IxOS Monitoring Solution.

Deploying the monitoring solution.

1) Clone / Download IxPortUtilizationPlotter locally

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

cd IxPortUtilizationPlotter

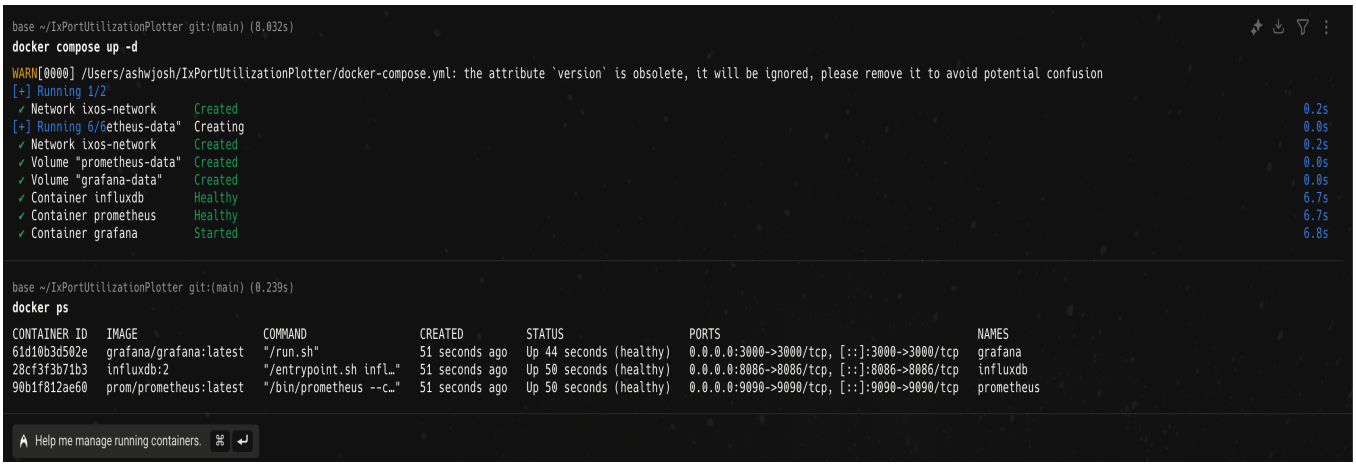
2) Setup the credentials and setup details to start the influxDB, Graphana and Prometheus container.

- 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**

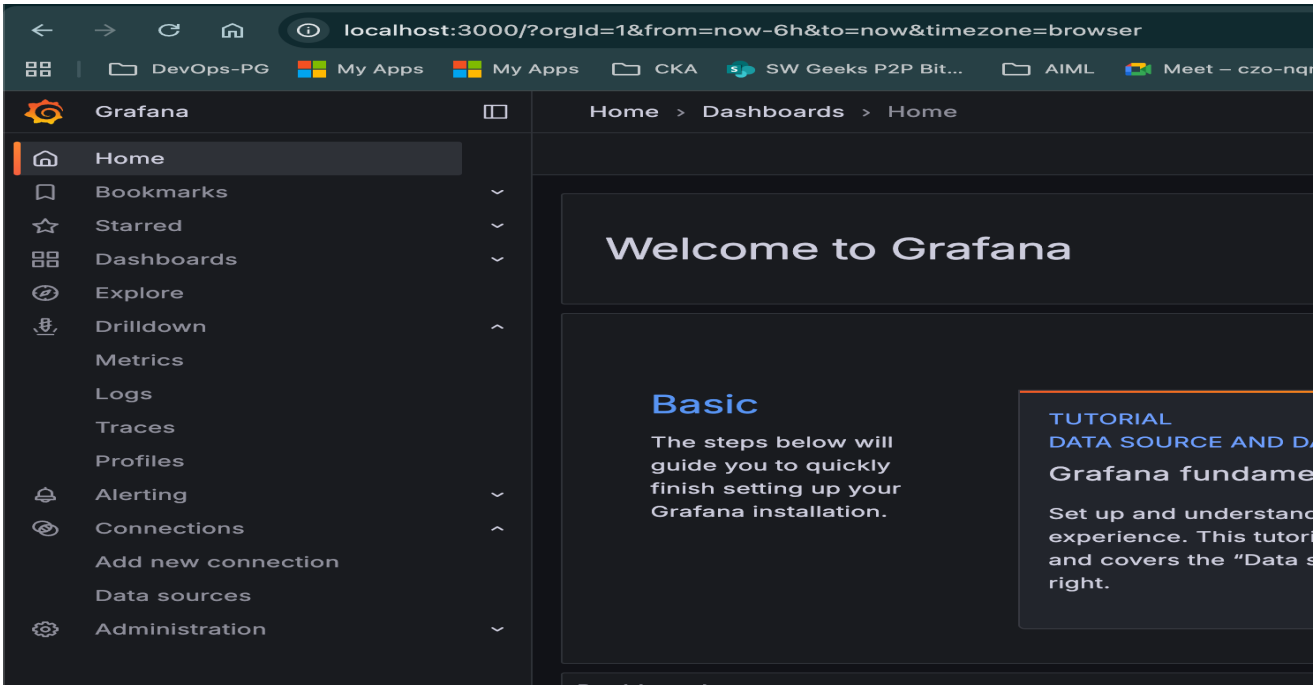
```
# Service Ports (customize if defaults conflict with existing services)
INFLUXDB_PORT=8086
PROMETHEUS_PORT=9090
GRAFANA_PORT=3000

# 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
```

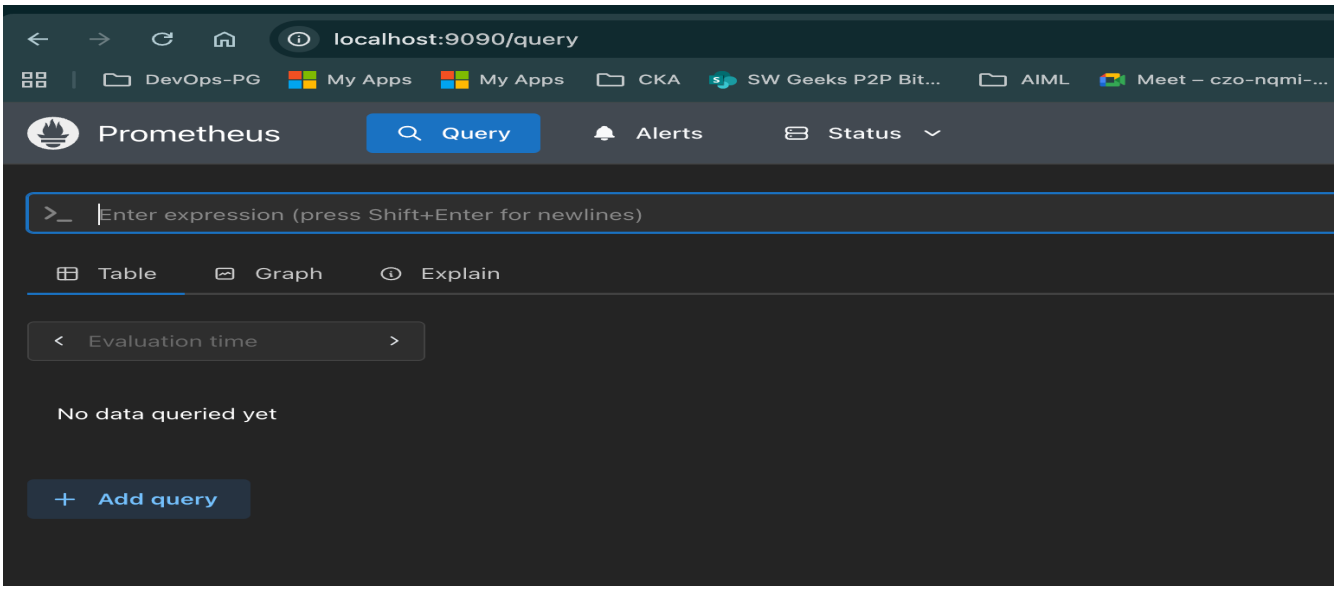
3) Start InfluxDB, Prometheus and Grapahana containers from the docker-compose file withing the repo

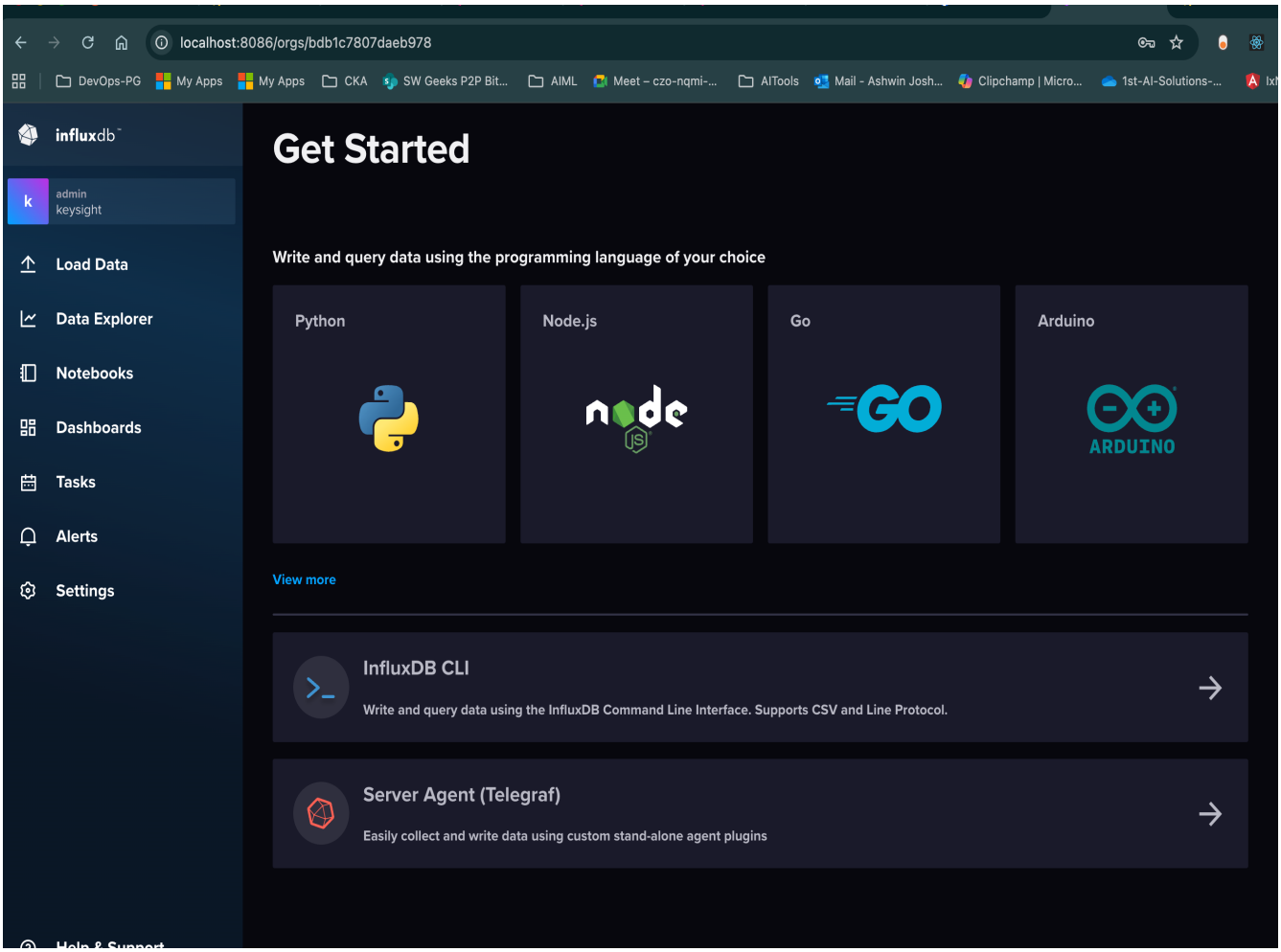
> docker compose up -d



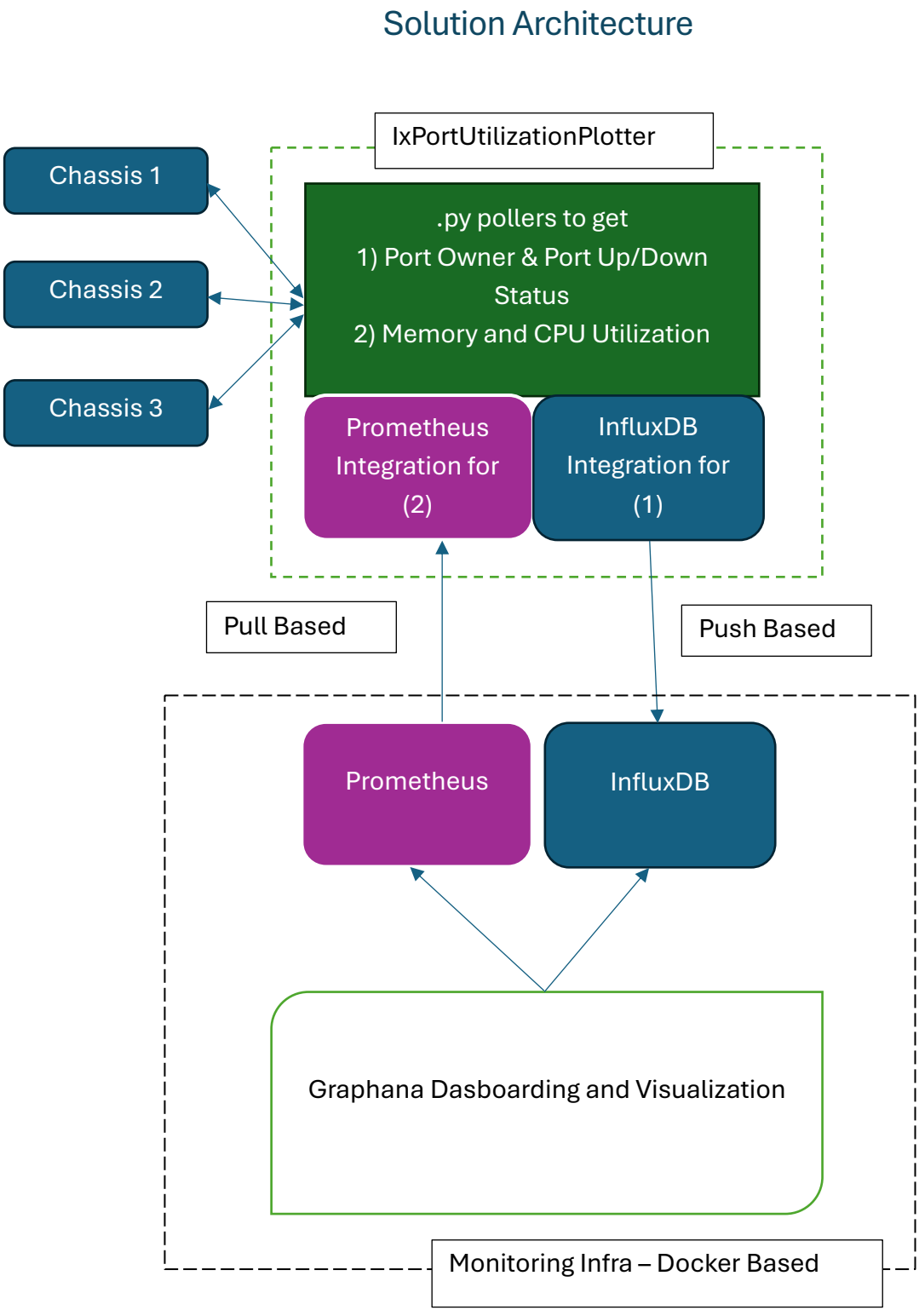
4) Confirm that containers started OK by logging in these portals (Note the UN/PW you set in .env OR then the defaults configured)




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


localhost:8086/orgs/bdb1c7807daeb978
influxdb
admin
keysight
Load Data
Data Explorer
Notebooks
Dashboards
Tasks
Alerts
Settings
Get Started
Write and query data using the programming language of your choice
Python
Node.js
Go
Arduino
ARDUINO
View more
InfluxDB CLI
Write and query data using the InfluxDB Command Line Interface. Supports CSV and Line Protocol.
Server Agent (Telegraf)
Easily collect and write data using custom stand-alone agent plugins

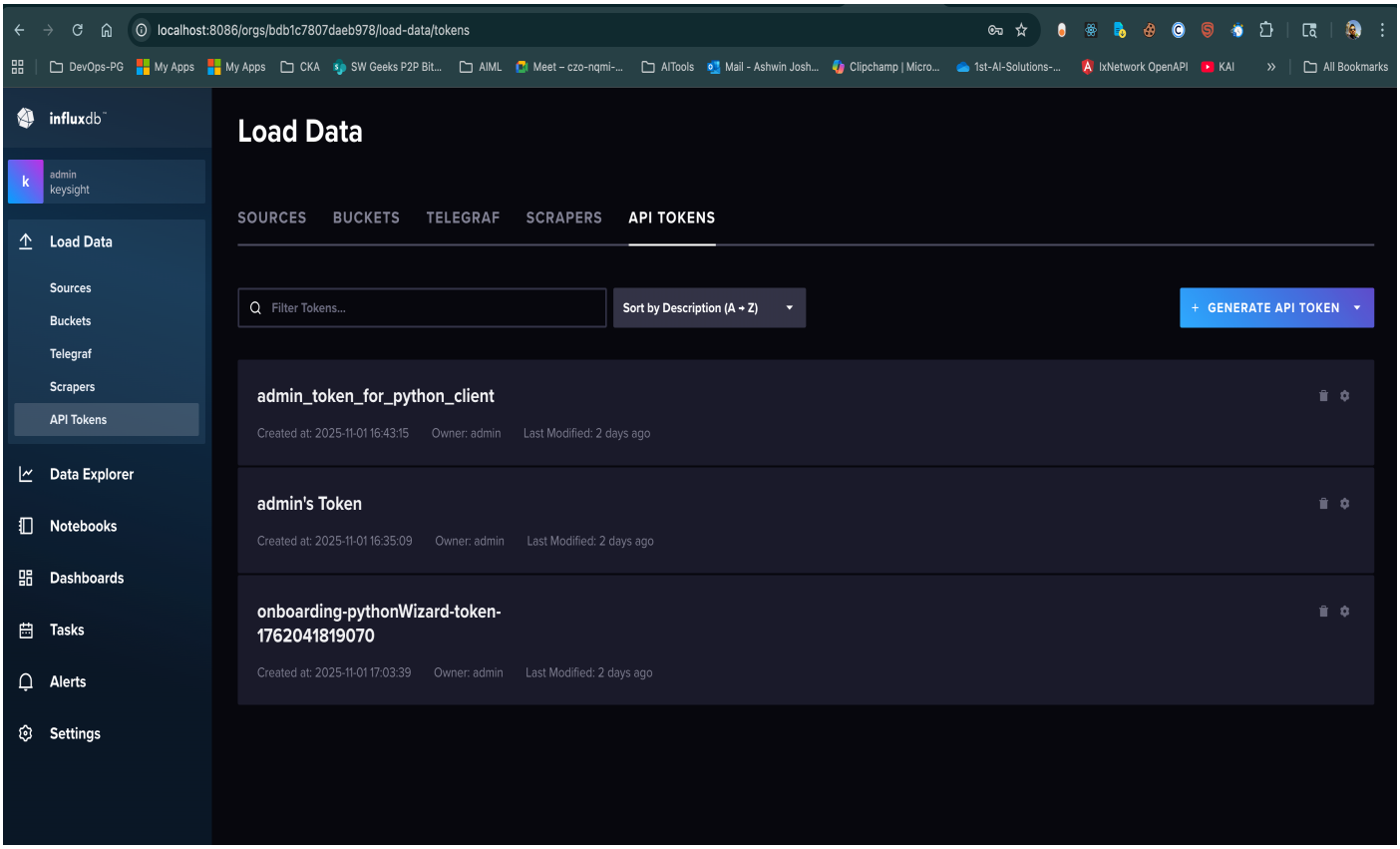
4) Metrics Collection: We will be taking Hybrid approach here, using Prometheus (for numeric metrics) and InfluxDB (for string-based metrics)



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.



Once you get the TOKEN add it to “<path>/IxPortUtilizationPlotter/config.py”

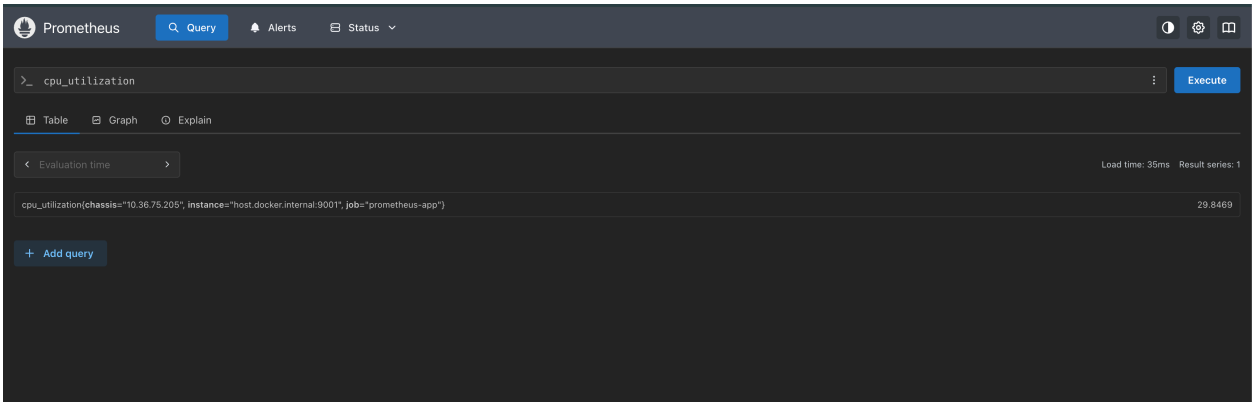
```
# InfluxDB URL (use 'influxdb' hostname in Docker, 'localhost' outside Docker)
INFLUXDB_URL = os.getenv(
    'INFLUXDB_URL',
    'http://localhost:8086'
)

# InfluxDB API Token (required for authentication)
INFLUXDB_TOKEN = os.getenv(
    'INFLUXDB_TOKEN',
    'eegHpR9kkgxg5KG7rklj2zQI86-5z7yNETx0P0qQpSnw1owDxSL5IF-uQruOP-J8M_xmrhT3KWECh-QGbsdyYA=='
)
```

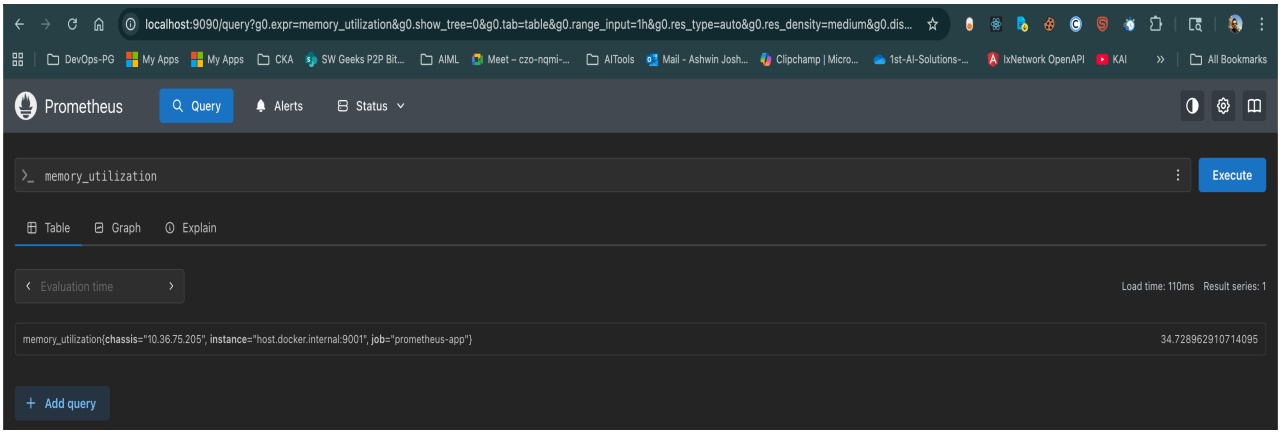
Step 1: Start to pollers

```
run_pollers.sh
```

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

In Prometheus:





In InfluxDB

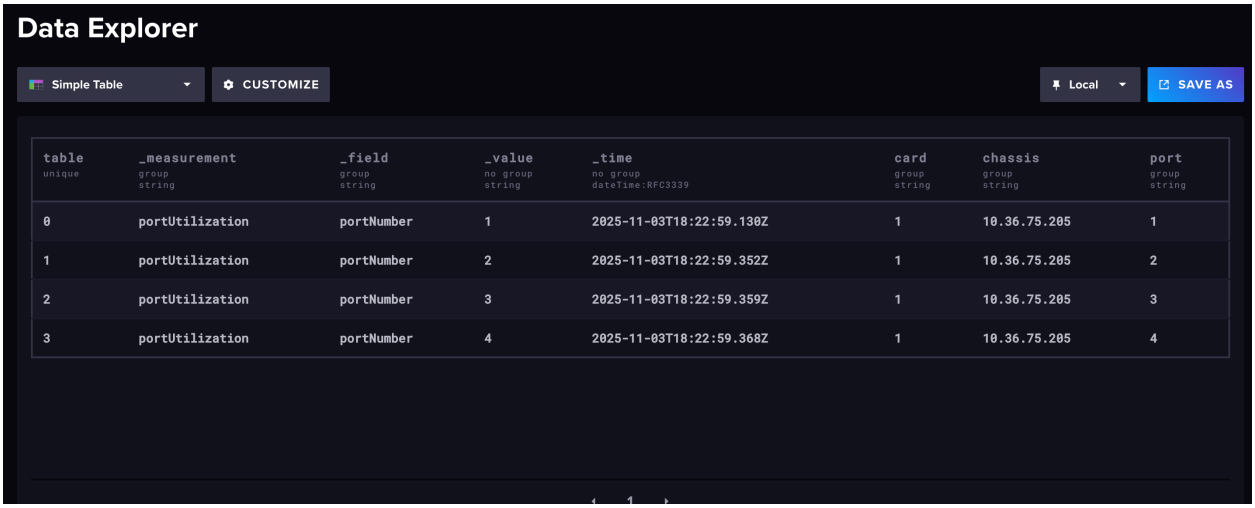


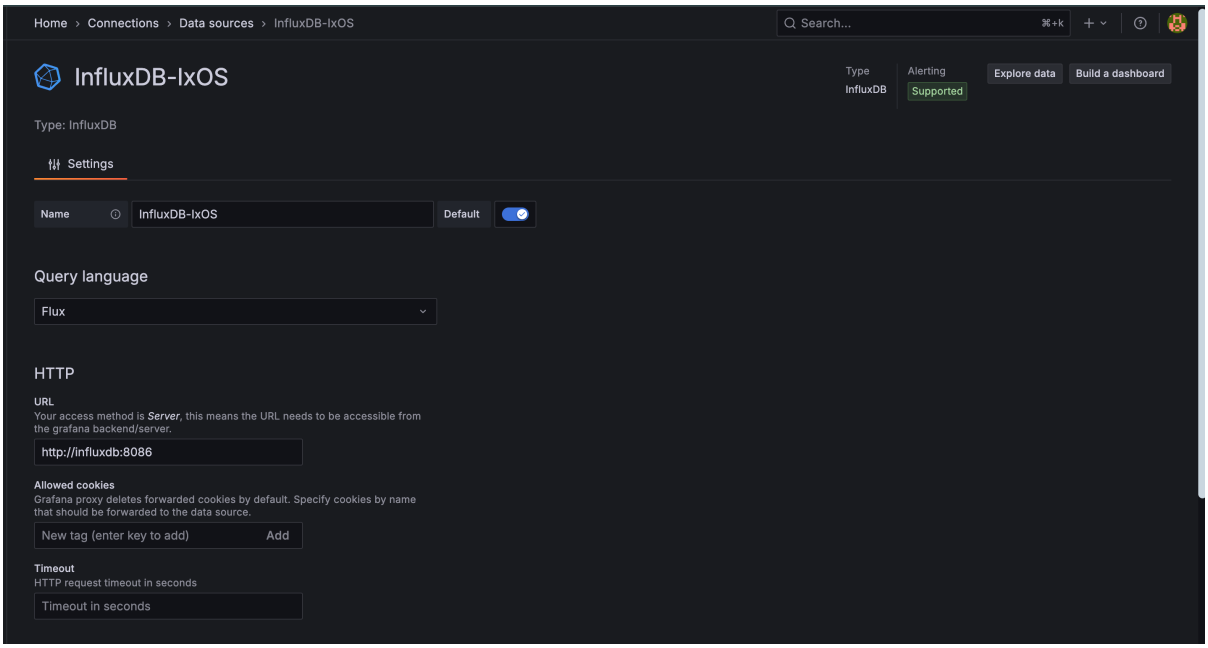
table unique	_measurement group string	_field group string	_value no group string	_time no group dateTime:RFC3339	card group string	chassis group string	port group string
0	portUtilization	portNumber	1	2025-11-03T18:22:59.130Z	1	10.36.75.205	1
1	portUtilization	portNumber	2	2025-11-03T18:22:59.352Z	1	10.36.75.205	2
2	portUtilization	portNumber	3	2025-11-03T18:22:59.359Z	1	10.36.75.205	3
3	portUtilization	portNumber	4	2025-11-03T18:22:59.368Z	1	10.36.75.205	4

5) Visualization of collected metrics into Graphana:

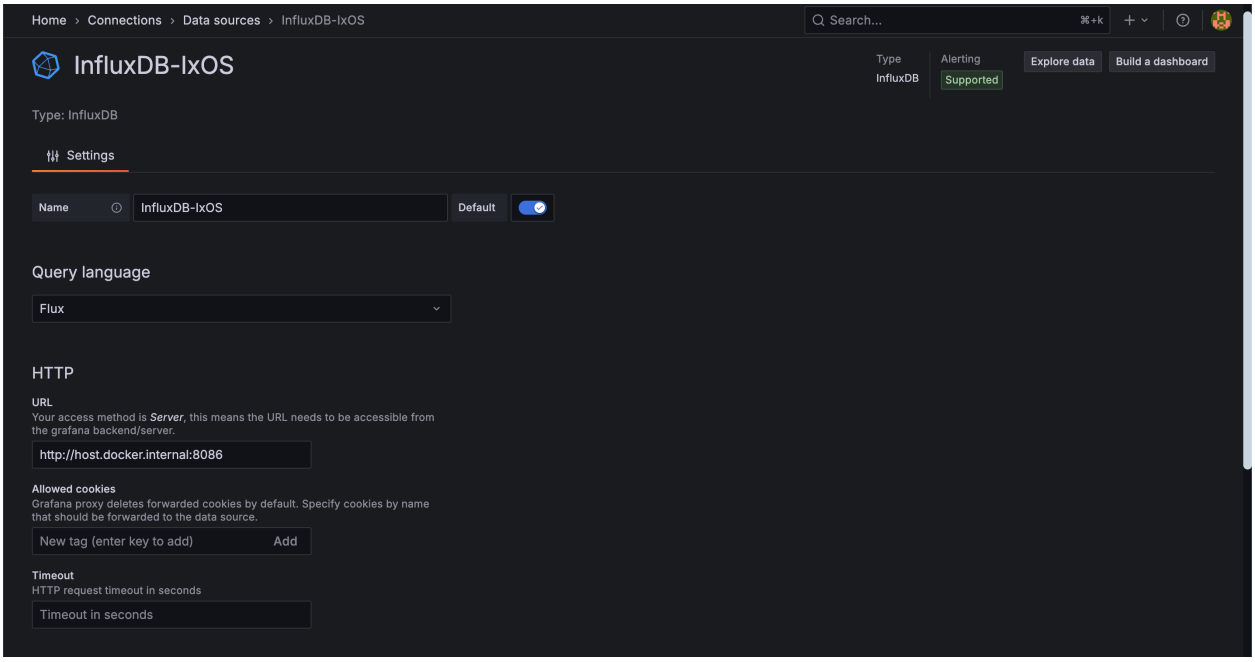
Now, that we see data is populated, let's connect these metric collectors to Graphana. Since all the docker containers are running locally we can use

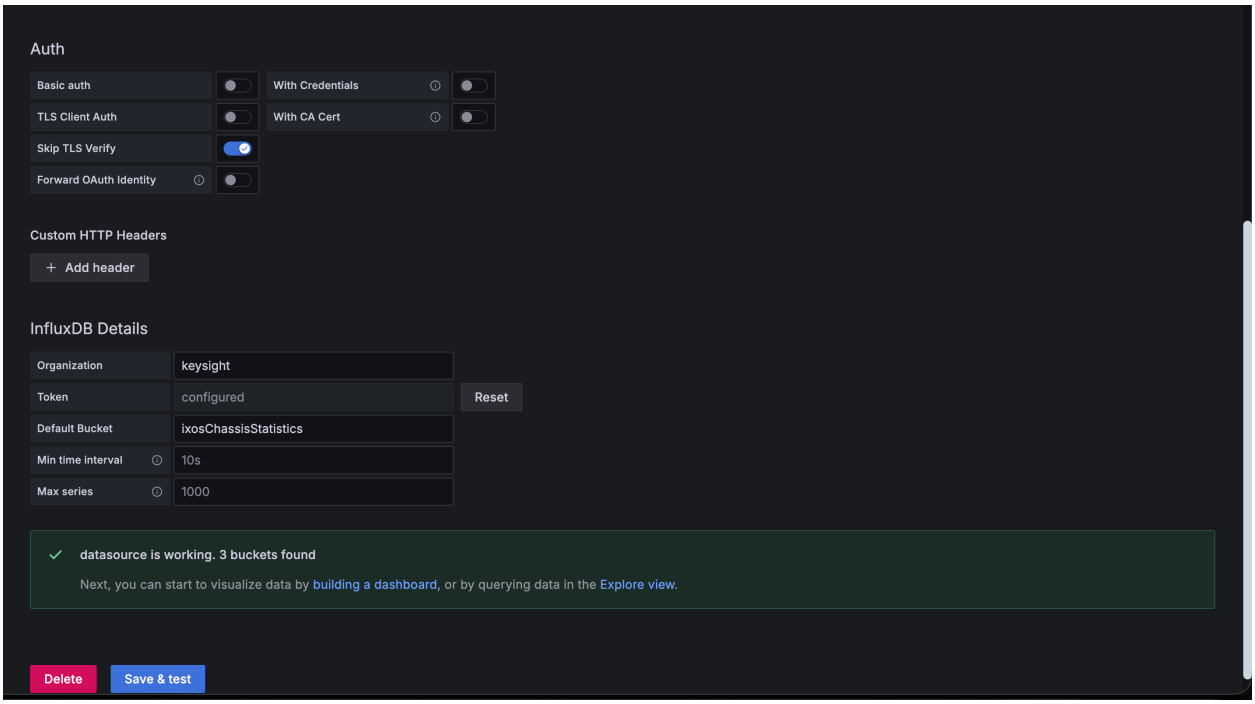
host.docker.internal OR **container_name.**

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

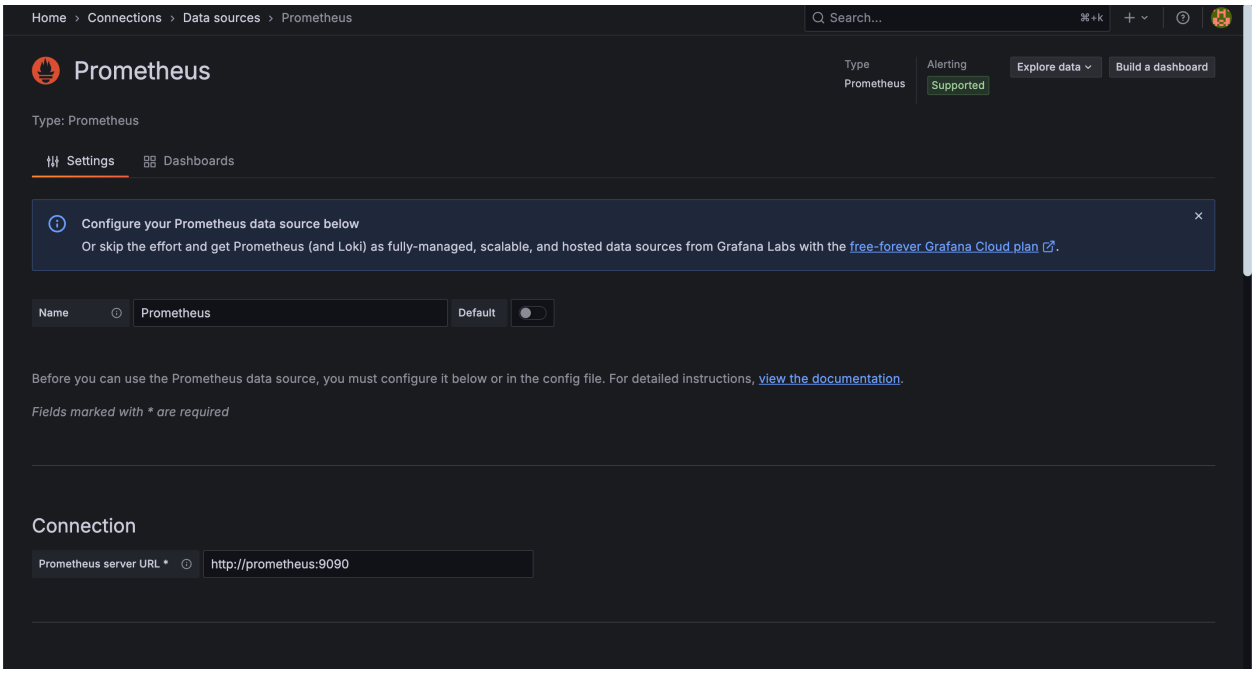


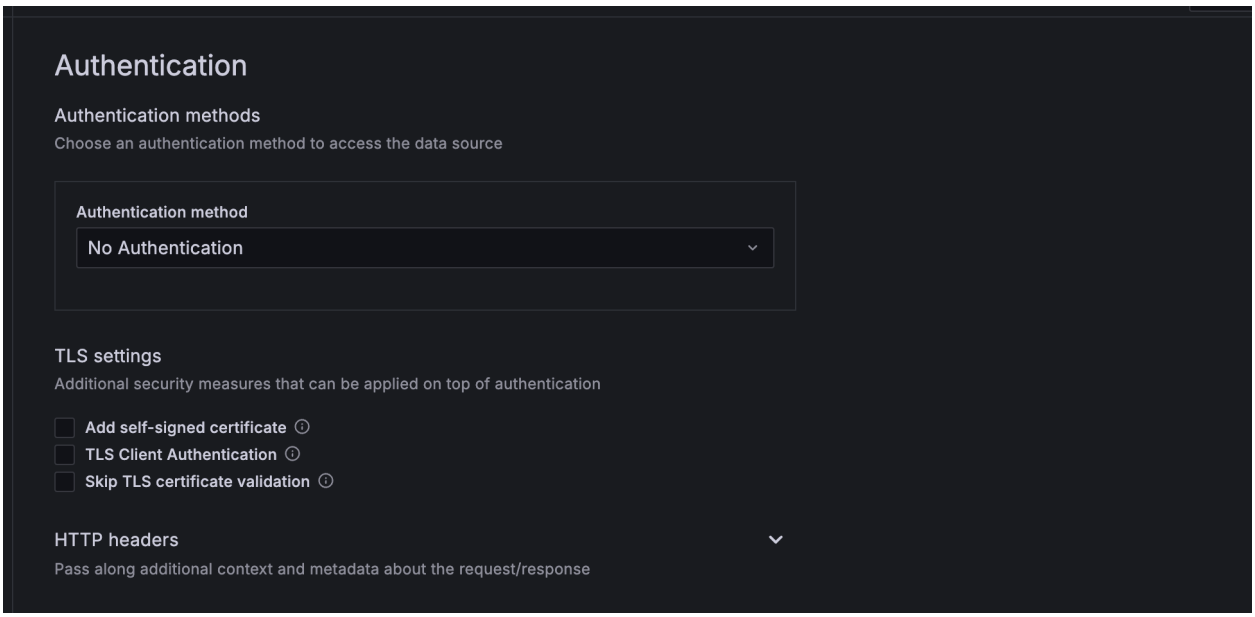
OR **host.docker.internal**. BOTH WORK WELL

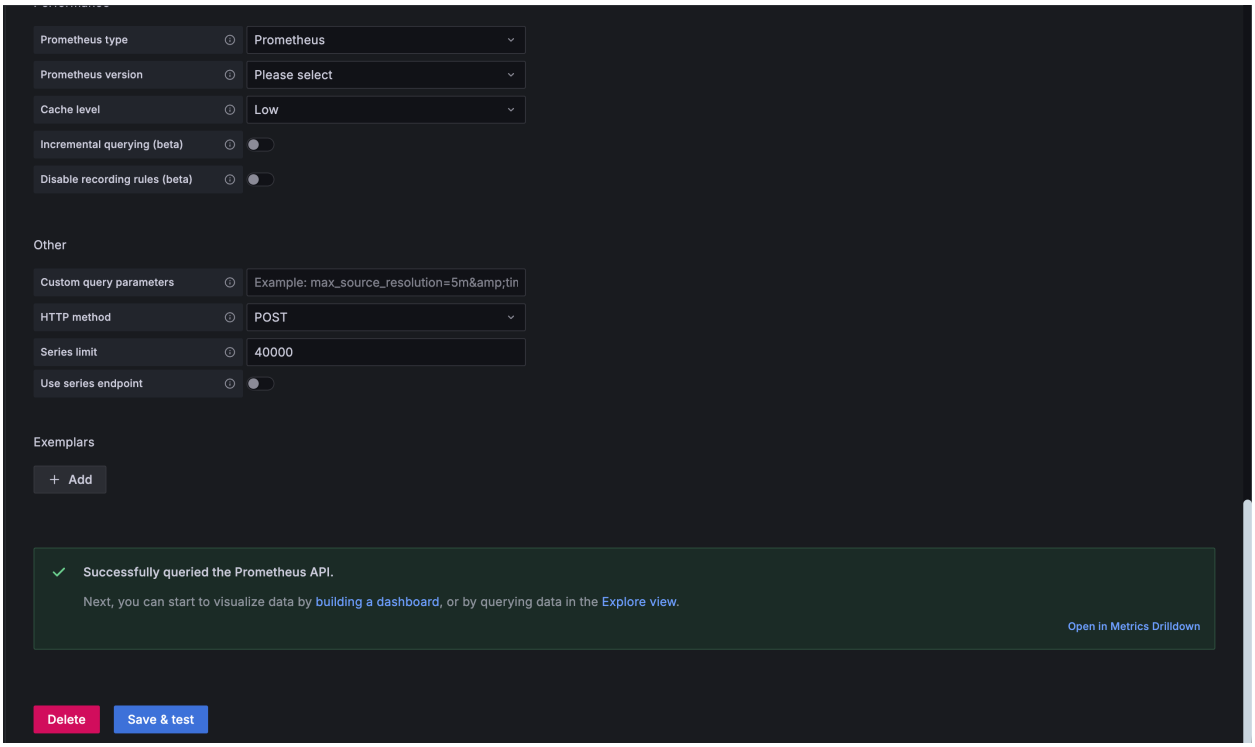

Home › Connections › Data sources › InfluxDB-IxOS
Search... ⌘+k
InfluxDB-IxOS
Type
InfluxDB
Alerting
Supported
Explore data
Build a dashboard
Type: InfluxDB
Settings
Name
InfluxDB-IxOS
Default
Query language
Flux
HTTP
URL
Your access method is Server, this means the URL needs to be accessible from the grafana backend/server.
http://host.docker.internal:8086
Allowed cookies
Grafana proxy deletes forwarded cookies by default. Specify cookies by name that should be forwarded to the data source.
New tag (enter key to add)
Add
Timeout
HTTP request timeout in seconds
Timeout in seconds


Auth
Basic auth
With Credentials
TLS Client Auth
With CA Cert
Skip TLS Verify
Forward OAuth Identity
Custom HTTP Headers
+ Add header
InfluxDB Details
Organization
keysight
Token
configured
Reset
Default Bucket
ixosChassisStatistics
Min time interval
10s
Max series
1000
datasource is working. 3 buckets found
Next, you can start to visualize data by building a dashboard, or by querying data in the Explore view.
Delete
Save & test

2) Prometheus







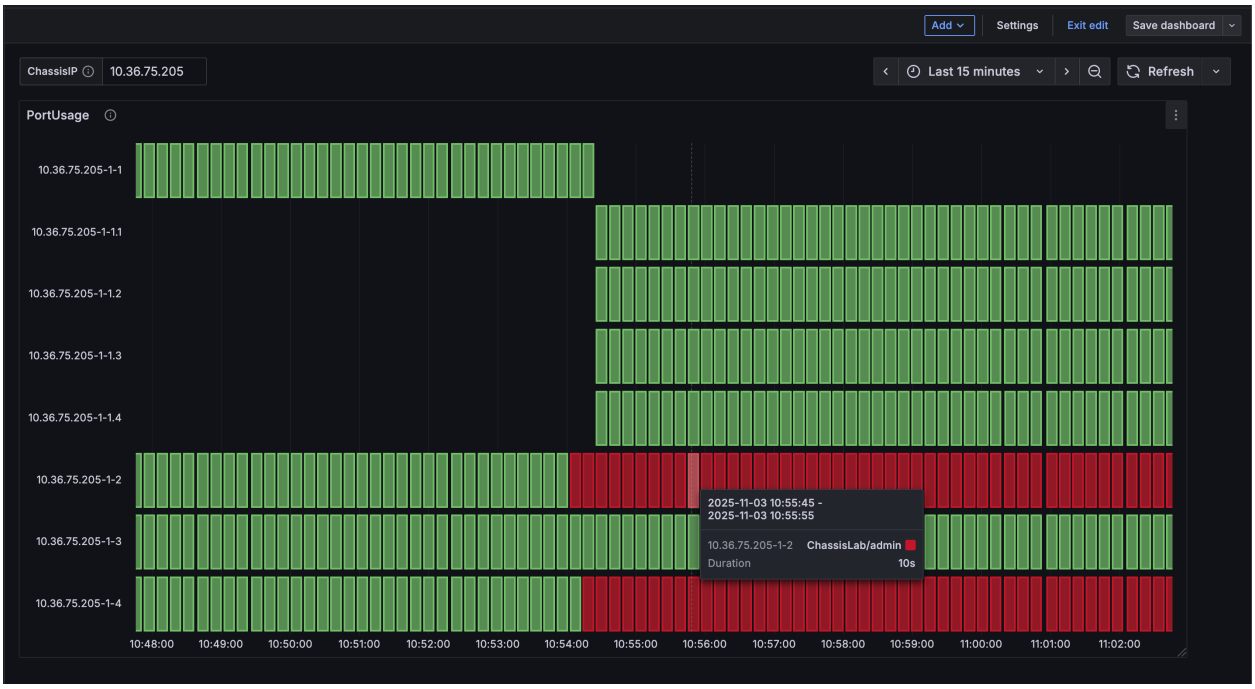
OK. So now that both monitoring sources are connected let's get to charting.

Port Usage Visualization Panel:

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

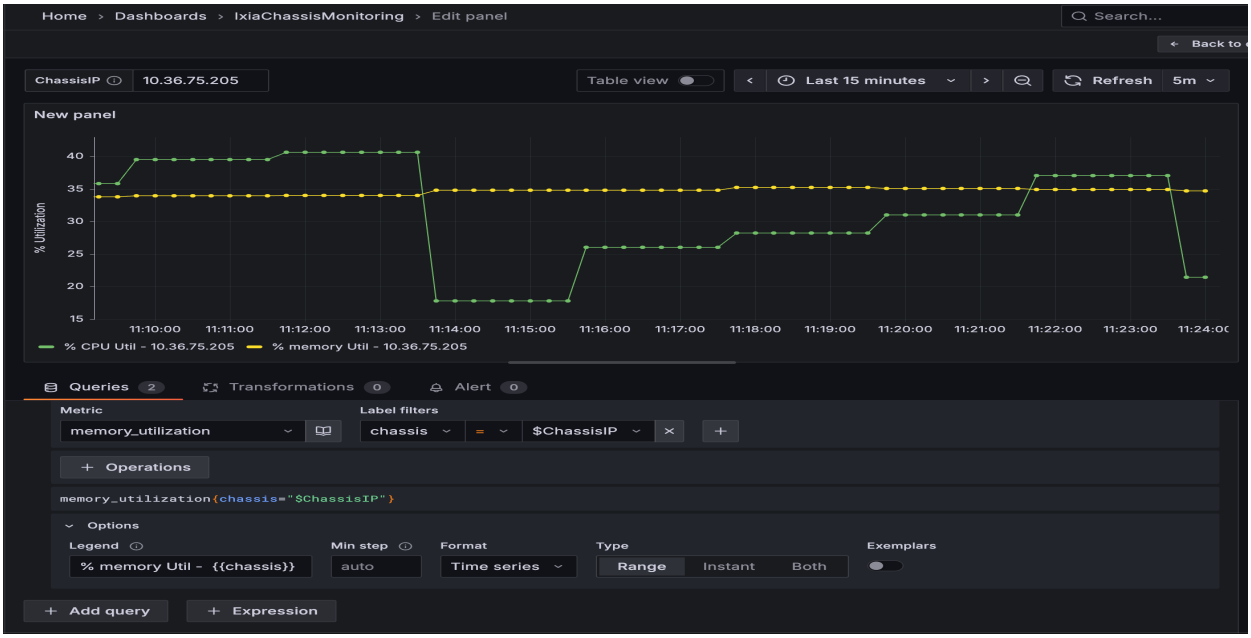
> Example Query for Ownership Visualization

```
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: