Monitoring Tools - Prometheus & Grafana

[Using GitHub Codespaces]

Here, we will learn how to set up a Monitoring Dashboard using Prometheus & Grafana for ML Applications.

Procedure in a nutshell

- Setup Prometheus using Codespaces
- Setup Grafana using Codespaces
- Run an application on Codespaces
- Run another application on Codespaces
- Access the applications' metrics in Prometheus
- Create Real-time Dashboard in Grafana
- Add Alerts in Grafana

Prometheus

Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud, later became a standalone open source project and maintained independently of any company.

Prometheus collects and stores its metrics as time series data, i.e. metrics information is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels.

Features

Prometheus's main features includes:

- A multi-dimensional data model with time series data identified by metric name and key/value pairs
- *PromQL*, a flexible query language to leverage this dimensionality
- No reliance on distributed storage; single server nodes are autonomous
- Time series collection happens via a pull model over HTTP
- Pushing time series is supported via an intermediary gateway
- Targets are discovered via service discovery or static configuration
- Multiple modes of graphing and dashboarding support

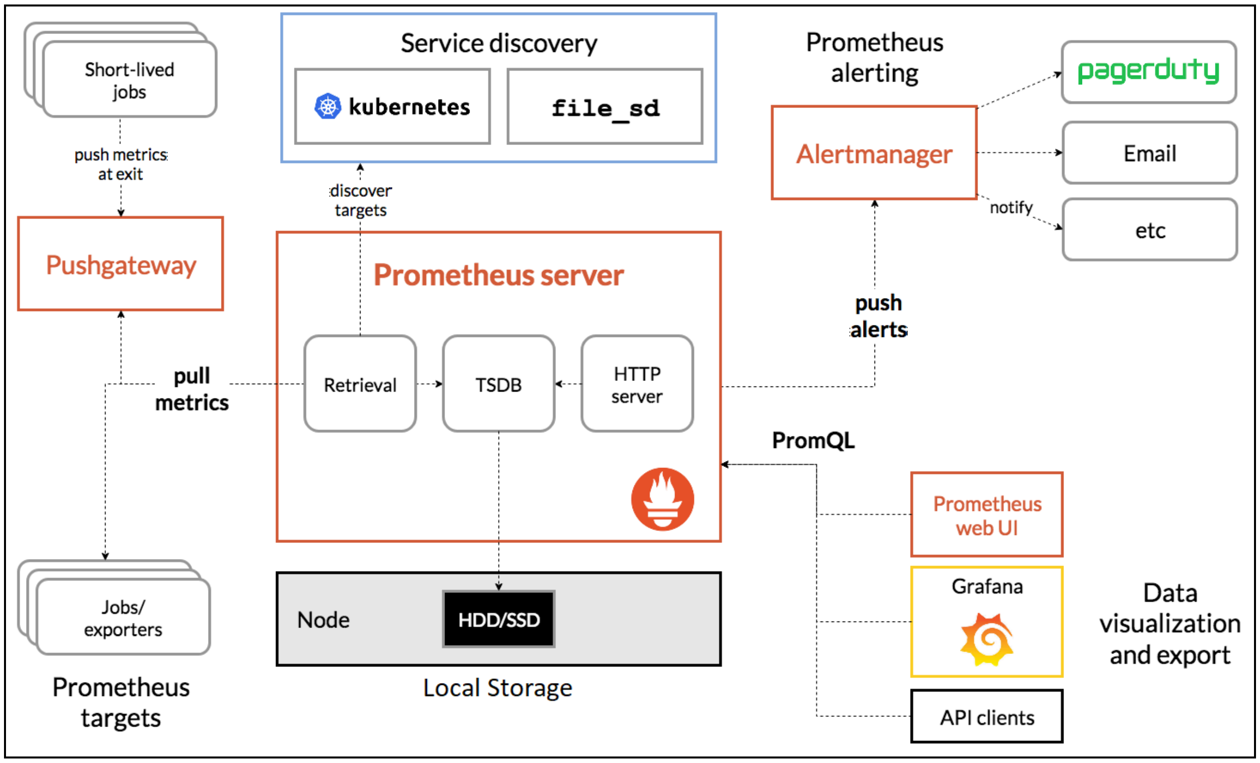
Components

The Prometheus ecosystem consists of multiple components, many of which are optional:

- The main **Prometheus server** which scrapes and stores time series data
- **Client libraries** for instrumenting application code such as *prometheus-client*
- A **push gateway** for supporting short-lived jobs
    - Pushgateway is an intermediary service which allows you to push metrics from jobs which cannot be scrapped.
- Special-purpose **exporters** for services that don't expose Prometheus-supported metrics by default
    - A Prometheus **exporter** aggregates and imports data from a non-Prometheus system to a Prometheus system.
- An **alertmanager** to handle alerts
- Various support tools such as Grafana

Architecture

This diagram illustrates the architecture of Prometheus and some of its ecosystem components:



Prometheus scrapes metrics from instrumented jobs, either directly or via an intermediary push gateway for short-lived jobs. It stores all scraped samples locally and runs rules over this data to either aggregate and record new time series from existing data or generate alerts. Grafana or other API consumers can be used to visualize the collected data.
To know more about Prometheus, refer here.

Grafana Open Source

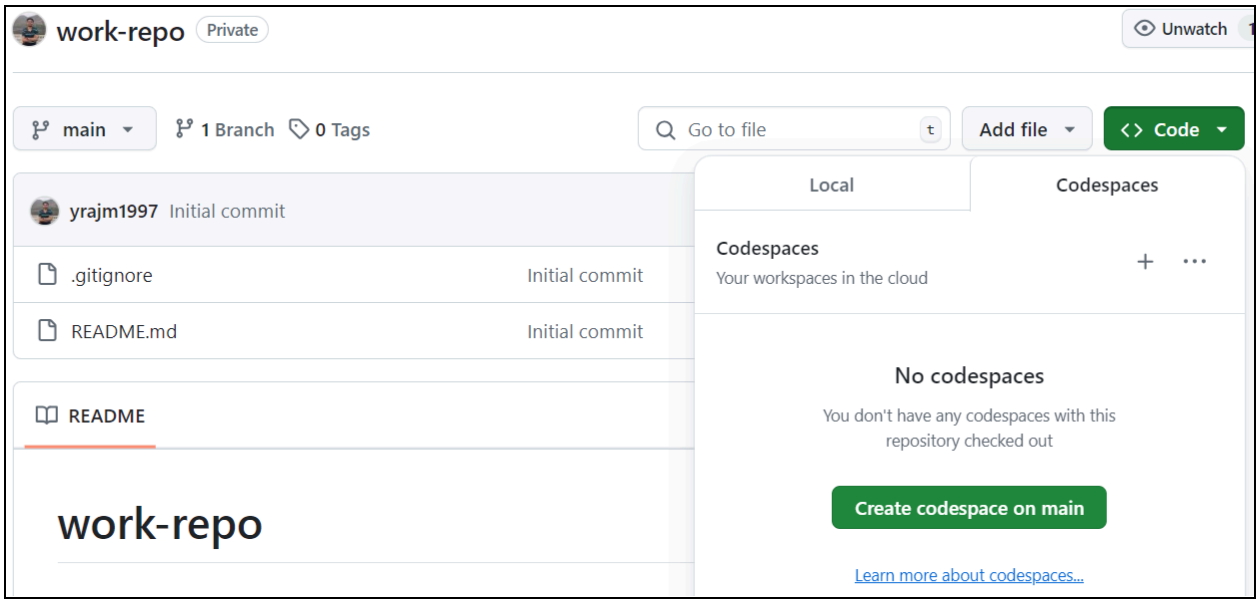
Grafana

Grafana open source is an open source visualization and analytics software. It allows you to query, visualize, alert on, and explore your metrics, logs, and traces irrespective of where they are stored. It provides you with tools to turn the time-series database (TSDB) data into insightful graphs and visualizations.

Setting up a Monitoring Dashboard for ML Applications

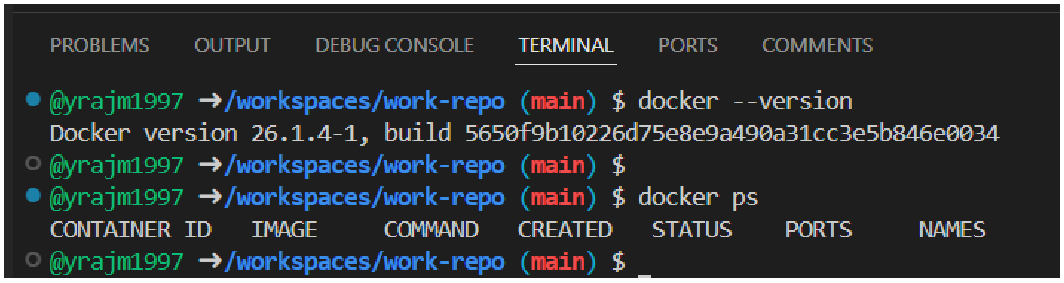
I. Setup Prometheus using Codespaces

1. Create a new repository on GitHub and start a Codespace.



2. Once the Codespace is created, you can change its color theme as per your choice by going to *File >> Preferences >> Theme >> Color Theme >> Dark (Visual Studio)*

3. Run the below command to see if the docker is installed or not, and docker daemon is running
```
docker --version
docker ps
```



```
@yrajm1997 →/workspaces/work-repo (main) $ docker --version
Docker version 26.1.4-1, build 5650f9b10226d75e8e9a490a31cc3e5b846e0034
@yrajm1997 →/workspaces/work-repo (main) $
@yrajm1997 →/workspaces/work-repo (main) $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
@yrajm1997 →/workspaces/work-repo (main) $ _
```

4. In Codespace IDE, create a parent directory for Prometheus and move into it

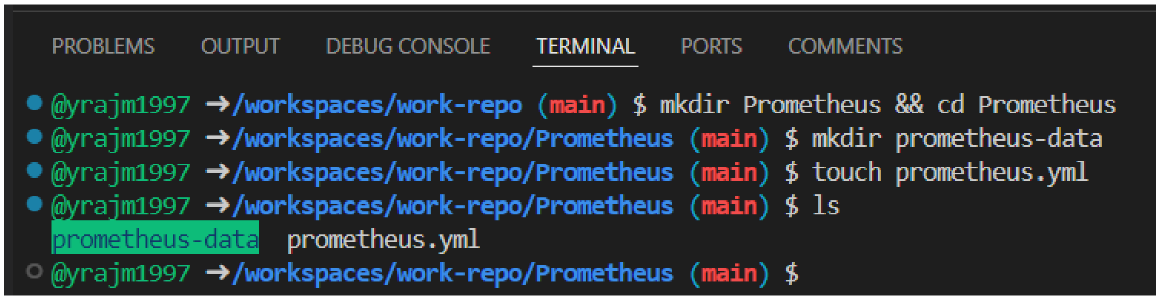
```
mkdir Prometheus && cd Prometheus
```

5. Create a directory named **prometheus-data**, to let Prometheus store the data.

```
mkdir prometheus-data
```

6. Create a configuration file - ***prometheus.yml***

```
touch prometheus.yml
```



```
@yrajm1997 →/workspaces/work-repo (main) $ mkdir Prometheus && cd Prometheus
@yrajm1997 →/workspaces/work-repo/Prometheus (main) $ mkdir prometheus-data
@yrajm1997 →/workspaces/work-repo/Prometheus (main) $ touch prometheus.yml
@yrajm1997 →/workspaces/work-repo/Prometheus (main) $ ls
prometheus-data  prometheus.yml
@yrajm1997 →/workspaces/work-repo/Prometheus (main) $
```

Once the file is created add below content to it: (Do check the indentation)

prometheus.yml:

```
# my global config
global:
   scrape_interval:  20s     # Set the scrape interval to every 20 seconds. Default is every 1 minute.
  evaluation_interval: 20s  # Evaluate rules every 20 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# A scrape configuration containing exactly one endpoint to scrape:
scrape_configs:
   # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.

  - job_name: "prometheus"
```

```
    metrics_path: '/metrics'
    scheme: 'https'
    scrape_interval: 10s       # override default value
    scrape_timeout: 10s       # override default value
    static_configs:
        - targets: ["<your-forwarded-address>"]       #  The target here it's
Prometheus itself.
```

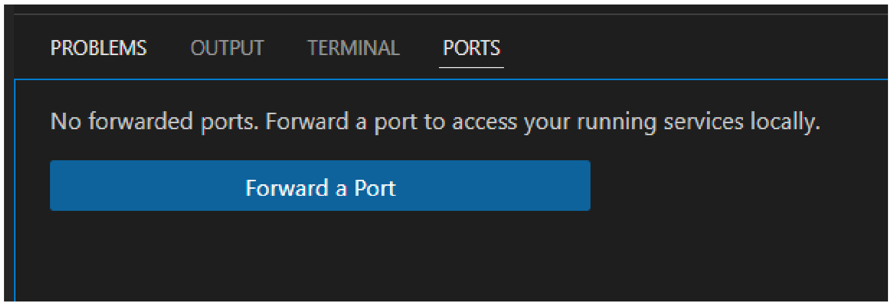
We will first start by considering Prometheus itself as one of the targets. You will need to provide your Codespace forwarded address in *targets*.

We will start the prometheus using docker in a container. By default, Prometheus will run on port 9090 when the container starts, and it will scrape the metrics from the **/metrics** endpoint of targets.

Note that the configuration file and the prometheus data directory will need to be mounted while running the Prometheus container.

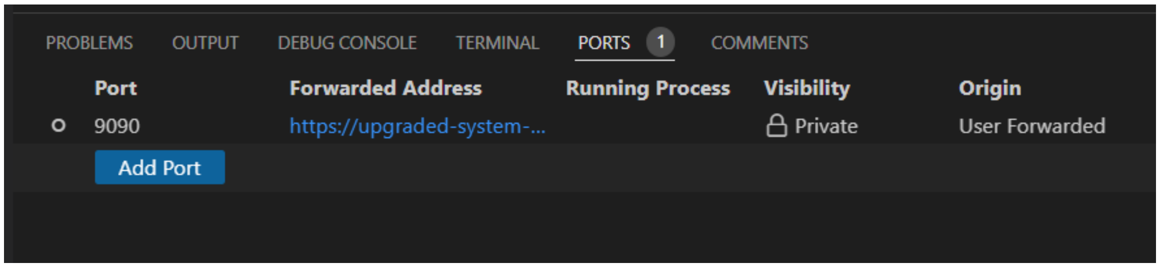
Once the container starts successfully, it will be running inside the Codespace. To let others access it, the port needs to be forwarded. We can forward a particular port even before starting a container that utilizes that port.

7. Forward the port 9090
   Go to the PORTS tab in the terminal window, then select *Forward a Port*.



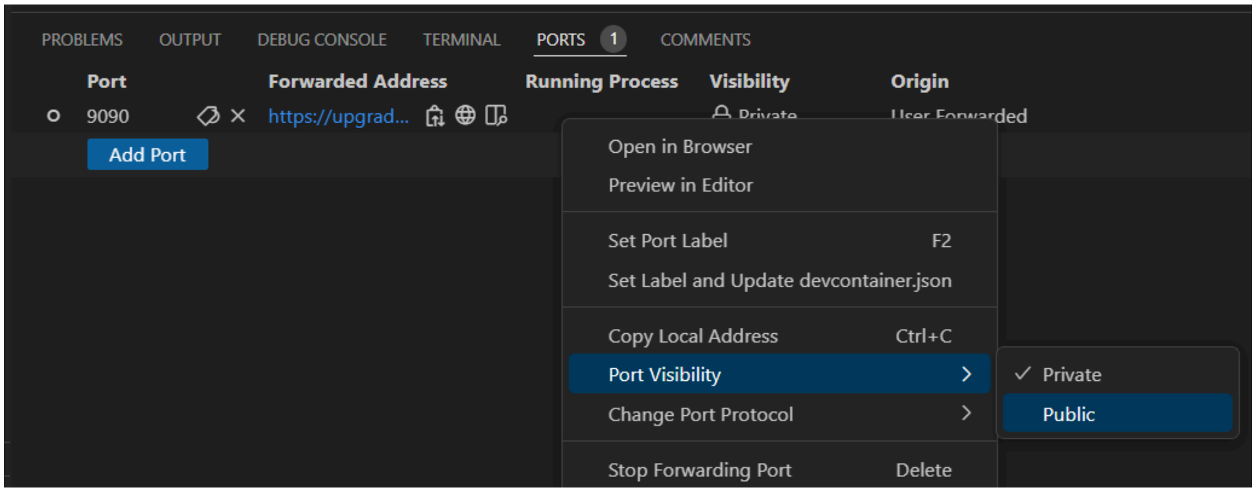
No forwarded ports. Forward a port to access your running services locally.

Enter the port number (eg. 9090) and hit enter. A new row will be added.

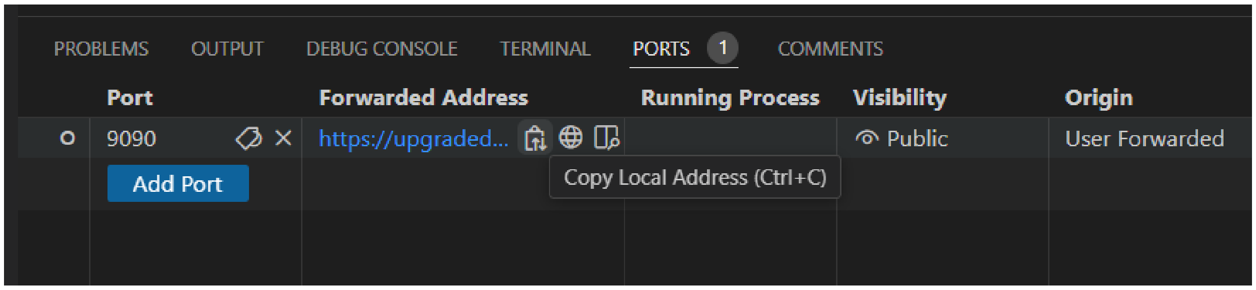


Port	Forwarded Address	Running Process	Visibility	Origin
9090	https://upgraded-system-...		Private	User Forwarded

Right click on it >> Port Visibility >> Public.



Then hover over the ‘Forwarded Address’, copy it, and paste in **prometheus.yml** as targets.



While adding the forwarded address in prometheus.yml, remove https and keep only the url similar to below:

```
- targets: ["upgraded-system-jw7wgp7xw563q74q-9090.app.github.dev"]
```

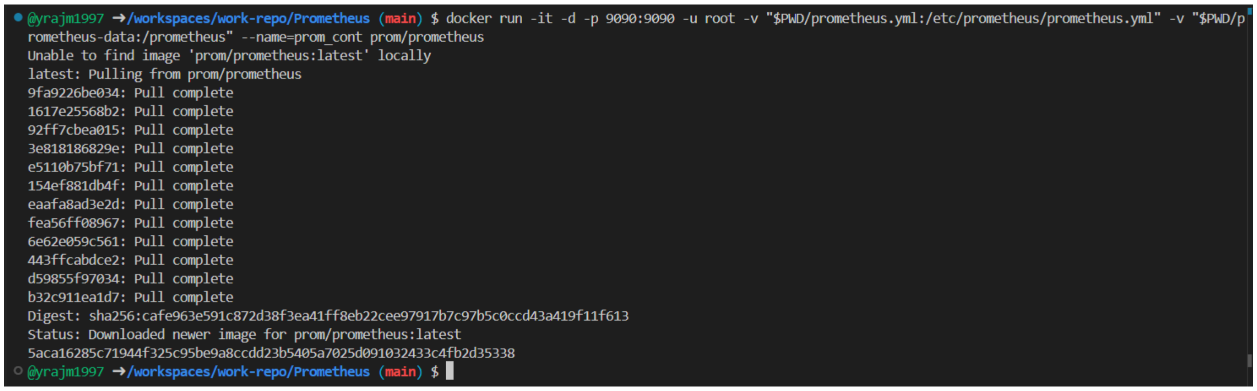
Now, you can start the Prometheus container.

8. From the *Prometheus* directory, run the below command to start Prometheus

```
docker run -it -d -p 9090:9090 -u root -v
"$PWD/prometheus.yml:/etc/prometheus/prometheus.yml" -v
"$PWD/prometheus-data:/prometheus" --name=prom_cont prom/prometheus
```

Here, the container will run

- with a pseudo terminal (`-it`)
- in a detached mode (`-d`)
- with container port mapped to host port (`-p 9090:9090`)
- as a root user (`-u root`)
- with config file prometheus.yml mounted at specified path in container using `-v` argument
- with prometheus-data directory mounted at specified path in container
- with name prom_cont (`--name=prom_cont`)
- using prometheus docker image (`prom/prometheus`)

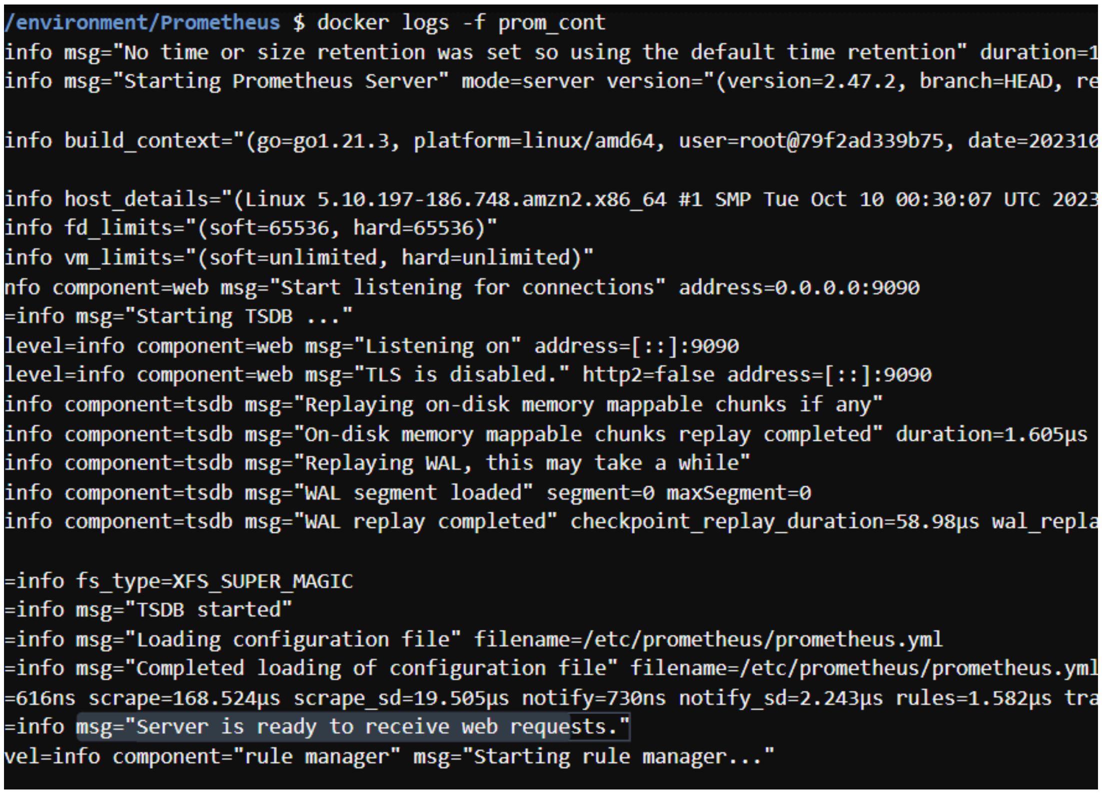


```
@yrajm1997 ➜/workspaces/work-repo/Prometheus (main) $ docker run -it -d -p 9090:9090 -u root -v "$PWD/prometheus.yml:/etc/prometheus/prometheus.yml" -v "$PWD/p
rometheus-data:/prometheus" --name=prom_cont prom/prometheus
Unable to find image 'prom/prometheus:latest' locally
latest: Pulling from prom/prometheus
9fa9226be034: Pull complete
1617e25568b2: Pull complete
92ff7cbea015: Pull complete
3e818186829e: Pull complete
e5110b75bf71: Pull complete
154ef881db4f: Pull complete
eaafa8ad3e2d: Pull complete
fea56ff08967: Pull complete
6e62e059c561: Pull complete
443ffcabdce2: Pull complete
d59855f97034: Pull complete
b32c911ea1d7: Pull complete
Digest: sha256:cafe963e591c872d38f3ea41ff8eb22cee97917b7c97b5c0ccd43a419f11f613
Status: Downloaded newer image for prom/prometheus:latest
5aca16285c71944f325c95be9a8ccdd23b5405a7025d091032433c4fb2d35338
@yrajm1997 ➜/workspaces/work-repo/Prometheus (main) $
```

9. Check the logs of the running container

   `docker logs -f prom_cont`

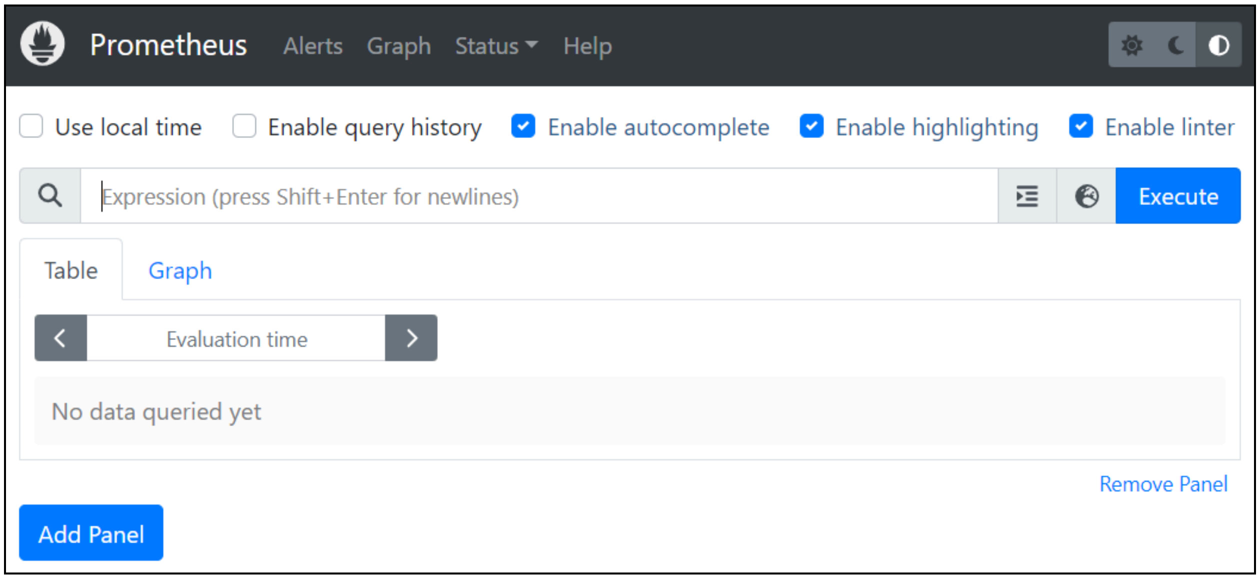
   You should see a message *"Server is ready to receive web requests."*



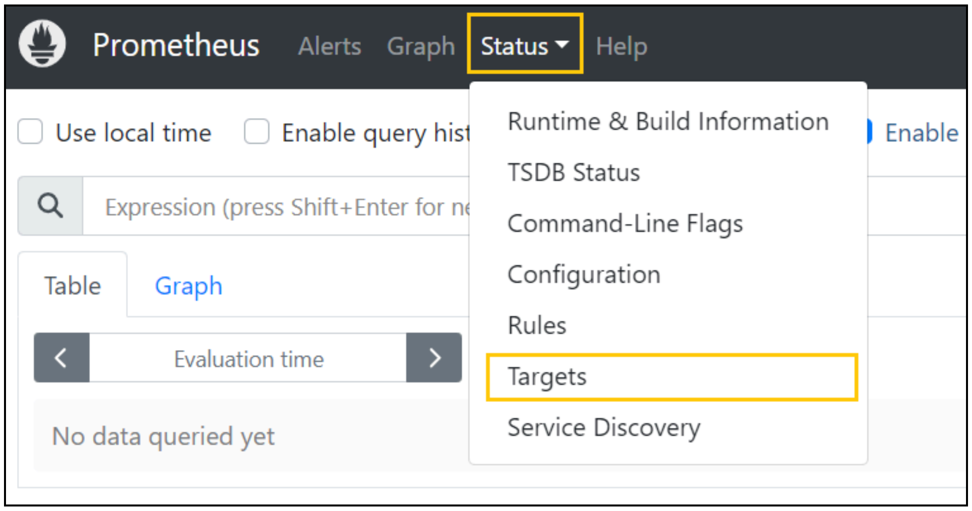
```
/environment/Prometheus $ docker logs -f prom_cont
info msg="No time or size retention was set so using the default time retention" duration=1
info msg="Starting Prometheus Server" mode=server version="(version=2.47.2, branch=HEAD, re

info build_context="(go=go1.21.3, platform=linux/amd64, user=root@79f2ad339b75, date=202310

info host_details="(Linux 5.10.197-186.748.amzn2.x86_64 #1 SMP Tue Oct 10 00:30:07 UTC 2023
info fd_limits="(soft=65536, hard=65536)"
info vm_limits="(soft=unlimited, hard=unlimited)"
nfo component=web msg="Start listening for connections" address=0.0.0.0:9090
=info msg="Starting TSDB ..."
level=info component=web msg="Listening on" address=[::]:9090
level=info component=web msg="TLS is disabled." http2=false address=[::]:9090
info component=tsdb msg="Replaying on-disk memory mappable chunks if any"
info component=tsdb msg="On-disk memory mappable chunks replay completed" duration=1.605µs
info component=tsdb msg="Replaying WAL, this may take a while"
info component=tsdb msg="WAL segment loaded" segment=0 maxSegment=0
info component=tsdb msg="WAL replay completed" checkpoint_replay_duration=58.98µs wal_repla

=info fs_type=XFS_SUPER_MAGIC
=info msg="TSDB started"
=info msg="Loading configuration file" filename=/etc/prometheus/prometheus.yml
=info msg="Completed loading of configuration file" filename=/etc/prometheus/prometheus.yml
=616ns scrape=168.524µs scrape_sd=19.505µs notify=730ns notify_sd=2.243µs rules=1.582µs tra
=info msg="Server is ready to receive web requests."
vel=info component="rule manager" msg="Starting rule manager..."
```

   (Use Ctrl+C to move out of the container logs.)

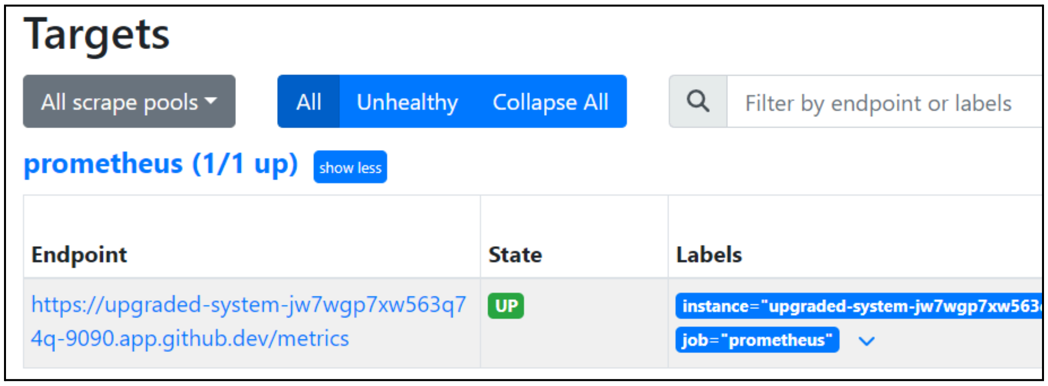
10. To access Prometheus, visit your Forwarded Address associated with port 9090 in the browser. By default, you will be taken to the *Graph* tab of Prometheus.



11. To see the configured targets, go to ***Status > Targets***



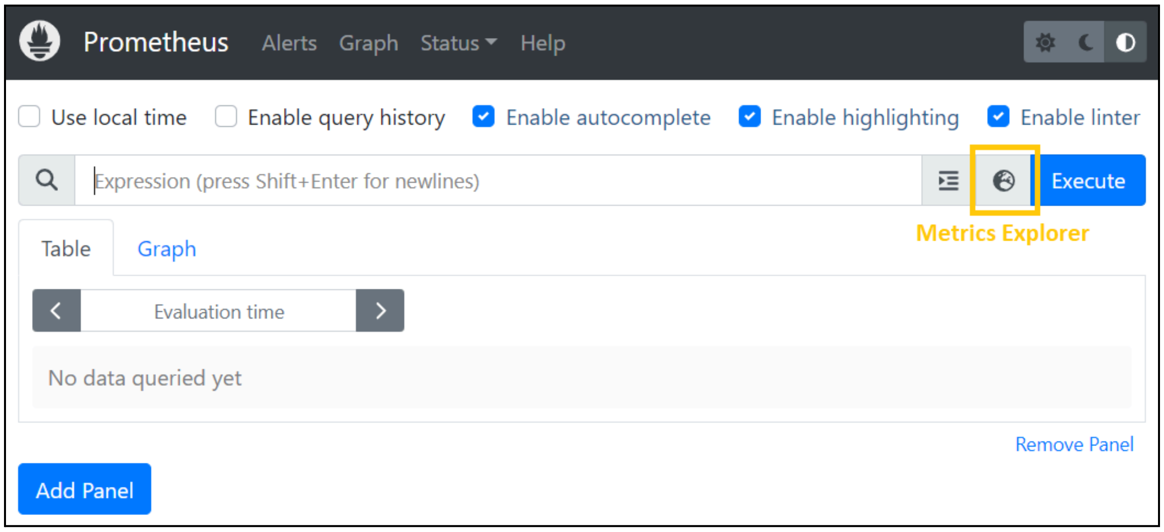
You will find the targets Endpoint Prometheus is scrapping the metrics from along with their State & Labels.



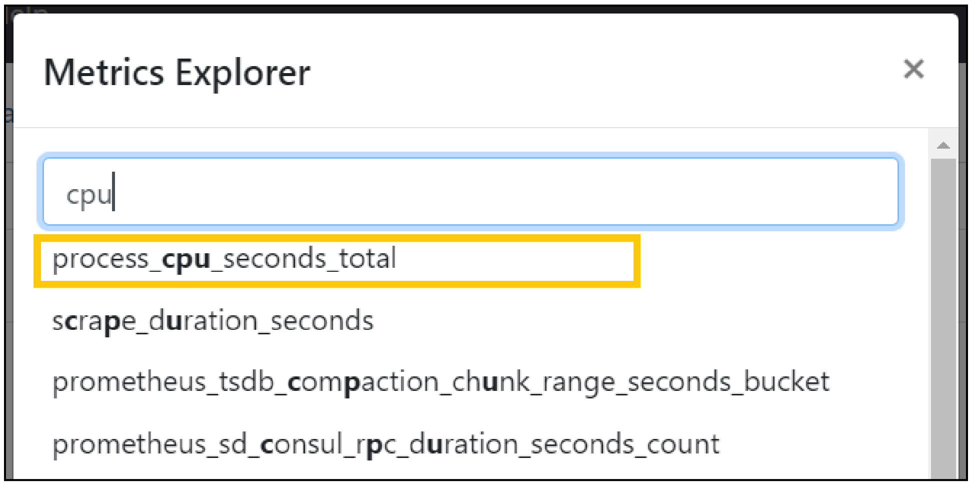
Endpoint	State	Labels
https://upgraded-system-jw7wgp7xw563q74q-9090.app.github.dev/metrics	UP	instance="upgraded-system-jw7wgp7xw563 job="prometheus"

12. To query the scrapped metrics, go to the ***Graph*** tab and click on the ***Metrics Explorer*** icon. Select a metric and press Execute.

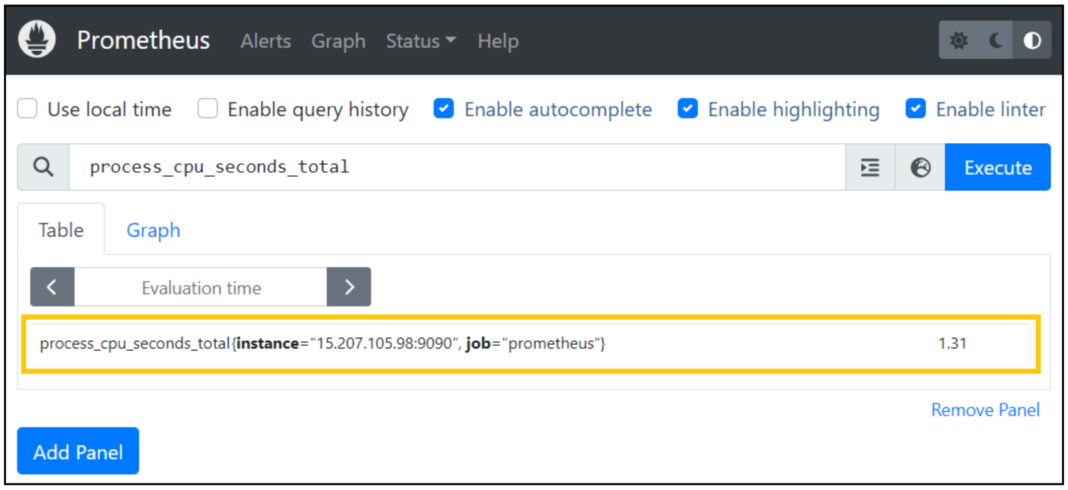
Click on Metrics Explorer



Select a metric


Metrics Explorer
cpu
process_cpu_seconds_total
scrape_duration_seconds
prometheus_tsdb_compaction_chunk_range_seconds_bucket
prometheus_sd_consul_rpc_duration_seconds_count

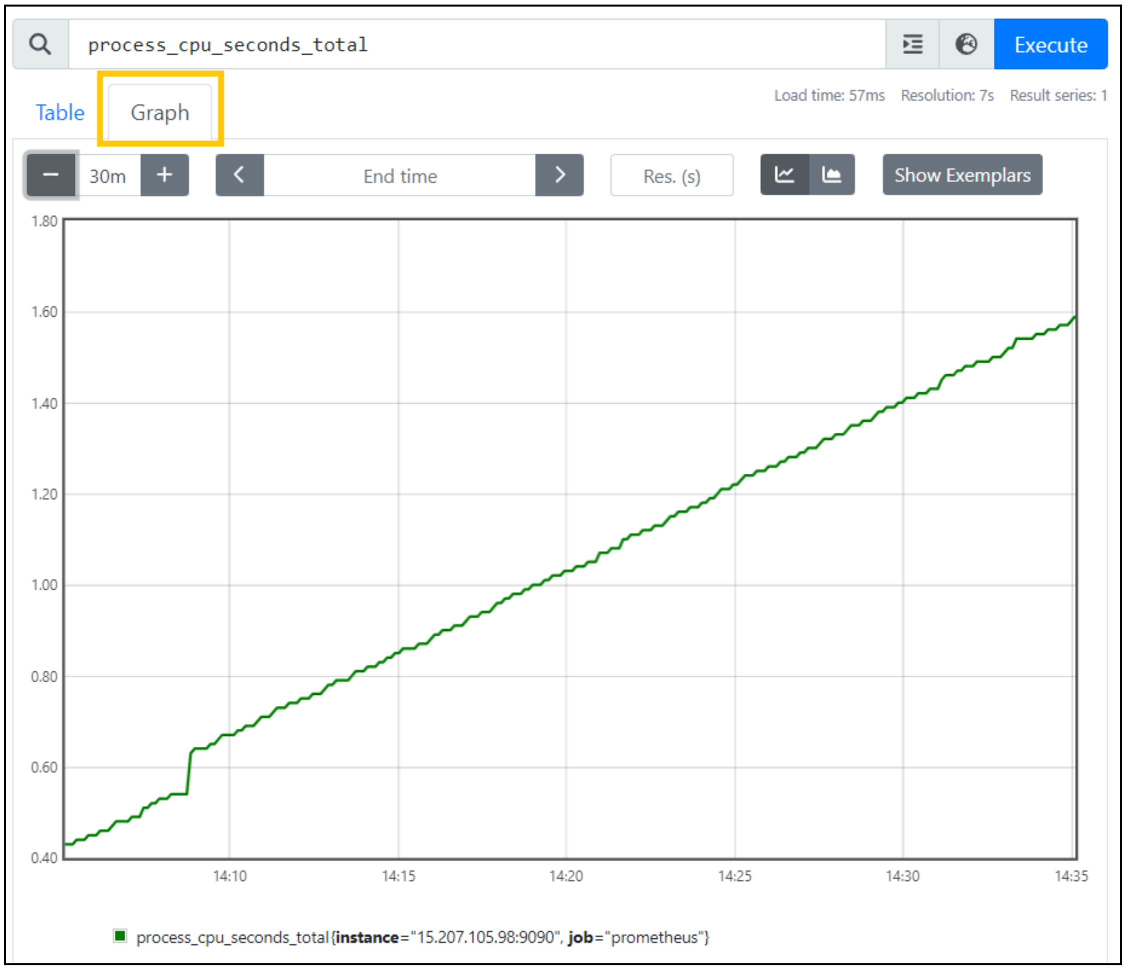
You will see the details of the metric such as the instance, job, and its last scrapped value.



```
process_cpu_seconds_total
```

```
process_cpu_seconds_total{instance="15.207.105.98:9090", job="prometheus"}    1.31
```

13. You can select the Graph option to see its visualization.



```
process_cpu_seconds_total
```

14. To see the Prometheus supported metrics format, go to **<your-forwarded-address>/metrics.** For example: *https://upgraded-system-jw7wgp7xw563q74q-9090.app.github.dev/metrics*

A simple example is given below.
Note that:

- Details of a metric is present in lines above that
- HELP will contain the metric description
- TYPE will contain the metric data type
    - Different data types include - *counter, gauge, summary, histogram, etc*.
- At last, the value of the metric

```
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.24
```

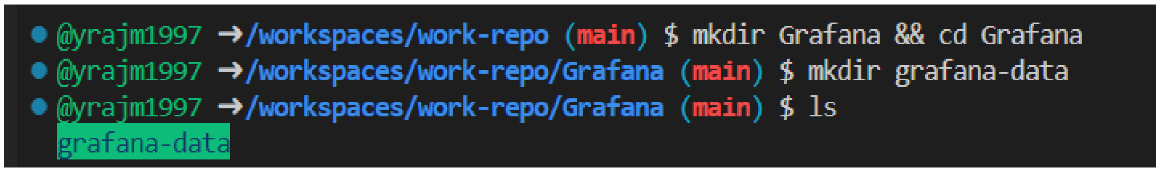
II. Setup Grafana using Codespaces

1. In Codespace, move outside of Prometheus directory, then create a parent directory for Grafana and move into it

```
mkdir Grafana && cd Grafana
```

2. Create a directory named **grafana-data**, to let Grafana store the data.

```
mkdir grafana-data
```



```
● @yrajm1997 ➜ /workspaces/work-repo (main) $ mkdir Grafana && cd Grafana
● @yrajm1997 ➜ /workspaces/work-repo/Grafana (main) $ mkdir grafana-data
● @yrajm1997 ➜ /workspaces/work-repo/Grafana (main) $ ls
  grafana-data
```

Note that the grafana data directory will need to be mounted while running the Grafana container.

3. Create a file, **env.list**, to store environment variables, to update a few environment variables while running the container.

```
touch env.list
```

4. Add below content to it.

env.list: [Make sure to remove the comments/any-extra-space given below]

```
GF_SMTP_ENABLED=true
GF_SMTP_HOST=smtp.gmail.com:587
GF_SMTP_USER=<your-mail-id>@gmail.com          # replace with your mail id
GF_SMTP_PASSWORD=<your-account-app_password>   # Refer below how to get this
GF_SMTP_FROM_ADDRESS=<your-mail-id>@gmail.com  # replace with your mail id
GF_SMTP_FROM_NAME=Grafana
GF_SMTP_SKIP_VERIFY=false
```

Create and use app password:

An app password is a 16-digit passcode that gives a less secure app or device permission to access your Google Account.

Note: App passwords can only be used with Google accounts that have 2-Step Verification turned on.

To get your account App Password,

1. Go to your Google Account.
2. Select *Security*.
3. In the search bar, search for *App passwords*.
4. Enter a name that helps you remember where you'll use the app password.
5. Select *Generate*.
6. Copy the generated app password. Remove spaces in between, it is a 16-character code.
7. Select Done.

If you've set up 2-Step Verification but can't find the option to add an app password, it might be because:

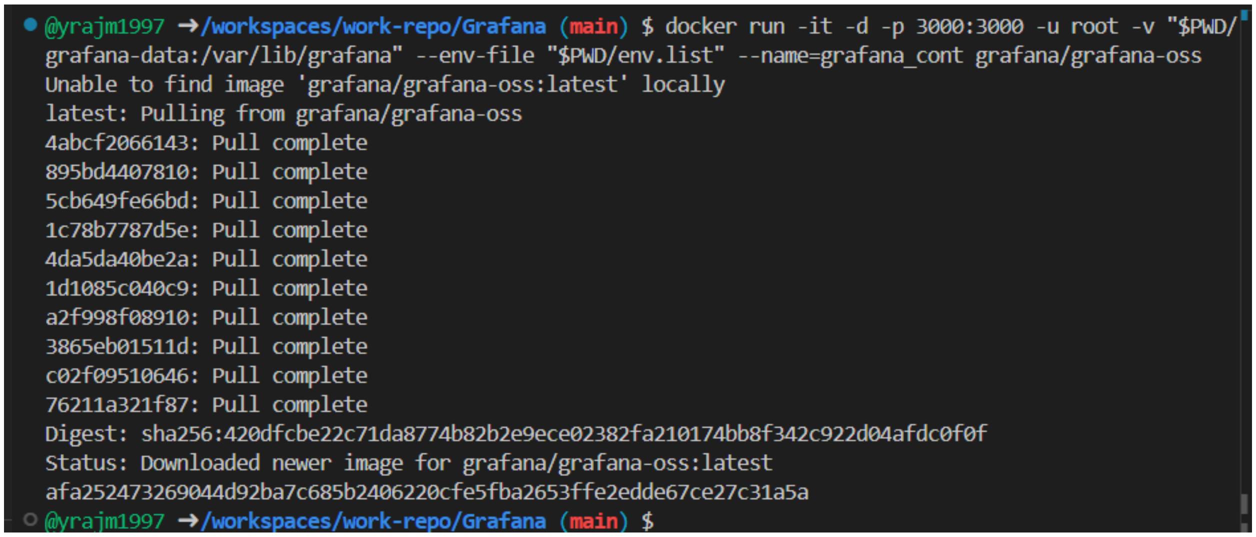
1. Your Google Account has 2-Step Verification set up only for security keys.
2. You're logged into a work, school, or another organization account.
3. Your Google Account has Advanced Protection.

5. From the *Grafana* directory, run the below command to start Grafana

```
docker run -it -d -p 3000:3000 -u root -v "$PWD/grafana-data:/var/lib/grafana"
--env-file "$PWD/env.list" --name=grafana_cont grafana/grafana-oss
```

Here, the container will run

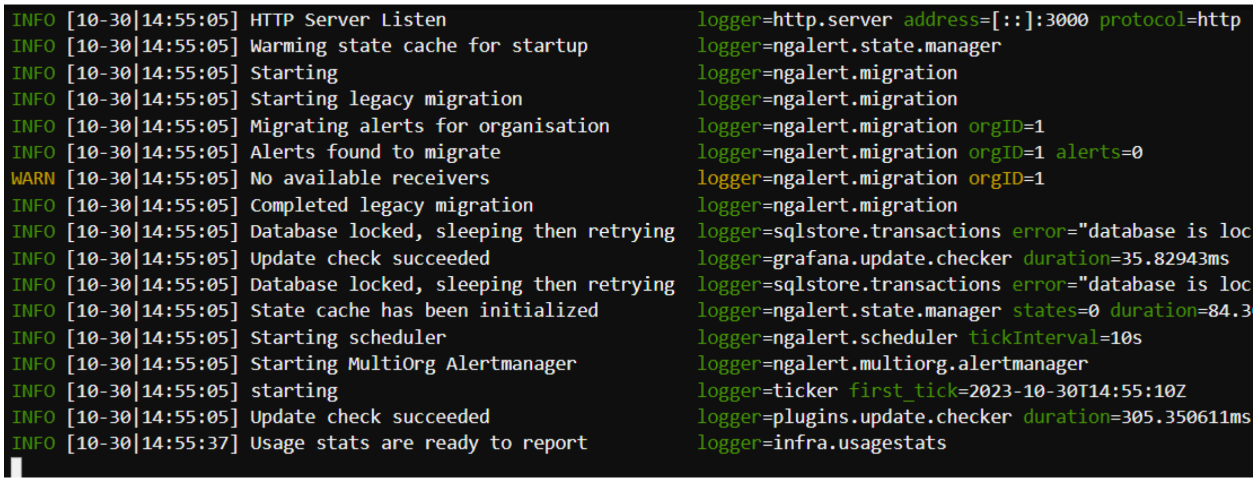
- with a pseudo terminal (`-it`)
- in a detached mode (`-d`)
- with container port mapped to host port (`-p 3000:3000`)
- as a root user (`-u root`)
- with grafana-data directory mounted at specified path in container
- with some env variables set/overwritten as per the file (`--env-file "$PWD/env.list"`)
- with name grafana_cont (`--name=grafana_cont`)
- using grafana open source docker image (`grafana/grafana-oss`)



```
● @yrajm1997 →/workspaces/work-repo/Grafana (main) $ docker run -it -d -p 3000:3000 -u root -v "$PWD/
  grafana-data:/var/lib/grafana" --env-file "$PWD/env.list" --name=grafana_cont grafana/grafana-oss
  Unable to find image 'grafana/grafana-oss:latest' locally
  latest: Pulling from grafana/grafana-oss
  4abcf2066143: Pull complete
  895bd4407810: Pull complete
  5cb649fe66bd: Pull complete
  1c78b7787d5e: Pull complete
  4da5da40be2a: Pull complete
  1d1085c040c9: Pull complete
  a2f998f08910: Pull complete
  3865eb01511d: Pull complete
  c02f09510646: Pull complete
  76211a321f87: Pull complete
  Digest: sha256:420dfcbe22c71da8774b82b2e9ece02382fa210174bb8f342c922d04afdc0f0f
  Status: Downloaded newer image for grafana/grafana-oss:latest
  afa252473269044d92ba7c685b2406220cfe5fba2653ffe2edde67ce27c31a5a
○ @yrajm1997 →/workspaces/work-repo/Grafana (main) $
```

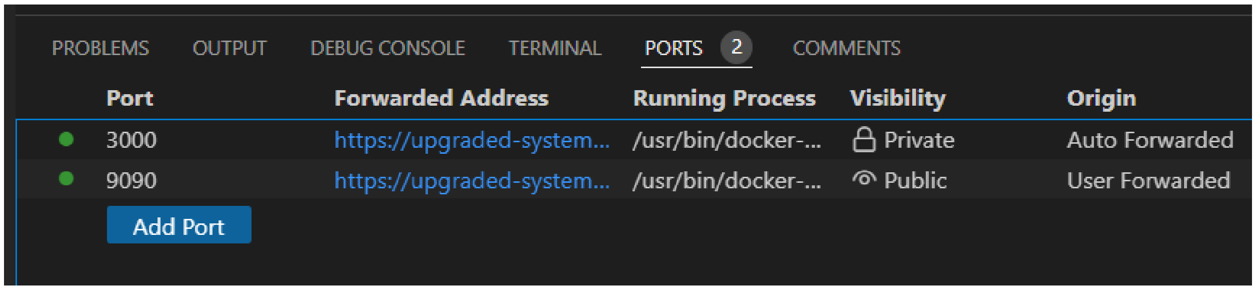
6. Check the logs of the running container

`docker logs -f grafana_cont`



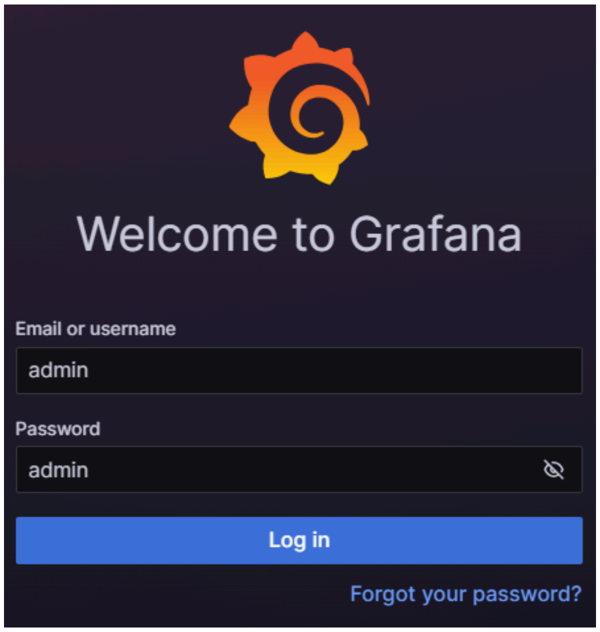
```
INFO [10-30|14:55:05] HTTP Server Listen                         logger=http.server address=[::]:3000 protocol=http
INFO [10-30|14:55:05] Warming state cache for startup            logger=ngalert.state.manager
INFO [10-30|14:55:05] Starting                                   logger=ngalert.migration
INFO [10-30|14:55:05] Starting legacy migration                  logger=ngalert.migration
INFO [10-30|14:55:05] Migrating alerts for organisation          logger=ngalert.migration orgID=1
INFO [10-30|14:55:05] Alerts found to migrate                    logger=ngalert.migration orgID=1 alerts=0
WARN [10-30|14:55:05] No available receivers                     logger=ngalert.migration orgID=1
INFO [10-30|14:55:05] Completed legacy migration                 logger=ngalert.migration
INFO [10-30|14:55:05] Database locked, sleeping then retrying    logger=sqlstore.transactions error="database is loc
INFO [10-30|14:55:05] Update check succeeded                     logger=grafana.update.checker duration=35.82943ms
INFO [10-30|14:55:05] Database locked, sleeping then retrying    logger=sqlstore.transactions error="database is loc
INFO [10-30|14:55:05] State cache has been initialized           logger=ngalert.state.manager states=0 duration=84.3
INFO [10-30|14:55:05] Starting scheduler                         logger=ngalert.scheduler tickInterval=10s
INFO [10-30|14:55:05] Starting MultiOrg Alertmanager             logger=ngalert.multiorg.alertmanager
INFO [10-30|14:55:05] starting                                   logger=ticker first_tick=2023-10-30T14:55:10Z
INFO [10-30|14:55:05] Update check succeeded                     logger=plugins.update.checker duration=305.350611ms
INFO [10-30|14:55:37] Usage stats are ready to report            logger=infra.usagestats
```

7. Now Grafana is also running in a container. Check the PORTS tab, one more port should be forwarded automatically for grafana i.e. port 3000 (else you can add it yourself).
Change its visibility also to *Public*.



Port	Forwarded Address	Running Process	Visibility	Origin
● 3000	https://upgraded-system...	/usr/bin/docker-...	Private	Auto Forwarded
● 9090	https://upgraded-system...	/usr/bin/docker-...	Public	User Forwarded

8. To access Grafana, visit your 'Forwarded Address' associated with port 3000 in the browser. While opening Grafana the first time, it will ask you to login. Give username and password both as ***admin***

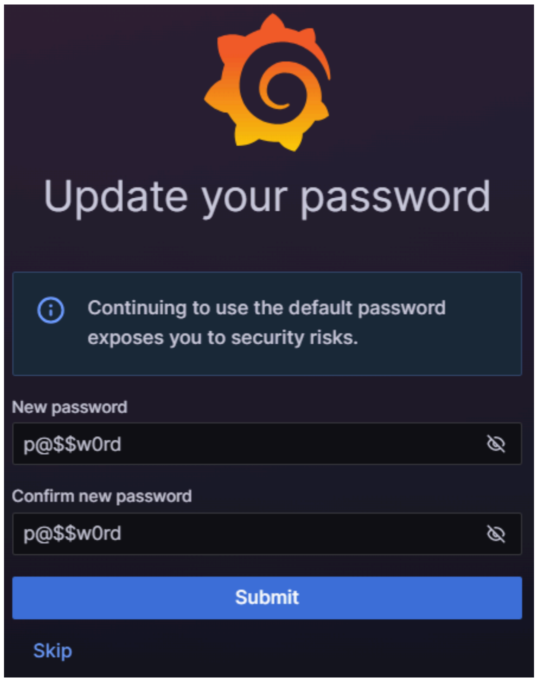


Welcome to Grafana

Email or username: admin

Password: admin

Then it will prompt you to change the password



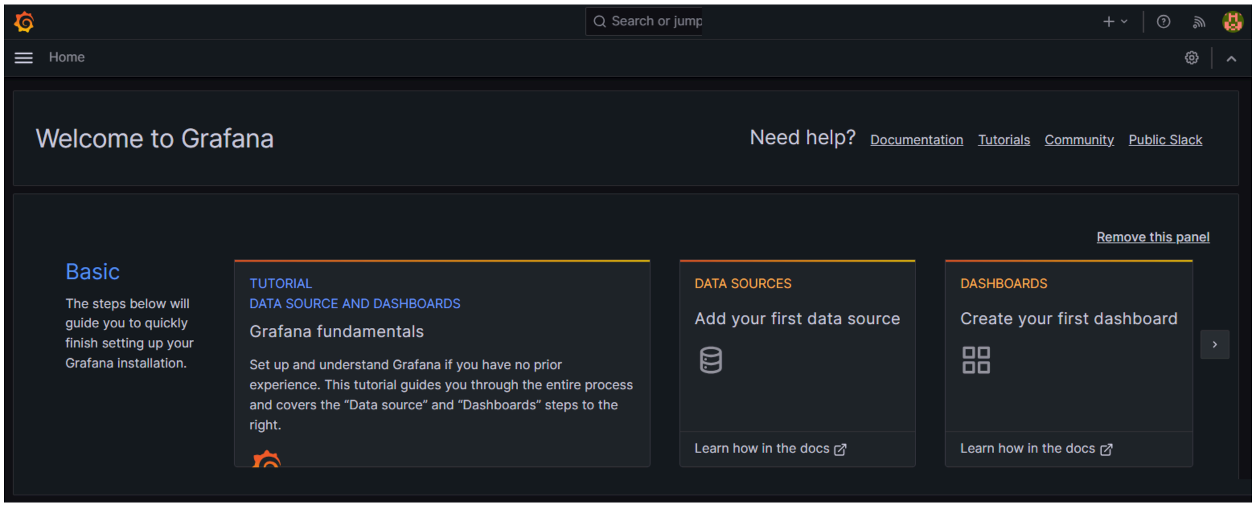
Update your password

Continuing to use the default password exposes you to security risks.

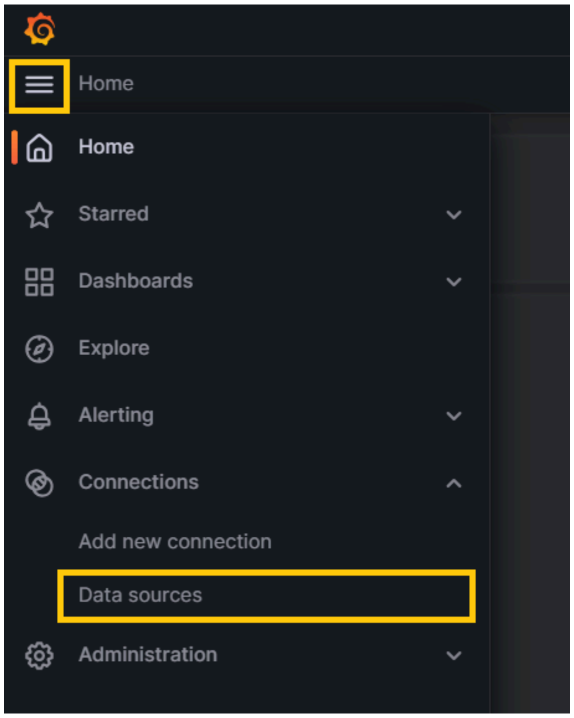
New password: p@$$w0rd

Confirm new password: p@$$w0rd

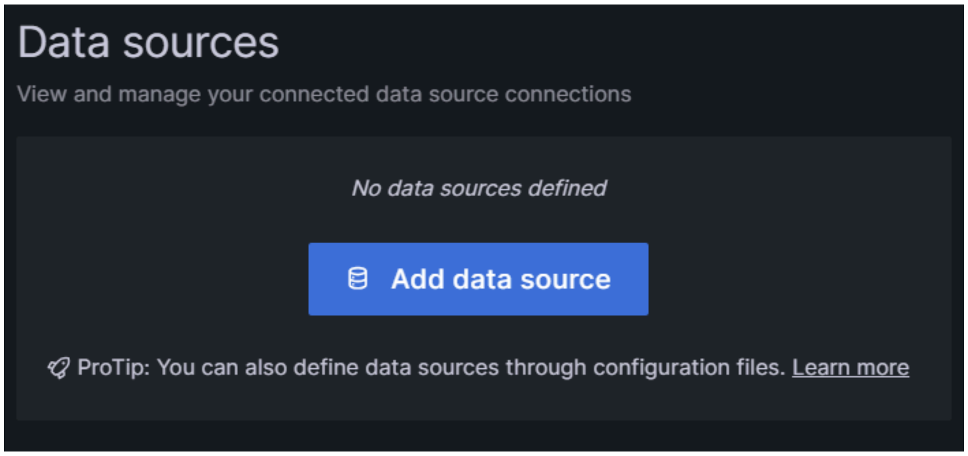
After that you will be directed to the *Home page* of Grafana.


Search or jump
Home
Welcome to Grafana
Need help? Documentation Tutorials Community Public Slack
Remove this panel
Basic
The steps below will guide you to quickly finish setting up your Grafana installation.
TUTORIAL
DATA SOURCE AND DASHBOARDS
Grafana fundamentals
Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.
DATA SOURCES
Add your first data source
Learn how in the docs
DASHBOARDS
Create your first dashboard
Learn how in the docs

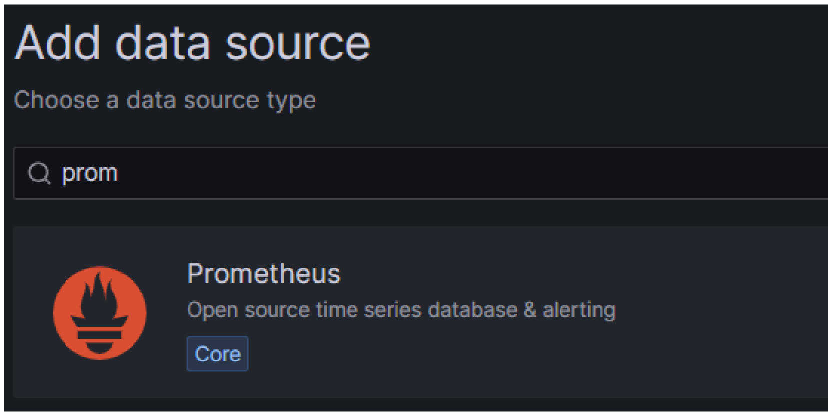
9. To add a Data Source to Grafana, go to ***Home > Connections > Data sources***



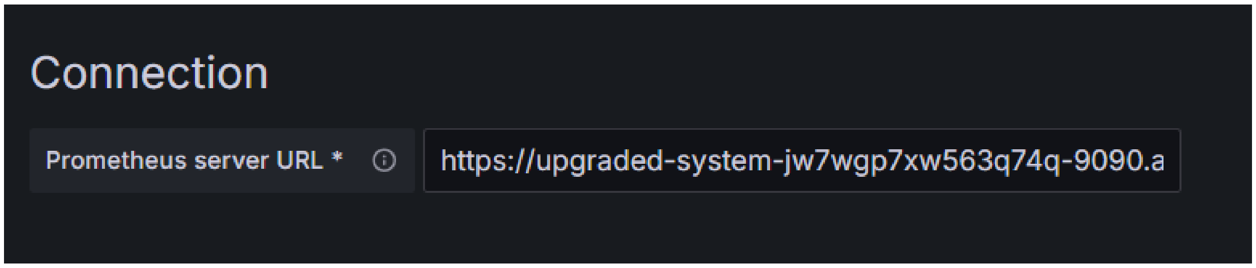
Select *Add data source*


Data sources
View and manage your connected data source connections
No data sources defined
Add data source
ProTip: You can also define data sources through configuration files. Learn more

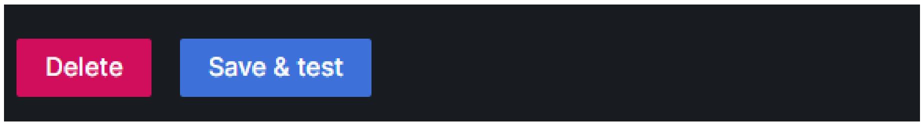
Type Prometheus and select it


Add data source
Choose a data source type
prom
Prometheus
Open source time series database & alerting
Core

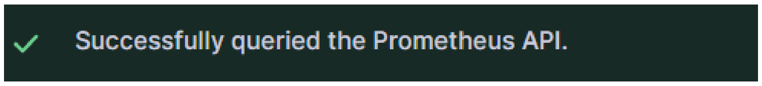
In ***Connection***, give the url for your Prometheus service. Keep rest settings default and select ***Save & test***.



Prometheus server URL * https://upgraded-system-jw7wgp7xw563q74q-9090.a

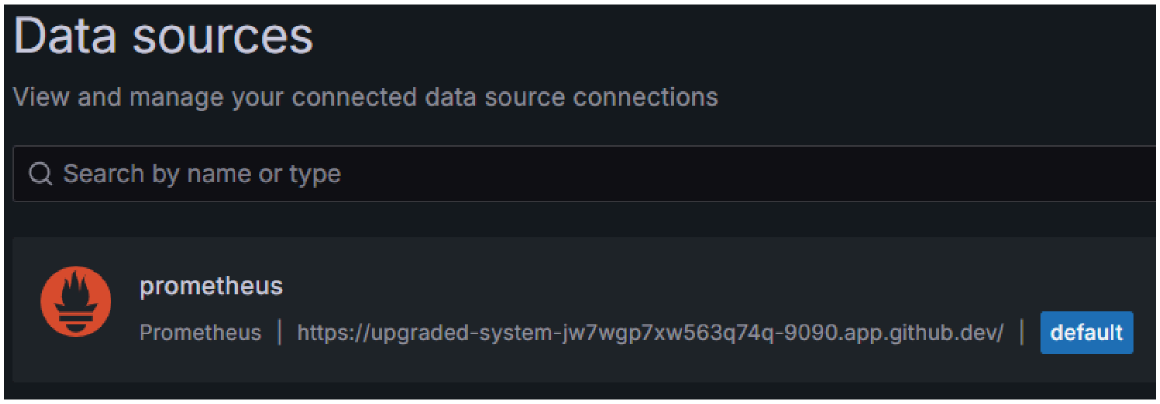


You will get a confirmation message.



Successfully queried the Prometheus API.

Then go back to *Data sources*, Prometheus should be added as one of the data sources.



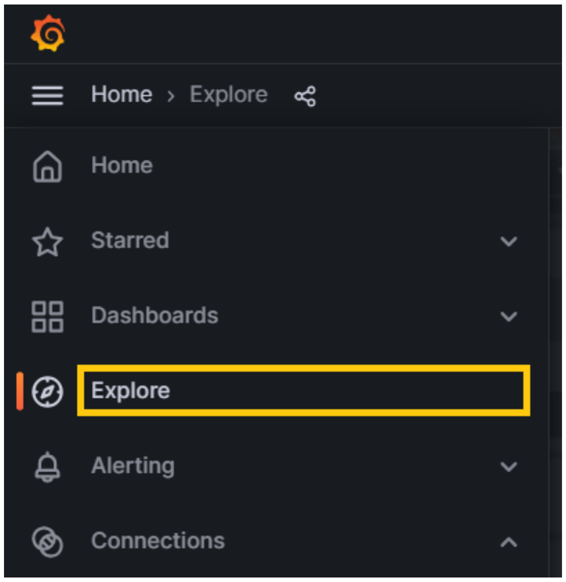
Data sources

View and manage your connected data source connections

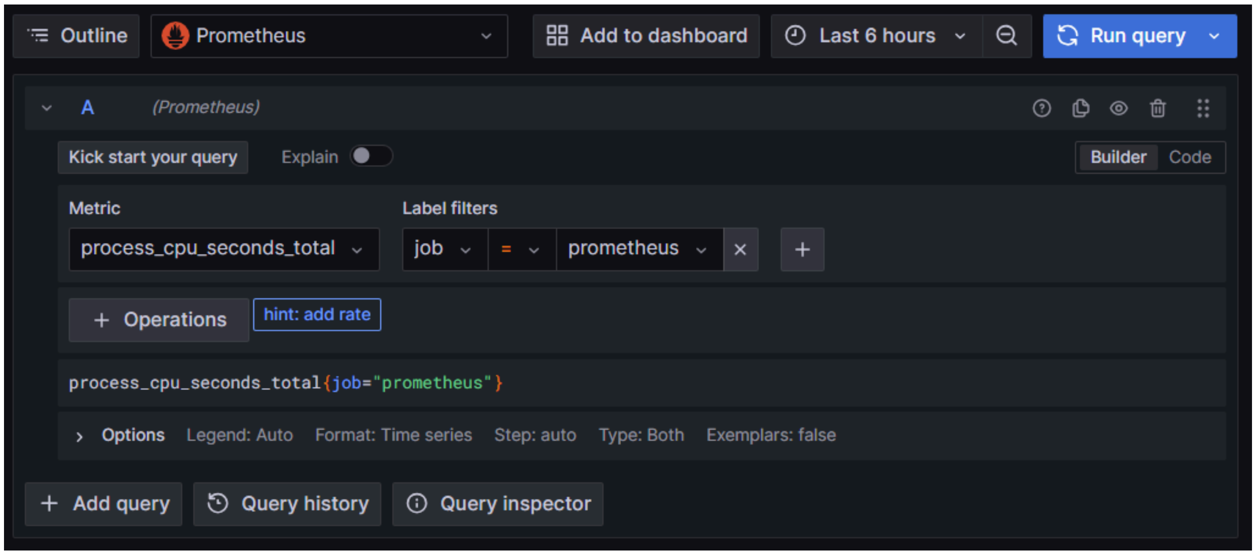
prometheus

Prometheus | https://upgraded-system-jw7wgp7xw563q74q-9090.app.github.dev/ | default

10. To explore the metrics coming from Prometheus, go to ***Home > Explore***



Here, by default Prometheus has been selected as the data source. You need to select the Metric, Label filters, and select ***Run query***.



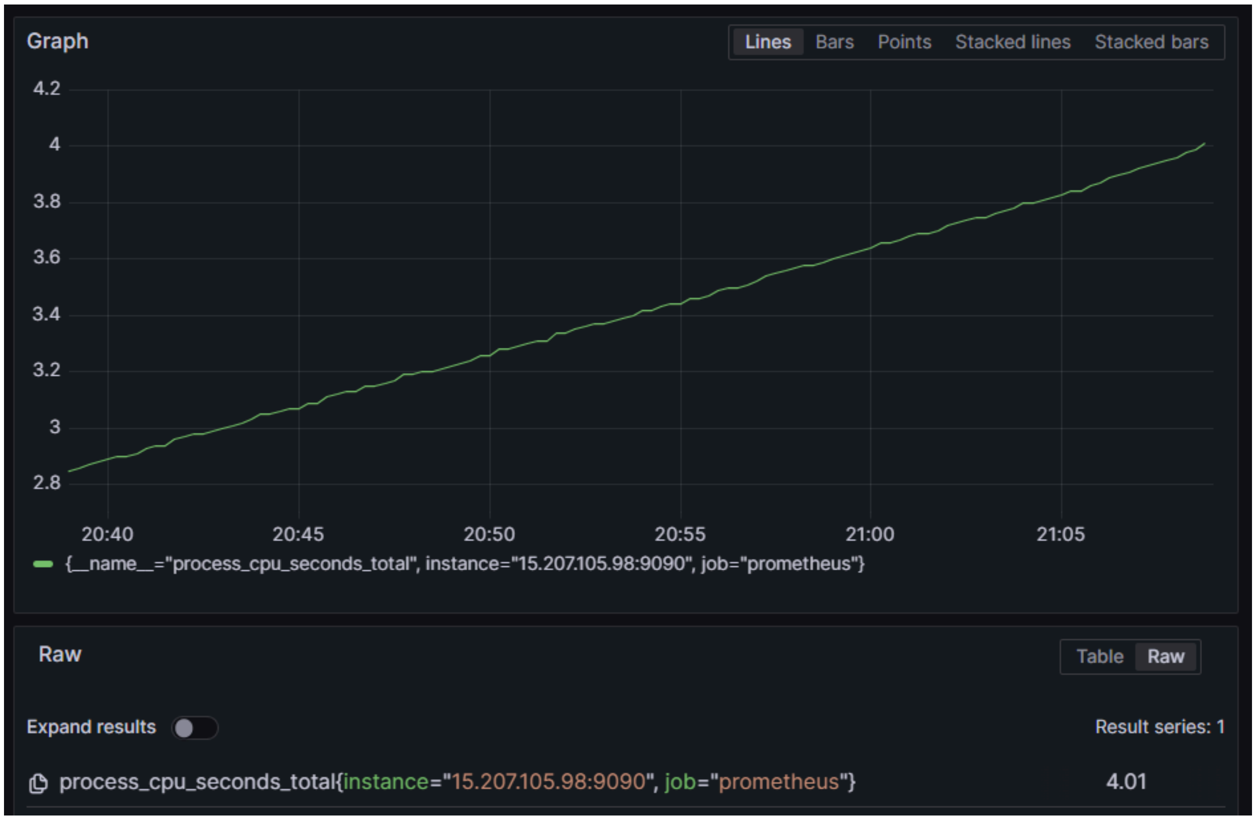
Metric: process_cpu_seconds_total

Label filters: job = prometheus

```
process_cpu_seconds_total{job="prometheus"}
```

Options Legend: Auto Format: Time series Step: auto Type: Both Exemplars: false

A real-time visualization graph will appear for that metric.



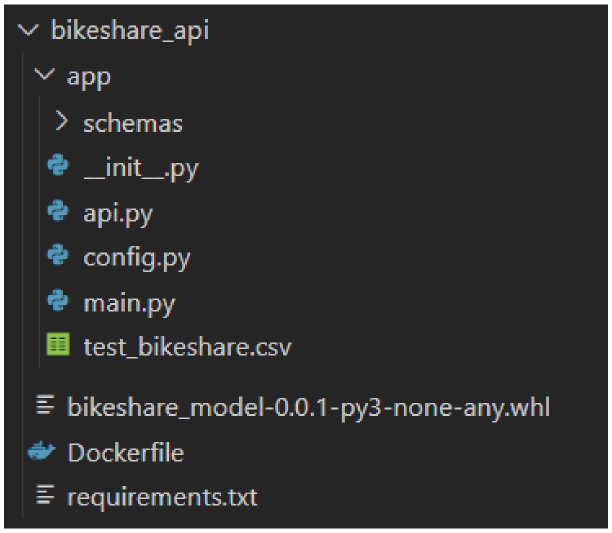
{__name__="process_cpu_seconds_total", instance="15.207.105.98:9090", job="prometheus"}

Result series: 1

```
process_cpu_seconds_total{instance="15.207.105.98:9090", job="prometheus"}    4.01
```

III. Run an application on Codespace

1. Download the folder bikeshare_api shared along with this document and upload it on Codespace
   It contains FastAPI files for the Bikeshare application.



```
bikeshare_api
  app
    schemas
    __init__.py
    api.py
    config.py
    main.py
    test_bikeshare.csv
  bikeshare_model-0.0.1-py3-none-any.whl
  Dockerfile
  requirements.txt
```

2. The main.py file has already been updated to include the prometheus-client library that will help to present the metrics in the Prometheus supported format, whenever a request is made to **/metrics** endpoint.
   Changes in the main.py file:

```
import pandas as pd
from sklearn.metrics import r2_score
import prometheus_client as prom

test_data = pd.read_csv(curr_path + "/test_bikeshare.csv")
r2_metric  =  prom.Gauge('bikeshare_r2_score',  'R2  score  for  random  100  test
samples')

# Function for updating metrics
def update_metrics():
    test = test_data.sample(100)
    test_feat = test.drop('cnt', axis=1)
    test_cnt = test['cnt'].values
    test_pred = make_prediction(input_data=test_feat)['predictions']
    r2 = r2_score(test_cnt, test_pred).round(3)
    r2_metric.set(r2)

@app.get("/metrics")
async def get_metrics():
    update_metrics()
    return Response(media_type="text/plain",
                    content = prom.generate_latest())
```

   Here,

- the prometheus-client library is imported as prom
- a Gauge metric object is created
- the value of the Gauge metric object will be updated with the R2 score whenever the *update_metrics()* function is called
- to present metrics in Prometheus supported format, *prom.generate_latest()* is used

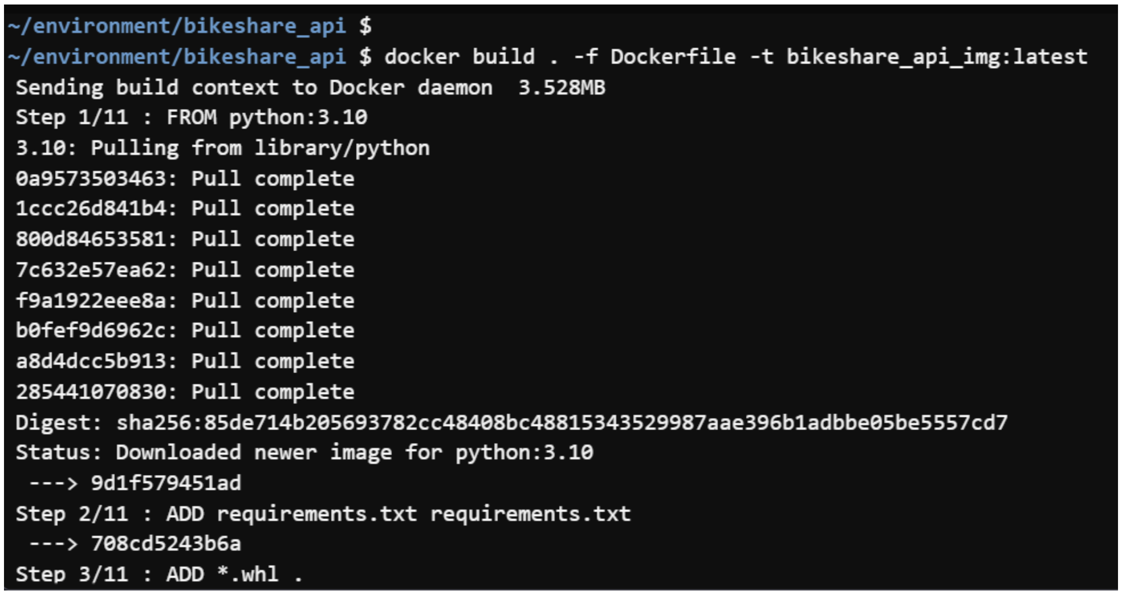
3. Dockerfile

```
# pull python base image
FROM python:3.10

ADD requirements.txt requirements.txt
ADD *.whl .

# update pip
RUN pip install --upgrade pip

# install dependencies
RUN pip install -r requirements.txt

RUN rm *.whl

# copy application files & change/give ownership to myuser
COPY app/. app/.

# expose port for application
EXPOSE 8080

# start fastapi application
CMD ["python", "app/main.py"]
```

4. On Codespace, build a docker image for bikeshare application

```
cd bikeshare_api
```

```
docker build . -f Dockerfile -t bikeshare_api_img:latest
```

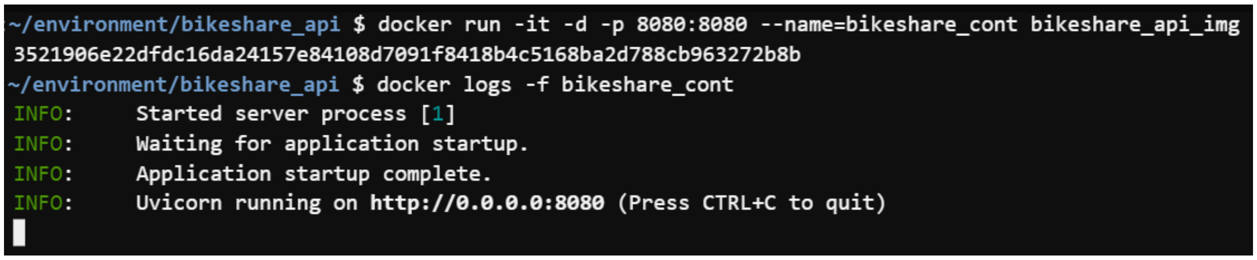


```
~/environment/bikeshare_api $
~/environment/bikeshare_api $ docker build . -f Dockerfile -t bikeshare_api_img:latest
Sending build context to Docker daemon  3.528MB
Step 1/11 : FROM python:3.10
3.10: Pulling from library/python
0a9573503463: Pull complete
1ccc26d841b4: Pull complete
800d84653581: Pull complete
7c632e57ea62: Pull complete
f9a1922eee8a: Pull complete
b0fef9d6962c: Pull complete
a8d4dcc5b913: Pull complete
285441070830: Pull complete
Digest: sha256:85de714b205693782cc48408bc48815343529987aae396b1adbbe05be5557cd7
Status: Downloaded newer image for python:3.10
 ---> 9d1f579451ad
Step 2/11 : ADD requirements.txt requirements.txt
 ---> 708cd5243b6a
Step 3/11 : ADD *.whl .
```

5. Start the application container and check the logs

   `docker run -it -d -p 8080:8080 --name=bikeshare_cont bikeshare_api_img`

   `docker logs -f bikeshare_cont`



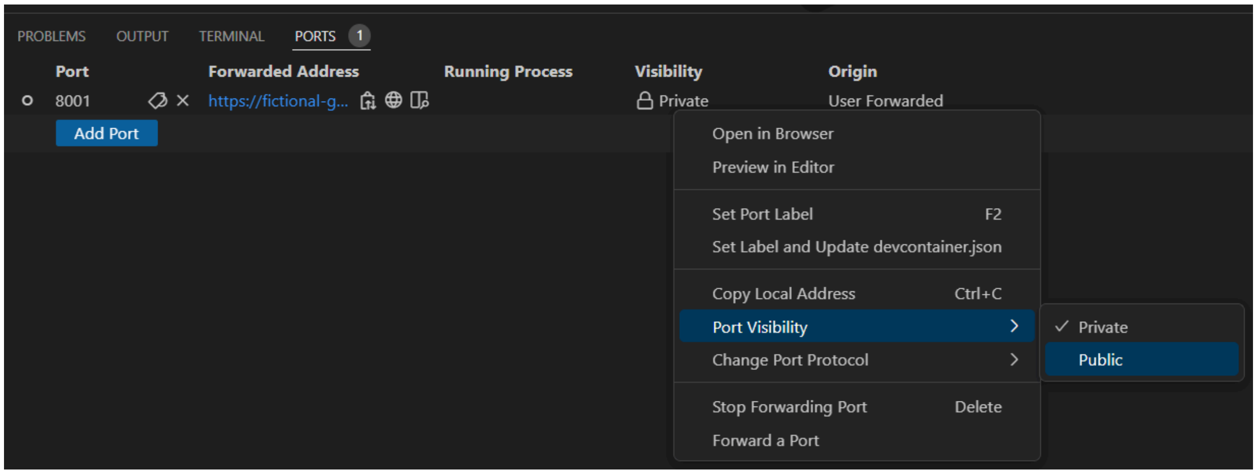
```
~/environment/bikeshare_api $ docker run -it -d -p 8080:8080 --name=bikeshare_cont bikeshare_api_img
3521906e22dfdc16da24157e84108d7091f8418b4c5168ba2d788cb963272b8b
~/environment/bikeshare_api $ docker logs -f bikeshare_cont
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

6. Then go to **PORTS** tab of you Codespaces (beside the Terminal tab)

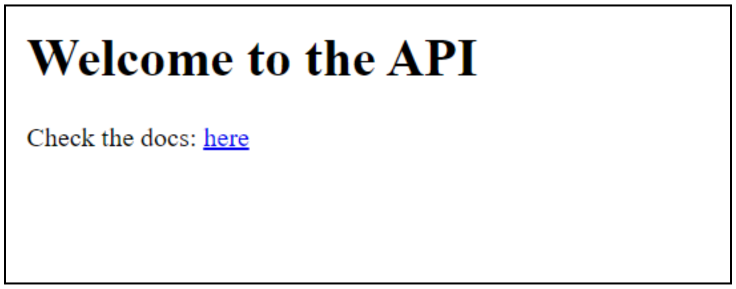
   There must be a port being forwarded for your container, else you can click on Add Port and provide the port number where the application is running.

   By default the **Visibility is Private**. Right click on it >> Port Visibility >> Public.

   After that copy the forwarded address url link.



7. Paste the forwarded address url in the browser. Application should be accessible.



Welcome to the API

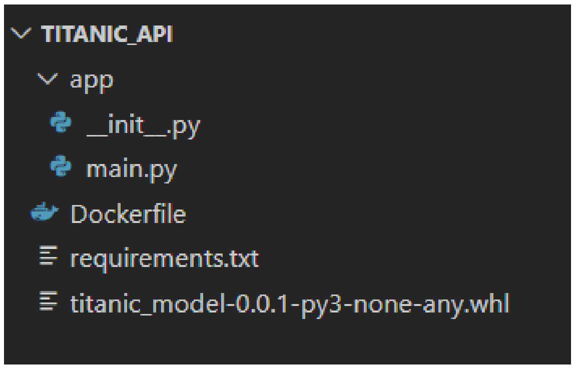
Check the docs: here

8. To access the application metrics, go to***/metrics*** endpoint.
   For example: https://upgraded-telegram-5prp97rwpj9cv4v9-8001.app.github.dev**/metrics**

```
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 2.35
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 10.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 32768.0
# HELP bikeshare_r2_score R2 score for random 100 test samples
# TYPE bikeshare_r2_score gauge
bikeshare_r2_score 0.911
```

IV. Run another application on Codespace

1. Download the folder titanic_api shared along with this document.
   It contains Gradio+FastAPI implementation of the Titanic Survival prediction application.



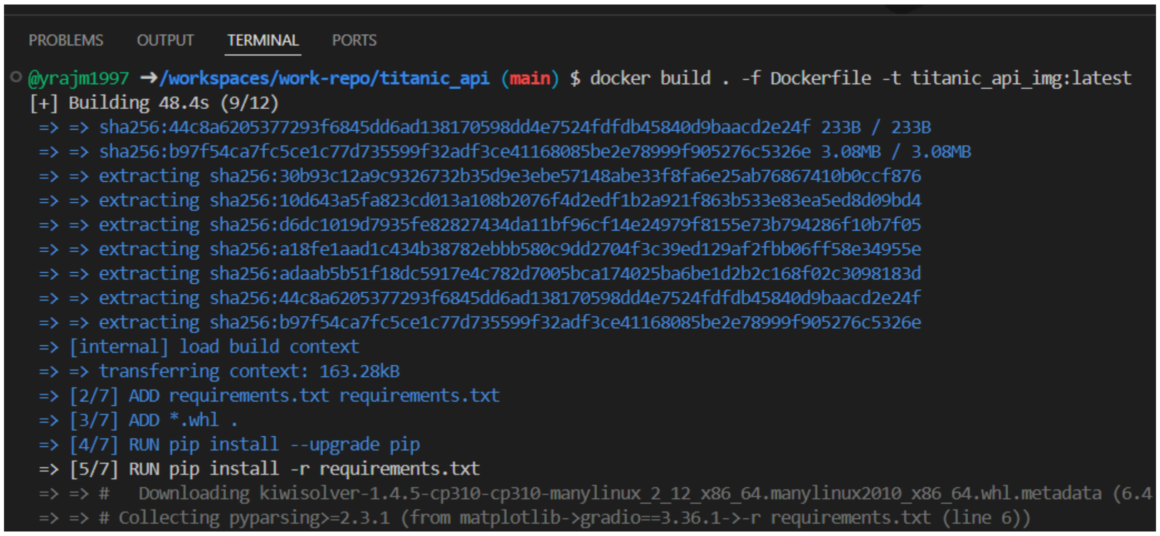
```
TITANIC_API
  app
    __init__.py
    main.py
  Dockerfile
  requirements.txt
  titanic_model-0.0.1-py3-none-any.whl
```

2. The main.py file has already been updated to include the prometheus-client library that will help to present the metrics in the Prometheus supported format, whenever a request is made to **/metrics** endpoint.
   Also, the Gradio interface is mounted within the FastAPI app as per this Reference > *Mounting Within Another FastAPI App*

3. Upload titanic_api folder to your Codespace.
4. On Codespaces, build a docker image for titanic application

   ```
   cd titanic_api
   ```

   ```
   docker build . -f Dockerfile -t titanic_api_img:latest
   ```

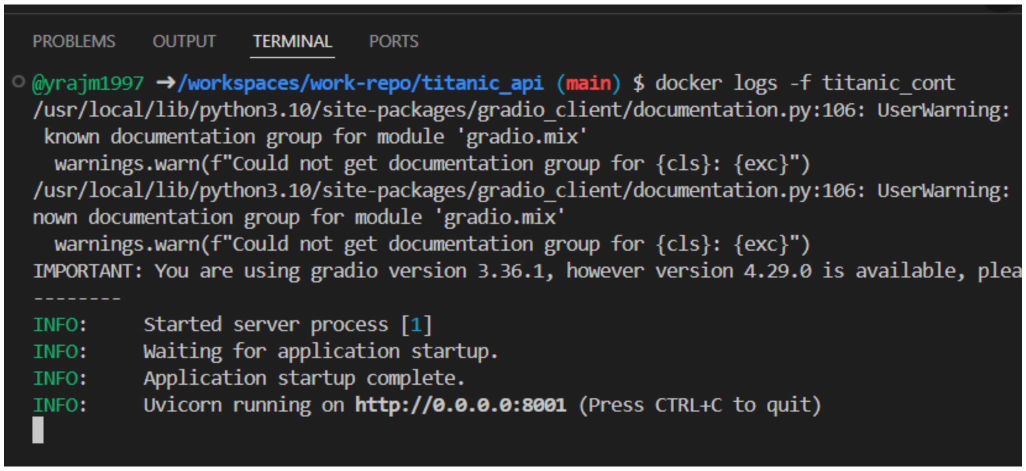


```
@yrajm1997 ➜/workspaces/work-repo/titanic_api (main) $ docker build . -f Dockerfile -t titanic_api_img:latest
[+] Building 48.4s (9/12)
 => => sha256:44c8a6205377293f6845dd6ad138170598dd4e7524fdfdb45840d9baacd2e24f 233B / 233B
 => => sha256:b97f54ca7fc5ce1c77d735599f32adf3ce41168085be2e78999f905276c5326e 3.08MB / 3.08MB
 => => extracting sha256:30b93c12a9c9326732b35d9e3ebe57148abe33f8fa6e25ab76867410b0ccf876
 => => extracting sha256:10d643a5fa823cd013a108b2076f4d2edf1b2a921f863b533e83ea5ed8d09bd4
 => => extracting sha256:d6dc1019d7935fe82827434da11bf96cf14e24979f8155e73b794286f10b7f05
 => => extracting sha256:a18fe1aad1c434b38782ebbb580c9dd2704f3c39ed129af2fbb06ff58e34955e
 => => extracting sha256:adaab5b51f18dc5917e4c782d7005bca174025ba6be1d2b2c168f02c3098183d
 => => extracting sha256:44c8a6205377293f6845dd6ad138170598dd4e7524fdfdb45840d9baacd2e24f
 => => extracting sha256:b97f54ca7fc5ce1c77d735599f32adf3ce41168085be2e78999f905276c5326e
 => [internal] load build context
 => => transferring context: 163.28kB
 => [2/7] ADD requirements.txt requirements.txt
 => [3/7] ADD *.whl .
 => [4/7] RUN pip install --upgrade pip
 => [5/7] RUN pip install -r requirements.txt
 => => #   Downloading kiwisolver-1.4.5-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.whl.metadata (6.4
 => => # Collecting pyparsing>=2.3.1 (from matplotlib->gradio==3.36.1->-r requirements.txt (line 6))
```

5. Start the application container and check the logs

   ```
   docker run -it -d -p 8001:8001 --name=titanic_cont titanic_api_img
   ```

   ```
   docker logs -f titanic_cont
   ```



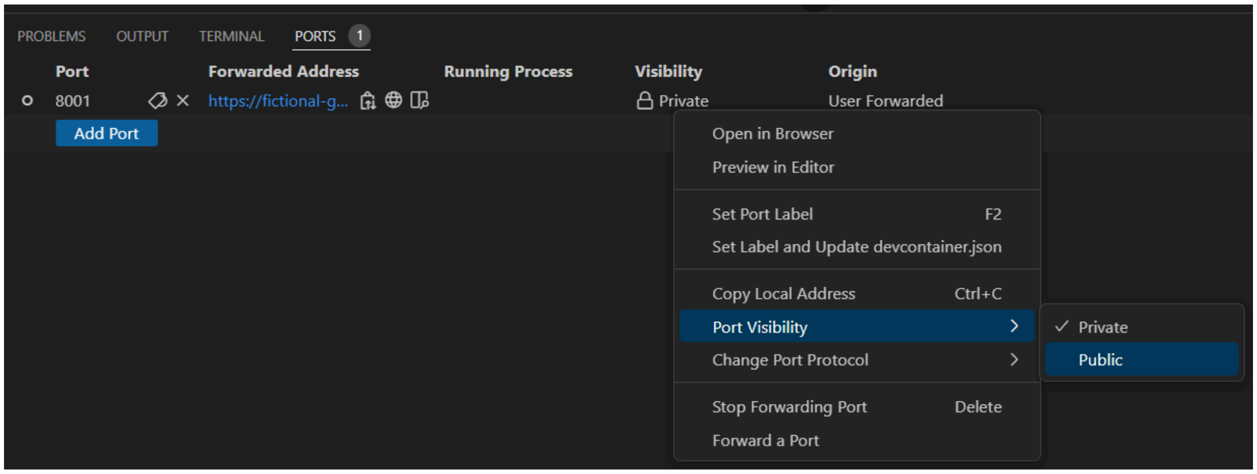
```
PROBLEMS   OUTPUT   TERMINAL   PORTS

@yrajm1997 ➜/workspaces/work-repo/titanic_api (main) $ docker logs -f titanic_cont
/usr/local/lib/python3.10/site-packages/gradio_client/documentation.py:106: UserWarning:
 known documentation group for module 'gradio.mix'
  warnings.warn(f"Could not get documentation group for {cls}: {exc}")
/usr/local/lib/python3.10/site-packages/gradio_client/documentation.py:106: UserWarning:
nown documentation group for module 'gradio.mix'
  warnings.warn(f"Could not get documentation group for {cls}: {exc}")
IMPORTANT: You are using gradio version 3.36.1, however version 4.29.0 is available, plea
--------
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
```

6. Then go to **PORTS** tab of you Codespaces (beside the Terminal tab)

   There must be a port being forwarded for your container, else you can click on Add Port and provide the port number where the application is running.

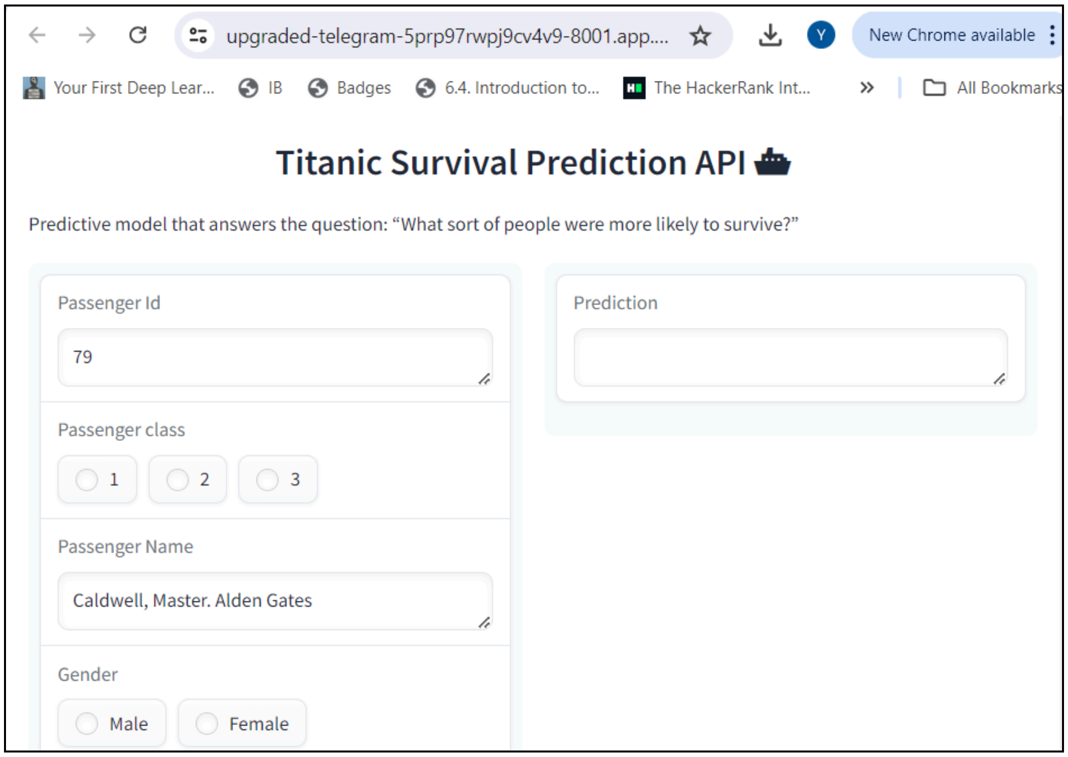
   By default the **Visibility is Private**. Right click on it >> Port Visibility >> Public.

   After that copy the forwarded address url link.



Port	Forwarded Address	Running Process	Visibility	Origin
8001	https://fictional-g...		Private	User Forwarded

7. Paste the forwarded address url in the browser. Application should be accessible.


upgraded-telegram-5prp97rwpj9cv4v9-8001.app....

Titanic Survival Prediction API

Predictive model that answers the question: "What sort of people were more likely to survive?"

Passenger Id: 79

Passenger class: 1 / 2 / 3

Passenger Name: Caldwell, Master. Alden Gates

Gender: Male / Female

Prediction:

8. To access the application metrics, go to ***/metrics*** endpoint.
   For example: https://upgraded-telegram-5prp97rwpj9cv4v9-8001.app.github.dev**/metrics**

```
# HELP titanic_accuracy_score Accuracy score for few random 100 test samples
# TYPE titanic_accuracy_score gauge
titanic_accuracy_score 0.829
# HELP titanic_f1_score F1 score for few random 100 test samples
# TYPE titanic_f1_score gauge
titanic_f1_score 0.796
# HELP titanic_precision_score Precision score for few random 100 test samples
# TYPE titanic_precision_score gauge
titanic_precision_score 0.854
# HELP titanic_recall_score Recall score for few random 100 test samples
# TYPE titanic_recall_score gauge
titanic_recall_score 0.745
```

V. Access the applications' metrics in Prometheus

1. On Codespace, update the *prometheus.yml* files to include both the applications as targets.

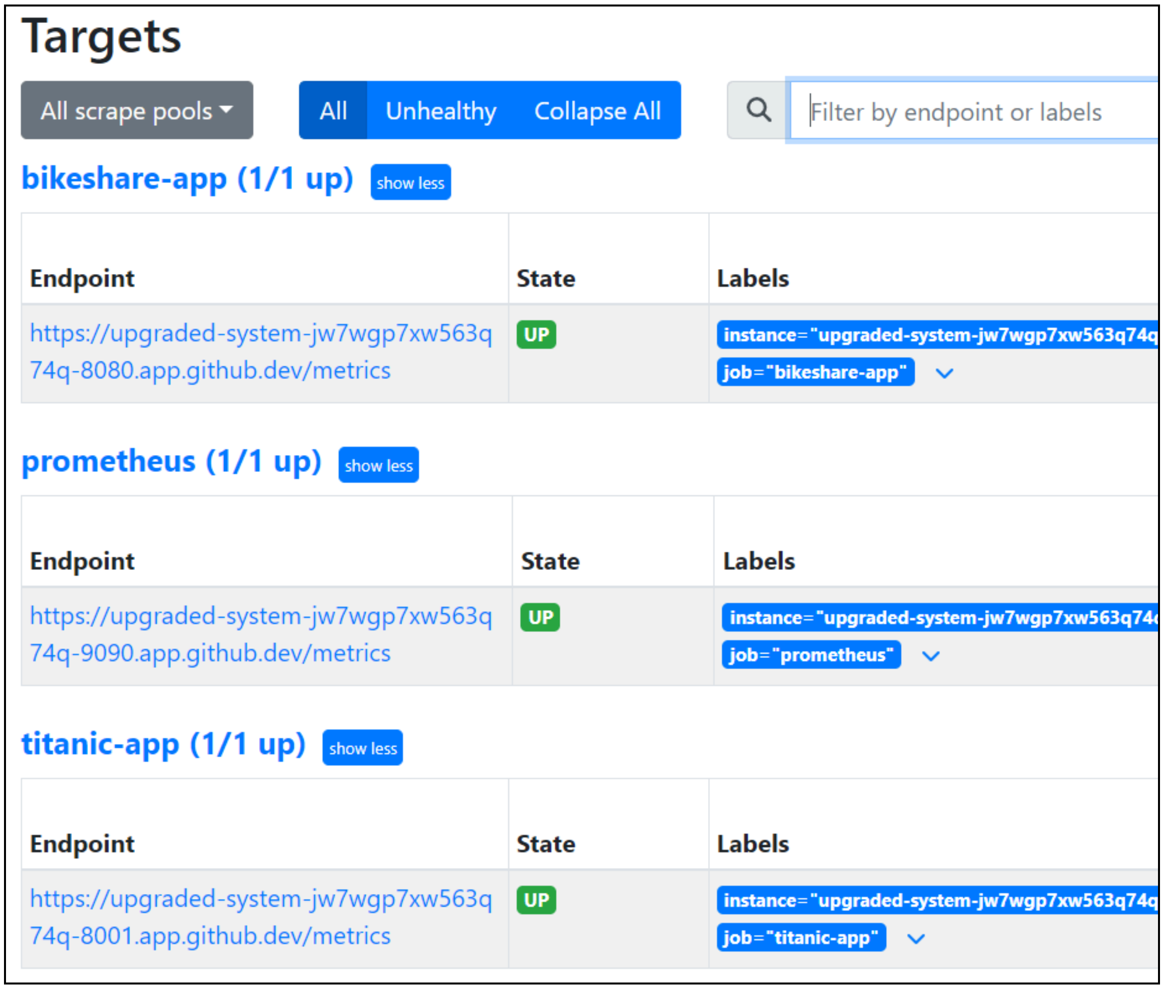
```
# my global config
global:
  scrape_interval: 20s
  evaluation_interval: 20s
```

```
# A scrape configuration containing exactly one endpoint to scrape:
scrape_configs:

  - job_name: "prometheus"
    metrics_path: '/metrics'
    scheme: 'https'
    scrape_interval: 10s       # override default value
    scrape_timeout: 10s       # override default value
    static_configs:
      - targets: ["<your-forwarded-address>"]     # Prometheus as target

  - job_name: "bikeshare-app"
    metrics_path: '/metrics'
    scheme: 'https'
    scrape_interval: 10s
    scrape_timeout: 10s
    static_configs:
      - targets: ["<your-forwarded-address>"]

  - job_name: "titanic-app"
    metrics_path: '/metrics'
    scheme: 'https'
    scrape_interval: 10s
    scrape_timeout: 10s
    static_configs:
      - targets: ["<your-forwarded-address>"]
```

2. Once the prometheus configuration file is updated, restart the prometheus container

   ```
   docker restart prom_cont
   ```

3. Check the Targets at Prometheus, the States should be up.


Targets

All scrape pools | All | Unhealthy | Collapse All | Filter by endpoint or labels

bikeshare-app (1/1 up) show less

Endpoint	State	Labels
https://upgraded-system-jw7wgp7xw563q74q-8080.app.github.dev/metrics	UP	instance="upgraded-system-jw7wgp7xw563q74q job="bikeshare-app"

prometheus (1/1 up) show less

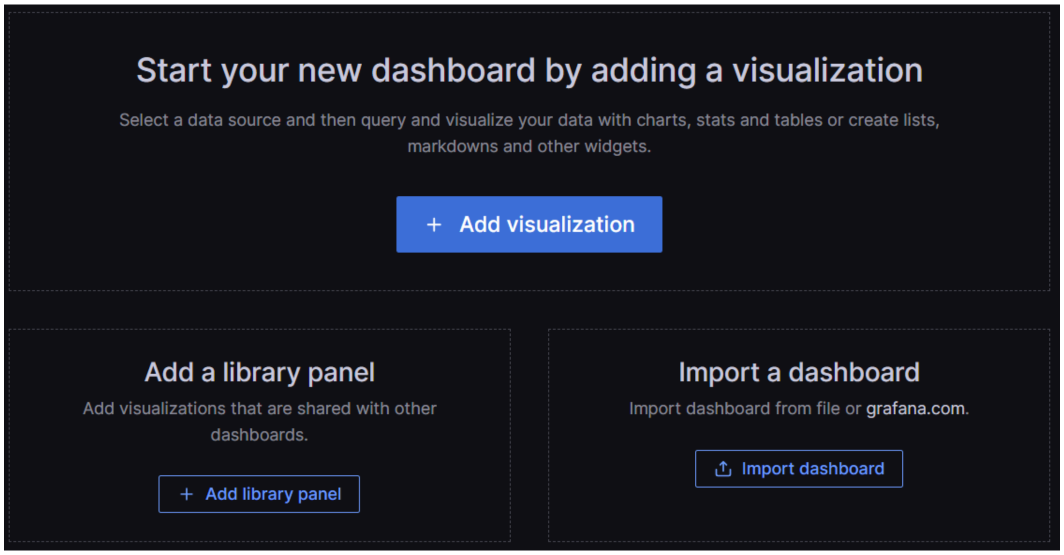
Endpoint	State	Labels
https://upgraded-system-jw7wgp7xw563q74q-9090.app.github.dev/metrics	UP	instance="upgraded-system-jw7wgp7xw563q74 job="prometheus"

titanic-app (1/1 up) show less

Endpoint	State	Labels
https://upgraded-system-jw7wgp7xw563q74q-8001.app.github.dev/metrics	UP	instance="upgraded-system-jw7wgp7xw563q74q job="titanic-app"

VI. Create Real-time Dashboard in Grafana

1. On Grafana, go to ***Home > Dashboards > Create Dashboard > Add visualization***


Start your new dashboard by adding a visualization

Select a data source and then query and visualize your data with charts, stats and tables or create lists, markdowns and other widgets.

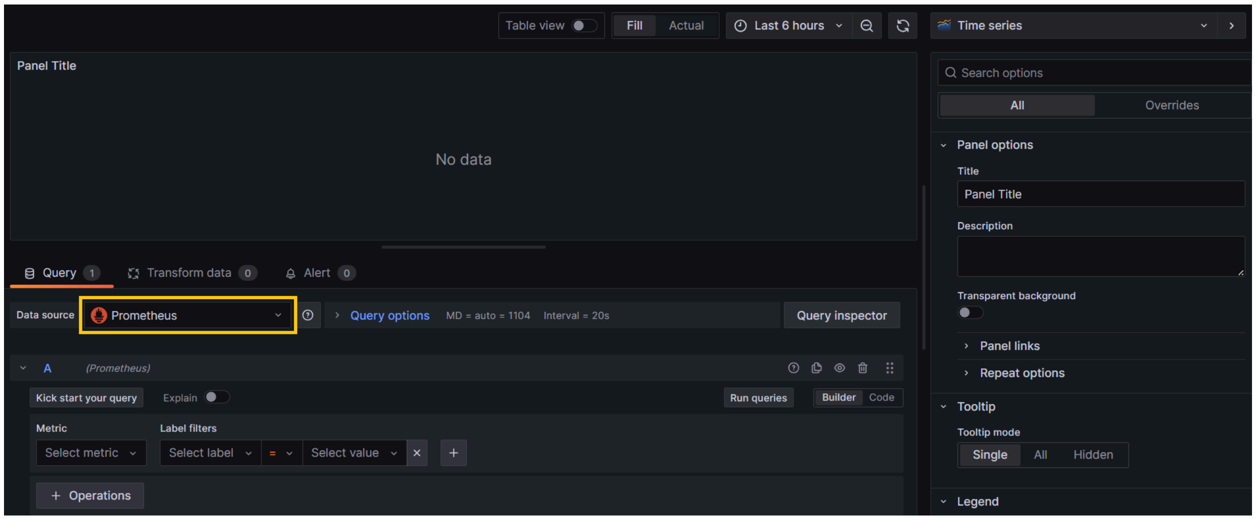
Add a library panel

Add visualizations that are shared with other dashboards.

Import a dashboard

Import dashboard from file or grafana.com.

2. Select Prometheus as Data source



Data source: Prometheus

MD = auto = 1104 Interval = 20s

A (Prometheus)

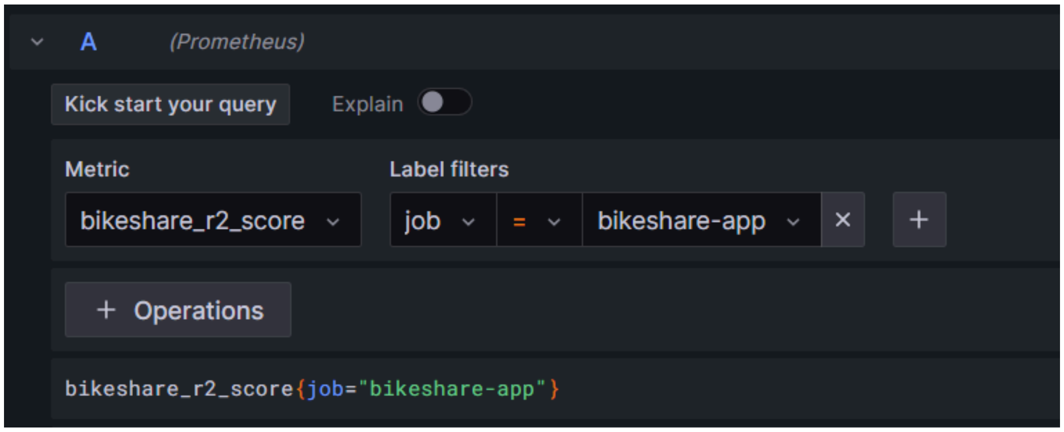
Title: Panel Title

Description

Transparent background

Tooltip mode: Single

3. Select the metric and Run query

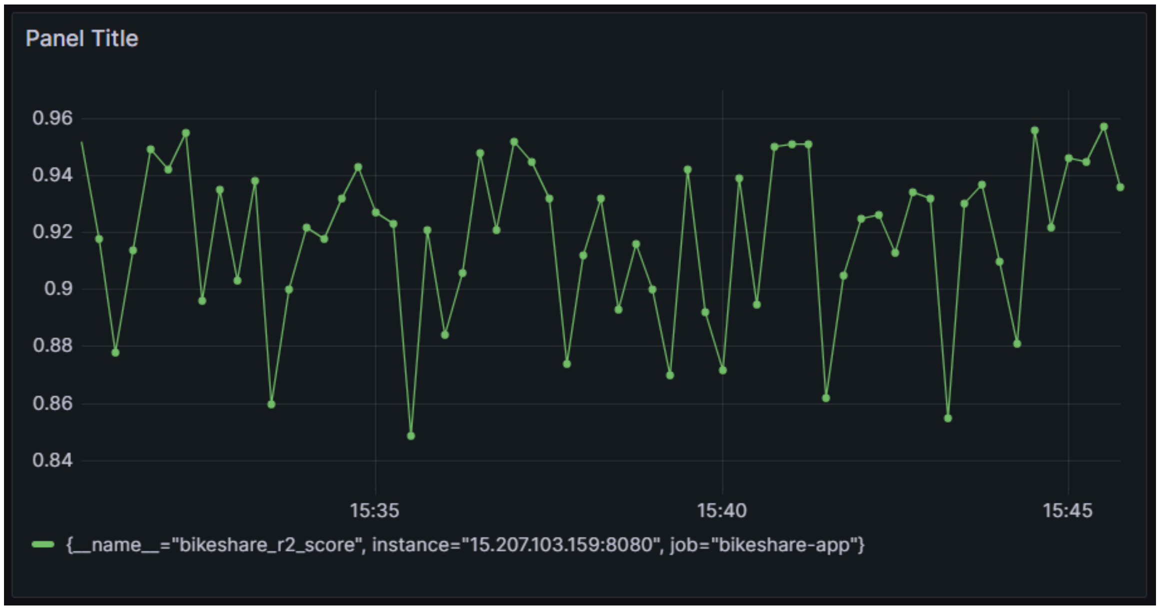


Metric: bikeshare_r2_score

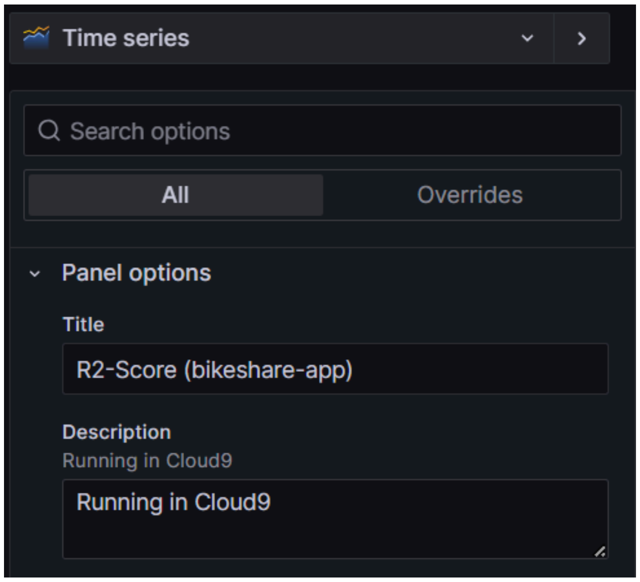
Label filters: job = bikeshare-app

```
bikeshare_r2_score{job="bikeshare-app"}
```

4. Visualization will appear



5. From the right side tool options, settings can be changed.
   Select Time series visualization, give Title and Description



Title

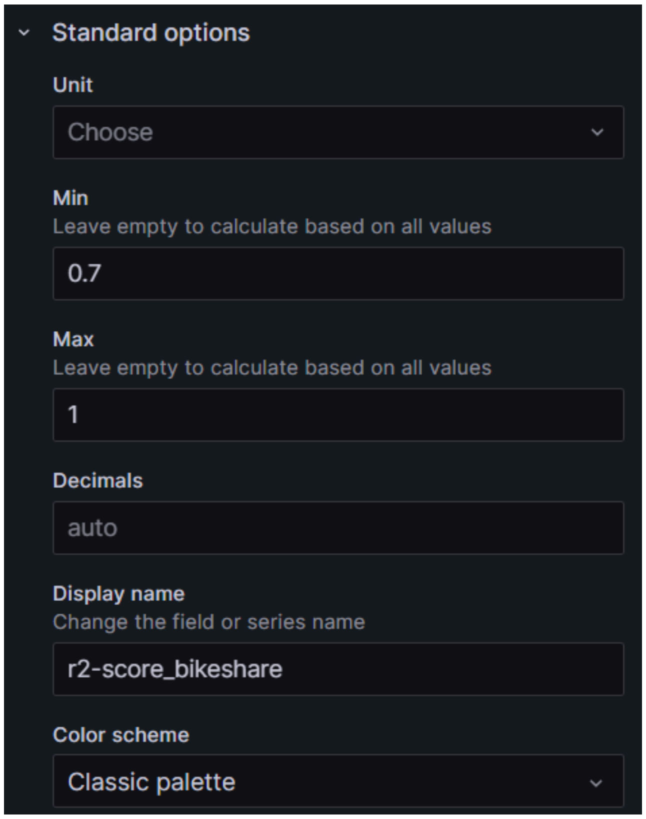
R2-Score (bikeshare-app)

Description

Running in Cloud9

Running in Cloud9

Give Min & Max range, Display name, change Color scheme



Standard options

Unit: Choose

Min
Leave empty to calculate based on all values
0.7

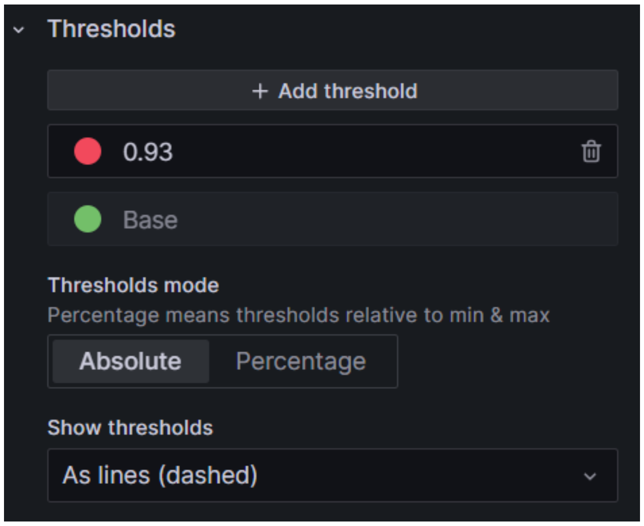
Max
Leave empty to calculate based on all values
1

Decimals: auto

Display name
Change the field or series name
r2-score_bikeshare

Color scheme: Classic palette

Give Threshold, make it visible as dashed line



Thresholds

+ Add threshold

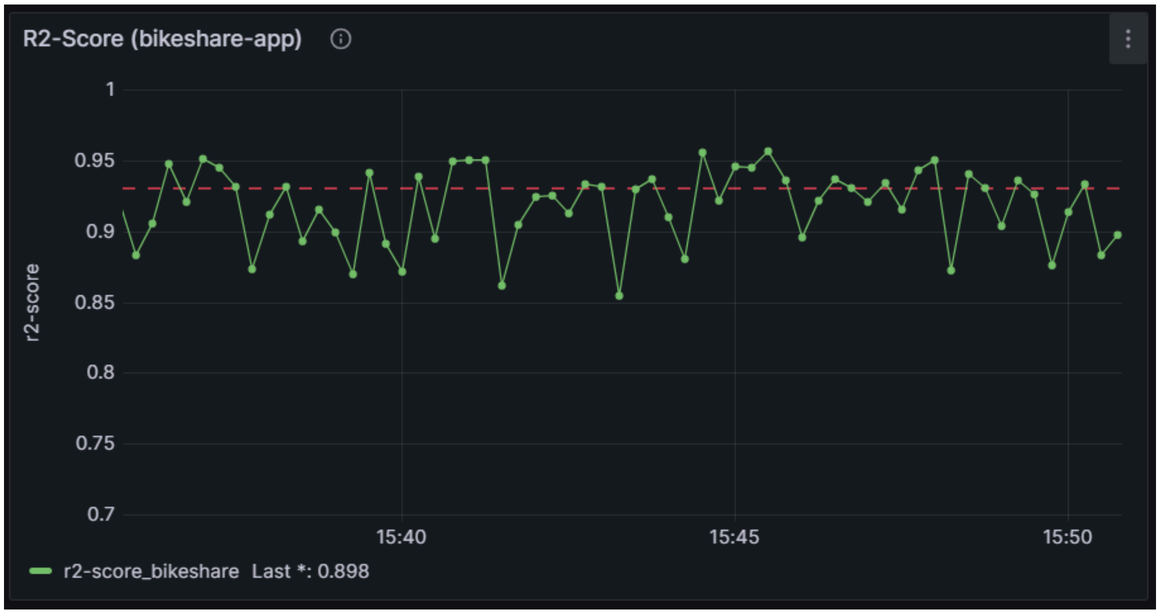
● 0.93

● Base

Thresholds mode
Percentage means thresholds relative to min & max
[✓ Absolute] Percentage

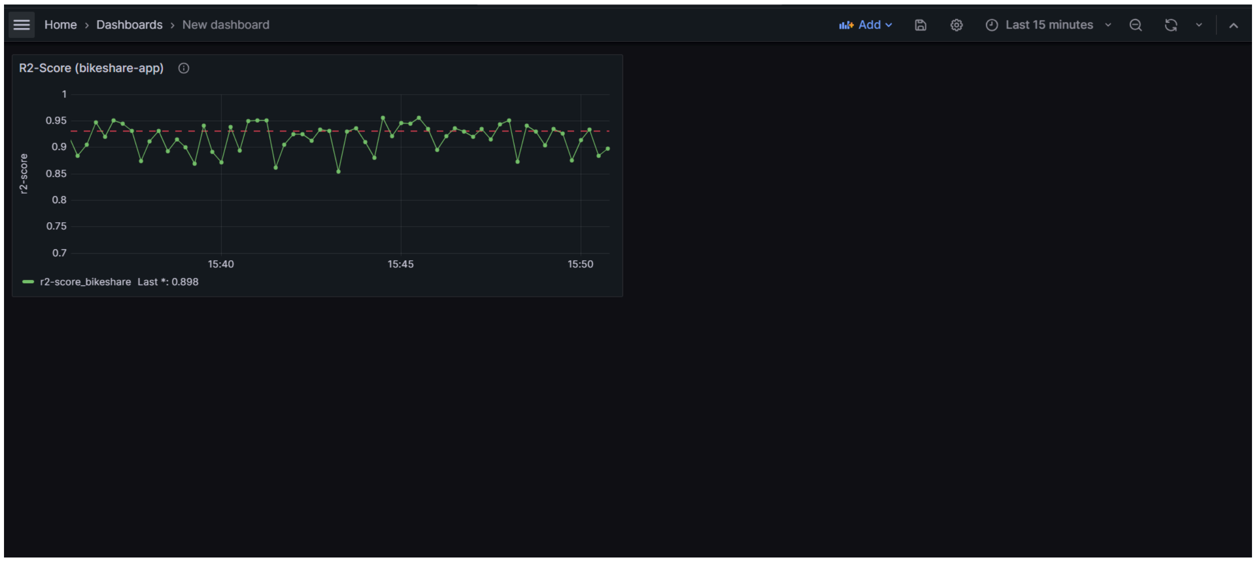
Show thresholds: As lines (dashed)

Plot will be updated

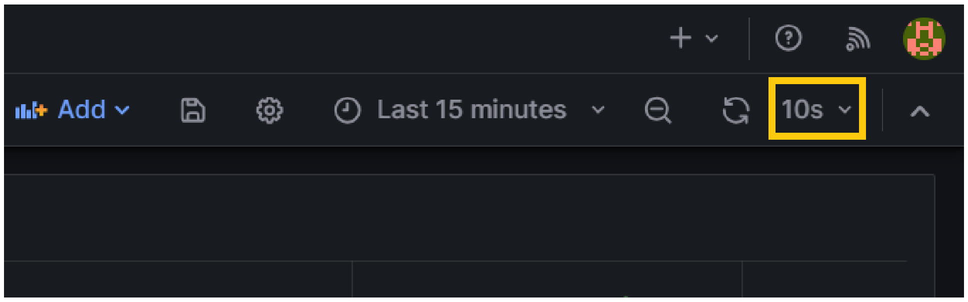


Click on ***Apply*** to apply the change to the dashboard

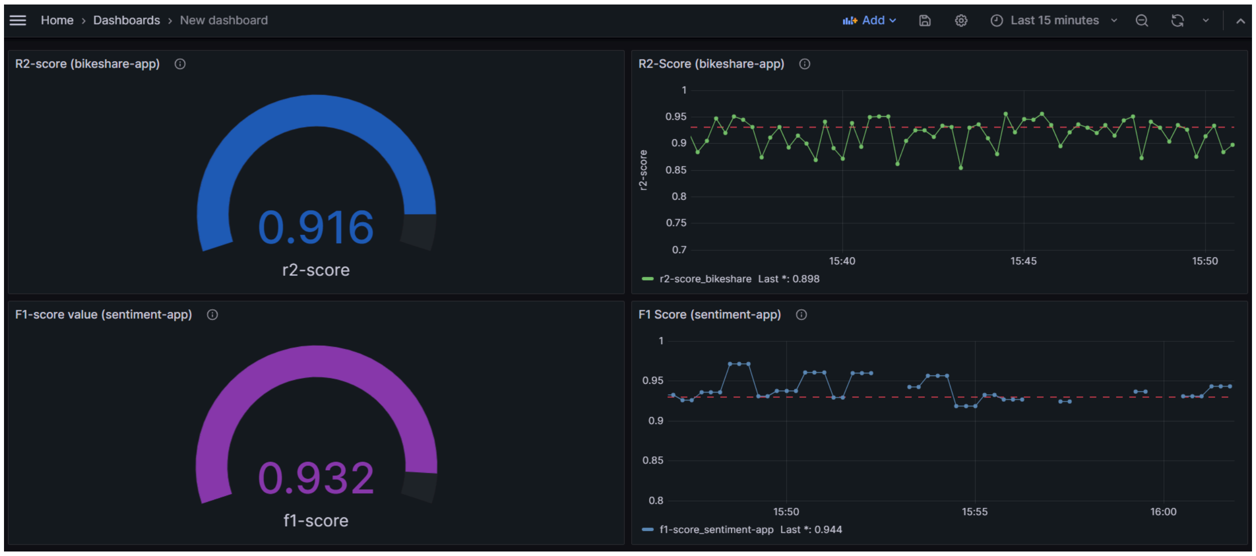
6. Changes will be updated on the Dashboard. Click on the Save icon to save the Dashboard.


Home › Dashboards › New dashboard
Add
Last 15 minutes
R2-Score (bikeshare-app)
r2-score
15:40
15:45
15:50
r2-score_bikeshare Last *: 0.898

7. From the right side corner, make the dashboard update every 10 seconds.

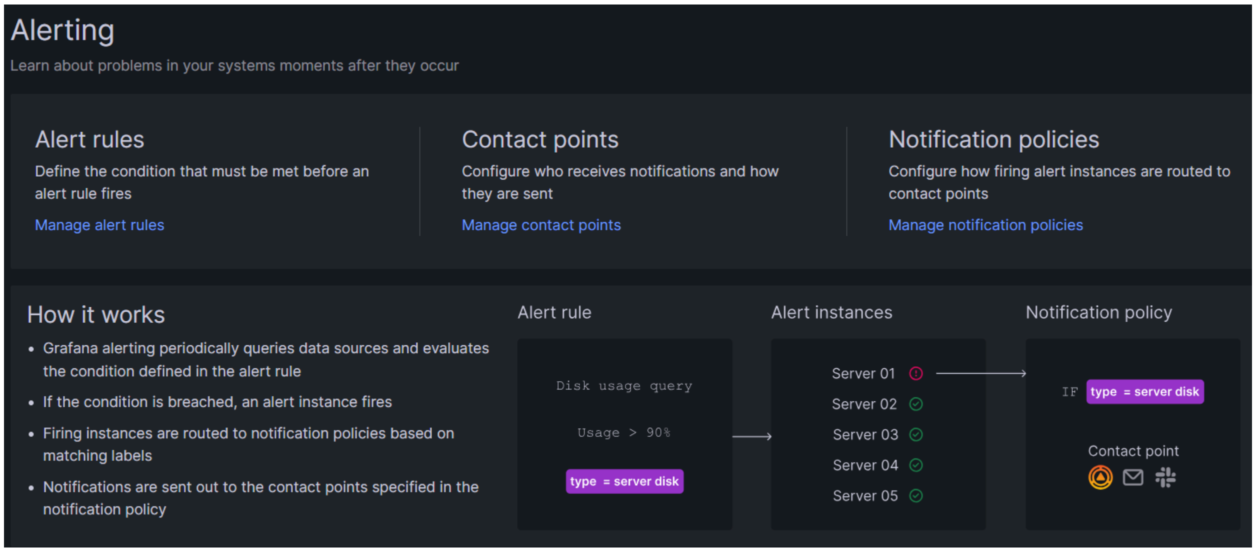


8. Using the ***Add***, more panels can be added


Home › Dashboards › New dashboard
Add
Last 15 minutes
R2-score (bikeshare-app)
0.916
r2-score
R2-Score (bikeshare-app)
r2-score_bikeshare Last *: 0.898
F1-score value (sentiment-app)
0.932
f1-score
F1 Score (sentiment-app)
f1-score_sentiment-app Last *: 0.944

VII. Add Alerts in Grafana

1. To add alerts in Grafana, go to ***Home > Alerting***


Alerting

Learn about problems in your systems moments after they occur

Alert rules

Define the condition that must be met before an alert rule fires

Manage alert rules

Contact points

Configure who receives notifications and how they are sent

Manage contact points

Notification policies

Configure how firing alert instances are routed to contact points

Manage notification policies

How it works

- Grafana alerting periodically queries data sources and evaluates the condition defined in the alert rule
- If the condition is breached, an alert instance fires
- Firing instances are routed to notification policies based on matching labels
- Notifications are sent out to the contact points specified in the notification policy

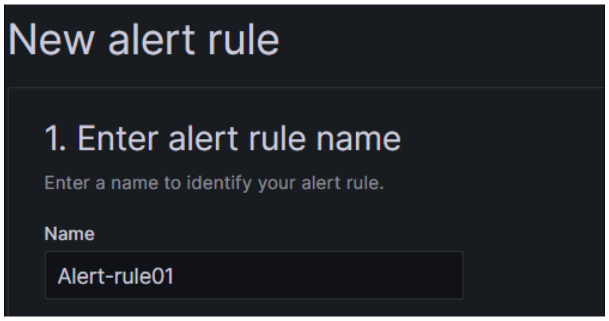
Alert rules: Define the condition that must be met before an alert rule fires

Contact points: Configure who receives notifications and how they are sent

Notification policies: Configure how firing alert instances are routed to contact points

2. To add an alert rule, go to ***Home > Alerting > Alert rules > New alert rule***

Give a name to your rule

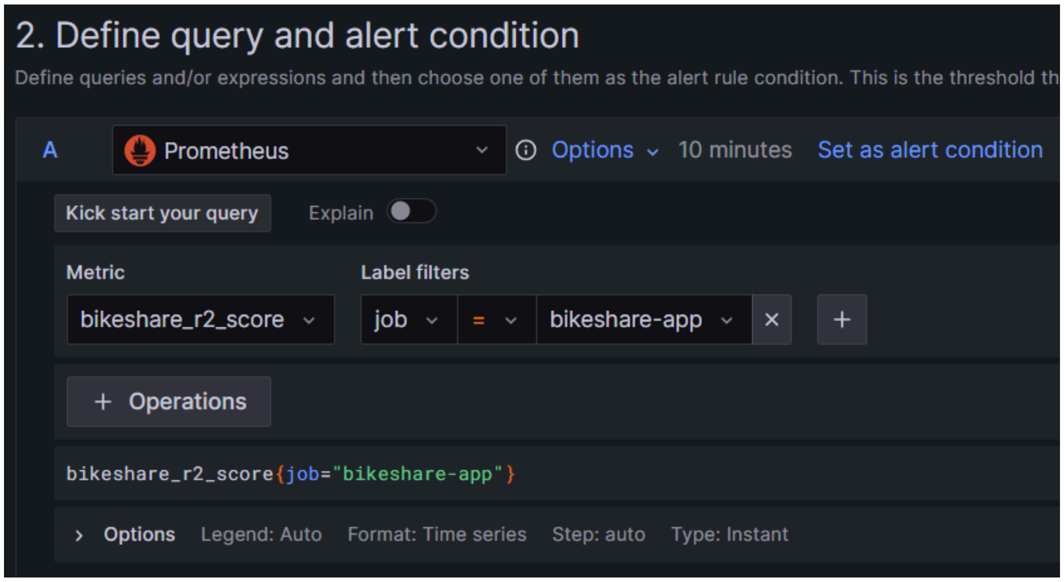

New alert rule

1. Enter alert rule name

Enter a name to identify your alert rule.

Name: Alert-rule01

In **A** block, select the Metric


2. Define query and alert condition

Define queries and/or expressions and then choose one of them as the alert rule condition. This is the threshold th

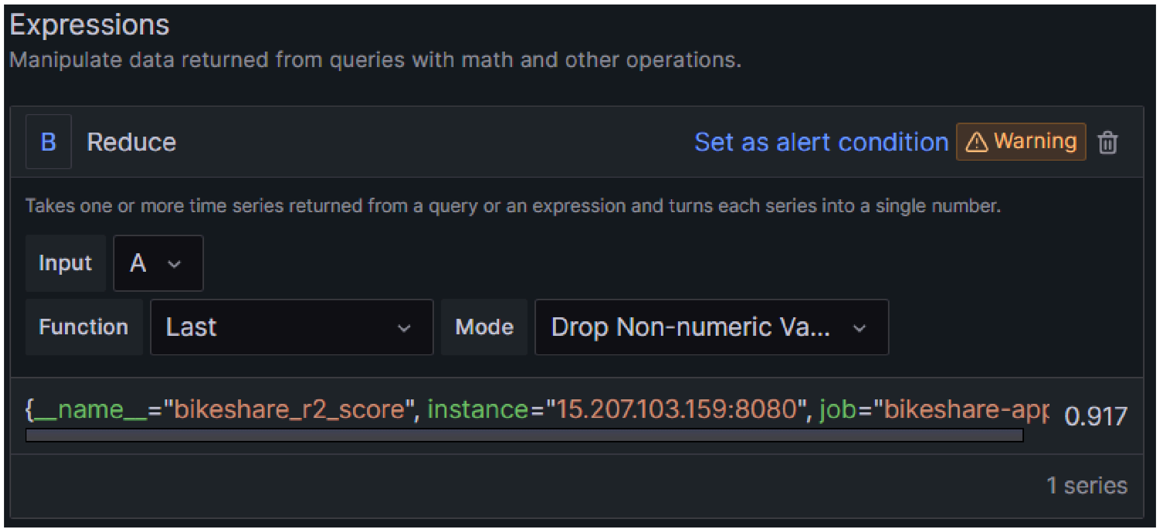
Metric: bikeshare_r2_score

Label filters: job = bikeshare-app

```
bikeshare_r2_score{job="bikeshare-app"}
```

Options Legend: Auto Format: Time series Step: auto Type: Instant

In **B**(Reduce) block, set Function as *Last* and Mode as *Drop Non-numeric Values*


Expressions

Manipulate data returned from queries with math and other operations.

Takes one or more time series returned from a query or an expression and turns each series into a single number.

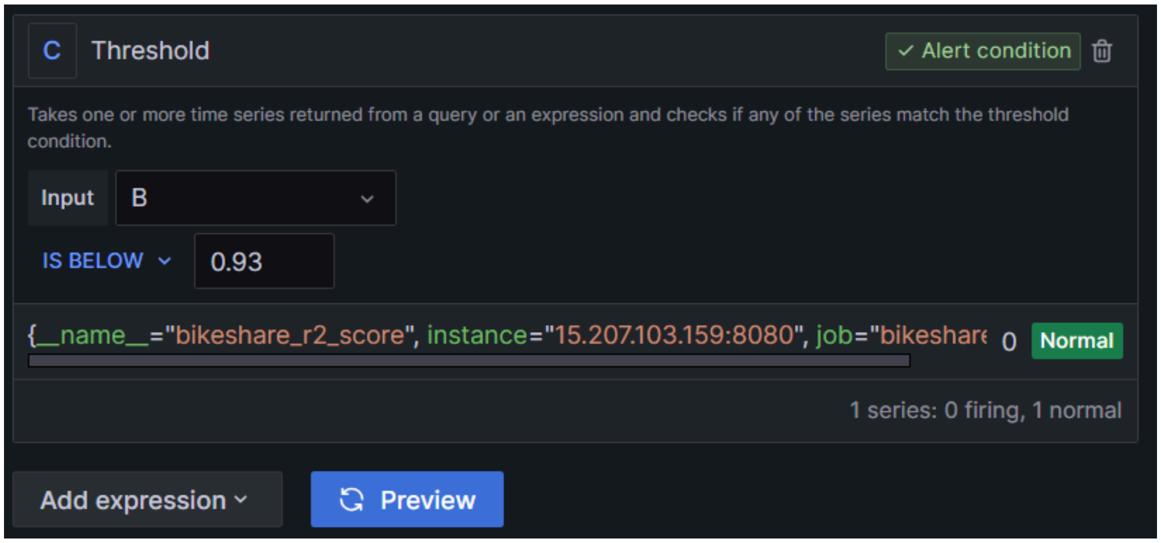
Input: A

Function: Last

Mode: Drop Non-numeric Va...

```
{__name__="bikeshare_r2_score", instance="15.207.103.159:8080", job="bikeshare-app  0.917
1 series
```

In the **C**(Threshold) block, set the condition to ***IS BELOW*** and give a threshold value ***0.93*** then click Preview to check the A B C blocks pipeline.


C Threshold

Takes one or more time series returned from a query or an expression and checks if any of the series match the threshold condition.

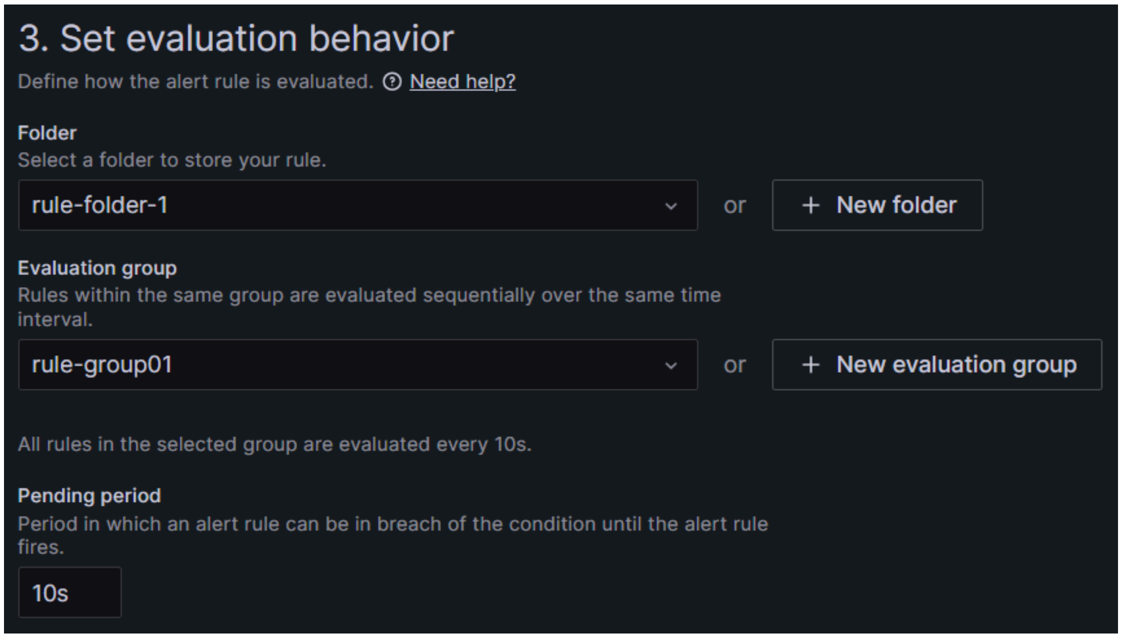
Input: B

IS BELOW 0.93

```
{__name__="bikeshare_r2_score", instance="15.207.103.159:8080", job="bikeshare  0  Normal
```

1 series: 0 firing, 1 normal

For Set evaluation behavior, create New Folder, create New evaluation group(give name and 10s), set the pending period as 10s.


3. Set evaluation behavior

Define how the alert rule is evaluated. Need help?

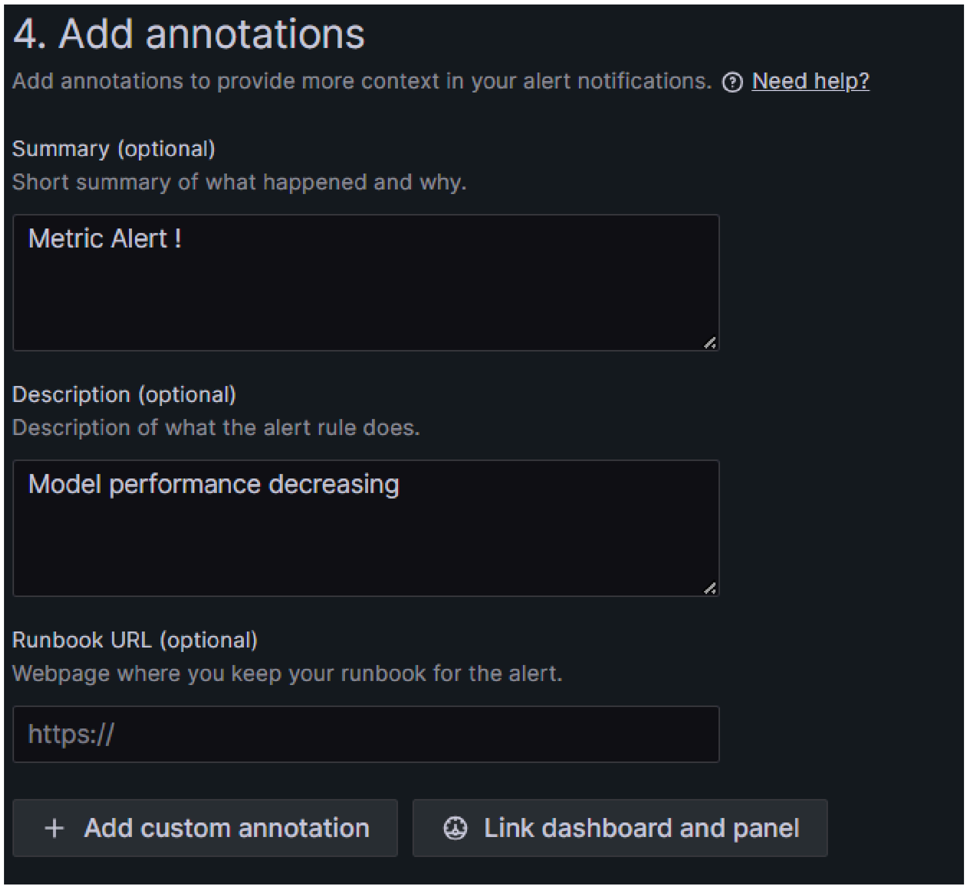
Folder
Select a folder to store your rule.
rule-folder-1

Evaluation group
Rules within the same group are evaluated sequentially over the same time interval.
rule-group01

All rules in the selected group are evaluated every 10s.

Pending period
Period in which an alert rule can be in breach of the condition until the alert rule fires.
10s

For Add annotations, give Summary and Description, then click on ***Link dashboard and panel***


4. Add annotations

Add annotations to provide more context in your alert notifications. Need help?

Summary (optional)
Short summary of what happened and why.

Metric Alert !

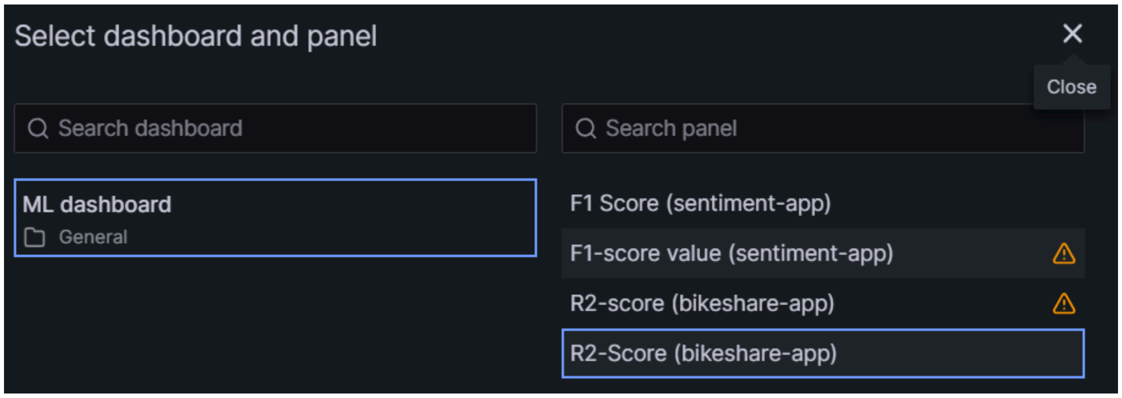
Description (optional)
Description of what the alert rule does.

Model performance decreasing

Runbook URL (optional)
Webpage where you keep your runbook for the alert.

https://

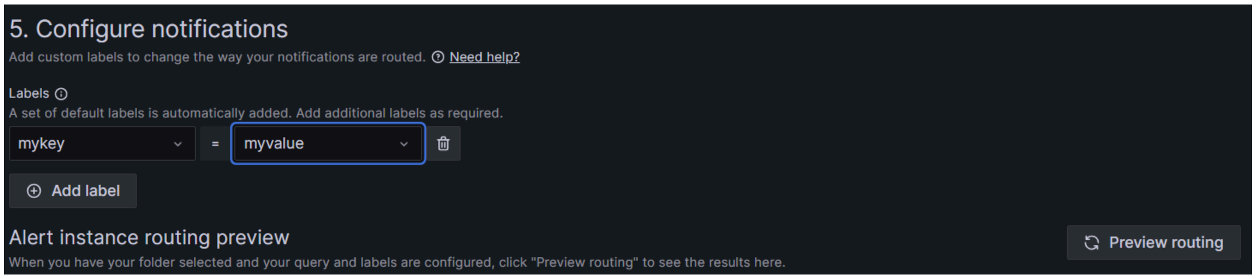
Select your Dashboard and panel, then click Confirm


Select dashboard and panel

ML dashboard
General

F1 Score (sentiment-app)
F1-score value (sentiment-app)
R2-score (bikeshare-app)
R2-Score (bikeshare-app)

For Configure notifications, give a key and value and enter, then select Preview Routing.


5. Configure notifications

Add custom labels to change the way your notifications are routed. Need help?

Labels

A set of default labels is automatically added. Add additional labels as required.

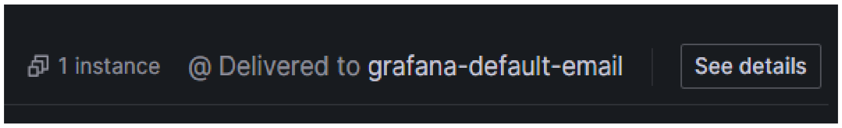
mykey = myvalue

Add label

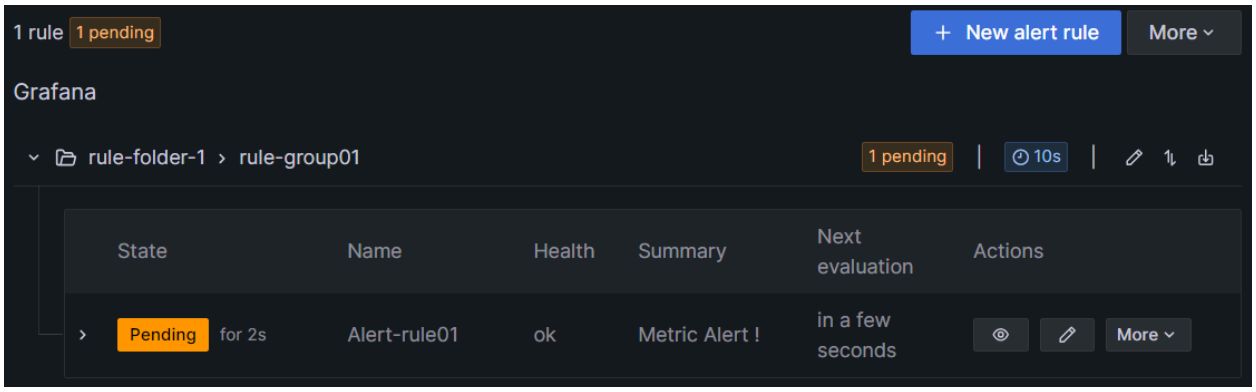
Alert instance routing preview

When you have your folder selected and your query and labels are configured, click "Preview routing" to see the results here.

It will notify the Default policy


1 instance @ Delivered to grafana-default-email

From the top-right corner, select ***Save rule and exit.*** Rule will be added.


1 rule 1 pending

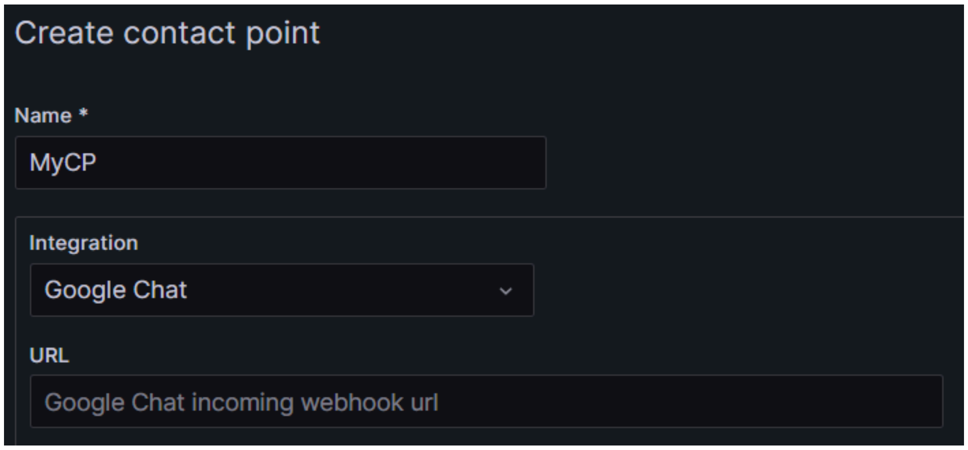
Grafana

rule-folder-1 › rule-group01 1 pending 10s

State	Name	Health	Summary	Next evaluation	Actions
Pending for 2s	Alert-rule01	ok	Metric Alert !	in a few seconds	More

3. To add a Contact point, go to ***Home > Alerting > Contact points > Add contact point***

   Give a name, select the Integration as ***Email*** and give the receiver's mail id.


Create contact point

Name *

MyCP

Integration

Google Chat

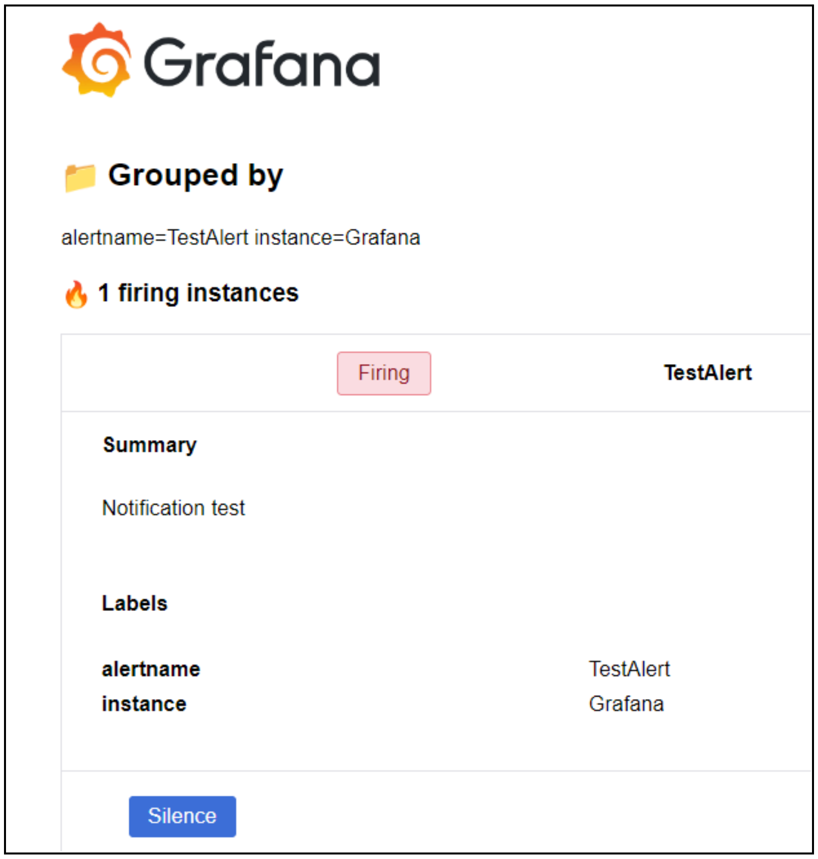
URL

Google Chat incoming webhook url

The sender's details are already known to Grafana via environment variables inside the "env.list".

Now, click on ***Test > Send test notification***.

You will receive a Test alert message in your mail.



Grafana

Grouped by

alertname=TestAlert instance=Grafana

1 firing instances

Firing	TestAlert

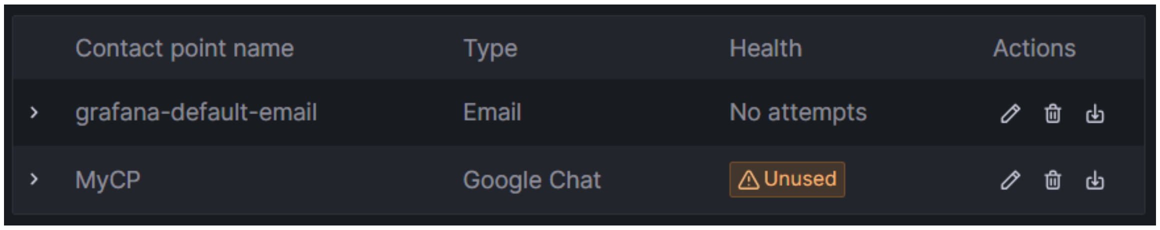
Summary

Notification test

Labels

alertname	TestAlert
instance	Grafana

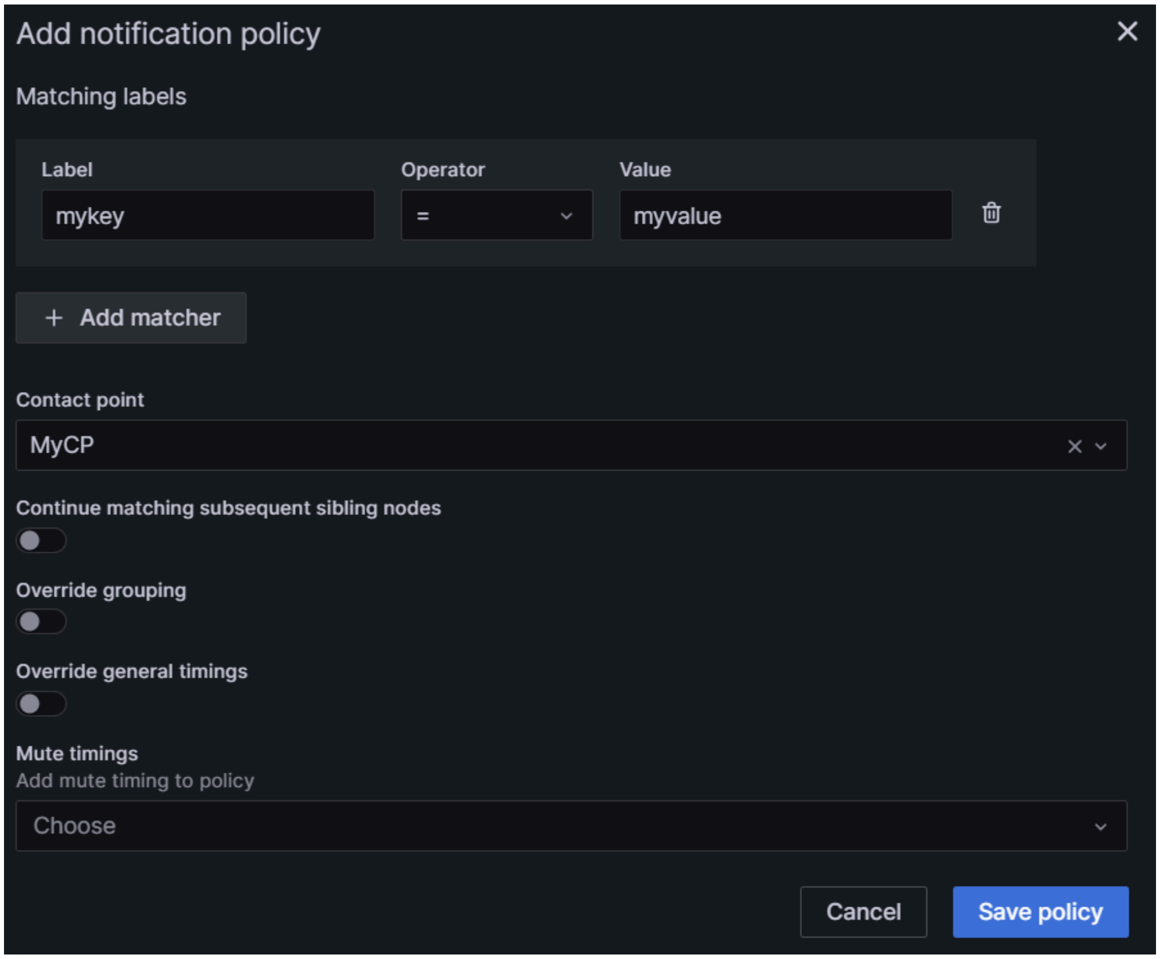
Once tested, click on ***Save contact point***. The contact point should be saved.



Contact point name	Type	Health	Actions
grafana-default-email	Email	No attempts	
MyCP	Google Chat	Unused	

4. Now go to ***Home > Alerting > Notification policies > New nested policy***

   Select the Key Label and Value, the ones you gave while creating an alert rule, then click on ***Save policy***.


Add notification policy

Matching labels

Label	Operator	Value
mykey	=	myvalue

Contact point: MyCP

Continue matching subsequent sibling nodes

Override grouping

Override general timings

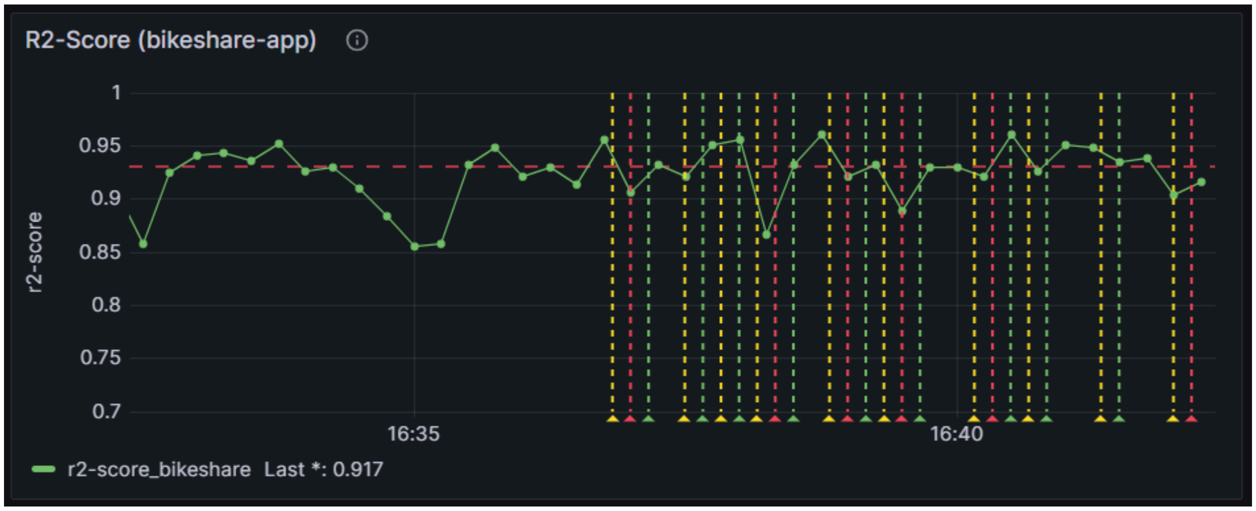
Mute timings
Add mute timing to policy
Choose

5. Now, go to your Dashboard. Alert vertical lines will be visible on your selected panel.

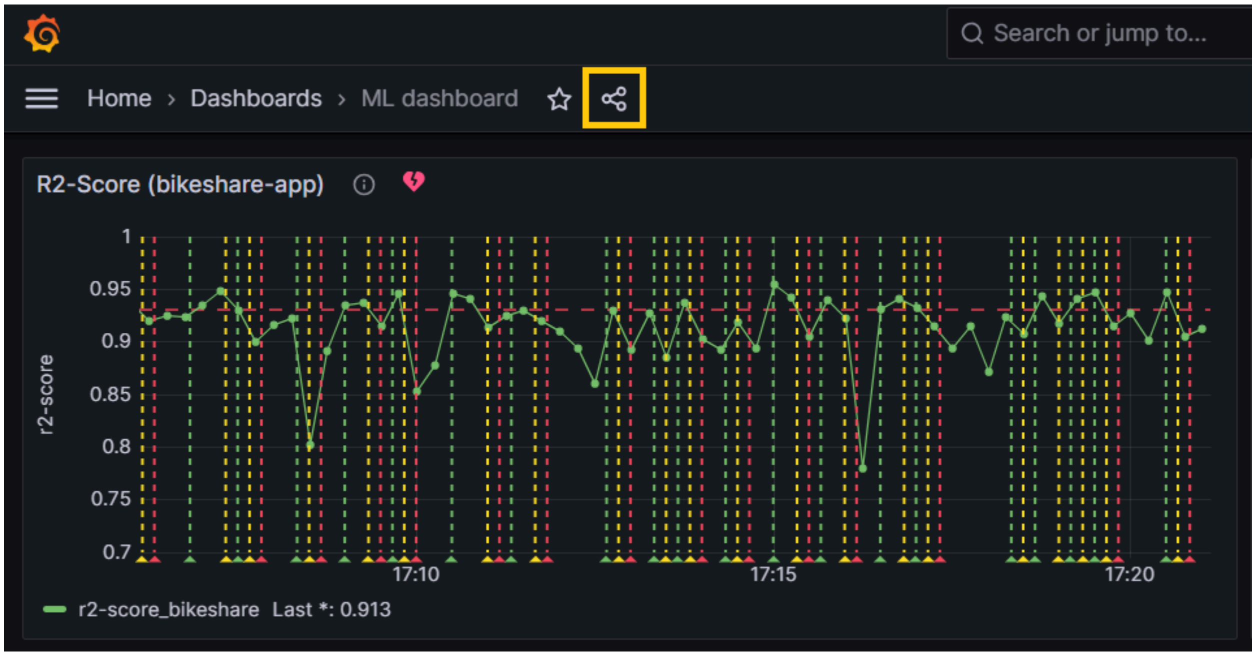
   Here,

   - yellow lines refers to the *pending* state
   - red lines refers to the *firing* state
   - green lines refers to the *normal* state

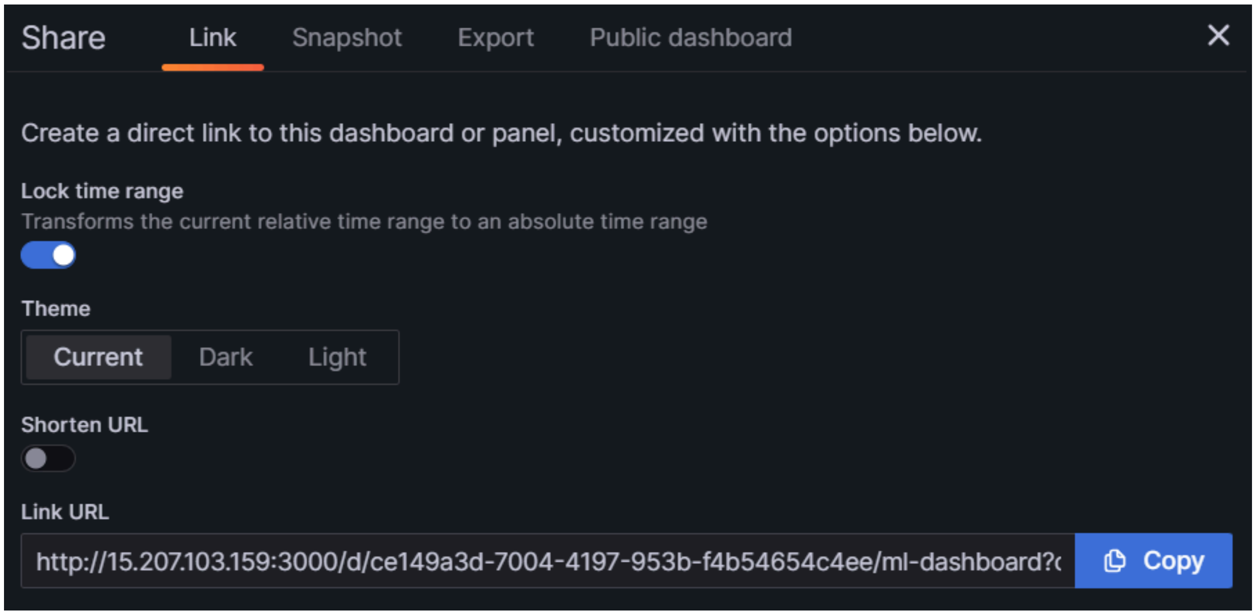
   Whenever the alert rule is in a breach(value < threshold), the yellow line will be indicated. After the pending period of the alert rule(here 10s), the alert will start firing and the red line will be indicated (You will also receive the notification when the alert fires). Next time when the alert rule is out of breach, the green line will be indicated.



6. You can also share your Dashboard using the Share icon



Copy the link and share


Share
Link
Snapshot
Export
Public dashboard

Create a direct link to this dashboard or panel, customized with the options below.

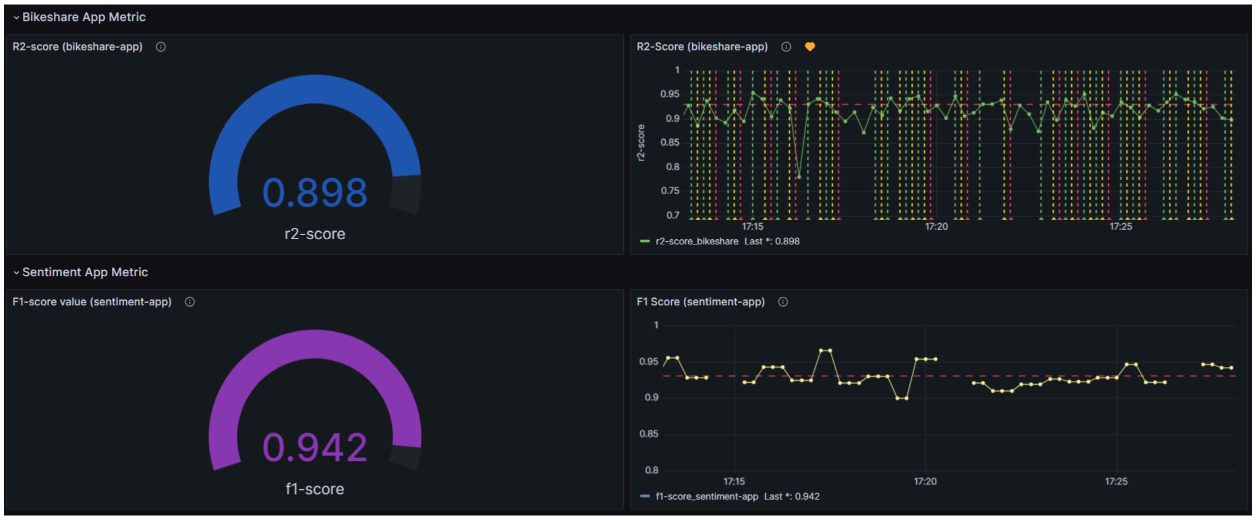
Lock time range
Transforms the current relative time range to an absolute time range

Theme
Current Dark Light

Shorten URL

Link URL
http://15.207.103.159:3000/d/ce149a3d-7004-4197-953b-f4b54654c4ee/ml-dashboard?c

Once shared, they will need to login with the same credentials you used to access Grafana and Dashboard will open for them.



Cleanup

When you are finished using Codespace, delete all the running containers and delete the codespace.