Lesson 07 Demo 04

Applying Health Checks and Monitoring to Microservices

Objective: To apply health checks and monitoring to microservices using Docker, Docker Compose, and a Prometheus image for enhanced reliability and observability

Tools required: Docker, Docker Compose, Python 3.x, and Flask

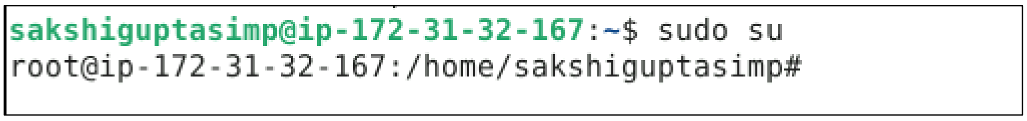
Prerequisites: None

Steps to be followed:

1. Create microservices
2. Create a requirements file for dependencies
3. Create a Dockerfile for each microservice
4. Create a Docker compose file and run the setup

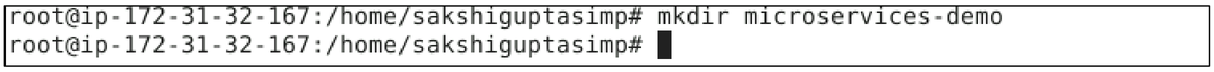
Step 1: Create microservices

1.1 Switch to the root user using the following command:
sudo su



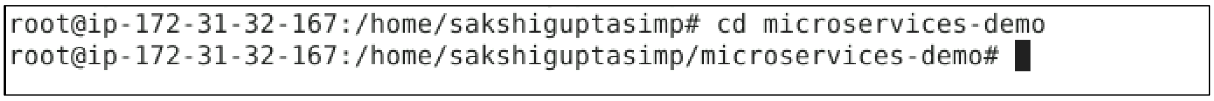
```
sakshiguptasimp@ip-172-31-32-167:~$ sudo su
root@ip-172-31-32-167:/home/sakshiguptasimp#
```

1.2 Create a directory for the microservices using the following command:
mkdir microservices-demo



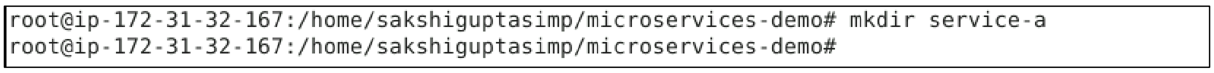
```
root@ip-172-31-32-167:/home/sakshiguptasimp# mkdir microservices-demo
root@ip-172-31-32-167:/home/sakshiguptasimp#
```

1.3 Navigate inside the created directory using the following command:
cd microservices-demo

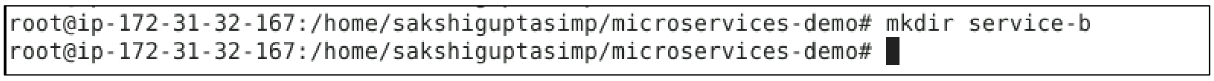


```
root@ip-172-31-32-167:/home/sakshiguptasimp# cd microservices-demo
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-demo#
```

1.4 Create two directories, **service-a** and **service-b**, for the respective microservices using the following commands:
mkdir service-a
mkdir service-b

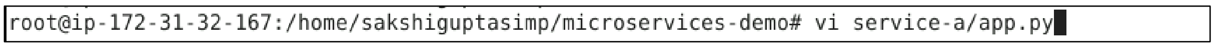


```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-demo# mkdir service-a
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-demo#
```



```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-demo# mkdir service-b
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-demo#
```

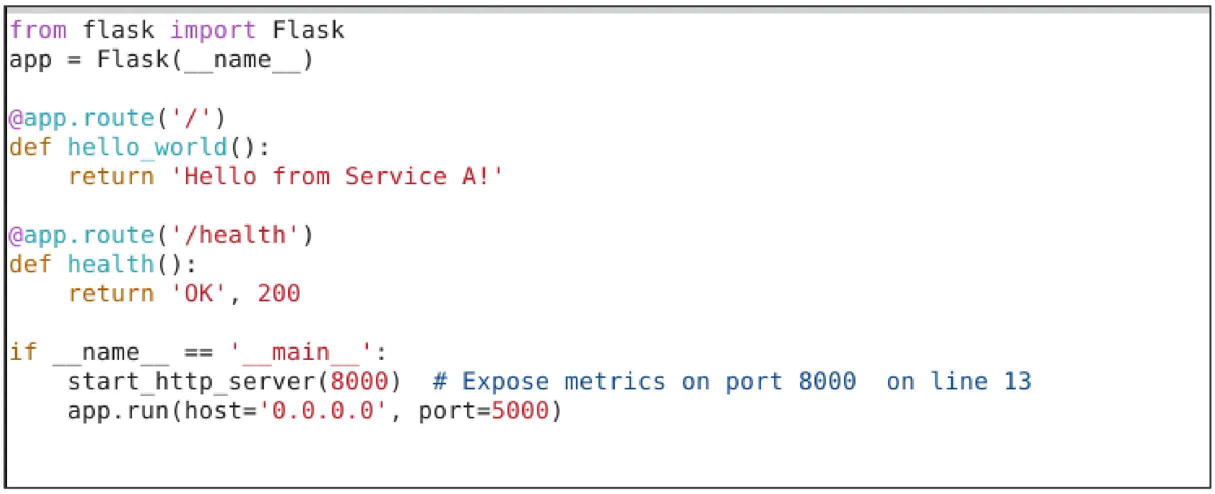
1.5 Create a Python file in the **service-a** directory using the following command:
vi service-a/app.py



```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-demo# vi service-a/app.py
```

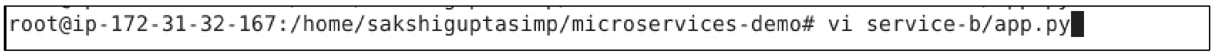
1.6 Set up a simple Flask web application for **service-a** using the following code:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
  return 'Hello from Service A!'

@app.route('/health')
def health():
  return 'OK', 200

if __name__ == '__main__':
        start_http_server(8000)  # Expose metrics on port 8000  on line 13
        app.run(host='0.0.0.0', port=5000)
```



```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello from Service A!'

@app.route('/health')
def health():
    return 'OK', 200

if __name__ == '__main__':
    start_http_server(8000)  # Expose metrics on port 8000  on line 13
    app.run(host='0.0.0.0', port=5000)
```

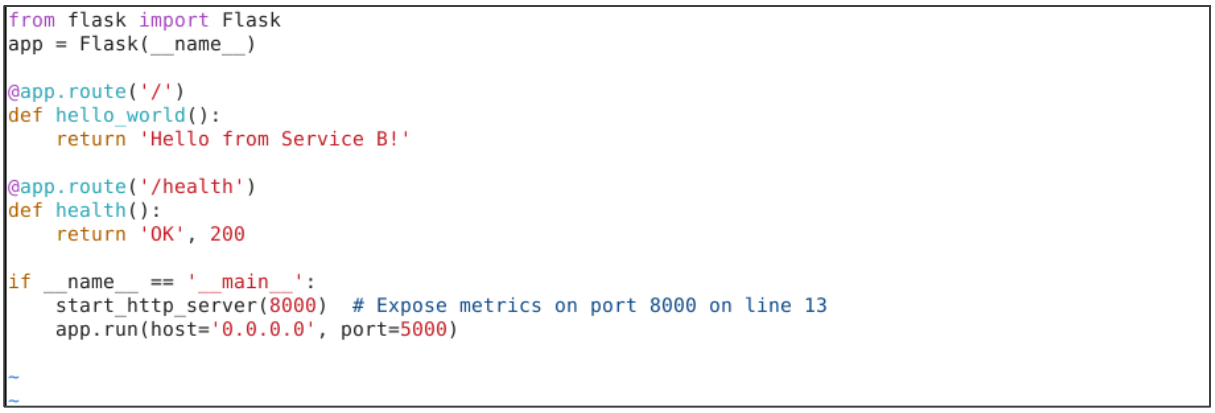
1.7 Create a Python file in the **service-b** directory using the following command:
vi service-b/app.py



```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-demo# vi service-b/app.py
```

1.8 Set up a simple Flask web application for **service-b** using the following code:

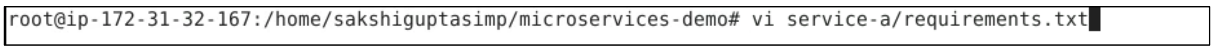
```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
   return 'Hello from Service B!'

@app.route('/health')
def health():
   return 'OK', 200

if __name__ == '__main__':
        start_http_server(8000)  # Expose metrics on port 8000  on line 13
        app.run(host='0.0.0.0', port=5000)
```



```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello from Service B!'

@app.route('/health')
def health():
    return 'OK', 200

if __name__ == '__main__':
    start_http_server(8000)  # Expose metrics on port 8000 on line 13
    app.run(host='0.0.0.0', port=5000)

~
~
```

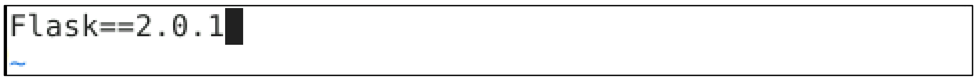
Step 2: Create a requirements file for dependencies

2.1 Create a requirements file in the **service-a** directory using the following command:
vi service-a/requirements.txt



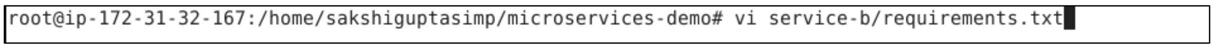
```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-demo# vi service-a/requirements.txt
```

2.2 Specify the Flask version required for **service-a** using the following code:
Flask==2.0.1



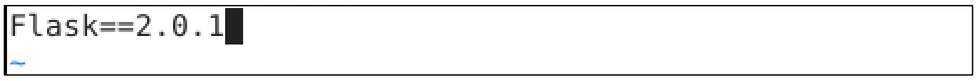
```
Flask==2.0.1
~
```

2.3 Create a requirements file in the **service-b** directory using the following command:
vi service-b/requirements.txt



```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-demo# vi service-b/requirements.txt
```

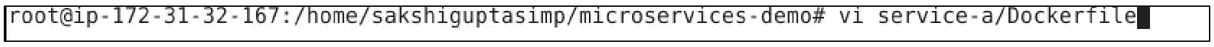
2.4 Specify the Flask version required for the **service-b** using the following code:
Flask==2.0.1



```
Flask==2.0.1
~
```

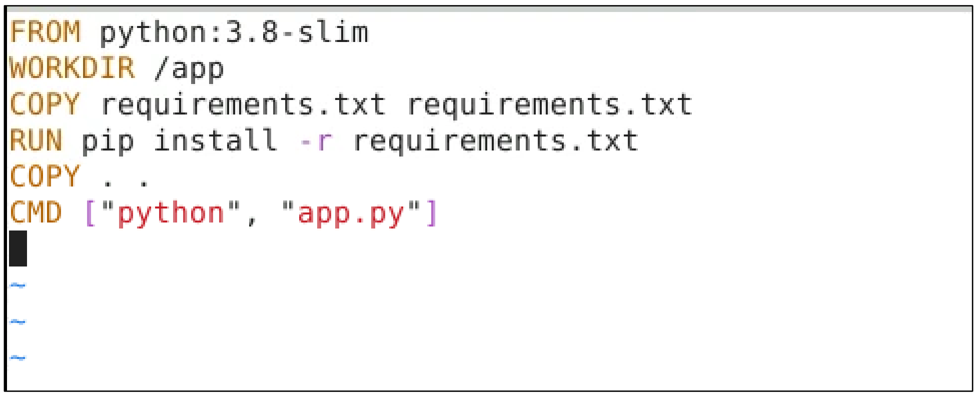
Step 3: Create a Dockerfile for each microservice

3.1 Create a Dockerfile in the **service-a** directory using the following command:
vi service-a/Dockerfile



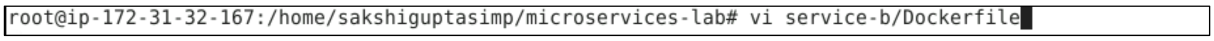
```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-demo# vi service-a/Dockerfile
```

3.2 Enter the following script in the created **Dockerfile** file:
FROM python:3.8-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]



```
FROM python:3.8-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]

~
~
~
```

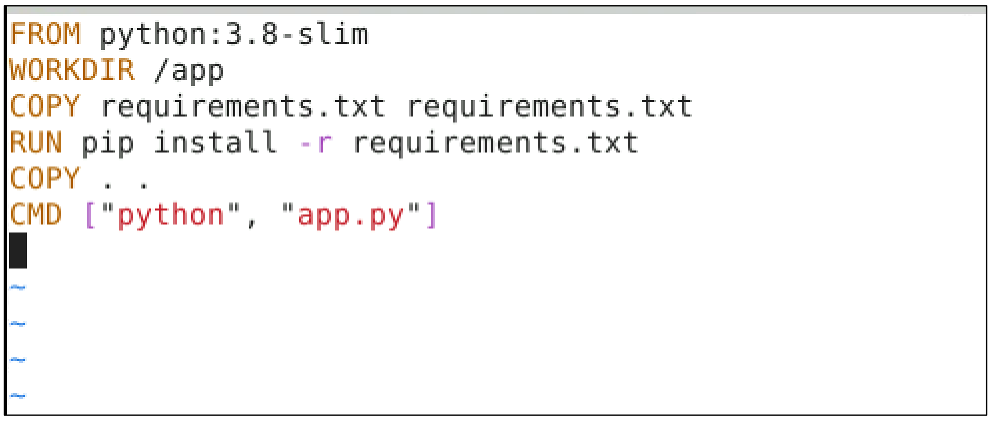
3.3 Create a Dockerfile in the **service-b** directory using the following command:
vi service-b/Dockerfile



```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-lab# vi service-b/Dockerfile
```

3.4 Enter the following code in the created **Dockerfile** file:

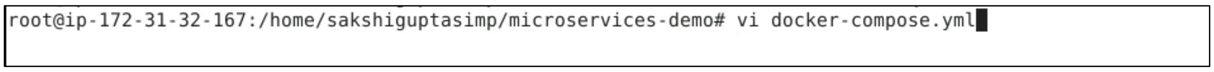
```
FROM python:3.8-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```



```
FROM python:3.8-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]

~
~
~
~
```

Step 4: Create a Docker compose file and run the setup

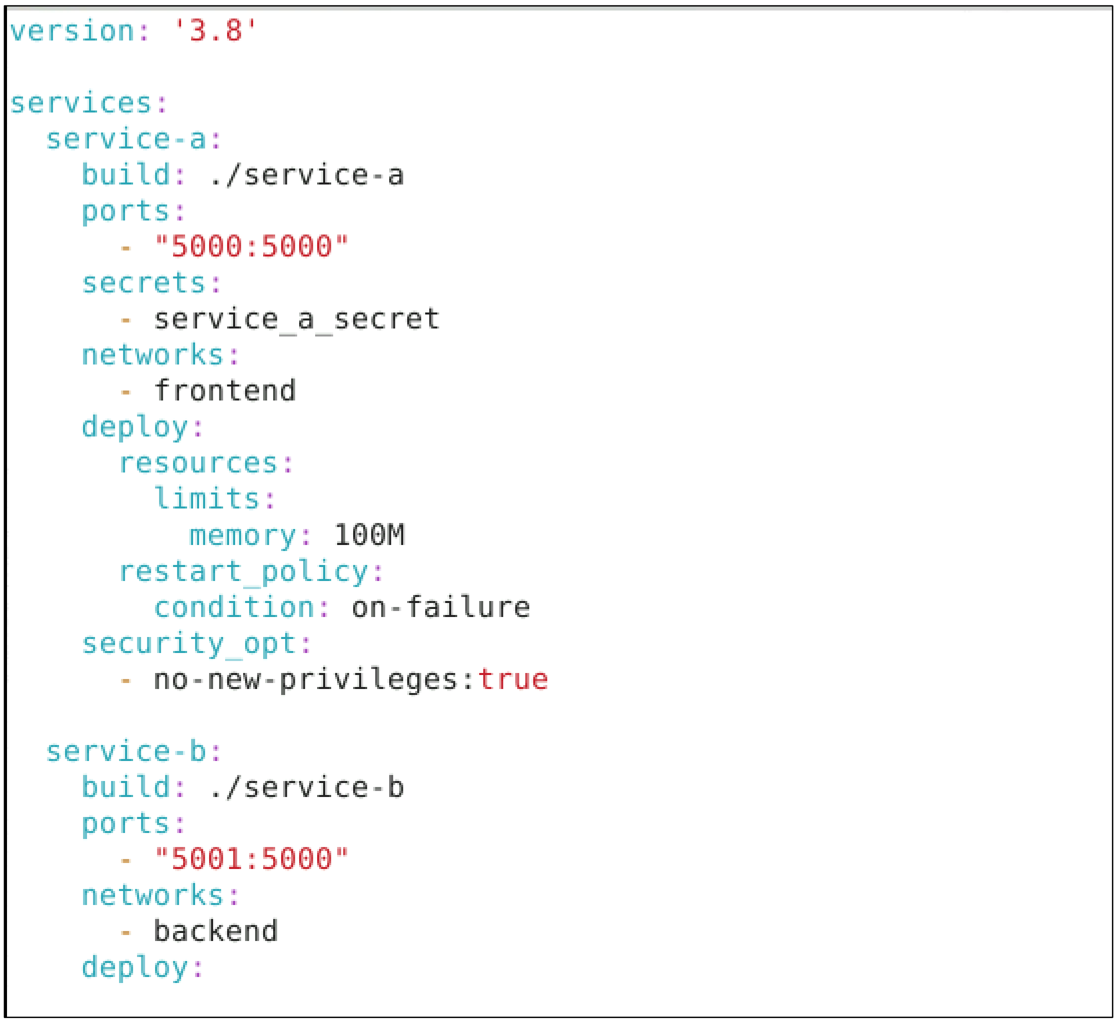
4.1 Create the **docker-compose** file using the following command:
vi docker-compose.yml



```
root@ip-172-31-32-167:/home/sakshiguptasimp/microservices-demo# vi docker-compose.yml
```

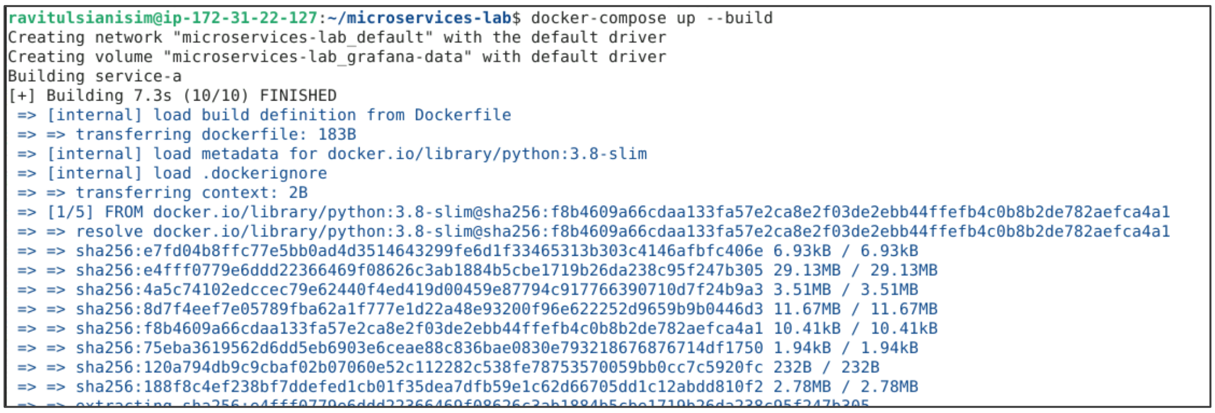
4.2 Add the following script in the created **docker-compose** file:

```
version: '3.8'

services:
  service-a:
    build: ./service-a
    ports:
      - "5000:5000"
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5000/health"]
      interval: 30s
      timeout: 10s
      retries: 3
  service-b:
    build: ./service-b
    ports:
      - "5001:5000"
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5001/health"]
      interval: 30s
      timeout: 10s
      retries: 3

  prometheus:
    image: prom/prometheus
    volumes:
      - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090"

  grafana:
    image: grafana/grafana
    ports:
      - "3000:3000"
    volumes:
      - grafana-data:/var/lib/grafana

volumes:
  grafana-data:
```

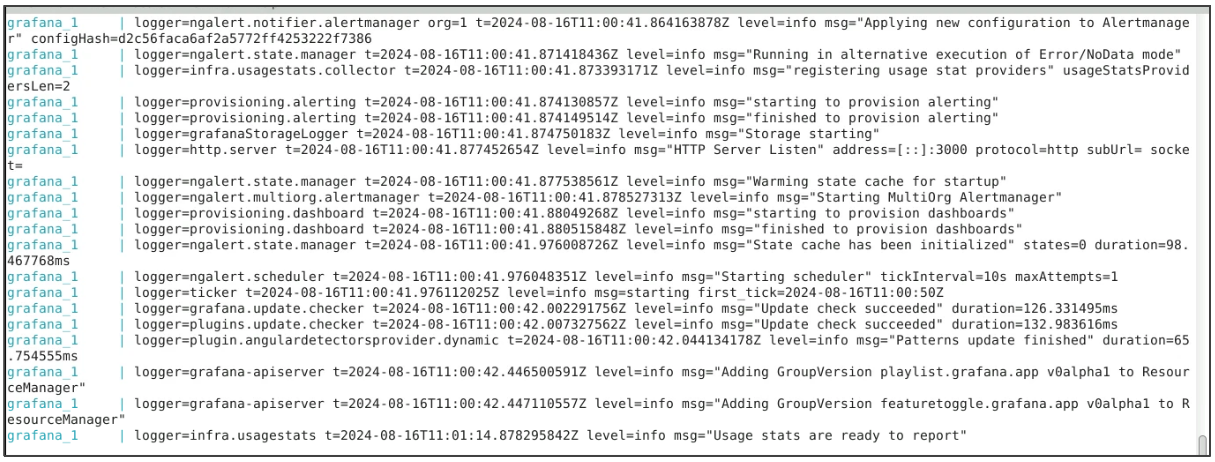


```
version: '3.8'

services:
  service-a:
    build: ./service-a
    ports:
      - "5000:5000"
    secrets:
      - service_a_secret
    networks:
      - frontend
    deploy:
      resources:
        limits:
          memory: 100M
      restart_policy:
        condition: on-failure
    security_opt:
      - no-new-privileges:true

  service-b:
    build: ./service-b
    ports:
      - "5001:5000"
    networks:
      - backend
    deploy:
```

4.3 Run the setup by using the following command:
docker-compose up --build

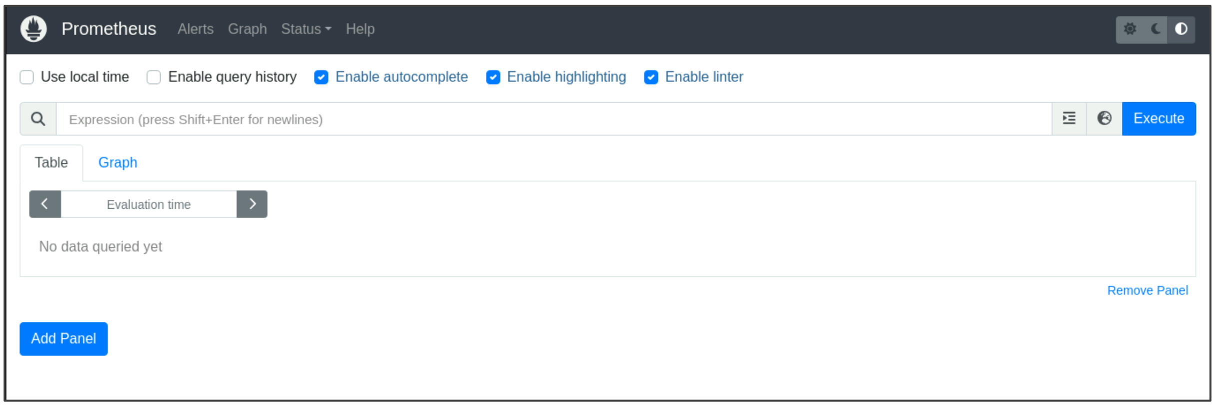


```
ravitulsianisim@ip-172-31-22-127:~/microservices-lab$ docker-compose up --build
Creating network "microservices-lab_default" with the default driver
Creating volume "microservices-lab_grafana-data" with default driver
Building service-a
[+] Building 7.3s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 183B
 => [internal] load metadata for docker.io/library/python:3.8-slim
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/5] FROM docker.io/library/python:3.8-slim@sha256:f8b4609a66cdaa133fa57e2ca8e2f03de2ebb44ffefb4c0b8b2de782aefca4a1
 => => resolve docker.io/library/python:3.8-slim@sha256:f8b4609a66cdaa133fa57e2ca8e2f03de2ebb44ffefb4c0b8b2de782aefca4a1
 => => sha256:e7fd04b8ffc77e5bb0ad4d3514643299fe6d1f33465313b303c4146afbfc406e 6.93kB / 6.93kB
 => => sha256:e4fff0779e6ddd22366469f08626c3ab1884b5cbe1719b26da238c95f247b305 29.13MB / 29.13MB
 => => sha256:4a5c74102edccec79e62440f4ed419d00459e87794c917766390710d7f24b9a3 3.51MB / 3.51MB
 => => sha256:8d7f4eef7e05789fba62a1f777e1d22a48e93200f96e622252d9659b9b0446d3 11.67MB / 11.67MB
 => => sha256:f8b4609a66cdaa133fa57e2ca8e2f03de2ebb44ffefb4c0b8b2de782aefca4a1 10.41kB / 10.41kB
 => => sha256:75eba3619562d6dd5eb6903e6ceae88c836bae0830e793218676876714df1750 1.94kB / 1.94kB
 => => sha256:120a794db9c9cbaf02b07060e52c112282c538fe78753570059bb0cc7c5920fc 232B / 232B
 => => sha256:188f8c4ef238bf7ddefed1cb01f35dea7dfb59e1c62d66705dd1c12abdd810f2 2.78MB / 2.78MB
 => => extracting sha256:e4fff0779e6ddd22366469f08626c3ab1884b5cbe1719b26da238c95f247b305
```



```
grafana_1     | logger=ngalert.notifier.alertmanager org=1 t=2024-08-16T11:00:41.864163878Z level=info msg="Applying new configuration to Alertmanage
r" configHash=d2c56faca6af2a5772ff4253222f7386
grafana_1     | logger=ngalert.state.manager t=2024-08-16T11:00:41.871418436Z level=info msg="Running in alternative execution of Error/NoData mode"
grafana_1     | logger=infra.usagestats.collector t=2024-08-16T11:00:41.873393171Z level=info msg="registering usage stat providers" usageStatsProvid
ersLen=2
grafana_1     | logger=provisioning.alerting t=2024-08-16T11:00:41.874130857Z level=info msg="starting to provision alerting"
grafana_1     | logger=provisioning.alerting t=2024-08-16T11:00:41.874149514Z level=info msg="finished to provision alerting"
grafana_1     | logger=grafanaStorageLogger t=2024-08-16T11:00:41.874750183Z level=info msg="Storage starting"
grafana_1     | logger=http.server t=2024-08-16T11:00:41.877452654Z level=info msg="HTTP Server Listen" address=[::]:3000 protocol=http subUrl= socke
t=
grafana_1     | logger=ngalert.state.manager t=2024-08-16T11:00:41.877538561Z level=info msg="Warming state cache for startup"
grafana_1     | logger=ngalert.multiorg.alertmanager t=2024-08-16T11:00:41.878527313Z level=info msg="Starting MultiOrg Alertmanager"
grafana_1     | logger=provisioning.dashboard t=2024-08-16T11:00:41.88049268Z level=info msg="starting to provision dashboards"
grafana_1     | logger=provisioning.dashboard t=2024-08-16T11:00:41.880515848Z level=info msg="finished to provision dashboards"
grafana_1     | logger=ngalert.state.manager t=2024-08-16T11:00:41.976008726Z level=info msg="State cache has been initialized" states=0 duration=98.
467768ms
grafana_1     | logger=ngalert.scheduler t=2024-08-16T11:00:41.976048351Z level=info msg="Starting scheduler" tickInterval=10s maxAttempts=1
grafana_1     | logger=ticker t=2024-08-16T11:00:41.976112025Z level=info msg=starting first_tick=2024-08-16T11:00:50Z
grafana_1     | logger=grafana.update.checker t=2024-08-16T11:00:42.002291756Z level=info msg="Update check succeeded" duration=126.331495ms
grafana_1     | logger=plugins.update.checker t=2024-08-16T11:00:42.007327562Z level=info msg="Update check succeeded" duration=132.983616ms
grafana_1     | logger=plugin.angulardetectorsprovider.dynamic t=2024-08-16T11:00:42.044134178Z level=info msg="Patterns update finished" duration=65
.754555ms
grafana_1     | logger=grafana-apiserver t=2024-08-16T11:00:42.446500591Z level=info msg="Adding GroupVersion playlist.grafana.app v0alpha1 to Resour
ceManager"
grafana_1     | logger=grafana-apiserver t=2024-08-16T11:00:42.447110557Z level=info msg="Adding GroupVersion featuretoggle.grafana.app v0alpha1 to R
esourceManager"
grafana_1     | logger=infra.usagestats t=2024-08-16T11:01:14.878295842Z level=info msg="Usage stats are ready to report"
```

4.4 Open the following URL in your default browser:
localhost:9090



By following these steps, you have successfully applied health checks and monitoring to microservices using Docker, Docker Compose, and Prometheus for enhanced reliability and observability.