gNMIc

An intuitive gNMI CLI and

a feature-rich telemetry collector

Karim Radhouani

@karimra

From Monitoring to Telemetry

Network size

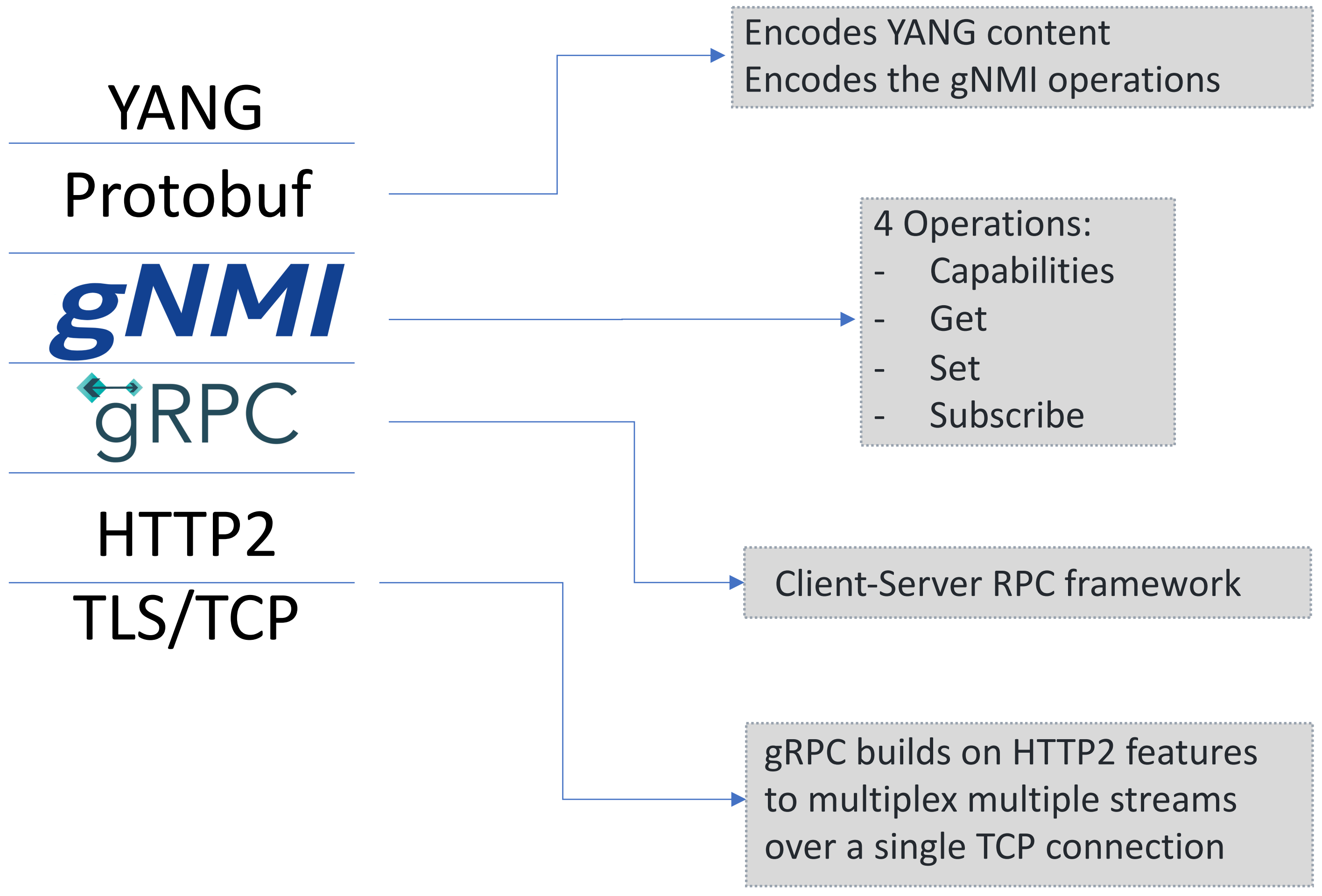
High data definition

CPU-friendly Push-based mode

Reliable delivery

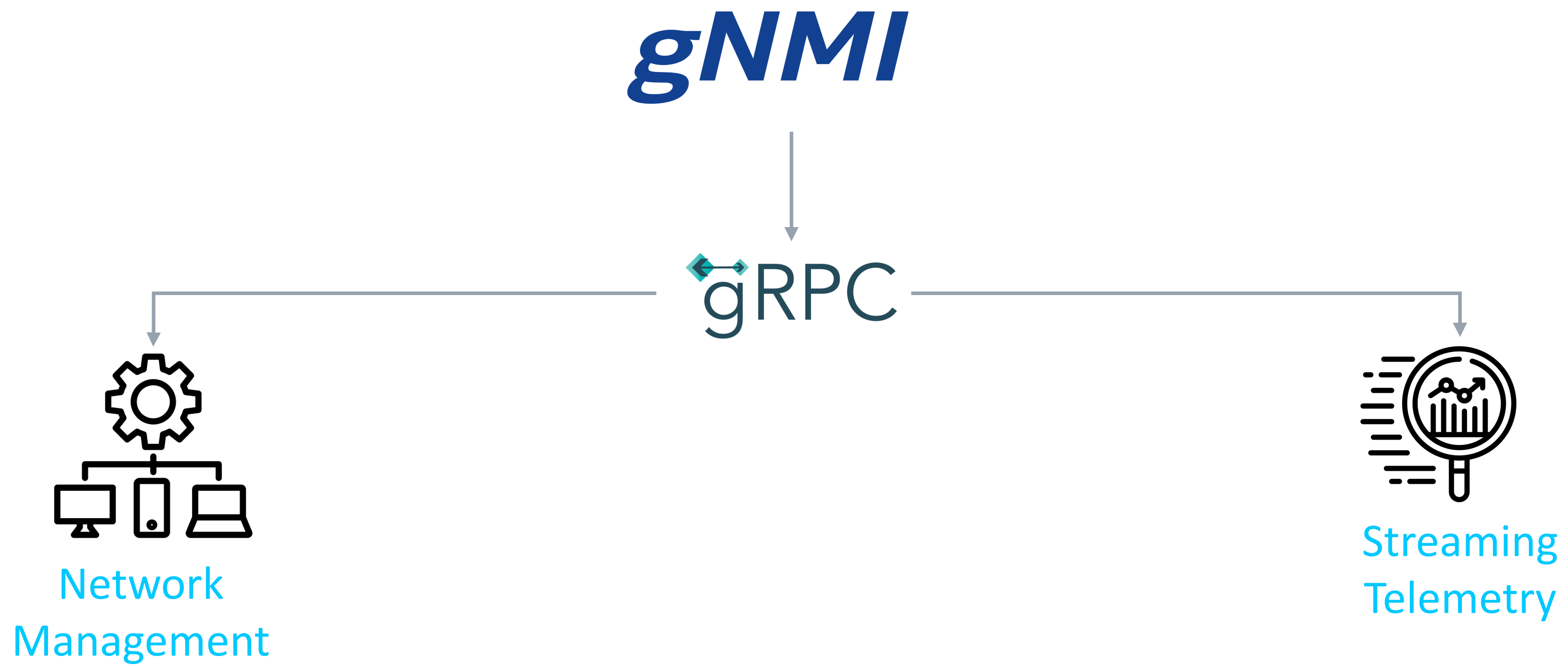
gNMIc

gRPC Network Management Interface


YANG
Protobuf
gNMI
gRPC
HTTP2
TLS/TCP
Encodes YANG content
Encodes the gNMI operations
4 Operations:
- Capabilities
- Get
- Set
- Subscribe
Client-Server RPC framework
gRPC builds on HTTP2 features to multiplex multiple streams over a single TCP connection

gNMIc

gRPC Network Management Interface

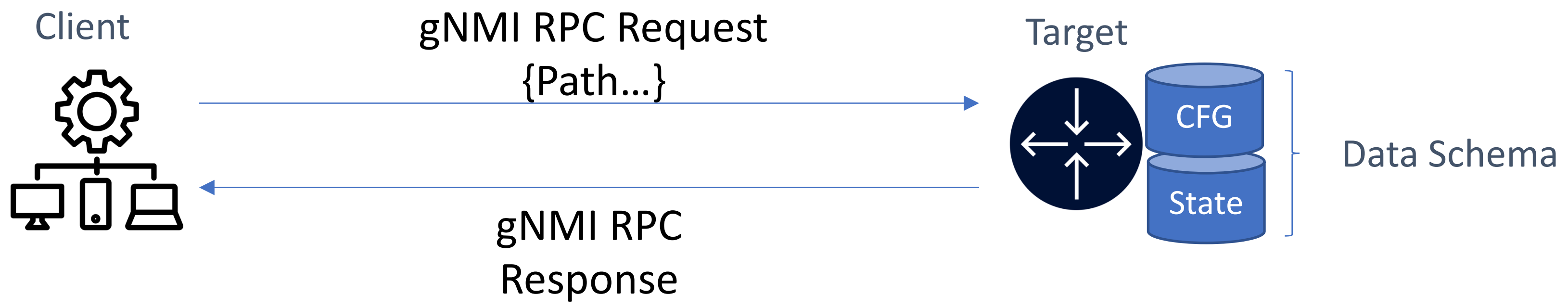


The intention is that a single gRPC service definition can cover both configuration and telemetry

gNMIc

gNMI Terminology

- **Target**: A device that owns the data that is queried or manipulated, i.e the network device
- **Client**: A system using the gNMI protocol to query or modify the data on a the target
- **Data Schema**: An instance of a YANG data model
- **Configuration**: Elements of the schema that the client can query or modify
- **Telemetry**: Streaming data from the target to the client, the data describes the target configuration or its operational state
- **Path**: An ordered list of elements that reference an object in the data schema



gNMIc

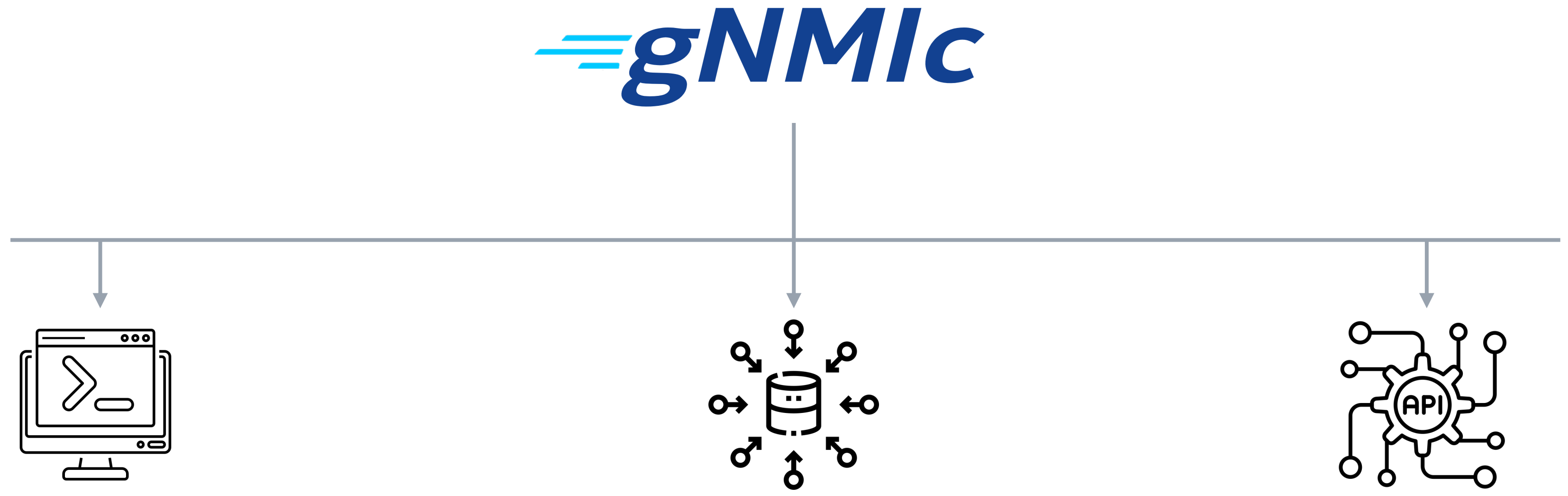
gNMI Service definition

- **Capabilities**: The client requests the target’s capabilities (supported YANG models and their revision date)
- **Get**: The client retrieves a snapshot of the data identified by a path from the target
- **Set**: The client modifies the configuration of the target
- **Subscribe**: The client can subscribe to data identifies a set of paths, the target will stream back the data to the client periodically or when it changes.

```
service gNMI {
  rpc Capabilities(CapabilityRequest) returns (CapabilityResponse);
  rpc Get(GetRequest) returns (GetResponse);
  rpc Set(SetRequest) returns (SetResponse);
  rpc Subscribe(stream SubscribeRequest) returns (stream SubscribeResponse);
}
```

gNMIc

What is gNMIc to gNMI?



Command Line Interface for gNMI

Highly Available and flexible collector

Go API for gNMI with a human touch

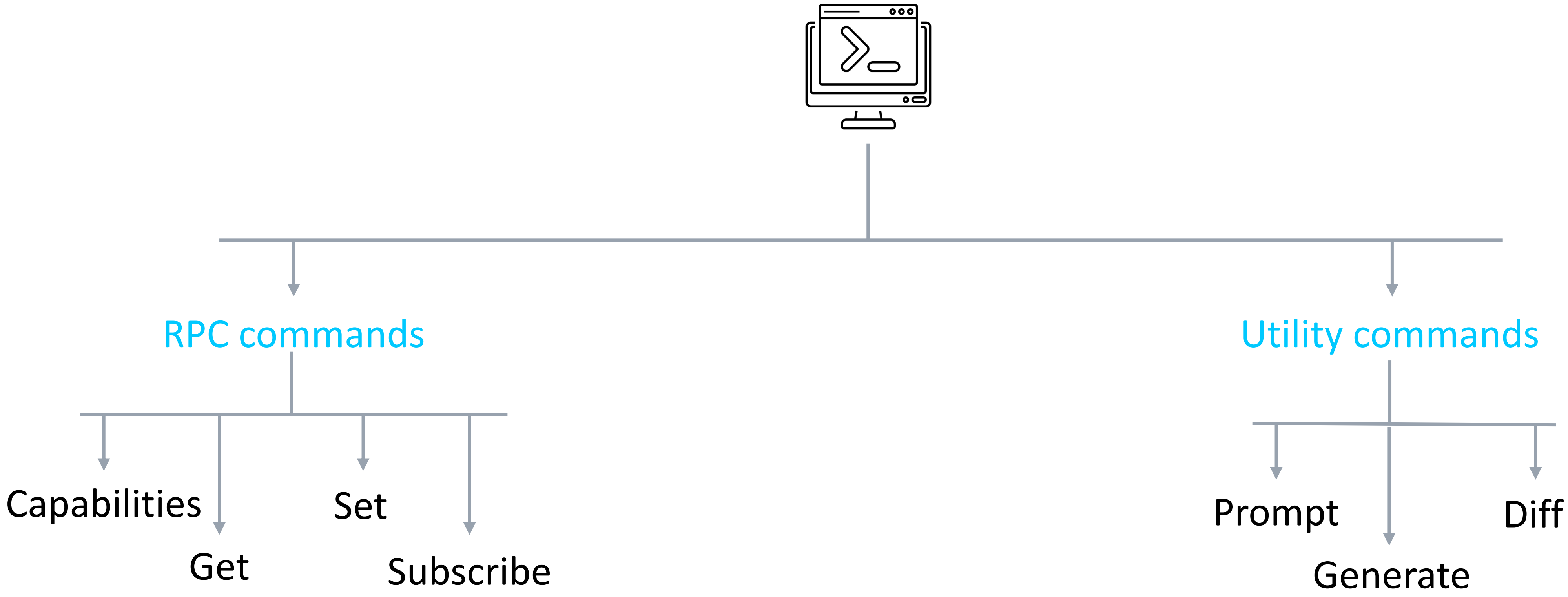
Nokia donated gNMIc to Openconfig

OPENCONFIG / gNMIc

openconfig/gnmic

gNMIc

Rich CLI to explore and test gNMI enabled targets



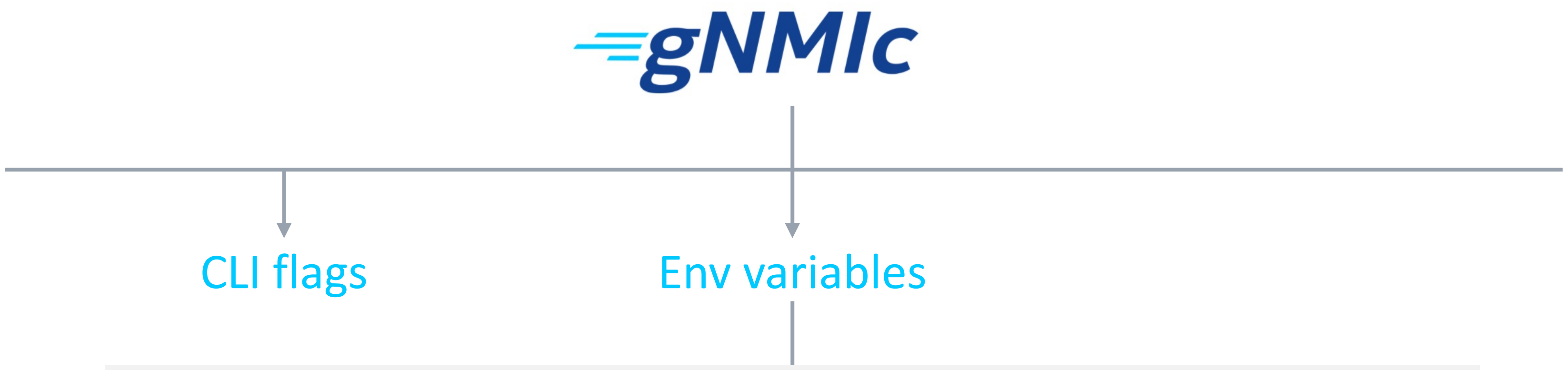
Configure the way you want

CLI flags

```
gnmic [global-flags] [command] [local-flags]
```

```
gnmic --address 10.0.0.1:57400 \
      --username admin --password admin\
      --skip-verify \
      get --path /interfaces/interface[name=mgmt0]
```

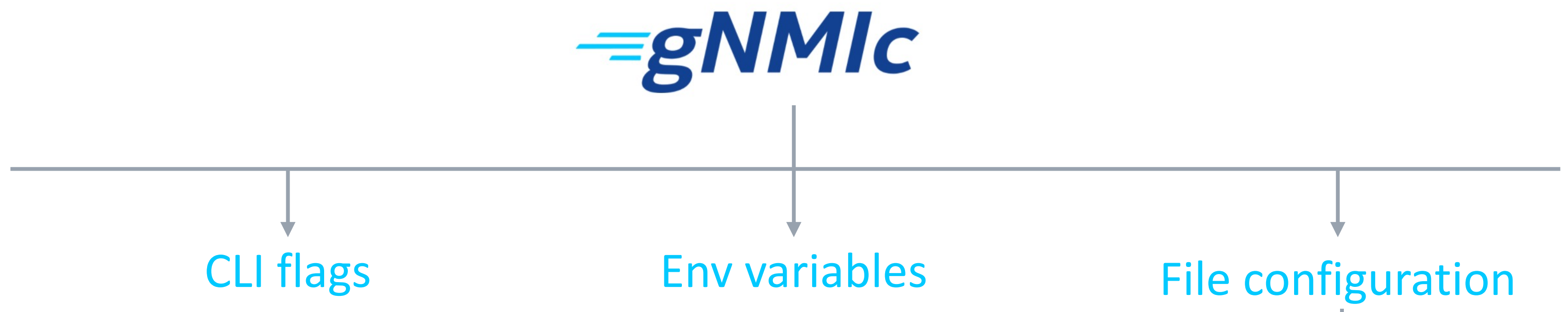
Configure the way you want



```
export GNMIC_ADDRESS=10.0.0.1:57400
export GNMIC_USERNAME=admin
export GNMIC_PASSWORD=admin
export GNMIC_SKIP_VERIFY=true

gnmic [global-flags] [command] [local-flags]
```

gNMIc

Configure the way you want



```
$ cat .gnmic.yaml

address: router1
username: admin
password: admin
insecure: true
encoding: json_ietf
get-path:
  - /interfaces/interface[name=mgmt0]

$ gnmic --config .gnmic.yaml get
```

gNMIc

Configuration file

- Preferred way to configure gnmic in daemon mode
- When using as CLI, config file helps setting common parameters once

```
$ gnmic --config gnmic.yml \
    -a target.name \
    get --path /interface/interface
```

Start gnmic using targets configured in the file

```
$ gnmic --config gnmic.yml \
    subscribe
```

gnmic.yml

```
# every CLI flag is possible to set
# in a config file
username: admin
password: admin
insecure: true
encoding: json_ietf

targets:
   :
subscriptions:
   :
outputs:
   :
processors:
   :
clustering:
   :
```

gNMIc

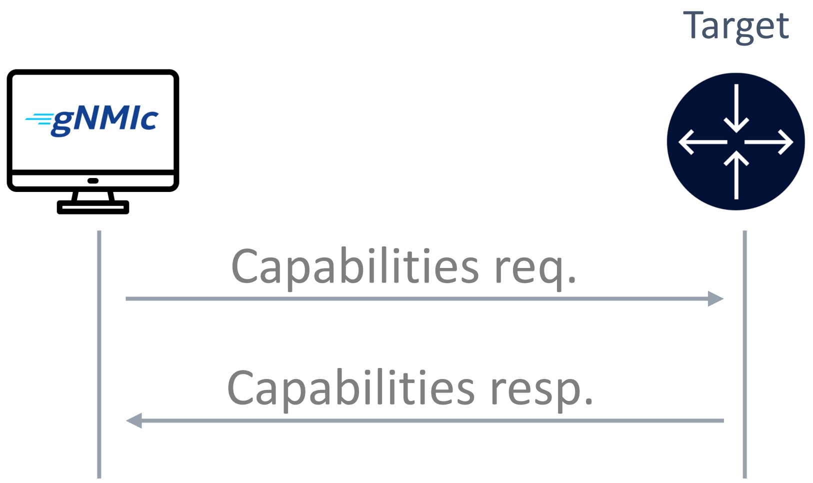
Capabilities command

- Discover supported YANG modules & encodings
- Identify supported gNMI version

```
$ gnmic -a target.name \
        -u admin –p admin \
        --skip-verify \ # or --insecure
        capabilities
```

Output

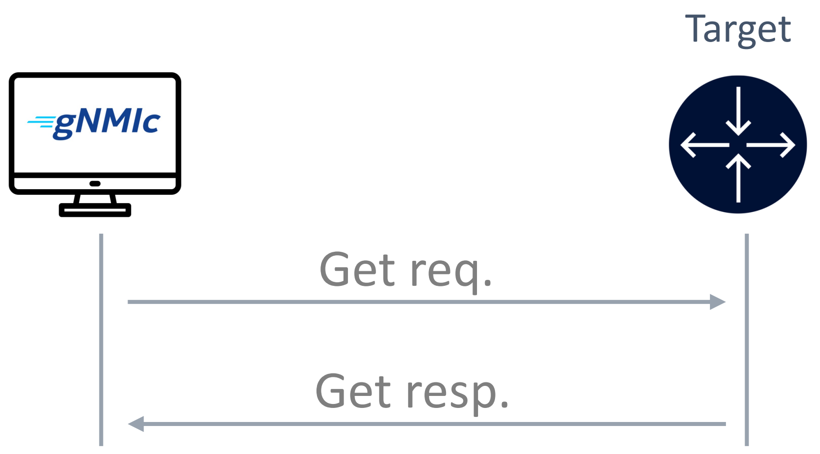
```
gNMI version: 0.7.0
supported models:
  - urn:srl_nokia/aaa:srl_nokia-aaa, Nokia, 2021-11-30
  - urn:srl_nokia/aaa-password:srl_nokia-aaa-password, Nokia, 2022-06-30
--snip--
supported encodings:
  - JSON_IETF
  - ASCII
  - PROTO
```



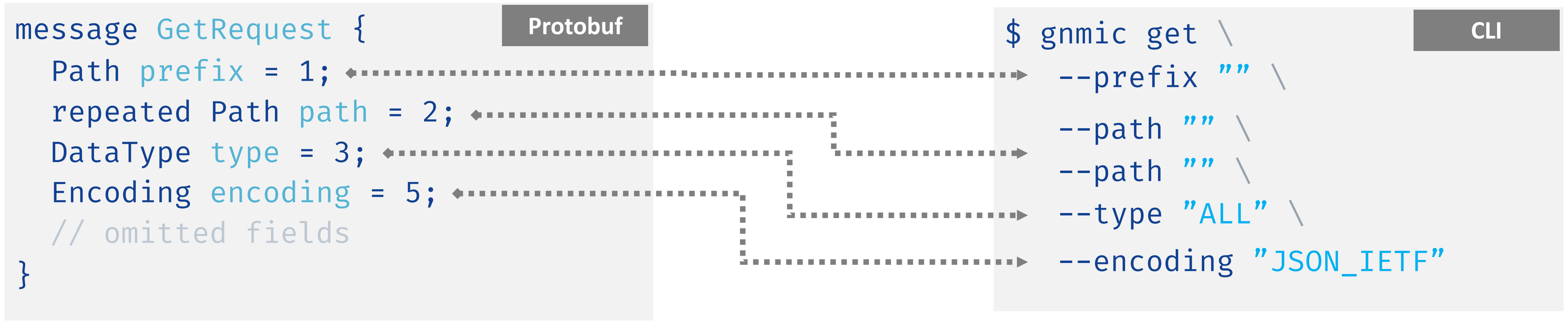
gNMIc

Get command

Retrieve a snapshot of the config/state that exists on the target



gNMIc CLI commands are modelled after the gNMI messages protobuf definition



Protobuf

```
message GetRequest {
  Path prefix = 1;
  repeated Path path = 2;
  DataType type = 3;
  Encoding encoding = 5;
  // omitted fields
}
```

CLI

```
$ gnmic get \
   --prefix "" \
   --path "" \
   --path "" \
   --type "ALL" \
   --encoding "JSON_IETF"
```

Get Command

gNMIc

Get command (cont.)

Intended for clients to retrieve relatively small sets of data as complete objects, for example a part of the configuration

```
$ gnmic -a clab-nanog87-leaf1 \
        -u admin -p admin \
        --skip-verify \
        --encoding json_ietf \
        --format prototext \
        get \
        --path /system/state/hostname
```

Output

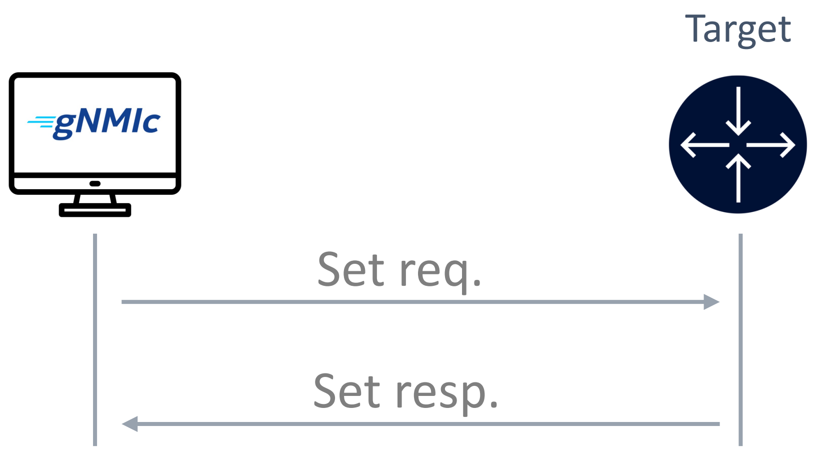
```
notification:  {
  timestamp:  1660164704012223553
  update:  {
    path:  {
      elem:  {
        name:  "openconfig-system:system"
      }
      elem:  {
        name:  "state"
      }
      elem:  {
        name:  "hostname"
      }
    }
    val:  {
      json_ietf_val:  "\"leaf1\""
    }
  }
}
```

Get Command

gNMIc

Set command

- Used by clients to modify the target's configuration.
- It allows updating, replacing or completely deleting configuration items. The operations order is significant.
- All operations in a Set request are considered a single transaction.



Protobuf

```
message SetRequest {
  Path                prefix  = 1;
  repeated Path       delete  = 2;
  repeated Update     replace = 3;
  repeated Update     update  = 4;
 // omitted fields
}
```

Protobuf

```
message Update {
  Path          path = 1;
  TypedValue    val  = 3;
  // omitted fields
}
```

Set Command

gNMIc

Set. Inline values

- Easy way to modify configuration via CLI
- Not suitable for complex JSON structures

```
$ gnmic -a target.name \
   set \
    --update-path "/interface[name=ethernet-1/1]/admin-state" \
    --update-value "enable" \
    --encoding json_ietf
```

Output

```
{
  "source": "target.name",
  …
  "results": [
    {
      "operation": "UPDATE",
      "path": "interface[name=ethernet-1/1]/admin-state"
    }
  ]
}
```

Set Command

gNMIc

Set. File-based values

Allows to modify configuration with complex values

- Nested objects
- Lists, leaf-lists

```
$ gnmic -a target.name \
   set \
   --update-path "/interface[name=ethernet-1/1]" \
   --update-file file.json
```

File

```
$ cat file.json
{
  "admin-state": "enable",
  "description": "to_spine1"
}
```

Output

```
{
  "source": "target.name",
  …
  "results": [
    {
      "operation": "UPDATE",
      "path": "interface[name=ethernet-1/1]/admin-state"
    }
  ]
}
```

Set Command

gNMIc

Set. Request file

- Define the whole set request in a single file.
- Define values in JSON or YAML
- Allows for templated Set requests

```
$ gnmic -a target.name \
   set --request-file set_req.yaml
```

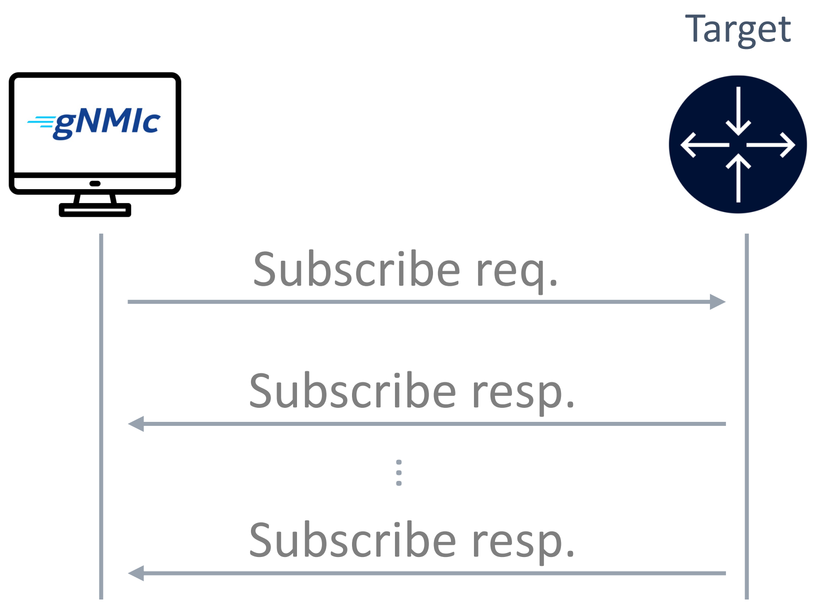
set_req.yaml

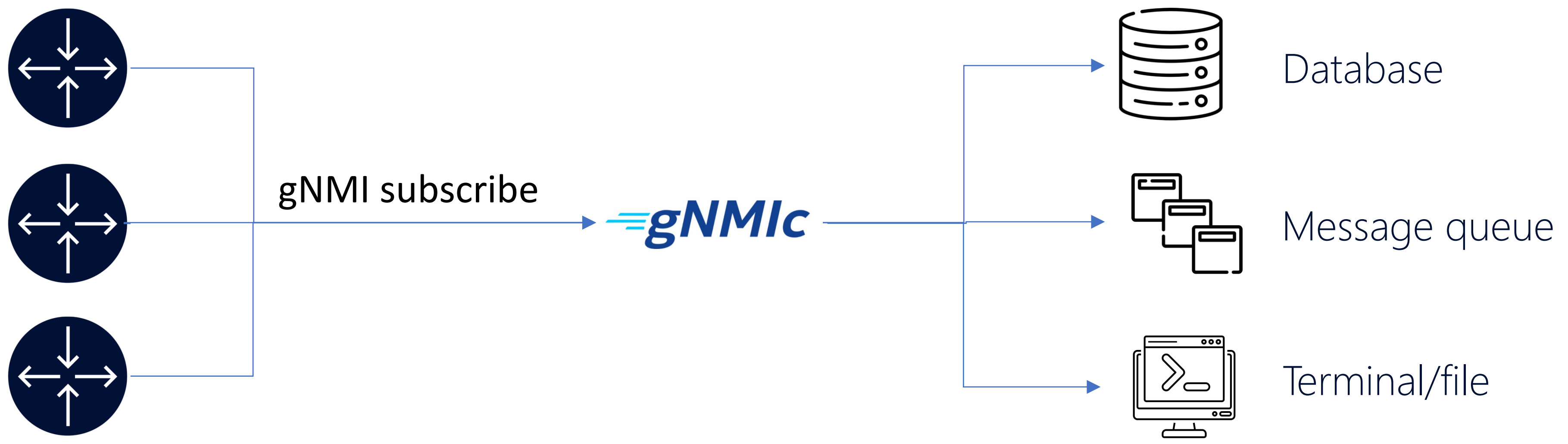
```
replaces:
  - path: /interface[name=ethernet-1/1]
    value:
      admin-state: enable
      description: to_spine1
    encoding: json_ietf

  - path: /interface[name=ethernet-1/2]
    value:
      admin-state: enable
      description: to_spine2
    encoding: json_ietf

deletes:
  - /interface[name=ethrenet-1/3]
  - /interface[name=ethrenet-1/4]
```

Set Command

Subscribe Command (1/2)

- Used by clients that wish to receive updates related to specific objects in the target config or state stores.
- The client creates a subscription that consists of a set of paths and a subscription mode.
- The client collects the streaming telemetry data for further processing and/or storage.





gNMIc

Subscribe Command (2/2)

```
$ gnmic -a clab-nanog87-leaf1 \
        -u admin -p admin \
        --skip-verify \
        --encoding json_ietf \
        subscribe \
        --path /interfaces/interface/state/counters \
        --mode stream \
        --stream-mode sample \
        --sample-interval 10s
```

Receive interfaces counters every 10s

```
$ gnmic -a clab-nanog87-leaf1 \
        -u admin -p admin \
        --skip-verify \
        -e json_ietf \
        subscribe \
        --path /interfaces/interface/state/oper-status \
        --mode stream \
        --stream-mode on-change
```

Get interfaces oper-status when it changes

gNMIc

Generating paths and configuration payloads

```
#
$ git clone https://github.com/openconfig/public
$ cd public

# generate configuration payload (YAML)
$ gnmic generate \
  --file release/models/interfaces \
  --dir release/models \
  --dir third_party/ietf
```

```
interfaces:
  interface:
    - aggregation:
        config:
          lag-type:
          min-links:
        switched-vlan:
          config:
            access-vlan:
            interface-mode:
            native-vlan:
            trunk-vlans:
      config:
        description:
        enabled:
```

Generating paths and configuration payloads

```
# download openconfig YANG models
$ git clone https://github.com/openconfig/public
$ cd public

# generate xpaths
$ gnmic generate path \
--file release/models/interfaces \
--dir release/models \
--dir third_party/ietf
```

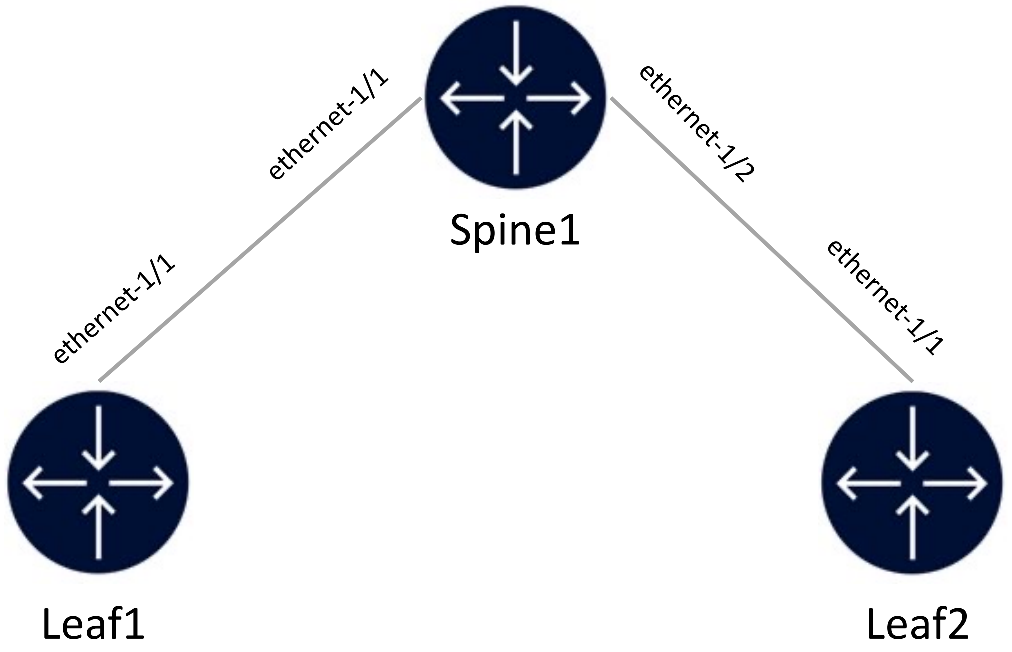
```
/interfaces-state/interface[name=*]/admin-status
/interfaces-state/interface[name=*]/higher-layer-if
/interfaces-state/interface[name=*]/if-index
/interfaces-state/interface[name=*]/last-change
/interfaces-state/interface[name=*]/lower-layer-if
/interfaces-state/interface[name=*]/name
/interfaces-state/interface[name=*]/oper-status
/interfaces-state/interface[name=*]/phys-address
/interfaces-state/interface[name=*]/speed
...
```

gNMIc

CLI tutorial section

The goals of this tutorial section will be to:

- Be able to run basic gNMIc commands
- Be able to establish unsecure/secure gNMI connections
- Be able to set gNMIc CLI attributes via flags, environment variables or config file
- Query the targets configuration
- Modify the targets configuration



Installation

```
# download and install the latest release
$ bash -c "$(curl -sL https://get-gnmic.openconfig.net)"

Downloading
https://github.com/openconfig/gnmic/releases/download/v0.28.0/gnmic_0.28.0_linux_x86_64.tar.gz
Preparing to install gnmic 0.28.0 into /usr/local/bin
gnmic installed into /usr/local/bin/gnmic
version : 0.28.0
 commit : 8315400
   date : 2022-12-07T17:02:16Z
 gitURL : https://github.com/openconfig/gnmic
   docs : https://gnmic.openconfig.net
```

```
# pull latest release from github registry
$ docker pull ghcr.io/openconfig/gnmic:latest
```

gnmic.openconfig.net/install/

gNMIc

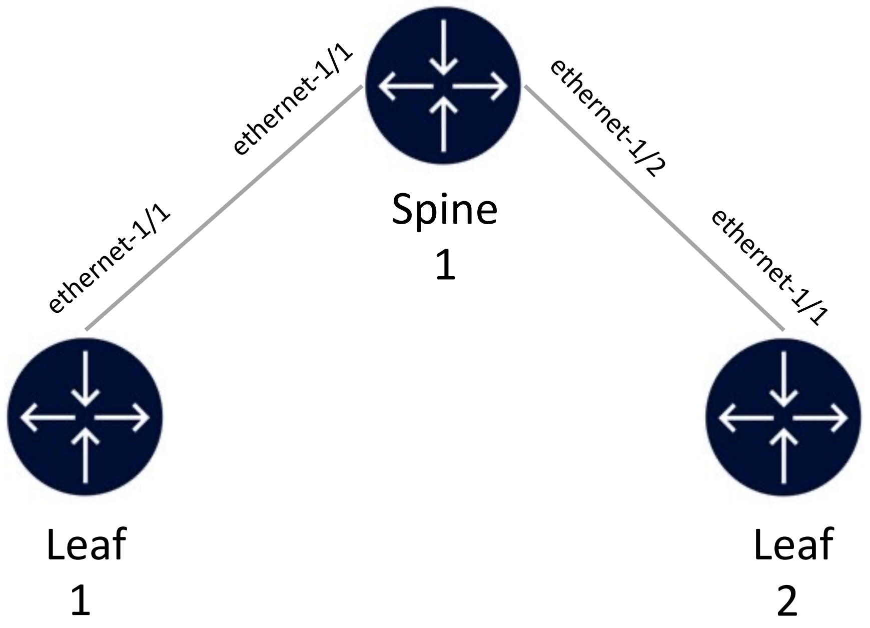
Tutorial topology

github.com/karimra/gnmic-nanog87

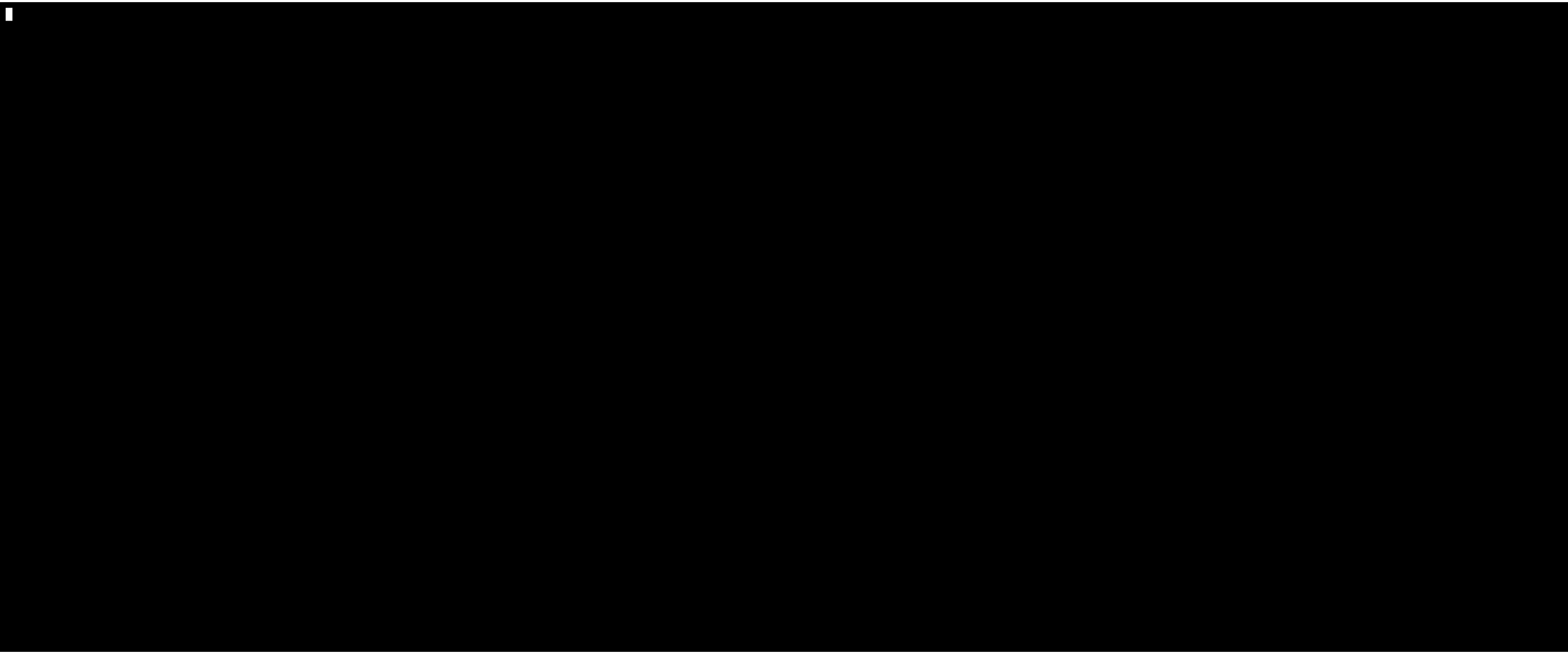
```
$ git clone https://github.com/karimra/gnmic–nanog87.git
$ cd gnmic-nanog87
```

```
$ sudo clab deploy -t topos/cli/nanog87.clab.yaml
```

```
$ sudo clab inspect --name nanog87
+---+---------------------+--------------+-----------------------+------+---------+----------------+-----------------------+
| # |        Name         | Container ID |         Image         | Kind |  State  |  IPv4 Address  |     IPv6 Address      |
+---+---------------------+--------------+-----------------------+------+---------+----------------+-----------------------+
| 1 | clab-nanog87-leaf1  | 35a16da0eafa | ghcr.io/nokia/srlinux | srl  | running | 172.20.20.4/24 | 2001:172:20:20::4/64 |
| 2 | clab-nanog87-leaf2  | 0df0978675c9 | ghcr.io/nokia/srlinux | srl  | running | 172.20.20.3/24 | 2001:172:20:20::3/64 |
| 3 | clab-nanog87-spine1 | 93ff4c9d1fd4 | ghcr.io/nokia/srlinux | srl  | running | 172.20.20.2/24 | 2001:172:20:20::2/64 |
+---+---------------------+--------------+-----------------------+------+---------+----------------+-----------------------+
```



Get RPC

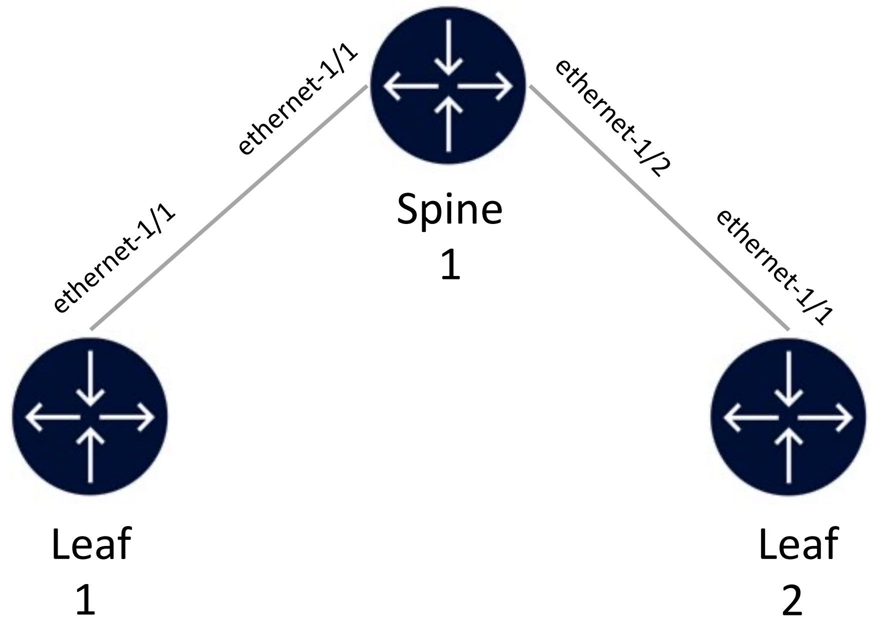


gNMIc

Set RPC

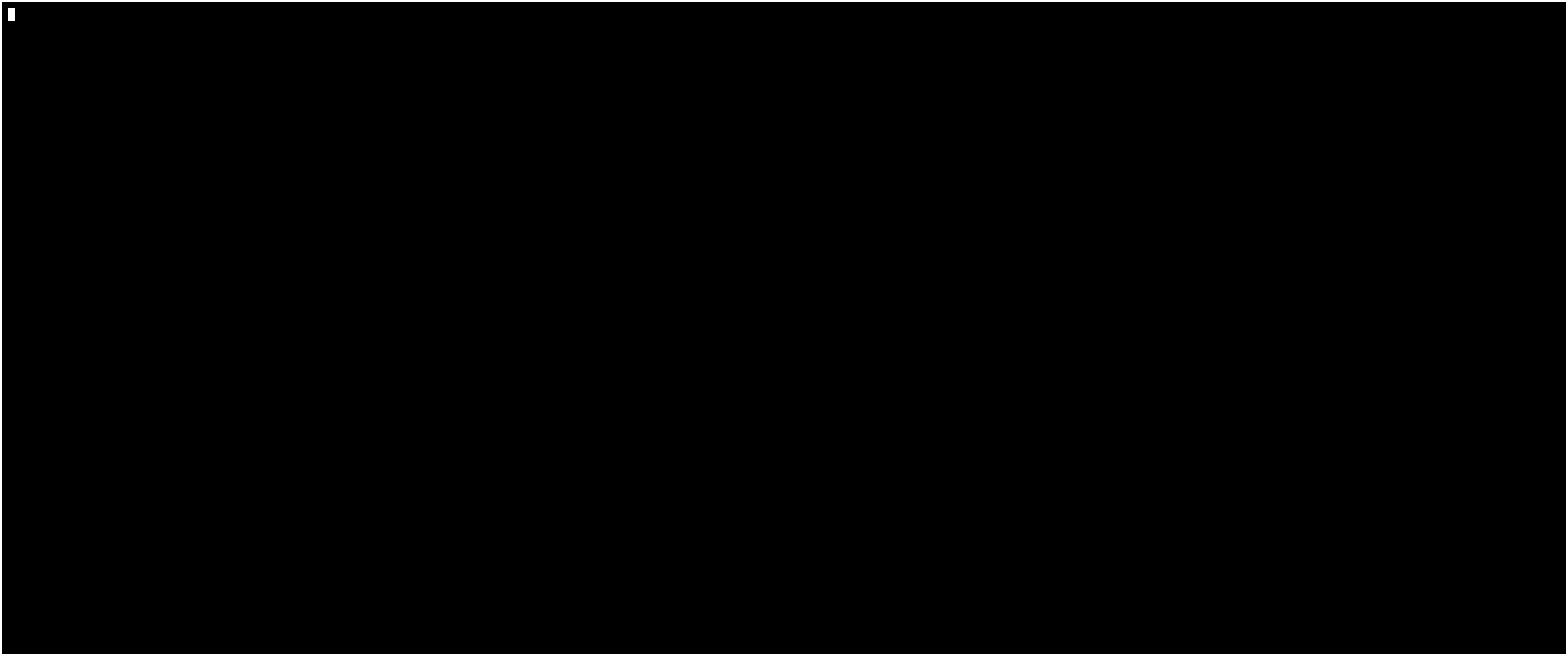
Configure interfaces and subinterfaces 3 different ways:

- Using CLI flags
- Using a configuration file (JSON/YAML)
- Using a templated Set Request



gNMIc

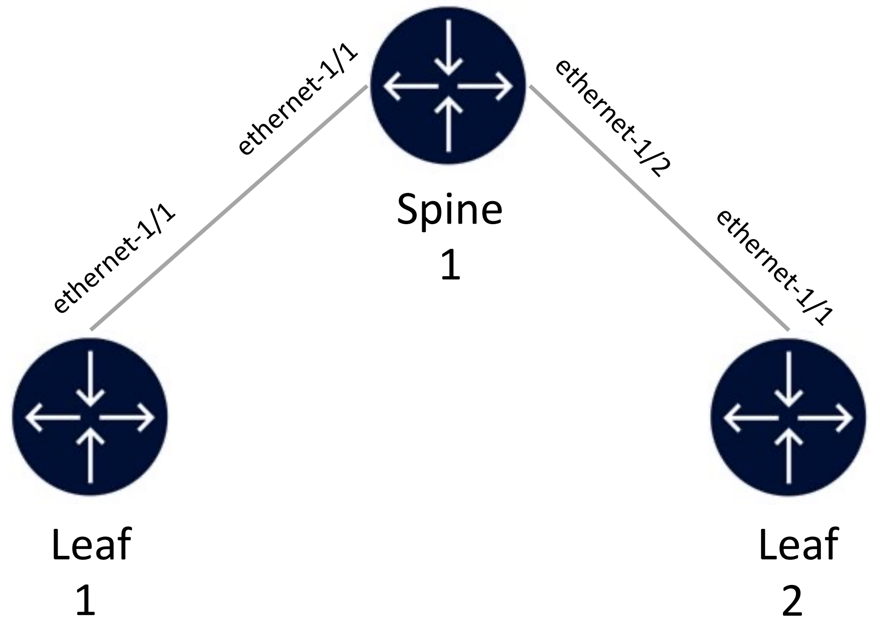
Set RPC



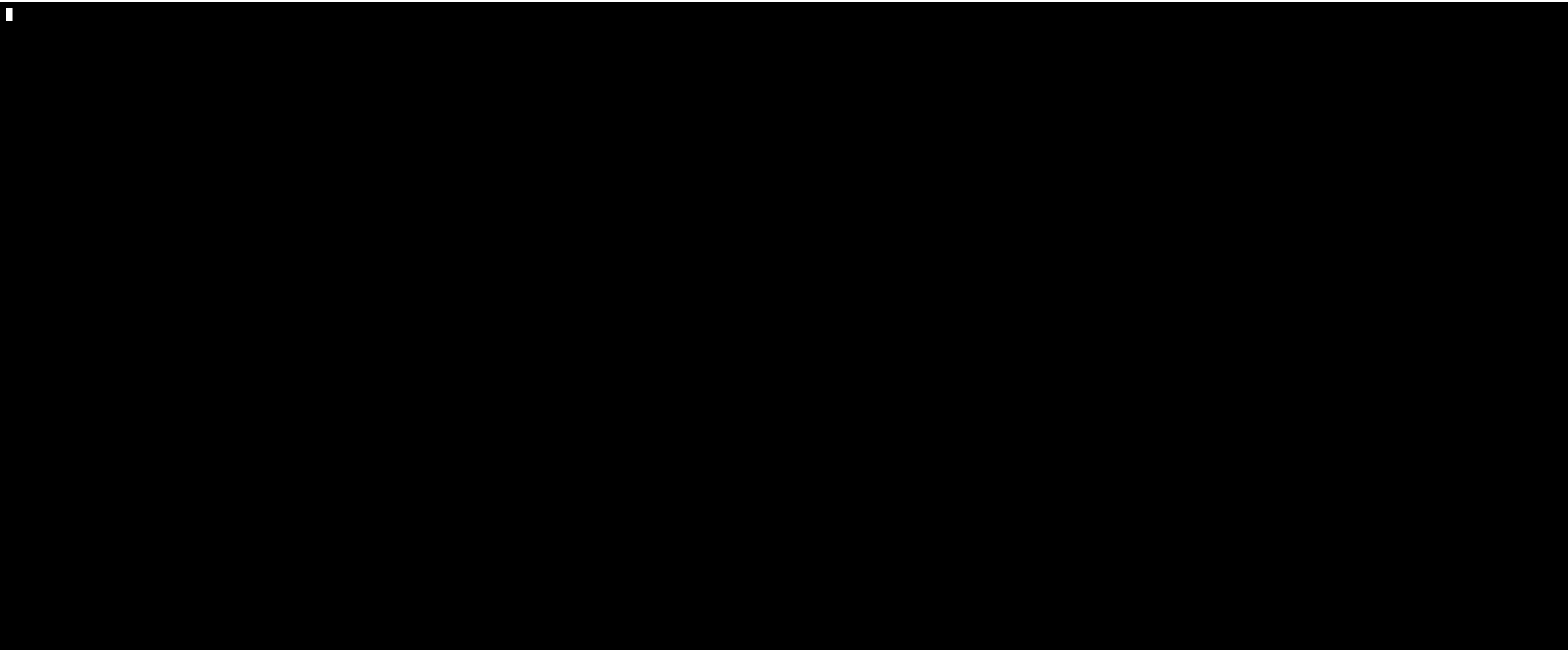
gNMIc

Subscribe RPC

Define different types of subscriptions using flags and the configuraiton file

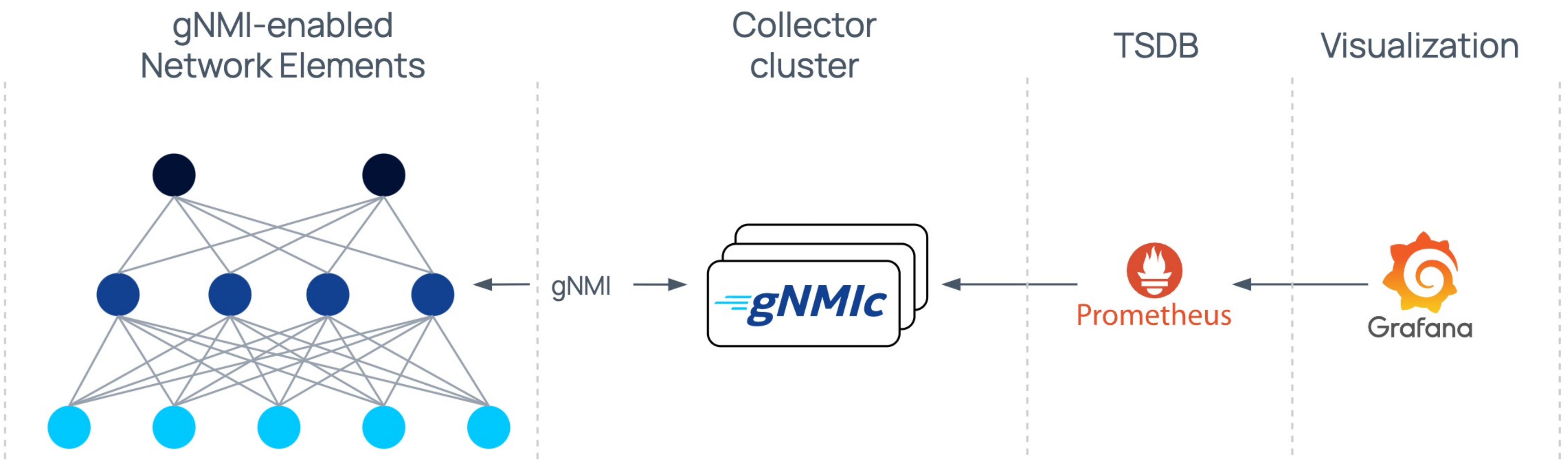


Subscribe RPC



gNMIc

A performant and highly available gNMI collector

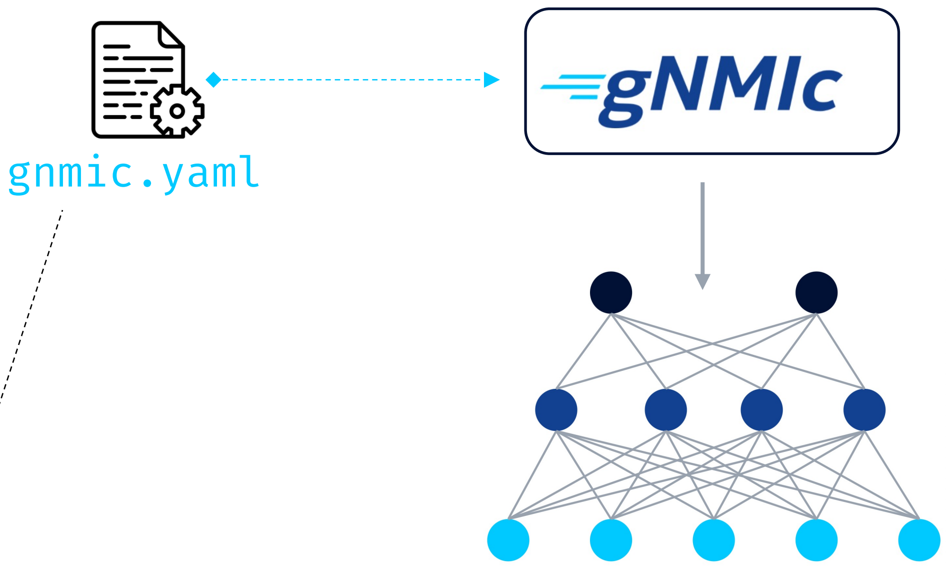


gNMIc

Defining targets

```
#
targets:
  leaf1:
    address: clab-nanog87-leaf1
    username: admin
    password: admin
    skip-verify: true
  leaf2:
    address: clab-nanog87-leaf2
    username: admin
    password: admin
    skip-verify: true
  spine1:
     address: clab-nanog87-spine2
     username: admin
     password: admin
     skip-verify: true
```

- Define per target connection options
- Bind specific subscriptions
- Bind specific outputs

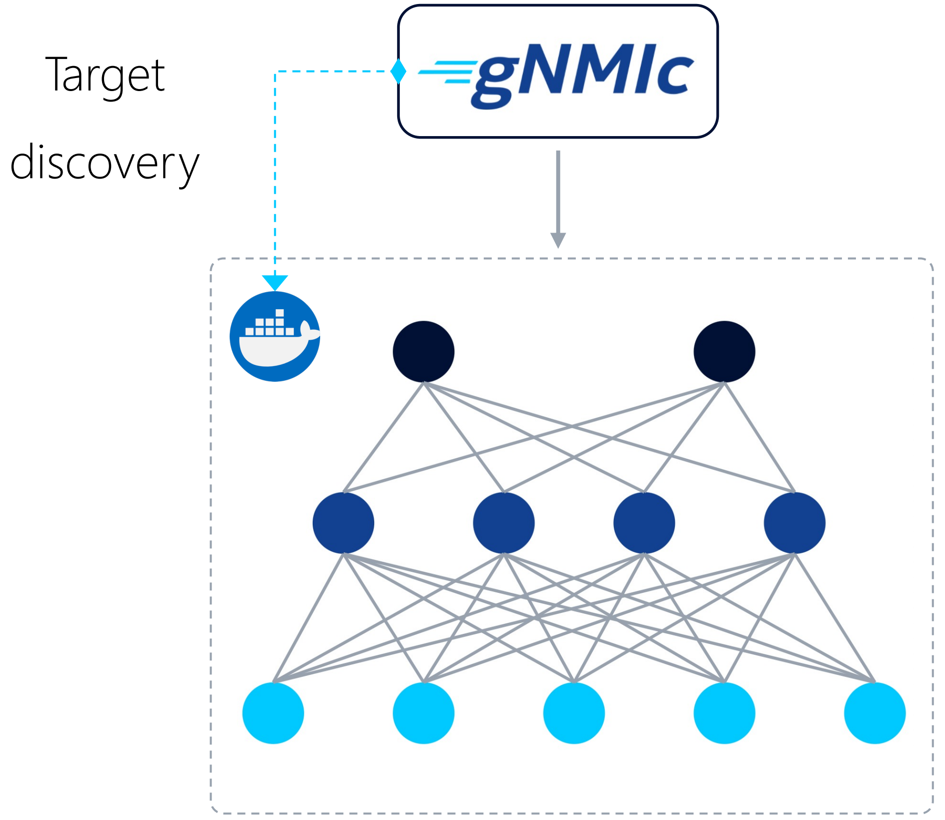


Discovering targets

```yaml
# file-based target discovery
loader:
  type: file
  path: ./targets-config.yaml
```

```yaml
# HTTP based target discovery
loader:
  type: http
  url: http://$addr:$port
```

```yaml
# consul service target discovery
loader:
  type: consul
  services:
    - name: fabric1
```



```yaml
# docker target discovery
loader:
  type: docker
  filters:
    - containers:
        - label: clab-node-kind=srl
```

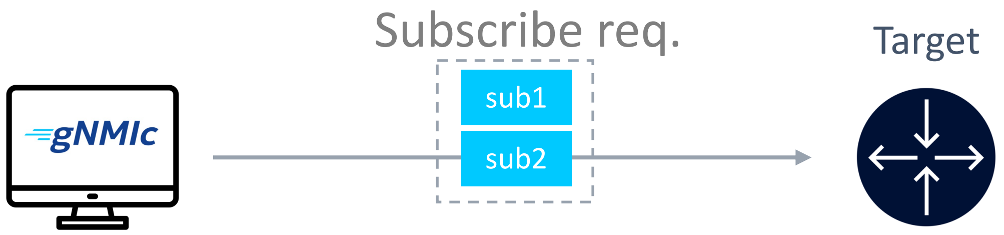
Target Discovery

gNMIc

Defining subscriptions

gnmic.yaml

```yaml
subscriptions:
  sub1:
    paths:
      - /interfaces/interface/state/counters
    mode: stream
    stream-mode: sample
    sample-interval: 10s
    encoding: json_ietf
    heartbeat-interval: 60s
    suppress-redundant: true
  sub2:
    paths:
      - /interfaces/interface/state/oper-status
    mode: stream
    stream-mode: on-change
    encoding: json_ietf
    heartbeat-interval: 60s
```



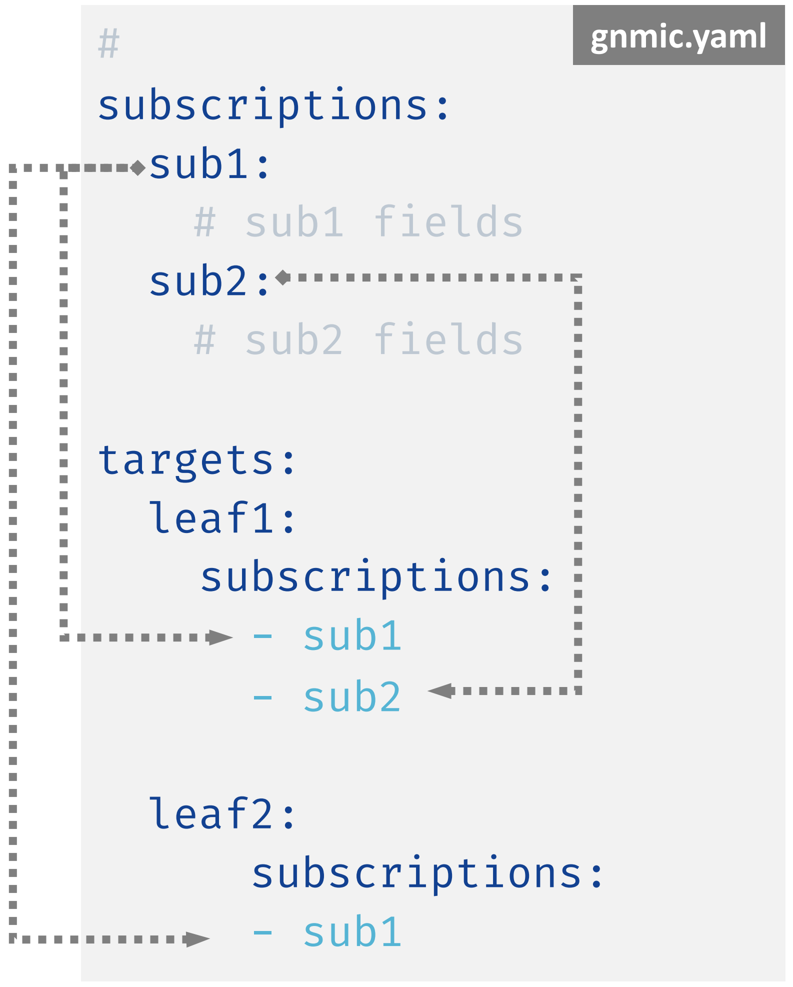
configuring subscriptions

gNMIc

Binding subscriptions to targets

- Associating a target with one or more subscription is as simple as listing the subscription name under the target configuration field “subscriptions”.

- If a target is not explicitly associated with any subscription, gNMIc will subscribe to all defined subscriptions in the file.

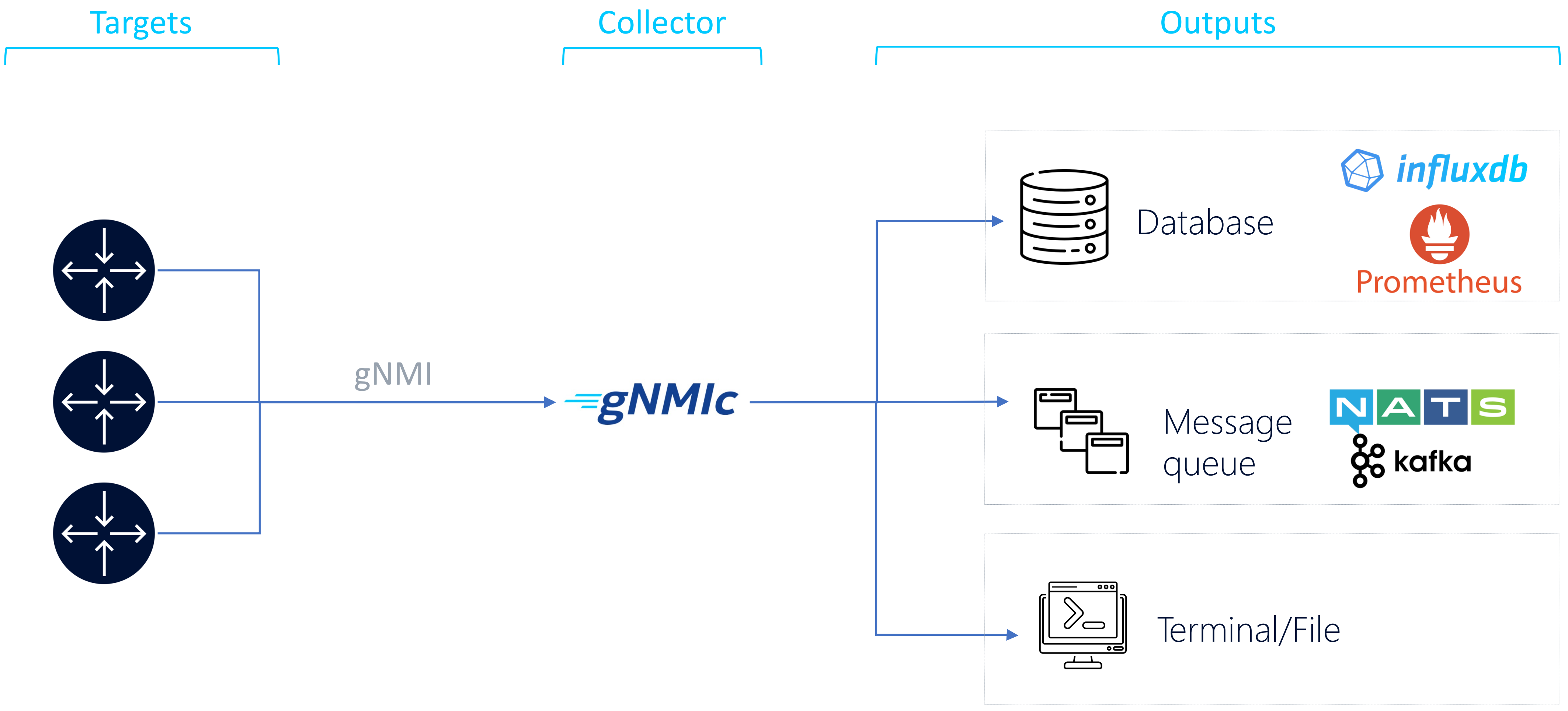

gnmic.yaml

```yaml
#
subscriptions:
  sub1:
    # sub1 fields
  sub2:
    # sub2 fields

targets:
  leaf1:
    subscriptions:
      - sub1
      - sub2

  leaf2:
      subscriptions:
      - sub1
```

configuring subscriptions

gNMIc

Collector outputs

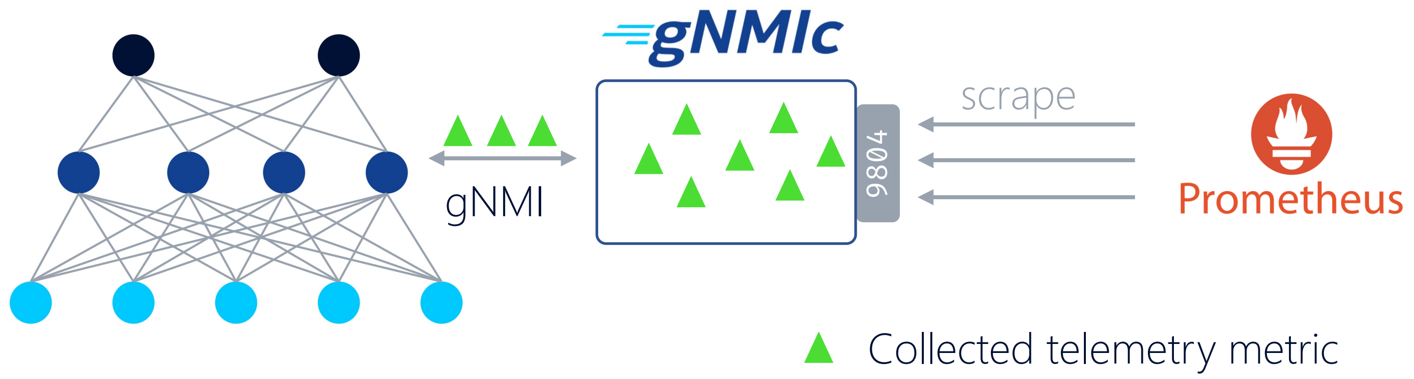


gNMIc

Defining outputs

```
outputs:
  output1:
    type: prometheus
    listen: “:9804”
    export-timestamps: false
    strings-as-labels: false
```

```
targets:
  leaf1:
    outputs:
    - output1
```



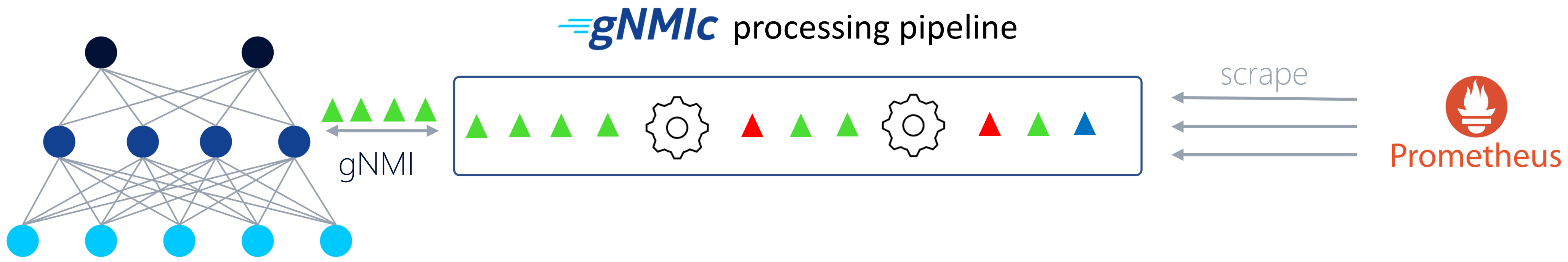
configuring outputs

gNMIc

Processors

- Data manipulation
  - Type conversion
  - Grouping
  - Message enrichment
- Filtering (allow/deny lists)
- Trigger (gNMI, HTTP, script, template)

```
outputs:
  output1:
    event-processors:
      - proc1
      - proc2
```



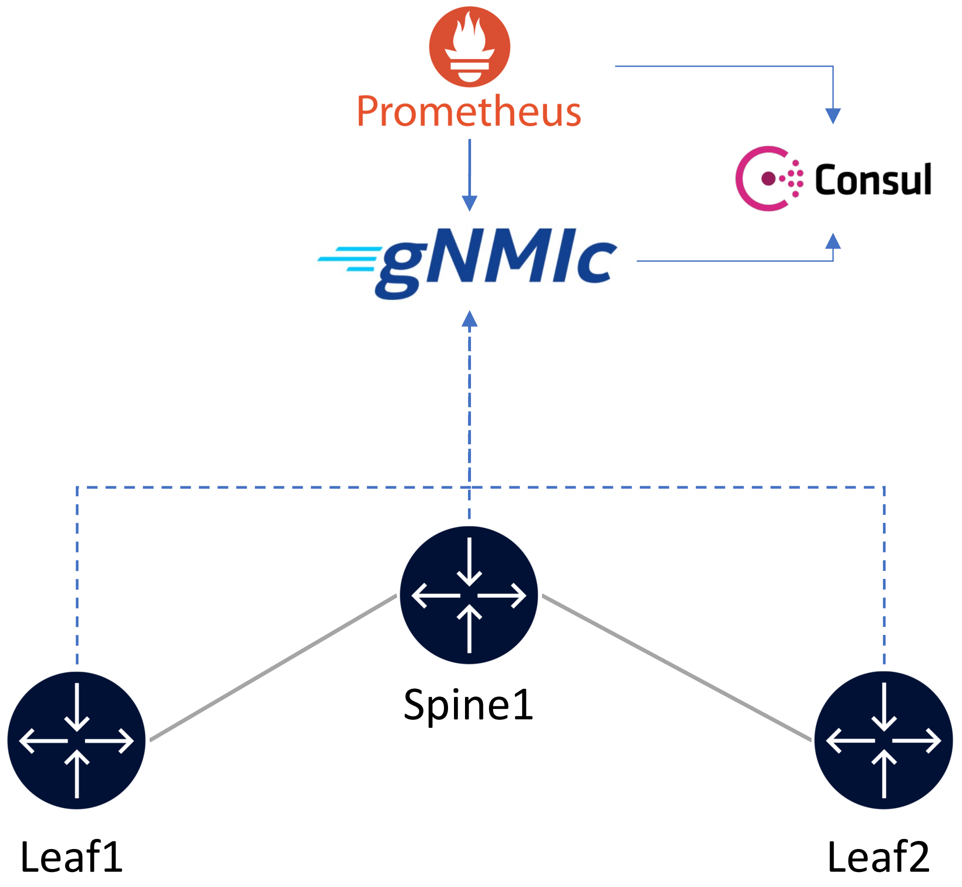
⚙ Processor
▲ Collected telemetry metric
▲▲ Changed/processed metric

gNMIc

Collector demo/tutorial section

The goals of this tutorial section will be to:

- Run gNMIc as a daemon
- Configure targets to be monitored
- Configure subscriptions and bind them to targets
- Configure outputs and bind them to targets
- Manipulate the collected notifications using gNMIc processors



gNMIc

Collector Tutorial

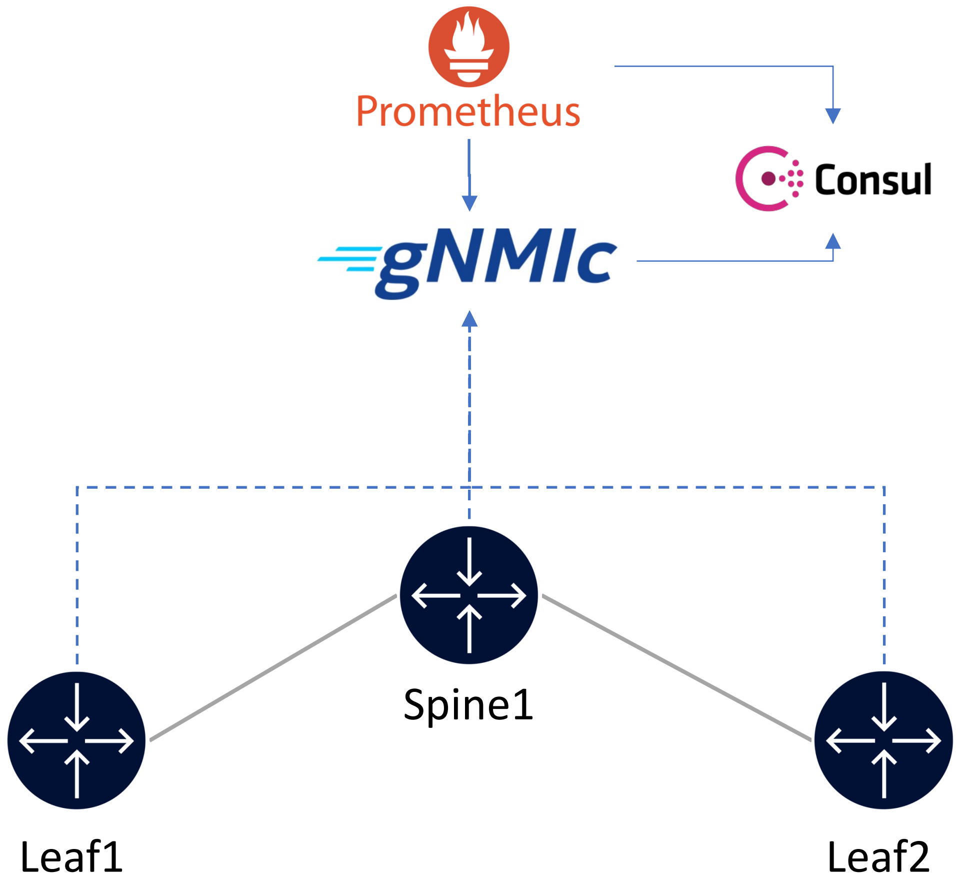
gnmic.yaml

```
username: admin
password: NokiaSrl1!
skip-verify: true

targets:
  clab-nanog87-leaf1:
  clab-nanog87-leaf2:
  clab-nanog87-spine1:

subscriptions:
  sub1:
    paths:
      - /interface/statistics
    encoding: ascii
    sample-interval: 10s

outputs:
  prom-output:
    type: prometheus
    listen: "clab-nanog87-gnmic:9804"
    service-registration:
      address: clab-nanog87-consul-agent:8500
```

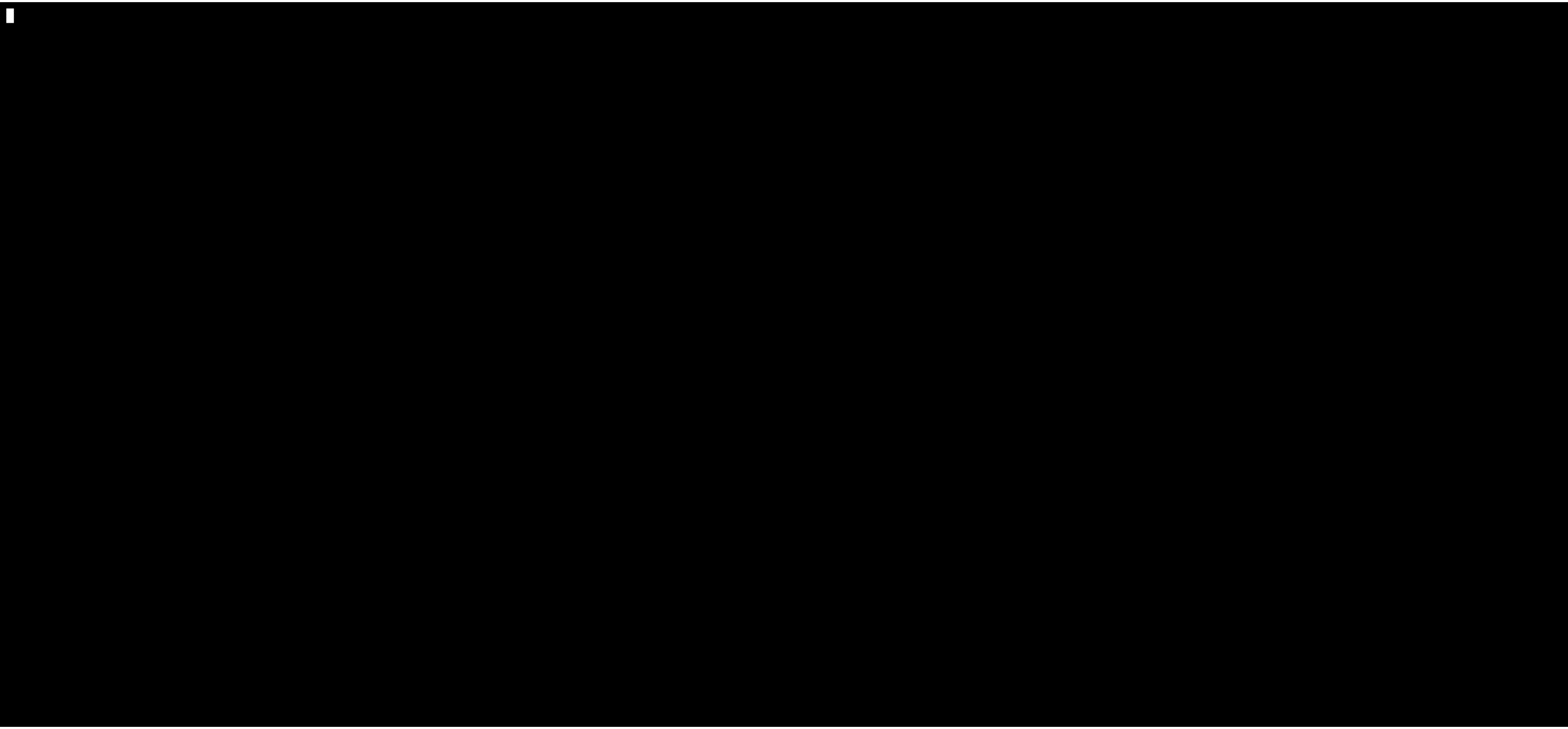
Targets definition

Subscriptions definition

Outputs definition



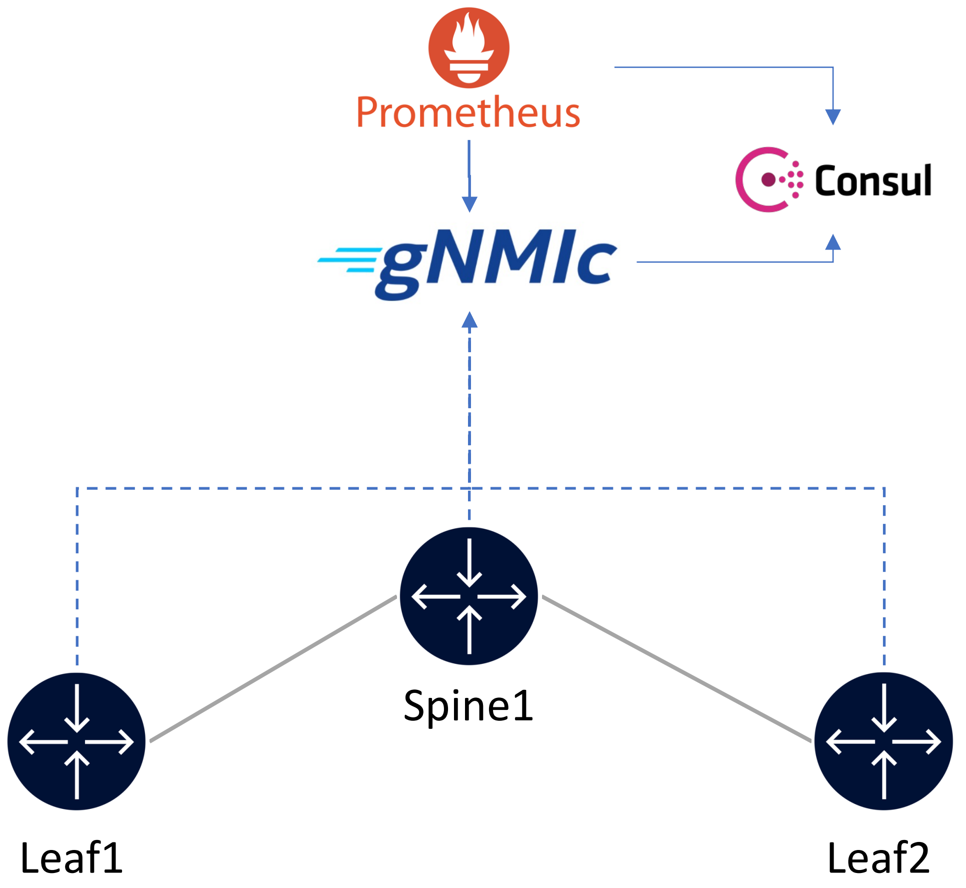
Collector Tutorial



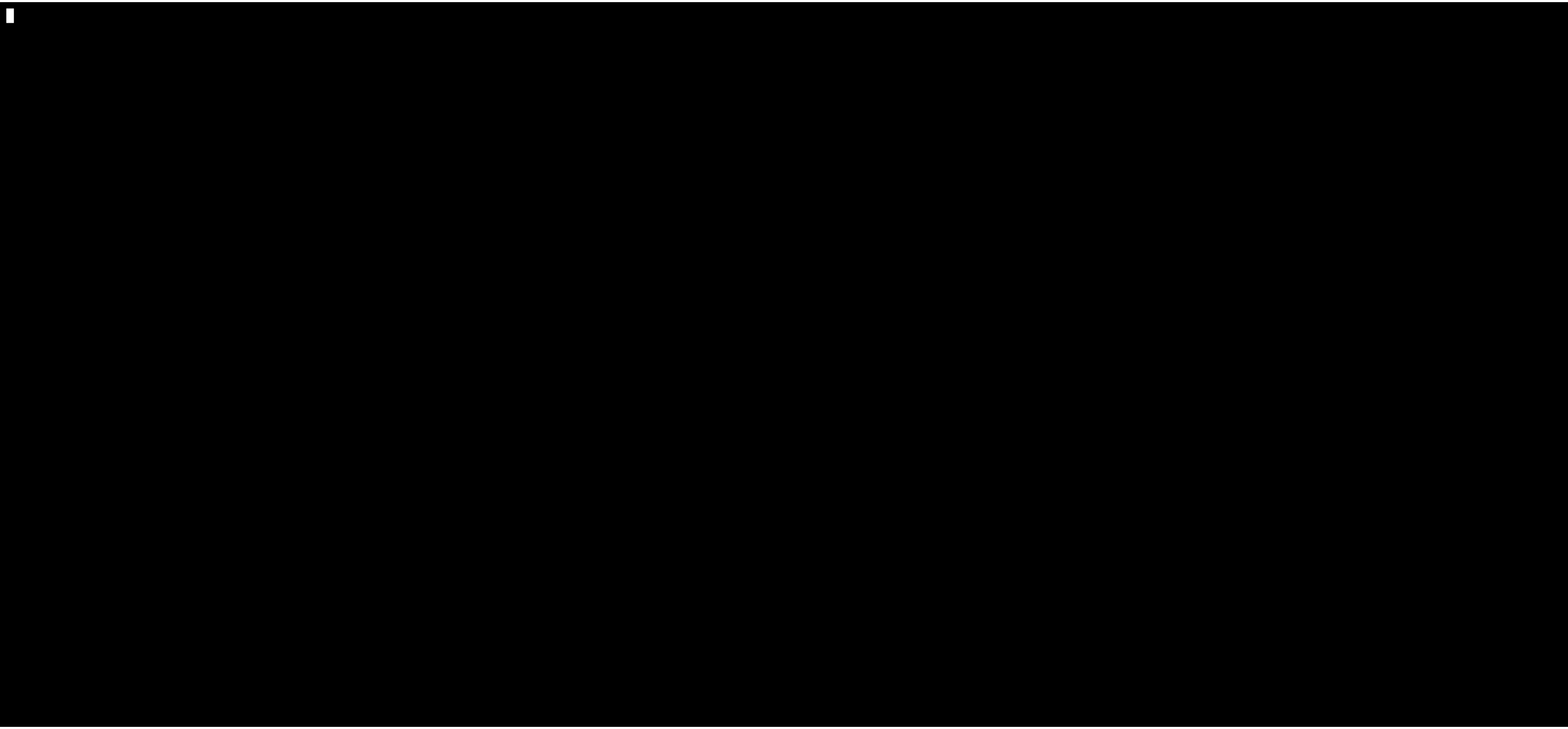
Collector Tutorial – deleting values/messages

gnmic.yaml

```
…

outputs:
  prom-output:
    type: prometheus
    listen: "clab-nanog87-gnmic:9804”
    service-registration:
      address: clab-nanog87-consul-agent:8500
    event-processors:
      - filtering-stats

processors:
  filtering-stats:
    event-delete:
      value-names:
        - ".*multicast.*"
        - ".*broadcast.*"
        - ".*carrier-transitions.*"
        - ".*unicast.*"
        - ".*error.*"
        - ".*discarded.*"
        - ".*mirror.*"
```

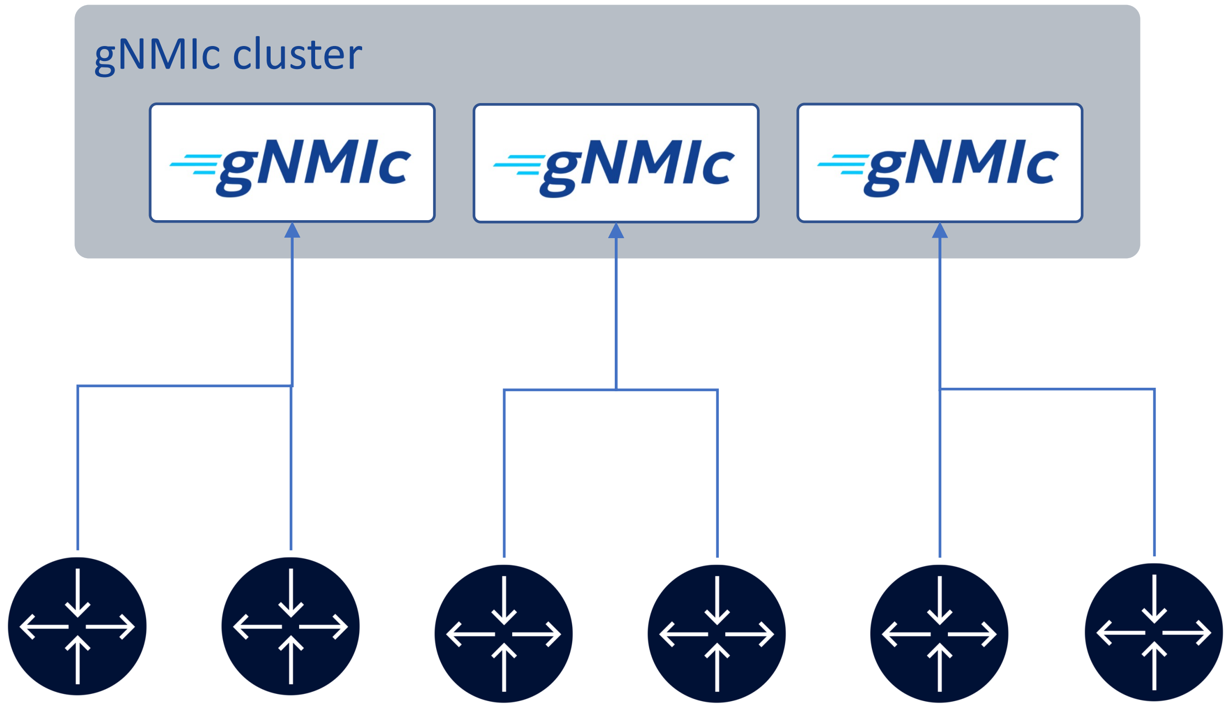


Collector Tutorial – filtering values

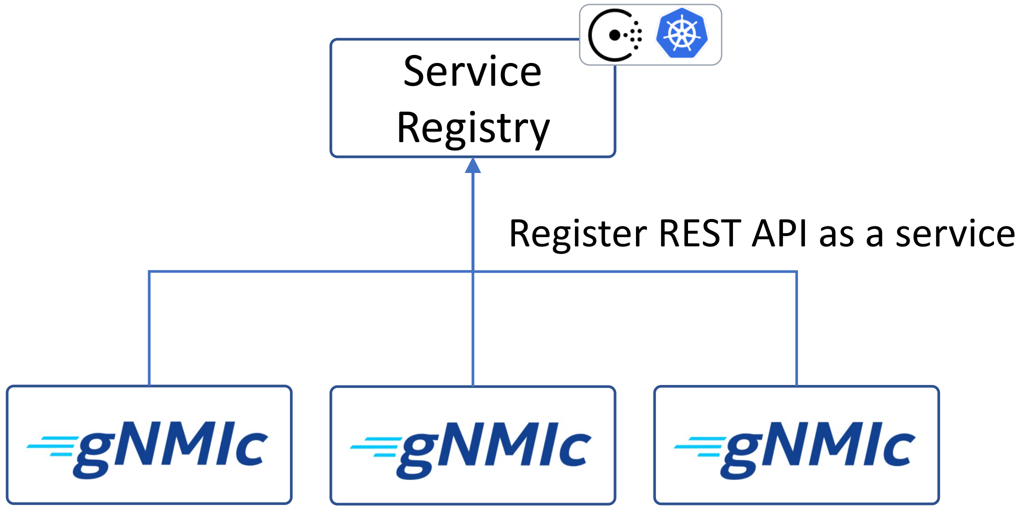


Clustering: The more the merrier

- High availability
- Scaling
- Automatic target redistribution



Clustering: Automatic cluster formation

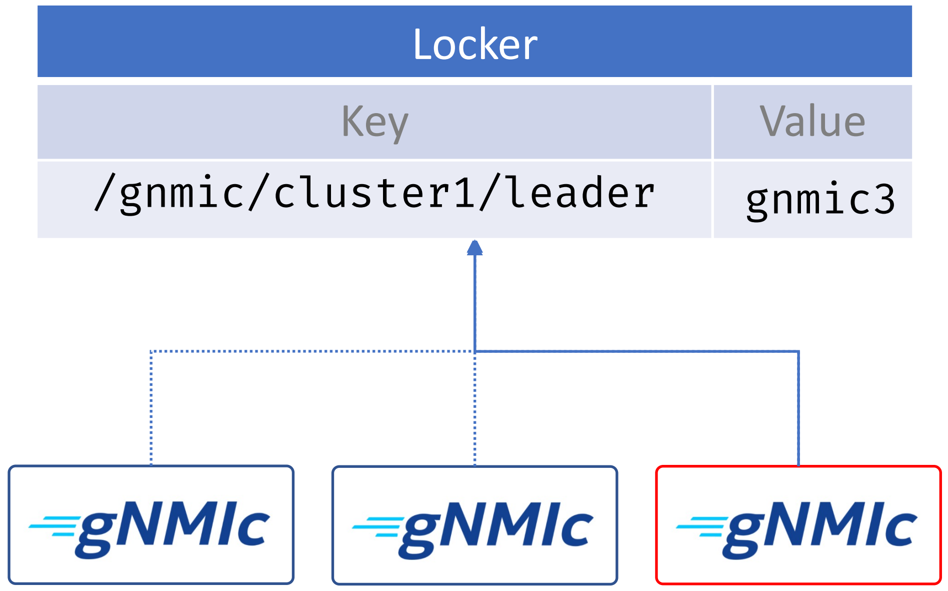


```
api-server:
  address: “:7890”

clustering:
  locker:
    type: consul
    address: clab-nanog87-consul:8500
```

```
api-server:
  address: “:7890”

clustering:
  locker:
    type: k8s
```

Clustering: Leader Election

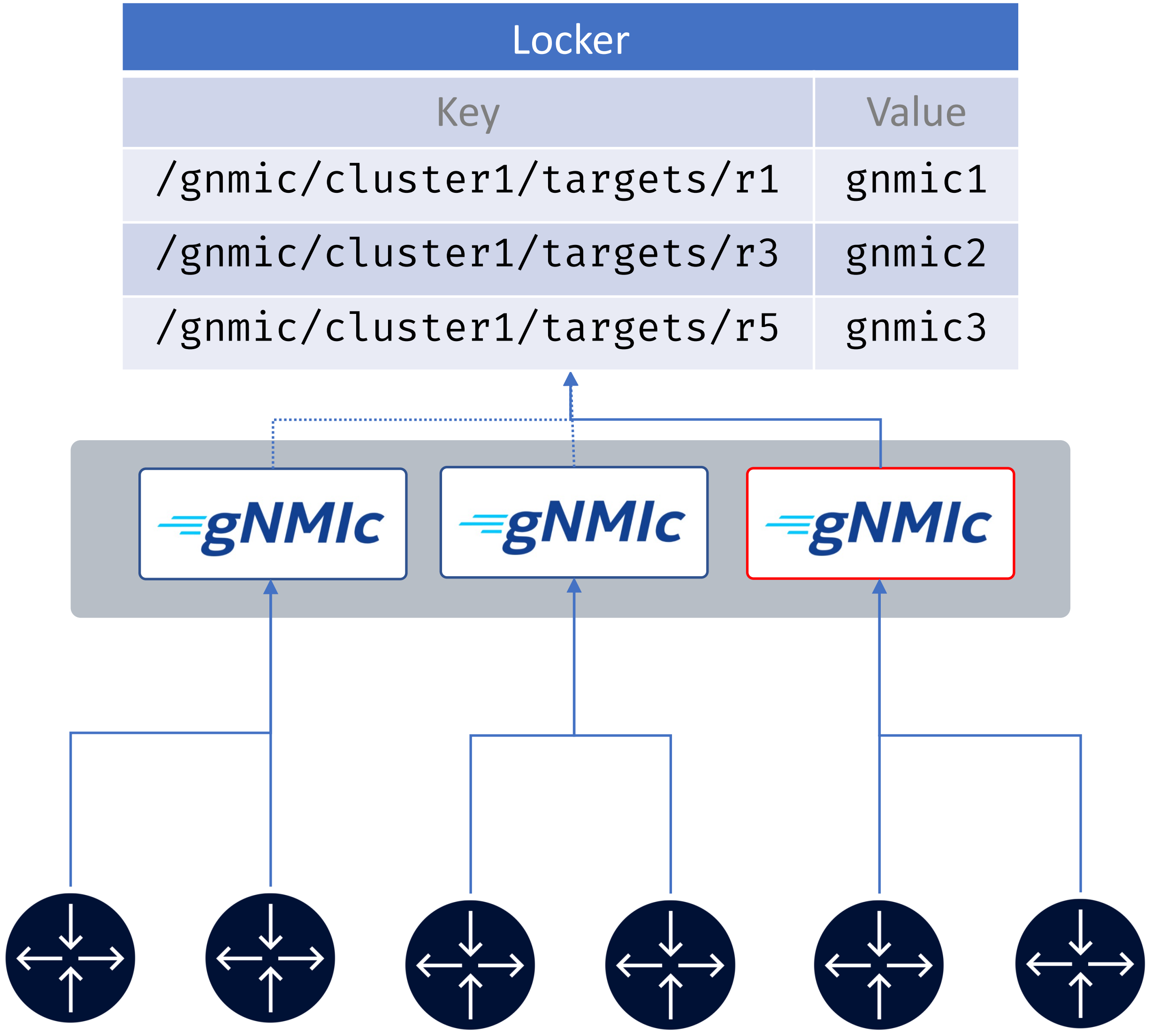
- At startup, all cluster instances attempt to acquire the lock of a well-defined key.
- The first gNMIc instance to lock the key becomes the leader.
- Instances which failed to become leader continue to try acquiring the key to take over in case the leader fails.
- When using Kubernetes as a locker, the Lease Resource is used as a key lock.



gNMIc

Clustering: Target Distribution

- The cluster leader is responsible for assigning targets to all gNMIc instances using the REST API.
- When assigned a target, a gNMIc instance creates the configured subscriptions and locks a key specific to the target, effectively claiming ownership over it.
- When choosing which instance should be assigned the next target, the leader considers the available gNMIc instances as well as the number of targets already assigned.

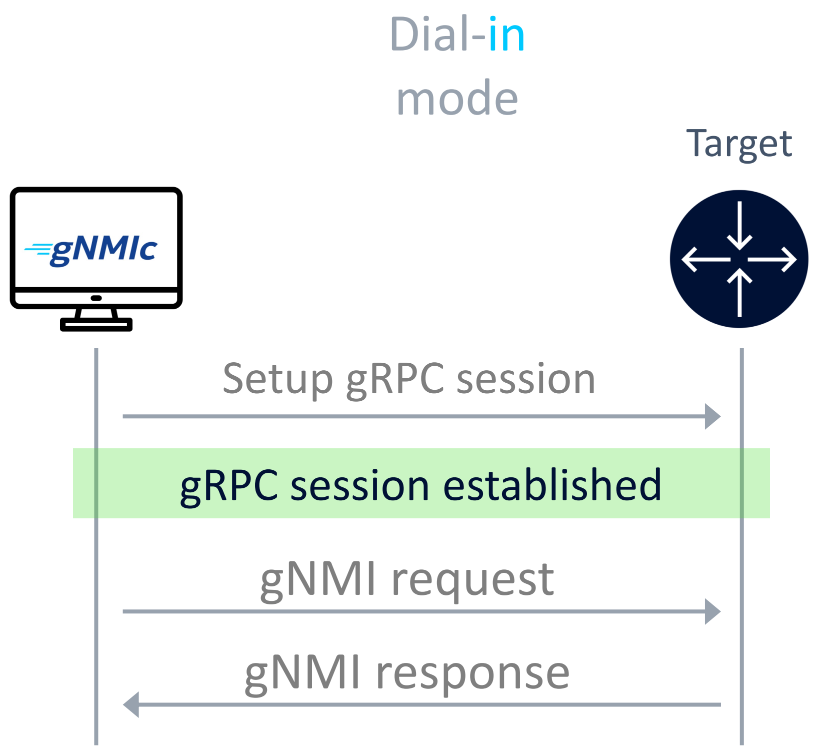


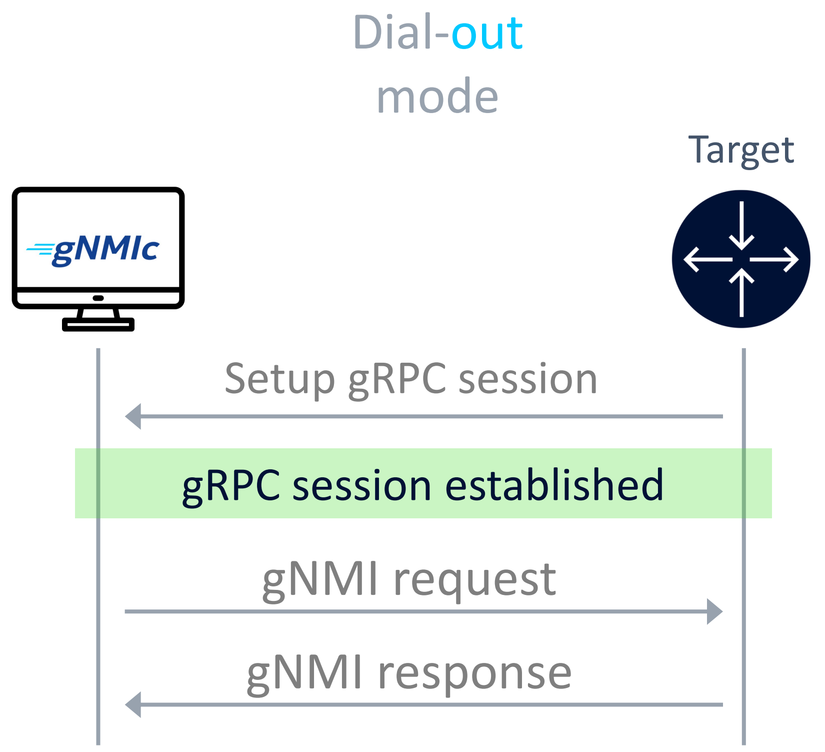
Locker

Key	Value
/gnmic/cluster1/targets/r1	gnmic1
/gnmic/cluster1/targets/r3	gnmic2
/gnmic/cluster1/targets/r5	gnmic3

Dial-out telemetry

Dial-out mode enables streaming telemetry applications for targets that otherwise can't be reached out from the collector

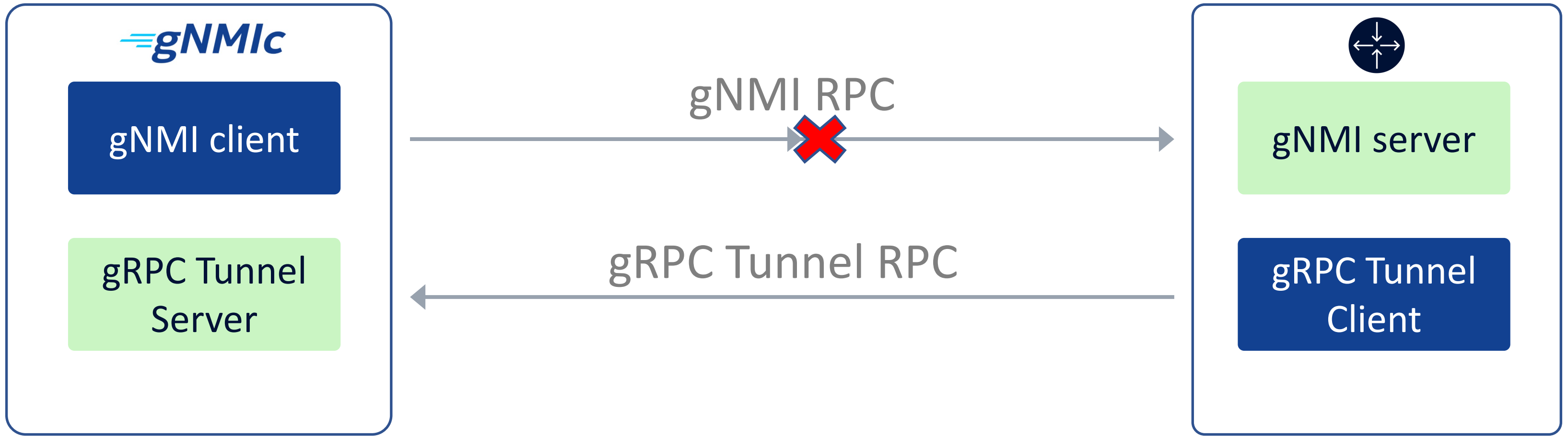




gNMIc as a gRPC tunnel server

Dialout telemetry is achieved using [openconfig/grpc-tunnel]()

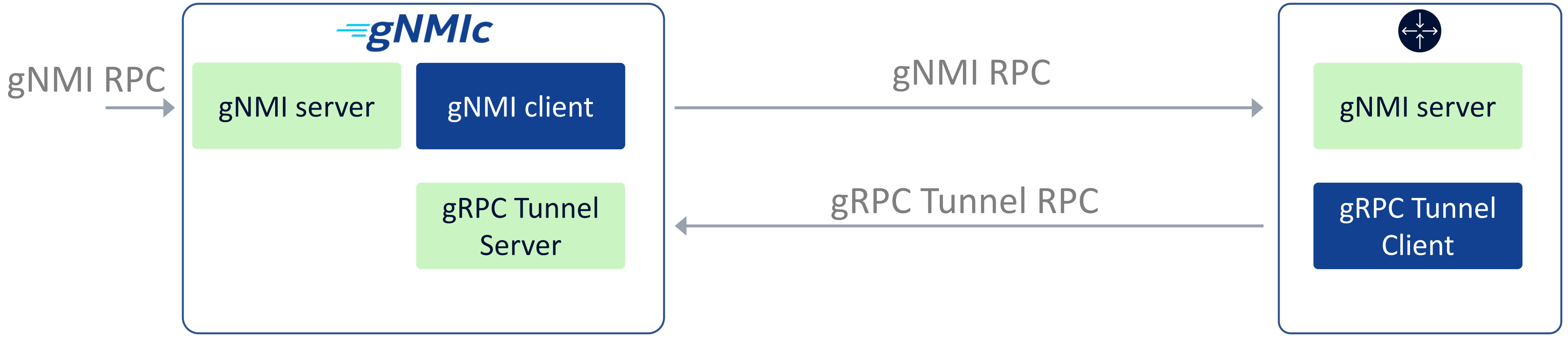
gNMIc acts as gRPC tunnel server allowing the target to establish a gRPC tunnel that gNMIc will use to send tunneled gNMI RPCs.



```
$ cat tunnel_server_config.yaml

tunnel-server:
  address: ":57401"
```

```
$ gnmic --config tunnel_server_config.yaml \
        --use-tunnel-server \
        subscribe
```

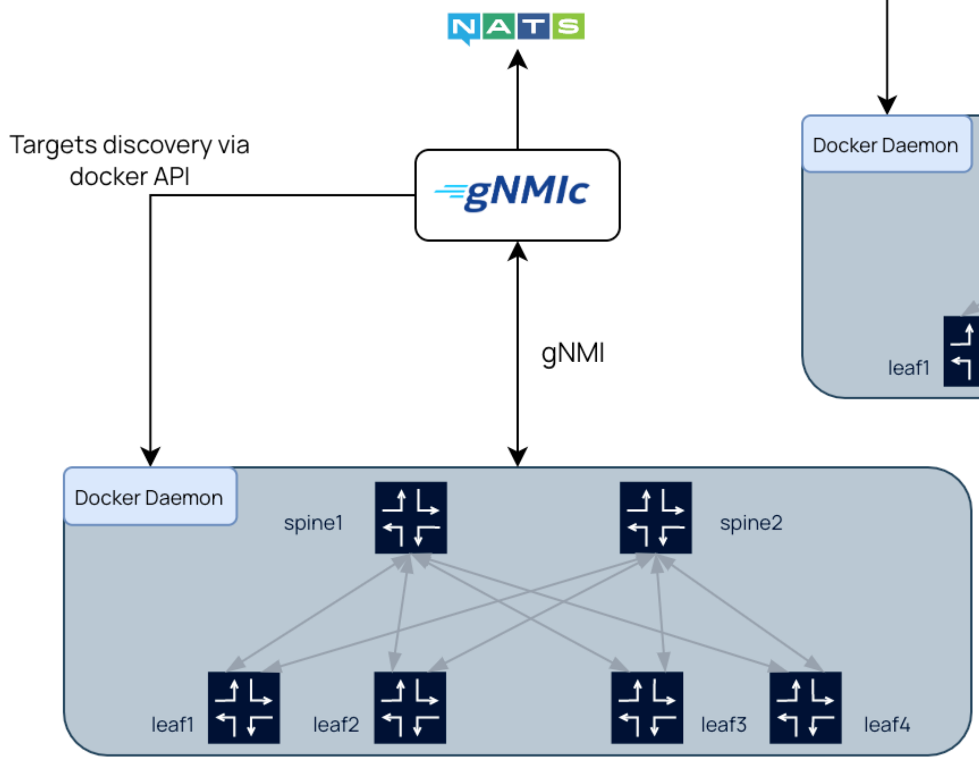
gNMIc as a combined gNMI and gRPC tunnel servers

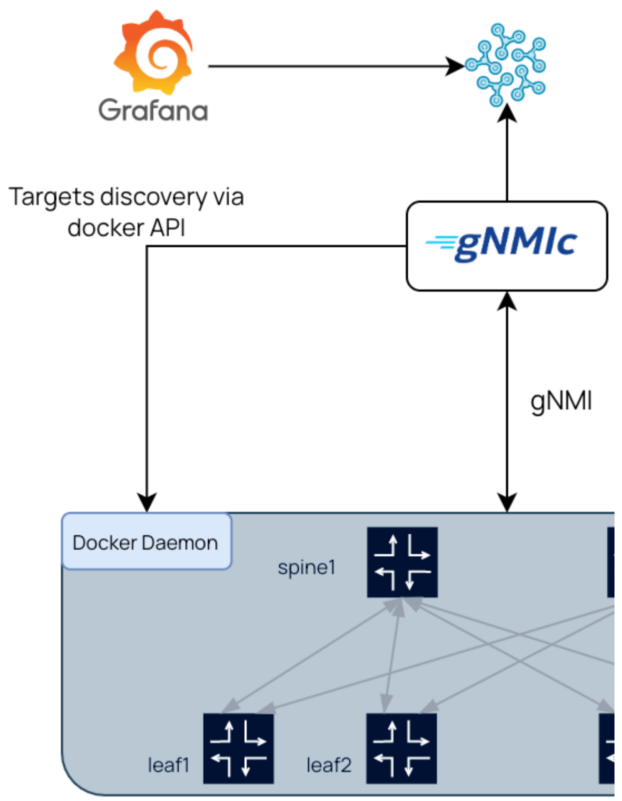


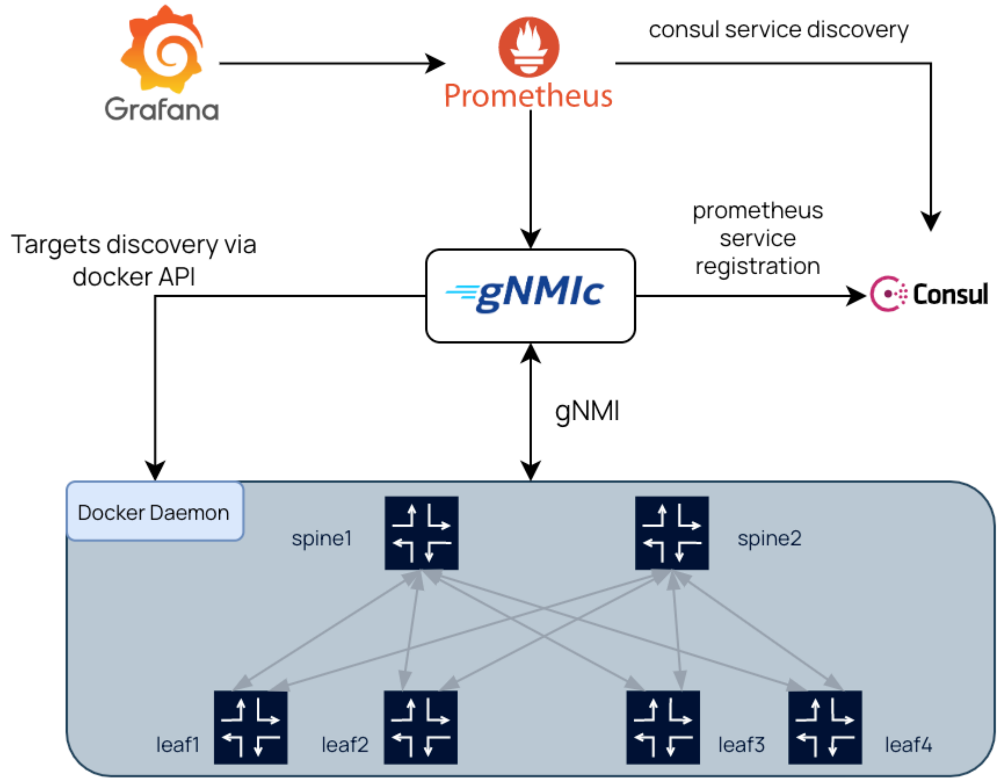
```
$ cat tunnel_server_config.yaml

tunnel-server:
  address: ":57401”

gnmi-server:
  address: ”:57400”
```

```
$ gnmic --address gnmic:57400 \
        --target router1 \
        subscribe
```

gNMIc
Deployment examples



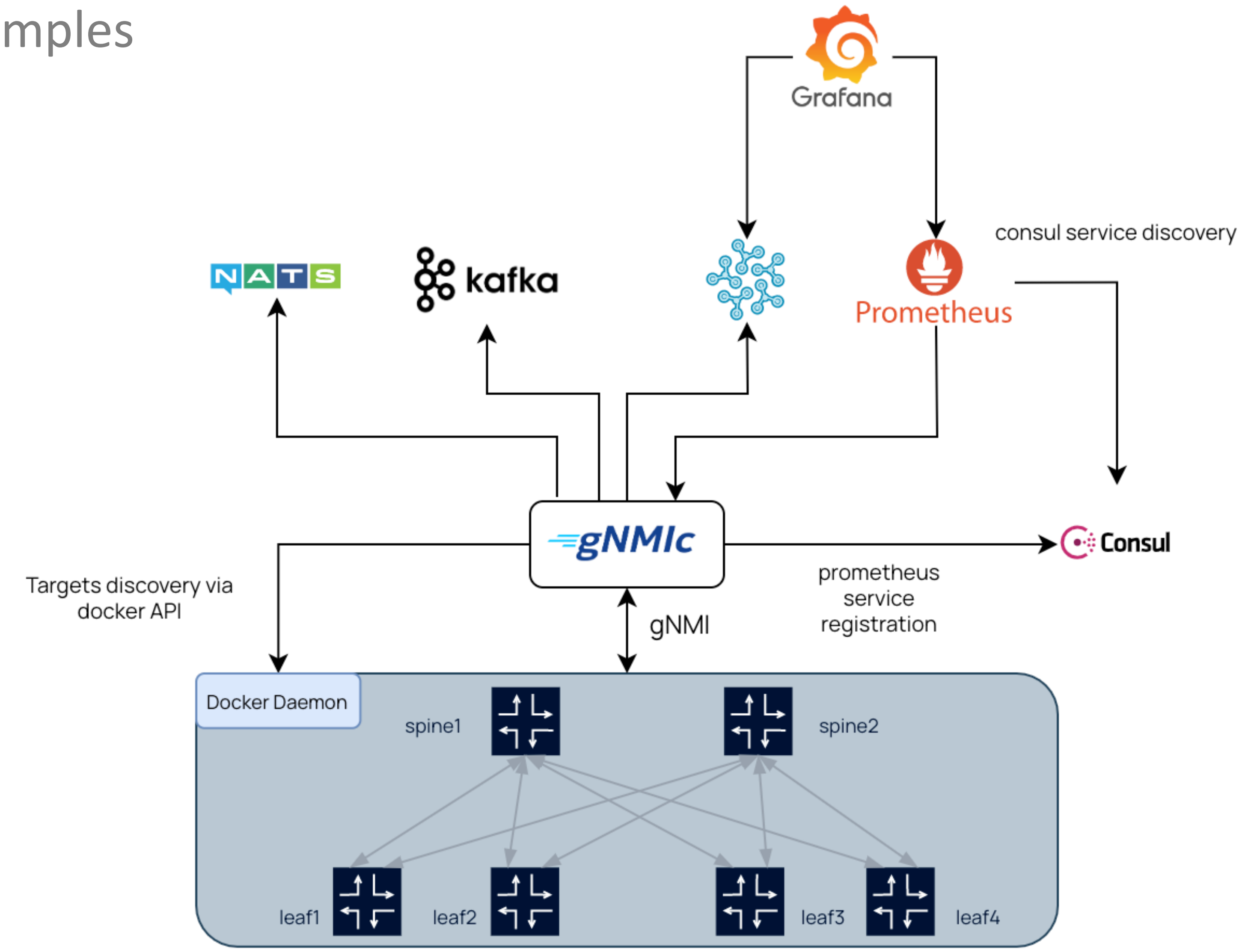




deployment examples

gNMIc

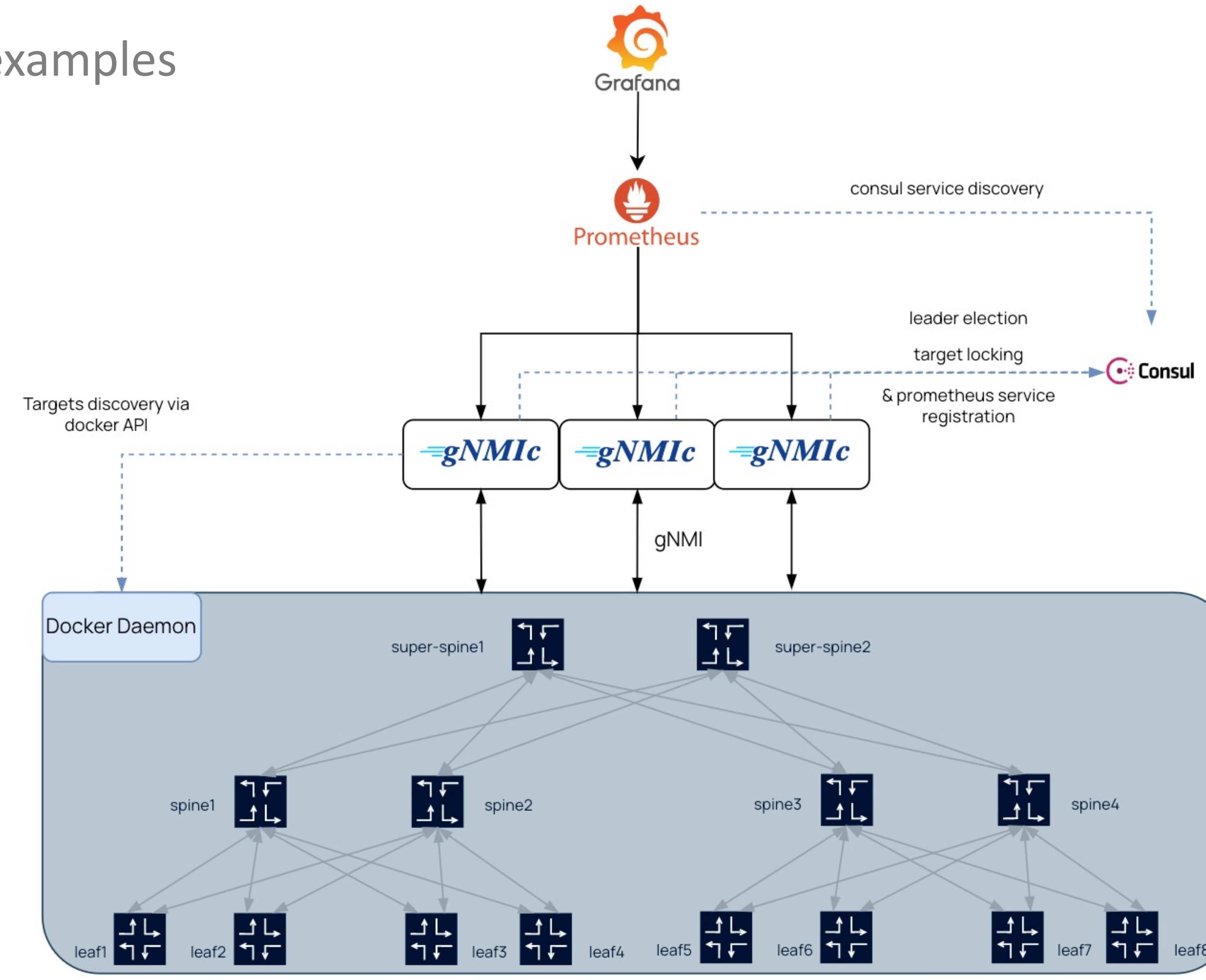
Deployment examples



deployment examples

gNMIc

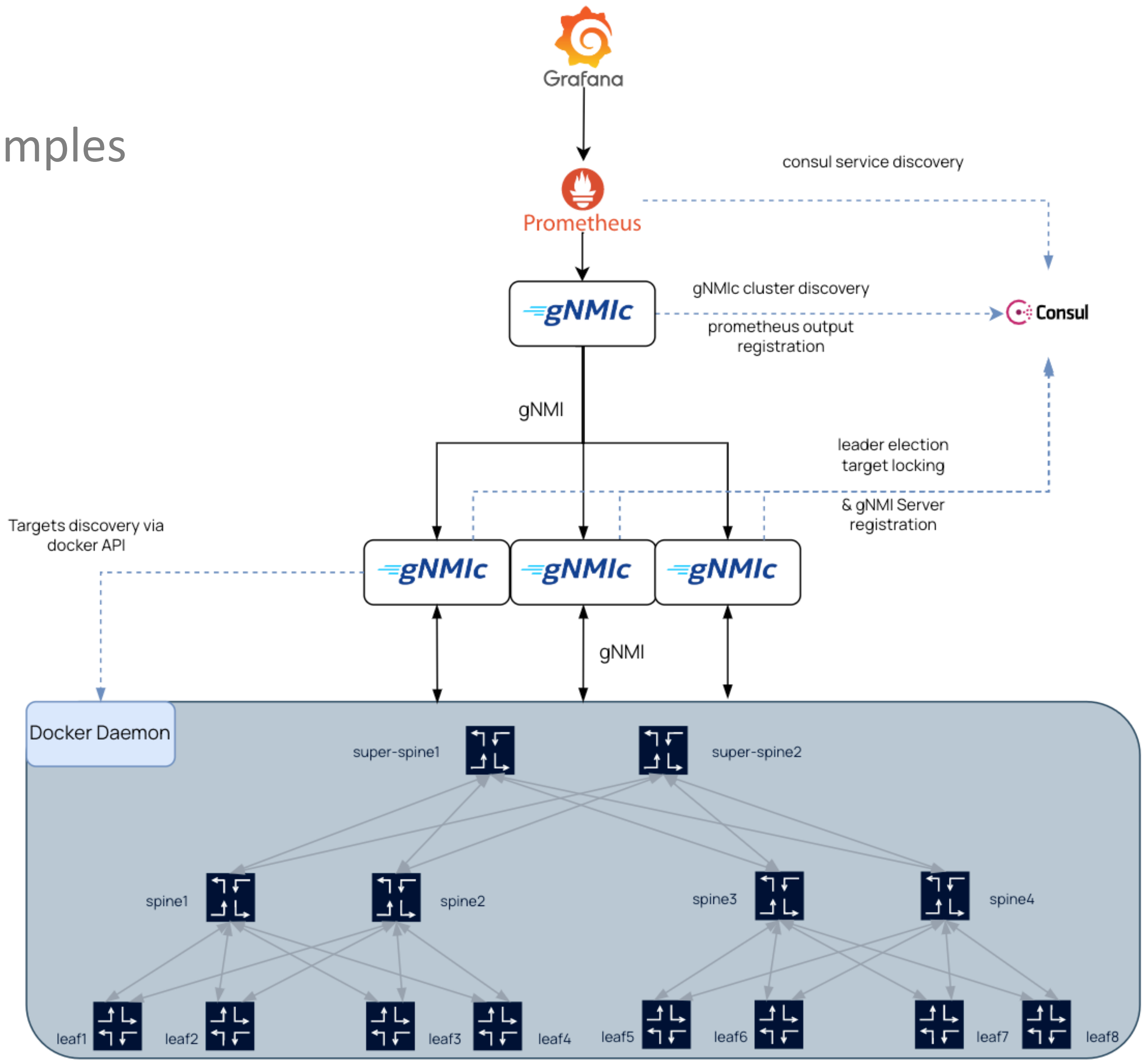
Deployment examples



[deployment examples]

gNMIc

Deployment examples



deployment examples

gNMIc

gNMI Golang API

```
import github.com/openconfig/gnmic/api
```

Create a gNMI target

```
router, err := api.NewTarget(
    api.Name("router1"),
    api.Address("10.0.0.1:57400"),
    api.Username("admin"),
    api.Password("S3cret!"),
    api.SkipVerify(true),
)
```

Create Request

```
getRequest, err := api.NewGetRequest(
    api.Encoding("json_ietf"),
    api.DataType("config"),
    api.Path("interfaces/interface"),
    api.Path("network-instances/network-instance"),
)
```

Run gNMI Get RPC

```
getResponse, err := router.Get(ctx, getRequest)
```

gNMIc

gNMI Golang API

```
import github.com/openconfig/gnmic/api
```

Create a gNMI target

```
router, err := api.NewTarget(
     api.Name("router1"),
     api.Address("10.0.0.1:57400"),
     api.Username("admin"),
     api.Password("S3cret!"),
     api.SkipVerify(true),
)
```

Create Request

```
setRequest, err := api.NewSetRequest(
     api.Update(
          api.Path("system/name"),
          api.Value("router1", "json_ietf"),
     ),
)
```

Run gNMI Set RPC

```
setResponse, err := router.Set(ctx, setRequest)
```

gNMIc

gNMI Golang API

```
import github.com/openconfig/gnmic/api
```

Create a gNMI target

```
router, err := api.NewTarget(
    api.Name("router1"),
    api.Address("10.0.0.1:57400"),
    api.Username("admin"),
    api.Password("S3cret!"),
    api.SkipVerify(true),
)
```

Create Request

```
subReq, err := api.NewSubscribeRequest(
        api.SubscriptionListMode("stream"),
        api.Subscription(
            api.Path("interfaces/interface"),
            api.SubscriptionMode("sample"),
            api.SampleInterval(10*time.Second),
        ),
)
```

Run gNMI Subscribe RPC

```
go router. Subscribe(ctx, subReq, "sub1")
defer router.StopSubscription(“sub1”)

rspCh, errCh := router.ReadSubscriptions()

for {
    select {
    case rsp := <-rspCh:
         // handle response
    case err := <-errCh:
         // handle error
    }
}
```

gNMIc

Sounds interesting, what’s next?

1 Explore gnmic.openconfig.net documentation portal

2 Try out the deployment examples

3 Missing feature, a problem, a nice idea? Reach out to us via Github Issues/Discussions

4 Give gNMIc repo a ⭐ and grab a gNMIc sticker