Driven by
Experience™

Telemetry in Junos for AI/ML Workloads

Author: Shalini Mukherjee

Table of Contents

Introduction

As AI cluster traffic requires lossless networks with high throughput and low latency, a critical element of the AI network is the collection of monitoring data. Junos Telemetry enables granular monitoring of key performance indicators, including thresholds and counters for congestion management and traffic load balancing. gRPC sessions support the streaming of telemetry data.

gRPC is a modern, open-source, high performance framework that is built on HTTP/2 transport. It empowers native bi-directional streaming capabilities and includes flexible custom-metadata in request headers.

The initial step in telemetry is to know what data is to be collected. We can then analyze this data in various formats. Once we collect the data, it is important to present it in a format that is easy to monitor, make decisions and improve the service being offered.

In this paper, we use a telemetry stack consisting of Telegraf, InfluxDB, and Grafana. This telemetry stack collects data using a push model. Traditional pull models are resource-intensive, require manual intervention, and could include information gaps in the data they collect. Push models overcome these limitations by delivering data asynchronously. They enrich the data by using user-friendly tags and names. Once the data is in a more readable format, we store it in a database and use it in an interactive visualization web application for analyzing the network. Figure. 1 shows us how this stack is designed for efficient data collection, storage, and visualization, from network devices pushing data to the collector to the data being displayed on dashboards for analysis.

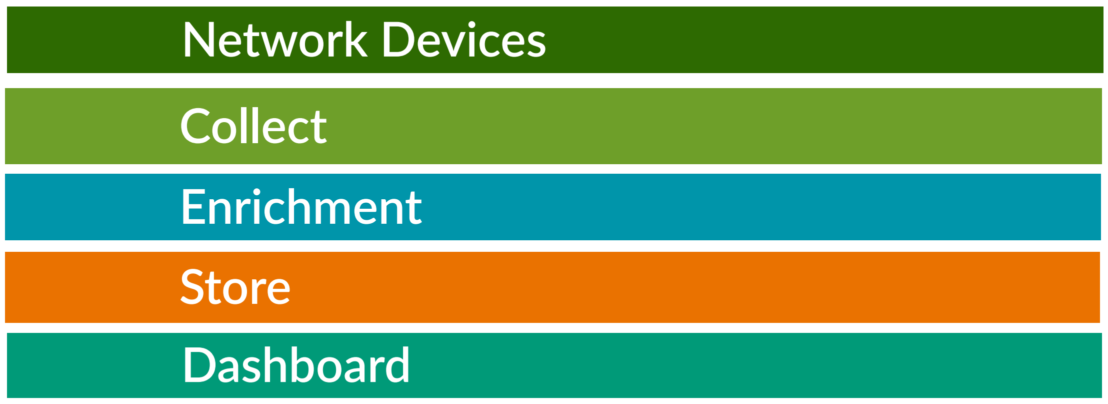


Figure 1

TIG Stack

We used an Ubuntu server to install all the software including the TIG stack.

Telegraf

To collect data, we use Telegraf on an Ubuntu server running 22.04.2. The Telegraf version running in this demo is 1.28.5. Telegraf is a plugin driven server agent for collecting and reporting metrics. It uses processor plugins to enrich and normalize the data. The output plugins are used to send this data to various data stores. In this document we use two plugins: one for openconfig sensors and the other for Juniper native sensors.

InfluxDB

To store the data in a time series database, we use InfluxDB. The output plugin in Telegraf sends the data to InfluxDB, which stores it in a highly efficient manner. We are using V1.8 as there is no CLI present for V2 and above.

Grafana

Grafana is used to visualize this data. Grafana pulls the data from InfluxDB and allows users to create rich and interactive dashboards. Here, we are running version 10.2.2.

Configuration On The Switch

To implement this stack, we first need to configure the switch as shown in Figure 2. We have used port 50051. Any port can be used here. Log in to the QFX switch and add the following configuration.

```
user@spine1> show configuration system services extension-service
request-response {
    grpc {
        clear-text {
            address 0.0.0.0;
            port 50051;
        }
        max-connections 30;
        routing-instance mgmt_junos;
        skip-authentication;
    }
}
notification {
    allow-clients {
        address 0.0.0.0/0;
    }
}
```

Figure 2

Note: This configuration is for labs/POCs as the password is transmitted in clear text. Use SSL to avoid this.

Environment

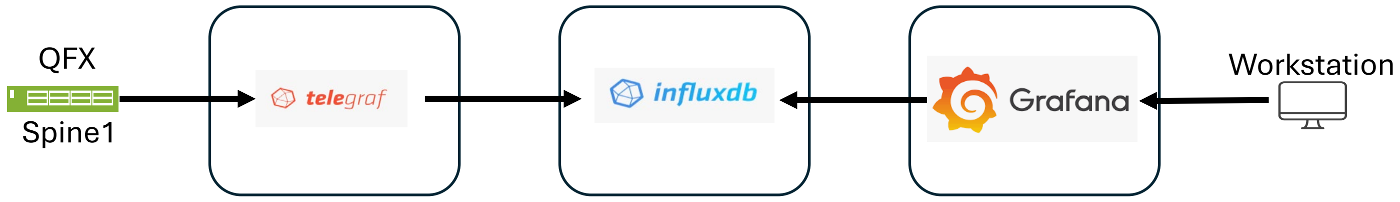


Figure 3

Nginx

This is needed if you are unable to expose the port on which Grafana is hosted. The next step is to install nginx on the Ubuntu server to serve as a reverse proxy agent. Once nginx is installed, add the lines shown in Figure 4 to the "default" file and move the file from /etc/nginx to /etc/nginx/sites-enabled.

```
user@server:/etc/nginx/sites-enabled$ cat default
# this is required to proxy Grafana Live WebSocket connections.
map $http_upgrade $connection_upgrade {
  default upgrade;
  '' close;
}

upstream grafana {
  server localhost:3000;
}

server {
  listen 80;
  root /usr/share/nginx/html;
  index index.html index.htm;

  location / {
    proxy_set_header Host $http_host;
    proxy_pass http://grafana;
  }

  # Proxy Grafana Live WebSocket connections.
  location /api/live/ {
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection $connection_upgrade;
    proxy_set_header Host $http_host;
    proxy_pass http://grafana;
  }

  location = /nginx_status {
      stub_status;
  }
}
```

Figure 4

Ensure that the firewall is adjusted to give full access to the nginx service as shown in Figure 5.

```
$ sudo ufw app list
Available applications:
  Nginx Full
  Nginx HTTP
  Nginx HTTPS
  OpenSSH
```

Figure 5

Once nginx is installed and the required changes are made, we should be able to access Grafana from a web browser by using the IP address of the Ubuntu server where all the software is installed.

There is a small glitch in Grafana that does not let you reset the default password. Use these steps if you run into this issue.

Steps to be performed on the Ubuntu server to set the password in Grafana:

- Go to /var/lib/grafana/grafana.db
- Install sqllite3
    - sudo apt install sqlite3
- Run this command on your terminal
    - sqlite3 grafana.db
- Sqlite command prompt opens; run the following query:
  >delete from user where login='admin'
- Restart grafana and type admin as username and password. It prompts for a new password.

Once all the software is installed, create the config file in Telegraf which will help pull the telemetry data from the switch and push it to InfluxDB.

Openconfig Sensor Plugin

On the Ubuntu server, edit the `/etc/telegraf/telegraf.conf` file to add all the required plugins and sensors. For the openconfig sensors, we use the gNMI plugin shown in Figure 6. For demo purposes, add the hostname as “spine1”, the port number “50051” that is used for gRPC, the username and password of the switch, and the number of seconds for redial in case of failure.

In the subscription stanza, add a unique name, “cpu” for this particular sensor, the sensor path, and the time interval for grabbing this data from the switch. Add the same plugin `inputs.gnmi` and `inputs.gnmi.subscription` for all the open config sensors. (Figure 6) .

```
[[inputs.gnmi]]
   addresses = ["spine1:50051"]
   username = "user11"
   password = "Juniper123"
   redial = "10s"

   [[inputs.gnmi.subscription]]
     name = "cpu"
     origin = "openconfig-platform"
     path = "/components/component/cpu/utilization"
     subscription_mode = "sample"
     sample_interval = "5s"
```

Figure 6

Native Sensor Plugin

This is a Juniper telemetry interface plugin used for native sensors. In the same telegraf.conf file, add the native sensor plugin `inputs.jti_openconfig_telemetry` where the fields are almost the same as openconfig. Use a unique client ID for every sensor; here, we use “telegraf3”. The unique name used here for this sensor is “mem” (Figure 7).

```
[[inputs.jti_openconfig_telemetry]]
   servers = ["spine2:50051"]
   username = "user11"
   password = "Juniper123"
   client_id = "telegraf3"
   sample_frequency = "2s"
   sensors = [
   "mem /junos/system/linecard/npu/memory/",
  ]
```

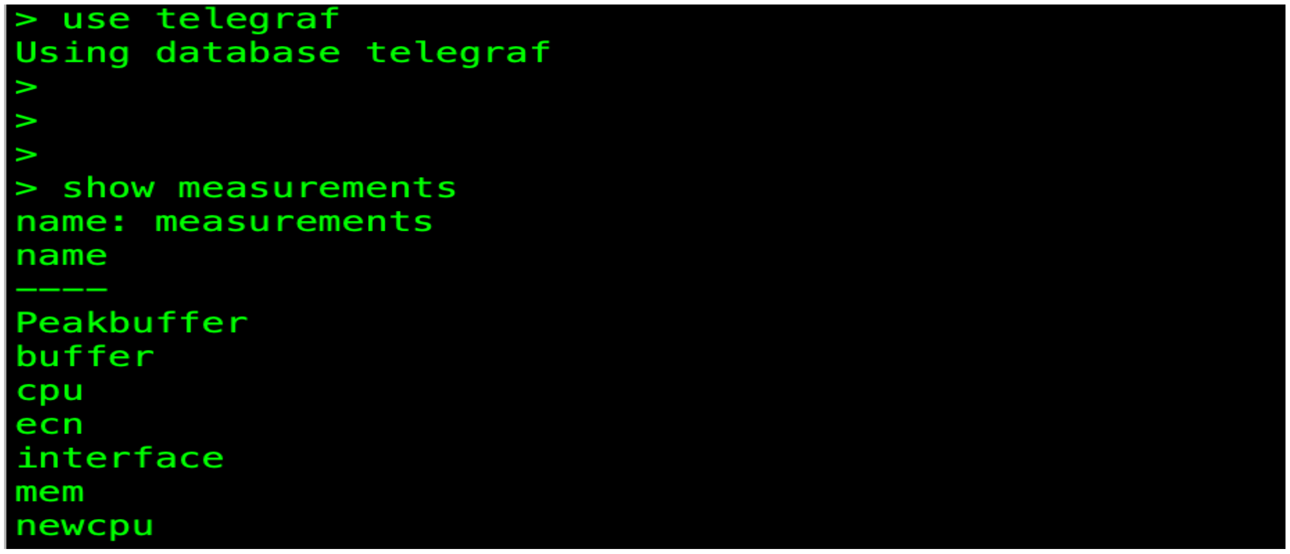
Figure 7

Lastly, add an output plugin `outputs.influxdb` to send this sensor data to InfluxDB. Here, the database is named “telegraf” with username as “influx” and password “influxdb” (Figure 8).

```
[[outputs.influxdb]]
    urls = ["http://localhost:8086"]
    database = "telegraf"
    retention_policy = "autogen"
    username = "influx"
    password = "influxdb"
    data_format = "influx"
```

Figure 8

Once you’ve edited the `telegraf.conf` file, restart the telegraf service. Now, check in the InfluxDB CLI to make sure if measurements are created for all the unique sensors. Type “influx” to enter the InfluxDB CLI.

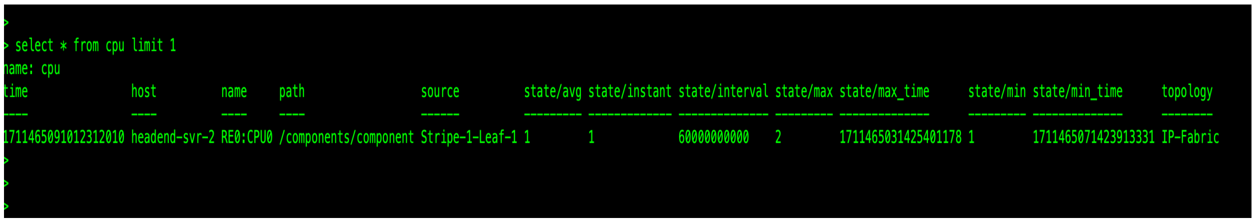


```
> use telegraf
Using database telegraf
>
>
>
> show measurements
name: measurements
name
----
Peakbuffer
buffer
cpu
ecn
interface
mem
newcpu
```

Figure 9

As seen in Figure. 9, enter the influxDB prompt and use the database “telegraf”. All the unique names given to the sensors are listed as measurements.

To see the output of any one measurement, just to make sure the telegraf file is correct and the sensor is working, use the command “select * from cpu limit 1” as shown in Figure 10.



```
>
> select * from cpu limit 1
name: cpu
time                host          name     path                    source         state/avg state/instant state/interval state/max state/max_time      state/min state/min_time      topology
----                ----          ----     ----                    ------         --------- ------------- -------------- --------- --------------      --------- --------------      --------
1711465091012312010 headend-svr-2 RE0:CPU0 /components/component Stripe-1-Leaf-1 1         1             60000000000    2         1711465031425401178 1         1711465071423913331 IP-Fabric
>
>
>
```

Figure 10

Every time changes are made to the telegraf.conf file, make sure to stop InfluxDB, restart Telegraf, and then start InfluxDB.

Log on to Grafana from the browser and create dashboards after ensuring that the data is being collected correctly.

Go to Connections > InfuxDB > Add new data source.

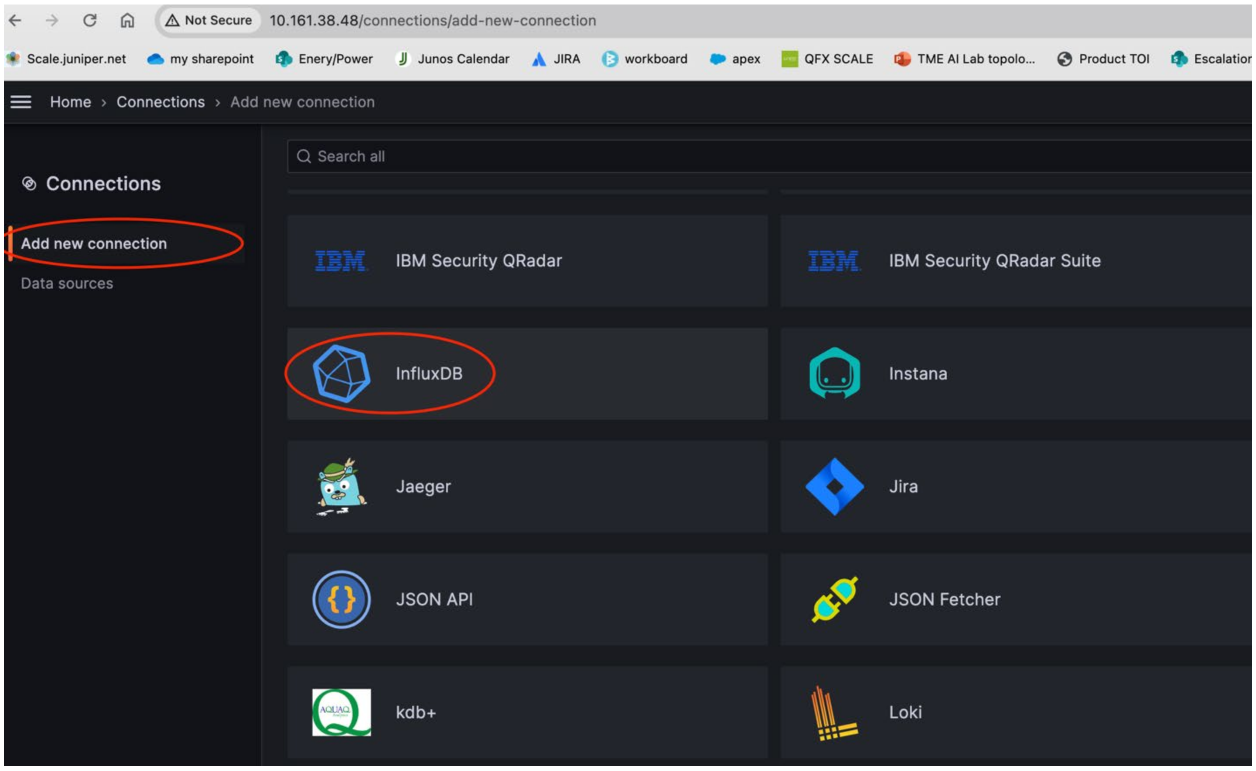
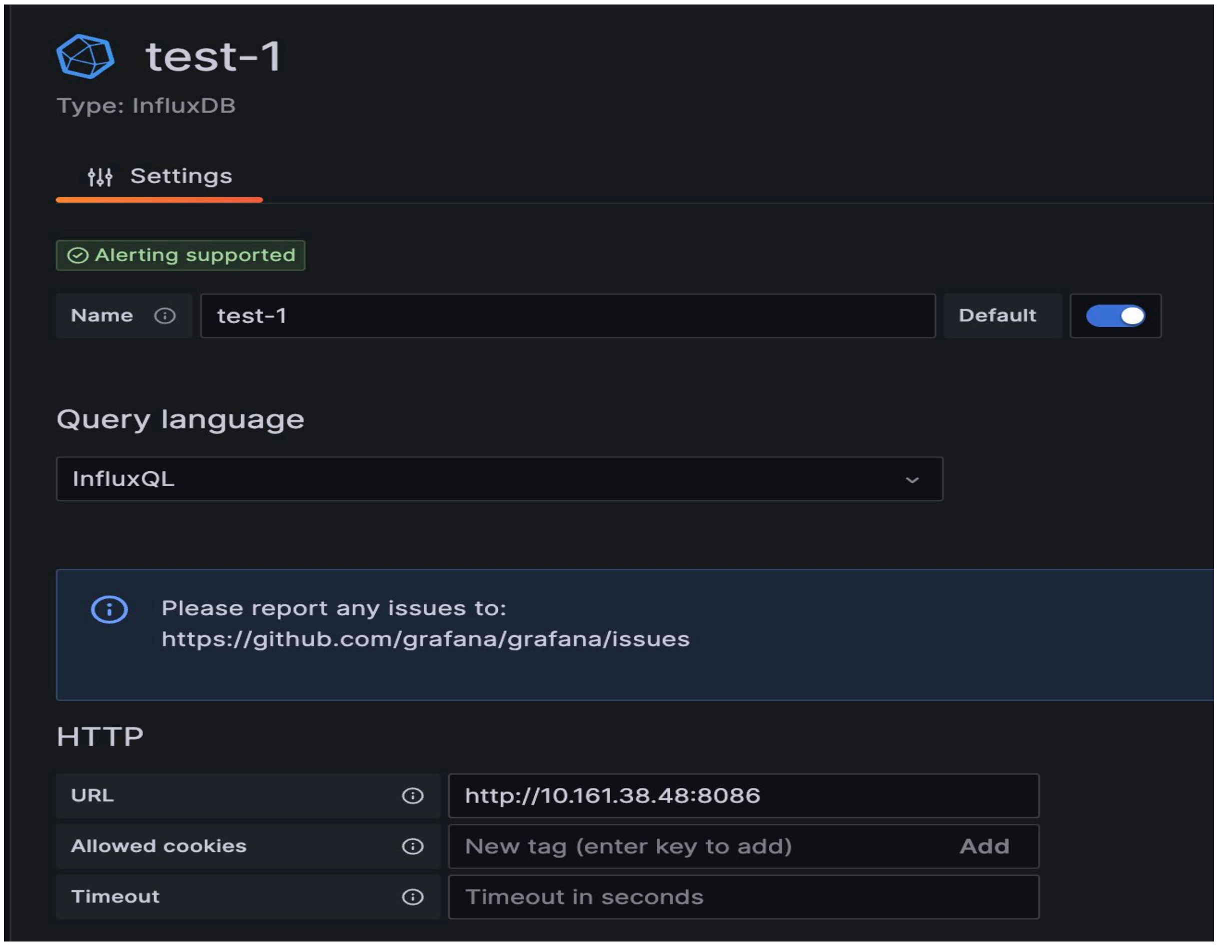


Figure 11

1. Give a name to this data source. In this demo it is “test-1”.
2. Under the HTTP stanza, use the Ubuntu server IP and 8086 port.



test-1

Type: InfluxDB

Name: test-1

Default: on

Query language

InfluxQL

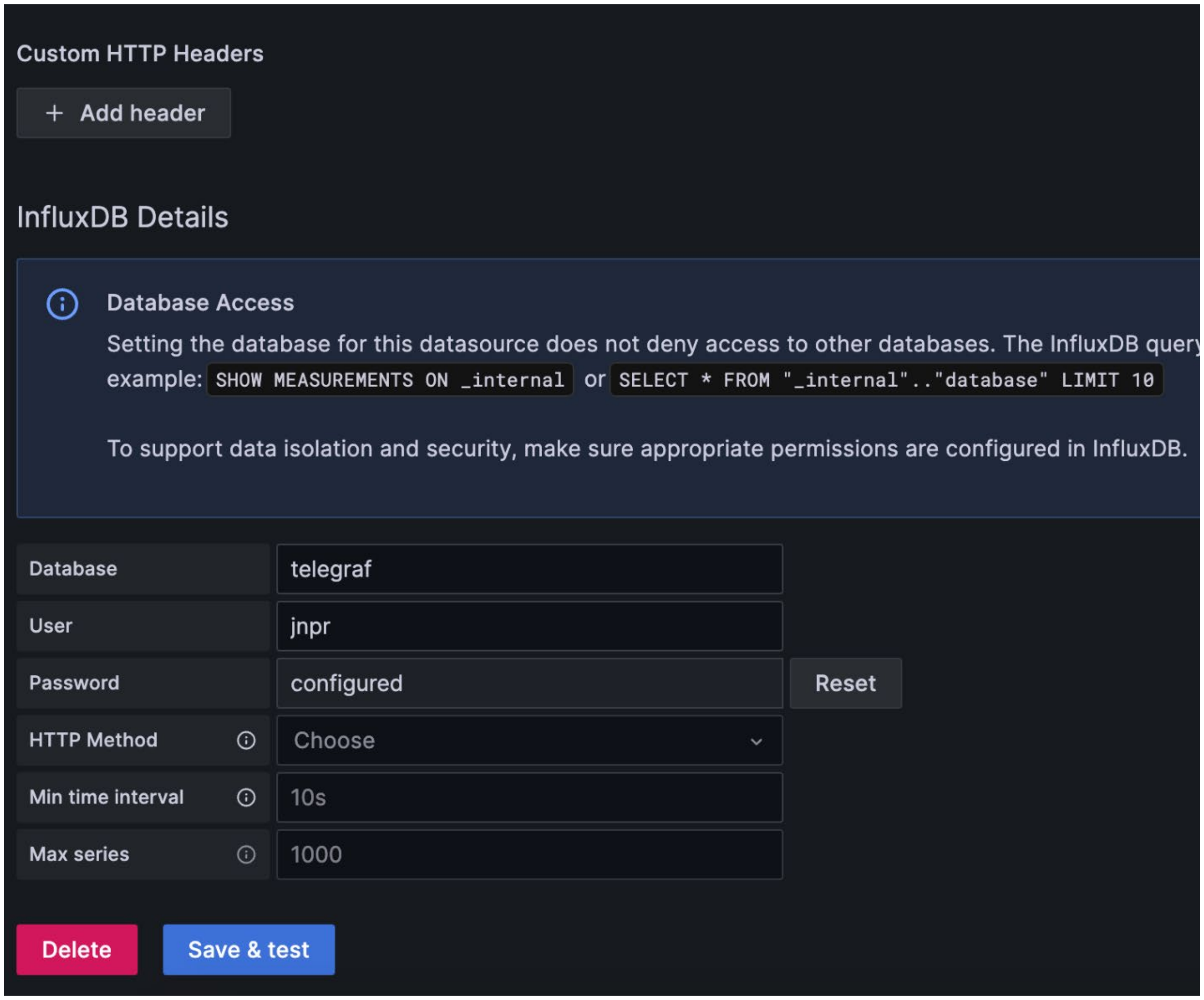
Please report any issues to:
https://github.com/grafana/grafana/issues

HTTP

URL	http://10.161.38.48:8086
Allowed cookies	New tag (enter key to add) Add
Timeout	Timeout in seconds

Figure 12

3. In the InfluxDB details, use the same database name, "telegraf," and provide the username and password of the Ubuntu server.
4. Click Save & test. Ensure that you see the message, "successful".


Custom HTTP Headers

+ Add header

InfluxDB Details

Database Access

Setting the database for this datasource does not deny access to other databases. The InfluxDB quer

example: `SHOW MEASUREMENTS ON _internal` or `SELECT * FROM "_internal".."database" LIMIT 10`

To support data isolation and security, make sure appropriate permissions are configured in InfluxDB.

Database	telegraf
User	jnpr
Password	configured
HTTP Method	Choose
Min time interval	10s
Max series	1000

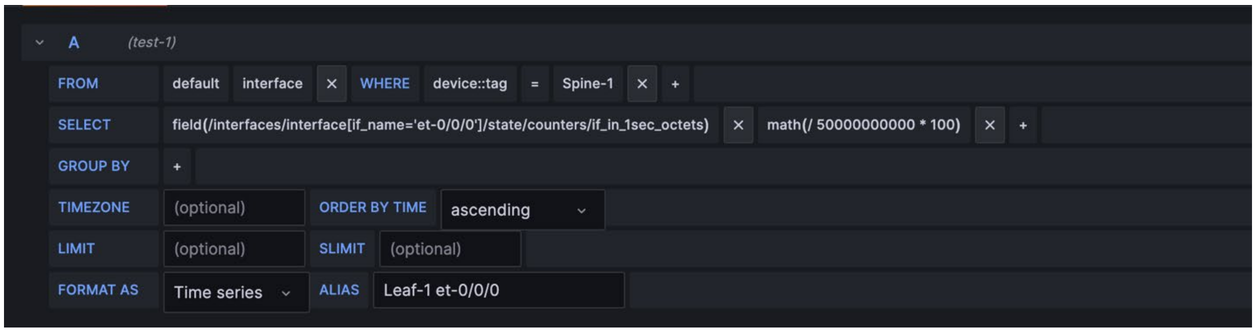
Figure 13

5. Once the data source is successfully added, go to Dashboards and click New. Let us create a few dashboards that are essential for AI/ML workloads in editor mode.

Examples Of Sensor Graphs

The following are examples of some major counters that are essential for monitoring an AI/ML network.

Percentage utilization for an ingress interface et-0/0/0 on spine-1



A (test-1)

FROM default interface WHERE device::tag = Spine-1

SELECT field(/interfaces/interface[if_name='et-0/0/0']/state/counters/if_in_1sec_octets) math(/ 50000000000 * 100)

GROUP BY +

TIMEZONE (optional) ORDER BY TIME ascending

LIMIT (optional) SLIMIT (optional)

FORMAT AS Time series ALIAS Leaf-1 et-0/0/0

Figure 14

- Select the data source as test-1.
- In the FROM section, select the measurement as "interface". This is the unique name used for this sensor path.
- In the WHERE section, select device::tag, and in the tag value, select the hostname of the switch, that is, spine-1.
- In the SELECT section, choose the sensor branch that you want to monitor; in this case choose "field*(/interfaces/interface[if_name='et-0/0/0']/state/counters/if_in_1s_octets*)". Now in the same section, click on "+" and add this calculation math (/50000000000 * 100). We are basically calculating the percentage utilization of a 400G interface.
- Make sure the FORMAT is "time-series," and name the graph in the ALIAS section.

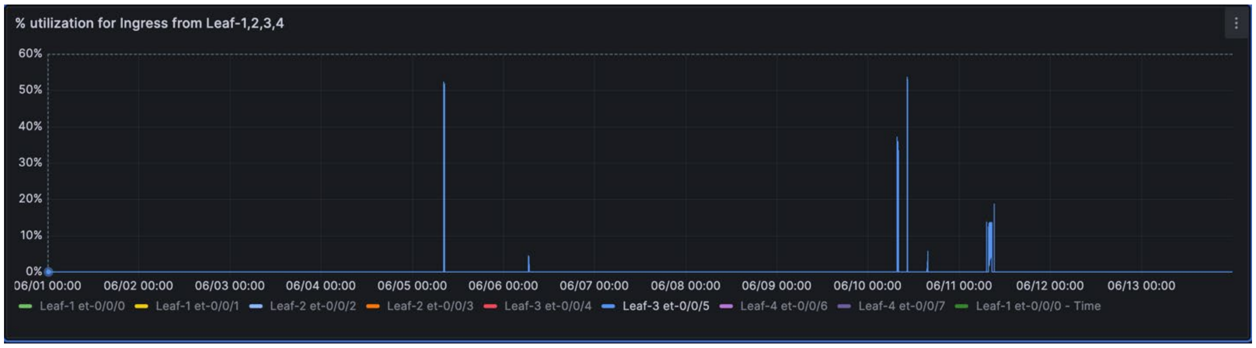
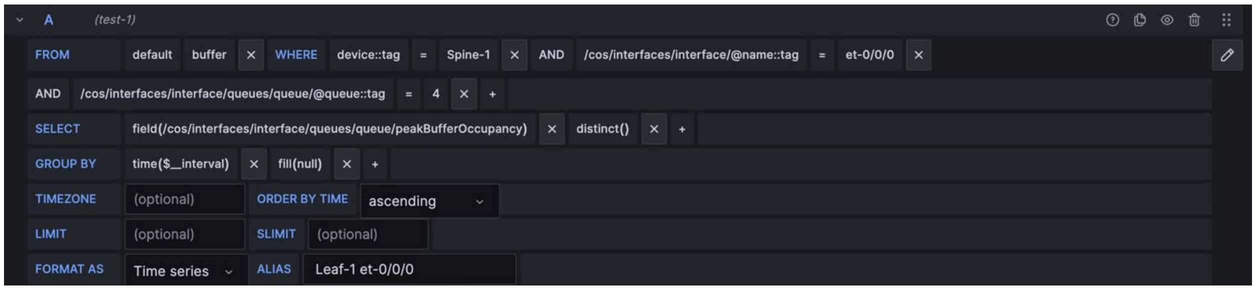


Figure 15

<u>Peak buffer occupancy for any queue</u>



A (test-1)

FROM default buffer WHERE device::tag = Spine-1 AND /cos/interfaces/interface/@name::tag = et-0/0/0

AND /cos/interfaces/interface/queues/queue/@queue::tag = 4

SELECT field(/cos/interfaces/interface/queues/queue/peakBufferOccupancy) distinct()

GROUP BY time($__interval) fill(null)

TIMEZONE (optional) ORDER BY TIME ascending

LIMIT (optional) SLIMIT (optional)

FORMAT AS Time series ALIAS Leaf-1 et-0/0/0

Figure 16

- Select the data source as test-1.
- In the FROM section, select the measurement as “buffer.”
- In the WHERE section, there are three fields to fill. Select device::tag, and in the tag value select the hostname of the switch (i.e. spine-1); AND select /cos/interfaces/interface/@name::tag and select the interface (i.e. et-0/0/0); AND select the queue as well, /cos/interfaces/interface/queues/queue/@queue::tag and choose the queue number 4.
- In the SELECT section, choose the sensor branch that you want to monitor; in this case choose “field(/cos/interfaces/interface/queues/queue/PeakBufferOccupancy).”
- Make sure the FORMAT is “time-series” and name the graph in the ALIAS section.

You can collate data for multiple interfaces on the same graph as seen in Figure 17 for et-0/0/0, et-0/0/1, et-0/0/2 etc.

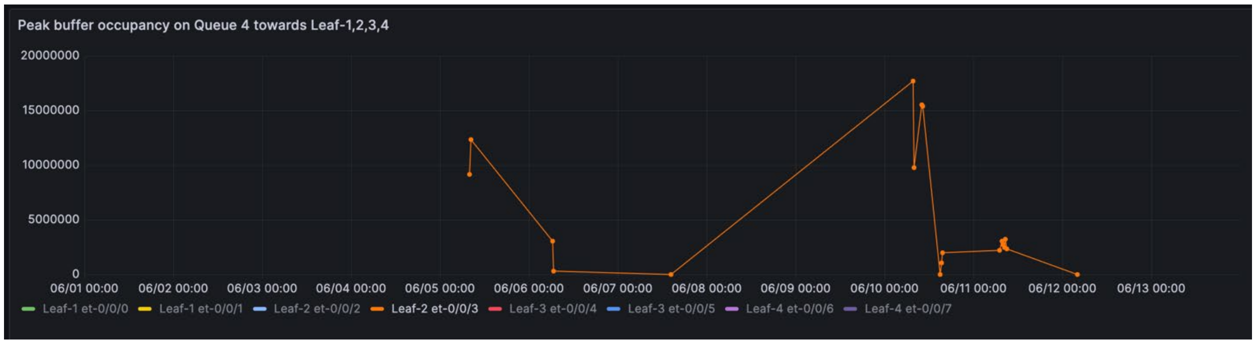
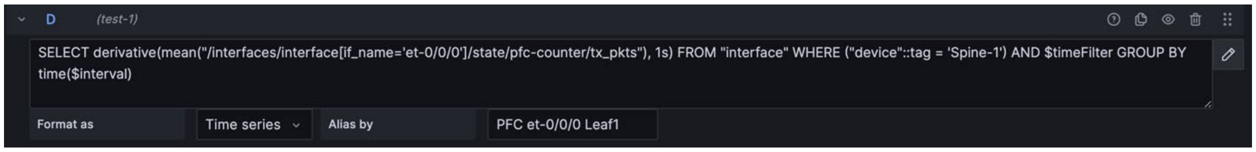


Figure 17

<u>PFC and ECN mean derivative</u>



D (test-1)

```
SELECT derivative(mean("/interfaces/interface[if_name='et-0/0/0']/state/pfc-counter/tx_pkts"), 1s) FROM "interface" WHERE ("device"::tag = 'Spine-1') AND $timeFilter GROUP BY time($interval)
```

Format as Time series Alias by PFC et-0/0/0 Leaf1

Figure 18

For finding the mean derivative (the difference in value within a time range), use the raw query mode.

This is the influx query that we have used to find the mean derivative between two PFC values on et-0/0/0 of Spine-1 in a sec.

SELECT derivative(mean("/interfaces/interface[if_name='et-0/0/0']/state/pfc-counter/tx_pkts"), 1s) FROM "interface" WHERE ("device"::tag = 'Spine-1') AND $timeFilter GROUP BY time($interval)

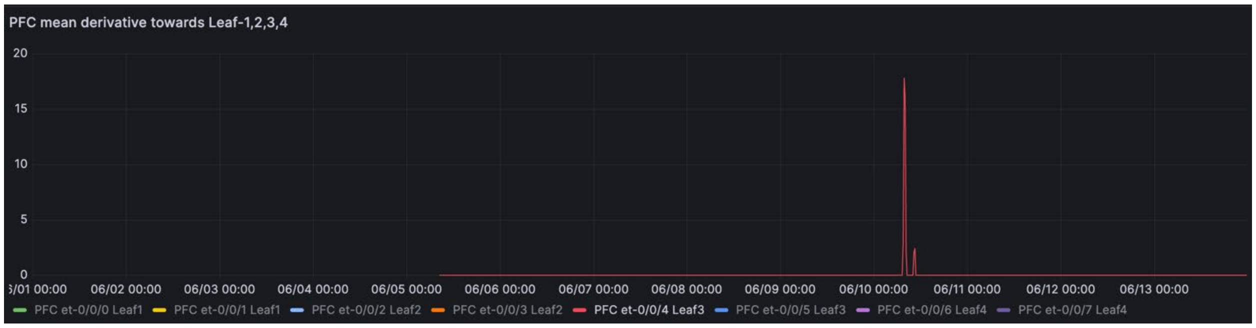
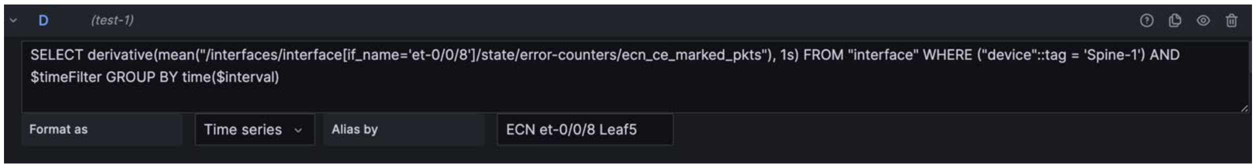


Figure 19

Similarly for ECN



```
SELECT derivative(mean("/interfaces/interface[if_name='et-0/0/8']/state/error-counters/ecn_ce_marked_pkts"), 1s) FROM "interface" WHERE ("device"::tag = 'Spine-1') AND
$timeFilter GROUP BY time($interval)
```

Alias by: ECN et-0/0/8 Leaf5

Figure 20

SELECT derivative(mean("/interfaces/interface[if_name='et-0/0/8']/state/error-counters/ecn_ce_marked_pkts"), 1s) FROM "interface" WHERE ("device"::tag = 'Spine-1') AND $timeFilter GROUP BY time($interval)

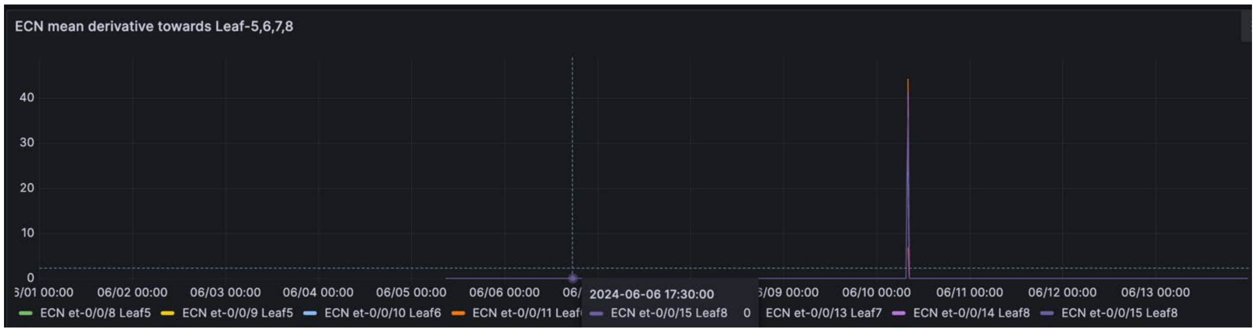
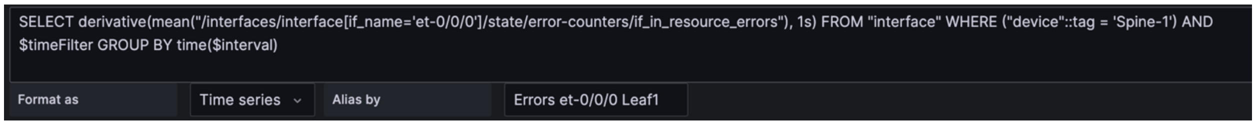


Figure 21

Input resource errors mean derivative



```
SELECT derivative(mean("/interfaces/interface[if_name='et-0/0/0']/state/error-counters/if_in_resource_errors"), 1s) FROM "interface" WHERE ("device"::tag = 'Spine-1') AND
$timeFilter GROUP BY time($interval)
```

Alias by: Errors et-0/0/0 Leaf1

Figure 22

The raw query for resource errors mean derivative is:

SELECT derivative(mean("/interfaces/interface[if_name='et-0/0/0']/state/error-counters/if_in_resource_errors"), 1s) FROM "interface" WHERE ("device"::tag = 'Spine-1') AND $timeFilter GROUP BY time($interval)

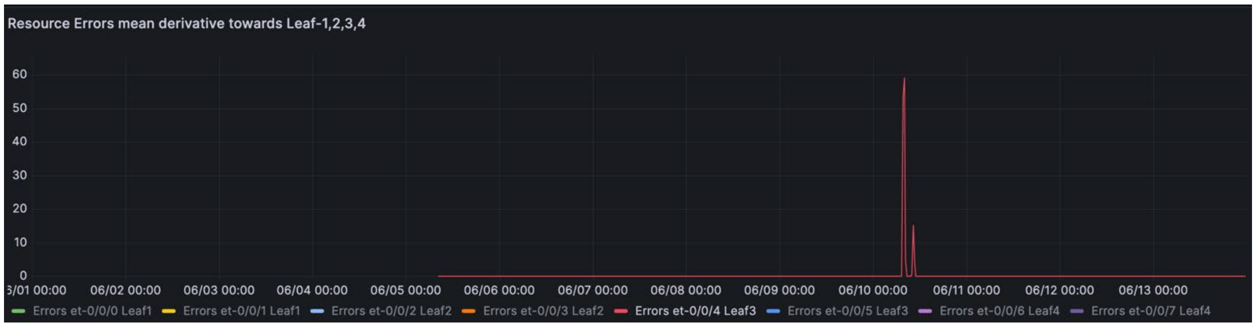
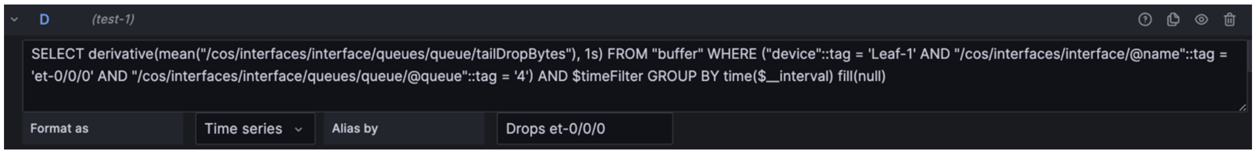


Figure 23

Tail drops mean derivative



```
SELECT derivative(mean("/cos/interfaces/interface/queues/queue/tailDropBytes"), 1s) FROM "buffer" WHERE ("device"::tag = 'Leaf-1' AND "/cos/interfaces/interface/@name"::tag = 'et-0/0/0' AND "/cos/interfaces/interface/queues/queue/@queue"::tag = '4') AND $timeFilter GROUP BY time($__interval) fill(null)
```

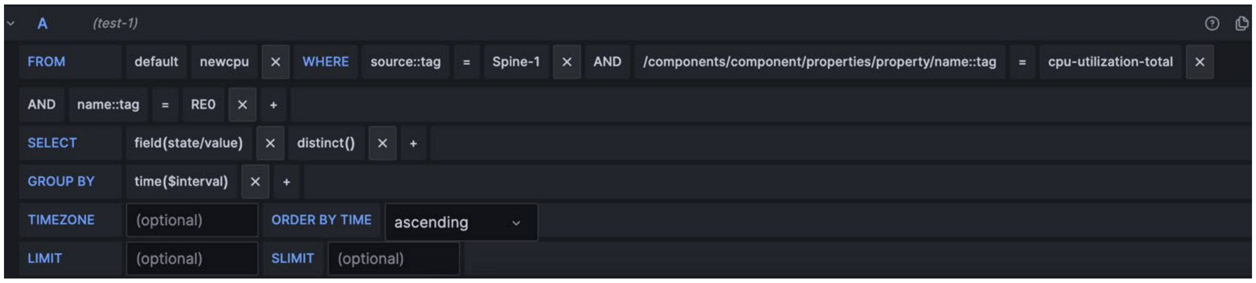
Format as: Time series
Alias by: Drops et-0/0/0

Figure 24

The raw query for tail drops mean derivative is:

SELECT derivative(mean("/cos/interfaces/interface/queues/queue/tailDropBytes"), 1s) FROM "buffer" WHERE ("device"::tag = 'Leaf-1' AND *"/cos/interfaces/interface/@name"::tag* = 'et-0/0/0' AND "/ *cos/interfaces/interface/queues/queue/@queue"::tag* = '4') AND $timeFilter GROUP BY time($__interval) fill(null)

CPU utilization



FROM default newcpu WHERE source::tag = Spine-1 AND /components/component/properties/property/name::tag = cpu-utilization-total AND name::tag = RE0
SELECT field(state/value) distinct()
GROUP BY time($interval)
TIMEZONE (optional) ORDER BY TIME ascending
LIMIT (optional) SLIMIT (optional)

Figure 25

- Select the data source as test-1.
- In the FROM section, select the measurement as "newcpu"
- In the WHERE, there are three fields to fill. Select device::tag and in the tag value select the hostname of the switch (i.e. spine-1). AND in */components/component/properties/property/name:tag,* and select cpu-utilization-total AND in name::tag select RE0.
- In the SELECT section, choose the sensor branch that you want to monitor. In this case, choose "field(*state/value*)".

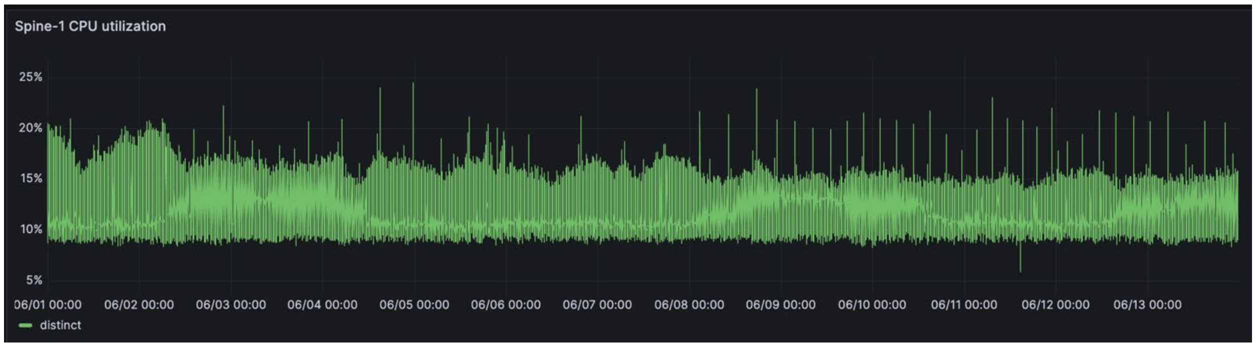
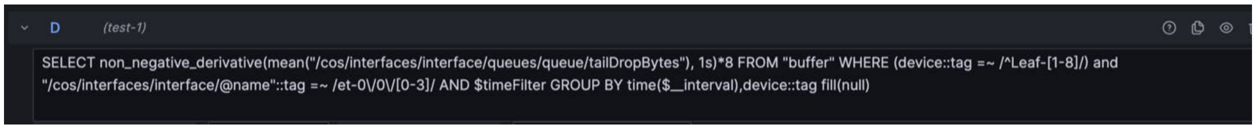


Figure 26

Tail drops non-negative derivative in bits/sec



```
SELECT non_negative_derivative(mean("/cos/interfaces/interface/queues/queue/tailDropBytes"), 1s)*8 FROM "buffer" WHERE (device::tag =~ /^Leaf-[1-8]/) and
"/cos/interfaces/interface/@name"::tag =~ /et-0\/0\/[0-3]/ AND $timeFilter GROUP BY time($__interval),device::tag fill(null)
```

Figure 27

The raw query for finding the non-negative derivative of tail drops for multiple switches on multiple interfaces in bits/sec.

*SELECT non_negative_derivative(mean("/cos/interfaces/interface/queues/queue/tailDropBytes"), 1s)*8 FROM "buffer" WHERE (device::tag =~ /^Spine-[1-2]$/) and ("/cos/interfaces/interface/@name"::tag =~ /et-0\/0\/[0-9]/ or "/cos/interfaces/interface/@name"::tag=~/et-0\/0\/1[0-5]/) AND $timeFilter GROUP BY time($__interval),device::tag fill(null)*

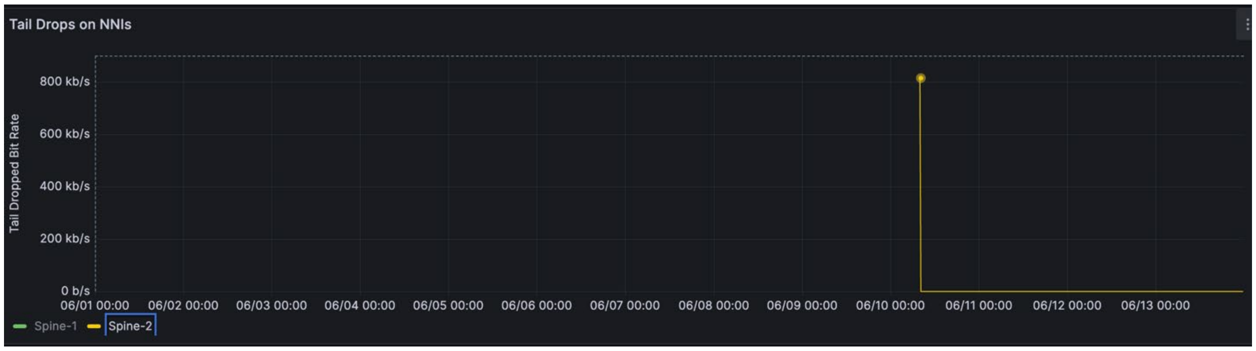


Figure 28

These were some of the examples of the graphs that can be created for monitoring an AI/ML network.

Summary

This paper illustrates the method of pulling telemetry data and visualizing it by creating graphs. This paper specifically talks about AI/ML sensors, both native and openconfig but the setup can be used for all kinds of sensors. We have also included solutions for multiple issues that you might face while creating the setup. The steps and outputs depicted in this paper are specific to the versions of the TIG stack mentioned earlier. It is subject to change depending on the version of the software, the sensors and the Junos version.

References

Juniper Yang Data Model Explorer for all sensor options
https://apps.juniper.net/ydm-explorer/

Openconfig forum for openconfig sensors
https://www.openconfig.net/projects/models/