IxOS Monitoring Solution.

Deploying the monitoring solution.

1) Clone / Download IxPortUtilizationPlotter locally

git clone https://github.com/yourusername/IxPortUtilizationPlotter.git

cd IxPortUtilizationPlotter

2) Setup the credentials and setup details to start the influxDB, Graphana and Prometheus container.

- Rename .env and add relevant values in it **(**cp env.example .env)
- Add the Chassis you want to monitor inside **.env** OR set it in the **config.py**

```
# Service Ports (customize if defaults conflict with existing services)
INFLUXDB_PORT=8086
PROMETHEUS_PORT=9090
GRAFANA_PORT=3000

# InfluxDB Configuration
INFLUXDB_ADMIN_USER=admin
INFLUXDB_ADMIN_PASSWORD=admin
INFLUXDB_ORG=keysight
INFLUXDB_BUCKET=ixosChassisStatistics
INFLUXDB_TOKEN=your-super-secret-token-change-me
INFLUXDB_RETENTION=0  # 0 = infinite retention, or specify in seconds

# Grafana Configuration
GRAFANA_ADMIN_USER=admin
GRAFANA_ADMIN_PASSWORD=admin

# ========================================
# Poller Configuration (Optional)
# ========================================
# The poller runs on your HOST machine (not in Docker)
# These variables are read by config.py if set

# If not set, config.py uses its default values

# InfluxDB Connection URL for poller
# Use localhost if services run on same machine, or remote host IP
INFLUXDB_URL=http://localhost:8086


# Polling interval in seconds - This is for my influxDB to select metrics push intevals
POLLING_INTERVAL=10
⌘L to chat, ⌘K to generate
```

> Setup a strong password for influxDB(**passwords must be between 8 and 72 characters long)**

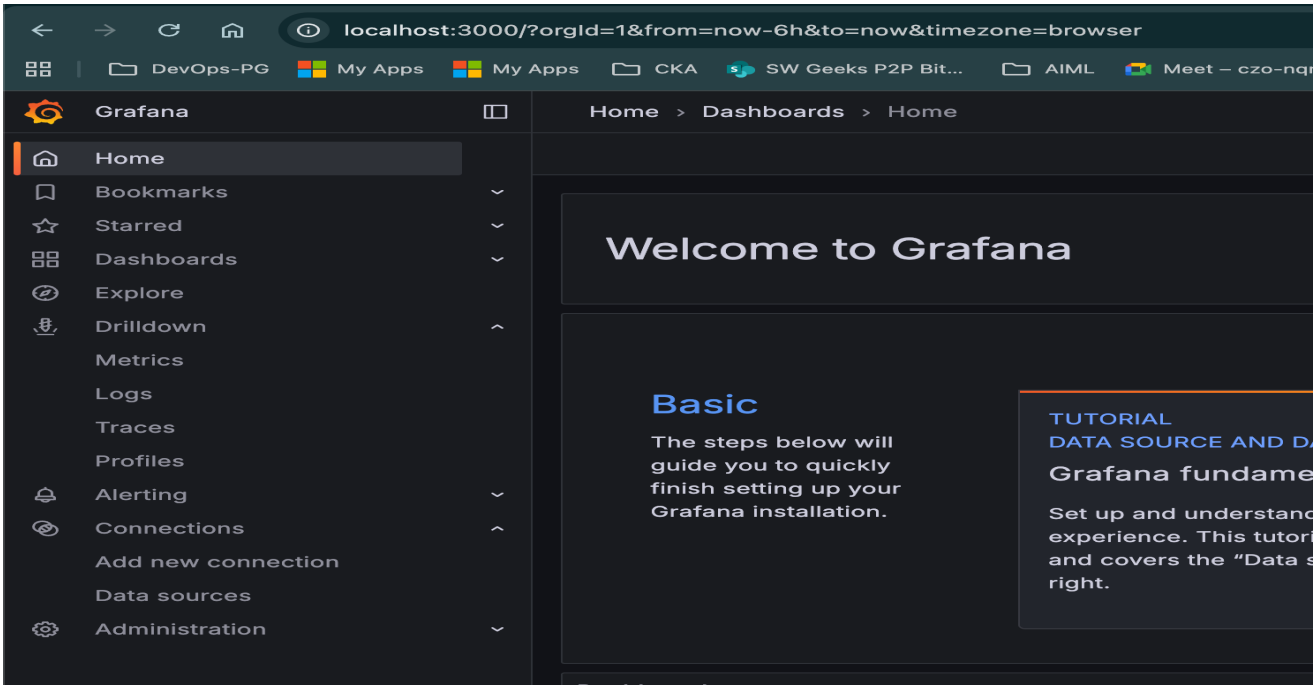
> Change the ports for the containers if there is conflict

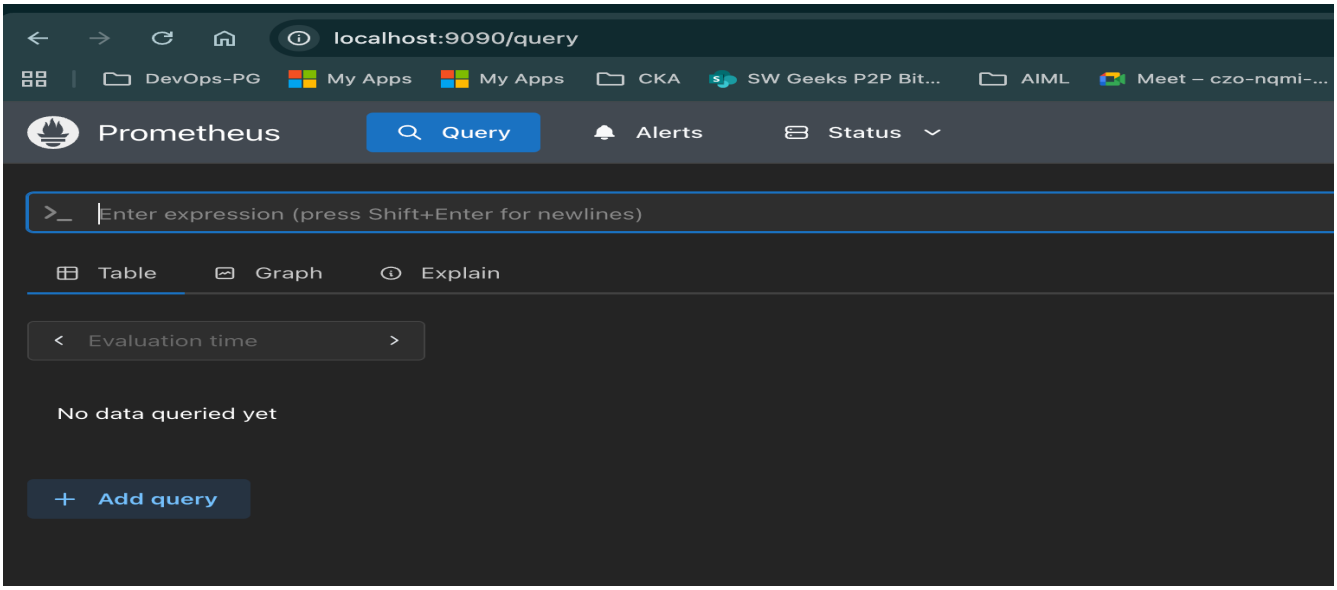
3) Start InfluxDB, Prometheus and Grafana containers from the docker-compose file withing the repo

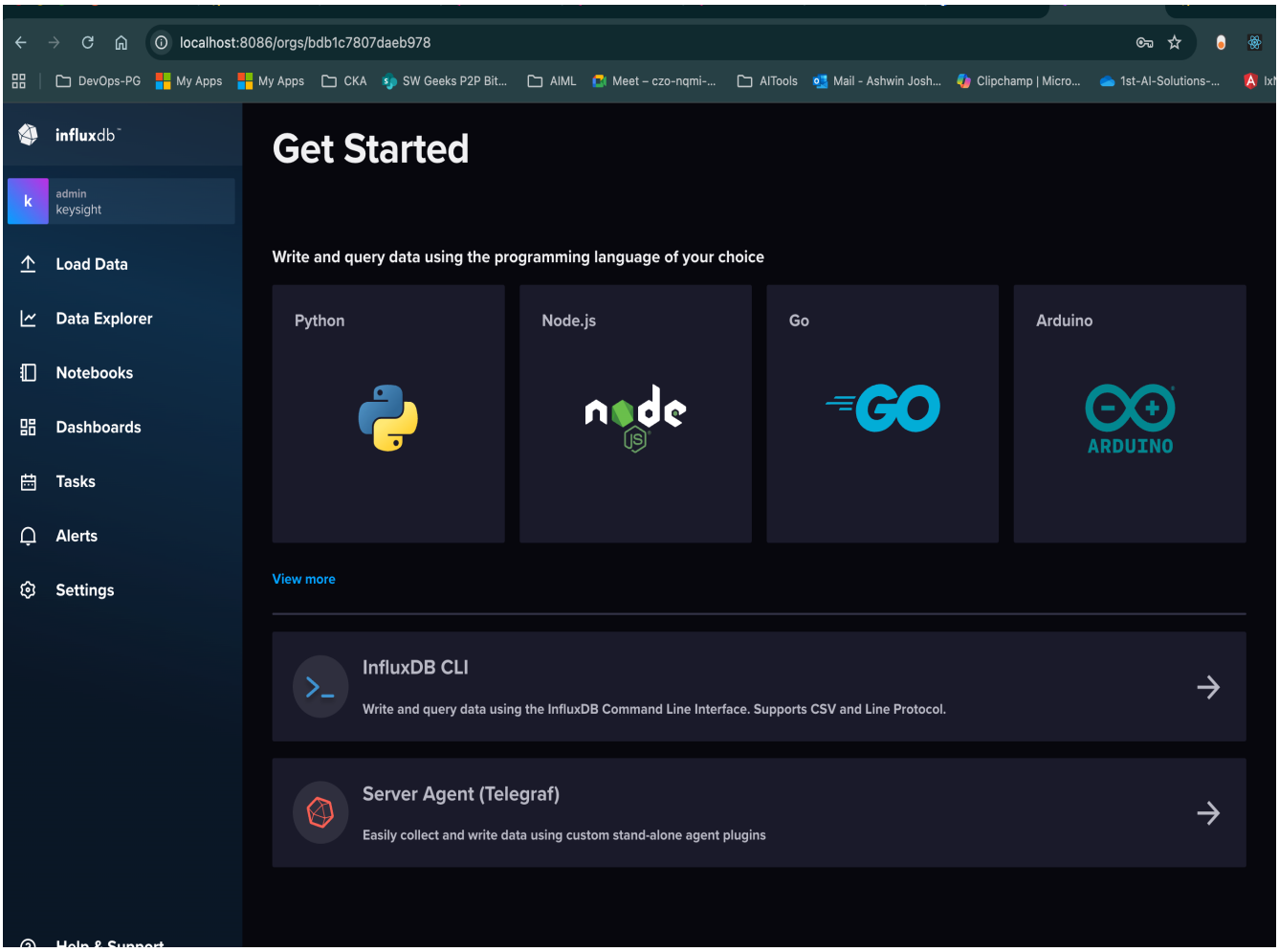
> docker compose up -d

```
ubuntusand@ubuntusand ~/IxPortUtilizationPlotter git:(main) (8.327s)
docker compose up -d
WARN[0000] /home/ubuntusand/IxPortUtilizationPlotter/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
WARN[0000] Found orphan containers ([ixportutilizationplotter-db-1]) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
[+] Running 3/3
 ✓ Container prometheus  Healthy                                                                                    2.5s
 ✓ Container influxdb    Healthy                                                                                    7.5s
 ✓ Container grafana     Started                                                                                    0.7s
```

4) Confirm that containers started OK by logging in these portals (Note the UN/PW you set in .env OR then the defaults configured)

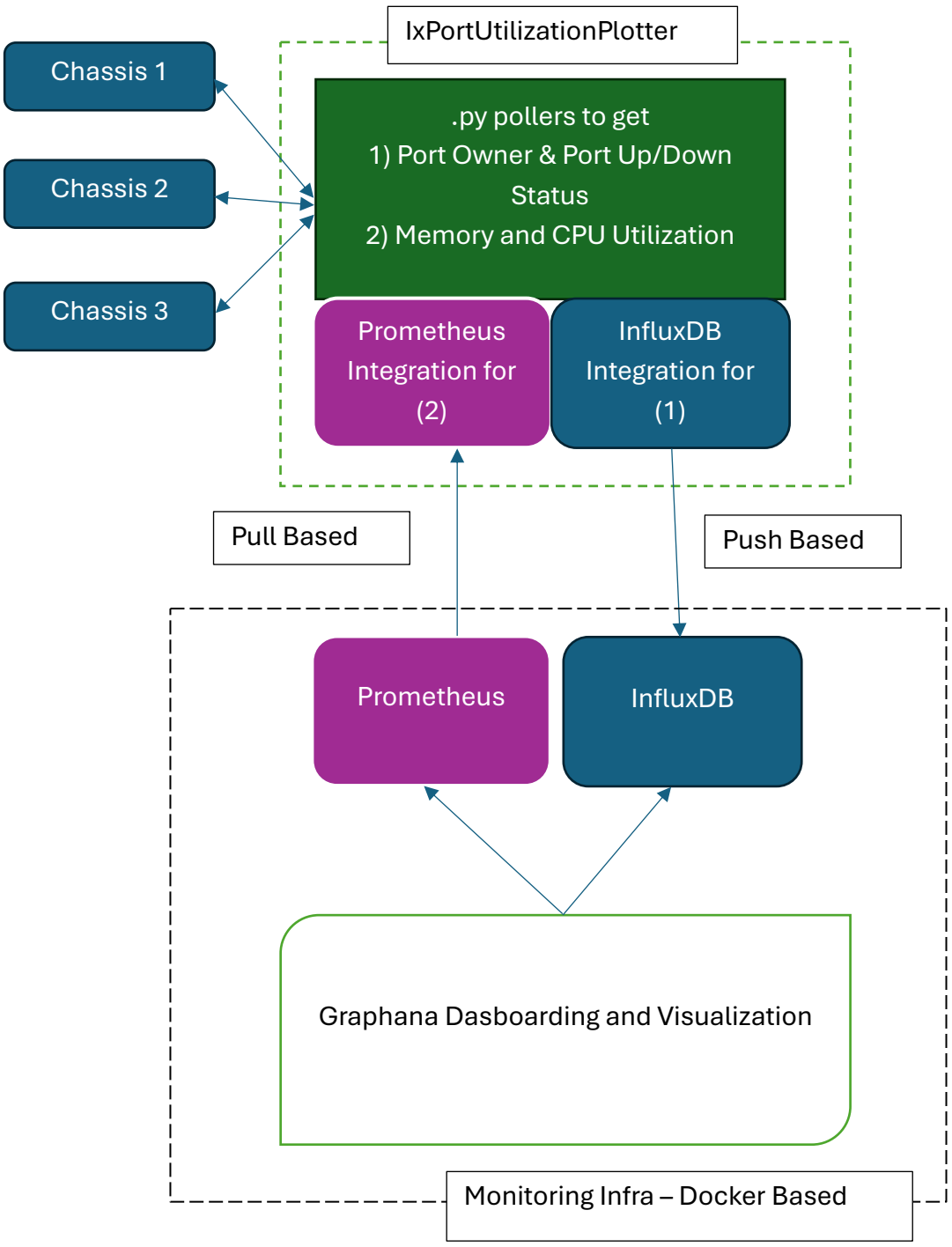

localhost:3000/?orgId=1&from=now-6h&to=now&timezone=browser
DevOps-PG
My Apps
My Apps
CKA
SW Geeks P2P Bit...
AIML
Grafana
Home > Dashboards > Home
Home
Bookmarks
Starred
Dashboards
Explore
Drilldown
Metrics
Logs
Traces
Profiles
Alerting
Connections
Add new connection
Data sources
Administration
Welcome to Grafana
Basic
The steps below will guide you to quickly finish setting up your Grafana installation.
TUTORIAL
Grafana fundame
Set up and understanc
experience. This tutori
and covers the "Data
right.


localhost:9090/query
DevOps-PG
My Apps
My Apps
CKA
SW Geeks P2P Bit...
AIML
Prometheus
Query
Alerts
Status
Enter expression (press Shift+Enter for newlines)
Table
Graph
Explain
Evaluation time
No data queried yet
Add query



4) Metrics Collection: We will be taking **Hybrid approach** here, using Prometheus (for numeric metrics) and InfluxDB (for string-based metrics)

Solution Architecture



Feature	Prometheus	InfluxDB
Data model	Metrics: numeric, time-series only	Time-series with both numeric and string (“tag”) support
Data source style	**Pull-based** (scrapes targets)	**Push-based** (you write data to it)
Typical use	System & application metrics (CPU, latency, errors, etc.)	Sensor data, IoT, logs, or custom app data (string or mixed types)
Query language	PromQL	InfluxQL / Flux
Storage type	Ephemeral (not long-term)	Long-term (retention policies, downsampling, etc.)

Let’s get stated with Metrics Collection:

Step 1: Create the **Automation TOKEN for you InfluxDB.** You need this to push the metric from python code into Influx DB. Prometheus needs no auth**.**

Manage tokens in the InfluxDB UI

To manage InfluxDB API Tokens in the InfluxDB UI, navigate to the **API Tokens** management page.

In the navigation menu on the left, select **Load Data** > **API Tokens**.

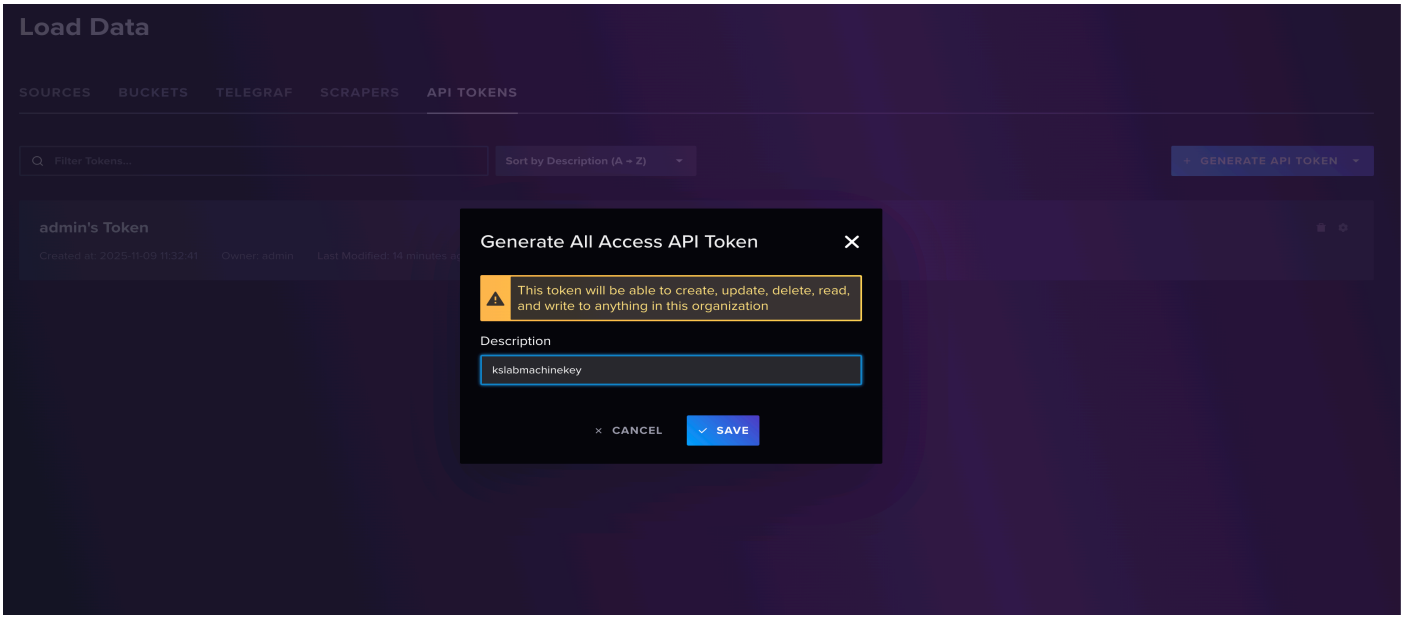
Create a token in the InfluxDB UI

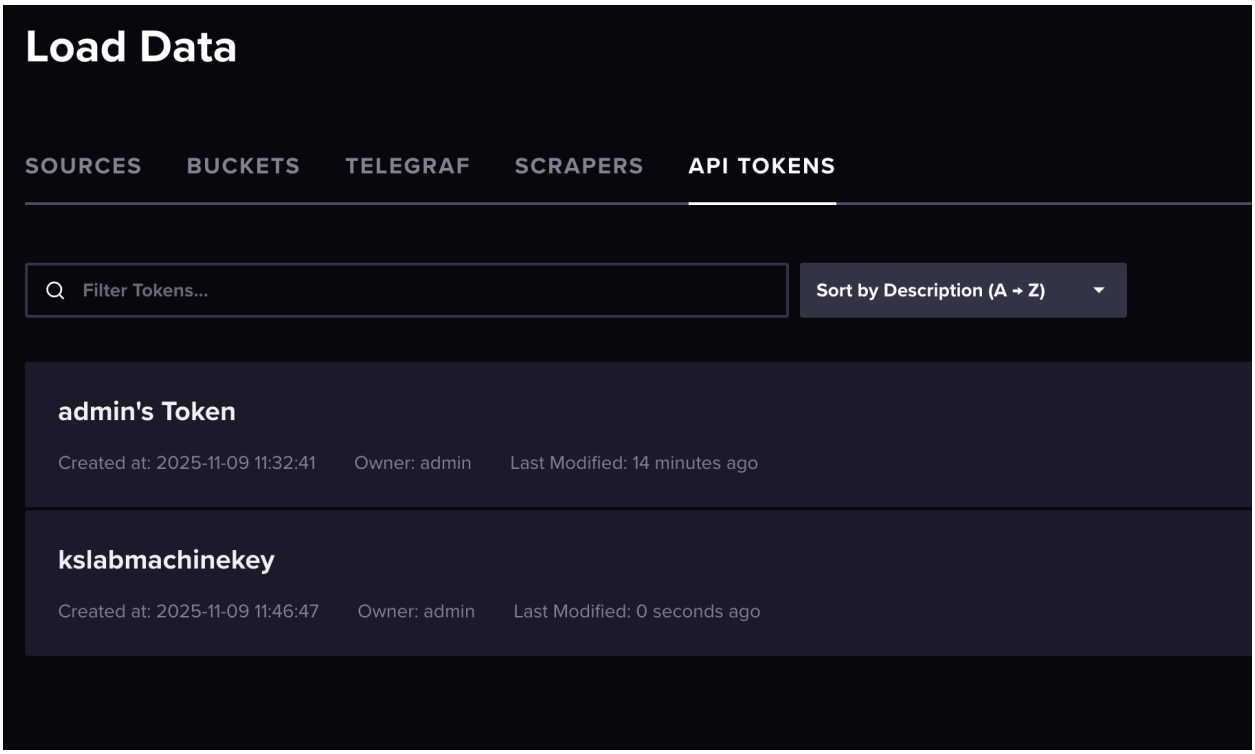
Create an All Access token

1. From the API Tokens management page, click the + GENERATE API TOKEN button.
2. Select **All Access API Token**.

Create a custom token

1. From the API Tokens management page, click the + GENERATE API TOKEN button.
2. Select **Custom API Token**.
3. When the **Generate a Personal API Token** window appears, enter a description. If you don't provide a description for the token, InfluxDB will generate a description from the permissions you assign. For example, if you select **Read** for a bucket named "_monitoring" and **Write** for a bucket named "_tasks", InfluxDB will generate the description "Read buckets _monitoring Write buckets _tasks".
4. Select the check boxes in the **Read** and **Write** columns to assign access permissions for the token. You can enable access to all buckets, individual buckets, Telegraf configurations, and other InfluxDB resources. By default, the new token has no access permissions.
5. When you're finished, click GENERATE.
6. When InfluxDB displays the token value, click COPY TO CLIPBOARD. This is your only chance to access and copy the token value from InfluxDB.





Once you get the API TOKEN add it in the .env file

```
# InfluxDB Configuration
INFLUXDB_ADMIN_USER=admin
INFLUXDB_ADMIN_PASSWORD=Keysight12345!
INFLUXDB_ORG=keysight
INFLUXDB_BUCKET=ixosChassisStatistics
INFLUXDB_TOKEN='eegHpR9kkgxg5KG7rklj2zQI86-5z7yNETx0P0qQpSnw1owDxSL5IF-uQruOP-J8M_xmrhT3KWECh-QGbsdyYA=='
INFLUXDB_RETENTION=0  # 0 = infinite retention, or specify in seconds
```

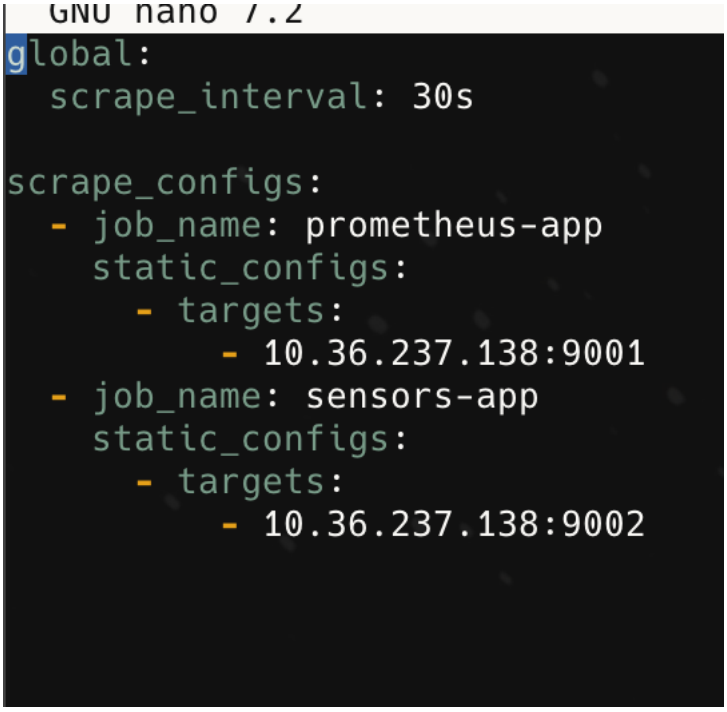
Add the chassis list in .env file with IP , Username and Password to monitor

```
CHASSIS_LIST = [{"ip": "10.36.75.205", "username": "admin", "password": "admin"}]
```

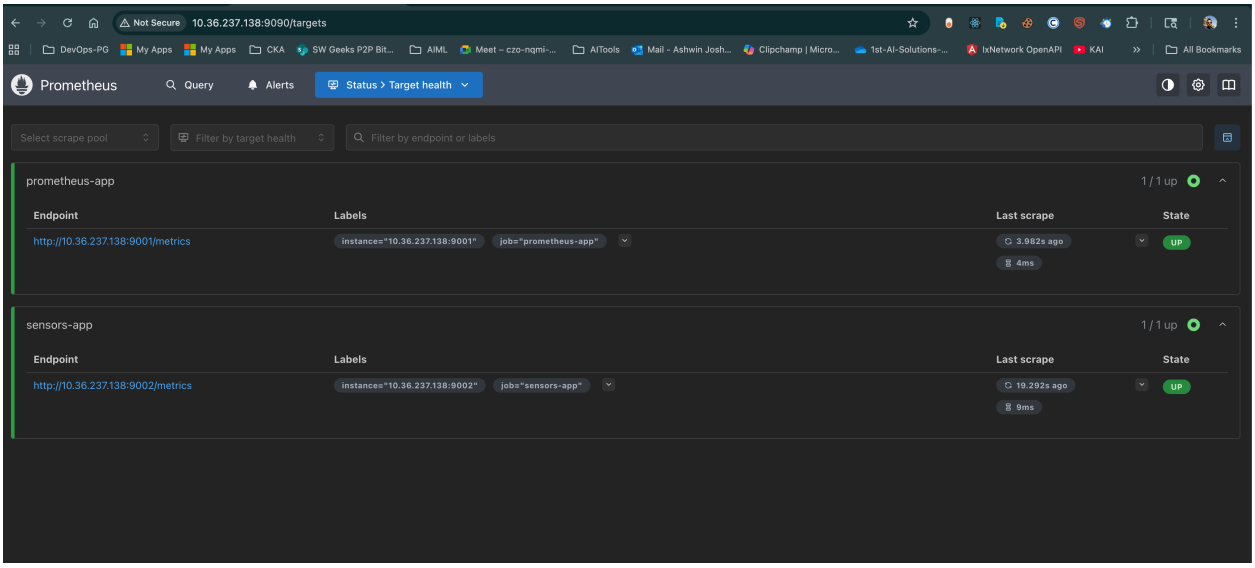
Also, add the polling intervals

```
# Polling interval in seconds - This is for my influxDB to select metrics push intevals
POLLING_INTERVAL=30
# Polling interval in seconds - This is for my prometheus to select metrics push intevals
POLLING_INTERVAL_PERF_METRICS=30
POLLING_INTERVAL_SENSOR_METRICS=30
```

Finally, modify the @prometheus.yaml with correct scrape interval and target IP. Targets are the IP on machine on which you will be running the Pollers.



Check the Prometheus targets to ensure that metrics targets configured ok.



Showtime !!!!!!

Step 1: Start all 3 pollers

chmod +x run_pollers.sh stop_pollers.sh
./run_pollers.sh

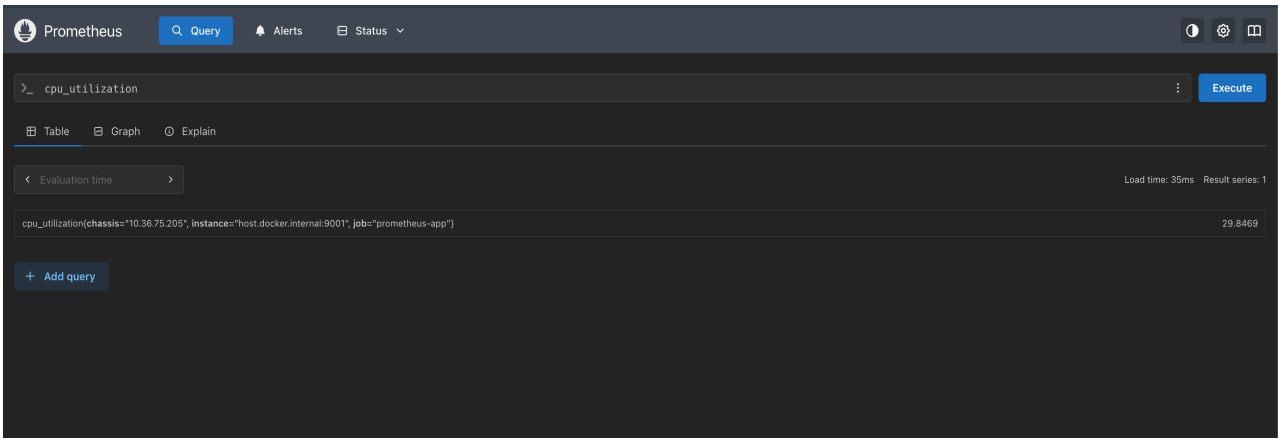
```
Starting IxOS Pollers...
========================
✓ Started portInfoPoller.py (PID: 3779636)
✓ Started perfMetricsPoller.py (PID: 3779638)
✓ Started sensorsPoller.py (PID: 3779640)

View logs:
  tail -f ./logs/portInfoPoller.log
  tail -f ./logs/perfMetricsPoller.log
  tail -f ./logs/sensorsPoller.log

Stop: sh ./stop_pollers.sh

Note: Virtual environment 'ixmon' is activated in this script.
      The pollers will run with the dependencies from this environment.
```

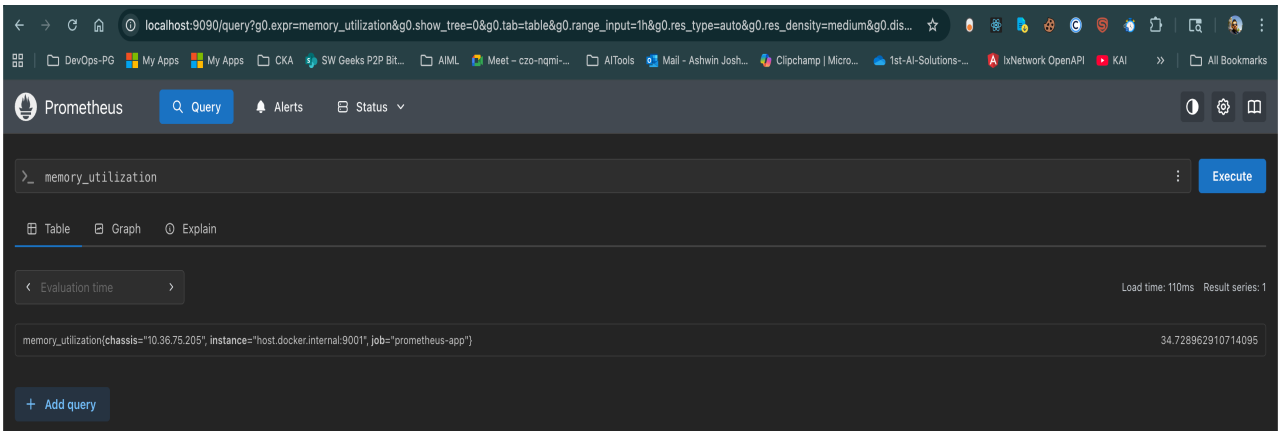
Let's confirm that data is being populated by querying prometheus , influxDB.

Wait for the

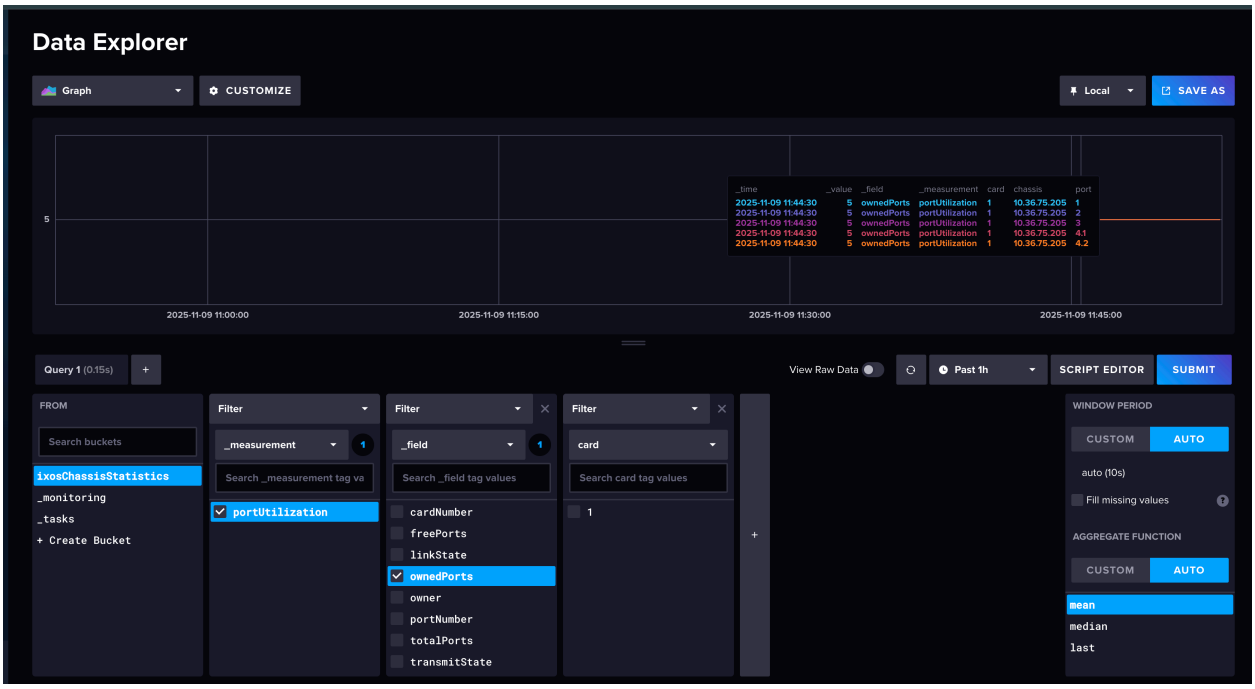
> scrape interval as per @prometheus.yaml for CPU/Mem utilization & Sensor Data
> POLLING_INTERVAL_PERF_METRICS=30 for PortUsage Metrics

In Prometheus:





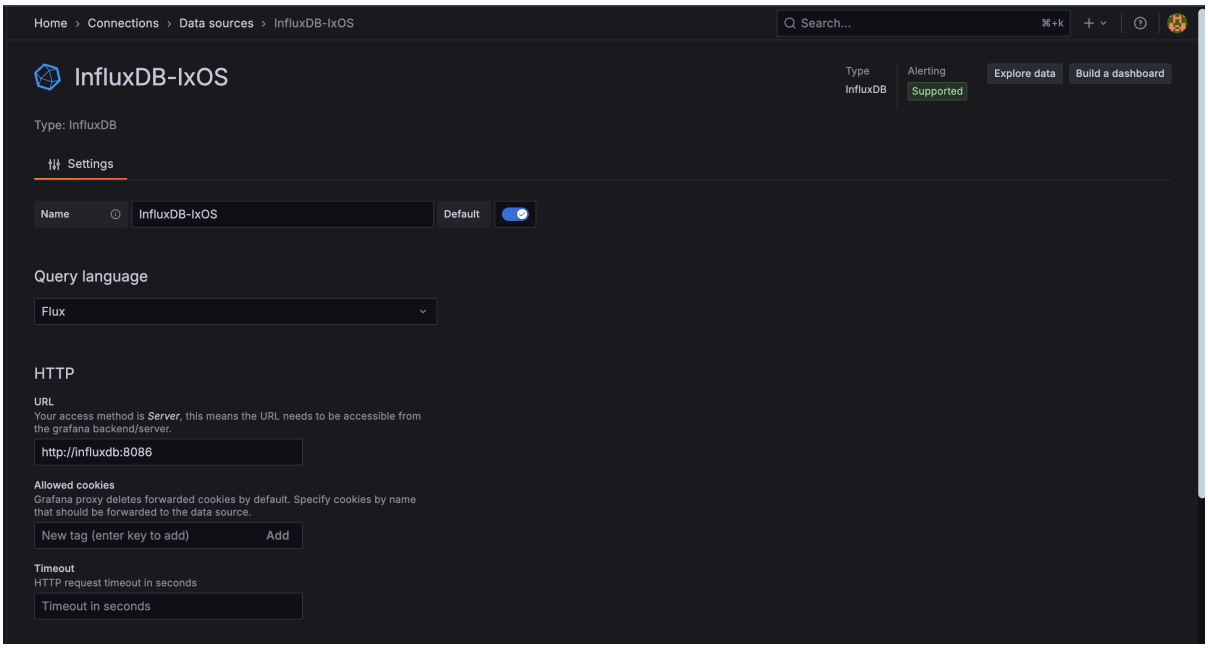
In InfluxDB

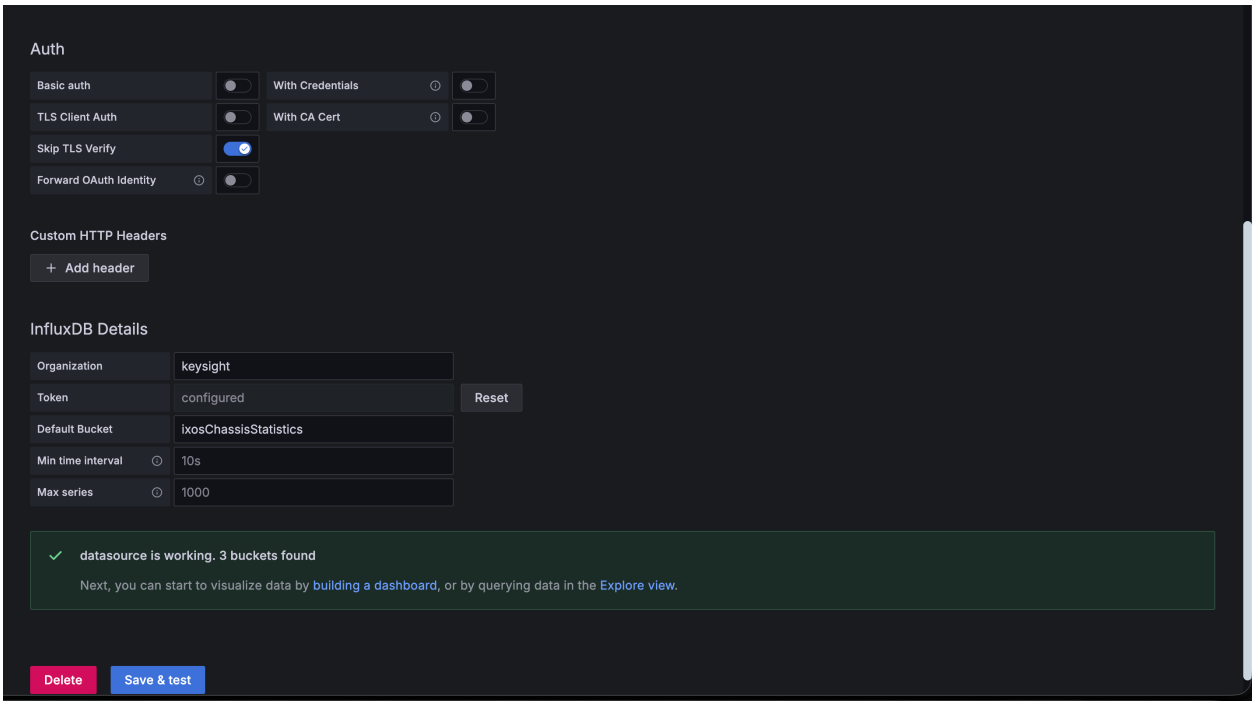


5) Visualization of collected metrics into Grafana:

Now, that we see data is populated, let's connect these metric collectors to **Grafana**. Since all the docker containers are running locally we will use **container_name.**

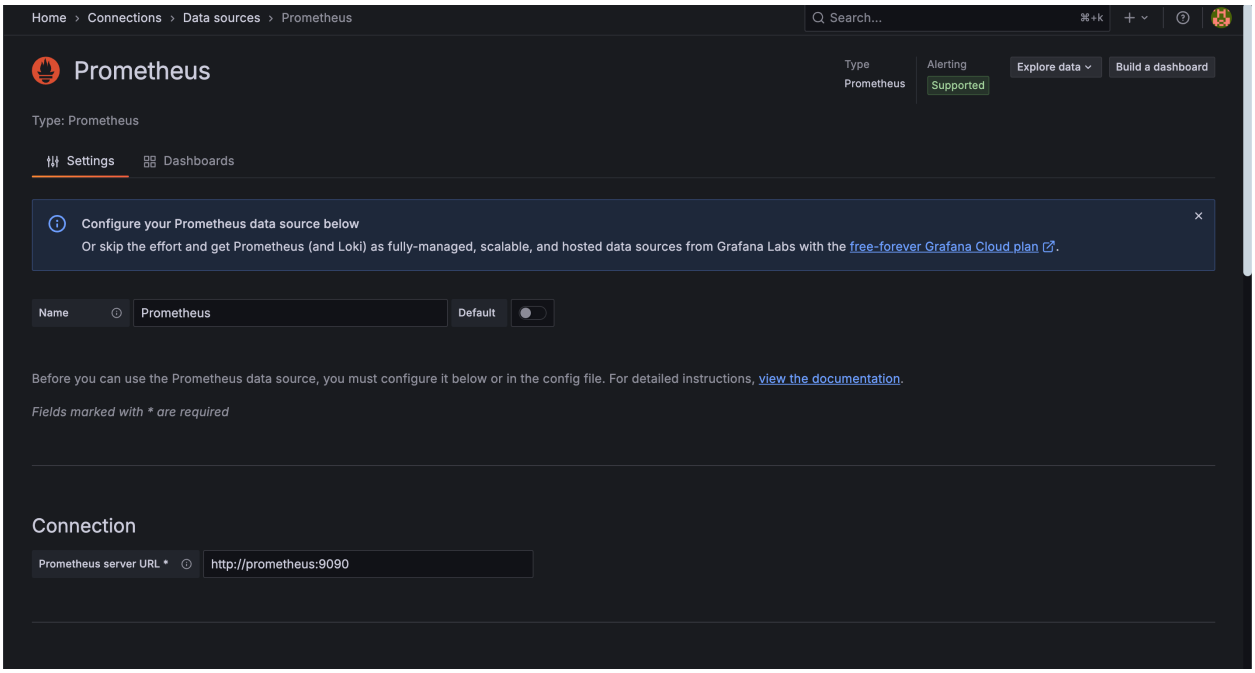
1) InfluxDB (with **container name)**

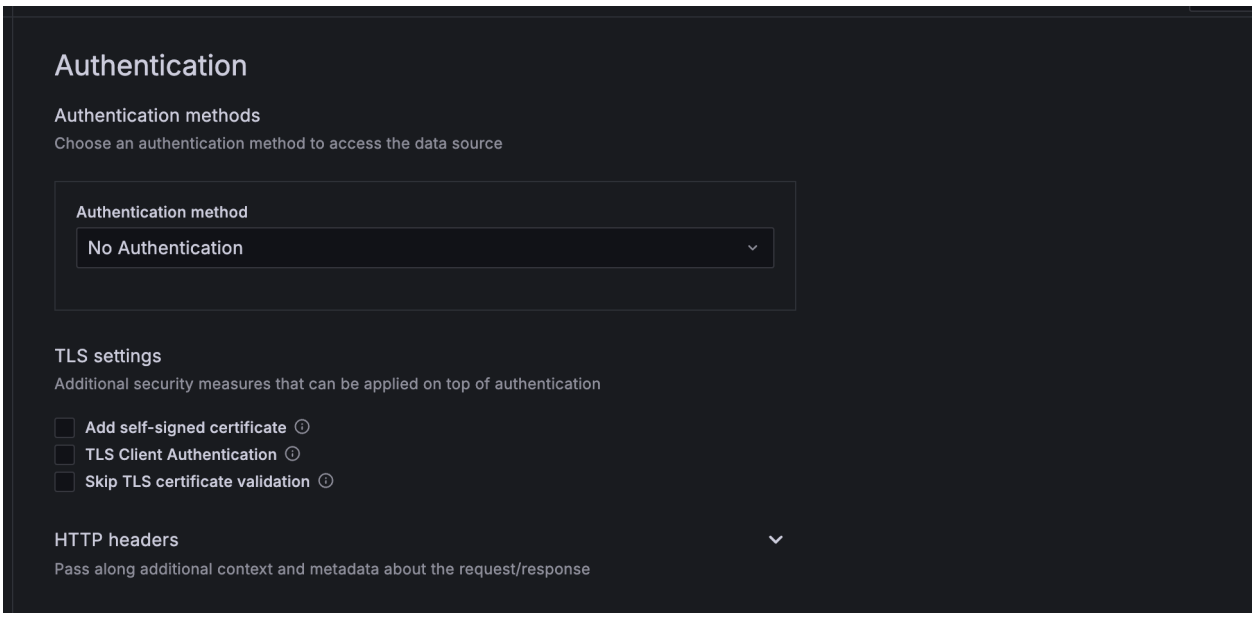


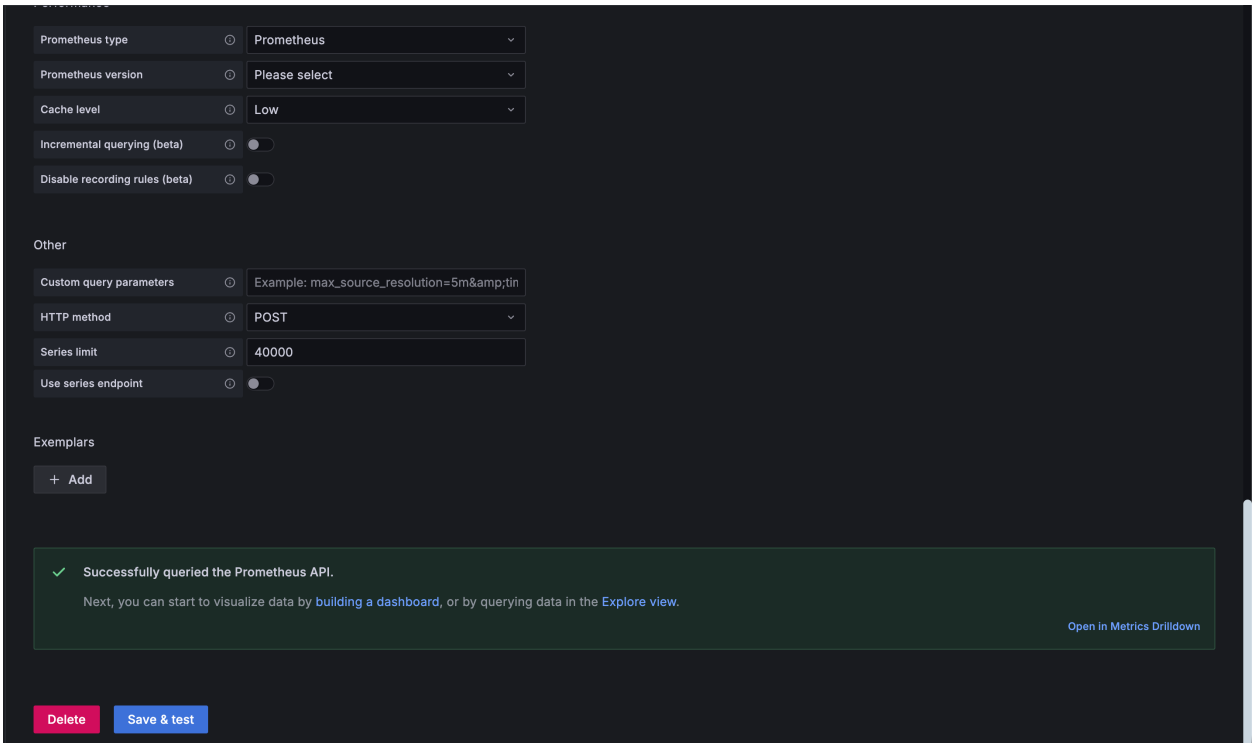


*** Note:** Every time you recreate the Grafana container you will need to write and save to the Token field. This is the API TOKEN you created in previous steps.

2) Prometheus







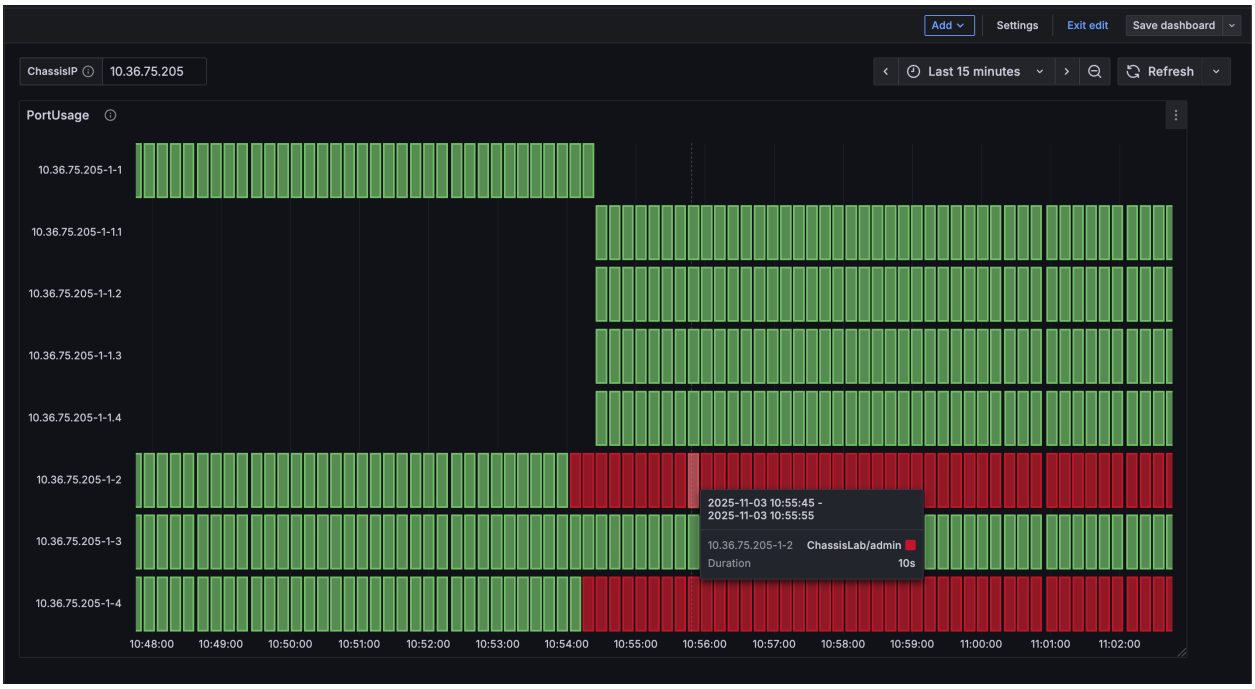
OK. So now that both monitoring sources are connected let's get to charting.

Port Usage Visualization Panel:

- We create a Dashboard and add visualization to it.
- Then we create a Dashboard Variable for ChassisIP

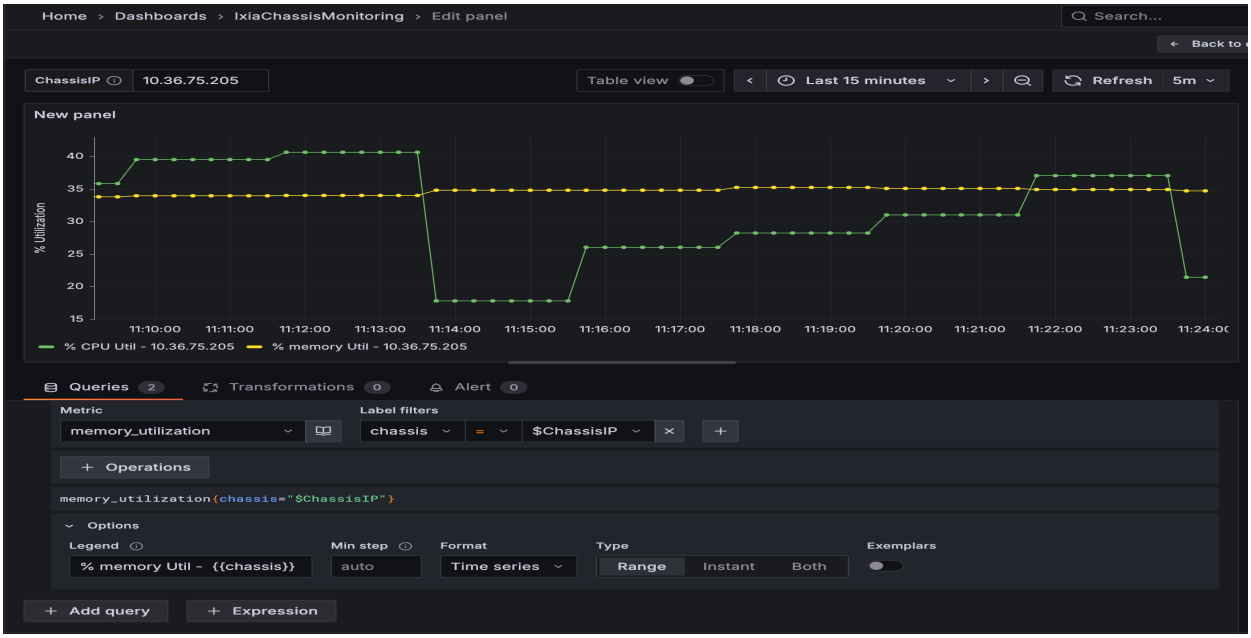
> Example Query for Ownership Visualization

```
from(bucket: "ixosChassisStatistics")
  |> range(start: -24h)
  |> filter(fn: (r) => r["_measurement"] == "portUtilization")
  |> filter(fn: (r) => r["chassis"] == "${ChassisIP}")
  |> filter(fn: (r) => r["_field"] == "owner")
```

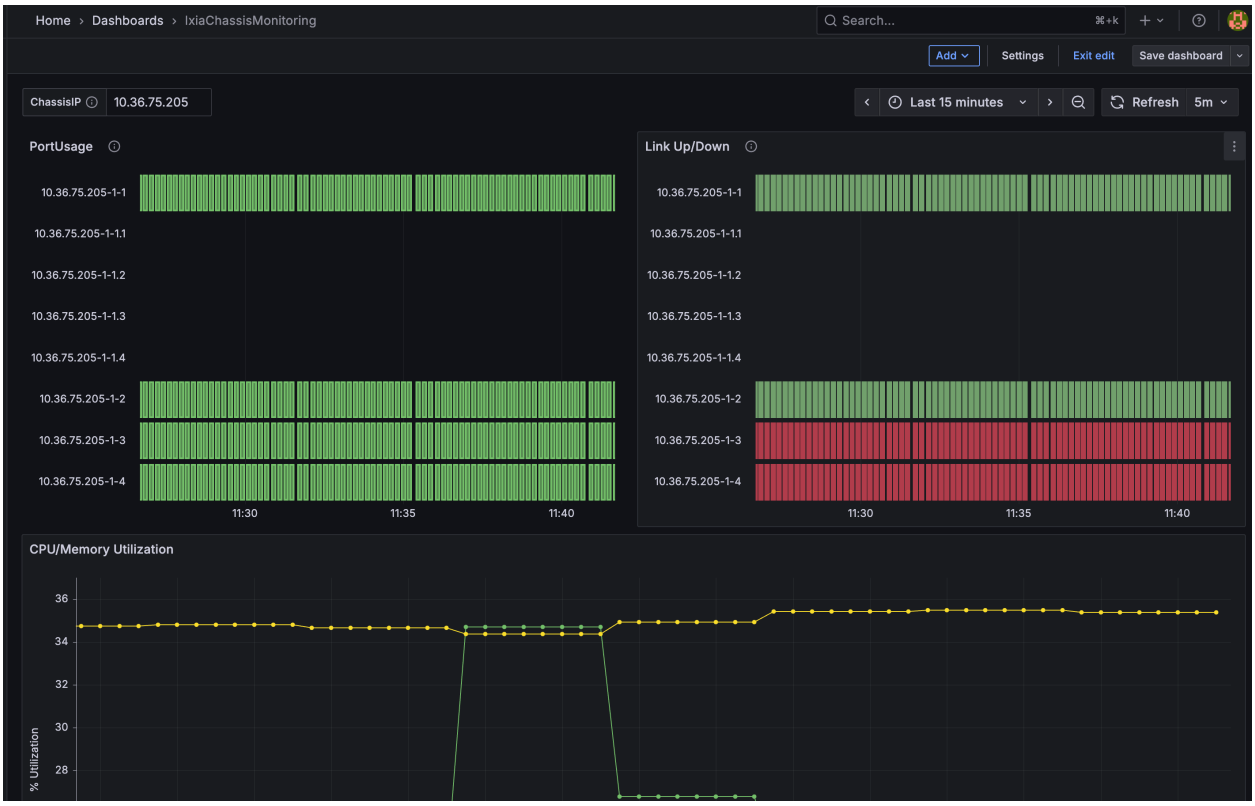


We add few more panels to view **PortTransmit State** and **Port UP/DOWN state**

Next, we will visualize performance metrics data coming from Prometheus. You can user Query Builder or the drop down query maker is good enough



Final Dashboard:



Sensors Dashboard:

Note: These dashboards are available at. Just go to Dashboards → Import

https://github.com/ashwinjo/IxPortUtilizationPlotter/blob/main/IxOSSensorData-1762728124980.json

https://github.com/ashwinjo/IxPortUtilizationPlotter/blob/main/IxiaChassisMonitoring-1762728398844.json