.	<pre>collection</pre>	.zip file
index pipelines	<pre>index-pipeline</pre>	.zip file
index profiles	<pre>index-profile</pre>	.zip file
query pipelines	<pre>query-pipeline</pre>	.zip file
query profiles	<pre>query-profile</pre>	.zip file
search cluster configurations	<pre>search-cluster</pre>	.zip file
schedules	<pre>task</pre> , <pre>job</pre> , or <pre>spark</pre>	.zip file
datasources	<pre>datasource</pre>	.zip file
parsing configurations	<pre>parser</pre>	.zip file
object group	<pre>group</pre>	.zip file
link	<pre>link</pre>	.zip file
blob	<pre>blob</pre>	.zip file

Exporting and importing collections

Collections are processed with these dependent objects:

- features
- index profiles
- query profiles

Datasources, parser configurations, and pipeline configurations are not included when collections are exported or imported. These must be exported and imported explicitly.

Only user-created collections are included by default. Certain types of collections are excluded:

- the "default" collection
 - collections whose type is not DATA
 - collections whose names start with "system_"
 - "Secondary" collections, that is, collections created by features
- Instead, create the same features on the target system; this automatically creates the corresponding secondary collections.

You can override these exclusions by specifying a collection, like this:

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/objects/export?collection.ids=default > collection.zip
```

Encrypted passwords

Some objects, such as datasources and query pipelines, include encrypted passwords for accessing secure data.

- On *export*, these encrypted values are replaced with one of the following:
 - `secret.dataSources.datasourceID.password`
 - `secret.queryPipelines.pipelineID.stages.active-directory-security-trimming.stageID.bindPassword`
- On *import*, the original, plaintext passwords must be provided in a separate JSON map as multipart form data:

```
{
  "secret.dataSources.datasourceID.password":"MyDatasourcePassword",
  "secret.queryPipelines.pipelineID.stages.active-directory-security-trimming.stageID.bindPassword":"MyADPassword"
}
```

Passwords can also be uploaded via the UI. Variables that do not start with `secret.` are ignored

Import policies

On import, the `importPolicy` parameter is required. It specifies what to do if any object in the import list already exists on the target system:

<code>abort</code>	If there are conflicts, then import nothing.
<code>merge</code>	If there are conflicts, then skip the conflicting objects.
<code>overwrite</code>	If there are conflicts, then overwrite or delete/create the conflicting objects on the target system.

Filtering on export

On export, there are two ways to specify the objects to include:

- by type
To export only specific types of objects, you can specify one or more object types using the `type` query string parameter. See *Supported objects* for a list of valid values.
- by type and ID

The `type.ids` parameter lets you list the IDs to match for the specified object type.

The `type` and `type.ids` parameters can be combined as needed.

Exporting linked objects

Related Fusion objects are linked. You can view linked objects using the *Links API* or the *Object Explorer*.

When exporting a specific Fusion object, you can also export its linked objects without specifying each one individually. To export all objects linked to the specified object, include the `deep="true"` query parameter in your request. See the *example* below.

Validation

Objects are validated before import. If any objects fail validation, the whole import request is rejected. A separate endpoint is available for validating objects without importing them.

Validation includes checking whether an object already exists on the target system and whether the user is authorized to create or modify the object.

For collection objects, the following special validation is performed:

- We check the `searchClusterId` of each collection and verify that a cluster with this ID exists on the target system or in the import file (error).
- We check that features, index profiles, and query profiles belong only to the collections specified in the import file (error).
- We check that a feature exists on the target system for each feature in the import file (error).
- We check for index profiles or query profiles that do not exist on the target system or in the import file (warning).

Status messages

<code>Validation completed with no errors</code>	The validation method was called and no errors found, though there may be warnings.
<code>Validation found errors</code>	The validation was called and errors found. Validation does not stop on the first error, so the complete list of errors is reported.
<code>Validation was not completed because of system error</code>	The validation was interrupted by system error.
<code>Import was not performed because validation errors exist</code>	The import method was called, but import didn't start because of validation errors.
<code>Import was not performed because of input data error</code>	The import method was called, but import didn't start, because Fusion could not find a substitution for one of the secret values in objects in import.
<code>Import was not completed because of system error</code>	The validation found no errors and import started, but it was interrupted by system error.
<code>Import was completed</code>	Validation found no errors and import finished successfully.

Examples

Export all objects

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/objects/export > all-objects.zip
```


Export all datasources

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/objects/export?
type=datasource > datasources.zip
```

Export a specific datasource and all its linked objects

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/objects/export?
datasource.ids=movies_csv-movies > movies.zip
```

Export all apps

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/objects/export?type=app >
all-apps.zip
```

Get all app IDs, then export one app by ID

```
> curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/apps [ { "id" :
"movies", "name" : "Movies", "description" : "Search the movielens database.",
"dataUri" : "/App-Tile-01-460x160.png", "properties" : { "headerImageName" :
"headerImage1", "tileColor" : "apps-darkblue", "previousCollectionId" : "movies"
} }, { "id" : "tech-pubs", "name" : "TechPubs", "description" : "Search the
documentation.", "dataUri" : "/App-Tile-02-460x160.png", "properties" : {
"headerImageName" : "headerImage2", "tileColor" : "apps-darkblue",
"previousCollectionId" : "tech-pubs" } } ] > curl -u USERNAME:PASSWORD
https://FUSION_HOST:8764/api/objects/export?app.ids=tech-pubs > techpubs.zip
```

When you export an app that includes objects that are shared with other apps, then all apps linked to that object are also exported.

Export two apps by ID

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/objects/export?
app.ids=tech-pubs,movies > two-apps.zip
```

Export all datasources and pipelines, plus two specific parsing configurations

```
curl -u USERNAME:PASSWORD 'https://FUSION_HOST:8764/api/objects/export?
type=datasource,index-pipeline,query-pipeline&parser.ids=movies,tech-pubs' >
export.zip
```

Import objects from a file and stop if there are conflicts

```
curl -u USERNAME:PASSWORD -H "Content-Type:multipart/form-data" -X POST -F
'importData=@/Users/admin/Fusion/export.zip'
https://FUSION_HOST:8764/api/objects/import?importPolicy=abort
```

Import objects, substitute the password variables, and merge any conflicts

```
curl -u USERNAME:PASSWORD -H "Content-Type:multipart/form-data" -X POST -F
'importData=@/Users/admin/Fusion/export.zip' -F
'variableValues=@password_file.json' https://FUSION_HOST:8764/api/objects/import?
importPolicy=merge
```

☐ `password_file.json` must contain plaintext passwords.

Import a zip file of Fusion objects and merge any conflicts

```
curl -u USERNAME:PASSWORD -H "Content-Type:multipart/form-data" -X POST -F
'importData=@/Users/admin/Fusion/techpubs.zip'
https://FUSION_HOST:8764/api/objects/import?importPolicy=merge
```

Import objects into the context of an app

```
curl -v -H 'Content-Type:multipart/form-data' -X POST -F
'importData=@app_part.zip' 'https://FUSION_HOST:8765/api/v1/objects/import?
importPolicy=overwrite&context=MyApp'
```

API specification

Parsers API

The Parser API works much like the *Index Pipelines API* and the *Query Pipelines API*, providing CRUD operations over parser configurations.

To configure an index pipeline to use a specific parser, see the *Index Pipelines API*.

In order to see this object within the <i>Fusion UI</i> , it must be associated with an app. To do this, create the object using the <code>/apps</code> endpoint.

For more information about structuring documents for indexing, see *Importing Data with the REST API*.

API specification

Query APIs

Query operations allow you to retrieve data from your index. There are several REST APIs for query operations:

- *Query API*
- *Query Pipelines API*
- *Query Profiles API*
- *Query Rewrite API*
- *Query Stages API*

Query API

You can run a query through a *query profile* by specifying the profile ID and appending the request with a *Solr query string*, as in `/api/query/<id>?<solrQuery>`.

For query profile CRUD operations, see the *Query Profiles API*.

API specification

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.

Query Pipeline Definition Properties

Property	Description
id <i>Required</i>	Required only when creating a new pipeline.
stages <i>Required</i>	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 <i>Query Pipeline Stages</i>, then those can be defined while defining the stage within the pipeline.

Examples

Get the definition for the default query pipeline:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/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:

In order to see this object within the *Fusion UI*, it **must** be associated with an app. To do this, create the object using the `/apps` endpoint.

REQUEST

```
curl -u USERNAME:PASSWORD -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}]}' https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD https://FUSION_HOST:8764/api/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" } } }
```

API specification

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 more information, see *Query Profiles*.

To run a query through a query profile, use the *Query API*.

API specification

Query Stages API

The Query Stages API provides endpoints to:

- List query stage configuration properties
- Manage query stage instances
- Test processing on a set of queries

Examples

See all defined query pipeline stages, regardless of type:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d'{ "type": "query-logging", "id": "detailed-logging", "detailed": true }' https://FUSION_HOST:8764/api/query-stages/instances
```

RESPONSE

```
{ "type": "query-logging", "id": "detailed-logging", "detailed": true, "skip": false, "label": "query-logging" }
```

Update a query stage:

REQUEST

```
curl -u USERNAME:PASSWORD -X PUT -H 'Content-type: application/json' -d'{ "type": "query-logging", "id": "detailed-logging", "detailed": true, "label": "detailed-query-logging" }' https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD -X DELETE https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d '{"type":"set-params", "params":[{"key":"rows", "value":"2", "policy":"append"}]}' https://FUSION_HOST:8764/api/query-stages/solr/test?q=*:*
```

RESPONSE

```
{ "request" : { "headers" : { "User-Agent" : [ "curl/7.30.0" ], "Content-Type" : [ "application/json" ], "Accept" : [ "/*/*" ], "Host" : [ "fusion-host:{api-port}" ], "Content-Length" : [ "80" ] }, "params" : { "q" : [ "*:*" ], "rows" : [ "2" ] } }, "response" : null, "totalTime" : 0 }
```

Get the properties for the "apply-defaults" type:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/query-stages/schema/set-params
```

RESPONSE


```
{ "type" : "object", "title" : "Additional Query Parameters", "description" :
"This stage allows you to set, append, and remove additional query parameters",
"properties" : { "skip" : { "type" : "boolean", "title" : "Skip This Stage",
"description" : "Set to true to skip this stage.", "default" : false, "hints" : [
"advanced" ] }, "label" : { "type" : "string", "title" : "Label", "description" :
"A unique label for this stage.", "hints" : [ "advanced" ], "maxLength" : 255 },
"condition" : { "type" : "string", "title" : "Condition", "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.", "hints" : [ "code", "javascript",
"advanced" ] }, "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" ], "default" :
"append" } } } } }, "category" : "Advanced", "categoryPriority" : 2
```

API specification

Recommendations API

The Recommendations REST API uses signals and aggregated signals to return a list of items that can be used for recommendations.

This API requires a *Fusion AI* license.

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 and Boosting* for a discussion of when to use the API and when to use recommender query stages.

Recommendation types

The API includes separate endpoints for retrieving different types of recommendations:

<code>itemsForQuery</code>	Get items for a defined query. The response is a list of <i>document IDs</i> and their weights.
<code>queriesForItem</code>	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 <i>query terms</i> and their weights.

Output

The output includes the following sections:

<code>header</code>	The query parameters (in a section named <code>queryParams</code>) and the total time it took to process the query.
<code>items</code>	Depending on the recommendation type: <code>itemsForQuery</code> The document IDs and the weights of aggregated events that match the query. This type supports a <code>debug</code> option that adds a <code>debug</code> section to the output. <code>queriesForItem</code> The queries and the weights of aggregated events that match the document ID.

Examples

Below are examples for each recommendation type.

itemsForQuery

Get the top items for the query 'ipod':

INPUT

```
curl -u USERNAME:PASSWORD
https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD
https://FUSION_HOST:8764/api/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 } ] }
```

API specification

Scheduler API

The scheduler REST API is used to define a schedule for system activities and manage the jobs that result from the schedule.

This API is deprecated in favor of the *Jobs API*.

All of the Fusion services are available for scheduling, as are any Solr activities and any other HTTP-based URI.

See *Jobs and Schedules* for more information about all of the options available for scheduling, and instructions for configuring scheduler jobs using the Fusion UI.

Schedule Definition Properties

Parameter	Description
id <i>Required</i>	The ID of the schedule. This is required when creating a schedule with a POST request.
creatorType <i>Optional</i>	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 <i>Optional</i>	An ID for the user that created the schedule.
startTime <i>Optional</i>	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 <i>Optional</i>	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 <i>Optional</i>	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 <i>Optional</i>	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 <i>Optional</i>	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: ◦ <code>http://</code> or <code>https://</code> ◦ <code>solr://{collection}/...</code> A SolrCloud request. ◦ <code>service://{serviceName}/{path}</code> 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.

Examples

Run a crawl of the datasource "LocalDocs2" every day:

REQUEST

```
curl -u USERNAME:PASSWORD -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"}}'
https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD -X POST -H 'Content-Type: application/json' -d
'{"id":"2", "creatorType":"human", "creatorId":"admin1", "repeatUnit":"HOURL",
"interval":1, "active":true, "callParams":{"uri":"solr://demo/update",
"method":"GET", "queryParams":{"stream.body":"<commit/>"}}}'
https://FUSION_HOST:8764/api/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" : "HOURL",
    "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 USERNAME:PASSWORD -X PUT -H 'Content-Type: application/json' -d
'{"creatorType":"human", "creatorId":"admin1", "repeatUnit":"HOURL", "interval":1,
"active":false, "callParams":{"uri":"solr://demo/update", "method":"GET",
"queryParams":{"stream.body":"<commit/>"}}}'
https://FUSION_HOST:8764/api/scheduler/schedules/2
```

There will be no response if the PUT request was successful.

List all scheduled jobs:

REQUEST

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/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" : "HOURL",
    "interval" : 1,
    "active" : true,
    "callParams" : {
      "uri" : "solr://demo/update",
      "method" : "GET",
      "queryParams" : {
        "stream.body" : "<commit/>"
      },
      "headers" : { },
      "entity" : null
    },
    "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
    },
    "headers" : { },
    "entity" : null
  }
} ]
```

List the history of job ID '1':

REQUEST


```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/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}"
    }
  ]
}
```

API specification

Search Cluster API

The search cluster API allows users to connect Fusion with any existing Solr instances in a Zookeeper-managed cluster.

Cluster operations are only supported when connecting through Zookeeper.

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.

Background

Solr has three different approaches on how you can control visibility of new documents in search:

- You can `commit` manually
- You can rely on Solr's `autoCommit` setting
- You can specify `commitWithin` when adding documents

Fusion uses `commitWithin` to avoid relying on specific Solr side configurations. Fusion controls `commitWithin` on a per-collection basis so you can have multiple collections with different commit frequencies (for example, product documents can be committed more often than signals).

Global setting for `commitWithin`:

```
curl
http://localhost:8765/api/v1/configurations/com.lucidworks.apollo.solr.commitWith
in 10000
```

`com.lucidworks.apollo.solr.commitWithin` is a global configuration property that defines default `commitWithin` for all documents added through Fusion. Every time you create a new collection in Fusion, per-collection `commitWithin` is initialized as the global default.

Per-collection setting: You can either specify this property when creating collection or update it with `PUT` later.

```
# create collection without specifying commitWithin sh> curl -H 'Content-type:
application/json' -X POST 'http://localhost:8765/api/v1/collections' -d '{"id" :
"test"}' { "id" : "test", ... "commitWithin" : 10000, ... } # create collection
and specify non default value sh> curl -H 'Content-type: application/json' -X
POST 'http://localhost:8765/api/v1/collections' -d '{"id" : "test2",
"commitWithin": 20000}' { "id" : "test2", ... "commitWithin" : 20000, ... } #
update commitWithin at a runtime sh> curl -H 'Content-type: application/json' -X
PUT 'http://localhost:8765/api/v1/collections/test' -d ' { "id" : "test",
"createdAt" : "2015-01-07T17:44:47.396Z", "searchClusterId" : "default",
"commitWithin" : 20000, "solrParams" : { "name" : "test", "numShards" : 1,
"replicationFactor" : 1 }, "type" : "DATA", "metadata" : { } }'
```

Search Cluster Definition Properties

Property	Description
id <i>Required</i>	The ID of the search cluster. This is only required when creating a new cluster definition with a POST request.
connectString <i>Required</i>	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 <i>Required</i>	Defines if the "cluster" being defined is a SolrCloud cluster (true) or a standalone Solr instance (false).
bufferFlushInterval <i>Optional</i>	Defines how often to flush the buffer when writing to this cluster. If not defined, the system will default to 1000 milliseconds.
bufferSize <i>Optional</i>	Defines the size of the buffer. If not defined, the system will default to 100 items in the buffer.
concurrency <i>Optional</i>	Defines the maximum number of concurrent /parallel requests to Solr servers when Fusion index pipeline <i>Solr Indexer stage</i> has property <i>bufferDocsForSolr</i> set to true.
zkClientTimeout <i>Optional</i>	The maximum amount of time to wait when communicating with the ZooKeeper ensemble for a SolrCloud instance.
zkConnectTimeout <i>Optional</i>	The maximum amount of time to wait when trying to connect to the ZooKeeper ensemble for a SolrCloud instance.

Examples

Create a new search cluster that is an existing SolrCloud cluster:

REQUEST

```
curl -u USERNAME:PASSWORD -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
}' https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD -X POST -H 'Content-type: application/json' -d
'{"id":"myOtherSolrCluster", "connectString":"https://FUSION_HOST:8983/solr",
"cloud":false}' https://FUSION_HOST:8764/api/searchCluster
```

RESPONSE

```
{
  "id" : "myOtherSolrCluster",
  "connectString" : "https://FUSION_HOST:8983/solr",
  "cloud" : false,
}
```

Show the status of each node of 'mySolrCluster':

REQUEST

```
curl https://FUSION_HOST:8764/api/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/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"
    }
  }
}
```

API specification

Signals API

The Signals API accepts a set of *signals*, encoded as JSON objects, for indexing into a signals collection.

This API requires a *Fusion AI license*.

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 signals, use a *SQL aggregation* job, which is a kind of *Spark job*.

You can manage aggregation jobs in the *Jobs manager* and *Scheduler* in the Admin UI, or with the *Spark jobs API*. We recommend using the Admin UI.

See *Signal types and structure* to learn how to send well-formed signals to this API.

Examples

Send two signal events to record user clicks:

REQUEST

```
curl \ -u USERNAME:PASSWORD -X POST -H 'Content-type:application/json' -d @- \
https://FUSION_HOST:8764/api/signals/docs?commit=true \ <<EOF [ {"params": {
"query": "Televisi\u00f3nes Panasonic 50 pulgadas", "filterQueries":
["cat00000","abcat0100000", "abcat0101000", "abcat0101001"], "docId": "2125233"
}, "type":"click", "timestamp": "2011-09-01T23:44:52.53Z" }, {"params": {
"query": "Sharp", "filterQueries": ["cat00000", "abcat0100000", "abcat0101000",
"abcat0101001"], "docId": "2009324" }, "type":"click", "timestamp": "2011-09-
05T12:25:37.42Z" } ] 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":"docId: 2125233", "wt":"json"} }, "response":
{"numFound":1198,"start":0,"docs":[ { "id": "7aee7b1f-5cde-4957-b73c-
c15881f559ec", "filters_s": "abcat0100000 $ abcat0101000 $ abcat0101001 $
cat00000", "query_orig_s": "Televisi\u00f3nes Panasonic 50 pulgadas", "params.user_s":
"000000df17cd56a5df4a94074e133c9d4739fae3", "docId": "2125233",
"params.query_time_s": "2011-09-01T23:43:59.75Z", "query_t": "televisi\u00f3nes
panasonic 50 pulgadas", "query_s": "televisi\u00f3nes 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":
"2011-09-01T23:44:52.53Z", "_version_": 1478892846857584600 }, { "id": "6789a209-
f5b5-457e-9df6-8033b8f7f317", "filters_s": "abcat0100000 $ abcat0101000 $
abcat0101001 $ cat00000", "query_orig_s": "Sharp", "params.user_s":
"0000001928162247ffaf63185cd8b2a244c78e7c6", "docId": "2009324",
"params.query_time_s": "2011-09-05T12:25:01.18Z", "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 } ] } }
```

API specification

Solr API

The Solr API is used to manage collection-level configurations.

The path for this request is:

```
/api/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.

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

```
https://FUSION_HOST:8764/api/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" ] } ] } } -----
```

API specification

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 Editor API*.

To manage Solr lists of stopwords, see *Stopwords Files*.

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.

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 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.

Examples

Show `_rest_managed.json` for the 'docs' collection, with the base64 encoded content:

INPUT

```
curl -u USERNAME:PASSWORD https://FUSION_HOST:8764/api/collections/docs/solr-config/_rest_managed.json?expand=true
```

OUTPUT

```
{
  "name" : "_rest_managed.json",
  "version" : 0,
  "isDir" : false,
  "value" : "eyJpbm10QXJncyI6e30sIm1hbmFnZWRMaXN0IjpbXX0NCg=="
}
```

Get the plain text version of 'synonyms.txt':

INPUT

```
curl -u USERNAME:PASSWORD -H 'Accept: text/plain'
https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD -X PUT -H 'Content-type: text/plain' --data-binary '@updated-synonyms.txt' https://FUSION_HOST:8764/api/collections/docs/solr-config/synonyms.txt?reload=true
```

OUTPUT

None.

Change Solr configurations in Fusion 5.x using the API

Add a search component and a request handler:

```
curl -u USERNAME:PASSWORD -H "Accept: application/octet-stream"  
https://{host}/api/collections/collectionName/solr-config/solrconfig.xml
```

API specification

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 is not recommended to use this with a collection-level command (such as a query, or document update).

Example

Here is an example to GET all CORE MBeans (<https://solr.apache.org/guide/solr/latest/deployment-guide/mbean-request-handler.html>) for the 'default' searchCluster, formatted in JSON:

REQUEST

```
https://FUSION_HOST:8764/api/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 } } ] }
```

API specification

Spark Jobs API

This is a set of endpoints for configuring and running Spark jobs.

Spark job subtypes

For the Spark *job type*, the available subtypes are listed below.

- *Aggregation*
Define an aggregation job.
- *Script*
Run a custom Scala script as a Fusion job.

Additional Spark jobs are available with a <i>Fusion AI</i> license.

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/configurations` which will update the stored value without restarting the service, therefore existing jobs and SparkContexts will not be affected. The Fusion endpoint `api/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 that checks a local Fusion installation running on port 8764:

```
curl -u username:password https://FUSION_HOST:8764/api/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 also supports dynamic allocation for all applications. This can be overridden per application. In practice, this means that even when there is 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
<code>spark.master.url</code>	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 is not set.
<code>spark.cores.max</code>	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.
<code>spark.executor.memory</code>	Amount of memory assigned to each application's executor. The default is 2G.
<code>spark.scheduler.mode</code>	Controls how tasks are assigned to available resources. Can be either 'FIFO' or 'FAIR'. Default value is 'FAIR'.
<code>spark.dynamicAllocation.enabled</code>	Boolean - whether or not to enable dynamic allocation of executors. Default value is 'TRUE'.
<code>spark.shuffle.service.enabled</code>	Boolean - whether or not to enable internal shuffle service for standalone Spark cluster. Default value is 'TRUE'.
<code>spark.dynamicAllocation.executorIdleTimeout</code>	Number of seconds after which idle executors are removed. Default value is '60s'.
<code>spark.dynamicAllocation.minExecutors</code>	Number of executors to leave running even when idle. Default value is 0.
<code>spark.eventLog.enabled</code>	Boolean - whether or not event log is enabled. Event log stores job details and can be accessed after application finishes. Default value is 'TRUE'.
<code>spark.eventLog.dir</code>	Directory that stores event logs. Default location is <code>\$FUSION_HOME/var/spark-eventlog</code> .
<code>spark.eventLog.compress</code>	Boolean - whether or not to compress event log data. Default value is 'TRUE'.
<code>spark.logConf</code>	Boolean - whether or not to log effective SparkConf of new SparkContext-s. Default value is 'TRUE'.
<code>spark.deploy.recoveryMode</code>	Default value is 'ZOOKEEPER'
<code>spark.deploy.zookeeper.url</code>	ZooKeeper connect string. Default value is <code>\$FUSION_ZK</code>
<code>spark.deploy.zookeeper.dir</code>	ZooKeeper path, default value is <code>/lucid/spark</code>
<code>spark.worker.cleanup.enabled</code>	Boolean - whether or not to periodically cleanup worker data. Default value is 'TRUE'.
<code>spark.worker.cleanup.appDataTtl</code>	Time-to-live in seconds. Default value is 86400 (24h).
<code>spark.deploy.retainedApplications</code>	The maximum number of applications to show in the UI. Default value is 50.

Property	Description
<code>spark.deploy.retainedDrivers</code>	The maximum number of drivers. Default value is 50.
<code>spark.worker.timeout</code>	The maximum timeout in seconds allowed before a worker is considered lost. The default value is 30.
<code>spark.worker.memory</code>	The maximum total heap allocated to all executors running on this worker. Defaults to value of the executor memory heap.

Fusion Configuration Properties

Property	Description
<code>fusion.spark.master.port</code>	Spark master job submission port. Default value is 8766.
<code>fusion.spark.master.ui.port</code>	Spark master web UI port. Default value is 8767.
<code>fusion.spark.idleTime</code>	Maximum idle time in seconds, after which the application (ie. SparkContext) is shut down. Default value is 300.
<code>fusion.spark.executor.memory.min</code>	Minimum executor memory in MB. Default value 450Mb, which is sufficient to let Fusion components in application task's to initialize themselves
<code>fusion.spark.executor.memory.fraction</code>	A float number in range <code>(0.0, 1.0]</code> indicating what portion of <code>spark.executor.memory</code> to allocate to this application. Default value is 1.0.
<code>fusion.spark.cores.fraction</code>	A float number in range <code>(0.0, 1.0]</code> indicating what portion of <code>spark.cores.max</code> to allocate to this application. Default value is 1.0.

API specification

To avoid losing your running history, Lucidworks recommends that you start jobs with the <i>Fusion Jobs API</i> rather than the Job-Config-Controller or Job-Controller .

Synonyms Editor API

API Objective: Update a synonyms file.

See *Synonyms Files* for an overview and examples of how to specify a set of synonyms.

Examples

Managing synonym file states

*List all available synonyms files along with its **READY** / **NOT_READY** statuses*

```
curl '/collections/logs/synonyms'  
[ {  
  "path" : "synonyms.txt",  
  "state" : "NOT_READY"  
} ]
```

*Changing the synonym file's state to **READY** opens all the synonyms browsing/editing APIs*

```
`curl -XPUT -H 'Content-type: application/json'  
'/collections/logs/synonyms/synonyms.txt' -d '{"state":"READY"}'`
```

*Check editing **READY** / **NOT_READY** status for a single synonym file*

```
curl '/collections/logs/synonyms/synonyms.txt'  
{  
  "path" : "synonyms.txt",  
  "state" : "READY"  
}
```

Managing synonym file entries

List all entries in a synonyms file

```
curl '/collections/logs/synonyms/synonyms.txt/items'
{
  "count" : 8,
  "items" : [ {
    "id" : "37488777-a668-49d8-a3fa-bc81a95a8d96",
    "mapping" : "aaafoo => aaabar",
    "enabled" : true,
    "type" : "EXPLICIT"
  }
  ...
  }, {
    "id" : "d68f2a35-93d3-41a0-8b10-4768bd4619e8",
    "mapping" : "MB,mib,megabyte,megabytes",
    "enabled" : true,
    "type" : "EQUIVALENT"
  }
  ...
}
```

Add new entry to a synonyms file

POST will return items with all of its data, including the ID and modified timestamp. Also note, the **type** property is read-only and is returned based on **=>** and not **,** mapping.

```
curl -XPOST -H 'Content-type: application/json'
'/collections/logs/synonyms/synonyms.txt/items' -d '
{
  "mapping": "term1, term2",
  "category": "cat1",
  "enabled": "false"
}'

{
  "id" : "01cd53f7-1941-499e-ba5e-fb35ec6bd20c",
  "mapping" : "term1, term2",
  "category" : "cat1",
  "modified" : "2015-11-03T01:30:21.530Z",
  "enabled" : false,
  "type" : "EQUIVALENT"
}
```


Update an entry in a synonyms file

```
curl -XPUT -H 'Content-type: application/json'
'/collections/logs/synonyms/synonyms.txt/items/01cd53f7-1941-499e-ba5e-
fb35ec6bd20c' -d '{
  "mapping": "term1, term2, term3",
  "category" : "cat1",
  "enabled": "true"
}'
```

Delete an entry in a synonyms file

```
curl -XDELETE '/collections/logs/synonyms/synonyms.txt/items/58f94a0e-473d-4719-
b87c-d9908c84c89b'
```

Search, filter, or sort synonyms

Fusion supports Solr query language, so you can use it to refine your search in the following ways:

- Show only explicit mappings with `q=type:EXPLICIT`
- Show only disabled mappings with `q=enabled:false`
- Sort using `sort=category asc`, `sort=modified desc`, or `sort=type asc`
- Filter by user (`q=user:<username>`) and modified date (`modified: [NOW-1DAY TO *]`)

```
curl '/collections/logs/synonyms/synonyms.txt/items?q=category:cat1'
{
  "count" : 1,
  "items" : [ {
    "id" : "01cd53f7-1941-499e-ba5e-fb35ec6bd20c",
    "mapping" : "term1, term2, term3",
    "category" : "cat1",
    "modified" : "2015-11-03T01:33:58.132Z",
    "enabled" : true,
    "type" : "EQUIVALENT"
  } ]
}
```

Get term suggestions from synonyms or categories for autocomplete/autosuggest purposes

- Get suggestions for categories

```
curl '/collections/logs/synonyms/synonyms.txt/suggestions?field=category&q=ca'
[ "cat1" ]
```

- Get suggestions for terms

```
curl '/collections/logs/synonyms/synonyms.txt/suggestions?field=mapping&q=te'
[ "term1", "term2", "television", "televisions", "term3" ]
```

- Get suggestions for both terms and categories

```
curl '/collections/logs/synonyms/synonyms.txt/suggestions?q=c'
[ "cat1", "cccbar", "cccbaz", "cccfoo" ]
```

Committing and flushing

Determine whether a commit or flush is required

```
curl '/collections/logs/synonyms/synonyms.txt'
{
  "path" : "synonyms.txt",
  "state" : "NOT_SAVED"
}
```

Flush or commit changes

```
curl -XPUT -H 'Content-type: application/json'
'/collections/logs/synonyms/synonyms.txt' -d '{"state":"READY"}
```

Importing and exporting

Export synonyms into a csv file

```
curl -v -H 'Accept: text/csv' '/collections/test/synonyms/synonyms.txt/items'
...
< HTTP/1.1 200 OK
< Set-Cookie: id=81e42c0c-bb71-43a4-ae79-39a832f2f3ab;HttpOnly;Path=/api
< Content-Type: text/csv
...
id,mapping,category,enabled,modified,user,type
"f52a08c5-6a32-4ff5-8603-2540ef9d7899","aaafoo => aaabar",,true,,,EXPLICIT
"da061429-37a0-439a-8117-2e2691d4de1a","bbbfoo => bbbfoo bbbbar",,true,,,EXPLICIT
"d1260556-eb2d-4fb2-83b0-f124b82a3c88","cccfoo => cccbar cccbaz",,true,,,EXPLICIT
"4c6955d7-1f1c-4b7a-92a2-de46fad310d0","fooaaa,baraaa,bazaaa",,true,,,EQUIVALENT
"a61a74db-41a0-4103-8738-111ae507bfdd","GB,gib,gigabyte,gigabytes",,true,,,EQUIVALENT
"e54f3477-7838-4b09-b048-415e1baeeb30","MB,mib,megabyte,megabytes",,true,,,EQUIVALENT
"ebf9e483-80c5-4bcf-b4f2-f0d59c14ef87","Television, Televisions, TV, TVs",,true,,,EQUIVALENT
"e9e32949-751d-4461-897f-3be04be3575b","pixima => pixma",,true,,,EXPLICIT
```

Import synonyms from csv file

```
curl -v '/collections/test/synonyms/synonyms.txt/items' -H 'Content-type: text/csv' -X POST -d 'id,mapping,category,enabled,modified,user,type
"f52a08c5-6a32-4ff5-8603-2540ef9d7899","aaafoo => aaabar",,true,,,EXPLICIT
'
...
> Content-type: text/csv
...
< HTTP/1.1 204 No Content
```

API specification

Tasks API

The Tasks API allows you to view and define Fusion *tasks*, which can be run using the *Scheduler*.

API specification

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".

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.

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 is not overtly specified here as it is managed by Fusion.

API specification

Webapps API

The Webapps API manipulates app objects managed by *Fusion App Studio*.

API specification

Webapps Admin API

The Webapps Admin API manipulates the webapps that ship with Fusion, such as the Fusion UI, App Insights, the DevOps Center, and so on.

API specification

ZooKeeper Export API

The ZooKeeper 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.

The service may be used to export ZooKeeper data for any Fusion release.

- 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 Objects by Hand*.

The REST API only supports requests to export ZooKeeper configurations. The Fusion distribution includes a utility script `zkImportExport.sh` which can be used to export ZooKeeper configuration from arbitrary Fusion instances.

ZooKeeper

Apache ZooKeeper (<http://zookeeper.apache.org/>) 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 hierarchical 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 (<http://zookeeper.apache.org/doc/current/>) for details about using and managing a ZooKeeper service.

Utility script `zkImportExport.sh`

This script is located in the top-level Fusion `scripts` directory. The script takes the following command-line arguments:

<code>-c,--cmd <arg></code>	Command, one of: 'export', 'import', 'update', 'delete'.
<code>-e,--encode <arg></code>	Type of encoding for znodes. Valid options: 'none', 'utf-8', 'base64', default is 'base64'. Option 'none' will not return any data from the znodes.
<code>-ep,--exclude <arg></code>	Exclude znode paths, followed by list of paths. Can only be used to exclude nodes one level below the root node.
<code>-eph,--ephemeral</code> default false.	Include ephemeral nodes while exporting znodes, boolean,
<code>-f,--filename <arg></code>	Name of file containing import/export data.
<code>-h,--help</code>	Display help page.
<code>-ip,--include <arg></code> paths.	Include znode paths to include, followed by a list of paths. Can only be used to include nodes one level below the root node.
<code>-nr,--non-recursive</code>	Do not perform recursive operations on znodes.
<code>-o,--overwrite</code> 'update' command.	Overwrite data for existing znodes. Valid only with
<code>-p,--path <arg></code> pipelines'.	Path from ZooKeeper root node, e.g. '/lucid/query-
<code>-r,--recursive</code>	Perform recursive operations on znodes.
<code>-z,--zkhost <arg></code>	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 fusion-host: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 fusion-host: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 fusion-host: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 fusion-host: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 fusion-host:9983 -cmd update -filename  
znode_lucid_dump.json
```

Remove a znode from Fusion's ZooKeeper service:

```
zkImportExport.sh -zkhost fusion-host:9983 -cmd delete -path /lucid/test
```

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 USERNAME:PASSWORD -X GET
https://FUSION_HOST:8764/api/zk/export/lucid/query-pipelines { "path" :
"/lucid/query-pipelines", "children" : [ { "path" : "/lucid/query-
pipelines/default", "data" :
"ewogICJpZCIq0iAiZGVmYXVsdCIsCiAgInN0YXVldcyIq0iBbIHsKICAgICJ0eXBliIiA6ICJzZWZyY2gt
ZmllbGRzIiwKICAgICJpZCIq0iAiMzc1NmI1ZDctY2MwMCM0MDAyLWJiOWQtNTQzNjQ4NzVjMjgyIiwKI
CAGICJyb3dzIiA6IDewLAogICAgInN0YXJ0IiA6IDAsCiAgICAic2tpcCIg0iBmYWxzZSwKICAgICJsYW
JlbCIg0iAic2VhcmNoLWZpZWxkcyIsCiAgICAidHlwZSIg0iAic2VhcmNoLWZpZWxkcyIKICB9LCB7CiA
gICAidHlwZSIg0iAiZmFjZXQiLAogICAgImkIiA6ICI3MTFmZWMM1Ni03MzRmLTRiYTAtOWY1NS0wZTQ4
YmU2NTllM2UiLAogICAgInNraXAiIDogZmFsc2UsCiAgICAibGFzZwWiIDogImZhY2V0IiwKICAgICJ0e
XBliIiA6ICJmYWNldCIKICB9LCB7CiAgICAidHlwZSIg0iAic29sci1xdWVyeSIsCiAgICAiaWQiIDogIj
YwNDU1YjU2LTdkN2MtNDZjYi1iN2QzLTlxOWU1N2U3MWNjMyIsCiAgICAiaHR0cE1ldGhvZCIg0iAiUE9
TVCIIsCiAgICAic2tpcCIg0iBmYWxzZSwKICAgICJsYWJlbCIg0iAic29sci1xdWVyeSIsCiAgICAidHlw
ZSIg0iAic29sci1xdWVyeSIKICB9IF0KfQ==" } ], "data" : "" }

```

Get info for node `/lucid/query-pipelines`. Do not expand the `znodes`

```

curl -u USERNAME:PASSWORD -X GET
https://FUSION_HOST:8764/api/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 USERNAME:PASSWORD -X GET
https://FUSION_HOST:8764/api/zk/export/lucid/query-pipelines?
recursive=true&encodeValues=utf-8 { "path" : "/lucid/query-pipelines", "children"
: [ { "path" : "/lucid/query-pipelines/default", "data" : "{\n \"id\" :
\n\"default\", \n \"stages\" : [ {\n \"type\" : \"search-fields\", \n \"id\" :
\n\"3756b5d7-cc00-4002-bb9d-54364875c282\", \n \"rows\" : 10, \n \"start\" : 0, \n
\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\" :
\n\"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 USERNAME:PASSWORD -X GET
https://FUSION_HOST:8764/api/zk/export/lucid/query-pipelines?
recursive=true&encodeValues=utf-8&excludePaths=/lucid/query-pipelines/default {
"path" : "/lucid/query-pipelines", "data" : "" }
```

API specification

Configure Fusion Logging in Fusion 4.0

Fusion uses the Apache Log4j 2 (<http://logging.apache.org/log4j/2.x/manual/index.html>) logging framework with Jetty to log each of the Fusion components.

Logging is configured with an XML configuration file named `log4j2.xml`. Log levels, frequencies, and log rotation policy can be configured by changing these configuration files:

API service	<code>https://FUSION_HOST:FUSION_PORT/conf/api-log4j2.xml</code>
Connectors	<code>https://FUSION_HOST:FUSION_PORT/conf/connectors-classic-log4j2.xml</code> <code>https://FUSION_HOST:FUSION_PORT/conf/connectors-rpc-agent-log4j2.xml</code> <code>https://FUSION_HOST:FUSION_PORT/conf/connectors-rpc-log4j2.xml</code>
Solr	<code>https://FUSION_HOST:FUSION_PORT/conf/solr-log4j2.xml</code>
Spark	<code>https://FUSION_HOST:FUSION_PORT/conf/spark-driver-launcher-log4j2.xml</code> <code>https://FUSION_HOST:FUSION_PORT/conf/spark-driver-log4j2.xml</code> <code>https://FUSION_HOST:FUSION_PORT/conf/spark-driver-scripted-log4j2.xml</code> <code>https://FUSION_HOST:FUSION_PORT/conf/spark-master-agent-log4j2.xml</code> <code>https://FUSION_HOST:FUSION_PORT/conf/spark-master-log4j2.xml</code> <code>https://FUSION_HOST:FUSION_PORT/conf/spark-worker-agent-log4j2.xml</code> <code>https://FUSION_HOST:FUSION_PORT/conf/spark-worker-log4j2.xml</code>
SQL	<code>https://FUSION_HOST:FUSION_PORT/conf/sql-agent-log4j2.xml</code> <code>https://FUSION_HOST:FUSION_PORT/conf/sql-log4j2.xml</code>
UI	<code>https://FUSION_HOST:FUSION_PORT/conf/admin-ui-log4j2.xml</code>
ZooKeeper	<code>https://FUSION_HOST:FUSION_PORT/conf/zk-log4j2.xml</code>
Other Fusion services	<code>https://FUSION_HOST:FUSION_PORT/conf/proxy-log4j2.xml</code>

The Log4j2 Configuration (<http://logging.apache.org/log4j/2.x/manual/configuration.html>) guide provides documentation and examples of all logging configuration options.

Query Operations

This section includes reference information for the following topics:

- **Custom JavaScript Stages Variables and Parameters** - 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. This article describes the variables and parameters used with *custom JavaScript stages*.
- **Solr Query Language Cheat Sheet** - This cheat sheet is a quick reference to the Solr query language. Use this syntax when querying Fusion via the *Query API*.

Configure Fusion for JWT

You can configure Fusion Server to use JSON Web Tokens (JWTs) for user authentication and authorization. Fusion can also use a shared secret key between the issuer and Fusion to encrypt the JWT payload.

Fusion 4.1.0 and 4.1.1 have known issues with JWT signature validations. If you are using either of these versions and want to utilize JWT, consider *upgrading to the latest version* of Fusion.

How the JWT realm works

A JWT is comprised of three distinct parts: the header, payload, and signing key. Each of these parts are separately encoded using Base64url encoding.

- The **header** identifies the algorithm used to generate the token.
Fusion uses both the HS256 and RS256 signature algorithms.
- The **payload** consists of data that will be passed with the token.
- The **signing key** validates the integrity of the token by using a "secret" to ensure the header and payload being submitted in the token match the header and payload stored in the signing key.

Within Fusion, the JWT realm uses an *authorization header* in the request to authenticate the user and the data inside the JWT token for the authorization. This authorization header uses the following format:

```
Bearer <jwt-token>
```

Upon receiving the authorization header, Fusion authenticates the token and emits a response accordingly.

Configure JWT for Fusion

Create a JWT token

Using the tool you use to validate users, create a JWT token. The token requires the following properties:

- **iss**: Issuer value. If the issuer value does not match the value configured in Fusion, the user will be denied access.
- **iat**: A JSON numeric date value. This value is calculated by counting the number of seconds between 1970-01-01T00:00:00Z UTC and the specified UTC date/time, ignoring leap seconds.
- **sub**: Subject. The name/id of the user. The user is logged in by this name.
- **groups**: The groups from the group-role mappings that this user belongs to. The groups key should match the one you specify while creating the JWT realm.

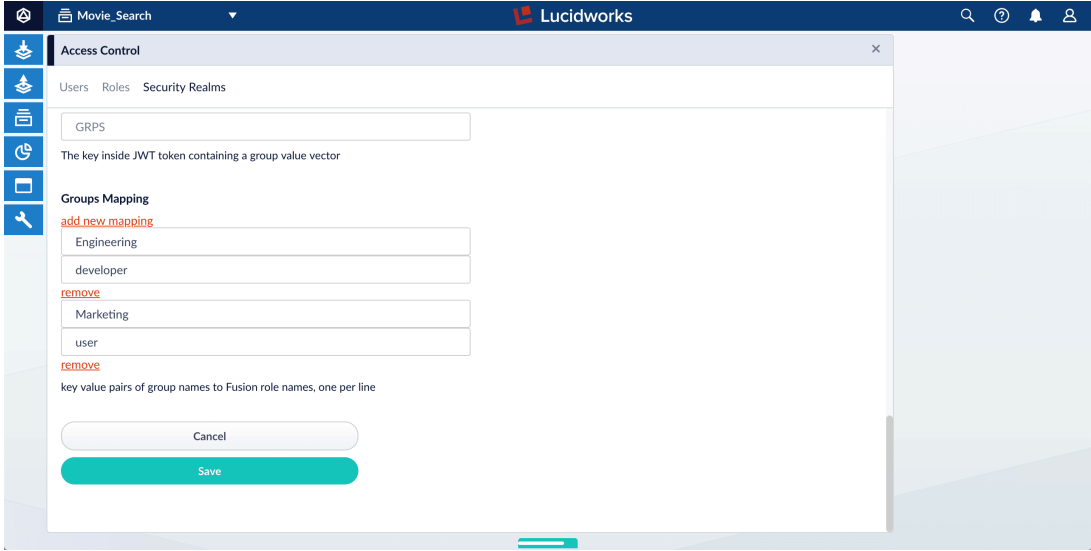
Example data inside token:

```
{
  "iss": "fusion-enterprise-app",
  "iat": "1562633069",
  "sub": "username",
  "groups": ["group-1", "group-2"]
}
```

Create the Fusion JWT realm

To create a Fusion realm, in the Fusion UI:

1. Click **System > Access Control**.
2. Click the **Security Realms** tab, then **Add Security Realm**.
3. Enter a realm name. Under **type**, select **jwt**.
4. Create the security realm with the following values:

Value	Description
Realm type	jwt
Roles	Check the roles that this realm provides after successfully authenticating a user by default.
JWT Issuer	A unique value that is used in the JWT authorization header. This value should match the value of the <code>iss</code> in the token, for example <code>fusion-enterprise-app</code> .
JWT Set URI	<i>Optional.</i> The URI that the Signing Key can be downloaded from. Either this field or <code>Signing Key</code> must be filled.
Signing Key	<i>Optional.</i> A string of secret characters that will be used to encrypt the JWT token. The key must be a shared public key. Either this field or <code>JWT Set URI</code> must be filled.
Groups Key	The JWT token value that contains the list of groups the user is in.
Groups Mapping	A list of pairs which associate groups to roles. For example: <code>{<name of group>, <role assigned to group>}</code>. 1. Click Add new mapping. Two rows appear. 2. In the first row, add a group. The JWT token contains the groups for a particular user. 3. In the second row, add the role or roles (separated by spaces) for that group. 
User ID Attribute	The field in JWT from which Fusion takes the name of the user.

5. Click **Save**.

Validate the new realm

When you send a request to Fusion, you should receive a response. The request to Fusion looks like this:

```
curl http://127.0.0.1:8764/api/users -H 'authorization: Bearer <token-header>.<token-payload>.<token-signing-key>'
```

☐ Bearer is case sensitive

If no signing key is used, truncate the JWT to read ... Bearer <token-header>.<token-payload>.

Enable diagnostic logs for debugging

1. Navigate to your Fusion home directory, then open the **conf** folder.
2. Edit the file `proxy-log4j2.xml`.
3. In the `<Loggers>` element, add the following:

```
<logger name="com.lucidworks.apollo-admin.models.realm-config" level="DEBUG" /> <logger name="com.lucidworks.apollo-admin.middleware.authc.jwt" level="DEBUG" />
```

4. Restart proxy for this change to take effect: `bin/proxy restart`

Now `var/log/proxy/proxy.log` will show some diagnostic logs to help you troubleshoot your issues.

Resources

JWT.io (<https://jwt.io>) is an excellent resource for learning about JWT, creating tokens, debugging tokens, and more.

Example of generating a JWT using PyJWT

Typically, you will generate your JWT token from the application you are integrating with Fusion APIs.

This example, however, uses Python to create a JWT token with the PyJWT Python egg.

1. Install the PyJWT egg:

```
pip install pyjwt
```

2. Inside a Python console, run the following command using your realm configurations:

```
import jwt from datetime import datetime, timedelta key = '<signing key>'
jwt_issuer = '<jwt issuer>' username = '<username>' groups = ['<groups assigned to user>'] iat = datetime.utcnow() exp = iat + timedelta(minutes=5) payload = {'iss': jwt_issuer, 'iat': iat, 'exp': exp, 'sub': username, 'groups': groups}
print (jwt.encode(payload, key, algorithm='HS256'))
```

A Bearer token will be printed for use in API requests:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJmdXNpb24tYXBwIiwiaWF0IjoxNTYyNjMzMZDY5LCJzdWIiOiJhZG1pbiIsImdyb3VwcyciOiJ6WyJncm91cDEiXX0.HYyv_XlijDe01zdZ1Cd_gifgcIAFiZ1ldkbgk0hnWro

3. You can now use this as a bearer token in an authorization header to authenticate to Fusion.

Test your Fusion API web service call with cURL. For example, if your role is an administrator, call:

```
curl -H "Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJmdXNpb24iLCJpYXQiOiJlMzY3ODI1MTc4NjQsInN1YiI6InRlc3QtZnVzaW9uIiwia2JvdXBzIjpjbImdyb3VwLW9uZSI6Imdyb3VwLXR3byJdfQ._ACGk4q3Y5g8QgvLFUUQicMN2ynACypcpei2JmYFlzc" http://localhost:8764/api/roles
```

Custom JavaScript Stages Variables and Parameters

JavaScript Stages Variables and Parameters

JavaScript (<http://en.wikipedia.org/wiki/JavaScript>) is a lightweight scripting language. The JavaScript in a JavaScript stage is standard ECMAScript (<http://en.wikipedia.org/wiki/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
<code>request</code>	<i>Request</i>	The Solr query information.
<code>response</code>	<i>Response</i>	The Solr response information.
<code>ctx</code>	<i>Context</i>	A reference to the container which holds a map over the pipeline properties. Used to update or modify this information for downstream pipeline stages.
<code>collection</code>	String	The name of the Fusion collection being indexed or queried.
<code>solrServer</code>	<i>BufferingSolrServer</i>	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 <code>SolrClient</code> (https://lucene.apache.org/solr/5_2_1/solr-solrj/org/apache/solr/client/solrj/SolrClient.html) for details.
<code>solrServerFactory</code>	<i>SolrClientFactory</i>	The <code>SolrCluster</code> server used for lookups by collection name which returns a Solr server instance for a that collection. For example: <code>var productsSolr = solrServerFactory.getSolrServer("products");</code>
<code>QueryRequestAndResponse</code>	<i>QueryRequestAndResponse</i>	Returns list of info as a string.

Syntax Variants

JavaScript stages are written using *function syntax*, which passes variables as function parameters.

Function Syntax

```
function(request,response) {  
    request.addParam("foo", "bar");  
}
```

Function syntax is used for the examples in this document.

Global variable `logger`

The global variable named `logger` writes messages to the log file of the server running the pipeline. This variable is truly global and does not need to be declared as part of the function parameter list.

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".

Solr Query Language Cheat Sheet

This cheat sheet is a quick reference to the Solr query language. Use this syntax when querying Fusion via the *Query API*

There are two ways to query a Fusion collection using the parameters below:

- Enter query parameters in the *Query Workbench* or the Quickstart.
- Append query parameters to the `/query/{id}` endpoint.

Wildcards and regular expressions are supported.

Wildcards

- `?`
Single-character wildcard
- `*`
Multi-character wildcard

Common query parameters

<code>q</code>	The full Solr query, using Lucene query syntax (https://lucene.apache.org/core/2_9_4/queryparsersyntax.html).
<code>fq</code>	Filter query (https://solr.apache.org/guide/6_6/common-query-parameters.html#CommonQueryParameters-Thefq_FilterQuery_Parameter). A query string that limits the query results without influencing their scores.
<code>sort</code>	Sort field/direction (https://solr.apache.org/guide/8_8/common-query-parameters.html#sort-parameter). The field on which to sort, followed by a space and direction (<code>desc</code> or <code>asc</code>). You can specify multiple sort fields like this: <code>sort=title asc,year desc</code> .
<code>rows</code>	Max results per page (https://solr.apache.org/guide/8_8/common-query-parameters.html#rows-parameter). This sets the "page size" for paginated search results.
<code>start</code>	Pagination offset (https://solr.apache.org/guide/8_8/common-query-parameters.html#start-parameter). The number of results to skip, for pagination purposes.
<code>fl</code>	Field List (https://solr.apache.org/guide/8_8/common-query-parameters.html#fl-field-list-parameter). The list of fields to return in the query results.
<code>wt</code>	Response writer (https://solr.apache.org/guide/8_8/common-query-parameters.html#wt-parameter). Select the response format by specifying one of Solr's response writers (http://wiki.apache.org/solr/QueryResponseWriter#List_of_Writers_Available).

See also the Solr facet parameters documentation (https://solr.apache.org/guide/6_6/faceting.html#Faceting-GeneralParameters).

Query examples

1. Search only the `title` field in the "docs" collection for the term "solr", and format the results as JSON:

```
http://FUSION_HOST:FUSION_PORT/api/query-  
pipelines/default/collections/docs/select?q=solr&fl=title&wt=json
```

2. Use the following query with wildcards and character escaping in the Query Workbench to search for both `BLU RAY MOVIES` and `CAMERA` classes in the `VIDEO/CD` and `PHOTO/CAMERA` departments:
 - `class_s:*RA* AND department_s:*O\C*`
3. Use a movie collection to select the movie ID and rating where the rating is equal to 4.

```
http://FUSION_HOST:FUSION_PORT/api/query-  
pipelines/default/collections/docs/select?q=*:*&fl=movie_id,rating&fq=rating:4
```

4. Search for movies with "black" in the title, starting at the 20th result for pagination, then return the results as XML:

```
http://FUSION_HOST:FUSION_PORT/api/query-  
pipelines/default/collections/docs/select?  
q=black&fl=title&start=20&rows=20&wt=xml
```

Querying Fusion SQL

In addition to the SQL functions provided by Spark, Fusion provides several additional functions to simplify common aggregation tasks. To recap, a UDAF aggregates multiple rows for the same group by key and a UDF performs some operation on a single row.

weighted_sum

The `weighted_sum` UDAF takes a weight, type, and type-weight mapping to produce an aggregated weight. For example, consider the following SQL snippet:

```
SELECT query, doc_id, filters, weighted_sum(typed_weight_d, type,
'click:1.0, cart:10.0') AS weight_d FROM signal_type_groups GROUP BY query,
doc_id, filters
```

When applied to the rows in the table below, the `weighted_sum` function produces a final `weight_d` of 12.0 ($2 \times 1.0 + 1 \times 10.0$). The UDAF is passed rows grouped by `query`, `doc_id`, and `filters`.

query	type	doc_id	filters	typed_weight_d
iPad	click	1	gear	2
iPad	cart	1	gear	1

significantTerms The `significantTerms` UDF allows the user to query data for significant terms within the results. This identifies terms that appear an unusually frequently in a specific result set relative to the index. It can be used to identify anomalies in data.

This UDF can be used on tokenized text and multi-value and single-value string fields.

Example:

```
select significantTerms(fieldX), score from collectionName where fieldy = 'my-
query' limit 10
```

timediff

The `timediff` UDF computes the difference, in milliseconds, between two timestamps in the same row. From the `session_rollup` job, the `timediff` function computes the difference between the current time and the last activity in a session.

click_pos

The `click_pos` UDF computes either a reciprocal rank or a raw click position (using a 0-based index) of a document in a page of results. This UDF is used to compute the mean reciprocal rank (MRR) for experiments. For example, given the following list of documents and a doc ID, the `click_pos` UDF will return 2:

```
docs: a,b,c,d
doc ID: c
```

concat_text

The `concat_text` UDF combines multivalued text fields coming from Solr into a field with a single value delimited by spaces. This UDF is useful when a field returned from Solr uses the `_txt` suffix, which indicates a multivalued text field.

Cumulative Probability (cumulative_probability)

The `cumulative_probability` function calculates the probability that a value drawn from a numeric field will be less than or equal to a specific value. The `cumulative_probability` function takes three parameters:

1. The numeric field to compute the cumulative probability from
2. The specific numeric value to compute the cumulative probability for
3. The sample size

Sample syntax

```
select cumulative_probability(sepal_length_d, 7.7, 150) as cumulative_prob from iris
```

Result set

The result set for the `cumulative_probability` function contains a single record with the calculated cumulative probability. The `cumulative_probability` function returns the cumulative probability calculation.

Sample result set in Apache Zeppelin

The screenshot shows the Apache Zeppelin interface. At the top right, it says "FINISHED" with a play button icon. Below this is a text area containing the SQL query: `select cumulative_probability(sepal_length_d, 7.7, 150) as cumulative_prob from iris`. Below the query is a toolbar with various icons for visualization and settings. Below the toolbar is a "settings" dropdown menu. Below the settings is a table with one column, "cumulative_prob", and one row with the value "0.99333". At the bottom, it says "Took 1 sec. Last updated by anonymous at July 31 2020, 9:11:51 AM. (outdated)".

cumulative_prob
0.99333

Visualization

Sample visualization of the `cumulative_probability` function using the Apache Zeppelin number visualization:

FINISHED    

```
select cumulative_probability(sepal_length_d, 7.7, 150) as cumulative_prob  
from iris
```



[settings](#) ▼

Cumulative Prob

0.99333

Took 1 sec. Last updated by anonymous at July 31 2020, 9:11:51 AM. (outdated)

Mann-Whitney (mann_whitney)

The `mann_whitney` function performs a Mann-Whitney U-test on two random samples drawn with different queries. The `mann_whitney` function takes four parameters:

1. the numeric field to perform the Mann-Whitney U-test on
2. the Lucene/Solr query for drawing the first sample
3. the Lucene/Solr query for drawing the second sample
4. the sample size for both samples

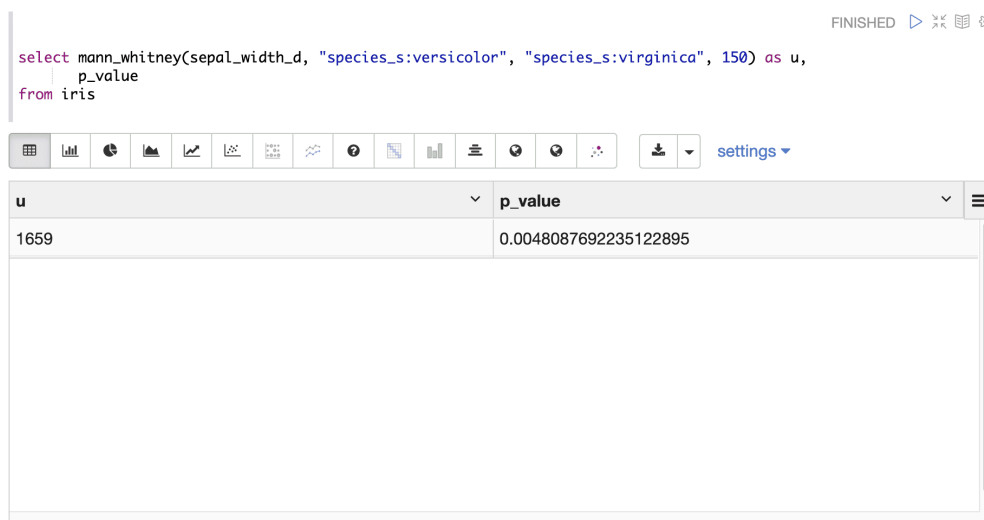
Sample syntax

```
select mann_whitney(sepal_width_d, "species_s:versicolor", "species_s:virginica",  
150) as u, p_value from iris
```

Result set

The result set for the `mann_whitney` function contains one row with the results of the `mann_whitney` test. The `mann_whitney` function returns the U-statistic. The `p_value` field can be selected to return the p-value for the Mann-Whitney test.

Sample result set in Apache Zeppelin



The screenshot shows the Apache Zeppelin interface. At the top, a SQL query is entered in a text area: `select mann_whitney(sepal_width_d, "species_s:versicolor", "species_s:virginica", 150) as u, p_value from iris`. The status bar indicates "FINISHED". Below the query, a toolbar contains various icons for visualization and settings. The result set is displayed in a table with two columns: "u" and "p_value". The first row shows the values "1659" and "0.0048087692235122895".

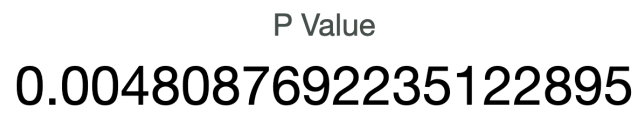
u	p_value
1659	0.0048087692235122895

Took 1 sec. Last updated by anonymous at July 31 2020, 10:38:56 AM. (outdated)

Visualization

Sample visualization of the `mann_whitney` function using the Apache Zeppelin number visualization:

```
select mann_whitney(sepal_width_d, "species_s:versicolor", "species_s:virginica", 150) as u,
       p_value
from iris
```



Took 1 sec. Last updated by anonymous at July 31 2020, 10:38:56 AM. (outdated)

Probability (probability)

The `probability` function calculates the probability that a value drawn from a numeric field will fall within specific range. The `probability` function takes four parameters:

1. the numeric field to compute the probability from
2. the numeric start range
3. the numeric end range
4. the sample size

Sample syntax

```
select probability(sepal_length_d, 5.1, 7.7, 150) as prob from iris
```

Result set

The result set for the `probability` function contains a single record with the calculated probability. The `probability` function returns the probability calculation.

Sample result set in Apache Zeppelin

The screenshot shows the Apache Zeppelin web interface. At the top, a SQL query is entered in a text area: `select probability(sepal_length_d, 5.1, 7.7, 150) as prob from iris`. To the right of the query, the status 'FINISHED' is displayed along with icons for running, undo, redo, and settings. Below the query area is a toolbar with various visualization icons: table, bar chart, pie chart, area chart, line chart, scatter plot, heatmap, gauge, treemap, sunburst, and others. A 'download' button and a 'settings' dropdown are also visible. The result set is displayed in a table with a single column named 'prob' and a single row containing the value '0.72'. The table has a dropdown arrow and a menu icon in the top right corner.

prob
0.72

Took 1 sec. Last updated by anonymous at July 31 2020, 9:56:26 AM. (outdated)

Visualization

Sample visualization of the probability function using the Apache Zeppelin number visualization:


```
select probability(sepal_length_d, 5.1, 7.7, 150) as prob
from iris
```

FINISHED ▶ 🔍 📖 ⚙️



settings ▼

Prob
0.72

Took 1 sec. Last updated by anonymous at July 31 2020, 9:56:26 AM. (outdated)

QQPlot

The `qqplot` function calculates percentiles (1-99) for two sample sets, based on queries, so they can be compared visually. The `qqplot` function takes four parameters:

1. The numeric field for which to calculate the percentiles
2. The Lucene/Solr query for the sample set A
3. The Lucene/Solr query for the sample set B
4. The sample size for both sample sets

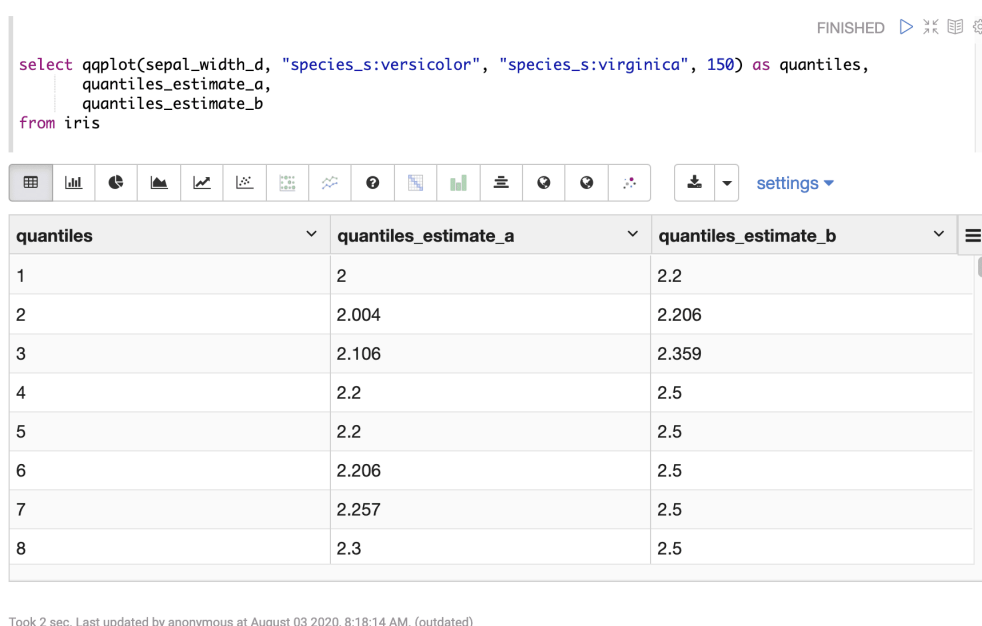
Sample syntax

```
select qqplot(sepal_width_d, "species_s:versicolor", "species_s:virginica", 150)
as quantiles, quantiles_estimate_a, quantiles_estimate_b from iris
```

Result set

The result set for the `qqplot` function contains one row for each percentile 1-99. The `qqplot` function returns the percentile (1-99). The `quantiles_estimate_a` and `quantiles_estimate_b` fields contain the estimated percentile value, for sample sets A and B, at each percentile.

Sample result set in Apache Zeppelin



The screenshot shows the Apache Zeppelin interface. At the top, a SQL query is entered in a text area: `select qqplot(sepal_width_d, "species_s:versicolor", "species_s:virginica", 150) as quantiles, quantiles_estimate_a, quantiles_estimate_b from iris`. Below the query, there is a toolbar with various icons for visualization and settings. The result set is displayed as a table with three columns: `quantiles`, `quantiles_estimate_a`, and `quantiles_estimate_b`. The table contains 8 rows of data, representing percentiles 1 through 8. The status bar at the bottom indicates "Took 2 sec. Last updated by anonymous at August 03 2020, 8:18:14 AM. (outdated)".

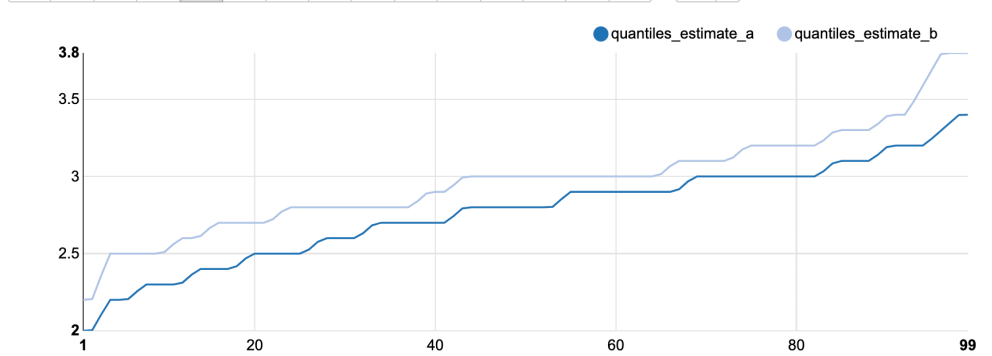
quantiles	quantiles_estimate_a	quantiles_estimate_b
1	2	2.2
2	2.004	2.206
3	2.106	2.359
4	2.2	2.5
5	2.2	2.5
6	2.206	2.5
7	2.257	2.5
8	2.3	2.5

Visualization

The `qqplot` function can be visualized by plotting the percentile (1-99) on the x-axis and the `quantiles_estimate_a` and `quantiles_estimate_b` columns on the y-axis.

Sample visualization of percentiles in Apache Zeppelin

```
select qqplot(sepal_width_d, "species_s:versicolor", "species_s:virginica", 150) as quantiles,
       quantiles_estimate_a,
       quantiles_estimate_b
from iris
```



Took 2 sec. Last updated by anonymous at August 03 2020, 8:18:14 AM. (outdated)

Time Series (date_format)

Fusion SQL provides a powerful and flexible time series aggregation query through the use of the `date_format` function. Aggregations that group by a `date_format` are translated to a Solr range facet query. This allows for fast, interactive time series reporting over large data sets.

An example of a time series aggregation is shown below:

```
select date_format(rec_time, 'yyyy-MM') as month, count(*) as cnt from logrecords
where rec_time > '2000-01-01' and rec_time < '2010-01-01' group by month
```

The `date_format` function is used to specify both the output format and the time interval in one compact pattern as specified by the Java SimpleDateFormat (<https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html>) class.

The example above is performing a monthly time series aggregation over the `rec_time` field which is a `datetime` field.

To switch to a daily time series aggregation all that is needed is to change the date pattern:

```
select date_format(rec_time, 'yyyy-MM-dd') as day, count(*) as cnt from
logrecords where rec_time > '2000-01-01' and rec_time < '2000-12-31' group by day
```

Interval Gap

The interval gap can be defined with the optional `gap` parameter. If the gap parameter is not defined it is inferred from the date format string. For example the date format string 'yyyy-MM' will have an inferred gap of 1MONTH.

To specify a custom gap you can add the gap parameter to the date format string as follows:

```
select date_format(rec_time, 'yyyy-MM-dd gap=2DAYS') as day, count(*) as cnt from
logrecords where rec_time > '2000-01-01' and rec_time < '2000-12-31' group by day
```

The time series above will compute a time series with 2 day intervals between buckets.

Interval gaps can be specified for: YEARS, MONTHS, DAYS, HOURS, MINUTES and SECONDS.

Date math predicates

Fusion SQL also supports date math predicates through the `date_add`, `date_sub`, and `current_date` functions.

Below is an example of the use of date math predicates.

```
select date_format(rec_time, 'yyyy-MM-dd gap=3DAYS') as day, count(*) as cnt from
logrecords where rec_time > date_sub(current_date(), 30) group by day
```

Auto-filling of time intervals

Fusion SQL automatically fills any time interval that does not contain data with zeroes. This ensures that the full time range is included in the output which makes the time series results easy to visualize in charts.

Sort Order

Time series aggregations are sorted by default in time ascending order. The `ORDER BY` clause can be used to sort time series aggregation results in a different order.

Having Clause

A `HAVING` clause can also be applied to a time series query to limit the results to rows that meet specific criteria.

click_pos

The `click_pos` UDF computes either a reciprocal rank or a raw click position (using a 0-based index) of a document in a page of results. This UDF is used to compute the mean reciprocal rank (MRR) for experiments. For example, given the following list of documents and a doc ID, the `click_pos` UDF will return 2:

```
docs: a,b,c,d  
doc ID: c
```

concat_text

The `concat_text` UDF combines multi-valued text fields coming from Solr into a field with a single value delimited by spaces. This UDF is useful when a field returned from Solr uses the `_txt` suffix, which indicates a multi-valued text field.

significantTerms

The `significantTerms` UDF allows the user to query data for significant terms within the results. This identifies terms that appear an unusually frequently in a specific result set relative to the index. It can be used to identify anomalies in data.

This UDF can be used on tokenized text and multi-value and single-value string fields.

Example:

```
select significantTerms(fieldX), score from collectionName where fieldy = 'my-  
query' limit 10
```

timediff

The `timediff` UDF computes the difference, in milliseconds, between two timestamps in the same row. From the `session_rollup` job, the `timediff` function computes the difference between the current time and the last activity in a session.

ttest

The `ttest` function performs a two sided t-test on two random samples drawn from two queries. The `ttest` function takes four parameters:

1. the numeric field on which to perform the t-test
2. the query for drawing the first sample
3. the query for drawing the second sample
4. the sample size for both samples

Sample syntax

```
select ttest(sepal_width_d, "species_s:versicolor", "species_s:virginica", 150)
as t, p_value from iris
```

Result set

The result set for the `ttest` function contains one row with the results of the `ttest`. The `ttest` function returns the t statistic. The `p_value` field can be selected to return the p-value for the `ttest`.

Sample result set in Apache Zeppelin:

FINISHED ▶ ⌕ ⌕ ⌕ ⌕

```
select ttest(sepal_width_d, "species_s:versicolor", "species_s:virginica", 150) as t,
       p_value
from iris
```



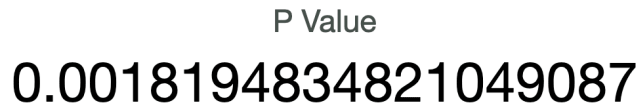
t	p_value
-3.2057607502218253	0.0018194834821049087

Took 0 sec. Last updated by anonymous at July 31 2020, 10:20:28 AM. (outdated)

Visualization

Sample visualization of the `ttest` function using the Apache Zeppelin number visualization:

```
select ttest(sepal_width_d, "species_s:versicolor", "species_s:virginica", 150) as t,
       p_value
from iris
```



Took 0 sec. Last updated by anonymous at July 31 2020, 10:20:28 AM. (outdated)

```
SELECT query, doc_id, filters, weighted_sum(typed_weight_d, type,
'click:1.0, cart:10.0') AS weight_d FROM signal_type_groups GROUP BY query,
doc_id, filters
```

query	type	doc_id	filters	typed_weight_d
iPad	click	1	gear	2
iPad	cart	1	gear	1

Fusion Server Directories, Files, and Ports

Configuration files

Fusion configuration files are stored in `https://FUSION_HOST:FUSION_PORT/conf/` (on Unix or macOS) or `fusion\4.1.x\conf\` (on Windows). The contents of this directory are as follows:

<code>fusion.properties</code>	Fusion's main configuration file, which defines the common environment variables used by the Fusion run scripts.
<code>hive-site.xml</code>	Configuration for Fusion's <i>Serializer/Deserializer (SerDe)</i> for Hive.
<code>zookeeper/commons-logging.properties</code> <code>zookeeper/zoo.cfg</code>	ZooKeeper configuration files.
<code>agent-log4j2.xml</code> <code>api-log4j2.xml</code> <code>connectors-log4j2.xml</code> <code>solr-log4j2.xml</code> <code>spark-driver-log4j2.xml</code> <code>spark-master-agent-log4j2.xml</code> <code>spark-master-log4j2.xml</code> <code>spark-worker-agent-log4j2.xml</code> <code>spark-worker-log4j2.xml</code> <code>sql-agent-log4j2.xml</code> <code>sql-log4j2.xml</code> <code>ui-log4j2.xml</code> <code>zk-log4j2.xml</code> <code>zookeeper/log4j2.xml</code> <code>zookeeper/log4j.properties</code>	Logging configuration files. Fusion uses the Apache Log4j 2 (http://logging.apache.org/log4j/2.x/manual/index.html) logging framework with Jetty. Log levels, frequencies, and log rotation policy can be configured by changing these configuration files. See the Log4j2 Configuration (http://logging.apache.org/log4j/2.x/manual/configuration.html) guide.

Port configuration

Fusion services run in their own JVM and listen for requests on a number of ports. Environment variables, set in a common configuration file, are used to specify the port a service uses. To change the port(s) a service uses, you must change the settings in the configuration file.

Default ports

This table lists the default port numbers used by Fusion processes. Port settings are defined in the `:fusion.properties` file in `https://FUSION_HOST:FUSION_PORT/conf/` (on Unix or macOS) or `fusion\4.1.x\conf\` (on Windows).

Port	Service
8091	Fusion agent
8763	Fusion UI service (use port 8764 to access the Fusion UI)
8764	Fusion proxy This service includes the Fusion Authorization Proxy.
8765	Fusion API Services
8766	Spark Master
8769	Spark Worker
8780	Web Apps This service delivers the UIs of Fusion apps.
8781	Log shipper Monitoring port that agent uses to check the health of the log shipper process. This port does not need to be accessible from other nodes.
8983	Solr This is the embedded Solr instance included in the Fusion distribution.
8984	Connectors Classic Service This service runs nondistributed connector jobs. It uses HTTP/1.1 and has no SDK.
9983	ZooKeeper The embedded ZooKeeper used by Fusion services. The ZooKeeper port is also defined in the configuration file for the embedded ZooKeeper, <code>https://FUSION_HOST:FUSION_PORT/conf/zookeeper/zoo.cfg</code> (on Unix or macOS) or <code>fusion\4.1.x\conf\zookeeper\zoo.cfg</code> (on Windows). Look for <code>clientPort</code>. If you run Fusion with the embedded ZooKeeper, remember to change the port number in both places.
4710 0- 4809 9	Apache Ignite TCP communication port range (used by the API, Connectors Classic, Connectors RPC, and Proxy services)
4810 0- 4819 9	Apache Ignite shared memory port range (used by the API, Connectors Classic, Connectors RPC, and Proxy services)
4920 0- 4929 9	Apache Ignite discovery port range (used by API, Connectors Classic, Connectors RPC, and Proxy services)

Jetty ports

Jetty is used to run the Admin UI, API, Connectors Classic, Proxy, Solr, and Web Apps services. For each of these services, Jetty runs the service on the assigned port and listens on a second port for shutdown requests. Therefore, `fusion.properties` defines pairs of ports for components running on Jetty, such as:

```
api.port = 8765 api.stopPort = 7765
```

Spark ports

This table lists the default port numbers used by Spark processes in Fusion.

Port number	Process
4040	SparkContext web UI
7337	Shuffle port for Apache Spark worker
8600-8616 (Fusion 4.0.x only.)	Akka ports used between Spark driver, master, workers, and API See the Akka documentation (https://doc.akka.io/docs/akka/snapshot/scala/remoting.html).
8767	Spark master web UI
8770	Spark worker web UI
8766	Spark master listening port
8769	Spark worker listening port
8772 (<code>spark.driver.port</code>)	Spark driver listening port
8788 (<code>spark.blockManager.port</code>)	Spark BlockManager port

If a port is not available, Spark uses the next available port by adding `1` to the assigned port number. For example, if 4040 is not available, Spark uses 4041 (if available, or 4042, and so forth).

Ensure that the ports in the above table are accessible, as well as a range of up to 16 subsequent ports. For example, open ports 8772 through 8787, and 8788 through 8804, because a single node can have more than one Spark driver and Spark BlockManager.

Directories

The directory where the Fusion files go for a specific version of Fusion is the *Fusion home directory*. The Fusion home directory is a version-numbered directory (for example, `4.1.0`) below the directory `fusion`. This installation strategy lets you install multiple versions of Fusion and switch between them.

The directories found in the Fusion home directory in `https://FUSION_HOST:FUSION_PORT/` (on Unix or macOS) or `fusion\4.1\` (on Windows) are:

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

Symbolic links on UNIX

To simplify access to the latest version of Fusion and to files in the `bin`, `conf`, and `var` directories, Fusion creates a symbolic link `latest` to the latest version and symbolic links `bin`, `conf`, and `var` to `latest/bin`, `latest/conf`, and `latest/var` respectively.

For example, if `latest` is `4.1.1`, then instead of entering this command to change to the `bin` directory:

```
cd /path/to/fusion/4.1.1/bin
```

You could just type:

```
cd /path/to/fusion/latest/bin
```

To avoid possible confusion in the documentation, we spell out the path below the `fusion` directory.

From the `fusion` directory, you can view the symbolic links by typing:

```
find . -maxdepth 1 -type l -ls
```

To change the version of Fusion to which the symbolic links refer, unlink and relink `latest`. For example:

```
cd /path/to/fusion unlink latest link -s 4.1.2 latest
```

Log files

Log files are found in directories under `https://FUSION_HOST:FUSION_PORT/var/log/` (on Unix or macOS) or `fusion\4.1.x\var\log\` (on Windows).

Because the Fusion components run in separate JVMs, each component has its own set of log files and files that monitor all garbage-collection events for that process.

Name	Description
<code>admin-ui, webapps</code>	Fusion UI messages. Messages are logged to <code>jetty-<date>.stderrout.log</code> .
<code>agent</code>	Fusion agent logging and error messages
<code>api</code>	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.
<code>connectors</code>	Fusion connector services logging and error messages. Fusion index pipeline logging stages write to this file.
<code>log-shipper</code>	See <i>Configure Fusion logging</i>
<code>proxy</code>	Messages from the Fusion proxy, responsible for authentication and HTTP load balancing.
<code>solr</code>	Messages from Solr
<code>spark-master</code>	Spark-master logs
<code>spark-worker</code>	Spark-worker logs
<code>sql</code>	SQL logs
<code>zookeeper</code>	ZooKeeper messages

Every component logs all messages to a log file named `<component>.log`. For example, the full path to the log file for the connectors services is `https://FUSION_HOST:FUSION_PORT/var/log/connectors/connectors.log` (on Unix or macOS) or `fusion\4.1.x\var\log\connectors\connectors.log` (on Windows).

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

The Fusion API, Fusion UI, Connectors Classic, Proxy, Web Apps, and Solr services all run inside a Jetty server. The Jetty server logs are also written to each component's log file directory. The Jetty server logs are named:

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

Template Expressions

Template expressions are a way to map field values in a document. They are used to configure some Fusion pipeline stage and messaging services. When the value of a configuration parameter is a template expression, that expression is dynamically evaluated at runtime by the StringTemplate (<https://github.com/antlr/stringtemplate4/blob/master/doc/cheatsheet.md>) library.

Fusion template expressions are delimited by angle bracket characters `<` and `>`. The expression consists of the name of a variable in the scope of that component.

Since these variables are Java objects, if object **foo** of type Foo has a field named **bar**, the expression `<foo.bar>` will evaluate to the string representation of the contents of field **bar**.

For example, given a PipelineDocument where a field named **title** has a value of **Star Wars**, the template expression `<doc.title>` evaluates to **Star Wars**.

See the reference guide for *index pipeline stage*, *query pipeline stage*, and *messaging service* template expression variables.

Index Pipeline Stage Template Expression Variables

Index stages have the following variables available:

- `ctx`. *Context*: used to pass variables between stages
- `doc`. *PipelineDocument*: Fusion internal representation of a Solr Document. Please note that if you change this class in ways that affect the serialized form (adding fields, for example) you need to change the deserialization code in `GenericJSONDocumentMessageReader`.

The format required is `<request.q>`.

Messaging Services Template Expression Variables

Messages have a set of variables which correspond to the parts of a system message:

- id
- to
- from
- subject
- body
- type
- schedule

A system message is the result of evaluating the following expression:

```
<id><to><from><subject><body><type><schedule>
```

Query Pipeline Stage Template Expression Variables

Query stages have the following variables available:

- `ctx`. *Context*: used to pass variables between stages
- `reqResp`. *QueryRequestAndResponse*: deserializes pre-request parameters and makes them available

The format required is `<request.q>`.

Example

Check that a query has an `fq` filter on a given field for a given value:

In the UI, set property name to `ids` and Property Value to `req.hasParam("fq")`

Property Name	Property Value
ids	<code>\${request.params.q}</code>

Resources and Support

In addition to this documentation, these resources are available to assist you.

Self-help resources

- **Blog** (<https://lucidworks.com/blog/>)
Learn from the experts in these in-depth topics.
- **Videos and webinars** (<https://lucidworks.com/videos/>)
View demos, webinars, and tutorials.
- **Answers** (<https://support.lucidworks.com/>)
Browse questions and answers from the Lucidworks community, or ask a new question.
- **Knowledge base** (<https://support.lucidworks.com/>)
Find solutions in these articles from the Lucidworks support team.
- **Fusion Server 4.x File Download Links**
Download Fusion products and drivers.
- **Fusion 4.x V1 Connector Downloads**
Download Fusion connectors.

* Customer login required.

Language Support in Fusion

The table below shows how various language are supported in the Fusion product suite. The meanings of the columns are as follows:

- **Encoding.** Fusion and App Studio support the language encodings of the selected languages both for search and display.
- **Language detection.** The superset of languages supported was determined based on our use of the Language Detection Library for Java that recognizes 71 languages and the supported languages in Solr.
- **Thesaurus.** Solr can support a supplied thesaurus for the selected language.
- **Did you mean (spell check).** Fusion can return spelling suggestions for the selected language.
- **Stemming.** Supported with an externally-supplied dictionary or out of the box with `HunspellStemFilter`.
- **Summary.** The ability to display matching text from a search request which may also include highlighting.
- **App Studio.** Text in the selected language can be displayed and Right-to-Left languages are also supported.
- **Administration tools.** The language is supported in the Fusion UI.

Supported language	Encoding	Language detection	Thesaurus	Did you mean (spell check)	Stemming	Summary	App Studio	Administration tools
Afrikaans	x	x	x	x	x (hunspell)	x	x	
Albanian	x	x	x	x	x (hunspell)	x	x	
Arabic	x	x	x	x	x	x	x	
Aragonese	x	x	x	x	x (hunspell)	x	x	
Asturian	x	x	x	x	x (hunspell)	x	x	
Basque	x	x	x	x	x (hunspell)	x	x	
Belarusian	x	x	x	x	x (hunspell)	x	x	
Bengali	x	x	x	x	x (hunspell)	x	x	
Breton	x	x	x	x	x (hunspell)	x	x	
Brazilian Portuguese	x	x	x	x	x	x	x	
Bulgarian	x	x	x	x	x	x	x	
Catalan	x	x	x	x	x	x	x	
Chinese	x	x	x	x	x (performed as part of dictionary-based tokenization; separate stemming step not required)	x	x	
Croatian	x	x	x	x	x (hunspell)	x	x	
Czech	x	x	x	x	x	x	x	
Danish	x	x	x	x	x	x	x	
Dutch	x	x	x	x	x	x	x	
English	x	x	x	x	x	x	x	x
Estonian	x	x	x	x	x (hunspell)	x	x	
Finnish	x	x	x	x	x	x	x	
French	x	x	x	x	x	x	x	

Supported language	Encoding	Language detection	Thesaurus	Did you mean (spell check)	Stemming	Summary	App Studio	Administration tools
Galician	x	x	x	x	x	x	x	
German	x	x	x	x	x	x	x	
Greek	x	x	x	x	x (hunspell)	x	x	
Gujarati	x	x	x	x	x (hunspell)	x	x	
Haitian	x	x	x	x	x (hunspell)	x	x	
Hebrew	x	x	x	x	x (hunspell)	x	x	
Hindi	x	x	x	x	x	x	x	
Hungarian	x	x	x	x	x (hunspell)	x	x	
Icelandic	x	x	x	x	x (hunspell)	x	x	
Indonesian	x	x	x	x	x	x	x	
Irish	x	x	x	x	x	x	x	
Italian	x	x	x	x	x	x	x	
Japanese	x	x	x	x	x	x	x	
Kannada	x	x	x	x	x (hunspell)	x	x	
Khmer	x	x	x	x	x (hunspell)	x	x	
Korean	x	x	x	x	x (hunspell)	x	x	
Lao	x	x	x	x		x	x	
Latvian	x	x	x	x	x	x	x	
Lithuania n	x	x	x	x	x (hunspell)	x	x	
Macedonian	x	x	x	x	x (hunspell)	x	x	
Malay	x	x	x	x	x (hunspell)	x	x	

Supported language	Encoding	Language detection	Thesaurus	Did you mean (spell check)	Stemming	Summary	App Studio	Administration tools
Malayalam	x	x	x	x	x (hunspell)	x	x	
Maltese	x	x	x	x	x (hunspell)	x	x	
Marathi	x	x	x	x	x (hunspell)	x	x	
Myanmar				x		x	x	
Nepali	x	x	x	x	x (hunspell)	x	x	
Norwegian	x	x	x	x	x	x	x	
Occitan	x	x	x	x	x (hunspell)	x	x	
Persian	x	x	x	x	x (hunspell)	x	x	
Polish	x	x	x	x	x	x	x	
Portuguese	x	x	x	x	x	x	x	
Punjabi	x	x	x	x	x (hunspell)	x	x	
Romanian	x	x	x	x	x	x	x	
Russian	x	x	x	x	x	x	x	
Serbian	x	x	x	x	x (hunspell)	x	x	
Slovak	x	x	x	x	x (hunspell)	x	x	
Slovene	x	x	x	x	x (hunspell)	x	x	
Somali	x	x	x	x	x (hunspell)	x	x	
Spanish	x	x	x	x	x	x	x	
Swahili	x	x	x	x	x (hunspell)	x	x	
Swedish	x	x	x	x	x	x	x	
Tagalog	x	x	x	x	x (hunspell)	x	x	
Tamil		x	x	x	x (hunspell)	x	x	

Supported language	Encoding	Language detection	Thesaurus	Did you mean (spell check)	Stemming	Summary	App Studio	Administration tools
Telugu		x	x	x	x (hunspell)	x	x	
Thai	x	x	x	x	x (hunspell)	x	x	
Turkish	x	x	x	x	x	x	x	
Ukrainian	x	x	x	x	x	x	x	
Urdu		x	x	x	x (hunspell)	x	x	
Vietnamese		x	x	x	x (hunspell)	x	x	
Walloon	x	x	x	x	x (hunspell)	x	x	
Welsh	x	x	x	x	x (hunspell)	x	x	
Yiddish	x	x	x	x	x (hunspell)	x	x	

Monitor Fusion 4.x.x

There are many ways to monitor Java Virtual Machines, Unix file systems, networks, and applications like Solr and Fusion. This includes the use of a UI, command-line utilities, or REST APIs and JMX Managed Beans (MBeans). A full-fledged monitoring tool such as Datadog, Nagios, Zabbix, New Relic, SolarWinds, or other comparable tool might be helpful. These tools assist in the analysis of the raw JMX values, which tend to be more informative than the numbers alone.

This topic focuses on generally available options, including: command-line utilities such as `top`, `du -h`, and `df`; REST APIs; and JMX MBean metrics.

Lucidworks does not explicitly recommend any particular monitoring product, nor do we imply suitability or compatibility with Lucidworks Fusion. The aforementioned products are not detailed or reviewed in this topic.

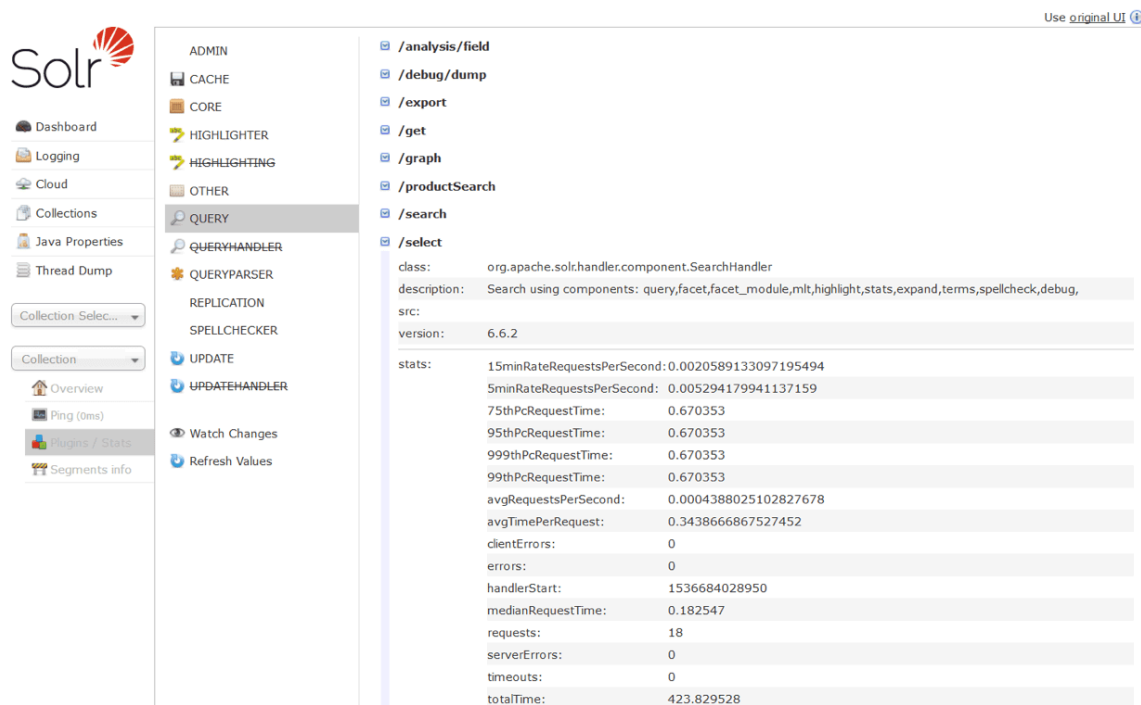
Monitoring via Solr

The Solr Admin console lists information for one Solr node. This can be obtained via HTTP as needed.

GET JVM metrics for a Solr node

```
http://${Solr_Server}:8983/solr/admin/info/system?wt=json
```

You can access detailed metrics in the Solr Admin UI by selecting an individual Solr core and selecting the Plugins/Stats option. Among the published metrics are averages for Query and Index Request Time, Requests Per Second, Commit rates, Transaction Log size, and more.



The screenshot shows the Solr Admin UI interface. On the left is a sidebar with navigation links: Dashboard, Logging, Cloud, Collections, Java Properties, Thread Dump, Collection Select..., Collection, Overview, Ping (0ms), Plugins / Stats (selected), and Segments info. The main content area is divided into three sections. The top section lists various Solr components: ADMIN, CACHE, CORE, HIGHLIGHTER, HIGHLIGHTING, OTHER, QUERY (selected), QUERYHANDLER, QUERYPARSER, REPLICATION, SPELLCHECKER, UPDATE, and UPDATEHANDLER. The middle section shows a list of plugins: /analysis/field, /debug/dump, /export, /get, /graph, /productSearch, /search, and /select (selected). The bottom section displays the details for the /select plugin, including its class (org.apache.solr.handler.component.SearchHandler), description (Search using components: query, facet, facet_module, mlt, highlight, stats, expand, terms, spellcheck, debug), src, version (6.6.2), and a comprehensive stats table.

stats:	
15minRateRequestsPerSecond:	0.0020589133097195494
5minRateRequestsPerSecond:	0.005294179941137159
75thPcRequestTime:	0.670353
95thPcRequestTime:	0.670353
999thPcRequestTime:	0.670353
99thPcRequestTime:	0.670353
avgRequestsPerSecond:	0.0004388025102827678
avgTimePerRequest:	0.3438666867527452
clientErrors:	0
errors:	0
handlerStart:	1536684028950
medianRequestTime:	0.182547
requests:	18
serverErrors:	0
timeouts:	0
totalTime:	423.829528

Obtain these same metrics via HTTP on a per-core basis:

```
http://${Solr_server}:8983/solr/${Collection}_${shard}_${replica}/admin/mbeans?stats=true&wt=json
```

Additionally, you can turn JMX monitoring on by starting Solr with `ENABLE_REMOTE_JMX_OPTS=true`. Refer to Apache's Solr reference guide for configuring JMX for more information.

Finding shard leaders in Solr

In Solr admin console

The Solr admin console's **Cloud > Graph** view can show the Shards and replicas for all Solr collections in a cluster. The nodes on the right-hand side with the filled in dot are the current shard leaders. Query operations are distributed across both leaders and followers but index operations require extra resources from leaders.



With the API

The Solr API allows you to fetch the cluster status, including collections, shards, replicas, configuration name, collection aliases, and cluster properties.

Requests are made with the following URL endpoint: `/admin/{parameter}?action=CLUSTERSTATUS`.

Parameter	Description
collection	The collection or alias name for which information is requested. If omitted, information on all collections in the cluster will be returned. If an alias is supplied, information on the collections in the alias will be returned.
shard	The shard(s) for which information is requested. Multiple shard names can be specified as a comma-separated list.
route	This can be used if you need the details of the shard where a particular document belongs to and you don't know which shard it falls under.

Input

`http://localhost:8983/solr/admin/collections?action=CLUSTERSTATUS`

Output

```
{ "responseHeader":{ "status":0, "QTime":333}, "cluster":{ "collections":{
"collection1":{ "shards":{ "shard1":{ "range":"80000000-ffffffff",
"state":"active", "replicas":{ "core_node1":{ "state":"active",
"core":"collection1", "node_name":"127.0.1.1:8983_solr",
"base_url":"http://127.0.1.1:8983/solr", "leader":"true"}, "core_node3":{
"state":"active", "core":"collection1", "node_name":"127.0.1.1:8900_solr",
"base_url":"http://127.0.1.1:8900/solr"}}}, "shard2":{ "range":"0-7fffffffff",
"state":"active", "replicas":{ "core_node2":{ "state":"active",
"core":"collection1", "node_name":"127.0.1.1:7574_solr",
"base_url":"http://127.0.1.1:7574/solr", "leader":"true"}, "core_node4":{
"state":"active", "core":"collection1", "node_name":"127.0.1.1:7500_solr",
"base_url":"http://127.0.1.1:7500/solr"}}}}, "maxShardsPerNode":"1", "router":
{"name":"compositeId"}, "replicationFactor":"1", "znodeVersion": 11,
"autoCreated":"true", "configName" : "my_config", "aliases":["both_collections"]
}, "collection2":{ "... " } }, "aliases":{
"both_collections":"collection1,collection2" }, "roles":{ "overseer":[
"127.0.1.1:8983_solr", "127.0.1.1:7574_solr"] }, "live_nodes":[
"127.0.1.1:7574_solr", "127.0.1.1:7500_solr", "127.0.1.1:8983_solr",
"127.0.1.1:8900_solr"] } }
```

Monitoring Fusion

The Fusion UI in Fusion 4.2 through 5.5 includes monitoring, troubleshooting, and incident investigation tools in the DevOps Center. The DevOps Center provides a set of dashboards and an interactive log viewer, providing views into this Fusion cluster's hosts and services using metrics and events.

You can monitor general system state and status via HTTP in several ways. For details, see *Checking System State*.

For more information, see the link for your Fusion release:

- *Fusion 4.2.x DevOps Center*
Fusion 4.x.x publishes a wide variety of metrics, including stage-by-stage performance for pipelines via the Fusion Metrics API. The Fusion metrics provide valuable insight into stage development, hot-spot identification, and overall throughput numbers. See *Use System Metrics* for more information.
- *Fusion 5.x.x DevOps Center*
Fusion 5.x.x includes Grafana, which provides enhanced metrics collection and querying. Kubernetes containers may make some access methods more difficult. For example, JMX beans on a container.

Check which Fusion services are running

```
curl https://FUSION_HOST:8764/api
```

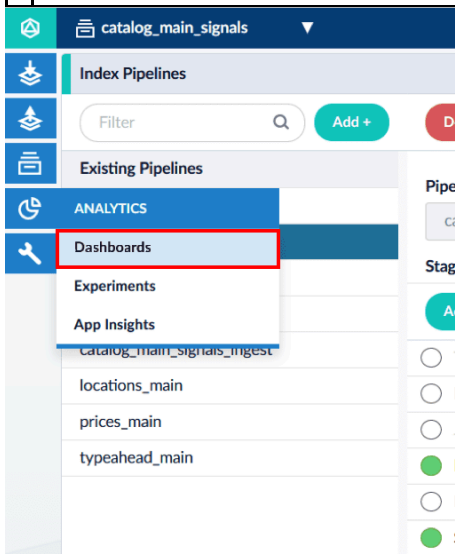
Check the cluster-wide status for a service

```
curl -u USERNAME:PASSWORD -X POST https://FUSION_HOST:8764/api/system/status
```

Monitor the logs collection

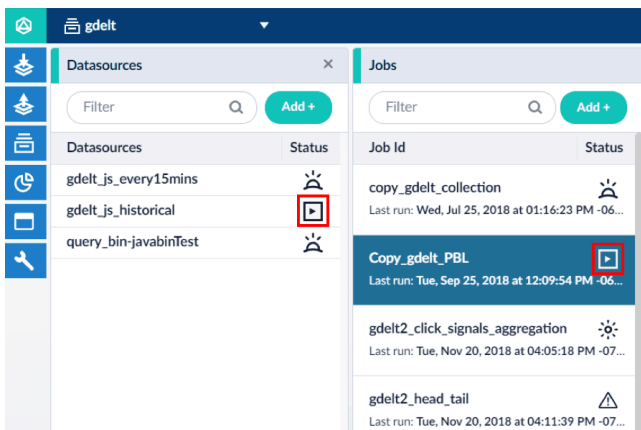
In addition to the on-disk log files, most of the key Fusion logging events are written to the Logs collection. You can monitor these logging events with the Logging Dashboard.

The log creation process can be altered in the `fusion.cors` (`fusion.properties` in Fusion 4.x) file located in `/fusion/latest.x/conf/`. If log files are not being created and stored in the logs collection, check the configurations in this file.



See what scheduled jobs are running

The Fusion UI's Datasources and Jobs pages show the most recent status for data ingestion jobs and other jobs that can be scheduled.



This can also be accomplished using the Jobs API (*Fusion 4.x Jobs API*, *Fusion 5.x Jobs API*).

What to monitor

Several levels of information should be monitored, including hardware resources, performance benchmarks, the Java virtual machine, Lucene/Solr, and others. The point at which a monitored metric should trigger a warning or alarm condition is highly dependent on user-specific factors, such as the target audience, time tolerance thresholds, and real-time needs.

Hardware and resources

This is just knowing what is going on with the overall system resources.

CPU Idle	This value helps determine whether a system is running out of CPU resources. Establishing average values, such as <code>avg(1 min)</code> , helps determine if the CPU Idle value is too high. Establishing these averages for each individual CPU core helps you avoid confusion if, for example, everything is paused on a single CPU core.
Load Average	This value indicates whether processes are being loaded into a queue to be processed. If this value is higher than the number of CPU cores on a system, the machine is failing to load processes. The Load Average value might not indicate a specific problem, but might rather indicate that a problem exists <i>somewhere</i> .
Swap	Most operating systems can extend the capabilities of their RAM by paging portions out to disk before moving it back into RAM as needed. This process is called "swap". Swap may potentially have a Severe impact on the performance of Solr. Avoid swap by reconfiguring services across different nodes or adding memory.
Page In/Out	This value indicates the use of swap space. There might be a small amount of swap space used and little page in/out activity. However, the use of swap space generally indicates the need for more memory.
Free Memory	This value indicates the amount of free memory available to the system. A system with low amounts of free memory can no longer increase disk caching as queries are run. Because of this, index accesses must wait for permanent disk storage, which is much slower. Take note of the amount of free memory and the disk cache in comparison to the overall index size. Ideally, the disk cache should be able to hold all of the index, although this is not as important when using newer SSD technologies.
I/O Wait	A high I/O Wait value indicates system processes are waiting on disk access, resulting in a slow system.
Free Disk Space	It is critical to monitor the amount of free disk space, because running out of free disk space can cause corruption in indexes. Avoid this by creating a mitigation plan to reallocate indexes, purge less important data, or add disk space. Create warning and alarm alerting for disk partitions containing application or log data.

Performance benchmarks

These benchmarks are available via JMX MBeans when using JConsole. They are found under the namespace `solr/<corename>`. Some of these metrics can also be obtained via the fields below.

Average Response Times	Monitoring tools can complete very useful calculations by measuring deltas between incremental values. For example, a user might want to calculate the average response time over the last five minutes. Use the <i>Runtime JMX MBean</i> as the divisor to get performance over time periods. The JMX MBean for the standard request handler is found at: <code>solr/<corename>:type=standard,id=org.apache.solr.handler.StandardRequestHandler,totalTime,requests]</code> This can be obtained for every Solr requestHandler being used.
nth Percentile Response Times	This value allows a user to see the response time for <code>75thPercentile</code>, <code>95thPercentile</code>, <code>99thPercentile</code>, and <code>999thPercentiles</code>. This value is a poor performance benchmark during short spikes in query times, but in other applications, it can show slow degradation problems and baseline behavior. JMX MBean example: <code>"solr/collection1:type=standard,id=org.apache.solr.handler.component.SearchHandler", 75thPcRequestTime</code> The same format is used for all percentiles.
Average QPS	Tracking peak QPS over several-minute intervals can show how well Fusion and Solr are handling queries and can be vital in benchmarking plausible load. JMX MBean is found at: <code>"solr/collection1:type=standard,id=org.apache.solr.handler.StandardRequestHandler"</code>, requests divided by <i>JMX Runtime</i> <code>java.lang:Runtime</code>.
Average Cache/Hit Ratios	This value can indicate potential problems in how Solr is warming searcher caches. A ratio below 1:2 should be considered problematic, as this may affect resource management when warming caches. Again, this value can be used with the Runtime JMX MBean to find changes over a time period. JMX MBeans are found at: <code>"solr/<corename>:type=<cachetype>id=org.apache.solr.search.<cache impl>", hitratio/cumulative hitratio</code>
External Query "HTTP Ping"	An external query HTTP ping lets you monitor the performance of the system by sending a request for a basic status response from the system. Because this request does not perform a query of data, the response is fast and consistent. The average response time can be used as a metric for measuring system load. Under light to moderate loads, the response to the HTTP ping request is returned quickly. Under heavy loads, the response lags. If the response lags considerably, this might be a sign that the service is close to breaking. External query HTTP pings can be configured to target query pipelines, which in turn query Solr. Using this method, you can monitor the uptime of the system by verifying Solr, API nodes, and the proxy service are taking requests. Every important collection should be monitored, as the performance of one collection may differ from another. There are many vendors which provide these services. This includes Datadog, New Relic, StatusCake, and Pingdom.

Java Virtual Machine

The Java Virtual Machine is found via JMX MBeans. Many are also available via REST API calls. See *System API* for details.

JMX runtime

The JMX runtime value is an important piece of data that helps determine time frames and gather statistics, such as performance over the last five minutes. A user can use the delta of the runtime value over the last time period as a divisor in other statistics. This metric can be used to determine how long a server has been running. This JMX MBean is found at:
`"java.lang:Runtime",Uptime`.

Last Full Garbage Collection	Monitoring the total time taken for full garbage collection cycles is important, because full GC activities typically pause processing for a whole application. Acceptable pause times vary, as they relate to how responsive your queries must be in a worst-case scenario. Generally, anything over a few seconds might indicate a problem. JMX MBean: <code>"java.lang:type=GarbageCollector,name=<GC Name>","LastGcInfo.duration"</code> Most fusion log directories also have detailed GC logs.
Full GC as % of Runtime	This value indicates the amount of time spent in full garbage collections. Large amounts of time spent in full GC cycles can indicate the amount of space given to the heap is too little.
Total GC Time as % of Runtime	An indexing server typically spends considerably more time in GC. This is due to all of the new data coming in. When tuning heap and generation sizes, it is important to know how much time is being spent in GC. If too little time is spent, GC triggers more frequently. If too much time is spent, pause frequency increases.
Total Threads	This is an important value to track because each thread takes up memory space, and too many threads can overload a system. In some cases, out of memory (OOM) errors are not indicative of the need for more memory, but rather that a system is overloaded with threads. If a system has too many threads opening, it might indicate a performance bottleneck or a lack of hardware resources. JMX MBean: <code>"java.lang:type=Threading,ThreadCount"</code>.

Lucene/Solr

This can be found via JMX as well.

Auto warming Times	This value indicates how long new searchers or caches take to initialize and load. The value applies to both searchers and caches. However, the <code>warmupTime</code> for the searcher is separate from that of caches. There is always a tradeoff between autowarming times and having prewarmed caches ready for search. If autowarming times are set to be longer than the interval for initializing new searchers, problems may arise. Searcher JMX MBean: <pre>"solr/collection1:type=searcher,id=org.apache.solr.search.SolrIndexSearcher",warmupTime. "solr/collection1:type=documentCache,id=org.apache.solr.search.LRUCache" "solr/collection1:type=fieldValueCache,id=org.apache.solr.search.FastLRUCache" "solr/collection1:type=filterCache,id=org.apache.solr.search.FastLRUCache" "solr/collection1:type=queryResultCache,id=org.apache.solr.search.LRUCache"</pre>

Solr Query Language Cheat Sheet

This cheat sheet is a quick reference to the Solr query language. Use this syntax when querying Fusion via the *Query API*

There are two ways to query a Fusion collection using the parameters below:

- Enter query parameters in the *Query Workbench* or the Quickstart.
- Append query parameters to the `/query/{id}` endpoint.

Wildcards and regular expressions are supported.

Wildcards

Wildcards and regular expressions are supported.

- `?`
Single-character wildcard
- `*`
Multi-character wildcard

Escape characters

Some characters in query fields must be escaped using a backslash `\`. Adding `\` before the following includes them in the query as literal values:

- `?`
- `*`
- `\`
- `/`
- `"`

Queries against facets are case sensitive.

Common query parameters

<code>q</code>	Defines a query using standard query syntax (https://solr.apache.org/guide/8_8/the-standard-query-parser.html#standard-query-parser-parameters). This parameter is mandatory.
<code>fq</code>	Filter query (https://solr.apache.org/guide/8_8/common-query-parameters.html#fq-filter-query-parameter). A query string that limits the query results without influencing their scores.
<code>sort</code>	Sort field/direction (https://solr.apache.org/guide/8_8/common-query-parameters.html#sort-parameter). The field on which to sort, followed by a space and direction (<code>desc</code> or <code>asc</code>). You can specify multiple sort fields like this: <code>sort=title asc,year desc</code> .
<code>rows</code>	Max results per page (https://solr.apache.org/guide/8_8/common-query-parameters.html#rows-parameter). This sets the "page size" for paginated search results.
<code>start</code>	Pagination offset (https://solr.apache.org/guide/8_8/common-query-parameters.html#start-parameter). The number of results to skip, for pagination purposes.
<code>fl</code>	Field List (https://solr.apache.org/guide/8_8/common-query-parameters.html#fl-field-list-parameter). The list of fields to return in the query results.
<code>df</code>	Default field (https://solr.apache.org/guide/8_8/the-standard-query-parser.html#standard-query-parser-parameters). Used to configure the <code>q</code> and <code>fq</code> parameters. If not specified, the default field is <code>text</code> .
<code>wt</code>	Response writer (https://solr.apache.org/guide/8_8/common-query-parameters.html#wt-parameter). Select the response format by specifying one of Solr's response writers (https://solr.apache.org/guide/8_8/response-writers.html).

Query examples

1. Search only the `title` field in the "docs" collection for the term "solr", and format the results as JSON:

```
http://localhost:{api-port}/api/query-pipelines/default/collections/docs/select?q=solr&fl=title&wt=json
```

2. Use the following query with wildcards and character escaping in the Query Workbench to search for both `BLU RAY MOVIES` and `CAMERA` classes in the `VIDEO/CD` and `PHOTO/CAMERA` departments:
 - o `class_s:*RA* AND department_s:*O\C*`
3. Use a movie collection to select the movie ID and rating where the rating is equal to 4.

```
http://FUSION_HOST:FUSION_PORT/api/query-pipelines/default/collections/docs/select?q=*&fl=movie_id,rating&fq=rating:4
```

4. Search for movies with "black" in the title, starting at the 20th result for pagination, then return the results as XML:

```
http://FUSION_HOST:FUSION_PORT/api/query-pipelines/default/collections/docs/select?q=black&fl=title&start=20&rows=20&wt=xml
```

Solr Reference Guide

See Solr Reference Guide (<https://solr.apache.org/guide/>) for all Solr guides.

Fusion release	Solr version
4.2.3 to 4.2.6	7.7.2 (https://solr.apache.org/guide/7_7/)
4.2.0 to 4.2.2	7.5.0 (https://solr.apache.org/guide/7_5/)
4.1.0 to 4.1.3	7.4.0 (https://solr.apache.org/guide/7_4/)
4.0.0 to 4.0.2	7.2.1 (https://solr.apache.org/guide/7_2/)

Spark Reference Guide

Fusion release	Spark version
4.2.1 to 4.2.6+	2.3.2 (https://spark.apache.org/docs/2.3.2/)
4.1.0 to 4.2.0	2.3.1 (https://spark.apache.org/docs/2.3.1/)
4.0.0 to 4.0.2	2.2.1 (https://spark.apache.org/docs/2.2.1/)

System Metrics

In version 4.1 and earlier, Fusion automatically creates the system collection `system_metrics`. It is empty until you manually enable `system_metrics` indexing; see below for instructions.

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.

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:8764/api/collections/lws5_metrics/stats", "queryParams" : { }, "userPrincipal" : null, "method" : "GET", "cookies" : { } }, { "requestUri" : "http://localhost:8764/api/collections/logs/stats", "queryParams" : { }, "userPrincipal" : null, "method" : "GET", "cookies" : { } }, { "requestUri" : "http://localhost:8764/api/collections/logs/stats", "queryParams" : { }, "userPrincipal" : null, "method" : "GET", "cookies" : { } }, { "requestUri" : "http://localhost:8764/api/collections/lws5_metrics/stats", "queryParams" : { }, "userPrincipal" : null, "method" : "GET", "cookies" : { } }, { "requestUri" : "http://localhost:8764/api/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" : 0.00333495, "p95" :
    0.0054104499999999992, "p98" : 0.0141957599999999965, "p99" : 0.024622300000000001,
    "p999" : 0.128502437000000002, "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, and so on.), and the mean rates over 1-, 5- and 15-minute intervals ('m1_rate', and so on.). 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', and so on.). 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" : 0.054440510000000004,
"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" : 0.481695500000000005, "p95" : 0.53017705, "p98" :
0.5617982399999999, "p99" : 0.628122180000000003, "p999" : 0.87108949700000004,
"stddev" : 0.2448979377578966, "m15_rate" : 0.02059326561557774, "m1_rate" :
0.03249432457272969, "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.

Time-based Partitioning

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 <code>timestampFieldName</code>	The name of the field from which to read timestamps. The default is "timestamp".
Partition Time Period <code>timePeriod</code>	The time range for each partition. The default is one day.
Max Active Partitions <code>maxActivePartitions</code>	The number of partitions to keep active.
Delete Expired Partitions <code>deleteExpired</code>	"True" to automatically delete partitions that fall outside of the <code>maxActivePartitions</code> window, at intervals of <code>scheduleIntervalMinutes</code> . The default is "false".
Preemptive Create Enabled <code>preemptiveCreateEnabled</code>	"True" (the default) to create partitions in advance.
Schedule Interval <code>scheduleIntervalMinutes</code>	The interval, in minutes, at which to perform background maintenance, including preemptively creating partitions (<code>preemptiveCreateEnabled</code>) and deleting expired partitions (<code>deleteExpired</code>). The default is five minutes.
Partition Num Shards <code>numShards</code>	The number of shards per partition. The default is the value configured for the main Fusion collection.
Partition Replication Factor <code>replicationFactor</code>	The number of copies to keep, per partition. The default is the value configured for the main Fusion collection.
Partition Config Name <code>configName</code>	The name of the Solr configuration set to be applied to new partitions; the default is the configuration used by the primary collection.

Examples

Create a new collection called "TimeSeries1":

```
curl -X PUT -H 'Content-type: application/json' -d '{
  "solrParams": {
    "numShards": 1,
    "replicationFactor": 1
  }
}' http://localhost:8764/api/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:8764/api/collections/TimeSeries1/features/partitionByTime
```

Verify that time-based partitioning is enabled:

```
curl -X GET
http://localhost:8764/api/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:8764/api/index-pipelines/TimeSeries1-  
default/collections/TimeSeries1/index
```

View the Solr configuration for this collection:

```
curl -X GET "http://localhost:8764/api/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,TimeS  
eries1_2016_02_24_00_15,TimeSeries1_2016_02_24_00_20</str>
```

Release Notes

Release notes provide detailed descriptions of the relevant changes included with each Fusion Server release, including new features, improvements, bug fixes, and more.

Upgrades

Keeping your Fusion deployment upgraded to the latest version enables you to take advantage of the latest features, improvements, bug fixes, and addressed security issues.

Process

To upgrade to the latest version of Fusion, see:

- **From Fusion Server 4.2.x**
 - *To Fusion Server 4.2.y*
 - *To latest SP*
- **From Fusion Server 4.1.x**
 - *To Fusion Server 4.2.y*
 - *To Fusion Server 4.1.y*
- **From Fusion Server 4.0.x**
 - *To Fusion Server 4.2.y*
 - *To Fusion Server 4.1.y*
 - *To Fusion Server 4.0.y*
- **From Fusion 3.1.x**
 - *To Fusion Server 4.2.y*
 - *To Fusion Server 4.1.y*
 - *To Fusion Server 4.0.y*

Deprecations and Removals

Every Fusion release adds or improves Fusion features and functionality. Occasionally, we also remove features and functionality. Before doing so, we “deprecate” them. Deprecated features are no longer officially supported by Lucidworks and will not be updated or maintained. In many cases, new features are introduced which replace and/or improve the functionality of the deprecated feature.

See *Deprecations and Removals* for a list of active deprecations and removals.

How long do I have before the feature is removed?

We give a minimum of one minor release cycle before removing a deprecated feature. For example:

- A feature is deprecated in Fusion 4.1.0, so it will not be removed before Fusion 4.2.0
- A feature is deprecated in Fusion 4.1.1, so it will not be removed before Fusion 4.2.0

Deprecations are typically made in major and minor releases only. Maintenance release deprecations are less common.

Fusion Server 4.1.3 Release Notes

Release date: 12 April 2019

Component versions:

Solr 7.4.0	ZooKeeper 3.4.13	Spark 2.3.1	Jetty 9.3.25.v20180904	Ignite 2.3.0

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

As of Fusion 4.1.1, the SSL configuration procedure has changed. See <i>SSL Security (Unix)</i> and <i>SSL Security (Windows)</i> for updated instructions.

New features

No new features were introduced in this release.

Improvements

- The Web connector and the SharePoint connector now support multiple start links using a new **Bulk Start Links**/`f.bulkStartLinks` property.
- Additional *Web VI connector* improvements:
 - The connector now supports HTTPS when configured for proxying in high-performance mode.
 - The connector now supports inclusive/exclusive regular expressions after a `?` in URL values.
 - New configuration properties:
 - **Maintain Cookies for No-Auth Crawls**/`f.maintainCookies`
If you are not using authentication, then by default cookies are not stored in between web requests (stateless). If checked, cookies will be maintained between requests during the web crawl even when you are not using authentication. If you are using authentication, this checkbox has no effect on the crawl and can be ignored.
 - **Obey link nofollow attributes**/`f.obeyLinkNoFollow`
If true, `rel='nofollow'` on links are obeyed.
 - **Obey robots meta tags and headers**/`f.obeyRobotsMeta`
If true, rules like `noindex`, `nofollow`, and others found in a robots meta tag on a page or in the headers of the HTTP response are obeyed.
- New Box.com connector configuration properties:
 - **API Connection Timeout (ms)**/`f.fs.connectTimeoutMs`
The Box API connection timeout in milliseconds.
 - **API Read Timeout (ms)**/`f.fs.readTimeoutMs`
The Box API read timeout in milliseconds.
- The Confluence connector has new security trimming configuration properties, **User group cache limit**/`f.cacheUserGroupLimit`. Only applicable when `cacheUserGroups` is enabled, this will limit the number of users who will have their groups cached. This is used for testing purposes only. The default of -1 will cause all users to be cached.
- Diagnostic mode is now available for the MongoDB connector.
- SharePoint and SharePoint Online connector improvements:
 - New configuration properties:
 - **Retry requests**/`f.enable_retry`
Enable this to make the connector automatically try again if a request to sharepoint fails unexpectedly.
 - **Number of retries**/`f.retry_attempts`
If **Retry requests** is enabled, this number determines how many times the connector should retry a request before giving up.
 - Addressed an issue with running the Sharepoint connector without an associated parser configuration.
 - The connectors now delete indexed documents after they are deleted from the SharePoint source.
- The HTML parser now parses documents whose `Mime-Type` value does not *exactly* match `text/html`, such as `text/html; charset=utf-8` and so on. This allows it to parse `.aspx` pages from SharePoint datasources, for example.

Other changes

- Fusion now redirects to the login page when you log out on Chrome 73.
- Migrated objects are now correctly named as `collectionName_objectName` instead of `objectName`.
- The Query Workbench now loads datasets that contain large numbers of fields.

- A display bug was fixed in the Object Explorer.
- The Confluence connector now updates document permissions when their space permissions have changed.
- Fixed an issue which cause a `Could not read triggers for job` error for some scheduled jobs.
- Fixed a problem where a job scheduled to run "After Another Job Completes" would fail to run when the job scheduler leadership changed.
- Fixed an issue with incremental ACL updates for changes with `_lw_confluence_anonymous` permission setting in the Confluence connector.

Fusion Server 4.1.2 Release Notes

Release date: 17 December 2018

Component versions:

Solr 7.4.0	ZooKeeper 3.4.13	Spark 2.3.1	Jetty 9.3.25.v20180904	Ignite 2.3.0

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

See also the *Fusion AI 4.1.2 release notes* and the *App Studio 4.1.2 release notes*.

As of Fusion 4.1.1, the SSL configuration procedure has changed. See *SSL Security (Unix)* and *SSL Security (Windows)* for updated instructions.

New features

No new features are introduced in this release.

Improvements

- Performance during security trimming operations is improved in this release.
- The *JavaScript evaluation* in addition to the high-performance mode.

Other changes

- *JDBC V1 connector datasources* no longer need to be cleared in order to update the value of the `convert_type` property.
- Spark jobs now run correctly when Solr is configured to use SSL.
- Single sign-on (SSO) now works correctly with Kerberos.
- The *selected with jsoup*.
- The *Windows Share SMB 2/3 connector* now uses Unix timestamp format for the `lastModified_dt` field instead of Windows format.
- The *Confluence connector* now accepts null values for the `f.includedSpaces` property.
- The *Box connector* now accepts a subfolder as `startLink`.
- The *Solr Partial Update Indexer pipeline stage* now correctly deletes field values in the index when they have been deleted in SharePoint or SharePoint Online.
- The *Solr connector* now respects the value of `solr_request_handler`/Solr request handler when its value has been modified from the default.
- Fixes for the *SharePoint Online* connectors:
 - The connectors now respect the value of `global_security_filter_cache`/Enable global security filter cache when its value has been modified from the default.
 - Special characters are now correctly escaped in URLs.

Known issues

The Tika parser and Tika stage **Return parsed content as XML** option works only if the input document is also HTML/XML.

Fusion Server 4.1.1 Release Notes

Release date: 7 November 2018

Component versions:

Solr 7.4.0	ZooKeeper 3.4.13	Spark 2.3.1	Jetty 9.3.25.v20180904	Ignite 2.3.0

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

See also the *Fusion AI 4.1.1 release notes*.

As of Fusion 4.1.1, the SSL configuration procedure has changed. See *SSL security for Unix* and *SSL security for Windows* for updated instructions.

New features

- You can now upload Javascript files to the blob store and use them in these new pipeline stages:
 - Managed JavaScript index stage*
 - Managed JavaScript query stage*

Improvements

- ZooKeeper has been upgraded from version 3.4.11 to 3.4.13.
See the ZooKeeper 3.4.13 release notes (<https://zookeeper.apache.org/doc/r3.4.13/releasenotes.html>).
- Jetty has been upgraded from version 9.3.8.v20160314 to 9.3.25.v20180904.
See the list of changes in 9.3.25.v20180904 (<https://github.com/eclipse/jetty.project/releases/tag/jetty-9.3.25.v20180904>).
- Two new *blob types* were added, to support the Managed Javascript *index stage* and *query stage*:
 - `file:js-index`
 - `file:js-query`
- Web V1 connector* improvements:
 - The connector can now send *custom headers* with a new `addedHeaders` property.
 - When the Web connector crawls a Web site and Server Name Indication (SNI) is enabled but the Web site does not support it, you may receive an `unrecognized name` error. A new property, `useIpAddressForSslConnections` configures the connector to use the IP address instead of the hostname.
 - Kerberos is now supported* with these new properties:
 - `kerberosEnabled`. Boolean; default false. If true, the connector attempts to perform Kerberos/Spnego authentication when a Web request returns a `401 WWW-Negotiate` challenge. If false, the Web connector refuses to attempt Kerberos/Spnego authentication.
 - `kerberosPrincipalName`. Optional - Use this principal name as the logged-in Kerberos user instead of the environment's default. If set, you must also specify a keytab with one either `kerberosKeytabFilePath` or `kerberosKeytabBase64` (see below).
 - `kerberosKeytabFile`. Optional - The path to the Kerberos keytab file that contains the credentials.
 - `kerberosKeytabBase64`. Optional - A base64-encoded Kerberos keytab file that contains the credentials.

See the *main topic* for additional configuration details.

- The *ServiceNow connector* now supports proxying.
- The *JDBC V1 Connector* has a new `convert_type` parameter for binary streaming data from a SQL Server
By default, a JDBC column will automatically use the Solr field type that matches the underlying database's type of the column. If checked, this will use the field name of the column to choose the Solr field type.
- When you import objects using the *Objects API*, a new `context` parameter specifies the name of an existing app which will be the new context for the imported objects. To support this, the *Links API* has two new `linkType` values:
 - `inContextOf`
 - `hasContext`
- Performance improvements in proxy.
The proxy service now reuses HTTP client instances for better throughput. Customers with high QPS expectations (~ 1000 QPS) are strongly encouraged to upgrade to 4.1.1.

Other changes

- The Fusion SQL Service (`bin/sql`) has been added to the default group in `conf/fusion.properties`. If you do not intend to run the SQL service for self-service analytics, you can remove it from the default group. See *Fusion SQL service* for more information on using the SQL service.
- When exporting and importing objects with secret keys, the format is now `secret.{object_type}.{object_id}.{number}.password`. In the case of data source objects, the format is now `secret.{object_type}.{object_id}.{datasource_type}.{number}.password`.
- The *Index Profiles API* no longer sets `bufferDocsForSolr` by default.
- The *Query Workbench* now correctly displays all metadata fields.
- JavaScript is now thread-safe in pipeline stages.
This resolves an issue in previous releases where variables in Javascript stages that were not declared with `var` were shared between threads and other stages using variables with the same name.
- The *Configure Fusion for JWT* now works with multiple JWTs.
- Headers missing from `request.headers` in the *trusted-http* security realm are available again. This includes:
 - `fusion-user-realm-name`
 - `fusion-user-realm-type`
 - `fusion-user-name`
 - `fusion-user-permissions` now has the role restriction assigned with the security realm.

Known issues

- The Tika parser and Tika stage **Return parsed content as XML** option works only if the input document is also HTML/XML.
- Vulnerability scanners are reporting false positives for security vulnerabilities.
- Fusion may not redirect to the login page for users of Google Chrome release version 73.0.3683.75.
- When exporting configuration data during a Fusion upgrade, the migrator may fail to log the complete process.
- Fusion UI may not display the complete job history, despite the history being complete in the jobs endpoint.
- When using *High Performance JavaScript Evaluation* mode in Google Chrome, the proxy must be used only for HTTP traffic, not HTTPS.
- In Red Hat Enterprise Linux (RHEL) 7.6, running the `install-high-perf-web-deps.sh` script fails to install Google Chrome.
- Fusion Proxy service's SAML library does not fully support Windows.
- The `responseHeaders_ss` field may be missing from PDF documents acquired by a web crawl search.
- In Fusion UI, the job history status icons may not accurately show the job status.
- Running `count(distinct query)` in the Fusion SQL service results in an inaccurate number of results. This can be mitigated by using a query similar to `select count(1) from (select distinct query from table) a`.
- When using ZooKeeper Import/Export API to export an app, the process may fail if the app contains large *blobs*.
- Using *JavaScript dedupe* may result in the creation of a ScriptRunner with a large default cache size. This may cause the connectors to run out of heap space during large crawls.
- Attempting to use JSONPath `$.id` in the JSON Parser to parse all document IDs only processes the first document in the list.
- Some Fusion SQL queries do not work with time-based partitions of collections.
- Clicking "Ok" while Fusion UI is in the process of deleting an app will cause a **503 Service Unavailable** error.
- In Fusion UI, deleting an app with a datasource and creating a new app with the same name may result in **404 Not Found** and **500 Server Error** errors when uploading a datasource. Reloading the page prior to uploading the datasource will resolve this issue.
- Updating a datasource may result in the loss of unsaved changes to pipeline or parser stages.
- User-created index pipeline or parser fields are not included in the Index Profile search's autocomplete function.
- When deleting a collection, selecting the option "Switch to another collection", not making a choice, and re-selecting "Or return to the launcher to pick a different app" results in the "Proceed to Delete Collection" button being disabled.
- In Fusion UI's Collection Manager, unchecking "Enable Search Logs" in **Manage All Collections > Collection Features** before saving results in a **500 Server Error** error.

- In Fusion UI's Collection Manager, unchecking "Enable Signals" in **Manage All Collections > Collection Features** when creating a collection does not result in signals being disabled.
- Starting and stopping Fusion services while in the Fusion UI may result in the components and fields being inaccessible.
- Setting a parser's `maxParserRecursionDepth` field to a negative value will not result in an error until attempting to implement it in the Index Workbench.
- The SQL service will not initialize by default unless the API is located on the same host.
- Indexing an Oracle JDBC datasource may fail due to an inability to update `ConvertType`.
- The *Hadoop Authentication Plugin* `hadoop-auth-3.1.1.jar` is incompatible with Jetty `9.4.12.v20180830.jar`. Reverting to `hadoop-auth-2.7.1.jar` inside `ROOT.war` fixes the issue.
- Users in a JSON Web Token (JWT) realm in Fusion may fail to authenticate. Use of a base64 encoded token may fix this issue.
- Stopping Fusion with a `TERM` signal fails to stop all Fusion services.
- Deleting folders from the `fusion/var/log/` directory causes Fusion services to fail during start.
- Creating an app with the same name as a previously deleted app may result in the error "Datasource ID does not exist".
- Fusion UI's Index Workbench may fail to display the correct job status until the page is reloaded.
- When performing a *Head/Tail Analysis Job*, the user may receive a `Job execution error: Error during job execution: empty collection` error, even if there are documents in the app collections.
- Some connectors are not visible in the datasource panel until after they have been installed.
- In Windows, SDK connectors may time out during the first run. Subsequent runs are successful.
- Blob types `file:js-index` and `file:js-query` cannot be added in Fusion UI but can be added with the API.
- If an incorrectly configured stage is disabled, results will not generate despite active stages being valid.
- No error message is produced during a signal aggregation job if the field name value being searched is inaccurate or non-existent.

Fusion Server 4.1.0 Release Notes

Release date: 17 July 2018

Component versions:

Solr 7.4.0	ZooKeeper 3.4.11	Spark 2.3.1	Jetty 9.3.8.v20160314	Ignite 2.3.0

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

New features

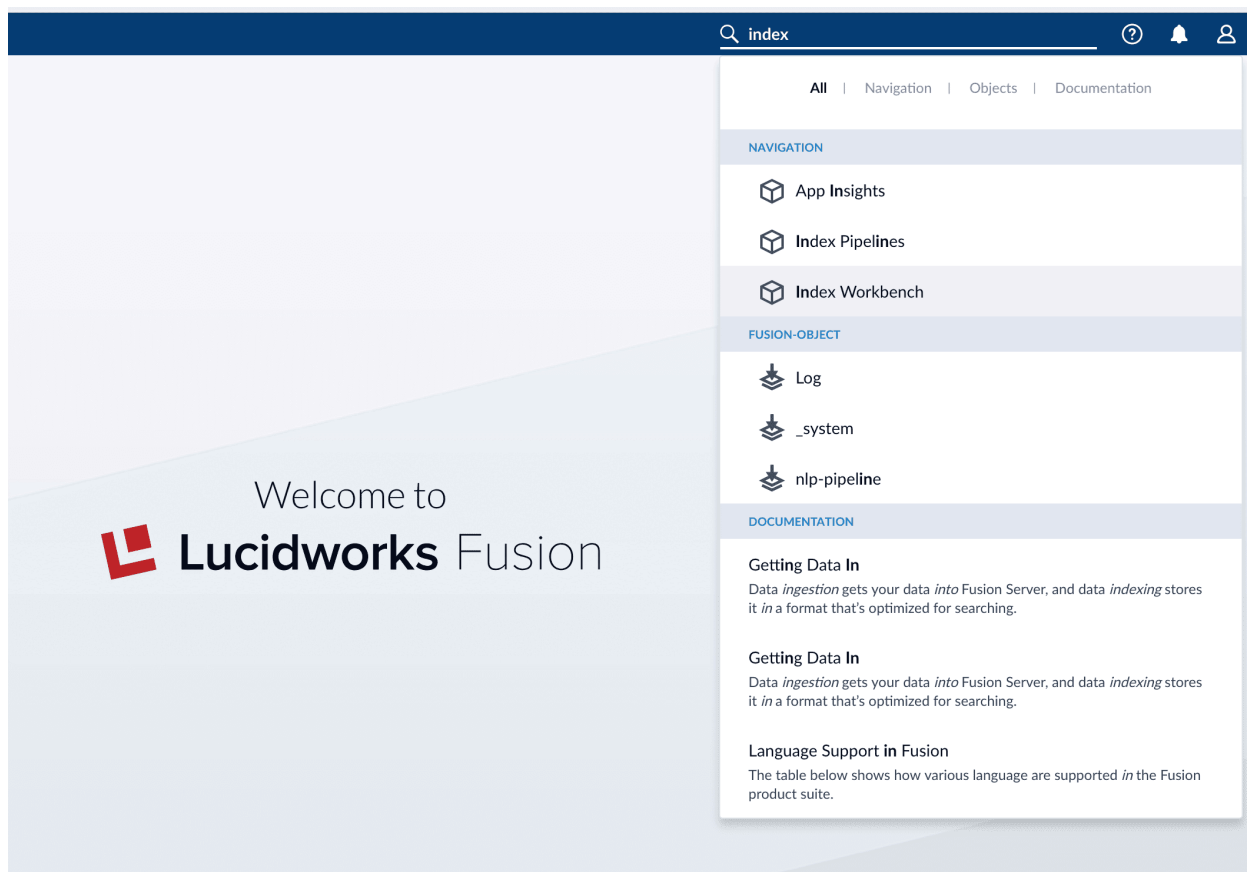
- **App Studio integration**

Fusion App Studio is now integrated into the Fusion Server UI for seamless search UI development. You can open App Studio from the Fusion workspace by navigating to **App Studio > Build new UI**.

The screenshot displays the Lucidworks Fusion Server UI. The top navigation bar includes the Lucidworks logo and a 'preview' button. The main content area shows a search results page with a sidebar listing categories like 'VODKA 80 PROOF', 'CANADIAN WHISKIES', etc. The results are displayed in a grid of product cards, including 'Seagrams V.o. Bl Canadian Whisky', 'Pinnacle Gin', 'Christian Bros Brandy', 'Jack Daniel's Tennessee Fire Mini', 'Dewars White Label Scotch', 'Cruzan Coconut Rum', and 'Hennessy VS'. Below the search results, the App Studio code editor is open, showing the 'search.html' file. The code includes comments and XML-like tags for defining a search platform, query, and response, along with a spinner widget.

- **Fusion Server UI search/autocomplete**

In the Fusion workspace, you can now search objects and documentation, with autocomplete to suggest relevant results.



- **New OneDrive connector**
- **New Download Blob job**
- **New log shipper and log console**

Improvements

- **Connector improvements**
 - The Confluence connector now indexes labels for spaces and pages.
 - The Web connector can be configured to deduplicate pages whose content is identical.
- **UI improvements**
 - The Object Explorer now supports filtering by object type.
 - Objects in the UI now include an "Explore" button that opens the Object Explorer view for that object.
 - UI panels can now be resized horizontally.
 - Text input areas can now be opened as pop-out modals for easier editing.
- **Connectors SDK**
- **Import/export improvements**
 - Additional parameter substitutions are now supported on object import.

Other changes

- Better heuristics for the `splits_per_shard` field in the *Catalog API*.
- Fusion's SparkSQL engine is no longer a beta release.
- Ports are now preconfigured for Spark multinode communication, using these parameters in the *Configurations API*:
 - `spark.driver.port`
 - `spark.executor.port`
 - `spark.blockManager.port`
 - `spark.fileserver.port`
- A request ID can now be defined in order to add context to logs.
- The Jobs API now provides better performance.
- The *system collections* named `audit_logs` and `logs` have been replaced and extended with a new system collection, `system_logs`.

Known issues

- Some headers are missing from `request.headers` in the *trusted-http* security realm, including the following:
 - `fusion-user-realm-name`
 - `fusion-user-realm-type`
 - `fusion-user-name`
 - `fusion-user-permissions` lacks the role restriction assigned with the security realm.
- The Tika parser and Tika stage **Return parsed content as XML** option works only if the input document is also HTML/XML.
- Users in a JSON Web Token (JWT) realm in Fusion may fail to authenticate. Use of a base64 encoded token may fix this issue.

Fusion Server 4.0.2 Release Notes

Release date: 22 May 2018

Component versions:

Solr 7.2.1	ZooKeeper 3.4.10	Spark 2.2.1	Jetty 3.8.10	Ignite 2.3.0

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

New features

- A new *Windows Share connector* provides support for SMB 2 and 3 (<https://msdn.microsoft.com/en-us/library/cc246482.aspx>).
The previous *Windows Share Connector* is deprecated.

Improvements

- Additional options for the *Zendesk connector*:
 - article_containers
 - enable_security_trimming
 - index_article_attachments
 - index_articles
 - index_tickets
 - page_size
 - zendesk_account_plan
- Web V1 connector* improvements
 - Instead of JWebDriver and Mozilla Firefox, the Web connector now uses PhantomJS and Google Chromium for JavaScript evaluation. When `f.crawlJS/"Evaluate JavaScript"` is enabled, then Fusion uses PhantomJS to evaluate JavaScript content. Lucidworks recommends also enabling `f.useHighPerfJsEval/"High Performance Mode"`, in which case Chromium is used, providing the fastest possible JavaScript evaluation.
This feature requires downloading a separate setup script. See the *Web V1 connector* topic for instructions.
 - Support for screenshots
When `f.crawlJS/"Evaluate JavaScript"` and `f.useHighPerfJsEval/"High Performance Mode"` are enabled, then `f.takeScreenshot/"Index a screenshot of rendered page"` can also be enabled.
When this feature is enabled, Fusion takes a screenshot of each fully-rendered page and indexes it in a field called `screenshot_bin`.
Your schema must specify this field as a binary field or indexing will fail. See the *Web V1 connector* topic for instructions.

New options for the Web connector:

- f.useHighPerfJsEval
- f.headlessBrowser
- f.takeScreenshot
- f.screenshotFullscreen
- f.viewportWidth
- f.viewportHeight
- f.deviceScreenFactor
- f.simulateMobile
- f.mobileScreenWidth
- f.mobileScreenHeight
- f.chromeBinaryPath
- f.quitTimeoutMs
- f.useRequestCounter
- f.requestCounterMinWaitMs
- f.requestCounterMaxWaitMs

The following configuration keys are deprecated and will be removed in a future release:

- f.useFirefox
- f.firefoxHeadlessBrowser
- f.firefoxBinaryPath

- The Jira connector now supports security trimming when single sign-on (SSO) is used to authenticate to the Jira servers.
- The *Jobs API* now performs faster.

Other changes

- This release fixes a bug in the authentication proxy which incorrectly redirected certain requests when SSL was enabled.
- Spark job IDs are now transformed so that invalid characters are replaced with underscores (`_`). Valid characters are `A-Z`, `a-z`, `0-9`, underscore (`_`), backslash (`\`), and hyphen (`-`).
- The Query Profiles and end points from earlier versions of Fusion have been replaced by new endpoints and profiles with additional functionality. Application/UI's submitting requests to Fusion will need to be updated to use the new endpoints.

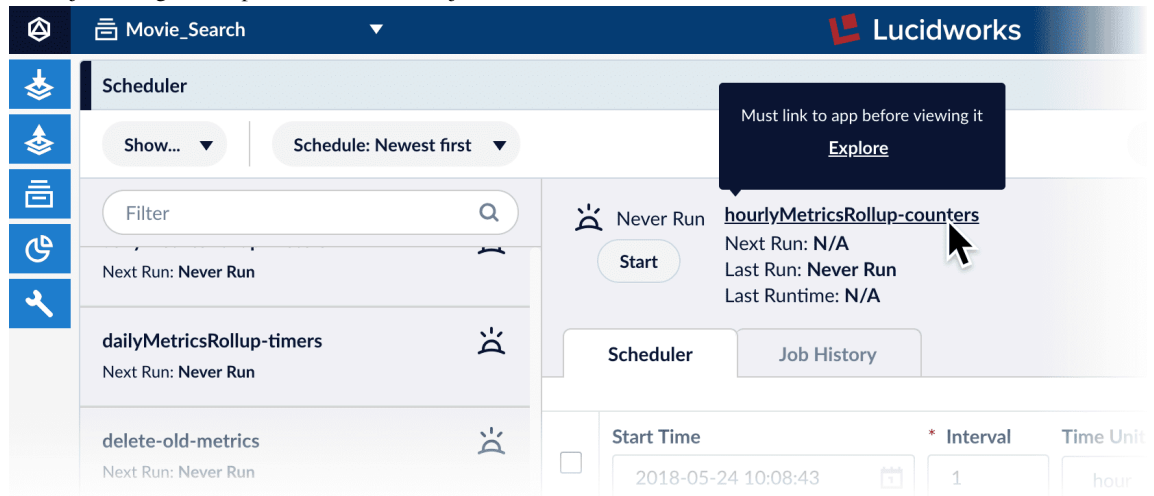
Known issues

- In the *Query Workbench*, the **Add a facet field** dropdown does not always list all potential facet fields.
Workaround:
Use the *Field Facet query pipeline stage* to add the desired facet field to the pipeline.
- App Insights shows "No data available" after logging out and in again.
After exiting App Insights, logging out of Fusion, then logging in to Fusion and returning to App Insights, you may see a "No data available" message.
Workaround:
Log out of Fusion, clear your browser's cookies, then log in to Fusion again.
- In system metrics hourly jobs, the value of the "Type field" property defaults to `type`, which does not exist as a field in the system metrics collection. The correct value is `type_s`.
The following jobs are affected by this issue:
 - hourlyMetricsRollup-timers
 - hourlyMetricsRollup-meters
 - hourlyMetricsRollup-histograms
 - hourlyMetricsRollup-gauges
 - hourlyMetricsRollup-counters
 - dailyMetricsRollup-timers
 - dailyMetricsRollup-meters
 - dailyMetricsRollup-histograms
 - dailyMetricsRollup-gauges

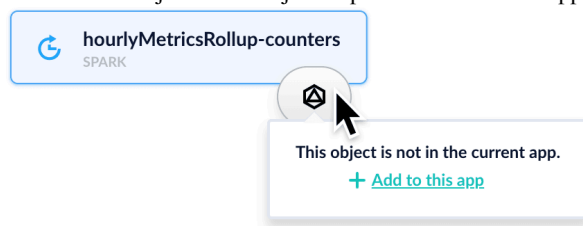
Workaround:

Manually update the jobs to change the Type Field property to "type_s". This is a one-time change for the cluster.

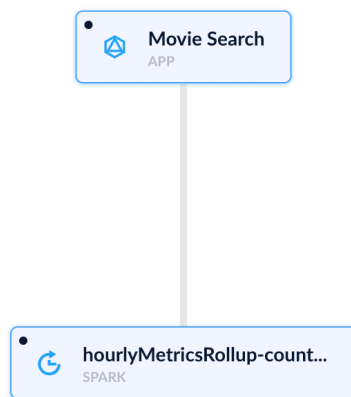
1. Create a "System" app, or use any existing app in Fusion.
2. Navigate to **System > Scheduler**.
3. For each of the above jobs in the schedules, link the job to the current app:
 - a. Select the job in the jobs list.
 - b. In the job configuration panel, hover over the job name.



- c. Click **Explore**.
The Object Explorer appears.
- d. Hover over the job in the Object Explorer and click the App icon:



- e. Click **Add to this app**.
The Object Explorer displays the job linked to the app:



- f. Close the Object Explorer.
4. Navigate to **Collections > Jobs**.
5. For each of the jobs listed above:
 - a. Click **Advanced**.
 - b. Change the value of the **Type Field** property from `type` to `type_s`.
 - c. Save the job configuration.

The system metrics jobs should now run correctly.

- Blobs of type "other" cannot be uploaded.
Fusion 4.0.2 rejects blob uploads when the blob type "other" is specified.
The Fusion UI gives this error:


```
API ERROR Code: BLOB-0006 Status: 400 Bad Requestvalue of [other] for parameter [resourceType] is not supported as a blob type
```

The API gives this error:

```
{"type":"RESError","statusCode":400,"error":"BLOB-0006","message":"Is a FusionResourceType but not a blob type","details":"value of [other] for parameter [resourceType] is not supported as a blob type","cause":null,"httpStatusMessage":"Bad Request","localizedMessage":"value of [other] for parameter [resourceType] is not supported as a blob type","suppressed":[],"class":null}
```

Workaround:

Select a different *blob type*, such as "file-upload".

- A user with search-only permissions receives an "invalid license" error when attempting to log in to the Fusion UI.
A user with search-only permissions (including users assigned to the built-in "search" role) cannot log in to the Fusion UI. Developer or admin permissions are required in order to log in to the UI.
If a user with search-only permissions encounters this error, there is no need to update your license.
- In Internet Explorer, pipeline stage configuration panels may appear empty.
After adding a new stage to a new pipeline in the Index Pipelines or Query Pipelines windows, clicking **Save** may result in an empty configuration panel for the new pipeline stage.

Workaround:

Refresh the page or click the name of the new stage in the list of pipeline stages. The Fusion UI should then display the stage configuration panel correctly.

- When creating a new collection after deleting the only remaining collection, the collection creation modal breaks after entering an invalid collection name.
When you delete the only user-created collection in an app, the UI prompts you to replace it by creating a new collection. If you enter a collection name that contains invalid characters and click "Proceed to Delete Collection", the button becomes disabled.

Workaround:

Close the modal and try again.

- Some headers are missing from `request.headers` in the *trusted-http* security realm, including the following:
 - `fusion-user-realm-name`
 - `fusion-user-realm-type`
 - `fusion-user-name`
 - `fusion-user-permissions` lacks the role restriction assigned with the security realm.
- The Tika parser and Tika stage **Return parsed content as XML** option works only if the input document is also HTML/XML.

Fusion Server 4.0.1 Release Notes

Release date: 28 February 2018

Component versions:

Solr 7.2.1	ZooKeeper 3.4.10	Spark 2.2.1	Jetty 3.8.10	Ignite 2.3.0

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

See also the *Fusion AI 4.0.1 release notes*.

New features

- **Import and export for index profiles and query profiles**

App import now includes *index profiles* and *query profiles*. These can also be exported and imported separately with the *Objects API*, using the `index-profile` and `query-profile` object types.

Improvements

None.

Other changes

- The Query Profiles and end points from earlier versions of Fusion have been replaced by new endpoints and profiles with additional functionality. Application/UI's submitting requests to Fusion will need to be updated to use the new endpoints.

Known issues

- After SSL is enabled on Fusion Server 4.0.1, the server incorrectly redirects certain requests, including requests for `https://fusion-server:port/` and `https://fusion-server:port/admin`, resulting in a failure to reach the login page. To log in to the Admin UI after enabling SSL, use the direct URL to the login page as a workaround: `https://fusion-server:port/admin/login`. Fusion Server 4.0.2 resolves this issue.
- Some headers are missing from `request.headers` in the *trusted-http* security realm, including the following:
 - `fusion-user-realm-name`
 - `fusion-user-realm-type`
 - `fusion-user-name`
 - `fusion-user-permissions` lacks the role restriction assigned with the security realm.
- The Tika parser and Tika stage **Return parsed content as XML** option works only if the input document is also HTML/XML.

Fusion Server 4.0.0 Release Notes

Release date: 21 February 2018

Component versions:

Solr 7.2.1	ZooKeeper 3.4.10	Spark 2.2.1	Jetty 3.8.10	Ignite 2.3.0

More information about support dates can be found at *Lucidworks Fusion Product Lifecycle*.

New features

- **Fusion AI**

Fusion's artificial intelligence and machine learning features are now licensed separately, as the new *Fusion AI* product. These features are now enabled only with a Fusion AI license:

- Signals
- App Insights (new in 4.0)
- Smart jobs
 - ALS Recommender Jobs
 - Bisecting KMeans Clustering Jobs
 - Cluster Labeling Jobs
 - Co-Occurrence Similarity Jobs
 - Collection Analysis Jobs
 - Document Clustering Jobs
 - Ground Truth Jobs
 - Head/Tail Analysis Jobs (new in 4.0)
 - Item Similarity Recommender Jobs
 - Levenshtein Spell Checking Jobs
 - Logistic Regression Classifier Training Jobs
 - Matrix Decomposition-Based Query-Query Similarity Jobs
 - Outlier Detection Jobs
 - Random Forest Classifier Training Jobs
 - Ranking Metrics Jobs
 - SQL-Based Experiment Metric Jobs
 - Statistically Interesting Phrases Jobs
 - Token and Phrase Spell Correction Jobs (new in 4.0)
- REST APIs
 - Experiments API
 - Recommendations API
 - Signals Aggregator API
 - Signals API
- Index pipeline stages
 - Machine Learning index stage
 - OpenNLP NER Extractor index stage
 - Signal Formatter index stage
 - Update Experiment index stage
- Query pipeline stages
 - Advanced Boosting query stage
 - Analytics Catalog query stage
 - Experiment query stage
 - Machine Learning query stage
 - More Like This query stage
 - Recommend Items for Item query stage
 - Recommend Items for User query stage
 - Recommendation Boosting query stage

- **Apps**

Fusion apps provide tailored search functionality to specific groups of users.

An app is a named set of linked *objects*, including collections, datasources, index and query pipelines, index and query profiles, parsers, and more. Using roles and security realms, you can define security on a per-app basis.

- o A new *Apps API* is available for managing apps.
 - o The Launcher page now contains apps instead of workflow contexts. Workflows are the same for all apps.
 - o Apps are objects that can be imported and exported using the *Objects API*, and their links can be managed using the *Links API*.
 - o App objects and their links can be viewed using the *Object Explorer*.
- **License management**
Fusion now requires a license. A free 30-day trial license is included in the download. You can now *manage your Fusion license* using the Fusion UI or REST API. For information about licensing for upgrades from 3.x to 4.0, contact Lucidworks.
- **Connectors SDK**
Fusion now comes with a Java kit for developing custom connectors. See *Developing Custom Connectors*.
- **Confluence connector**
This new connector retrieves data from the Atlassian Confluence Wiki CMS. You can configure this datasource to crawl pages, spaces, blog posts, comments, and attachments. See the *Confluence Connector* reference topic.
- **Webapps API**
This API lays the groundwork for a future release in which Fusion App Studio can interface directly with Fusion Server to customize the front-end interfaces for your apps. Though this API is not usable in this release, you can preview it in the *Webapps API* reference topic.
- **New index pipeline stages**
 - o The *GeoIP Lookup index stage* looks up the geographical location of an IP address in your data.
 - o The *Update Related Document* stage modifies indexed documents, using fields from a separate datasource.

Improvements

- **Connector improvements**
 - o Sharepoint connector:
 - Performance improvements
 - Security trimming: support for groups, domain/username
 - XML parsing updates
 - Active Directory Federation Services (ADFS) support
 - o Jive connector:
 - hierarchical crawl support from root Jive space
 - Security trimming: support for objects that inherit ACLs
 - Update permissions on recrawl for Spaces
 - indexing of events, videos, and ideas
 - o Jira connector:
 - Attachments and work logs now support issue reporter inheritance
 - o Crawl stability for Box and Google Drive
 - o Web connector: headless Firefox crawling for JS
- **New APIs for profiles**
Index profiles and query profiles are now created with definitions that include the collection, pipeline, and handler. As such, the endpoints for profiles are no longer located under the `/collections` endpoint; this change supports experiment management in Fusion AI. New top-level endpoints have been added in this release.
 - o The endpoint at `/collections/COLLECTION_NAME/query-profiles` has been replaced by two new endpoints:
 - *Query Profiles API* (`/query-profiles`)
Create, read, update, and delete query profiles.
 - *Query API* (`/query`)
Run a query through a query profile by specifying the profile ID and appending the request with a *Solr query string*, as in `/api/apollo/query/<id>?<solrQuery>`.
 - o Similarly, the endpoint at `/collections/COLLECTION_NAME/index-profiles` has been replaced by two new endpoints:
 - *Index Profiles API* (`/index-profiles`)
Create, read, update, and delete index profiles.
 - *Index API* (`/index`)
Index data through an index profile by specifying the profile ID.
- **UI redesign**
The Fusion UI has a new look and feel, and menus have been redesigned. See *Fusion Server UI* for a brief tour.
- **Objects API improvements**

The *Objects API* now outputs a `.zip` file for every export, and supports the new `app` object type. Another new object type, `appkit-app`, is for use in future releases.

- **Solr cluster management**

- **SQL for aggregations**

By default, aggregation jobs are now defined using Spark SQL. To use the old aggregation configuration scheme in the Fusion UI, click **Advanced** in the Aggregation job configuration panel.

- **Multiple Hadoop connectors**

- **Blob Store API changes**

- A new `/blobs/convert` endpoint migrates blob manifests from 3.x to 4.x format.
- The `/blobs` endpoint has new parameters to support querying specific directories in the blob store: `dir`, `recurse`, and `showdirs`.
- When uploading a blob via the `/blobs/{id}` endpoint, new parameters are supported: `fusion-blob-type`, `commit`, and `update`.
- The `/blobs/{id}` endpoint now supports a `HEAD` method to retrieve blob metadata, equivalent to the `/blobs/{id}/manifest` endpoint.
- The `/blobs/{id}/manifest` endpoint has been removed.
- Fusion no longer supports uploading blobs via forms.

See the *Blob Store API* reference topic for details.

- **New framework for connectors**

A new connectors framework allows connectors to be hosted or remote:

- Hosted connectors are cluster-aware, so that cluster nodes are notified when connector services start up on other nodes. This allows easy scaling of the crawl process.
- Remote connectors act as lightweight Fusion clients which can be located wherever your data is stored.

See *Connectors* for more details.

In the 4.0.0 release, the Lucidworks-provided connectors follow the classic framework. The new framework will be applied to Lucidworks-provided connectors in future releases. You can use the new *Java Connector Development* to develop custom connectors using the new framework.

Other changes

- Solr 4.x is no longer supported. This release of Fusion Server requires Solr 6.x or later.
- The REST API endpoints for production deployments now use the path `/api/` and not `/api/apollo/`. The old path remains available but deprecated.
- A new `webapps-role` role and `webapps-system-account` realm were added to support apps.
- The `COLLECTION_NAME_logs` and `COLLECTION_NAME_signals` collections are now combined into a single `COLLECTION_NAME_signals` collection.

The `_signals` collection schema has changed in this release. If you have existing signals data, you must use the migrator script to convert it correctly. The migrator script will be provided following this 4.0 release. See *Signals* for details about the new schema.

- The Dropbox connector is no longer available.
- The Synonyms API has been removed.
- Anonymous metrics submission
- Aggregator API is deprecated
The `/aggregator` endpoints are deprecated in this release. Aggregations are now managed through the `/spark` endpoints.
- The Experiments API is deprecated.
Experiments are now managed through the `/spark` endpoints.
- The Query Profiles and end points from earlier versions of Fusion have been replaced by new endpoints and profiles with additional functionality. Application/UI's submitting requests to Fusion will need to be updated to use the new endpoints.

Known issues

- After SSL is enabled on Fusion Server 4.0.1, the server incorrectly redirects certain requests, including requests for `https://fusion-server:port/` and `https://fusion-server:port/admin`, resulting in a failure to reach the login page. To log in to the Admin UI after enabling SSL, use the direct URL to the login page as a workaround: `https://fusion-server:port/admin/login`.
Fusion Server 4.0.2 resolves this issue.

- Some headers are missing from `request.headers` in the *trusted-http* security realm, including the following:
 - `fusion-user-realm-name`
 - `fusion-user-realm-type`
 - `fusion-user-name`
 - `fusion-user-permissions` lacks the role restriction assigned with the security realm.
- The Tika parser and Tika stage **Return parsed content as XML** option works only if the input document is also HTML/XML.

Deprecations and Removals

Active Deprecations

The expected removal versions listed in the **Active Deprecations** table below may not correspond to a planned release version. Removals take place *on or after* the expected removal version. If the expected removal is listed as 4.1.4, but Fusion 4.1.4 is not a planned release, the removal may take place in Fusion 4.2.0.

Fusion Connectors deprecations and removals Deprecation and removal details for Fusion Connectors are found at <i>Fusion Connectors Deprecations and Removals</i> .			
Feature	Deprecated	Expected Removal	Notes
Webapps API	4.2.0	4.3.0	Webapps Appkit API
Experiments API	4.0.0	4.1.0	Experiments are now managed through the <code>/spark</code> endpoints.
Aggregator API	4.0.0	4.1.0	Aggregations are now managed through the <code>/spark</code> endpoints.

Removals

Feature	Removed	Notes
<code>/blobs/{id}/manifest</code> endpoint	4.0.0	-
Synonyms API	4.0.0	-