EKS AND MONITORING WITH OPENTELEMENTRY

FINAL REPORT FOR FINAL PROJECT

GROUP - 6

Team Members :

Neonidh Singh - 116009102

Goutham Patchipulusu - 120112669

Muskan Sagar - 120152764

Venkata Umesh Chandra Pothugunta - 120427121

Bhanu Teja Panguluri - 120193378

TABLE OF CONTENTS

OVERVIEW

This project focuses on deploying a scalable, fault-tolerant application on AWS Kubernetes (EKS) while incorporating modern DevOps practices. The implementation is divided into six comprehensive phases, each addressing a critical aspect of application deployment, monitoring, and automation.

- **Phase 1:** Set up Docker on EC2 to validate initial deployment, then configure an EKS cluster with Kubernetes for application hosting.
- **Phase 2:** Organize YAML files for modular deployment, enabling targeted updates and easier debugging.
- **Phase 3:** Use Helm to deploy the application and simplify resource management, streamline updates, and enable quick rollbacks.
- **Phase 4:** Customize Grafana dashboards for monitoring clusters, pods, and nodes while setting alerts for critical metrics.
- **Phase 5:** Implement alerting with Prometheus and AWS SES to notify on pod restarts via email.
- **Phase 6:** Automate build, test, and deployment processes using a CI/CD pipeline with rollback capabilities.

This structured approach ensures efficient deployment, monitoring, and automation in a production-ready environment.

ENVIRONMENT AND INITIAL APPLICATION SETUP

Objective: The primary aim of this phase is to deploy and validate the OpenTelemetry demo application in two distinct environments: Docker and Kubernetes.

For the Docker environment, we'll deploy the application on a dedicated EC2 instance using the `*docker-compose.yaml*` file from the GitHub repository. This setup ensures that all services defined in the composition file are operational and accessible.

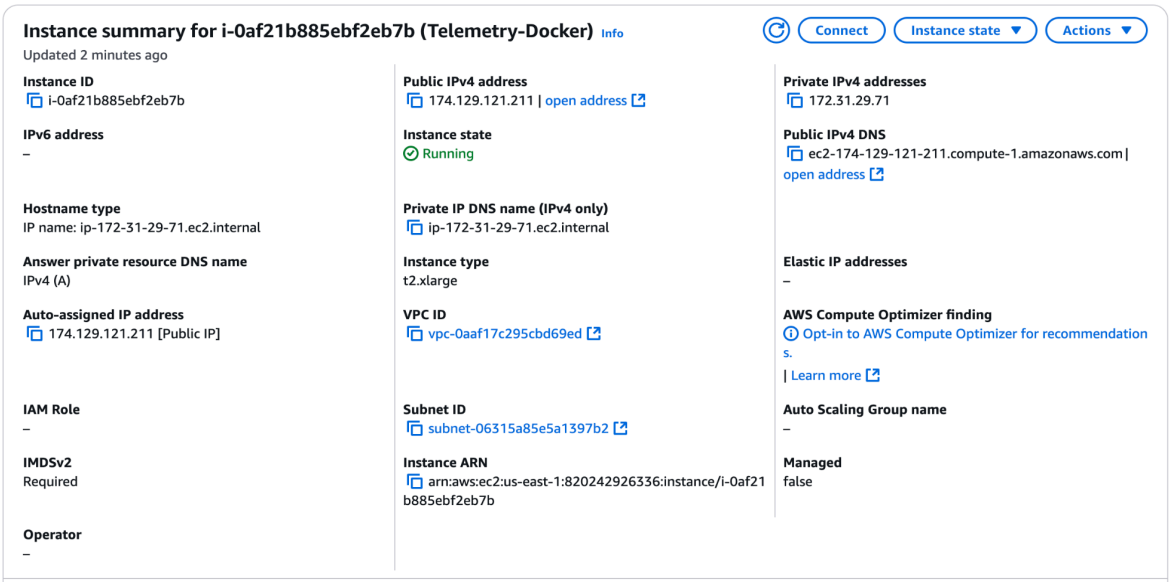
In the Kubernetes environment, we'll deploy the application on an EKS cluster with at least two worker nodes. This deployment will utilize the `*opentelemetry-demo.yaml*` manifest file to set up all necessary components.

Both deployments will undergo thorough validation to confirm service functionality, endpoint accessibility, and resource health. This phase establishes a solid foundation for exploring OpenTelemetry's observability features across different deployment scenarios.

Implementation:

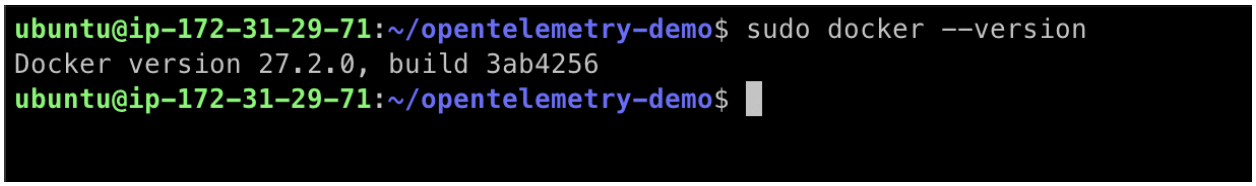
Docker Environment:

1. We start by configuring an EC-2 instance (Telemetry-Docker) with the following specifications:
    a. **AMI:** Ubuntu Server 24.04 LTS(HVM), SSD Volume Type
    b. **Instance Type:** t2.xlarge
    c. **Security Group:** allows SSH and HTTP traffic.
    d. **EBS Storage:** 16 GB



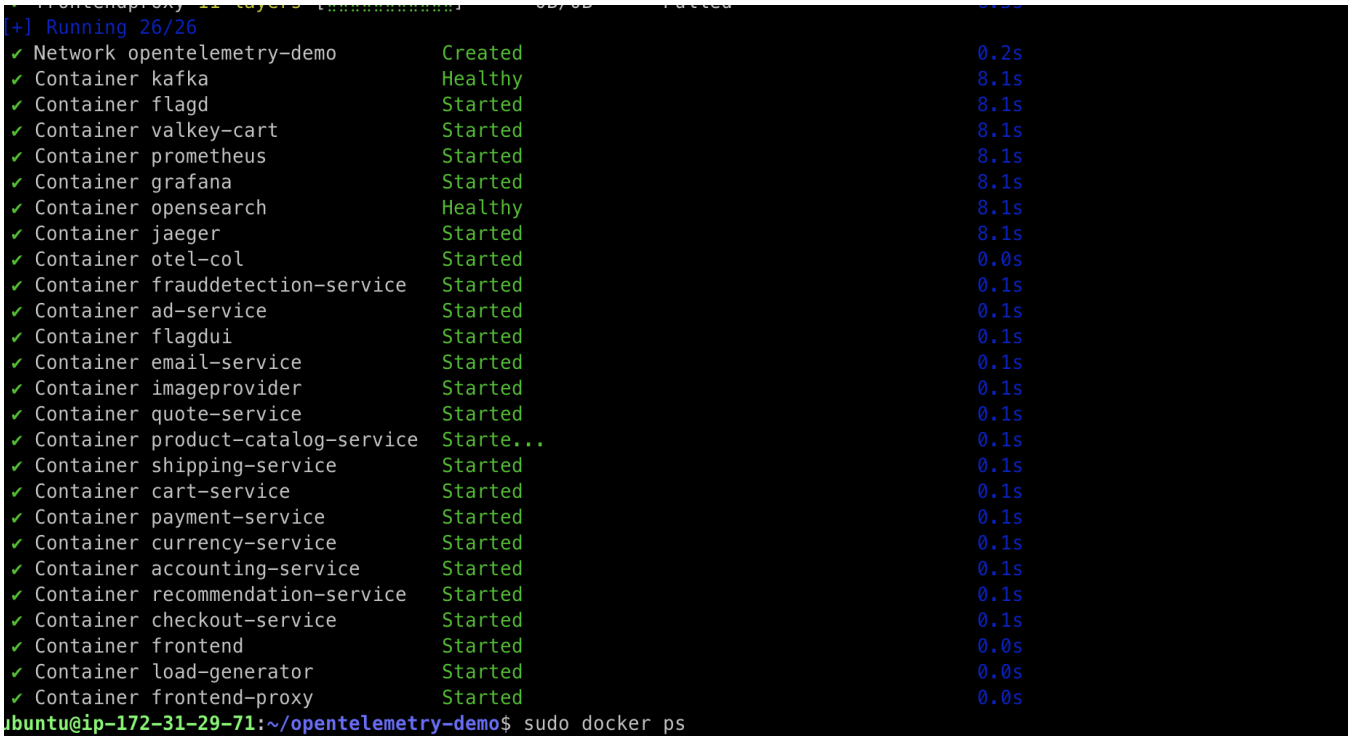
2. To use the '*docker-compose.yaml*' from github, we first have to install GitHub and other necessary dependencies.

```
Unset
# Update the system
sudo apt update -y

# Install dependencies
sudo apt install apt-transport-https ca-certificates curl
software-properties-common -y

# Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg

# Set up the stable repository
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null

# Update the package index again
sudo apt update -y

# Install Docker
sudo apt install docker-ce docker-ce-cli containerd.io -y

# Verify Docker installation
sudo docker --version
```




```
Unset
# Download Docker Compose
```

```
sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(un
ame -s)-$(uname -m)" -o /usr/local/bin/docker-compose

# Set executable permissions
sudo chmod +x /usr/local/bin/docker-compose

# Verify installation
docker-compose --version

# Install Git if not already installed
sudo apt install git -y

# Clone the repository
git clone https://github.com/open-telemetry/opentelemetry-demo.git

# Navigate to the project directory
cd opentelemetry-demo

# Bring up the services defined in the docker-compose.yml file
sudo docker-compose up --force-recreate --remove-orphans --detach
```



3. To validate the services defined in the '*docker-compose.yaml*' file, we run the following commands:

```
Unset
docker ps
```

```
ubuntu@ip-172-31-29-71:~/opentelemetry-demo$ sudo docker ps
CONTAINER ID   IMAGE                                                  COMMAND                  CREATED         STATUS                   PORTS
                                                                                                                                 NAMES
56d92f996b45   ghcr.io/open-telemetry/demo:latest-frontendproxy       "/bin/sh -c 'envsubs…"   9 minutes ago   Up 8 minutes             0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 0.0.
0.0:10000->10000/tcp, :::10000->10000/tcp                                                                                        frontend-proxy
1948250166f5   ghcr.io/open-telemetry/demo:latest-loadgenerator       "locust --skip-log-s…"   9 minutes ago   Up 8 minutes             0.0.0.0:32789->8089/tcp, [::]:32789->8089/tcp
                                                                                                                                 load-generator
bce2b625b433   ghcr.io/open-telemetry/demo:latest-frontend            "npm start"              9 minutes ago   Up 8 minutes             0.0.0.0:32788->8080/tcp, [::]:32788->8080/tcp
                                                                                                                                 frontend
bb4bf1720937   ghcr.io/open-telemetry/demo:latest-checkoutservice     "./checkoutservice"      9 minutes ago   Up 8 minutes             0.0.0.0:32787->5050/tcp, [::]:32787->5050/tcp
                                                                                                                                 checkout-service
191d9fa57a53   ghcr.io/open-telemetry/demo:latest-recommendationservice   "opentelemetry-instr…"   9 minutes ago   Up 8 minutes         0.0.0.0:32786->9001/tcp, [::]:32786->9001/tcp
                                                                                                                                 recommendation-service
9ee09cb8a8f0   ghcr.io/open-telemetry/demo:latest-accountingservice   "./instrument.sh dot…"   9 minutes ago   Up 8 minutes
                                                                                                                                 accounting-service
b97f39acd5b0   ghcr.io/open-telemetry/demo:latest-cartservice         "./cartservice"          9 minutes ago   Up 8 minutes             0.0.0.0:32784->7070/tcp, [::]:32784->7070/tcp
                                                                                                                                 cart-service
fd7cae30a3af   ghcr.io/open-telemetry/demo:latest-paymentservice      "npm run start"          9 minutes ago   Up 8 minutes             0.0.0.0:32782->50051/tcp, [::]:32782->50051/tcp
                                                                                                                                 payment-service
4ab2dbdfc32d   ghcr.io/open-telemetry/demo:latest-productcatalogservice   "./productcatalogser…"   9 minutes ago   Up 8 minutes         0.0.0.0:32781->3550/tcp, [::]:32781->3550/tcp
                                                                                                                                 product-catalog-service
fa704607fc00   ghcr.io/open-telemetry/demo:latest-imageprovider       "/docker-entrypoint.…"   9 minutes ago   Up 8 minutes             80/tcp, 0.0.0.0:32777->8081/tcp, [::]:32777->80
81/tcp                                                                                                                           imageprovider
4c3430d4a639   ghcr.io/open-telemetry/demo:latest-currencyservice     "sh -c './usr/local/…"   9 minutes ago   Up 8 minutes             0.0.0.0:32776->7001/tcp, [::]:32776->7001/tcp
                                                                                                                                 currency-service
b57a95934c85   ghcr.io/open-telemetry/demo:latest-shippingservice     "/app/shippingservice"   9 minutes ago   Up 8 minutes             0.0.0.0:32780->50050/tcp, [::]:32780->50050/tcp
                                                                                                                                 shipping-service
924e3d04f109   ghcr.io/open-telemetry/demo:latest-quoteservice        "docker-php-entrypoi…"   9 minutes ago   Up 8 minutes             0.0.0.0:32785->8090/tcp, [::]:32785->8090/tcp
                                                                                                                                 quote-service
330489ac03cf   ghcr.io/open-telemetry/demo:latest-adservice           "./build/install/ope…"   9 minutes ago   Up 8 minutes             0.0.0.0:32783->9555/tcp, [::]:32783->9555/tcp
                                                                                                                                 ad-service
a6ab0060aea2   ghcr.io/open-telemetry/demo:latest-frauddetectionservice   "java -jar frauddete…"   9 minutes ago   Up 8 minutes
                                                                                                                                 frauddetection-service
a514cae4f38f   ghcr.io/open-telemetry/demo:latest-flagdui             "docker-entrypoint.s…"   9 minutes ago   Up 2 minutes             0.0.0.0:32790->4000/tcp, [::]:32790->4000/tcp
                                                                                                                                 flagdui
4a7d2f6cf82f   ghcr.io/open-telemetry/demo:latest-emailservice        "bundle exec ruby em…"   9 minutes ago   Up 8 minutes             0.0.0.0:32779->6060/tcp, [::]:32779->6060/tcp
                                                                                                                                 email-service
4e233a08e01d   otel/opentelemetry-collector-contrib:0.113.0           "/otelcol-contrib --…"   9 minutes ago   Up 8 minutes             55678-55679/tcp, 0.0.0.0:32774->4317/tcp, [::]:
32774->4317/tcp, 0.0.0.0:32775->4318/tcp, [::]:32775->4318/tcp                                                                   otel-col
06bcc65dedc5   valkey/valkey:8.0-alpine                               "docker-entrypoint.s…"   9 minutes ago   Up 9 minutes             0.0.0.0:32773->6379/tcp, [::]:32773->6379/tcp
                                                                                                                                 valkey-cart
3e02c60f56bc   grafana/grafana:11.3.0                                 "/run.sh"                9 minutes ago   Up 9 minutes             0.0.0.0:32770->3000/tcp, [::]:32770->3000/tcp
                                                                                                                                 grafana
9a0434923b45   quay.io/prometheus/prometheus:v2.55.1                  "/bin/prometheus --w…"   9 minutes ago   Up 9 minutes             0.0.0.0:9090->9090/tcp, :::9090->9090/tcp
                                                                                                                                 prometheus
036282717c75   jaegertracing/all-in-one:1.62.0                        "/go/bin/all-in-one-…"   9 minutes ago   Up 9 minutes             4318/tcp, 5775/udp, 5778/tcp, 9411/tcp, 14250/t
cp, 14268/tcp, 6831-6832/udp, 0.0.0.0:32769->4317/tcp, [::]:32769->4317/tcp, 0.0.0.0:32771->16686/tcp, [::]:32771->16686/tcp   jaeger
556d8f999c76   ghcr.io/open-feature/flagd:v0.11.4                     "/flagd-build start …"   9 minutes ago   Up 9 minutes             0.0.0.0:32768->8013/tcp, [::]:32768->8013/tcp
                                                                                                                                 flagd
```

```
Unset
sudo docker-compose logs --tail 5
```

```
prometheus              | ts=2024-12-05T21:33:52.241Z caller=write_handler.go:279 level=warn component=web msg="Error on ingesting out-of-order exemplars"
num_dropped=1
prometheus              | ts=2024-12-05T21:33:56.856Z caller=write_handler.go:279 level=warn component=web msg="Error on ingesting out-of-order exemplars"
num_dropped=18
prometheus              | ts=2024-12-05T21:33:58.862Z caller=write_handler.go:279 level=warn component=web msg="Error on ingesting out-of-order exemplars"
num_dropped=20
prometheus              | ts=2024-12-05T21:34:04.896Z caller=write_handler.go:279 level=warn component=web msg="Error on ingesting out-of-order exemplars"
num_dropped=20
prometheus              | ts=2024-12-05T21:34:14.922Z caller=write_handler.go:279 level=warn component=web msg="Error on ingesting out-of-order exemplars"
num_dropped=20
valkey-cart             | 1:M 05 Dec 2024 21:30:19.001 * 100 changes in 300 seconds. Saving...
valkey-cart             | 1:M 05 Dec 2024 21:30:19.002 * Background saving started by pid 12
valkey-cart             | 12:C 05 Dec 2024 21:30:19.009 * DB saved on disk
valkey-cart             | 12:C 05 Dec 2024 21:30:19.009 * Fork CoW for RDB: current 0 MB, peak 0 MB, average 0 MB
valkey-cart             | 1:M 05 Dec 2024 21:30:19.102 * Background saving terminated with success
quote-service           | [2024-12-05 21:33:10+00:00] "POST /getquote HTTP/1.1" 200 5
quote-service           | [2024-12-05 21:33:23+00:00] "POST /getquote HTTP/1.1" 200 5
quote-service           | [2024-12-05 21:33:29+00:00] "POST /getquote HTTP/1.1" 200 5
quote-service           | [2024-12-05 21:33:31+00:00] "POST /getquote HTTP/1.1" 200 4
quote-service           | [2024-12-05 21:34:06+00:00] "POST /getquote HTTP/1.1" 200 5
recommendation-service  | 2024-12-05 21:34:17,099 INFO [main] [recommendation_server.py:47] [trace_id=7708ea5478ecc6d044485557f4778c14 span_id=650d4fe208f8
7a5d resource.service.name=recommendationservice trace_sampled=True] - Receive ListRecommendations for product ids:['OLJCESPC7Z', '6E92ZMYYFZ', 'L9ECAV7KIM'
, 'LS4PSXUNUM', '9SIQT8TOJO']
recommendation-service  | 2024-12-05 21:34:17,134 INFO [main] [recommendation_server.py:47] [trace_id=ff89ce480b324b17a3b3efd679a6626b span_id=20a1ec0f4c1b
8bd6 resource.service.name=recommendationservice trace_sampled=True] - Receive ListRecommendations for product ids:['0PUK6V6EV0', 'L9ECAV7KIM', '9SIQT8TOJO'
, 'HQTGWGPNH4', '6E92ZMYYFZ']
recommendation-service  | 2024-12-05 21:34:18,228 INFO [main] [recommendation_server.py:47] [trace_id=68417dbafe88671a1503b36a78885e76 span_id=5d08813d1417
d646 resource.service.name=recommendationservice trace_sampled=True] - Receive ListRecommendations for product ids:['OLJCESPC7Z', '2ZYFJ3GM2N', '0PUK6V6EV0'
, 'HQTGWGPNH4', 'LS4PSXUNUM']
recommendation-service  | 2024-12-05 21:34:18,276 INFO [main] [recommendation_server.py:47] [trace_id=493849c8ca5a5ad779905ada40900654 span_id=28f49f72bd1d
f0d9 resource.service.name=recommendationservice trace_sampled=True] - Receive ListRecommendations for product ids:['9SIQT8TOJO', '1YMWWN1N4O', 'HQTGWGPNH4'
, 'L9ECAV7KIM', '6E92ZMYYFZ']
recommendation-service  | 2024-12-05 21:34:18,332 INFO [main] [recommendation_server.py:47] [trace_id=63c45a3f5b8c72be5d4bc3f4ff570277 span_id=815a5e28af49
2adc resource.service.name=recommendationservice trace_sampled=True] - Receive ListRecommendations for product ids:['9SIQT8TOJO', 'HQTGWGPNH4', 'LS4PSXUNUM'
, '1YMWWN1N4O', '2ZYFJ3GM2N']
product-catalog-service | time="2024-12-05T21:25:50Z" level=info msg="Loaded 10 products"
product-catalog-service | time="2024-12-05T21:25:50Z" level=info msg="ProductCatalogService gRPC server started on port: 3550"
shipping-service        | 21:33:31 [INFO] Sending Quote: 1308.0
shipping-service        | 21:33:31 [INFO] Tracking ID Created: d6af0fa0-1094-47dc-aa87-04ad9e23c620
shipping-service        | 21:34:06 [INFO] Received quote: "553.6"
shipping-service        | 21:34:06 [INFO] Sending Quote: 553.60
shipping-service        | 21:34:06 [INFO] Tracking ID Created: 90e3e753-03d8-4d41-b56a-9c9b2c71bd73
```

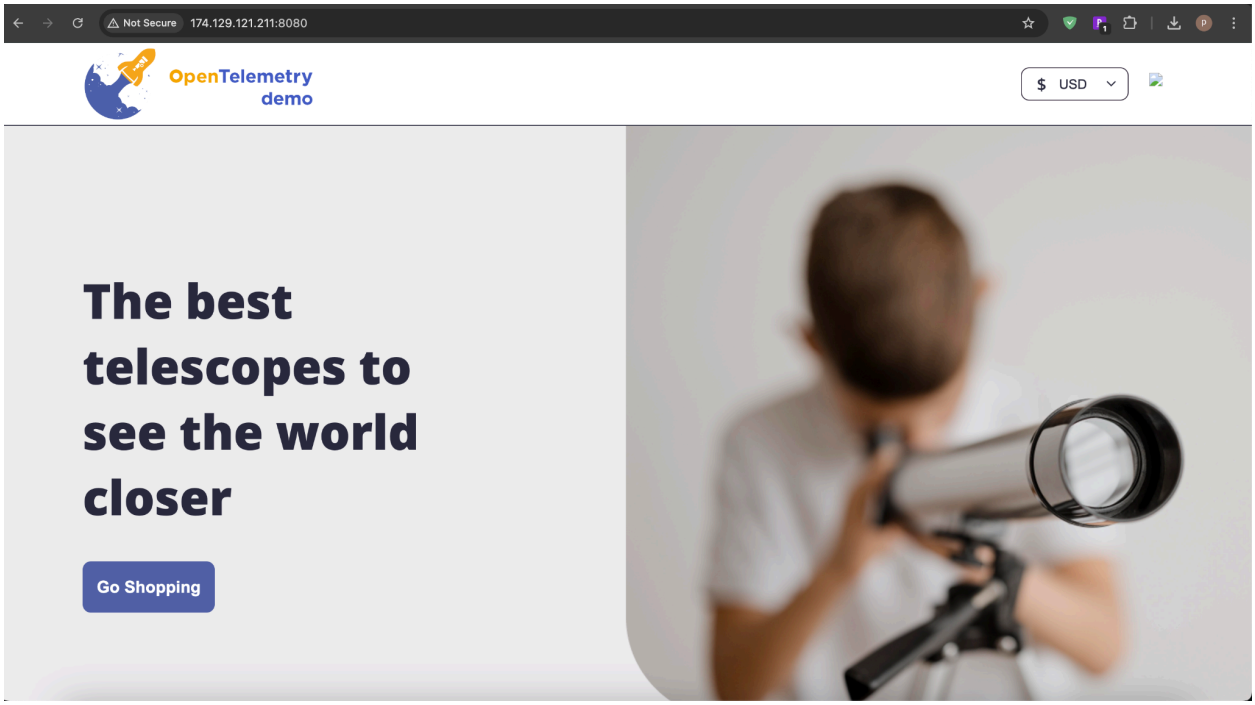
4. We confirm the service functionality by accessing the application endpoints:

```
Unset
sudo docker-compose logs frontend
```

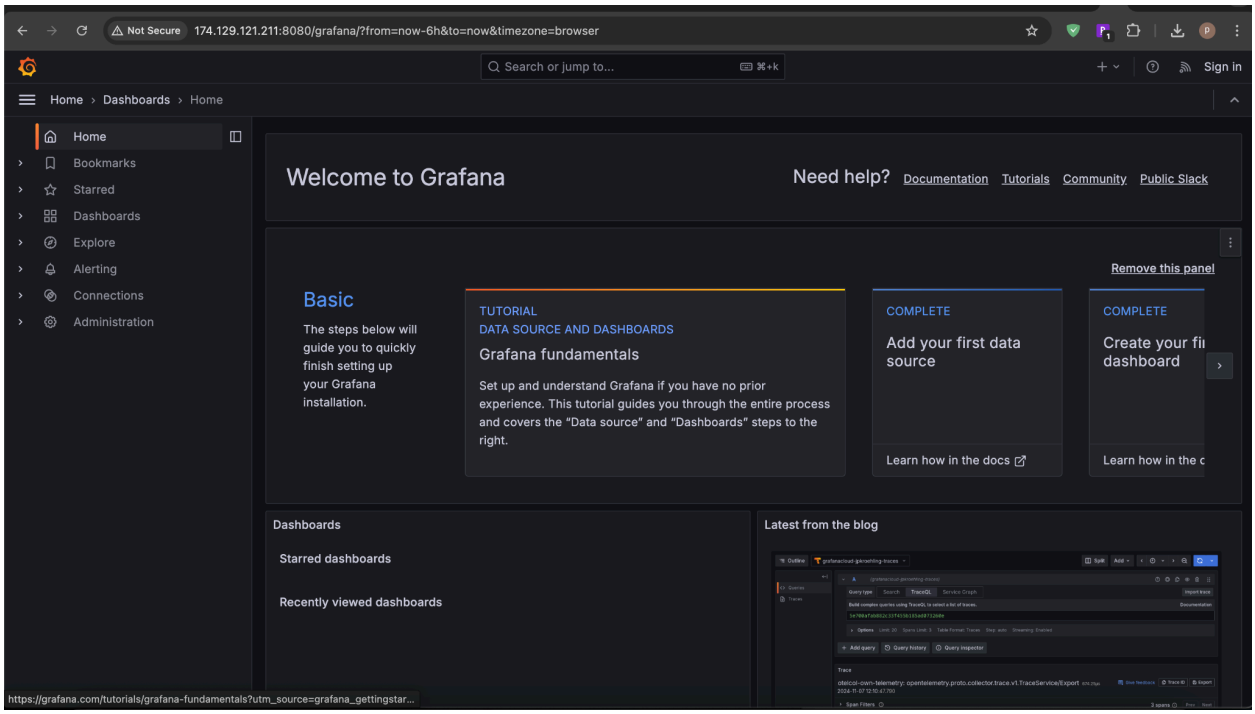
```
ubuntu@ip-172-31-32-196:~/opentelemetry-demo$ sudo docker-compose logs frontend
Attaching to frontend
frontend                |
frontend                | > frontend@0.1.0 start
frontend                | > node --require ./Instrumentation.js server.js
frontend                |
frontend                |   ▲ Next.js 14.2.5
frontend                |   - Local:        http://6fe515b21287:8080
frontend                |   - Network:      http://172.18.0.24:8080
frontend                |
frontend                |  ✓ Starting...
frontend                |  ✓ Ready in 955ms
```

We can see the services are available at port 8080.

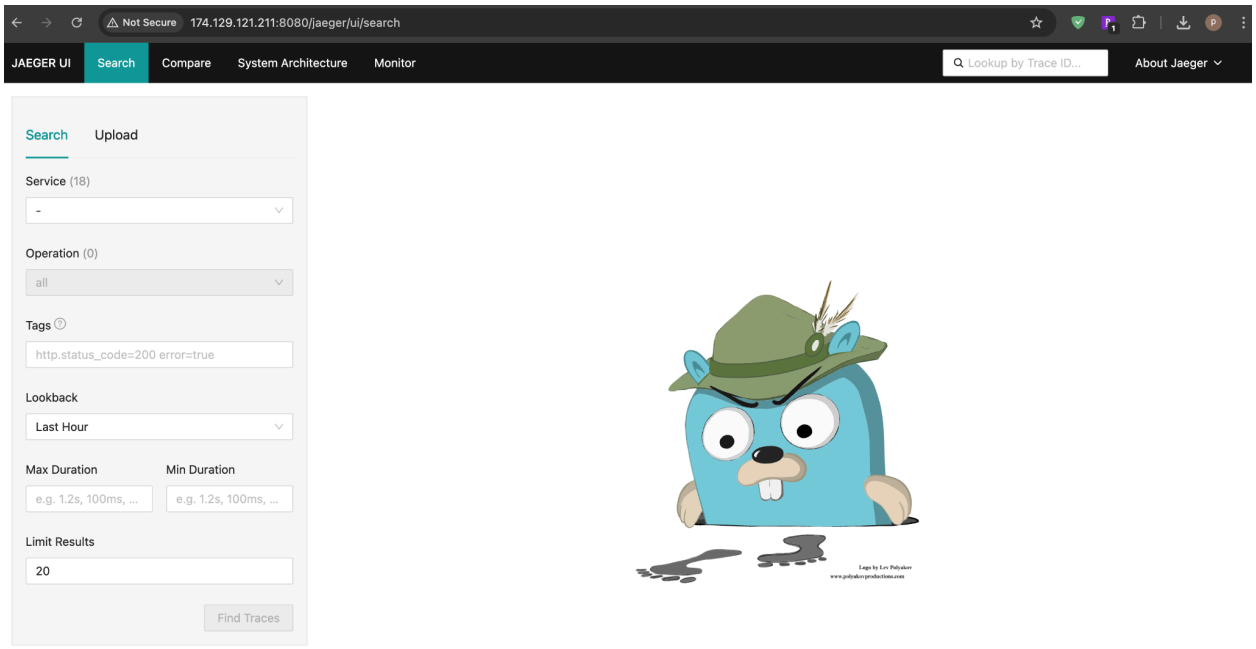
- **Web Store** (*http://174.129.121.211:8080*)



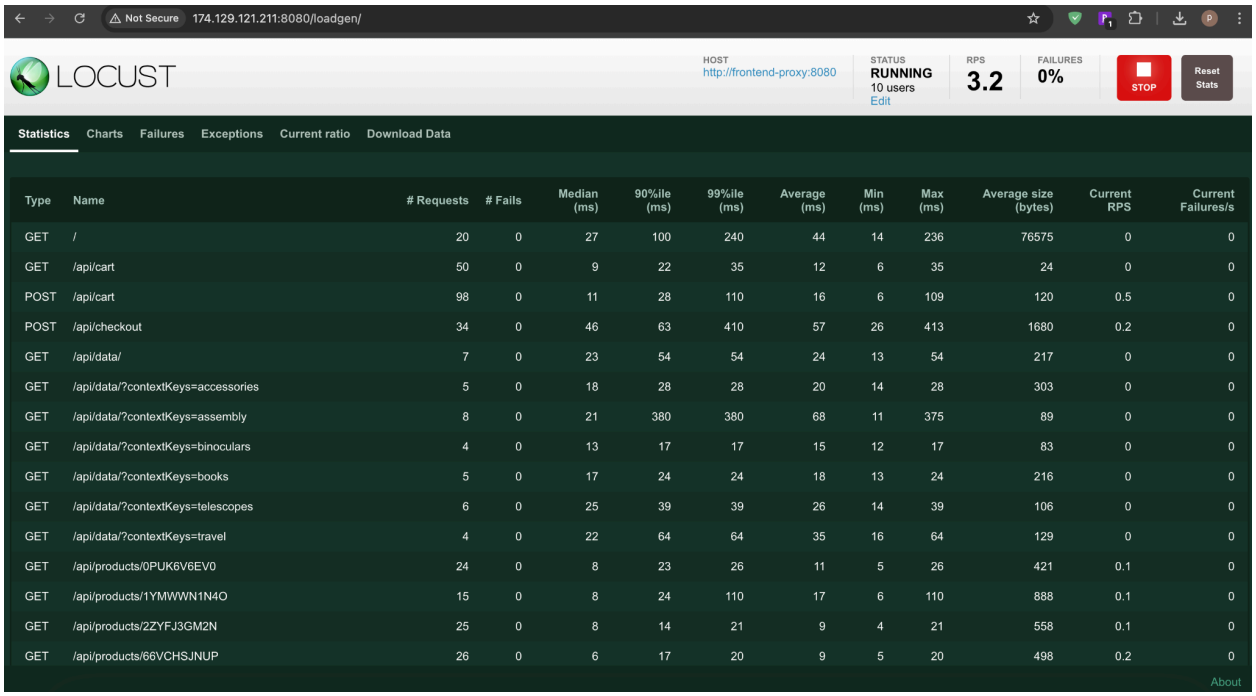
- **Grafana** (*http://174.129.121.211:8080/grafana/*)



- **Jaeger UI** (*http://174.129.121.211:8080/jaeger/ui/*)

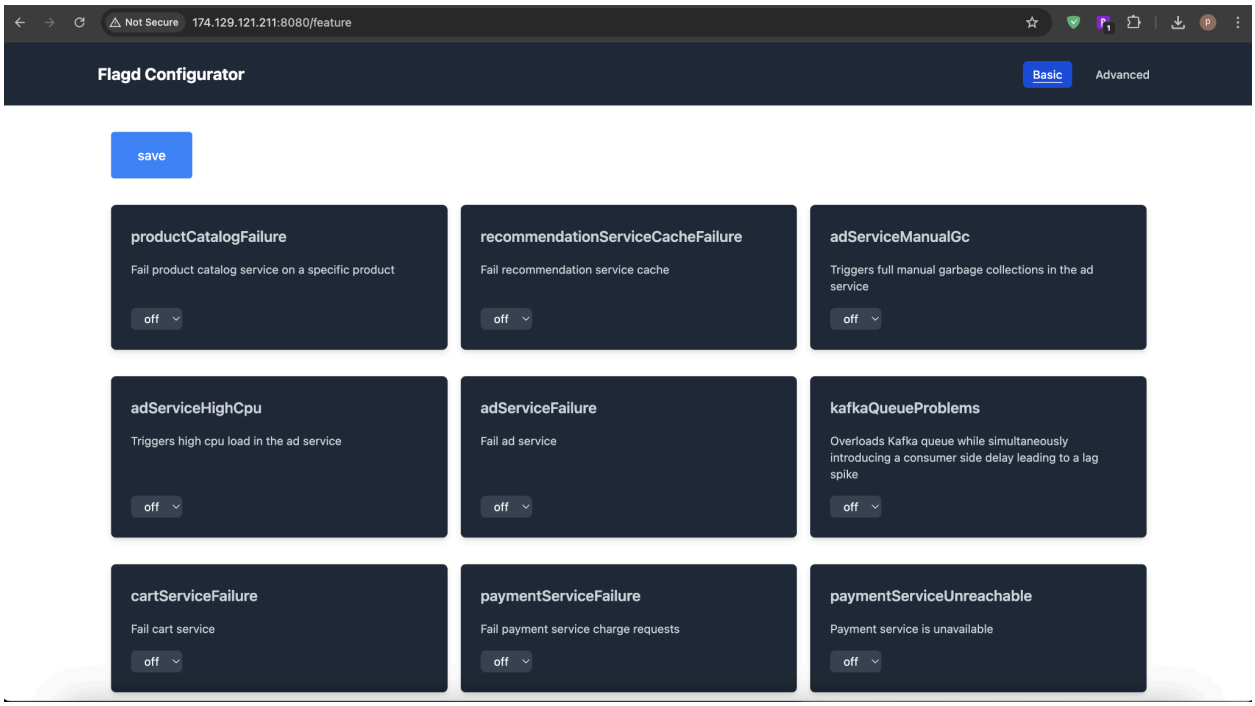


- **Load Generator UI** (*http://174.129.121.211:8080/loadgen/*)



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	20	0	27	100	240	44	14	236	76575	0	0
GET	/api/cart	50	0	9	22	35	12	6	35	24	0	0
POST	/api/cart	98	0	11	28	110	16	6	109	120	0.5	0
POST	/api/checkout	34	0	46	63	410	57	26	413	1680	0.2	0
GET	/api/data/	7	0	23	54	54	24	13	54	217	0	0
GET	/api/data/?contextKeys=accessories	5	0	18	28	28	20	14	28	303	0	0
GET	/api/data/?contextKeys=assembly	8	0	21	380	380	68	11	375	89	0	0
GET	/api/data/?contextKeys=binoculars	4	0	13	17	17	15	12	17	83	0	0
GET	/api/data/?contextKeys=books	5	0	17	24	24	18	13	24	216	0	0
GET	/api/data/?contextKeys=telescopes	6	0	25	39	39	26	14	39	106	0	0
GET	/api/data/?contextKeys=travel	4	0	22	64	64	35	16	64	129	0	0
GET	/api/products/0PUK6V6EV0	24	0	8	23	26	11	5	26	421	0.1	0
GET	/api/products/1YMWWN1N4O	15	0	8	24	110	17	6	110	888	0.1	0
GET	/api/products/2ZYFJ3GM2N	25	0	8	14	21	9	4	21	558	0.1	0
GET	/api/products/66VCHSJNUP	26	0	6	17	20	9	5	20	498	0.2	0

- **Flagd Configurator UI** (*http://174.129.121.211:8080/feature/*)



Conclusion

We completed the Docker implementation with all planned tasks executed. We validated the application endpoints and confirmed they were accessible, verifying that all services were running properly. In the Docker deployment, we launched and verified all containers specified in the *docker-compose.yaml* file using Docker commands, and confirmed service functionality through endpoint testing.

Implementation:

Kubernetes Environment:

1. We want to set up a dedicated EC2 instance as an EKS cluster with at least 2 worker nodes. So firstly, we configure an EC2 instance (EKS-Client) with the following specifications:
    a. **AMI:** Ubuntu Server 24.04 LTS(HVM), SSD Volume Type
    b. **Instance Type:** t2.medium
    c. **Security Group:** allows SSH and HTTP traffic.
    d. **EBS Storage:** 8 GB

2. Before setting up the cluster, we should install a few required dependencies..

- Install AWS CLI: Run the following commands to install AWS CLI v2:

```
Unset

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
aws --version  # Check if AWS CLI is installed
aws configure
```

- Install kubectl:

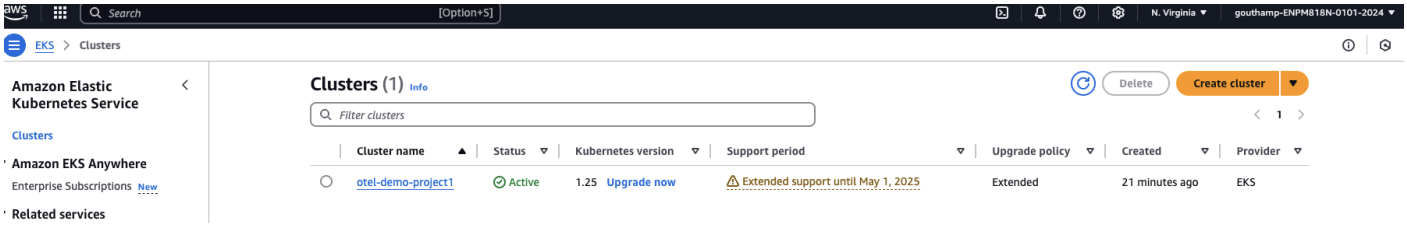
```
Unset

curl -LO
"https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bi
n/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
kubectl version --client  # Verify kubectl installation
```

- Install eksctl:

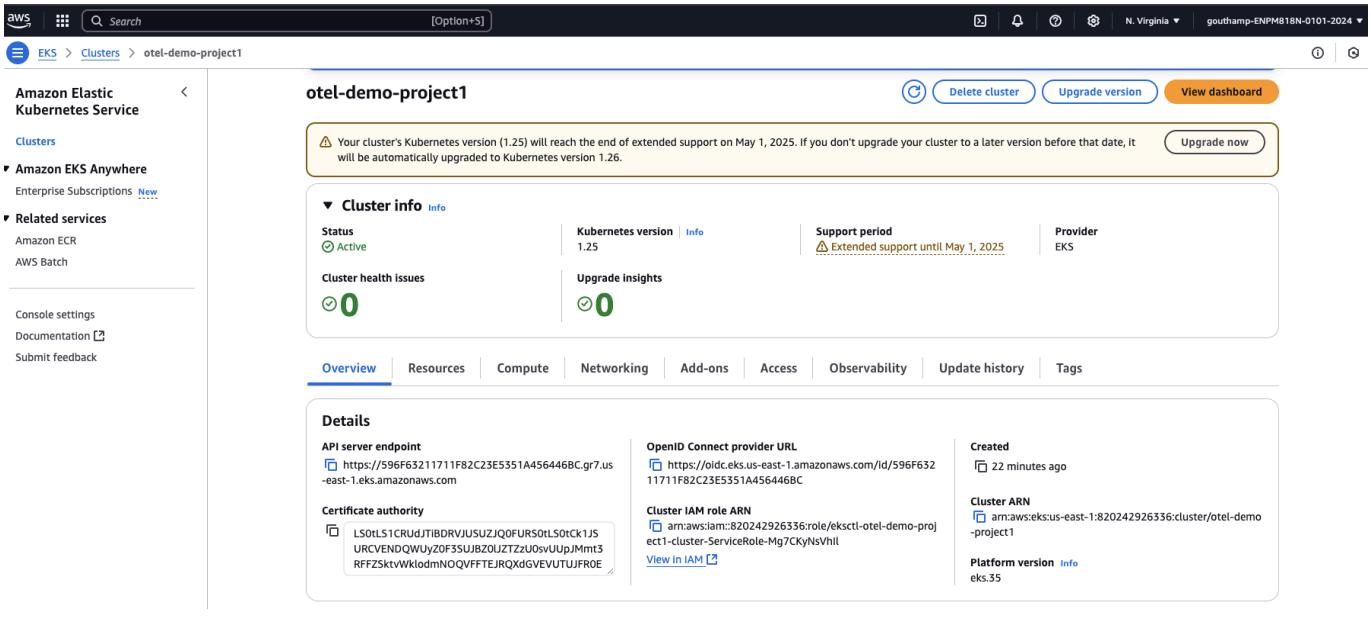
```
Unset

curl --silent --location
"https://github.com/weaveworks/eksctl/releases/download/v0.138.0/eksctl_
Linux_amd64.tar.gz" | tar xz -C /tmp
sudo mv /tmp/eksctl /usr/local/bin
eksctl version  # Verify eksctl installation
```

3. After installing the necessary dependencies, we set up the cluster using eksctl:

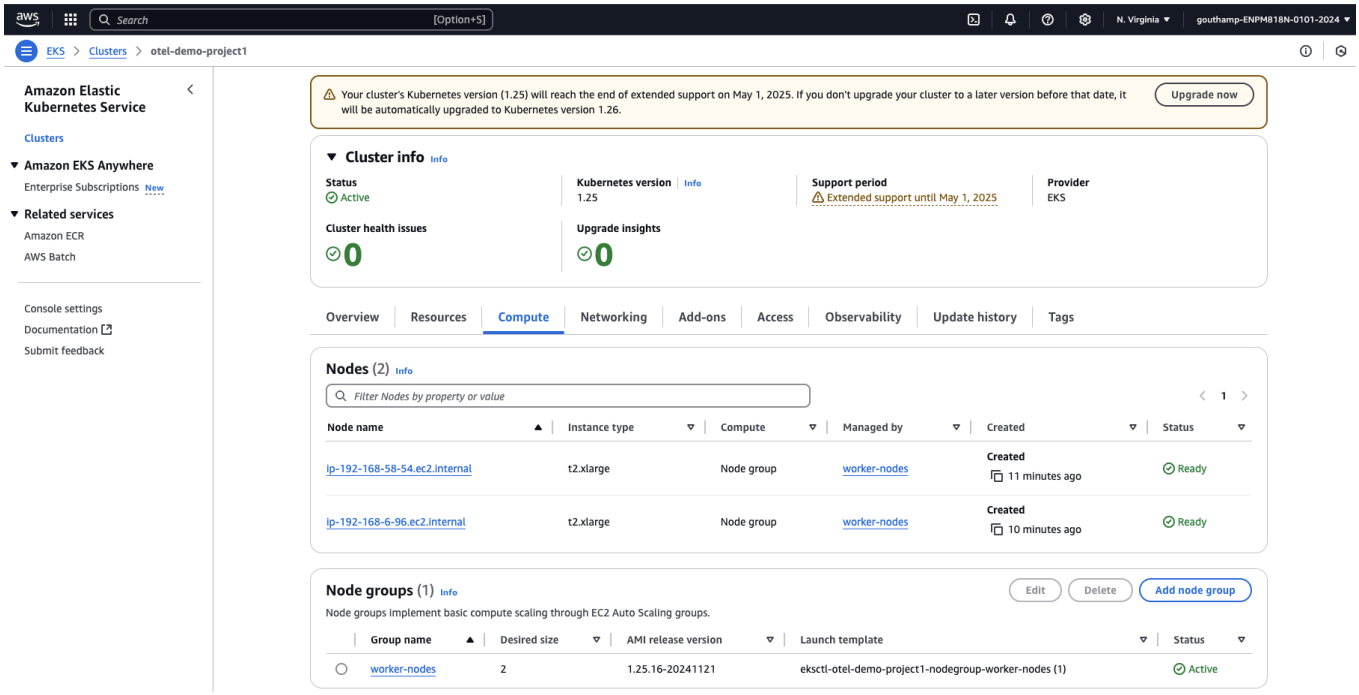
```
Unset

eksctl create cluster \
  --name otel-demo-project1 \
  --region us-east-1 \
  --nodegroup-name worker-nodes \
  --node-type t2.xlarge \
  --nodes 2 \
  --nodes-min 2 \
  --nodes-max 3 \
  --managed
```



- Cluster



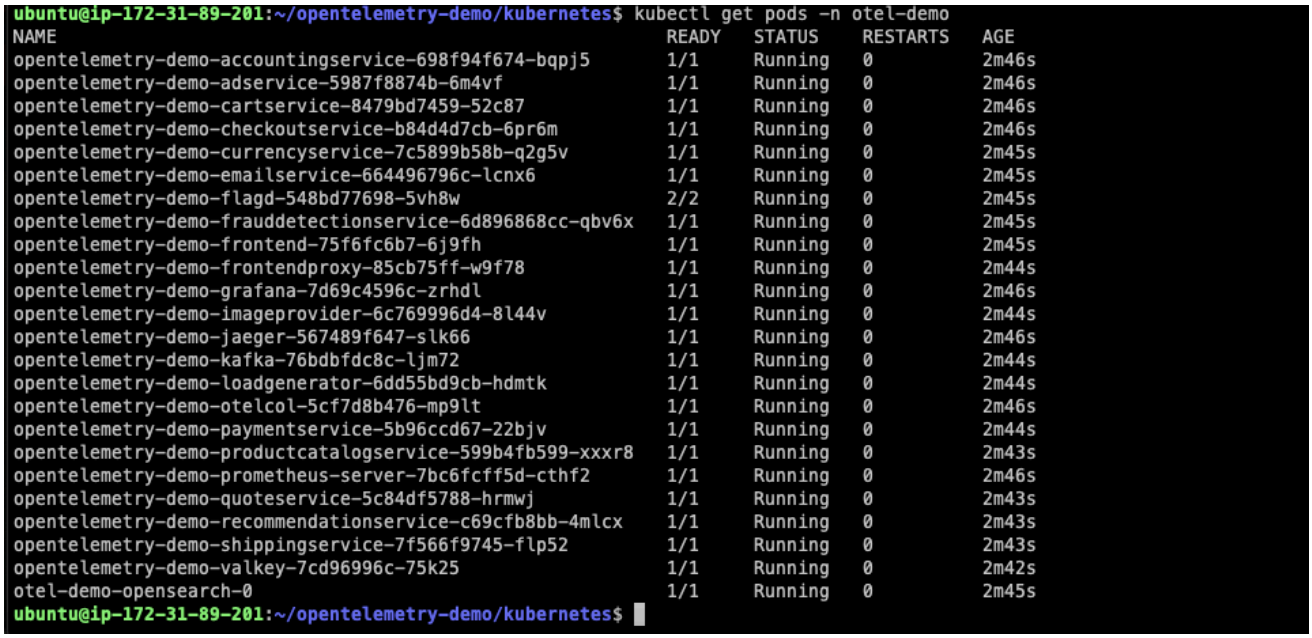
- otel-demo-project1 overview



- Cluster Nodes and Node Groups

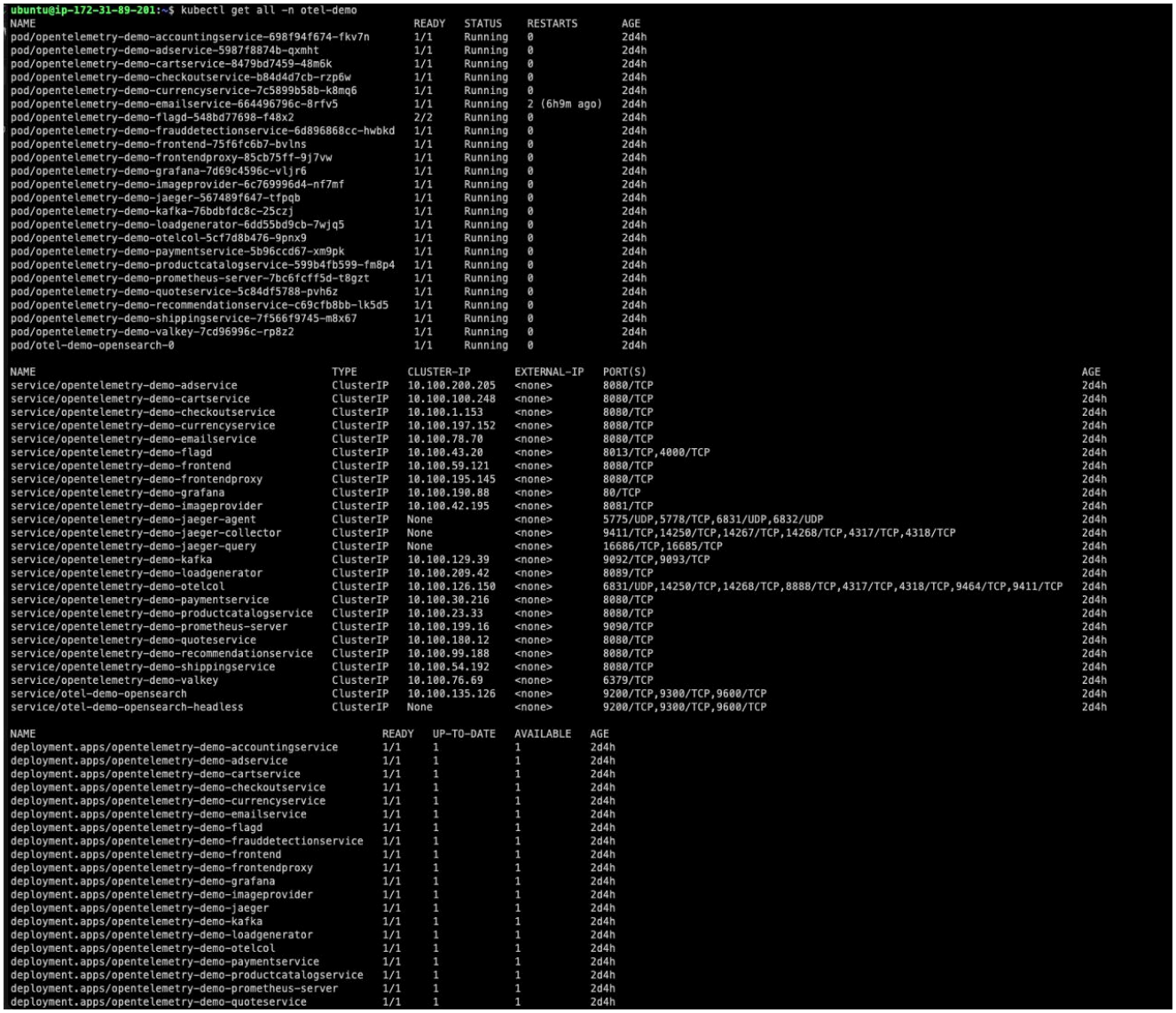
4. Now that the cluster is set up, we deploy the OpenTelemetry Demo application to EKS using kubectl

```
Unset

kubectl apply --namespace otel-demo -f
https://raw.githubusercontent.com/open-telemetry/opentelemetry-demo/main
/kubernetes/opentelemetry-demo.yaml

kubectl get pods -n otel-demo #display all pods
kubectl get svc -n otel-demo #dispaly all services
```



Unset

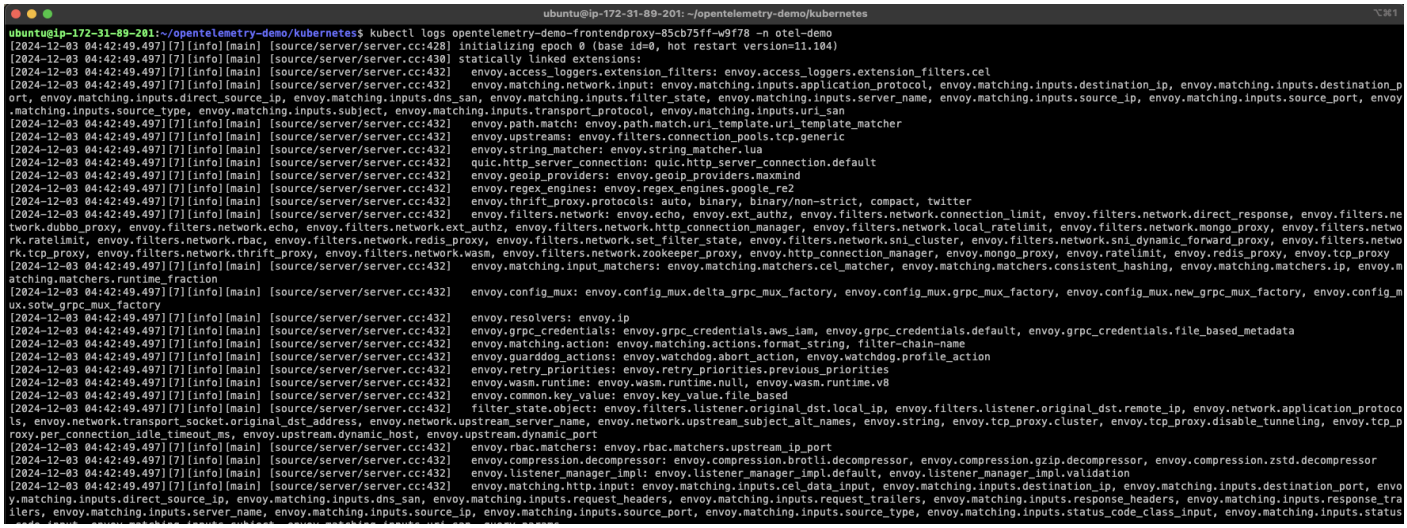
```
kubectl get all -n otel-demo
```



```
ubuntu@ip-172-31-89-201:~$ kubectl get all -n otel-demo
NAME                                                     READY   STATUS    RESTARTS       AGE
pod/opentelemetry-demo-accountingservice-698f94f674-fkv7n       1/1     Running   0              2d4h
pod/opentelemetry-demo-adservice-5987f8874b-qxmht               1/1     Running   0              2d4h
pod/opentelemetry-demo-cartservice-8479bd7459-48m6k             1/1     Running   0              2d4h
pod/opentelemetry-demo-checkoutservice-b84d4d7cb-rzp6w          1/1     Running   0              2d4h
pod/opentelemetry-demo-currencyservice-7c5899b58b-k8mq6         1/1     Running   0              2d4h
pod/opentelemetry-demo-emailservice-664496796c-8rfv5            1/1     Running   2 (6h9m ago)   2d4h
pod/opentelemetry-demo-flagd-548bd77698-f48x2                   2/2     Running   0              2d4h
pod/opentelemetry-demo-frauddetectionservice-6d896868cc-hwbkd   1/1     Running   0              2d4h
pod/opentelemetry-demo-frontend-75f6fc6b7-bvlns                 1/1     Running   0              2d4h
pod/opentelemetry-demo-frontendproxy-85cb75ff-9j7vw             1/1     Running   0              2d4h
pod/opentelemetry-demo-grafana-7d69c4596c-vljr6                 1/1     Running   0              2d4h
pod/opentelemetry-demo-imageprovider-6c769996d4-nf7mf           1/1     Running   0              2d4h
pod/opentelemetry-demo-jaeger-567489f647-tfpqb                  1/1     Running   0              2d4h
pod/opentelemetry-demo-kafka-76bdbfdc8c-25czj                   1/1     Running   0              2d4h
pod/opentelemetry-demo-loadgenerator-6dd55bd9cb-7wjq5           1/1     Running   0              2d4h
pod/opentelemetry-demo-otelcol-5cf7d8b476-9pnx9                 1/1     Running   0              2d4h
pod/opentelemetry-demo-paymentservice-5b96ccd67-xm9pk           1/1     Running   0              2d4h
pod/opentelemetry-demo-productcatalogservice-599b4fb599-fm8p4   1/1     Running   0              2d4h
pod/opentelemetry-demo-prometheus-server-7bc6fcff5d-t8gzt       1/1     Running   0              2d4h
pod/opentelemetry-demo-quoteservice-5c84df5788-pvh6z            1/1     Running   0              2d4h
pod/opentelemetry-demo-recommendationservice-c69cfb8bb-lk5d5    1/1     Running   0              2d4h
pod/opentelemetry-demo-shippingservice-7f566f9745-m8x67         1/1     Running   0              2d4h
pod/opentelemetry-demo-valkey-7cd96996c-rp8z2                   1/1     Running   0              2d4h
pod/otel-demo-opensearch-0                                      1/1     Running   0              2d4h

NAME                                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                                                                    AGE
service/opentelemetry-demo-adservice          ClusterIP   10.100.200.205   <none>        8080/TCP                                                                   2d4h
service/opentelemetry-demo-cartservice        ClusterIP   10.100.100.248   <none>        8080/TCP                                                                   2d4h
service/opentelemetry-demo-checkoutservice    ClusterIP   10.100.1.153     <none>        8080/TCP                                                                   2d4h
service/opentelemetry-demo-currencyservice    ClusterIP   10.100.197.152   <none>        8080/TCP                                                                   2d4h
service/opentelemetry-demo-emailservice       ClusterIP   10.100.78.70     <none>        8080/TCP                                                                   2d4h
service/opentelemetry-demo-flagd              ClusterIP   10.100.43.20     <none>        8013/TCP,4000/TCP                                                          2d4h
service/opentelemetry-demo-frontend           ClusterIP   10.100.59.121    <none>        8080/TCP                                                                   2d4h
service/opentelemetry-demo-frontendproxy      ClusterIP   10.100.195.145   <none>        8080/TCP                                                                   2d4h
service/opentelemetry-demo-grafana            ClusterIP   10.100.190.88    <none>        80/TCP                                                                     2d4h
service/opentelemetry-demo-imageprovider      ClusterIP   10.100.42.195    <none>        8081/TCP                                                                   2d4h
service/opentelemetry-demo-jaeger-agent       ClusterIP   None             <none>        5775/UDP,5778/TCP,6831/UDP,6832/UDP                                        2d4h
service/opentelemetry-demo-jaeger-collector   ClusterIP   None             <none>        9411/TCP,14250/TCP,14267/TCP,14268/TCP,4317/TCP,4318/TCP                   2d4h
service/opentelemetry-demo-jaeger-query       ClusterIP   None             <none>        16686/TCP,16685/TCP                                                        2d4h
service/opentelemetry-demo-kafka              ClusterIP   10.100.129.39    <none>        9092/TCP,9093/TCP                                                          2d4h
service/opentelemetry-demo-loadgenerator      ClusterIP   10.100.209.42    <none>        8089/TCP                                                                   2d4h
service/opentelemetry-demo-otelcol            ClusterIP   10.100.126.150   <none>        6831/UDP,14250/TCP,14268/TCP,8888/TCP,4317/TCP,4318/TCP,9464/TCP,9411/TCP  2d4h
service/opentelemetry-demo-paymentservice     ClusterIP   10.100.30.216    <none>        8080/TCP                                                                   2d4h
service/opentelemetry-demo-productcatalogservice   ClusterIP   10.100.23.33     <none>        8080/TCP                                                              2d4h
service/opentelemetry-demo-prometheus-server  ClusterIP   10.100.199.16    <none>        9090/TCP                                                                   2d4h
service/opentelemetry-demo-quoteservice       ClusterIP   10.100.180.12    <none>        8080/TCP                                                                   2d4h
service/opentelemetry-demo-recommendationservice   ClusterIP   10.100.99.188    <none>        8080/TCP                                                              2d4h
service/opentelemetry-demo-shippingservice    ClusterIP   10.100.54.192    <none>        8080/TCP                                                                   2d4h
service/opentelemetry-demo-valkey             ClusterIP   10.100.76.69     <none>        6379/TCP                                                                   2d4h
service/otel-demo-opensearch                  ClusterIP   10.100.135.126   <none>        9200/TCP,9300/TCP,9600/TCP                                                 2d4h
service/otel-demo-opensearch-headless         ClusterIP   None             <none>        9200/TCP,9300/TCP,9600/TCP                                                 2d4h

NAME                                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/opentelemetry-demo-accountingservice      1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-adservice              1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-cartservice            1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-checkoutservice        1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-currencyservice        1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-emailservice           1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-flagd                  1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-frauddetectionservice  1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-frontend               1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-frontendproxy          1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-grafana                1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-imageprovider          1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-jaeger                 1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-kafka                  1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-loadgenerator          1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-otelcol                1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-paymentservice         1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-productcatalogservice  1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-prometheus-server      1/1     1            1           2d4h
deployment.apps/opentelemetry-demo-quoteservice           1/1     1            1           2d4h
```

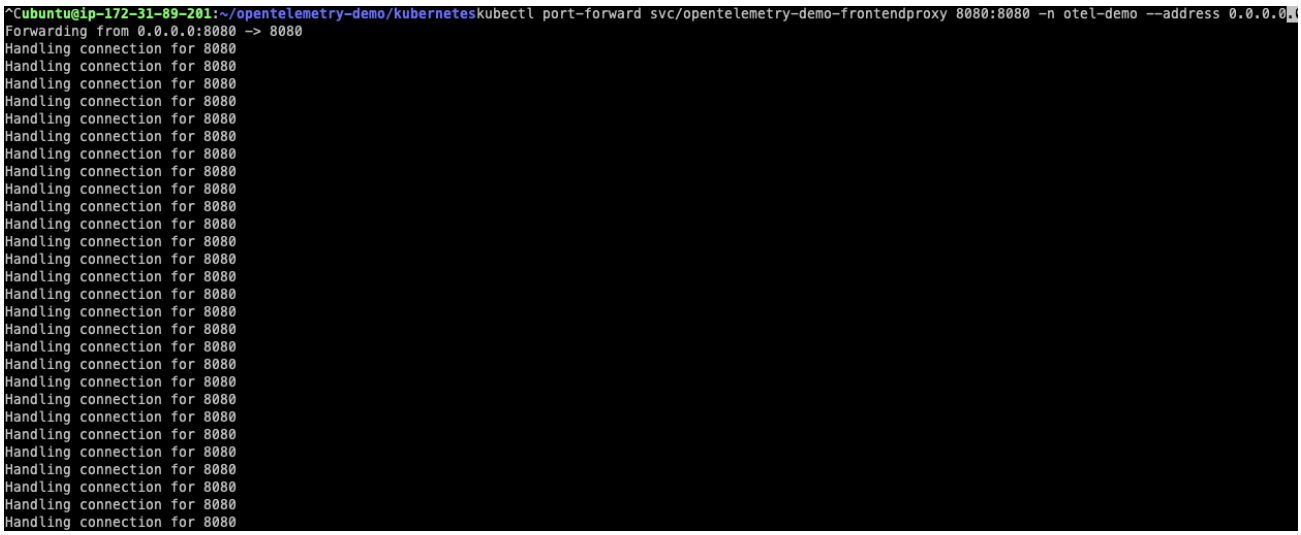
Let's check the logs of opentelemetry-demo-frontendproxy pod:

```
Unset
kubectl logs opentelemetry-demo-frontendproxy-85cb75ff-w9f78 -n otel-demo
```



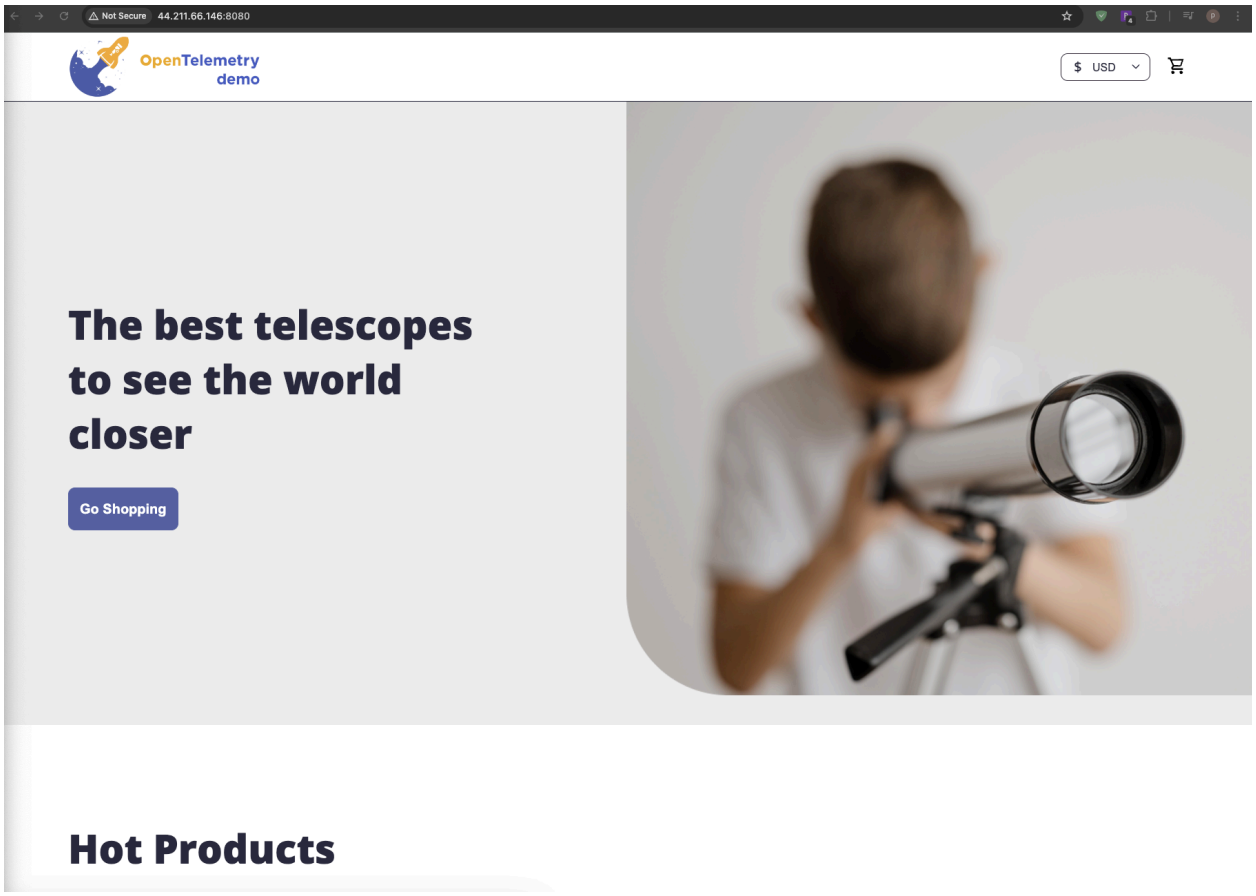
5. The following command allows external users or tools to access the opentelemetry-demo-frontendproxy service on port 8080 of the local machine. The forwarded traffic is routed to port 8080 of the service running in the otel-demo namespace.

```
Unset
kubectl port-forward svc/opentelemetry-demo-frontendproxy 8080:8080 -n otel-demo --address 0.0.0.0
```

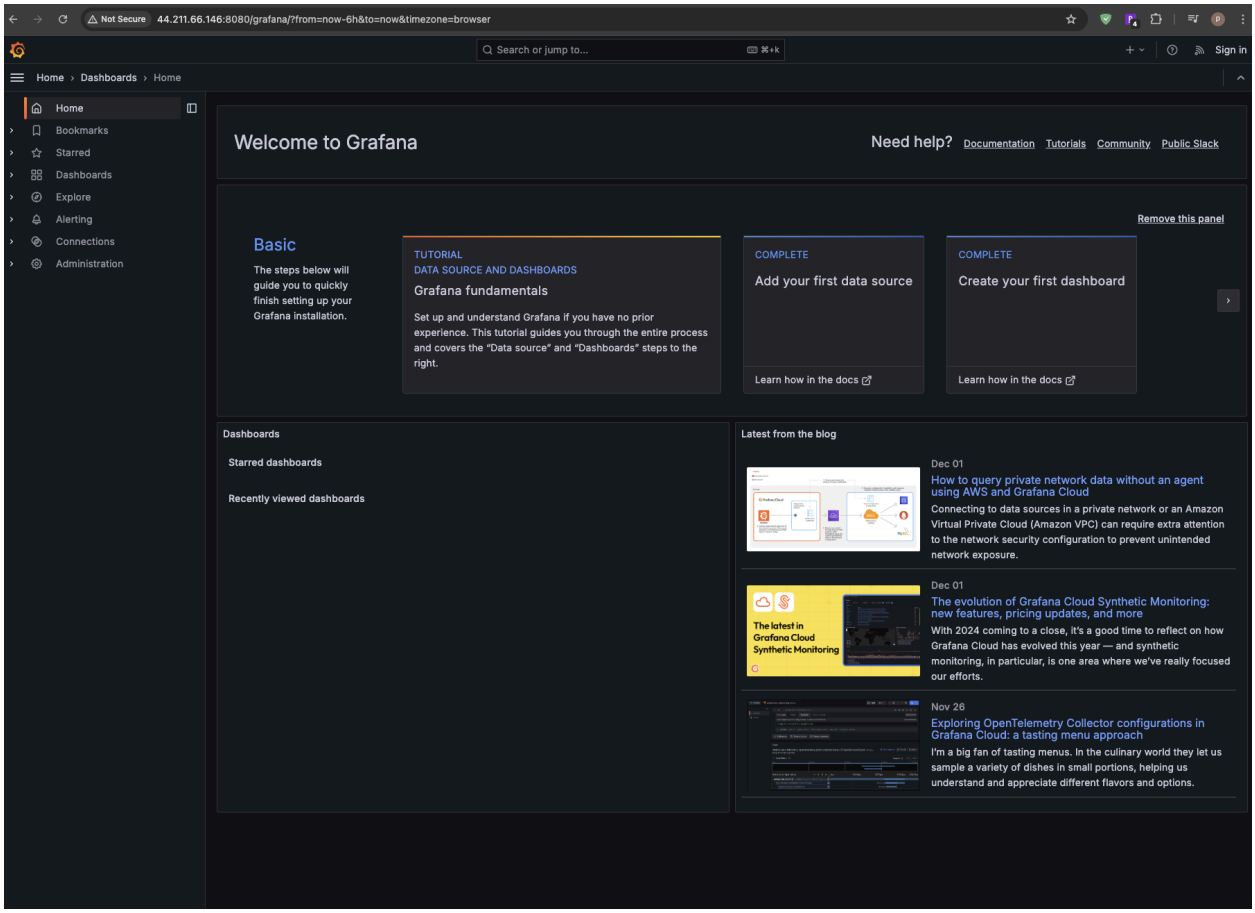


Now, the service will be available via *http://44.211.66.146:8080.* Let's access the applications:

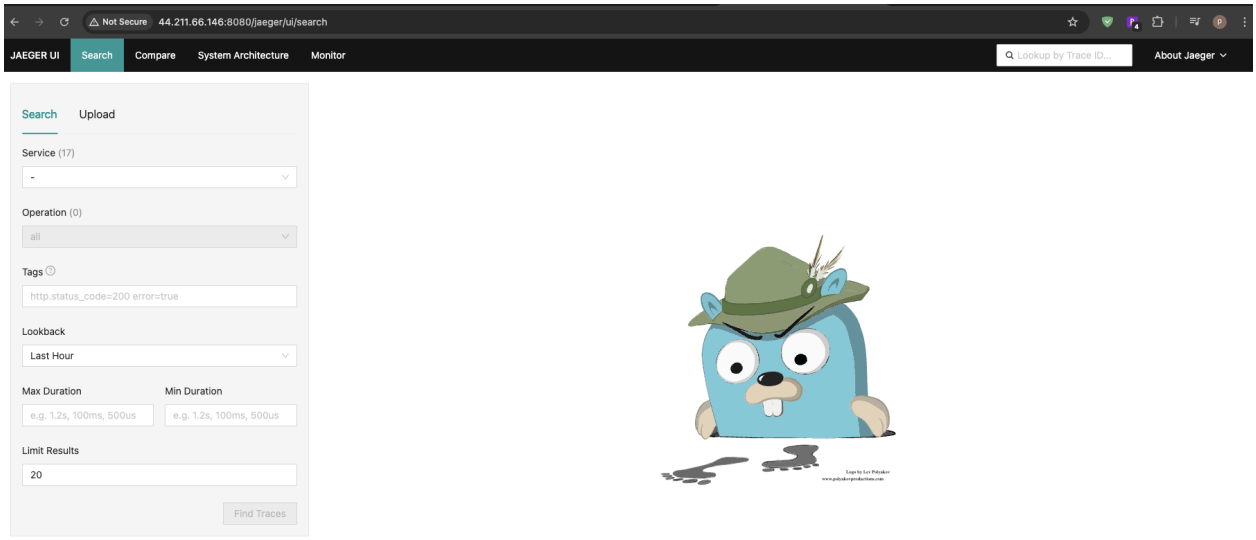
- **Web Store** (*http://44.211.66.146:8080*)



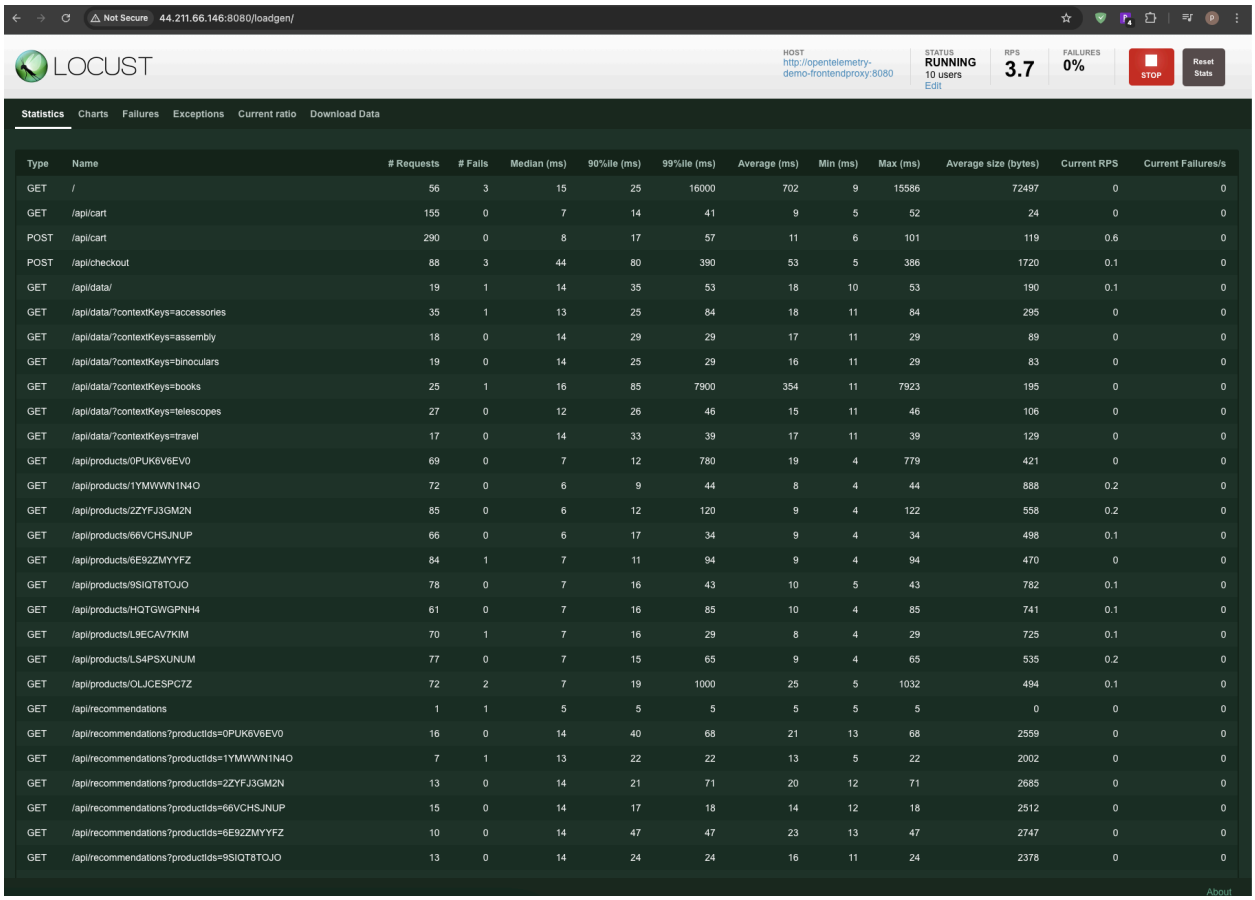
- **Grafana** (*http://44.211.66.146:8080/grafana*)



- **Jaeger UI** (*http://44.211.66.146:8080/jaeger/ui/*)

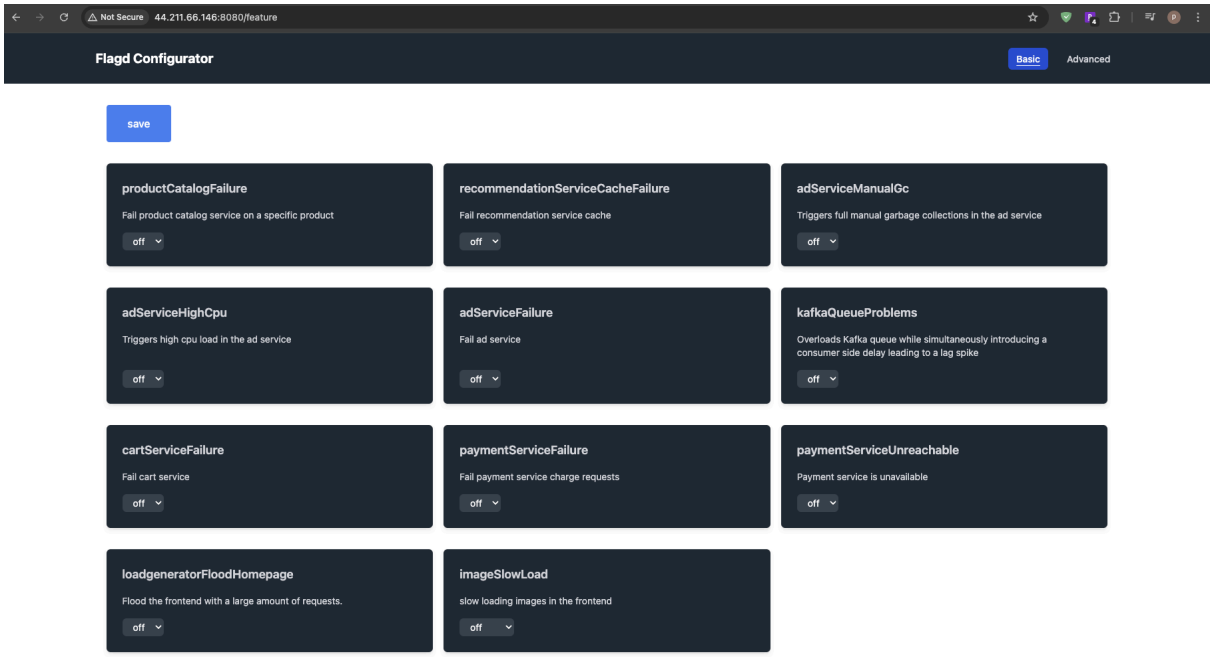


- **Load Generator UI** (*http://44.211.66.146:8080/loadgen/*)



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	56	3	15	25	16000	702	9	15586	72497	0	0
GET	/api/cart	155	0	7	14	41	9	5	52	24	0	0
POST	/api/cart	290	0	8	17	57	11	6	101	119	0.6	0
POST	/api/checkout	88	3	44	80	390	53	5	386	1720	0.1	0
GET	/api/data/	19	1	14	35	53	18	10	53	190	0.1	0
GET	/api/data/?contextKeys=accessories	35	1	13	25	84	18	11	84	295	0	0
GET	/api/data/?contextKeys=assembly	18	0	14	29	29	17	11	29	89	0	0
GET	/api/data/?contextKeys=binoculars	19	0	14	25	29	16	11	29	83	0	0
GET	/api/data/?contextKeys=books	25	1	16	85	7900	354	11	7923	195	0	0
GET	/api/data/?contextKeys=telescopes	27	0	12	26	46	15	11	46	106	0	0
GET	/api/data/?contextKeys=travel	17	0	14	33	39	17	11	39	129	0	0
GET	/api/products/0PUK6V6EV0	69	0	7	12	780	19	4	779	421	0	0
GET	/api/products/1YMWWN1N4O	72	0	6	9	44	8	4	44	888	0.2	0
GET	/api/products/2ZYFJ3GM2N	85	0	6	12	120	9	4	122	558	0.2	0
GET	/api/products/66VCHSJNUP	66	0	6	17	34	9	4	34	498	0.1	0
GET	/api/products/6E92ZMYYFZ	84	1	7	11	94	9	4	94	470	0	0
GET	/api/products/9SIQT8TOJO	78	0	7	16	43	10	5	43	782	0.1	0
GET	/api/products/HQTGWGPNH4	61	0	7	16	85	10	4	85	741	0.1	0
GET	/api/products/L9ECAV7KIM	70	1	7	16	29	8	4	29	725	0.1	0
GET	/api/products/LS4PSXUNUM	77	0	7	15	65	9	4	65	535	0.2	0
GET	/api/products/OLJCESPC7Z	72	2	7	19	1000	25	5	1032	494	0.1	0
GET	/api/recommendations	1	1	5	5	5	5	5	5	0	0	0
GET	/api/recommendations?productIds=0PUK6V6EV0	16	0	14	40	68	21	13	68	2559	0	0
GET	/api/recommendations?productIds=1YMWWN1N4O	7	1	13	22	22	13	5	22	2002	0	0
GET	/api/recommendations?productIds=2ZYFJ3GM2N	13	0	14	21	71	20	12	71	2685	0	0
GET	/api/recommendations?productIds=66VCHSJNUP	15	0	14	17	18	14	12	18	2512	0	0
GET	/api/recommendations?productIds=6E92ZMYYFZ	10	0	14	47	47	23	13	47	2747	0	0
GET	/api/recommendations?productIds=9SIQT8TOJO	13	0	14	24	24	16	11	24	2378	0	0

- **Flagd Configurator UI**(*http://44.211.66.146:8080/feature*)



Conclusion: Similarly, for the Kubernetes setup, we deployed the application to the EKS cluster, where all pods and services ran seamlessly in the designated namespace. Our validation process, which included verifying pod and node statuses and testing the application endpoints, confirmed a robust and fully functional deployment environment.

YAML SPLITTING AND MODULAR DEPLOYMENT

Objective: The aim is to split the deployment yaml file into modular components so that we can improve the manageability and scalability of the deployment. Thereby making it easier to troubleshoot and update individual services without affecting the entire deployment.

In this phase, we will organize the `*opentelemetry-demo.yaml*` by resource type, like ConfigMaps, Deployments, Secrets and Services. By applying each YAML file recursively, we ensure that the deployment process is more efficient and flexible to fine tune for each service. This allows for pinpointing errors and ensuring minimal disruption to the overall application.

Through this approach, we aim to streamline the deployment process, address fault tolerance, and set the foundation for an organized application management strategy in Kubernetes.

Implementation:

We will split the `*opentelemetry-demo.yaml*` file into smaller and service specific YAML files to improve the manageability and scalability. Each resource, such as Deployments, Services, ConfigMaps, and Secrets, has been separated into individual YAML files. These files are then stored in a dedicated folder, ensuring a clear organizational structure and easy access to each service file.

1. **Folder Structure**
   We created a new folder structure to organize the YAML files with the following structure.

```
Unset
/opentelemetry-demo/
    ├── /kubernetes/
        ├── deployment/
            ├── opentelemetry-demo-frontend.yaml
            ├── opentelemetry-demo-loadgen.yaml
            ├── opentelemetry-demo-jaeger.yaml
            └── opentelemetry-demo-grafana.yaml ...
        ├── services/
            ├── opentelemetry-demo-frontend-service.yaml
            ├── opentelemetry-demo-backend-service.yaml
            └── opentelemetry-demo-loadgen-service.yaml ...
        ├── configmaps/
            └── opentelemetry-demo-app-configmap.yaml ...
        ├── secrets/
            └── opentelemetry-demo-app-secret.yaml ...
...
```

2. **Splitting YAML using Python Script**
   We wrote the script to automatically split the main demo YAML into individual files and stored in a proper folder structure

Python

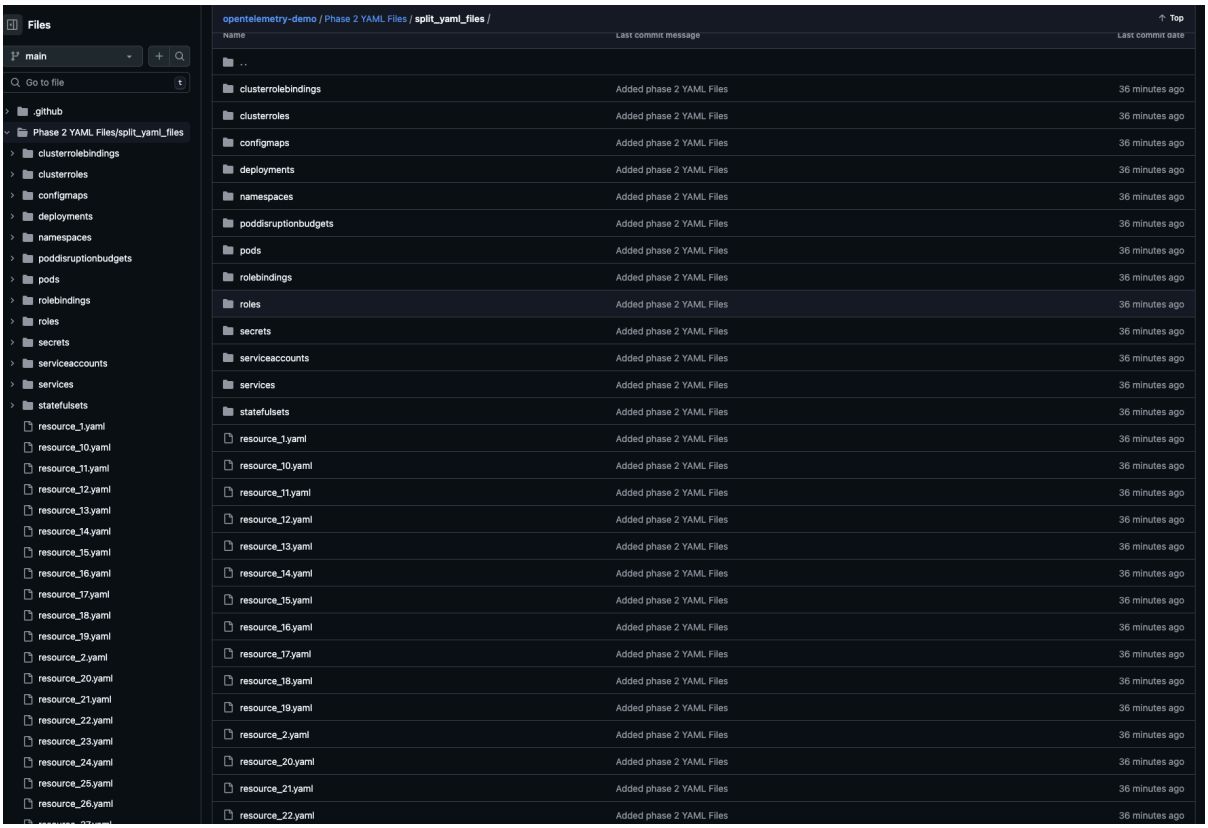
```python
import yaml
import os

# Read the input YAML file
input_file = "opentelemetry-demo.yaml"
output_dir = "Splitted YAMLs"

# Create the output directory if it doesn't exist
os.makedirs(output_dir, exist_ok=True)

with open(input_file, 'r') as file:
    # Load all YAML documents using yaml.safe_load_all
    # This handles multiple YAML documents in a single file correctly
    for parsed_doc in yaml.safe_load_all(file):
        if parsed_doc:  # Check if the document is not None
            try:
                # Extract the name from the metadata section
                name = parsed_doc.get('metadata', {}).get('name', None)

                # Extract the kind from the kind section to determine the
subfolder
                kind = parsed_doc.get('kind', None)

                if name and kind:
                    # Define the subfolder based on the kind (you can modify
this logic as needed)
                    subfolder = os.path.join(output_dir, kind.lower())
                    os.makedirs(subfolder, exist_ok=True)  # Create the
subfolder if it doesn't exist

                    # Define the output filename
                    output_file = os.path.join(subfolder, f'{name}.yaml')

                    # Write the document to a new file
                    with open(output_file, 'w') as outfile:
                        yaml.dump(parsed_doc, outfile,
default_flow_style=False)
                    print(f'Saved: {output_file}')
```

```
                else:
                    print("No 'metadata.name' or 'kind' found in document,
skipping.")
            except yaml.YAMLError as e:
                print(f"Error processing YAML document: {e}")
        else:
            print("Skipping empty document.")
```

This script automatically reads the `*opentelemetry-demo.yaml*` file, extracts each document as they are separated by “---" and saves them into a proper directory structure.

The YAML files, after being split into modular components, are uploaded to a shared GitHub repository. This ensures they are securely stored, making it easier for team members to access,and share the files as needed.

Github Link : https://github.com/pothuguntaumesh/opentelemetry-demo/tree/main/Phase%202%20YAML%20Files/split_yaml_files



- Github Snapshot containing the splitted files

3. **Transfer Files to EC2**
   We copied the splitted modular files generated to the home directory on the EC2 instance

```
Unset
scp -i cc-final.pem -r splitted_yaml_files
ubuntu@ec2-3-81-224-229.compute-1.amazonaws.com:/home/ubuntu/
```

- List of all the splitted YAML files

4. **Recursive Apply of YAML files**
   Kubernetes provides the ability to recursively apply all the yaml files in a directory. This feature enables us to deploy all resources defined in the modular YAML files with a single command.

```
Unset
kubectl apply -f splitted_yaml_files/ --recursive
```

   This command ensures that all components are applied in the correct order without manually applying each file, making the deployment process faster and more efficient.

- Recursive Apply fo YAML files

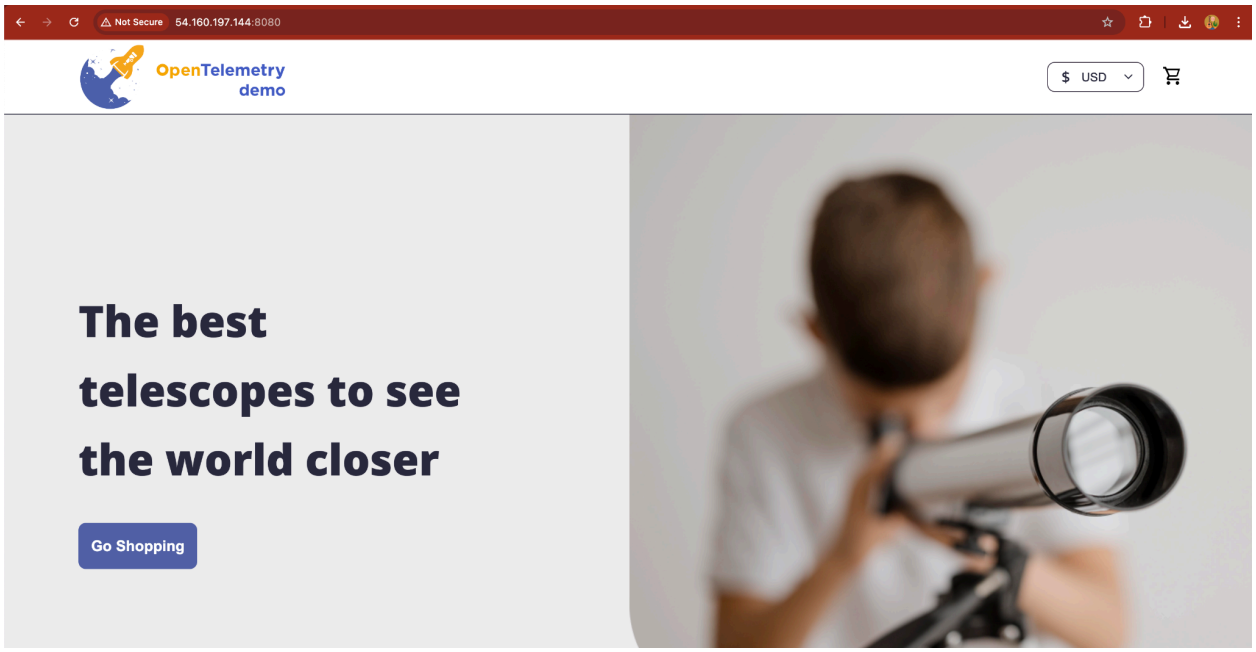
5. **Verification and Testing**
   After the deployment, we verified that all services were running correctly and were exposed correctly for external access.

```
Unset
kubectl get pods -n otel-demo
kubectl get svc -n otel-demo
kubectl get all -n otel-demo
```

```
ubuntu@ip-172-31-89-201:~/opentelemetry-demo/kubernetes$ kubectl get pods -n otel-demo
NAME                                                READY   STATUS              RESTARTS   AGE
opentelemetry-demo-accountingservice-698f94f674-lq9g2       1/1     Running             0          52s
opentelemetry-demo-adservice-5987f8874b-tkb4m               1/1     Running             0          52s
opentelemetry-demo-cartservice-8479bd7459-82p8f             1/1     Running             0          52s
opentelemetry-demo-checkoutservice-b84d4d7cb-56jsv          1/1     Running             0          52s
opentelemetry-demo-currencyservice-7c5899b58b-grhqm         1/1     Running             0          52s
opentelemetry-demo-emailservice-664496796c-bzp9s            1/1     Running             0          52s
opentelemetry-demo-flagd-548bd77698-zbwxv                   2/2     Running             0          52s
opentelemetry-demo-frauddetectionservice-6d896868cc-rwrn6   1/1     Running             0          51s
opentelemetry-demo-frontend-75f6fc6b7-62hcq                 1/1     Running             0          51s
opentelemetry-demo-frontendproxy-85cb75ff-hpr9w             1/1     Running             0          51s
opentelemetry-demo-grafana-7d69c4596c-hplpv                 0/1     Running             0          52s
opentelemetry-demo-imageprovider-6c769996d4-x4zpt           1/1     Running             0          51s
opentelemetry-demo-jaeger-567489f647-jkk5c                  1/1     Running             0          52s
opentelemetry-demo-kafka-76bdbfdc8c-zp4vh                   1/1     Running             0          51s
opentelemetry-demo-loadgenerator-6dd55bd9cb-sg4ld           1/1     Running             0          50s
opentelemetry-demo-otelcol-5cf7d8b476-lztx4                 0/1     ContainerCreating   0          52s
opentelemetry-demo-paymentservice-5b96ccd67-qbmd6           1/1     Running             0          50s
opentelemetry-demo-productcatalogservice-599b4fb599-c5459   1/1     Running             0          50s
opentelemetry-demo-prometheus-server-7bc6fcff5d-64pj7       1/1     Running             0          52s
opentelemetry-demo-quoteservice-5c84df5788-czbzl            1/1     Running             0          50s
opentelemetry-demo-recommendationservice-c69cfb8bb-pg7wz    1/1     Running             0          49s
opentelemetry-demo-shippingservice-7f566f9745-2552s         1/1     Running             0          49s
opentelemetry-demo-valkey-7cd96996c-bpjgn                   1/1     Running             0          49s
otel-demo-opensearch-0                              0/1     Running             0          52s
ubuntu@ip-172-31-89-201:~/opentelemetry-demo/kubernetes$
```

6. **Accessing the Application Endpoints**
   Similar to Phase 1, we verified if the application was accessible through the frontend proxy service. We used the port forward feature to access the application.

```
Unset
kubectl port-forward svc/opentelemetry-demo-frontendproxy 8080:8080 -n
otel-demo --address 0.0.0.0
```



Conclusion: By splitting the Kubernetes deployment YAML into modular files, we improved the maintainability and scalability of the application. The recursive apply method allowed for efficient deployment of all resources, also enabling easier troubleshooting, updates, and scaling of individual services. The deployment was successfully validated, and all endpoints were tested and verified for functionality.

INTEGRATING HELM FOR DEPLOYMENT

Objective: The goal of this task was to streamline the deployment and management of Kubernetes resources by using Helm, a robust package manager designed specifically for Kubernetes environments. Helm simplifies the process by enabling the use of pre-configured charts that define application resources and configurations, ensuring consistency across deployments. It automates the installation and upgrading of complex applications, reducing the risk of errors and manual intervention. With Helm, rolling back to a previous stable version is straightforward, providing resilience in case of faulty updates. Additionally, Helm allows for easy customizations, enabling flexibility to adapt deployments based on specific requirements. This approach significantly improves operational efficiency and management of Kubernetes workloads.

Implementation:

Used the provided Helm Chart

1. The deployment leveraged a pre-configured Helm chart from the OpenTelemetry Helm repository, which contained all the necessary configurations to deploy the application effectively. This chart included detailed definitions for Kubernetes resources such as pods, services, deployments, and configuration maps, ensuring a consistent and repeatable deployment process. By using this chart, the setup of OpenTelemetry components, such as monitoring and logging tools, was automated, reducing manual configuration errors.

Added the Helm Repository:

```
Unset
helm repo add opentelemetry
https://open-telemetry.github.io/opentelemetry-helm-charts
```

2.To ensure that the most up-to-date and stable version of the Helm chart is used for the application deployment, the Helm repository was updated. This process involved running the helm repo update command to fetch the latest chart versions from the configured Helm repositories.

```
Unset
helm repo update
```

```
ubuntu@ip-172-31-89-201:~$ helm repo add open-telemetry https://open-telemetry.github.io/opentelemetry-helm-charts
helm repo update
"open-telemetry" already exists with the same configuration, skipping
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "open-telemetry" chart repository
...Successfully got an update from the "prometheus-community" chart repository
Update Complete. *Happy Helming!*
ubuntu@ip-172-31-89-201:~$ helm search repo open-telemetry
NAME                                    CHART VERSION   APP VERSION     DESCRIPTION
open-telemetry/opentelemetry-collector  0.110.5         0.114.0         OpenTelemetry Collector Helm chart for Kubernetes
open-telemetry/opentelemetry-demo       0.33.7          1.12.0          opentelemetry demo helm chart
open-telemetry/opentelemetry-ebpf       0.1.4           v0.10.2         OpenTelemetry eBPF Helm chart for Kubernetes
open-telemetry/opentelemetry-kube-stack 0.3.5           0.107.0         OpenTelemetry Quickstart chart for Kubernetes. ...
open-telemetry/opentelemetry-operator   0.75.0          0.114.1         OpenTelemetry Operator Helm chart for Kubernetes
```

Deployed the application Using Helm

1.Helm was used to deploy the application resources in a Kubernetes cluster, simplifying the management of complex configurations. The deployment was executed within a dedicated namespace to ensure proper isolation of application resources from other applications running in the cluster. By utilizing Helm charts, all Kubernetes resources—such as pods, services, and config maps—were deployed together as a single unit, ensuring consistency and reducing manual errors.

Created a Namespace for the application:

```
Unset
kubectl create namespace otel-helm
```

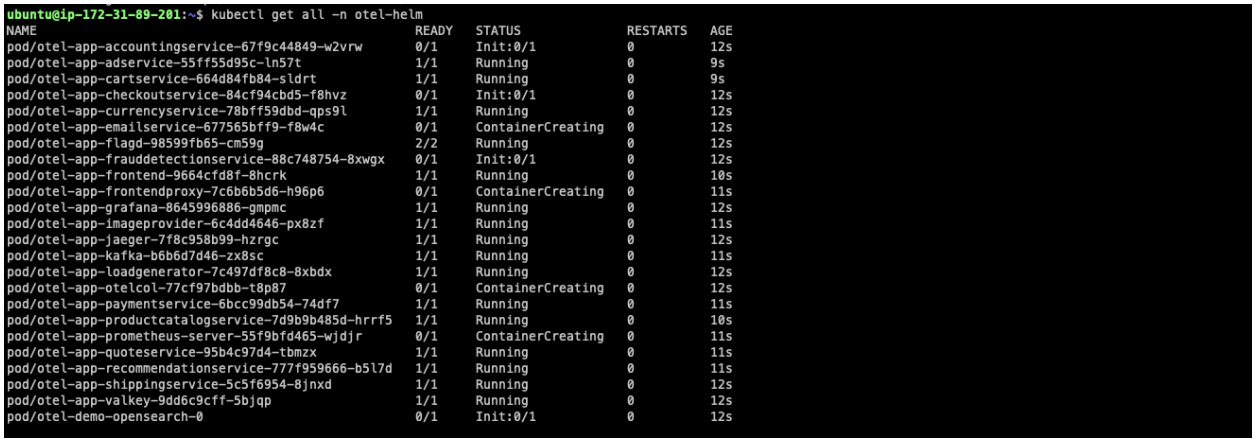
```
ubuntu@ip-172-31-89-201:~$ kubectl create namespace otel-helm
namespace/otel-helm created
```

2. Once the repository was ready, we ran the helm install command to deploy the application to the cluster. This command takes the Helm chart and applies it to the Kubernetes environment, creating all the necessary resources such as pods, deployments, services, and config maps, based on the configurations specified in the chart.

```
Unset
helm install otel-app open-telemetry/opentelemetry-demo -n otel-helm
```

3.After deploying the application using Helm, it was crucial to validate that all Kubernetes resources, including pods, services, and deployments, were created successfully. This was done by using kubectl get all to check the status of resources in the designated namespace. Ensuring that all pods were in the "Running" state and services were correctly exposed confirmed the deployment was successful.

```
Unset
kubectl get all -n otel-helm
```



Upgrade and Rollback

1.Helm's capabilities for upgrading and rolling back deployments were tested to ensure resilience and flexibility in managing the application. Specifically, during the upgrade process, the number of replicas for the application was increased from 1 to 3 to improve scalability and availability. This change was made by modifying the OTEL_SERVICE_NAME.

```yaml
# yaml-language-server: $schema=./values.schema.json
default:
  # List of environment variables applied to all components
  env:
    - name: OTEL_SERVICE_NAME
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: "metadata.labels['app.kubernetes.io/component']"
    - name: OTEL_COLLECTOR_NAME
      value: '{{ include "otel-demo.name" . }}-otelcol'
    - name: OTEL_EXPORTER_OTLP_METRICS_TEMPORALITY_PREFERENCE
      value: cumulative
    - name: OTEL_RESOURCE_ATTRIBUTES
      value: 'service.name=$(OTEL_SERVICE_NAME),service.namespace=opentelemetry-demo,service.version={{ .Chart.AppVersion }}'
  # Allows overriding and additions to .Values.default.env
  envOverrides: []
  #  - name: OTEL_K8S_NODE_NAME
  #    value: "someConstantValue"
  image:
    repository: ghcr.io/open-telemetry/demo
    # Overrides the image tag whose default is the chart appVersion.
    # The service's name will be applied to the end of this value.
    tag: ""
    pullPolicy: IfNotPresent
    pullSecrets: []
  # Default # of replicas for all components
  replicas: 3
  # default revisionHistoryLimit for all components (number of old ReplicaSets to retain)
  revisionHistoryLimit: 10
  # Default schedulingRules for all components
  schedulingRules:
    nodeSelector: {}
    affinity: {}
    tolerations: []
  # Default securityContext for all components
  securityContext: {}

serviceAccount:
  # Specifies whether a service account should be created
  create: true
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname template
  name: ""
```

2.Verified the updated deployment by using the following command.Helm automatically detected the changes in the configuration and managed the creation of new replicas as specified. This process ensured that the application was scaled efficiently without any manual intervention. The number of replicas was increased from 1 to 3, which helped enhance the application's availability and load distribution.

```
Unset
kubectl get pods -n otel-helm
```

```
ubuntu@ip-172-31-89-201:~$ kubectl get pods -n otel-helm
NAME                                          READY   STATUS              RESTARTS   AGE
otel-app-accountingservice-67f9c44849-svbrn   1/1     Running             0          60s
otel-app-accountingservice-67f9c44849-w2vrw   1/1     Running             0          6m16s
otel-app-accountingservice-67f9c44849-wfmvv   1/1     Running             0          60s
otel-app-adservice-55ff55d95c-fkbvt           1/1     Running             0          60s
otel-app-adservice-55ff55d95c-hd2cz           1/1     Running             0          60s
otel-app-adservice-55ff55d95c-ln57t           1/1     Running             0          6m13s
otel-app-cartservice-664d84fb84-c7xxw         0/1     PodInitializing     0          59s
otel-app-cartservice-664d84fb84-rt6vg         0/1     PodInitializing     0          60s
otel-app-cartservice-664d84fb84-sldrt         1/1     Running             0          6m13s
otel-app-checkoutservice-84cf94cbd5-cd96v     1/1     Running             0          60s
otel-app-checkoutservice-84cf94cbd5-f8hvz     1/1     Running             0          6m16s
otel-app-checkoutservice-84cf94cbd5-tzk5v     1/1     Running             0          59s
otel-app-currencyservice-78bff59dbd-qps9l     1/1     Running             0          6m16s
otel-app-currencyservice-78bff59dbd-rfq4r     1/1     Running             0          60s
otel-app-currencyservice-78bff59dbd-wdvq7     1/1     Running             0          59s
otel-app-emailservice-677565bff9-csf8b        1/1     Running             0          59s
otel-app-emailservice-677565bff9-f8w4c        1/1     Running             0          6m16s
otel-app-emailservice-677565bff9-jmbxl        1/1     Running             0          60s
otel-app-flagd-98599fb65-cm59g                2/2     Running             0          6m16s
otel-app-frauddetectionservice-88c748754-8xwgx   1/1     Running             0          6m16s
otel-app-frauddetectionservice-88c748754-tksrh   1/1     Running             0          59s
otel-app-frauddetectionservice-88c748754-zrxjd   1/1     Running             0          59s
otel-app-frontend-9664cfd8f-4ndrx             0/1     ContainerCreating   0          59s
otel-app-frontend-9664cfd8f-8hcrk             1/1     Running             0          6m14s
otel-app-frontend-9664cfd8f-bmcc9             0/1     ContainerCreating   0          59s
otel-app-frontendproxy-7c6b6b5d6-c97r8        1/1     Running             0          59s
otel-app-frontendproxy-7c6b6b5d6-h96p6        1/1     Running             0          6m15s
otel-app-frontendproxy-7c6b6b5d6-jhb6k        1/1     Running             0          59s
otel-app-grafana-8645996886-gmpmc             1/1     Running             0          6m16s
otel-app-imageprovider-6c4dd4646-l5g5t        0/1     ContainerCreating   0          59s
otel-app-imageprovider-6c4dd4646-px8zf        1/1     Running             0          6m15s
otel-app-imageprovider-6c4dd4646-pzsjc        0/1     ContainerCreating   0          59s
otel-app-jaeger-7f8c958b99-hzrgc              1/1     Running             0          6m16s
otel-app-kafka-b6b6d7d46-zx8sc                1/1     Running             0          6m15s
otel-app-loadgenerator-7c497df8c8-2rnq2       1/1     Running             0          59s
otel-app-loadgenerator-7c497df8c8-8xbdx       1/1     Running             0          6m16s
otel-app-loadgenerator-7c497df8c8-ffpkh       1/1     Running             0          59s
otel-app-otelcol-77cf97bdbb-t8p87             1/1     Running             0          6m16s
otel-app-paymentservice-6bcc99db54-74df7      1/1     Running             0          6m15s
otel-app-paymentservice-6bcc99db54-q8nbh      0/1     ContainerCreating   0          58s
otel-app-paymentservice-6bcc99db54-stc8j      0/1     ContainerCreating   0          58s
otel-app-productcatalogservice-7d9b9b485d-7vhvs   0/1     Pending             0          58s
otel-app-productcatalogservice-7d9b9b485d-hrrf5   1/1     Running             0          6m14s
otel-app-productcatalogservice-7d9b9b485d-nlqfz   1/1     Running             0          58s
otel-app-prometheus-server-55f9bfd465-wjdjr   1/1     Running             0          6m15s
otel-app-quoteservice-95b4c97d4-66f8c         0/1     ContainerCreating   0          58s
otel-app-quoteservice-95b4c97d4-b464k         0/1     Pending             0          58s
otel-app-quoteservice-95b4c97d4-tbmzx         1/1     Running             0          6m15s
otel-app-recommendationservice-777f959666-2vbvr   0/1     Pending             0          57s
otel-app-recommendationservice-777f959666-5fmqp   0/1     Pending             0          58s
otel-app-recommendationservice-777f959666-b5l7d   1/1     Running             0          6m15s
otel-app-shippingservice-5c5f6954-8jnxd       1/1     Running             0          6m16s
otel-app-shippingservice-5c5f6954-mvxt6       0/1     Pending             0          57s
otel-app-shippingservice-5c5f6954-v8fzt       0/1     Pending             0          58s
otel-app-valkey-9dd6c9cff-5bjqp               1/1     Running             0          6m16s
otel-demo-opensearch-0                        1/1     Running             0          6m16s
ubuntu@ip-172-31-89-201:~$
```

3.After the upgrade, the application was validated to ensure that all three replicas were running successfully, providing higher availability and load distribution.Helm handled the orchestration of the Kubernetes resources, ensuring that new pods were created, deployed, and integrated into the existing service seamlessly. This automated scaling provided flexibility and improved the overall resilience of the application.

Unset

```
helm history otel-app -n otel-helm
```

```
ubuntu@ip-172-31-89-201:~$ helm history otel-app -n otel-helm
REVISION        UPDATED                         STATUS          CHART                           APP VERSION     DESCRIPTION
1               Sun Dec  8 08:05:47 2024        superseded      opentelemetry-demo-0.33.7       1.12.0          Install complete
2               Sun Dec  8 08:11:01 2024        deployed        opentelemetry-demo-0.33.7       1.12.0          Upgrade complete
ubuntu@ip-172-31-89-201:~$
```

4.To further test Helm's rollback functionality, a scenario was simulated where the update was reversed, restoring the number of replicas back to 1. The rollback was performed using Helm's rollback command, and it was verified that the application reverted to its previous stable state without any downtime or issues.

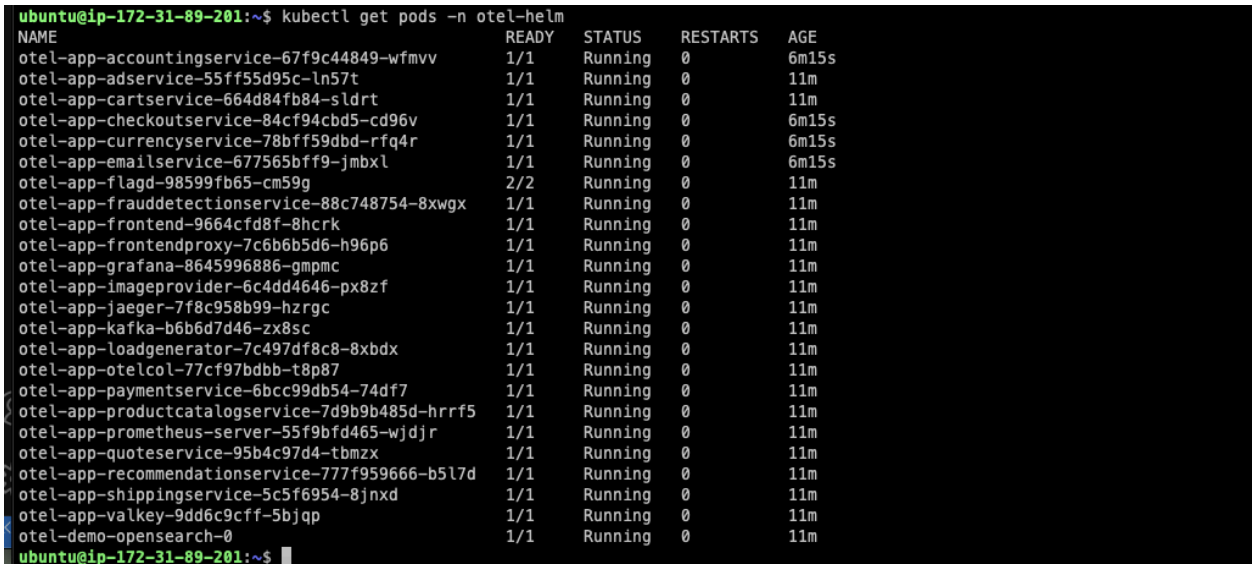
Unset

```
helm rollback otel-app 1 -n otel-helm
```

```
ubuntu@ip-172-31-89-201:~$ helm rollback otel-app 1 -n otel-helm
Rollback was a success! Happy Helming!
ubuntu@ip-172-31-89-201:~$ kubectl get pods -n otel-helm
NAME                                          READY   STATUS        RESTARTS   AGE
otel-app-accountingservice-67f9c44849-svbrn   1/1     Terminating   0          3m31s
otel-app-accountingservice-67f9c44849-w2vrw   1/1     Terminating   0          8m47s
otel-app-accountingservice-67f9c44849-wfmvv   1/1     Running       0          3m31s
otel-app-adservice-55ff55d95c-ln57t           1/1     Running       0          8m44s
otel-app-cartservice-664d84fb84-c7xxw         1/1     Terminating   0          3m30s
otel-app-cartservice-664d84fb84-rt6vg         1/1     Terminating   0          3m31s
otel-app-cartservice-664d84fb84-sldrt         1/1     Running       0          8m44s
otel-app-checkoutservice-84cf94cbd5-cd96v     1/1     Running       0          3m31s
otel-app-currencyservice-78bff59dbd-qps9l     1/1     Terminating   0          8m47s
otel-app-currencyservice-78bff59dbd-rfq4r     1/1     Running       0          3m31s
otel-app-currencyservice-78bff59dbd-wdvq7     1/1     Terminating   0          3m30s
otel-app-emailservice-677565bff9-csf8b        1/1     Terminating   0          3m30s
otel-app-emailservice-677565bff9-f8w4c        1/1     Terminating   0          8m47s
otel-app-emailservice-677565bff9-jmbxl        1/1     Running       0          3m31s
otel-app-flagd-98599fb65-cm59g                2/2     Running       0          8m47s
otel-app-frauddetectionservice-88c748754-8xwgx  1/1   Running       0          8m47s
otel-app-frauddetectionservice-88c748754-tksrh  1/1   Terminating   0          3m30s
otel-app-frauddetectionservice-88c748754-zrxjd  1/1   Terminating   0          3m30s
otel-app-frontend-9664cfd8f-4ndrx             0/1     Terminating   0          3m30s
otel-app-frontend-9664cfd8f-8hcrk             1/1     Running       0          8m45s
otel-app-frontend-9664cfd8f-bmcc9             0/1     Terminating   0          3m30s
otel-app-frontendproxy-7c6b6b5d6-c97r8        1/1     Terminating   0          3m30s
otel-app-frontendproxy-7c6b6b5d6-h96p6        1/1     Running       0          8m46s
otel-app-frontendproxy-7c6b6b5d6-jhb6k        1/1     Terminating   0          3m30s
otel-app-grafana-8645996886-gmpmc             1/1     Running       0          8m47s
otel-app-imageprovider-6c4dd4646-l5g5t        1/1     Terminating   0          3m30s
otel-app-imageprovider-6c4dd4646-px8zf        1/1     Running       0          8m46s
otel-app-imageprovider-6c4dd4646-pzsjc        1/1     Terminating   0          3m30s
otel-app-jaeger-7f8c958b99-hzrgc              1/1     Running       0          8m47s
otel-app-kafka-b6b6d7d46-zx8sc                1/1     Running       0          8m46s
otel-app-loadgenerator-7c497df8c8-2rnq2       1/1     Terminating   0          3m30s
otel-app-loadgenerator-7c497df8c8-8xbdx       1/1     Running       0          8m47s
otel-app-loadgenerator-7c497df8c8-ffpkh       1/1     Terminating   0          3m30s
otel-app-otelcol-77cf97bdbb-t8p87             1/1     Running       0          8m47s
otel-app-paymentservice-6bcc99db54-74df7      1/1     Running       0          8m46s
otel-app-paymentservice-6bcc99db54-q8nbh      1/1     Terminating   0          3m29s
otel-app-paymentservice-6bcc99db54-stc8j      1/1     Terminating   0          3m29s
otel-app-productcatalogservice-7d9b9b485d-hrrf5  1/1  Running       0          8m45s
otel-app-productcatalogservice-7d9b9b485d-nlqfz  1/1  Terminating   0          3m29s
otel-app-prometheus-server-55f9bfd465-wjdjr   1/1     Running       0          8m46s
otel-app-quoteservice-95b4c97d4-66f8c         0/1     Terminating   0          3m29s
otel-app-quoteservice-95b4c97d4-tbmzx         1/1     Running       0          8m46s
otel-app-recommendationservice-777f959666-b5l7d  1/1  Running       0          8m46s
otel-app-shippingservice-5c5f6954-8jnxd       1/1     Running       0          8m47s
otel-app-valkey-9dd6c9cff-5bjqp               1/1     Running       0          8m47s
otel-demo-opensearch-0                        1/1     Running       0          8m47s
```

5.Rollback was verified that the application reverted to its previous stable state without any downtime or issues. This demonstrated the resilience and flexibility of using Helm to manage updates and rollbacks in a Kubernetes environment, ensuring that the application could be easily scaled and maintained without disruption.

```
Unset
kubectl get pods -n otel-helm
```



```
ubuntu@ip-172-31-89-201:~$ kubectl get pods -n otel-helm
NAME                                          READY   STATUS    RESTARTS   AGE
otel-app-accountingservice-67f9c44849-wfmvv   1/1     Running   0          6m15s
otel-app-adservice-55ff55d95c-ln57t           1/1     Running   0          11m
otel-app-cartservice-664d84fb84-sldrt         1/1     Running   0          11m
otel-app-checkoutservice-84cf94cbd5-cd96v     1/1     Running   0          6m15s
otel-app-currencyservice-78bff59dbd-rfq4r     1/1     Running   0          6m15s
otel-app-emailservice-677565bff9-jmbxl        1/1     Running   0          6m15s
otel-app-flagd-98599fb65-cm59g                2/2     Running   0          11m
otel-app-frauddetectionservice-88c748754-8xwgx  1/1   Running   0          11m
otel-app-frontend-9664cfd8f-8hcrk             1/1     Running   0          11m
otel-app-frontendproxy-7c6b6b5d6-h96p6        1/1     Running   0          11m
otel-app-grafana-8645996886-gmpmc             1/1     Running   0          11m
otel-app-imageprovider-6c4dd4646-px8zf        1/1     Running   0          11m
otel-app-jaeger-7f8c958b99-hzrgc              1/1     Running   0          11m
otel-app-kafka-b6b6d7d46-zx8sc                1/1     Running   0          11m
otel-app-loadgenerator-7c497df8c8-8xbdx       1/1     Running   0          11m
otel-app-otelcol-77cf97bdbb-t8p87             1/1     Running   0          11m
otel-app-paymentservice-6bcc99db54-74df7      1/1     Running   0          11m
otel-app-productcatalogservice-7d9b9b485d-hrrf5  1/1  Running   0          11m
otel-app-prometheus-server-55f9bfd465-wjdjr   1/1     Running   0          11m
otel-app-quoteservice-95b4c97d4-tbmzx         1/1     Running   0          11m
otel-app-recommendationservice-777f959666-b5l7d  1/1  Running   0          11m
otel-app-shippingservice-5c5f6954-8jnxd       1/1     Running   0          11m
otel-app-valkey-9dd6c9cff-5bjqp               1/1     Running   0          11m
otel-demo-opensearch-0                        1/1     Running   0          11m
ubuntu@ip-172-31-89-201:~$
```

Conclusion -

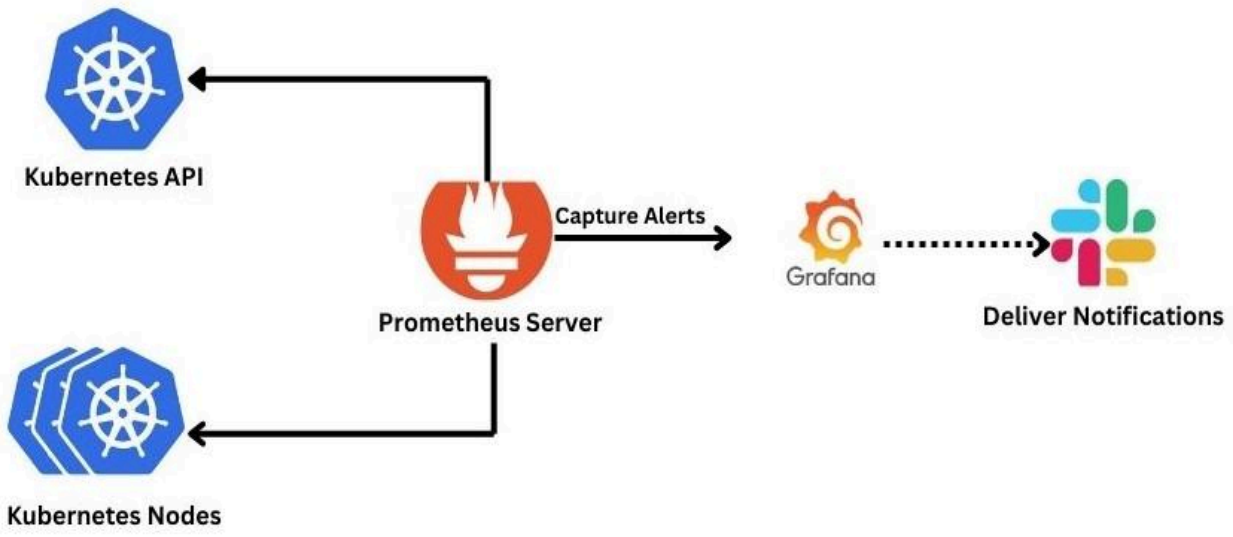
Integrating Helm greatly simplified the deployment and management of Kubernetes resources on AWS EKS, making it easier to handle complex applications in a scalable and efficient manner. By port-forwarding the Prometheus service, we accessed the UI and verified that metrics were being collected and displayed correctly. Helm's package management system allowed for the deployment of pre-configured charts, ensuring consistency across different environments and reducing the potential for human error. The ability to easily upgrade or roll back applications with Helm ensured minimal downtime during updates, as it automatically handles resource changes and version management. Additionally, Helm's templating features enabled customized configurations, making it adaptable to specific needs without modifying underlying infrastructure. The integration of Helm with AWS EKS further enhanced the scalability, resilience, and flexibility of the application, allowing it to grow and adapt quickly. This approach significantly reduced operational overhead, streamlined the management of Kubernetes clusters, and improved overall application stability and reliability.

CUSTOMIZING GRAFANA DASHBOARDS FOR EKS MONITORING

Objective:

We integrated the OpenTelemetry demo with our EKS cluster to enhance observability and monitoring. Using the Prometheus server, we captured essential metrics and alerts, which we visualized through Grafana. This setup enabled us to leverage pre-configured dashboards for monitoring traces, metrics, and logs from the EKS cluster. We then enhanced these dashboards with detailed insights into pod-to-pod network traffic and persistent storage usage, providing a comprehensive view of cluster performance.

For proactive monitoring, we set up alert thresholds for critical metrics including CPU usage, memory consumption, and node availability. We then validated the dashboards' accuracy by generating simulated traffic loads, confirming that they accurately display real-time data and effectively identify potential system bottlenecks.



Below is the step-by-step process of how we implemented the above process:

1. **Leveraging Pre-Built OpenTelemetry Dashboards:**

   The below configuration enables Prometheus to scrape metrics from all services in the otel-demo namespace through the metrics endpoint on the tcp-service port, ensuring observability and monitoring for applications.

```
Unset
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
       name: otel-demo-servicemonitor
       namespace: monitoring
       labels:
              release: prometheus-stack # Match the Prometheus selector in your
setup
spec:
       jobLabel: app.kubernetes.io/name
       namespaceSelector:
              matchNames:
                     - otel-demo # Target the otel-demo namespace
       selector:
              matchLabels: {} # Matches all services in the namespace
       endpoints:
              - port: tcp-service # Ensure this matches the port name in your
services
              interval: 15s
              path: /metrics # Default metrics path
```

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: otel-demo-servicemonitor
  namespace: monitoring
  labels:
    release: prometheus-stack  # Match the Prometheus selector in your setup
spec:
  jobLabel: app.kubernetes.io/name
  namespaceSelector:
    matchNames:
      - otel-demo  # Target the otel-demo namespace
  selector:
    matchLabels: {}  # Matches all services in the namespace
  endpoints:
    - port: tcp-service       # Ensure this matches the port name in your services
      interval: 15s
      path: /metrics          # Default metrics path
```

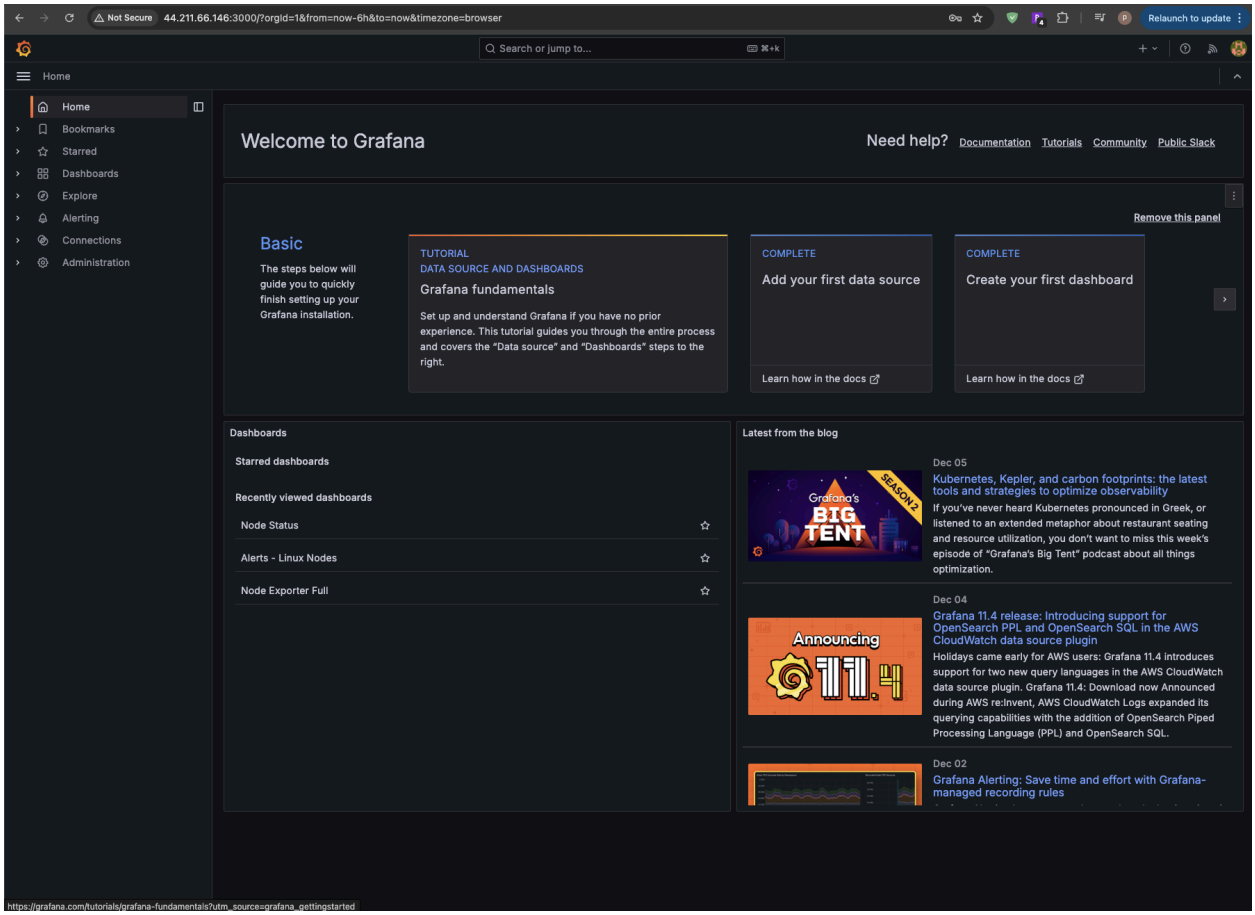
We identified the monitoring services available in our cluster and set up port forwarding to enable external access through a web browser using Grafana.

Unset

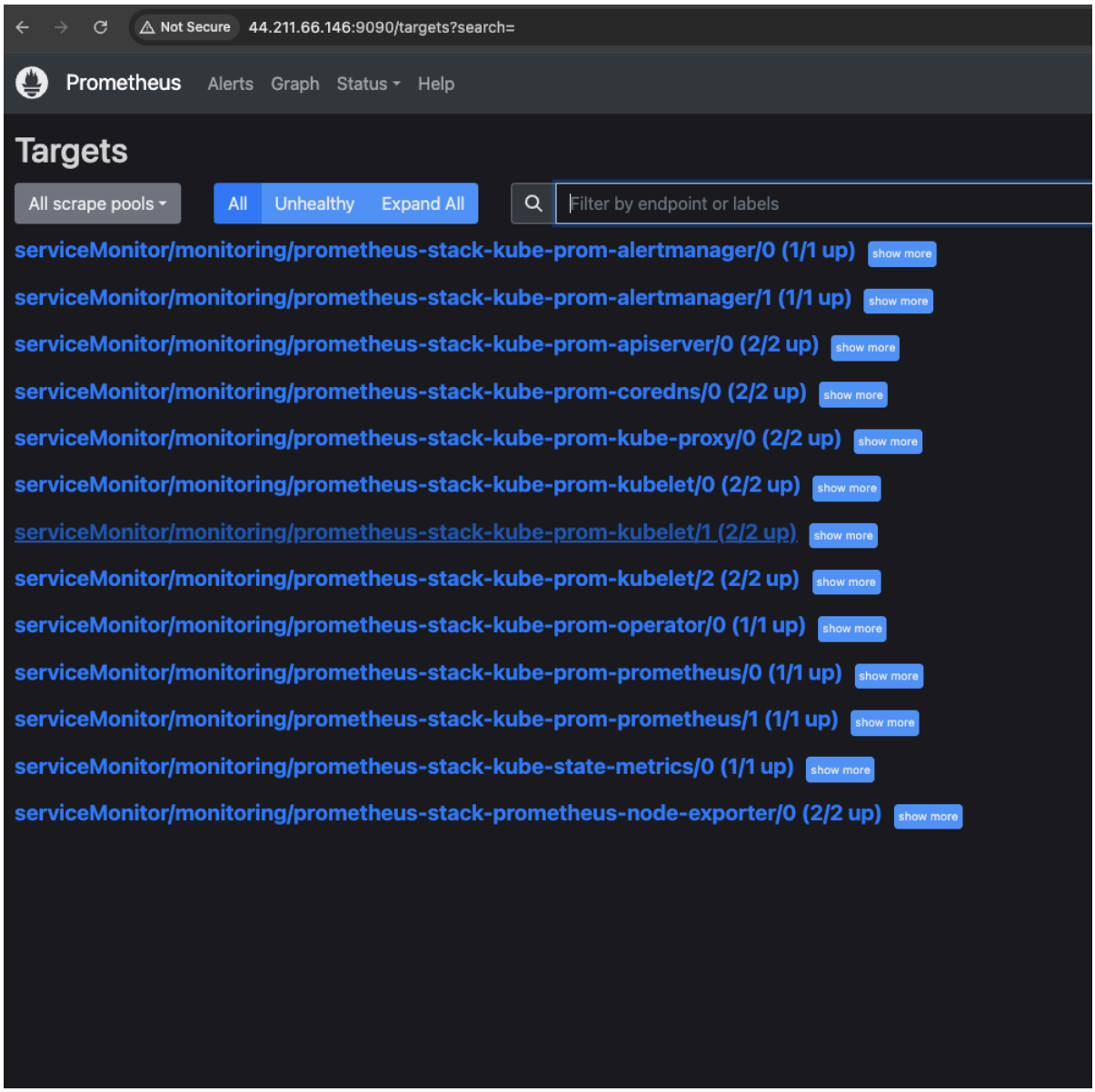
```
kubectl apply -f otel-demo-servicemonitor.yaml
kubectl get servicemonitor -n monitoring
kubectl port-forward svc/prometheus-stack-grafana 3000:80 -n monitoring
--address 0.0.0.0
```

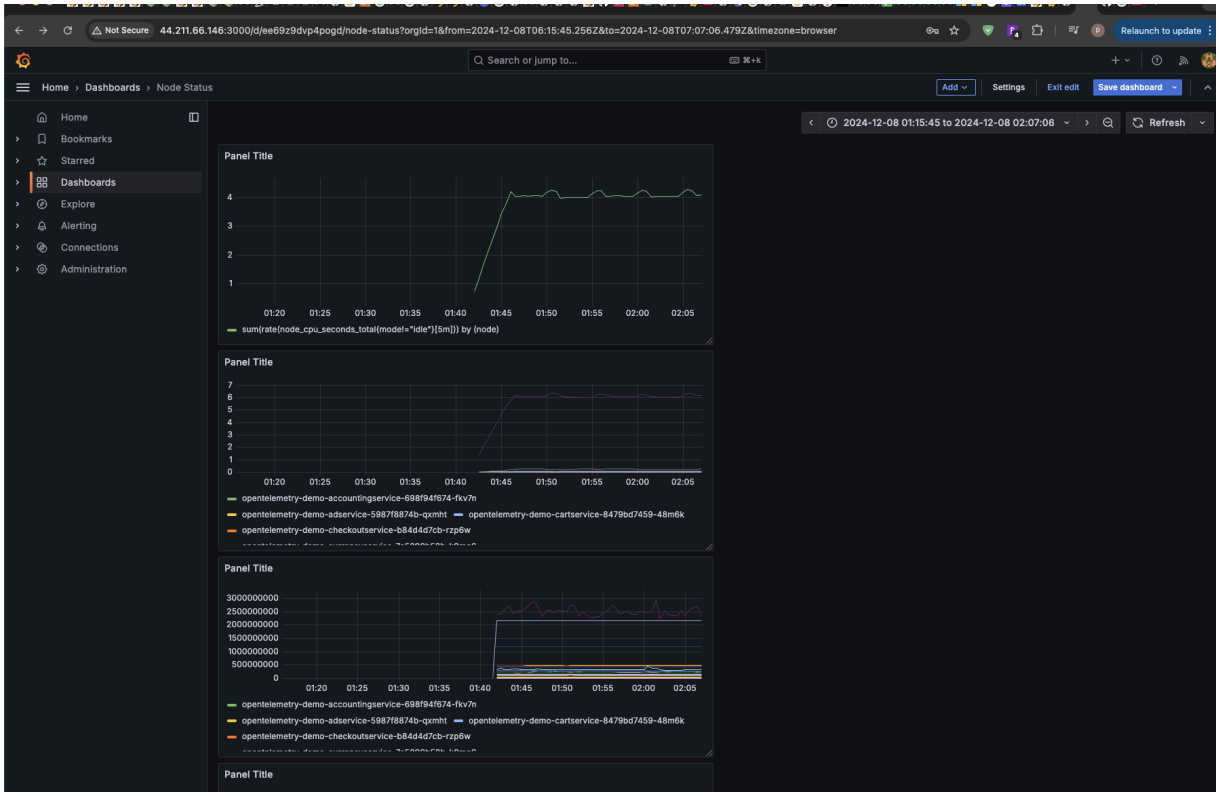
```
ubuntu@ip-172-31-89-201:~$ kubectl get svc -n monitoring
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
alertmanager-operated                     ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP   4h38m
prometheus-operated                       ClusterIP   None             <none>        9090/TCP                     4h38m
prometheus-stack-grafana                  ClusterIP   10.100.187.104   <none>        80/TCP                       4h38m
prometheus-stack-kube-prom-alertmanager   ClusterIP   10.100.67.253    <none>        9093/TCP,8080/TCP            4h38m
prometheus-stack-kube-prom-operator       ClusterIP   10.100.111.29    <none>        443/TCP                      4h38m
prometheus-stack-kube-prom-prometheus     ClusterIP   10.100.196.112   <none>        9090/TCP,8080/TCP            4h38m
prometheus-stack-kube-state-metrics       ClusterIP   10.100.159.94    <none>        8080/TCP                     4h38m
prometheus-stack-prometheus-node-exporter ClusterIP   10.100.6.45      <none>        9100/TCP                     4h38m
ubuntu@ip-172-31-89-201:~$ kubectl port-forward svc/prometheus-stack-grafana 3000:80 -n monitoring
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
^Cubuntu@ip-172-31-89-201:~$ kubectl port-forward svc/prometheus-stack-grafana 3000:80 -n monitoring --address 0.0.0.0
Forwarding from 0.0.0.0:3000 -> 3000
^[[C^[[D^Cubuntu@ip-172-31-89-201:~$ kubectl port-forward svc/prometheus-stack-grafana 3000:80 -n monitoring --address 0.0.0.0
Forwarding from 0.0.0.0:3000 -> 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
```

Below is a glimpse of the Grafana dashboard:

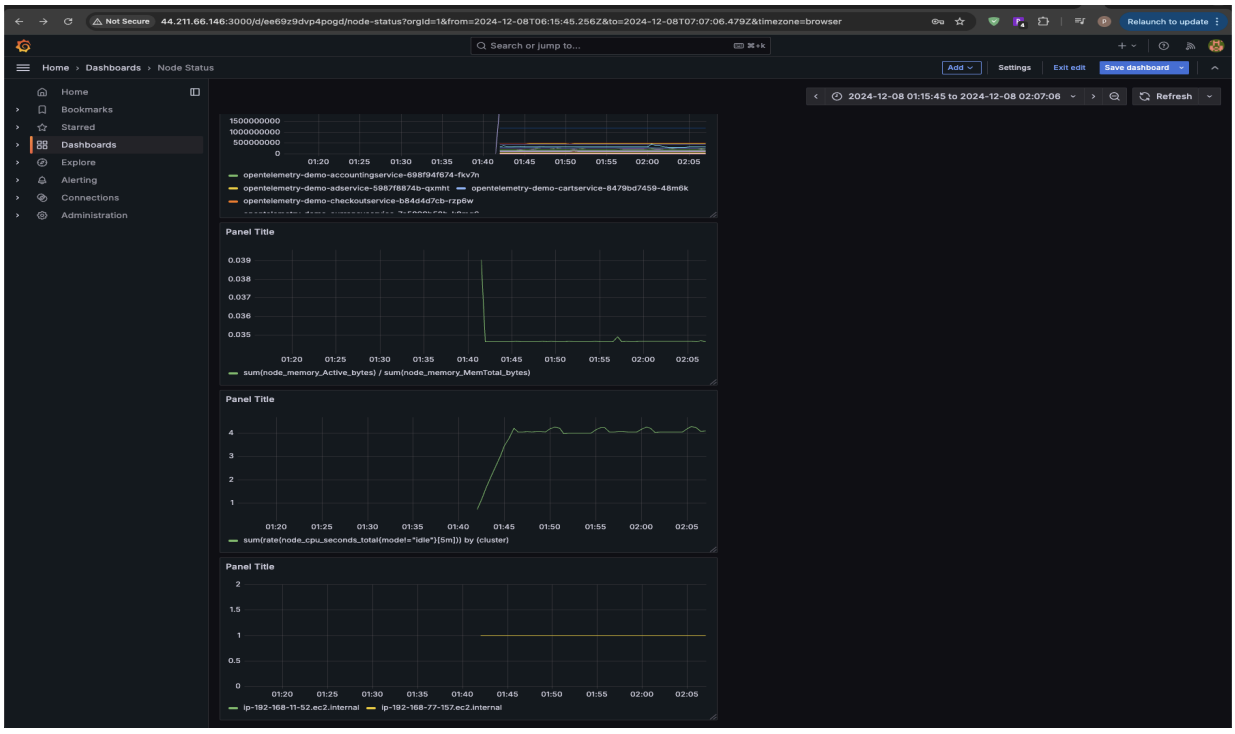


- These are the services that have been scraped by Prometheus from our EKS:





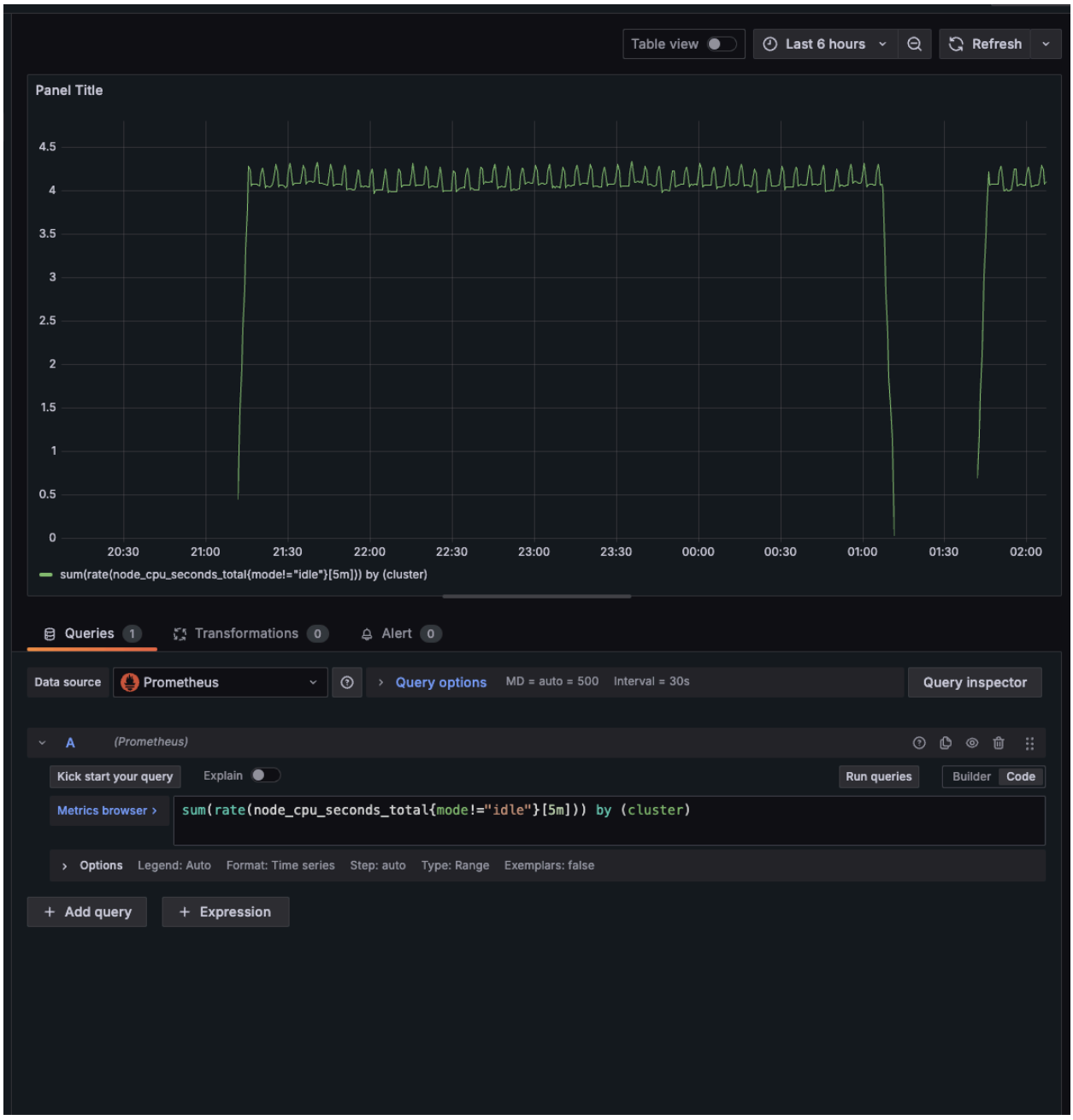
- **Dashboard Overview**



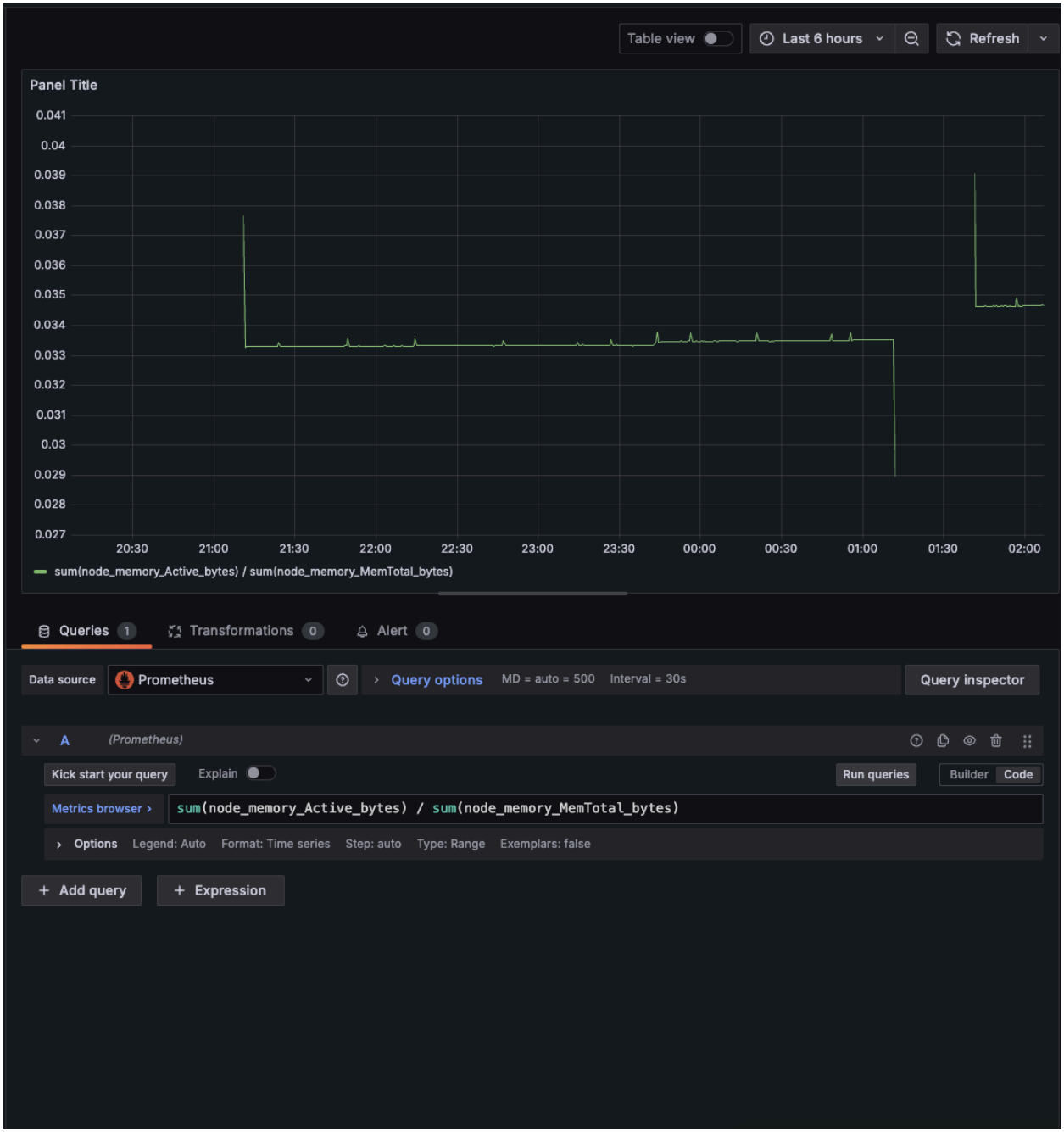
- **Dashboard Overview**

2. Customizing Dashboards for EKS Metrics:

- Node Details:

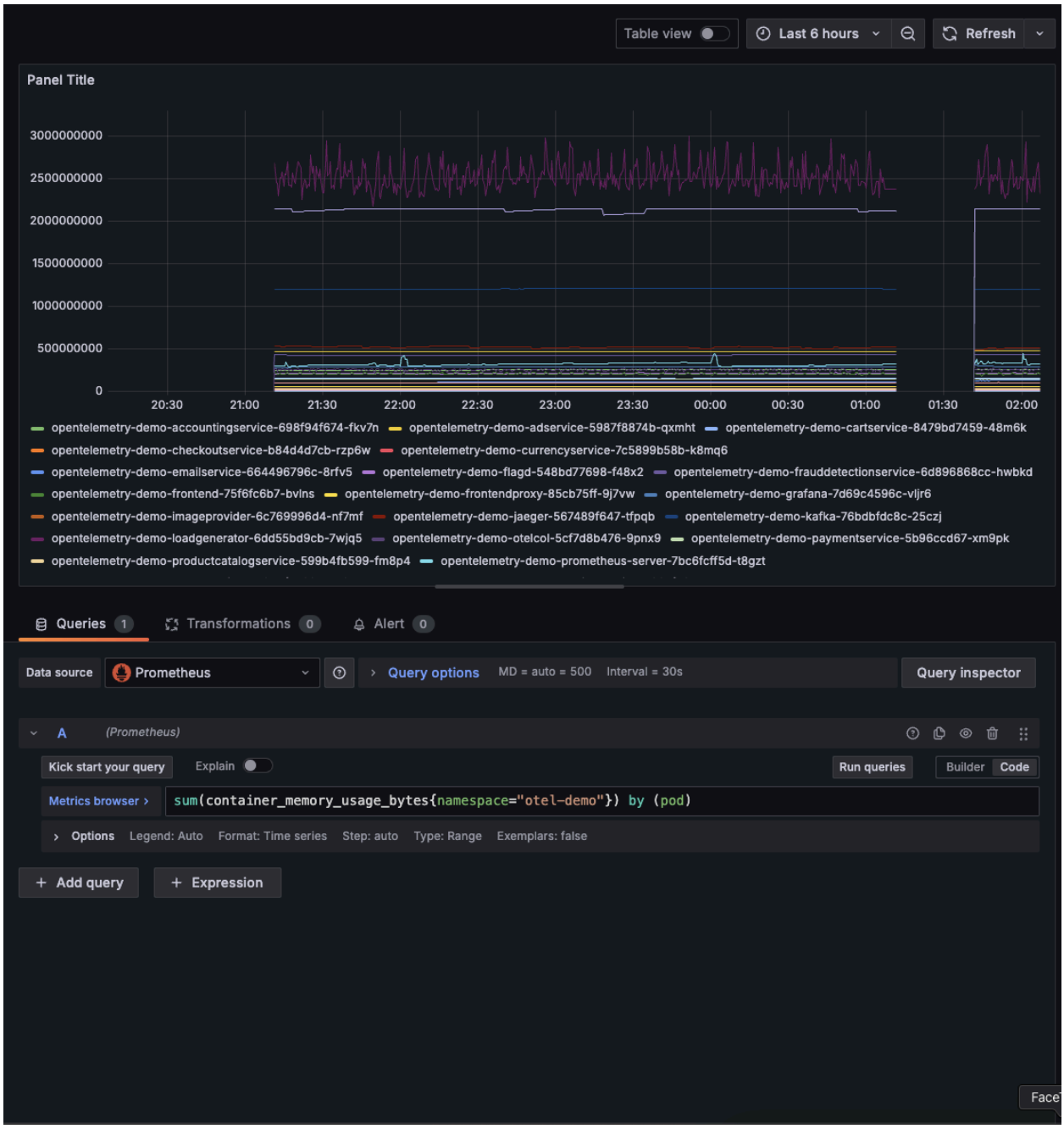


- **Total Amount of CPU Time Spent by the Node**

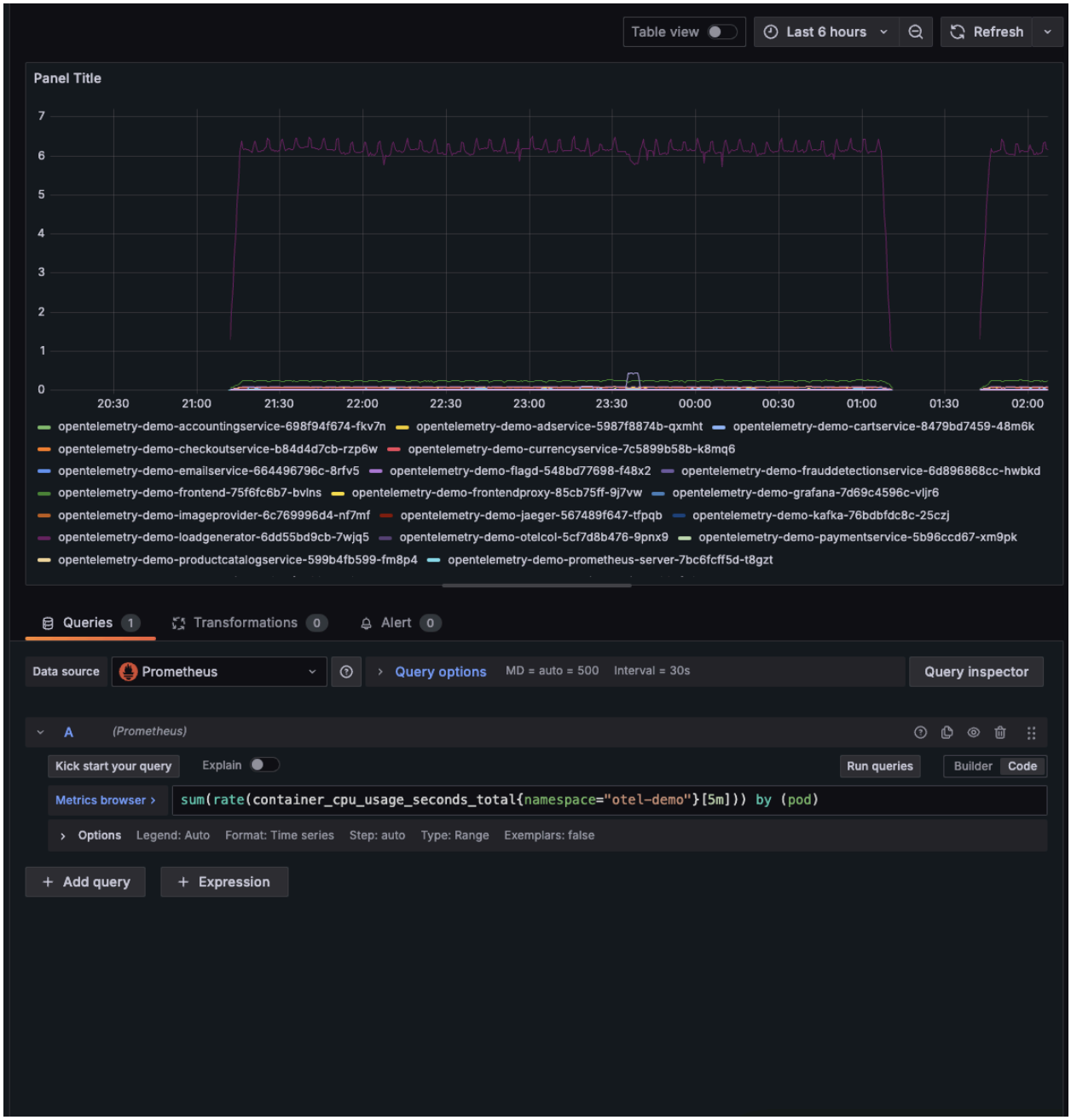


- **Amount of memory actively used by the system**

- **Pod level:**

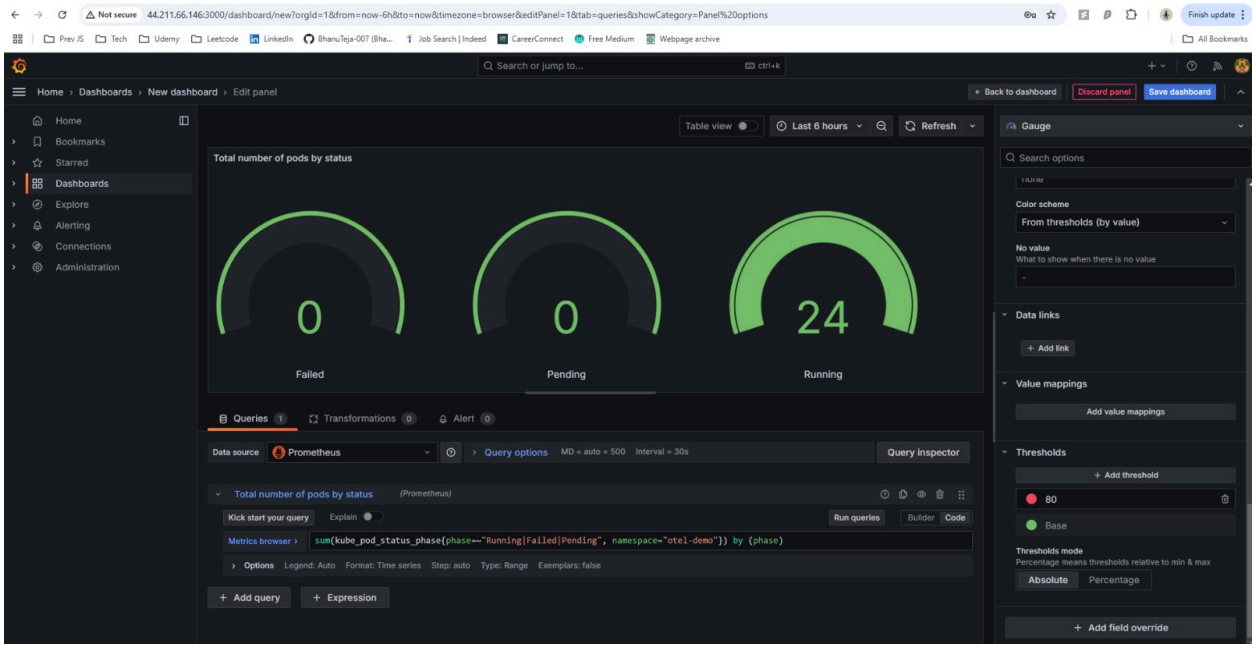


- **Container Memory Consumption**

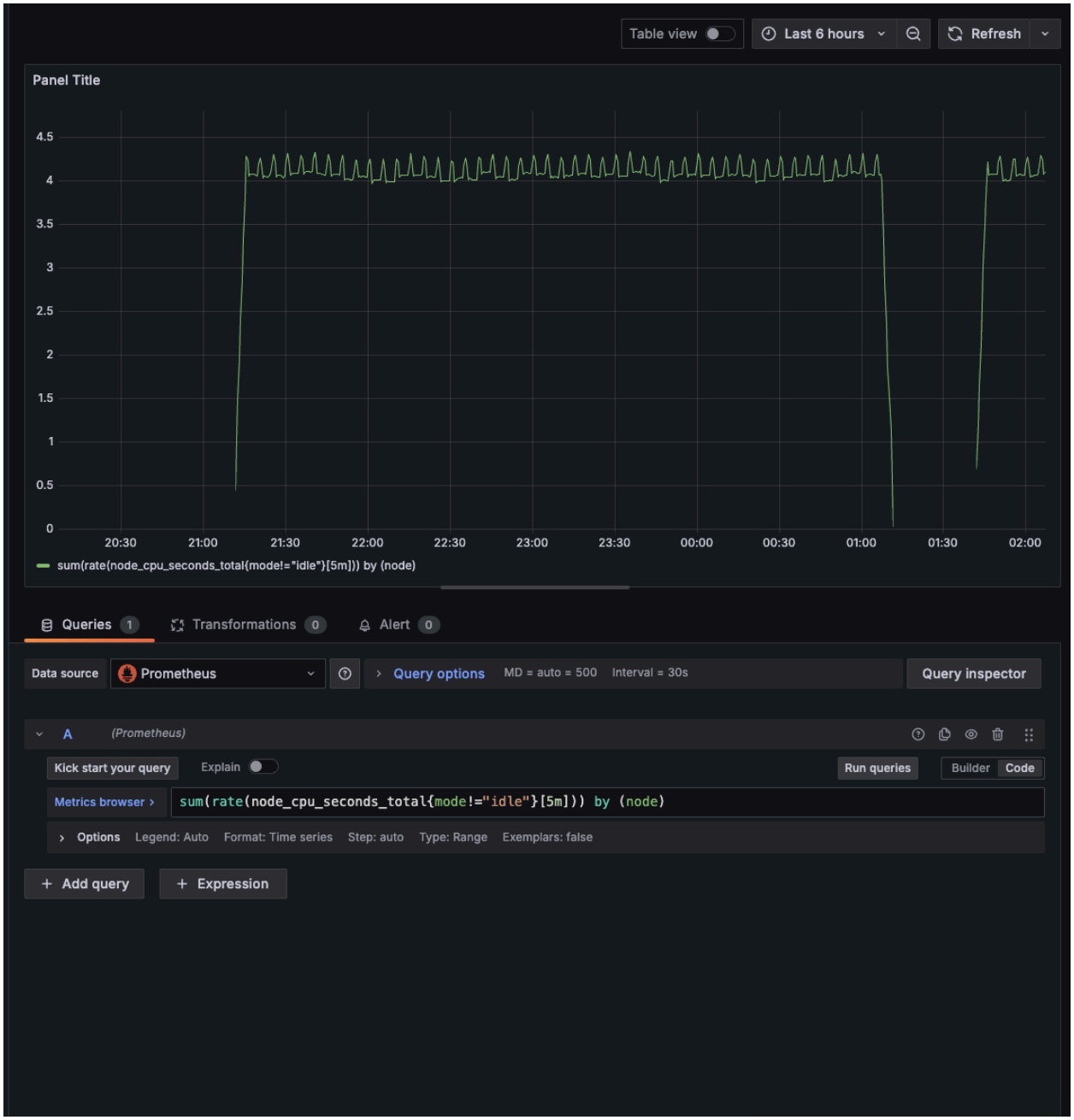


- **Total Cumulative CPU Time**

- **Cluster Health:**



- **Total number of pods by status**

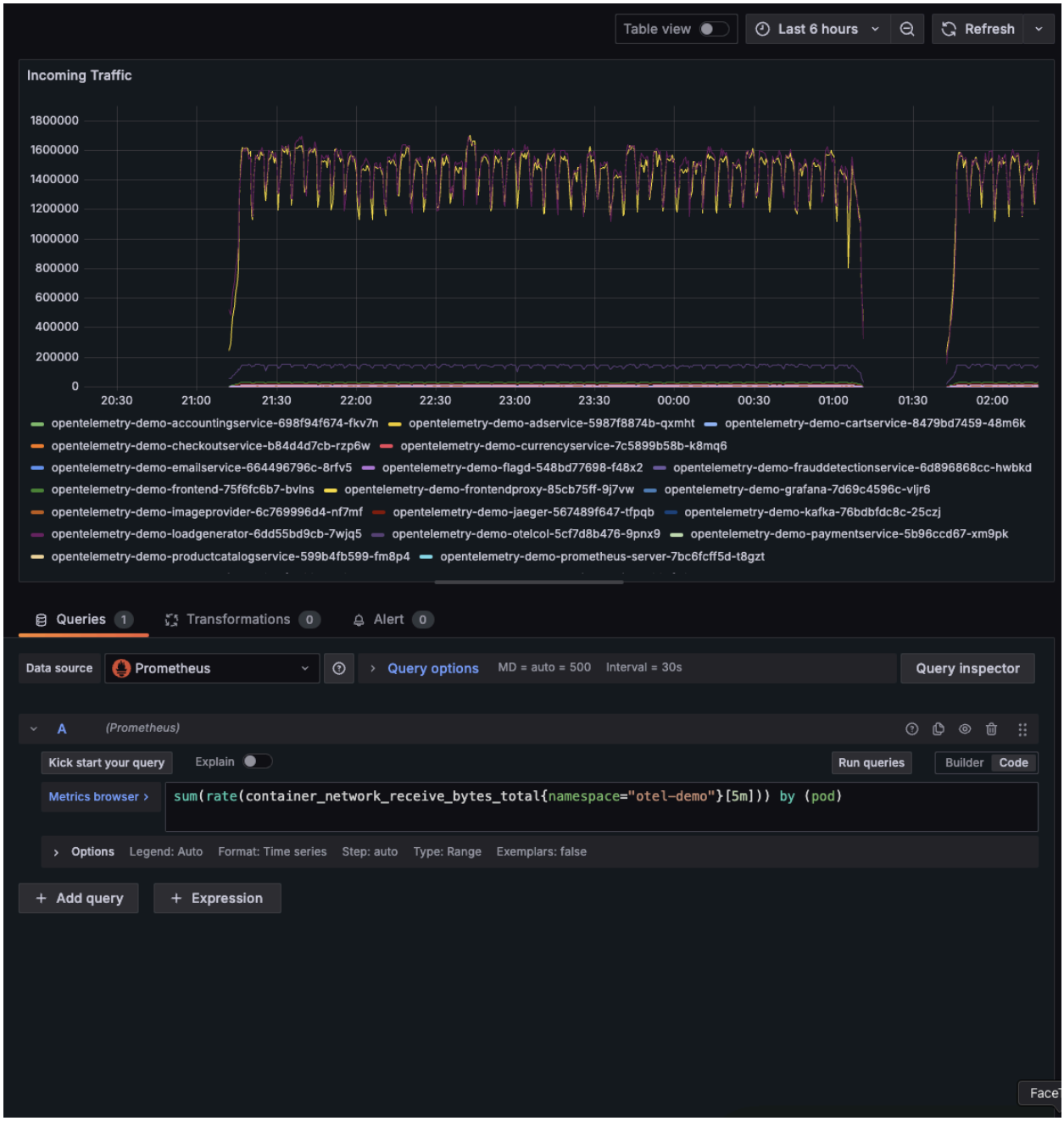


- **CPU Utilization on a Node**

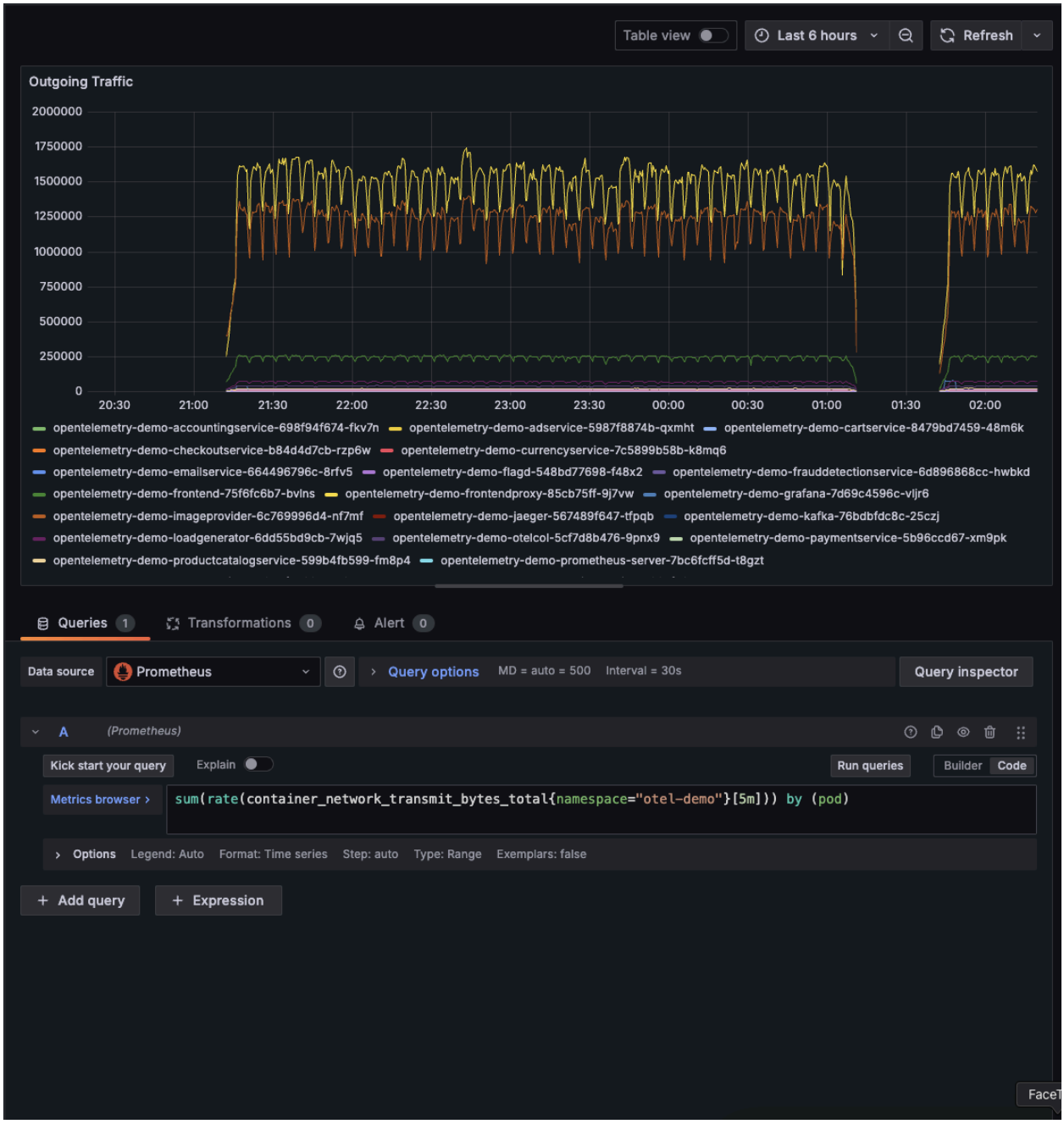
3. **Extending our dashboard top include network traffic between pods and persistent storage usage**

   We expanded our dashboard by integrating metrics for network traffic between pods and persistent storage usage. For network monitoring, we tracked using container_network_receive_bytes_total and container_network_transmit_bytes_total, metrics that measure bytes received and transmitted by each container. This monitoring helps identify network bottlenecks and unusual traffic patterns in pod-to-pod communication.

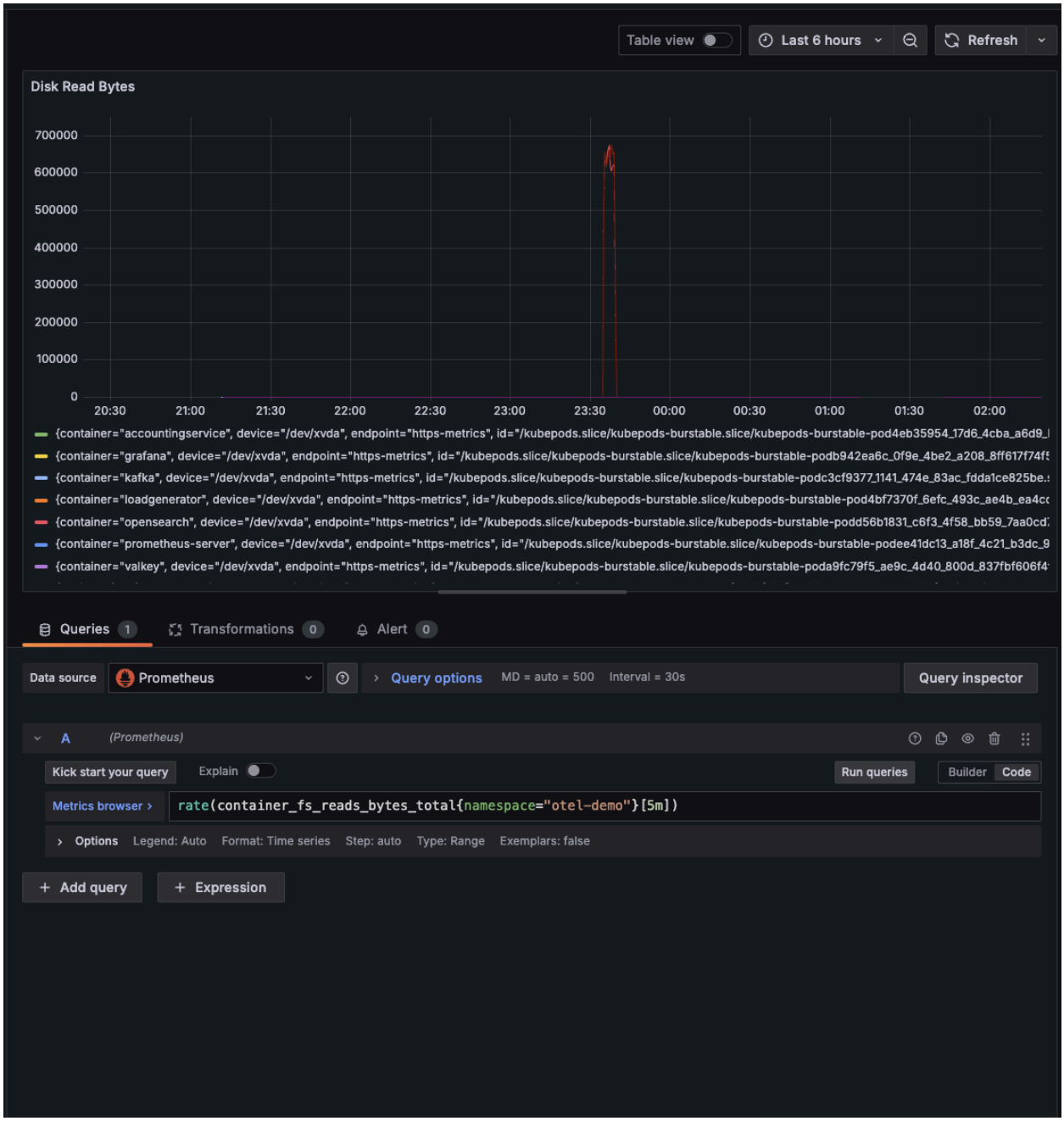
   For storage monitoring, we implemented container_fs_reads_bytes_total and container_fs_writes_bytes_total to track the number of bytes read and written from the container's filesystem. These metrics enable us to monitor storage consumption trends, identify rapid disk usage growth, and trigger alerts for low storage scenarios. These additions will strengthen our dashboard with essential performance insights and improve resource management in our Kubernetes environment.



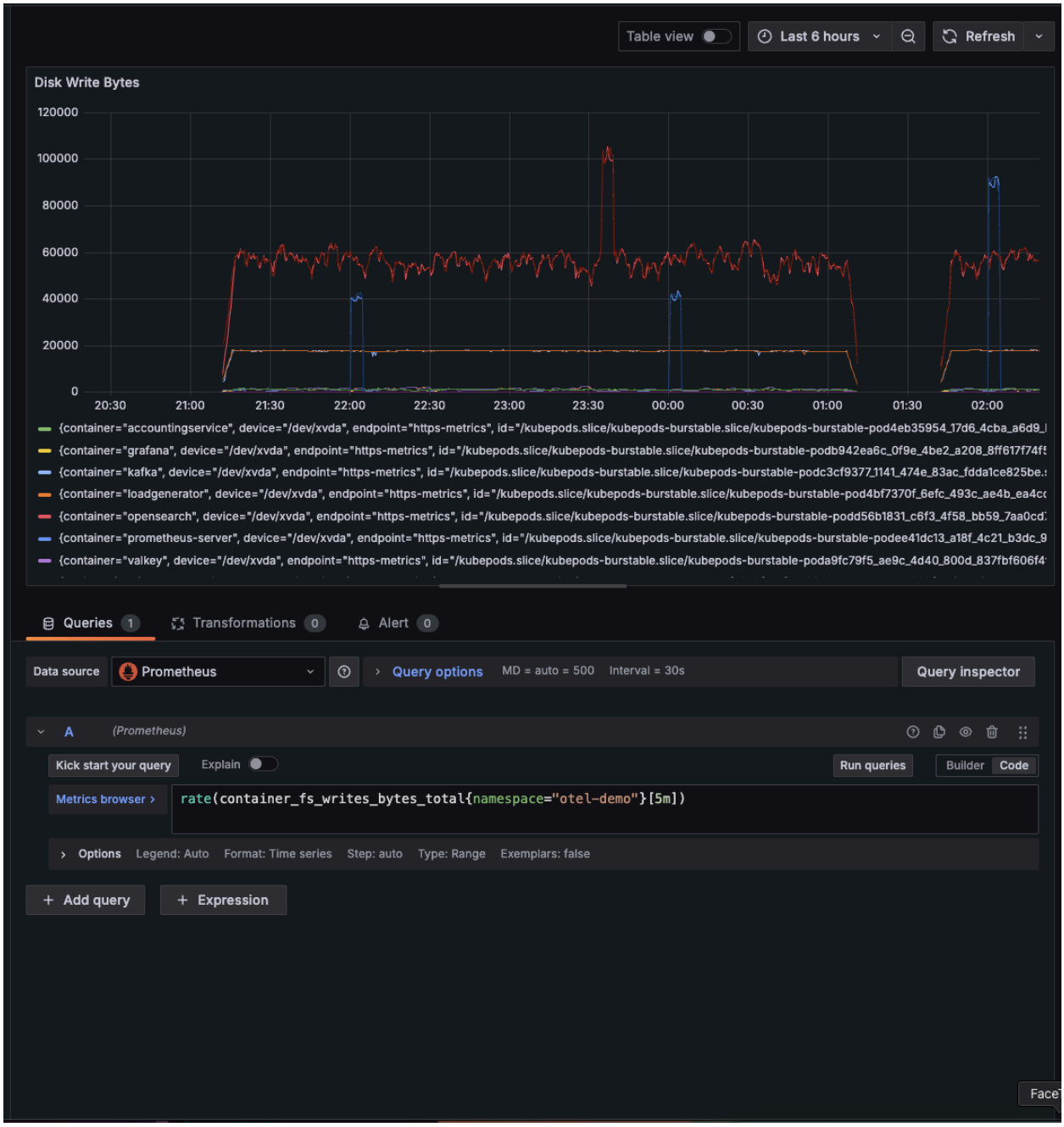
- **Total Number of Bytes Received over the Network by a Container**



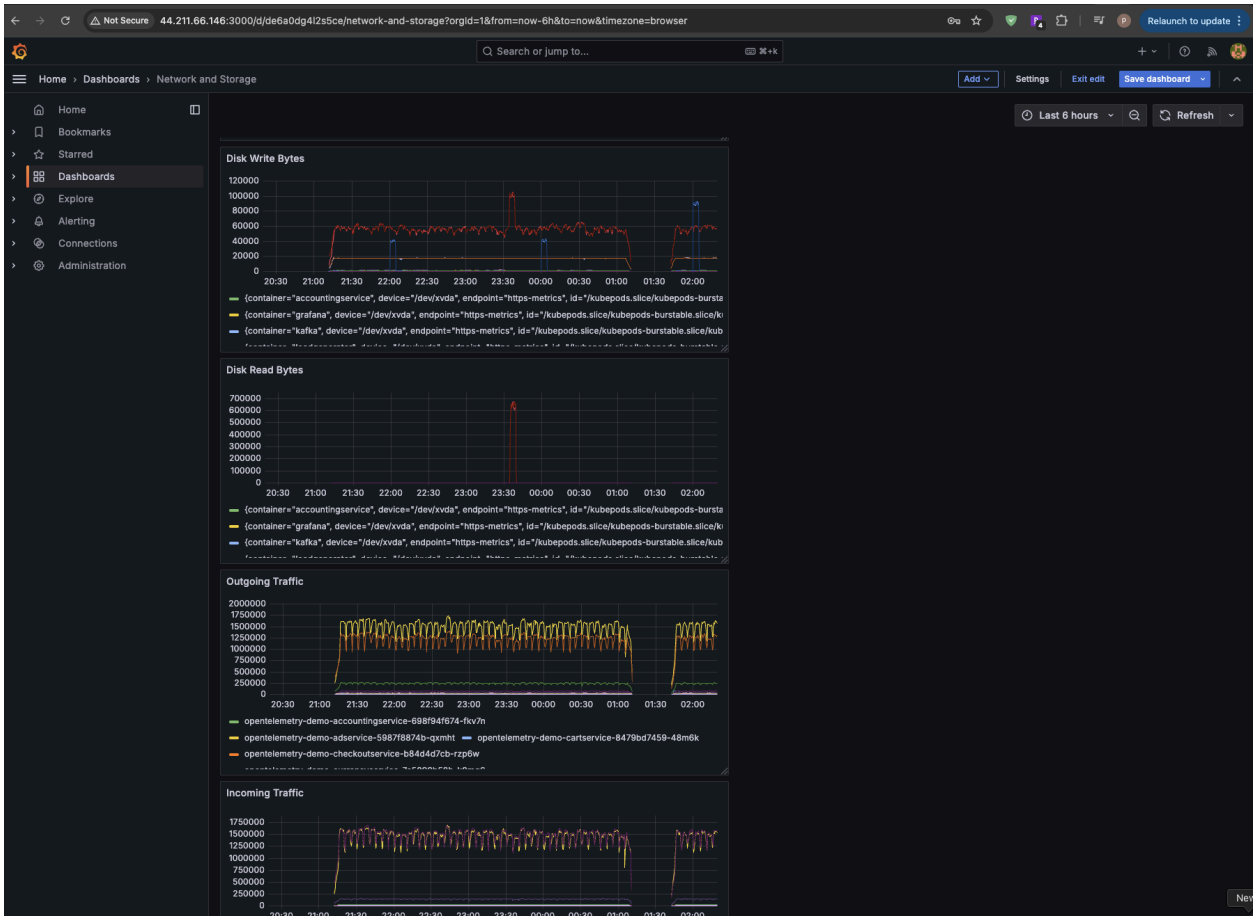
- **Total Number of Bytes Transmitted over the Network by a Container**



- **Total Number of Bytes Read from the Container's FileSystem**



- **Total Number of Bytes Written to the Container's FileSystem**



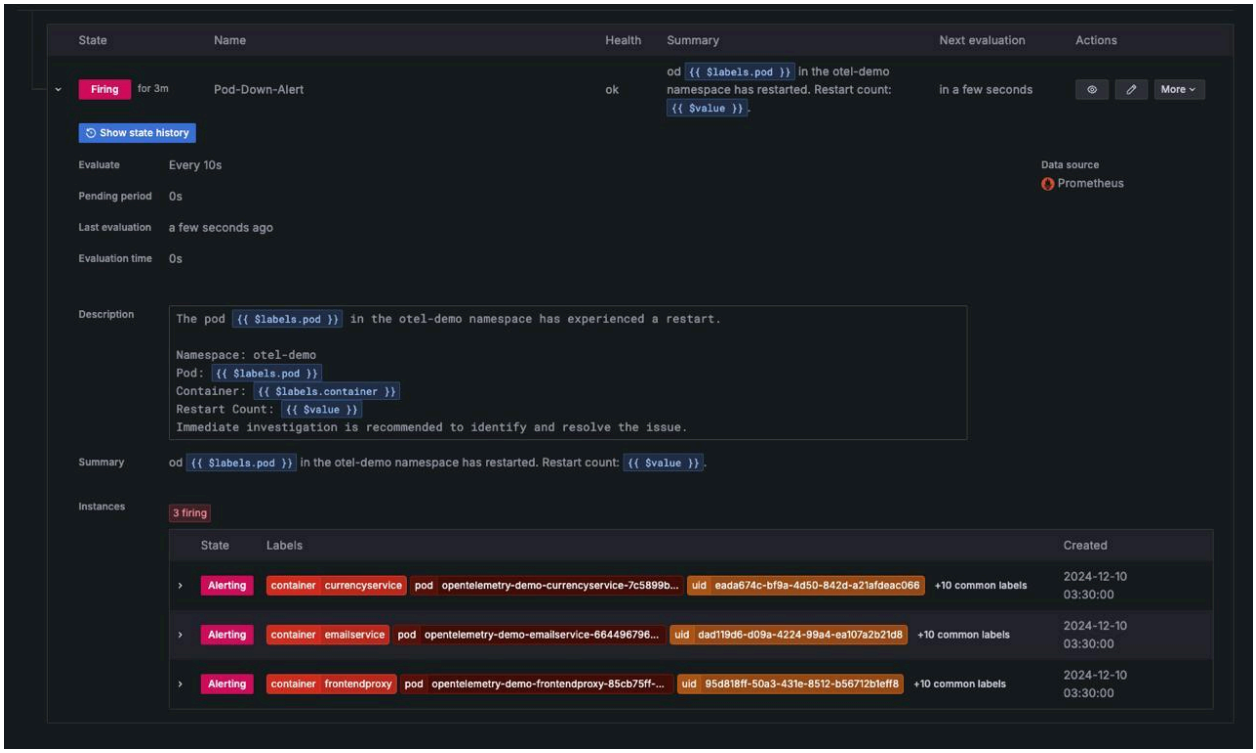
- **Network and Storage Dashboard**

4. Setting Up Alerts in Grafana:

```
ubuntu@ip-172-31-89-201:~$ kubectl get pod opentelemetry-demo-frontendproxy-85cb75ff-9j7vw -n otel-demo
NAME                                              READY   STATUS    RESTARTS        AGE
opentelemetry-demo-frontendproxy-85cb75ff-9j7vw   1/1     Running   4 (4m23s ago)   3d7h
ubuntu@ip-172-31-89-201:~$
```

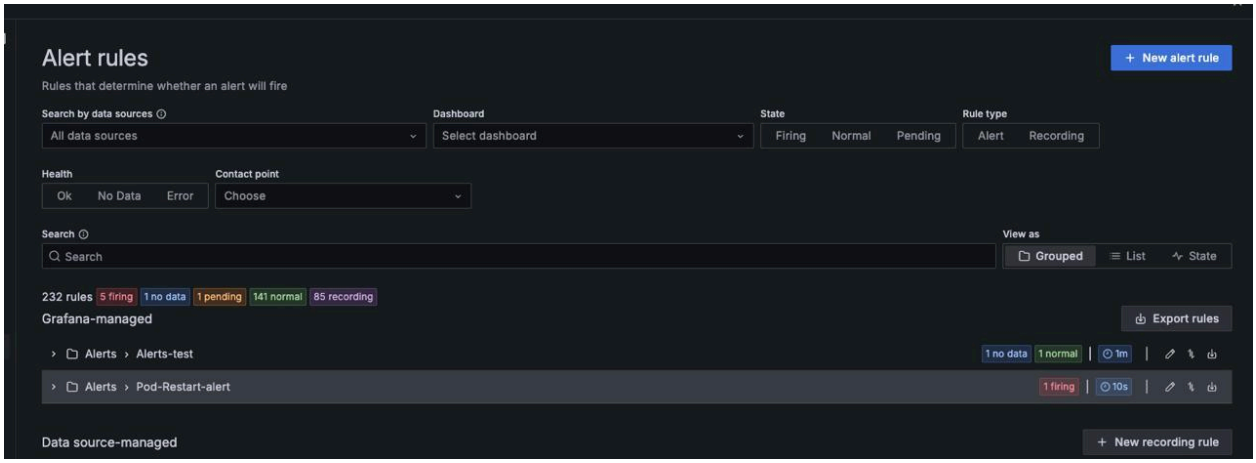
We configured alert notifications to Slack using OpenTelemetry's built-in metrics for critical system monitoring.

- **Pod Down Alert:**

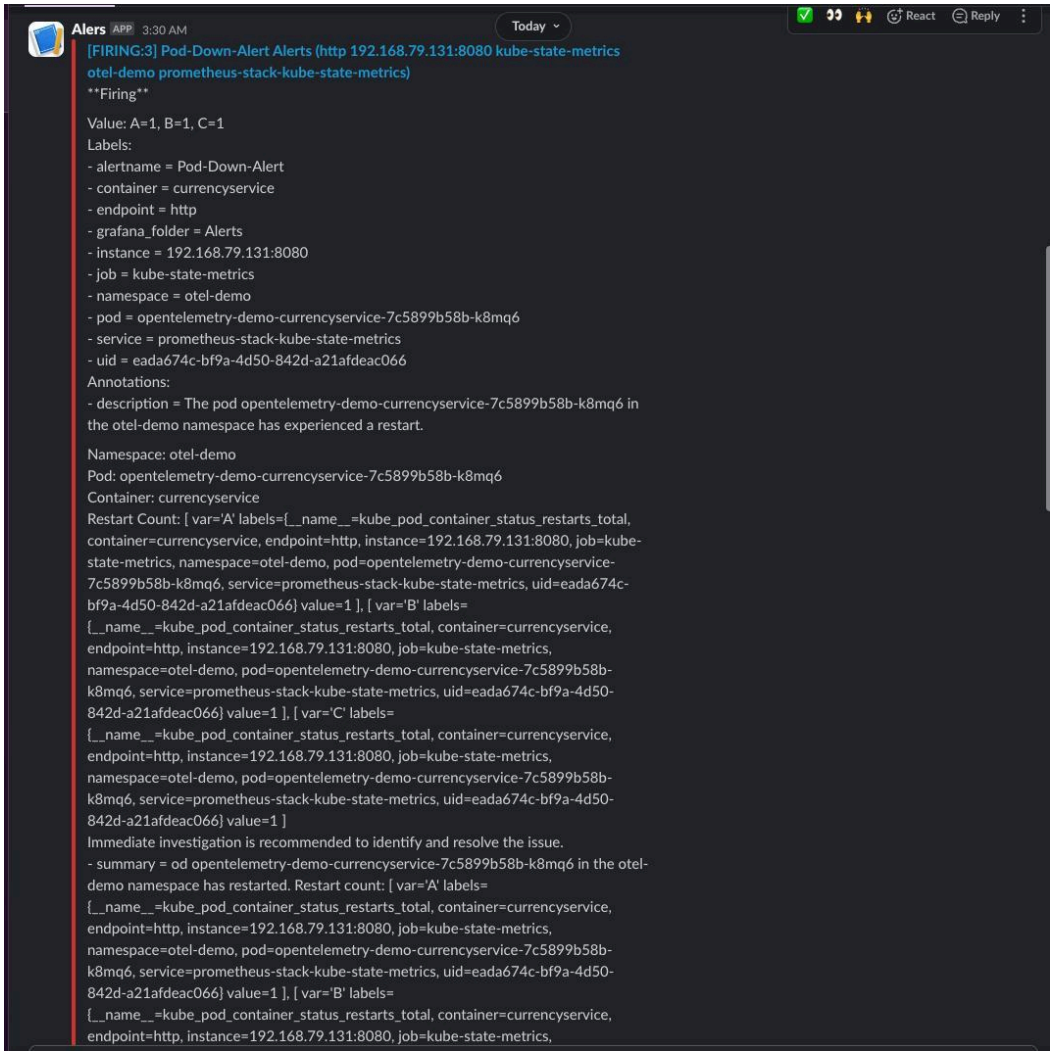


- **Pod Restart Alert:**

- **Alert Rules**



- **Slack Notifications for Alerts:**

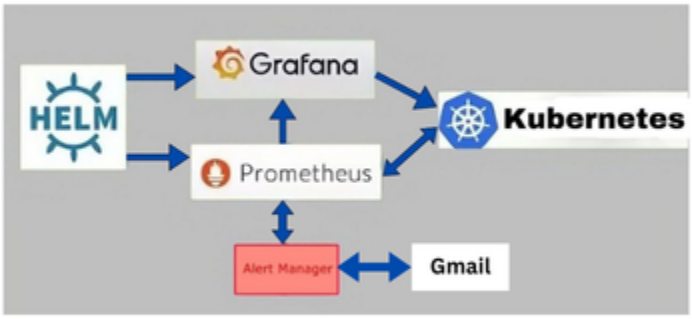


Conclusion:

Phase 4 of the project concluded successfully with the customization of Grafana dashboards for comprehensive EKS monitoring. We utilized pre-built OpenTelemetry dashboards to visualize crucial metrics, including cluster health, pod-level performance, node details, and service metrics. Our monitoring capabilities were enhanced by expanding the dashboards to track network traffic and storage usage. We implemented critical metric alerts with notifications through Slack and email channels, enabling effective monitoring and early issue detection.

ALERTING SERVICE AND NOTIFICATIONS

Objective : The objective of this task was to set up a monitoring and alerting system for Kubernetes using Prometheus and Grafana. Helm was used to deploy these tools to collect and visualize pod metrics. An alerting rule was configured in Prometheus to monitor pod restart counts, and Alertmanager was integrated to send email notifications via Gmail when a pod's restart count exceeded two times within five minutes. This system ensures proactive monitoring and quick issue resolution.



Implementation:

Monitoring setup

1.Prometheus was deployed as the monitoring tool in the Kubernetes cluster using the Helm chart for easy installation and configuration. It was set up to scrape metrics from various Kubernetes components, including nodes, pods, and services, providing real-time performance data. Key metrics like pod restarts were monitored to ensure application health. The deployment was validated by checking the Prometheus pod status and using its web UI to verify that metrics were being correctly collected. This setup forms the foundation for further alerting and visualization through Grafana, enabling proactive monitoring and issue detection.

```
Unset
helm repo add prometheus-community
https://prometheus-community.github.io/helm-charts
```

```
ubuntu@ip-172-31-89-201:~$ kubectl get pods -n monitoring
NAME                                                     READY   STATUS    RESTARTS   AGE
alertmanager-prometheus-stack-kube-prom-alertmanager-0   2/2     Running   0          2d22h
prometheus-prometheus-stack-kube-prom-prometheus-0       2/2     Running   0          2d22h
prometheus-stack-grafana-67db47c9c-zrsn4                 3/3     Running   0          18h
prometheus-stack-kube-prom-operator-6d88b67f9b-46jch     1/1     Running   0          2d22h
prometheus-stack-kube-state-metrics-75ddc4c44-wv8hh      1/1     Running   0          2d22h
prometheus-stack-prometheus-node-exporter-27bdw          1/1     Running   0          2d22h
prometheus-stack-prometheus-node-exporter-2h9lz          1/1     Running   0          2d22h
ubuntu@ip-172-31-89-201:~$
```

2. The monitoring setup was validated by first ensuring that the Prometheus pods were running smoothly within the Kubernetes cluster. This was done by using the kubectl get pods command to check the status of the Prometheus pods, confirming that they were in a "Running" state.

Unset

```
kubectl get all -n monitoring
```

```
ubuntu@ip-172-31-89-201:~$ kubectl get all -n monitoring
NAME                                                     READY   STATUS    RESTARTS   AGE
pod/alertmanager-prometheus-stack-kube-prom-alertmanager-0   2/2     Running   0          2d22h
pod/prometheus-prometheus-stack-kube-prom-prometheus-0       2/2     Running   0          2d22h
pod/prometheus-stack-grafana-67db47c9c-zrsn4                 3/3     Running   0          18h
pod/prometheus-stack-kube-prom-operator-6d88b67f9b-46jch     1/1     Running   0          2d22h
pod/prometheus-stack-kube-state-metrics-75ddc4c44-wv8hh      1/1     Running   0          2d22h
pod/prometheus-stack-prometheus-node-exporter-27bdw          1/1     Running   0          2d22h
pod/prometheus-stack-prometheus-node-exporter-2h9lz          1/1     Running   0          2d22h

NAME                                              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
service/alertmanager-operated                     ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP   2d22h
service/prometheus-operated                       ClusterIP   None             <none>        9090/TCP                     2d22h
service/prometheus-stack-grafana                  ClusterIP   10.100.187.104   <none>        80/TCP                       2d22h
service/prometheus-stack-kube-prom-alertmanager   ClusterIP   10.100.67.253    <none>        9093/TCP,8080/TCP            2d22h
service/prometheus-stack-kube-prom-operator       ClusterIP   10.100.111.29    <none>        443/TCP                      2d22h
service/prometheus-stack-kube-prom-prometheus     ClusterIP   10.100.196.112   <none>        9090/TCP,8080/TCP            2d22h
service/prometheus-stack-kube-state-metrics       ClusterIP   10.100.159.94    <none>        8080/TCP                     2d22h
service/prometheus-stack-prometheus-node-exporter ClusterIP   10.100.6.45      <none>        9100/TCP                     2d22h

NAME                                                    DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
daemonset.apps/prometheus-stack-prometheus-node-exporter   2         2         2       2            2           kubernetes.io/os=linux   2d22h

NAME                                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/prometheus-stack-grafana           1/1     1            1           18h
deployment.apps/prometheus-stack-kube-prom-operator   1/1     1            1           2d22h
deployment.apps/prometheus-stack-kube-state-metrics   1/1     1            1           2d22h

NAME                                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/prometheus-stack-grafana-67db47c9c    1         1         1       18h
replicaset.apps/prometheus-stack-kube-prom-operator-6d88b67f9b   1         1         1       2d22h
replicaset.apps/prometheus-stack-kube-state-metrics-75ddc4c44    1         1         1       2d22h

NAME                                                READY   AGE
statefulset.apps/alertmanager-prometheus-stack-kube-prom-alertmanager   1/1     2d22h
statefulset.apps/prometheus-prometheus-stack-kube-prom-prometheus       1/1     2d22h
ubuntu@ip-172-31-89-201:~$
```

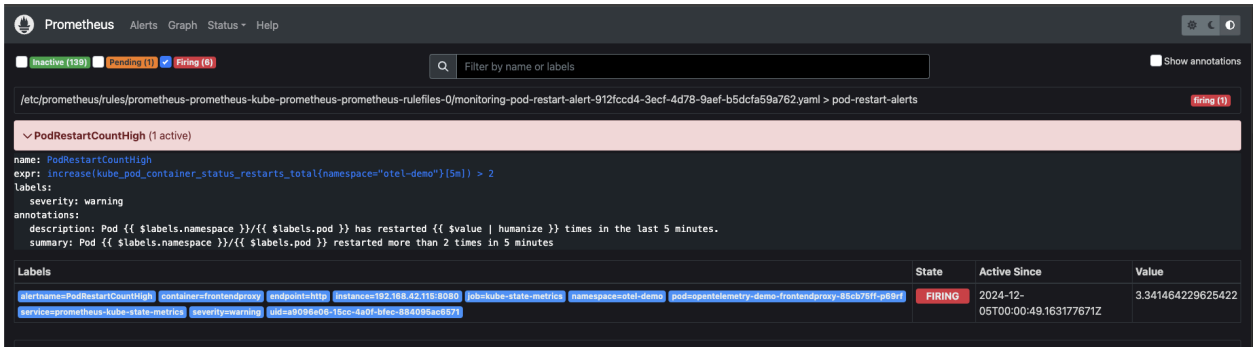
Defined Alert rules

1.An alerting rule was configured in Prometheus to monitor pod restart counts and trigger an alert if a pod's restart count exceeded 2 within a 5-minute window. This was achieved by defining a custom Prometheus query that tracks pod restart metrics using the kube_pod_container_status_restarts_total metric. The rule was designed to fire when the metric

exceeds the threshold of 2 restarts in the specified time interval. By setting this rule, it ensures that any potential issues with application stability, such as frequent crashes or misconfigurations, are promptly detected. The alert was integrated with Alertmanager to send notifications for quick resolution. This proactive monitoring helps maintain the health and reliability of the application running in the Kubernetes cluster.

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: pod-restart-alert
  namespace: monitoring
  labels:
    release: prometheus
spec:
  groups:
    - name: pod-restart-alerts
      rules:
        - alert: PodRestartCountHigh
          expr: |
            increase(kube_pod_container_status_restarts_total{
              namespace="otel-demo"
            }[5m]) > 2
          for: 0m
          labels:
            severity: warning
          annotations:
            summary: "Pod {{ $labels.namespace }}/{{ $labels.pod }} restarted more than 2 times in 5 minutes"
            description: "Pod {{ $labels.namespace }}/{{ $labels.pod }} has restarted {{ $value | humanize }} tim
es in the last 5 minutes."
~
~
~
```

2.Capturing pod restart through prometheus



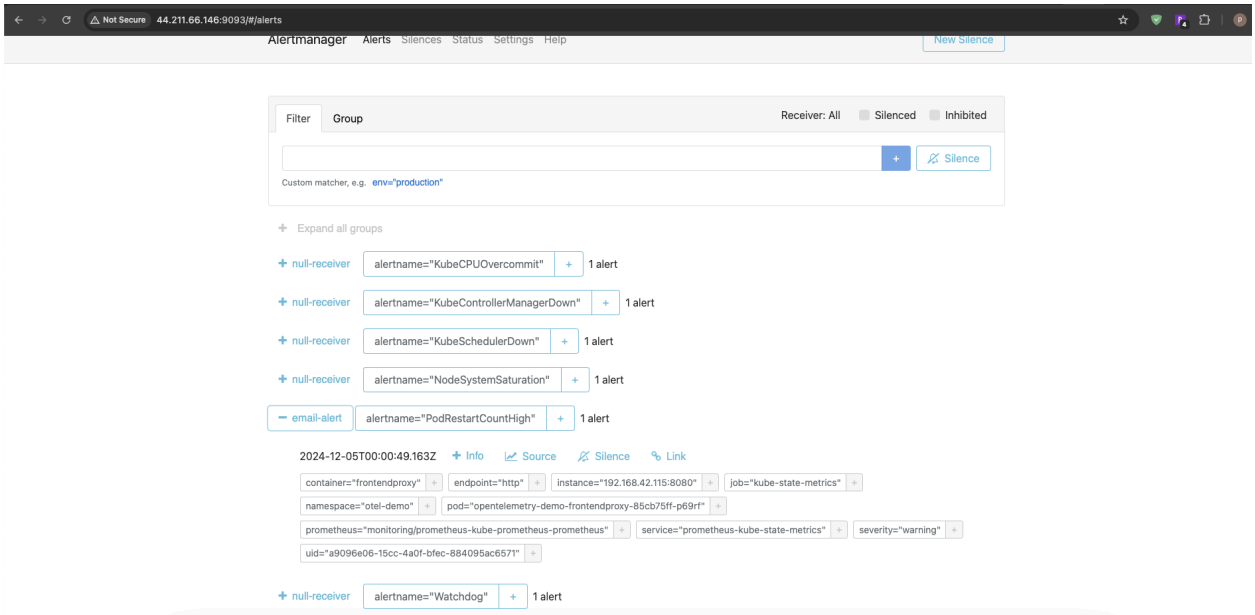
Configured Email notifications

1.AWS Simple Email Service (SES) was integrated with Alertmanager to enable email notifications when an alert is triggered in Prometheus. The integration involved configuring Alertmanager to use AWS SES as the notification channel by specifying the SES SMTP settings in the Alertmanager configuration file. Once configured, Alertmanager automatically sends email alerts when the defined Prometheus alerting rule, such as the pod restart count exceeding 2, is triggered.

Configured alert manager

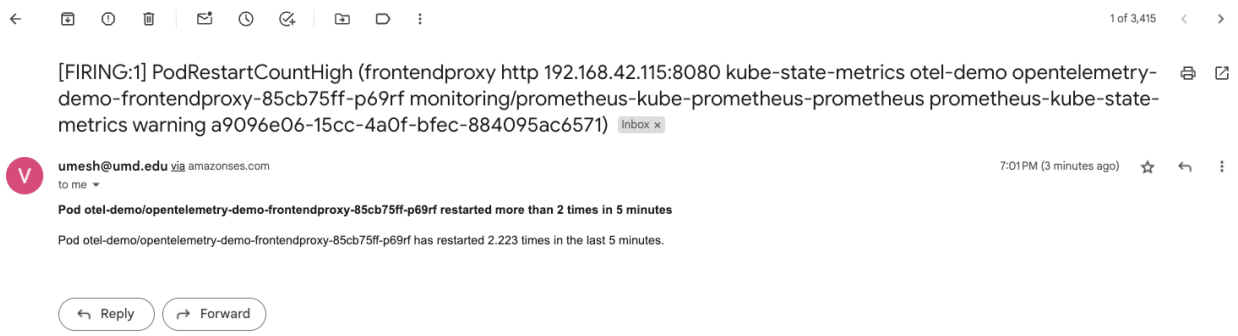
```yaml
apiVersion: v1
kind: Secret
metadata:
  name: alertmanager-config
  namespace: monitoring
stringData:
  alertmanager.yaml: |
    global:
      smtp_smarthost: "email-smtp.us-east-1.amazonaws.com:587"
      smtp_from: "umesh@umd.edu"
      smtp_auth_username: "AKIA356SJZMAFZTBYHWN"
      smtp_auth_password: "BKlD5GXyPCM3A4KNoeo2h3fUQAhm9ffYSEBe5G4s2PC9"
      smtp_require_tls: true
    route:
      receiver: "null-receiver"  # Default receiver for unmatched alerts
      group_by: ['alertname']
      routes:
        - receiver: "email-alert"
          matchers:
            - alertname="PodRestartCountHigh"
          group_wait: 30s
          group_interval: 5m
          repeat_interval: 12h
    receivers:
      - name: "email-alert"
        email_configs:
          - to: "umesh@umd.edu"
            send_resolved: true
            html: |
              <b>{{ .CommonAnnotations.summary }}</b><br>
              <p>{{ .CommonAnnotations.description }}</p>
      - name: "null-receiver"
```

2. Captured alert through alert manager



3.SES was set up to handle the sending of emails reliably, ensuring that relevant stakeholders receive timely notifications. This integration helps ensure that issues are detected and communicated promptly, allowing for quick action to resolve potential problems. The use of SES also ensures scalability and efficiency in handling large volumes of notifications.

Triggered the alert by creating a scenario where a pod exceeded the 2 restart count in 5 minutes



CI/CD INTEGRATION

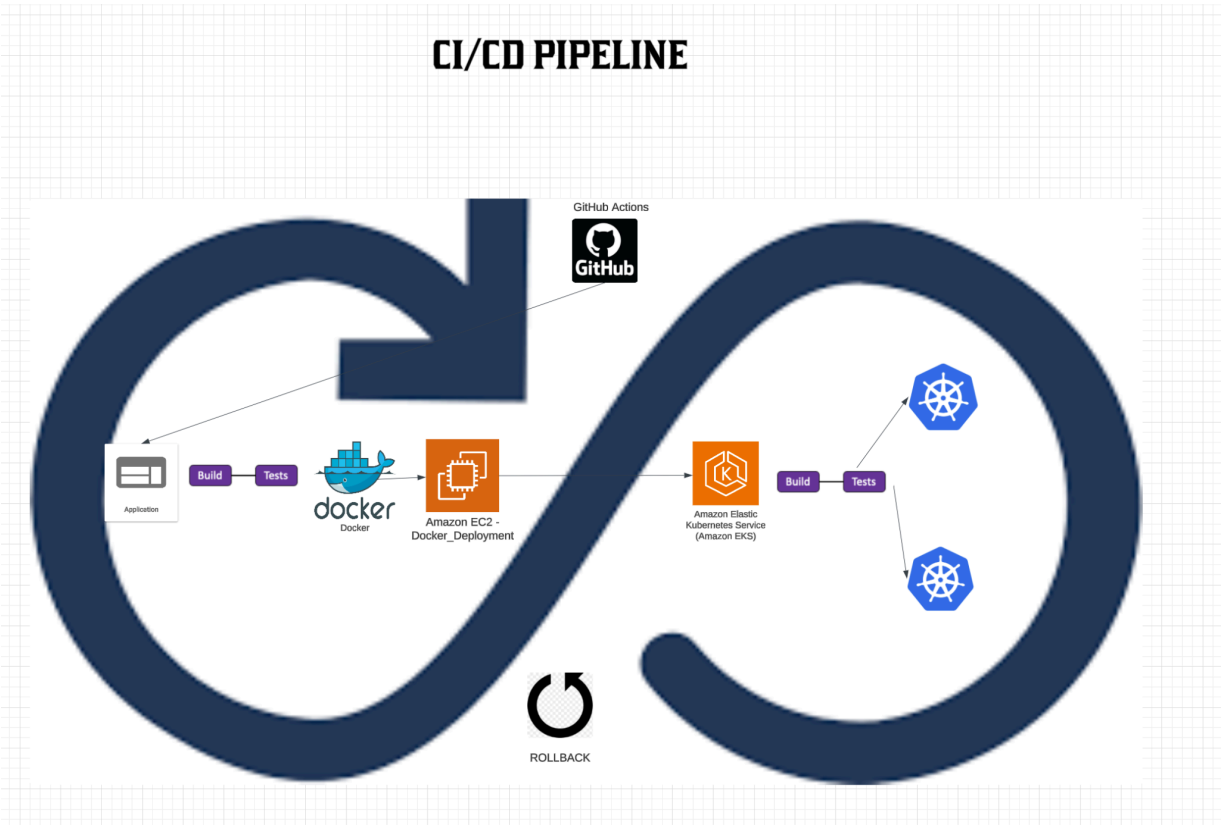
Our GitHub Repository Link: https://github.com/pothuguntaumesh/opentelemetry-demo
Objective :

CI/CD (Continuous Integration and Continuous Deployment) is all about streamlining the software development process to make it faster, and more reliable. Continuous Integration (CI) means regularly integrating code changes into a shared repository, where automated tests check for any issues, ensuring new code doesn't break the existing system. Continuous Deployment (CD) takes it further by automatically deploying the application to staging or production once the tests pass, allowing for quick updates with minimal manual effort.

At the heart of CI/CD are pipelines, which are automated workflows that handle everything from building and testing the code to deploying it, ensuring consistency and repeatability across the process. These workflows define the order and logic of each step—like running tests, checking code quality, and handling deployment—making sure each task happens in the right sequence.

The main benefits of CI/CD pipelines are faster release cycles, fewer chances for human error, better code quality, and a quicker response to bugs or new features. All of this leads to more productive developers and better experiences for end users.

Implementation :



We successfully set up a fully automated CI/CD pipeline using **GitHub Actions**. Our pipeline automates several key processes that streamline the application lifecycle. We started by automating the **building of container images** for the application. Each time new code is pushed, the pipeline automatically builds the Docker images and pushes them to a container registry. Following this, we integrated **automated tests** to run after every build, ensuring that the application's core functionality is validated and no issues are introduced in the process.

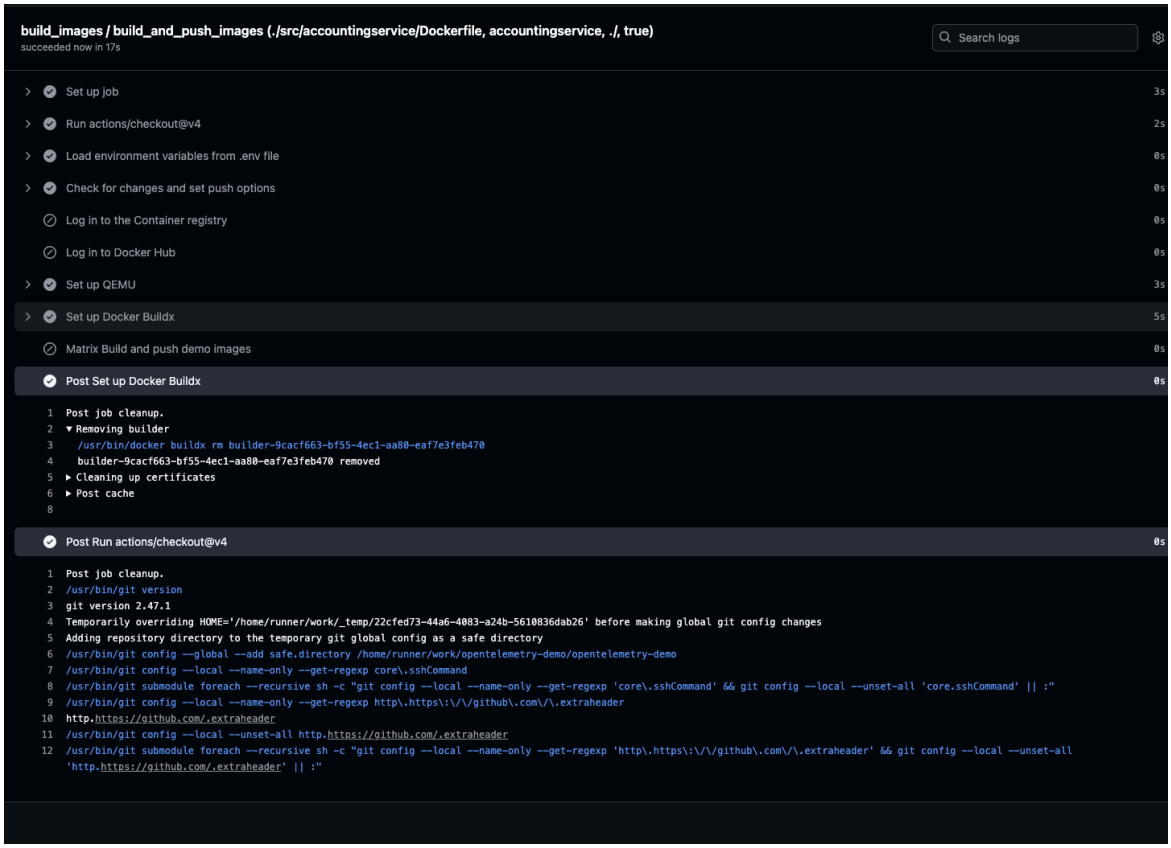
Once the tests pass, the pipeline proceeds to **automatically deploy the application** to a Kubernetes cluster, ensuring a seamless deployment process every time. To address potential deployment issues, we incorporated a **rollback mechanism** into the pipeline. This allows the application to revert to the last stable deployment in case of any failures.

Our deliverables include fully functional **CI/CD pipeline configuration files** that define the steps for building, testing, and deploying the application, integrated with a container registry for storing images. Additionally, we provided logs and screenshots demonstrating successful **builds**, **tests**, **deployments**, and **rollback actions**. To ensure security, we implemented proper management of sensitive data, such as credentials and environment variables, using GitHub Secrets, safeguarding against unauthorized access. Through this process, we've not only automated the application lifecycle but also ensured a smooth, error-free deployment process with rapid recovery in case of failures.

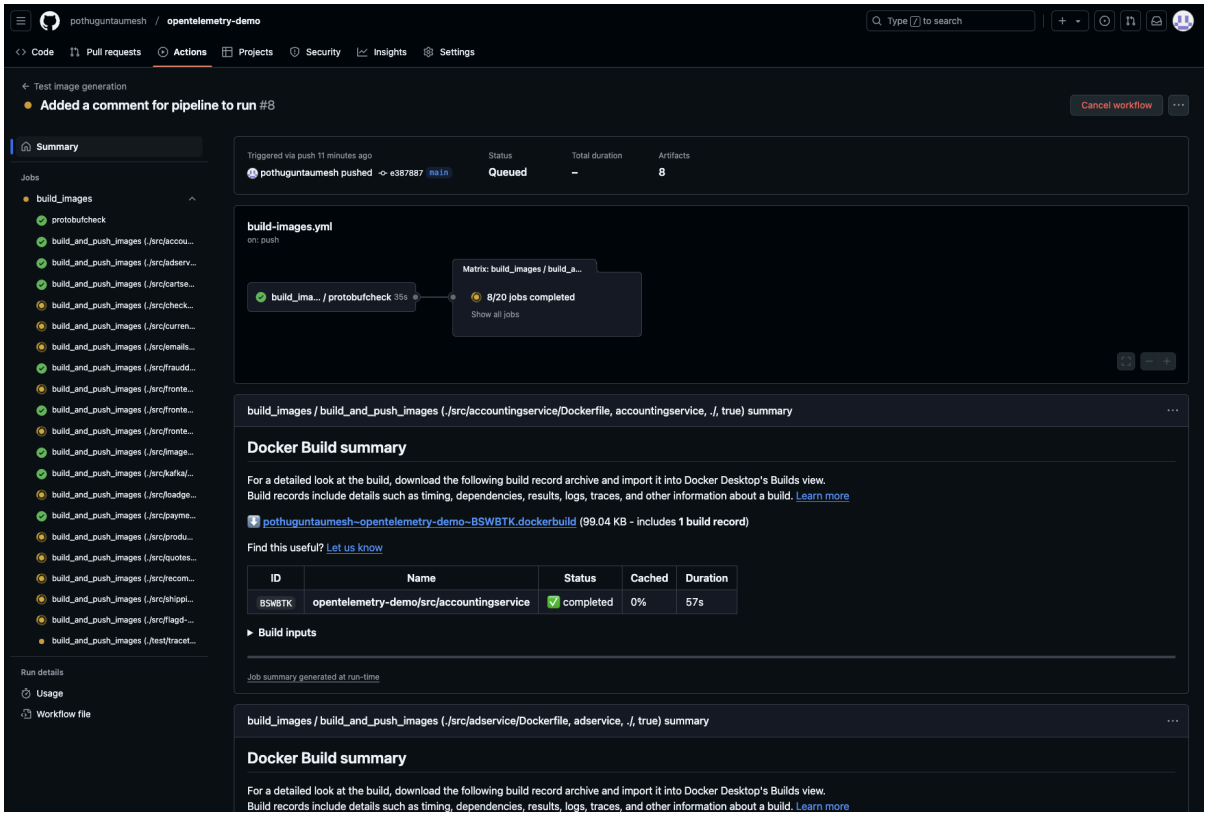
Screenshots and Steps:

Building container images for the application:

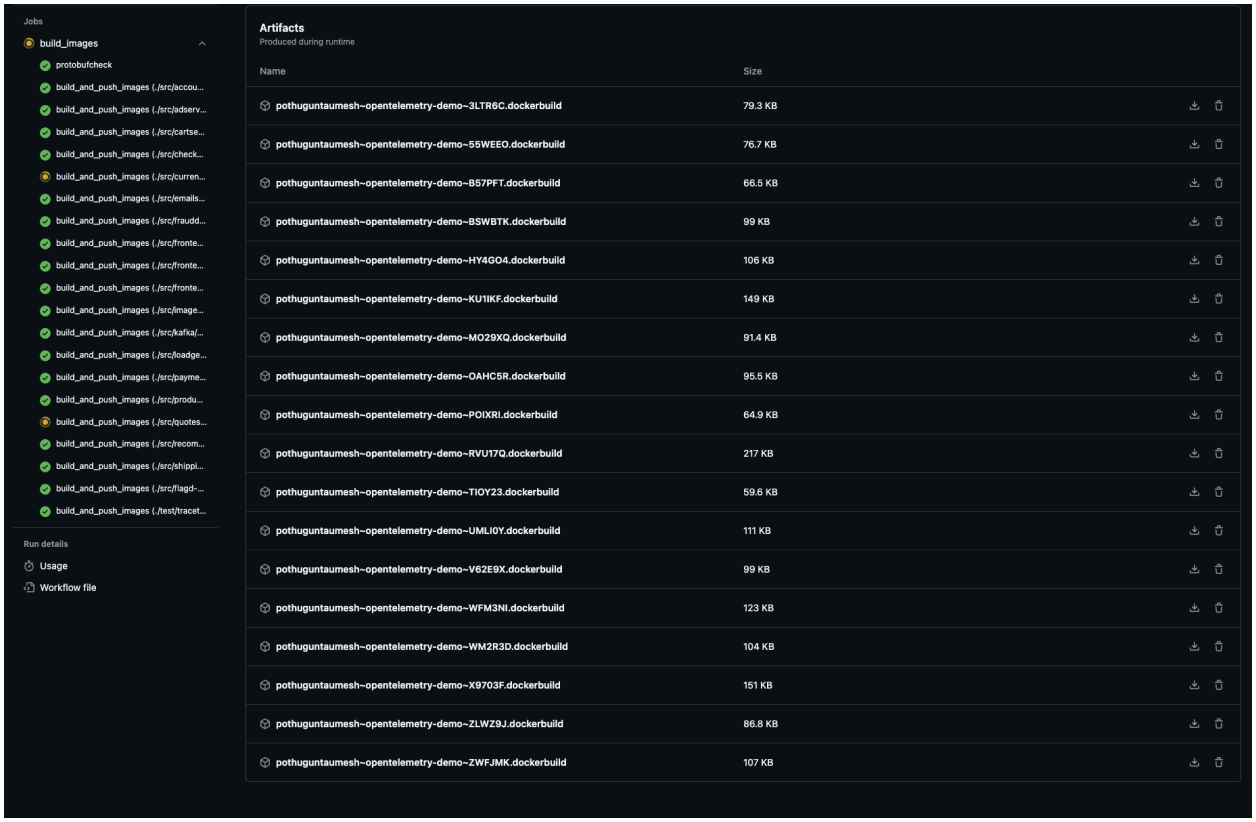
In this pipeline, an image is built for each service, such as the account service example here, by first compiling the necessary code and dependencies into a Docker image. After the build, the image is pushed to GitHub's container registry, where the YAML file fetches it for deployment. Following the build process, a cleanup step is executed to remove any temporary cache and certificates, ensuring the environment remains clean and optimized for future builds.



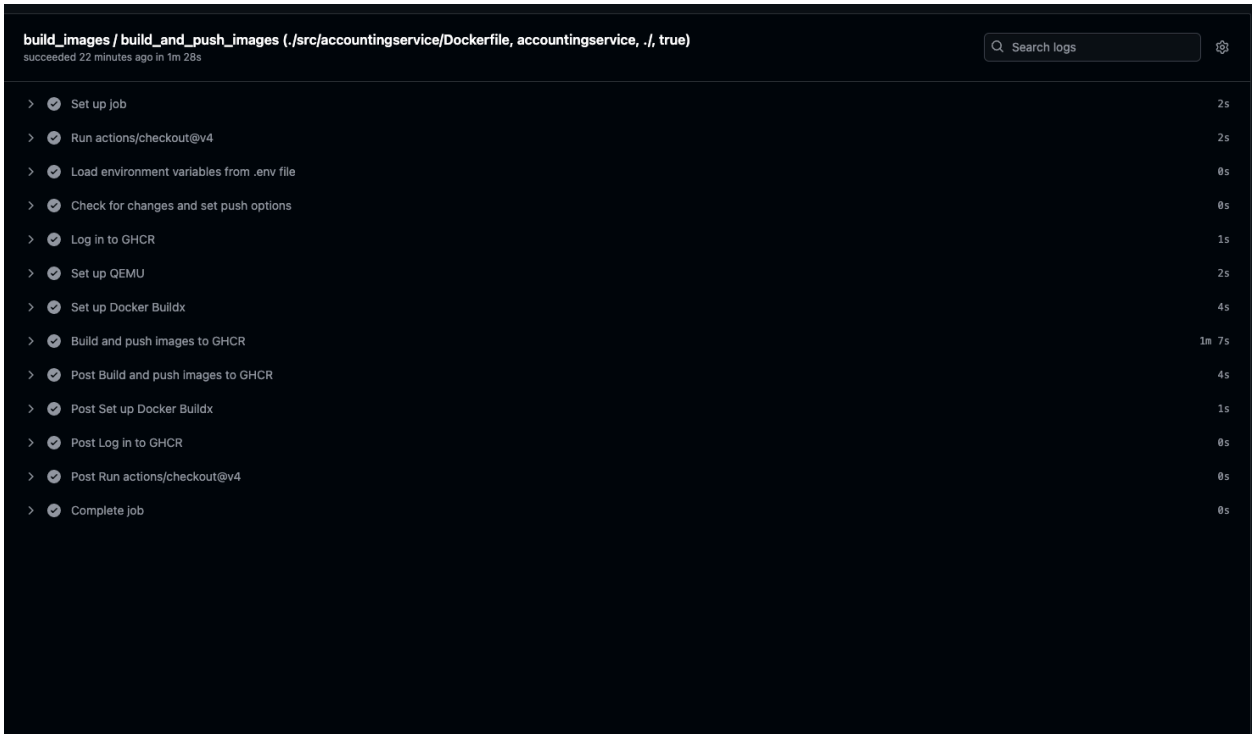
Here, container images for the application are automatically built whenever new code is pushed to the repository. The pipeline compiles the application, packages it into a Docker image, and then pushes the image to a container registry, ready for deployment. This automated process ensures that each update is consistently built and stored, streamlining the development and deployment workflow.

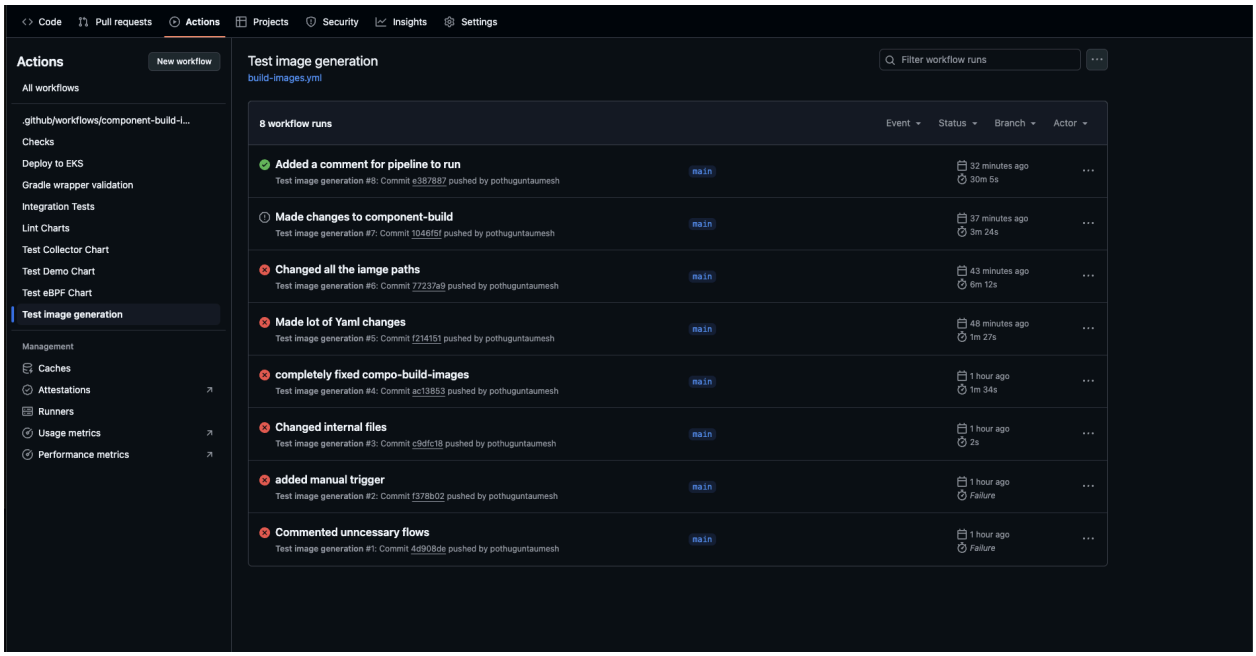


Once the container images are successfully built in the GitHub Actions pipeline, the images are stored in the container registry, ready for deployment. At this stage, the application is fully packaged, with each service image built and tested, ensuring that the code is stable and the necessary dependencies are included. The final result is a complete, deployable set of images that can be seamlessly fetched for deployment to the target environment, ensuring a smooth and efficient release process.



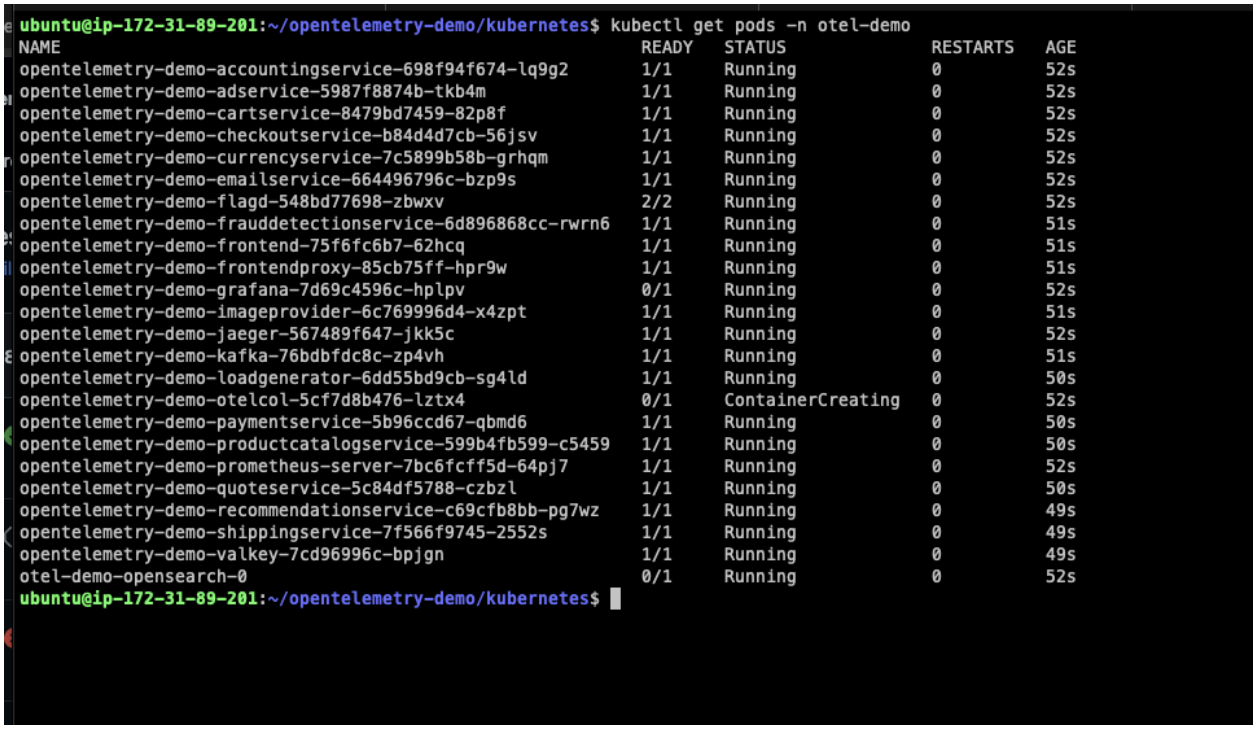
Once the pipeline is deployed, automated tests are triggered to validate the application's functionality. These tests ensure that the code changes have not introduced any issues and that the application behaves as expected, maintaining stability and performance throughout the deployment process.


build_images / build_and_push_images (./src/accountingservice/Dockerfile, accountingservice, ./, true)
succeeded 22 minutes ago in 1m 28s
Search logs
Set up job 2s
Run actions/checkout@v4 2s
Load environment variables from .env file 0s
Check for changes and set push options 0s
Log in to GHCR 1s
Set up QEMU 2s
Set up Docker Buildx 4s
Build and push images to GHCR 1m 7s
Post Build and push images to GHCR 4s
Post Set up Docker Buildx 1s
Post Log in to GHCR 0s
Post Run actions/checkout@v4 0s
Complete job 0s

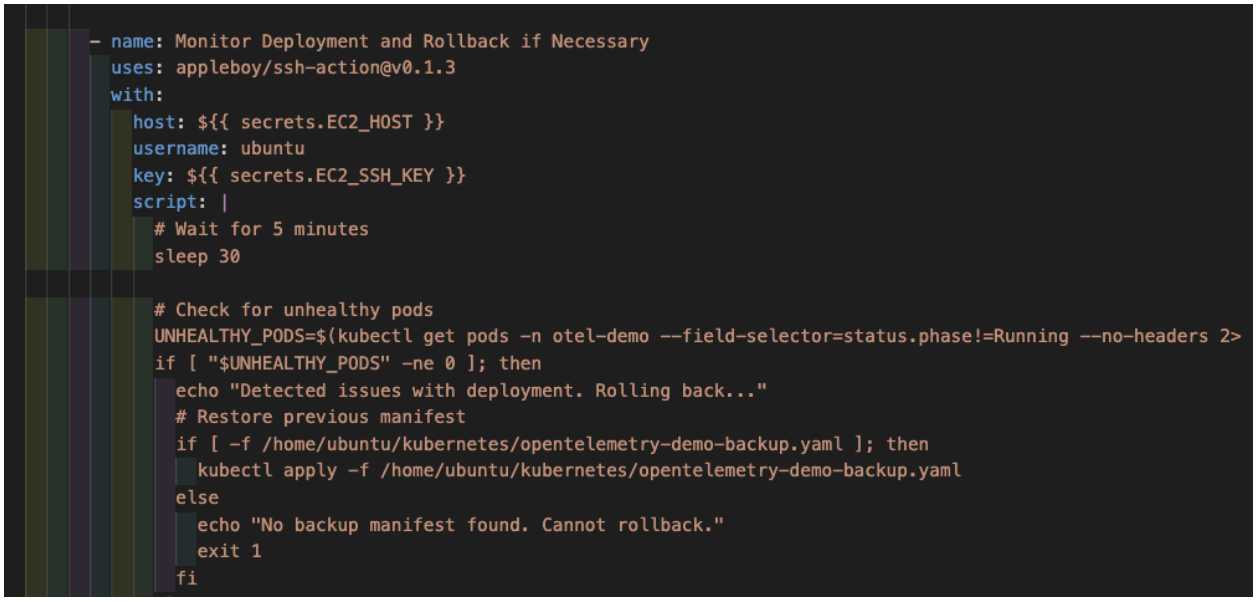

Code
Pull requests
Actions
Projects
Security
Insights
Settings
Actions
New workflow
All workflows
.github/workflows/component-build-i...
Checks
Deploy to EKS
Gradle wrapper validation
Integration Tests
Lint Charts
Test Collector Chart
Test Demo Chart
Test eBPF Chart
Test image generation
Management
Caches
Attestations
Runners
Usage metrics
Performance metrics
Test image generation
build-images.yml
Filter workflow runs
8 workflow runs
Event
Status
Branch
Actor
Added a comment for pipeline to run
Test image generation #8: Commit e387887 pushed by pothuguntaumesh
main
32 minutes ago
30m 5s
Made changes to component-build
Test image generation #7: Commit 1046f5f pushed by pothuguntaumesh
main
37 minutes ago
3m 24s
Changed all the iamge paths
Test image generation #6: Commit 77237a9 pushed by pothuguntaumesh
main
43 minutes ago
6m 12s
Made lot of Yaml changes
Test image generation #5: Commit f214151 pushed by pothuguntaumesh
main
48 minutes ago
1m 27s
completely fixed compo-build-images
Test image generation #4: Commit ac13853 pushed by pothuguntaumesh
main
1 hour ago
1m 34s
Changed internal files
Test image generation #3: Commit c9dfc18 pushed by pothuguntaumesh
main
1 hour ago
2s
added manual trigger
Test image generation #2: Commit f378b02 pushed by pothuguntaumesh
main
1 hour ago
Failure
Commented unncessary flows
Test image generation #1: Commit 4d908de pushed by pothuguntaumesh
main
1 hour ago
Failure

After successful testing, the application is deployed to the Kubernetes cluster. By executing the `kubectl` command, we can verify that the pods are running correctly and ensure the deployment is functioning as expected.

```
Unset
kubectl get pods -n otel-demo
```



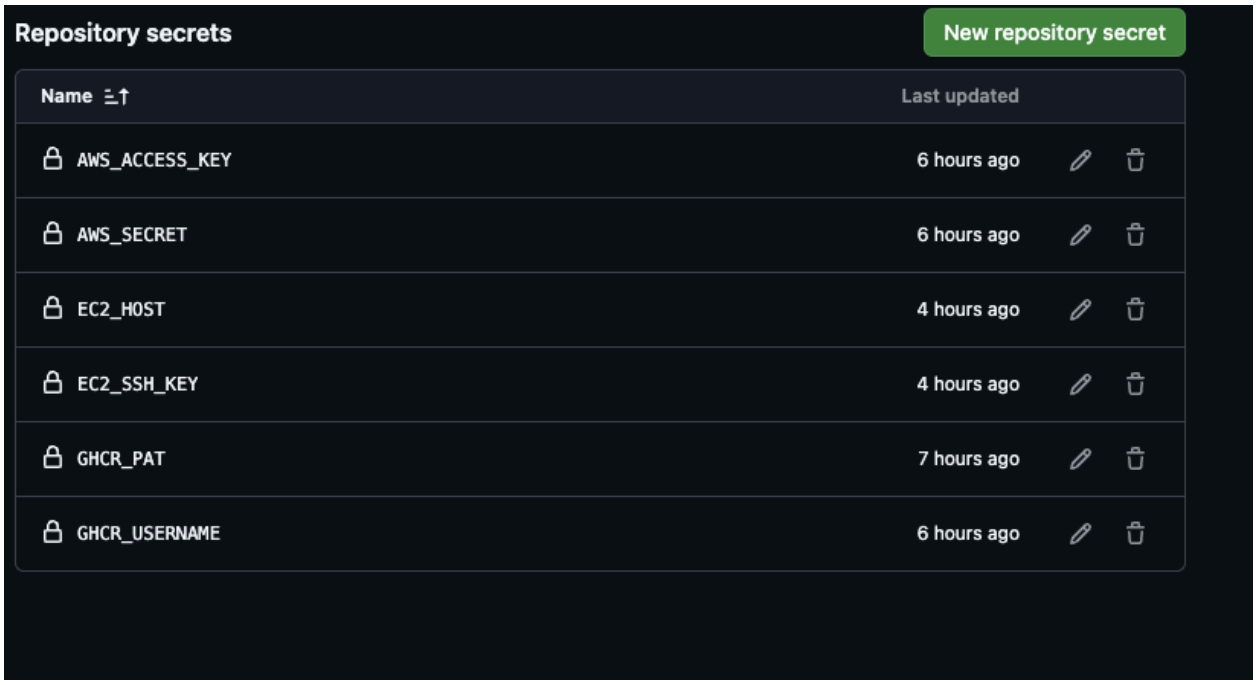
A rollback mechanism has been integrated to automatically revert to the most recent stable deployment in case of any issues. This ensures minimal disruption and quick recovery, maintaining application stability during potential failures.



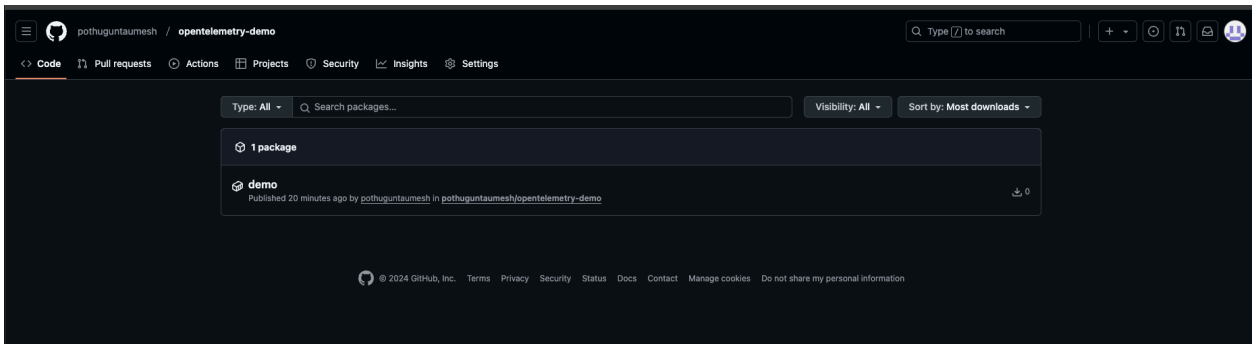
```
- name: Monitor Deployment and Rollback if Necessary
  uses: appleboy/ssh-action@v0.1.3
  with:
    host: ${{ secrets.EC2_HOST }}
    username: ubuntu
    key: ${{ secrets.EC2_SSH_KEY }}
    script: |
      # Wait for 5 minutes
      sleep 30

      # Check for unhealthy pods
      UNHEALTHY_PODS=$(kubectl get pods -n otel-demo --field-selector=status.phase!=Running --no-headers 2>
      if [ "$UNHEALTHY_PODS" -ne 0 ]; then
        echo "Detected issues with deployment. Rolling back..."
        # Restore previous manifest
        if [ -f /home/ubuntu/kubernetes/opentelemetry-demo-backup.yaml ]; then
          kubectl apply -f /home/ubuntu/kubernetes/opentelemetry-demo-backup.yaml
        else
          echo "No backup manifest found. Cannot rollback."
          exit 1
        fi
```

Sensitive data within the code is securely managed by storing all secrets in GitHub Secrets for the repository. This ensures that credentials and other sensitive information are protected and not exposed in the codebase.

```
      fi
  - name: Log in to GHCR
    uses: docker/login-action@v3
    with:
      registry: ghcr.io
      username: ${{ github.repository_owner }}
      password: ${{ secrets.GITHUB_TOKEN }}
    if: ${{ inputs.push }}
  - name: Set up QEMU
    if: ${{ matrix.file_tag.setup-qemu }}
    uses: docker/setup-qemu-action@v3
    with:
      image: tonistiigi/binfmt:master
  - name: Set up Docker Buildx
    uses: docker/setup-buildx-action@v3
    with:
      buildkitd-config-inline: |
```



The "Demo" package contains the container images that are built and generated by the pipeline. These images are stored within the package and are ready to be deployed, running the application services as defined in the pipeline.



The `deploy.yaml` file serves as the central configuration for our deployment pipeline, defining the steps to build, test, and deploy the application. It ensures a consistent, automated workflow by managing tasks like image building, testing, and deployment to the Kubernetes cluster, making the entire process repeatable and efficient.

```C/C++
# Copyright The OpenTelemetry Authors
# SPDX-License-Identifier: Apache-2.0
name: Deploy to AWS EKS

on:
  workflow_run:
    workflows: ["Integration Workflow"]
    types:
      - completed

jobs:
  deploy-to-cluster:
    runs-on: ubuntu-latest
    if: ${{ github.event.workflow_run.conclusion == 'success' }}

    steps:
      - name: Checkout Repository
        uses: actions/checkout@v3

      - name: Install Helm
        uses: azure/setup-helm@v3
        with:
```

```
          version: v3.14.4

      - name: Monitor Deployment Health and Rollback if Needed
        uses: appleboy/ssh-action@v0.1.3
        with:
          host: ${{ secrets.AWS_EC2_HOST }}
          username: ubuntu
          key: ${{ secrets.AWS_EC2_SSH_KEY }}
          script: |
            # Wait for 3 minutes
            sleep 180

            # Check for unhealthy pods
            UNHEALTHY_PODS=$(kubectl get pods -n demo-app
--field-selector=status.phase!=Running --no-headers 2> /dev/null | wc -l)
            if [ "$UNHEALTHY_PODS" -ne 0 ]; then
              echo "Deployment issue detected. Rolling back..."
              # Restore the backup config
              if [ -f /home/ubuntu/kubernetes/demo-app-config-backup.yaml ];
then
                kubectl apply -f
/home/ubuntu/kubernetes/demo-app-config-backup.yaml
              else
                echo "No backup configuration found. Unable to rollback."
                exit 1
              fi
            else
              echo "Deployment is healthy, no rollback necessary."
            fi

      - name: Generate Kubernetes Config
        run: |
          make generate-k8s-config

      - name: Initialize Environment
        uses: ./.github/actions/environment-setup

      - name: Transfer Config to EC2 Server
        uses: appleboy/scp-action@v0.1.2
        with:
          host: ${{ secrets.AWS_EC2_HOST }}
          username: ubuntu
          key: ${{ secrets.AWS_EC2_SSH_KEY }}
          source: "./kubernetes/demo-app-config.yaml"
```

```
          target: "/home/ubuntu/"

      - name: Apply Kubernetes Config via SSH
        uses: appleboy/ssh-action@v0.1.3
        with:
          host: ${{ secrets.AWS_EC2_HOST }}
          username: ubuntu
          key: ${{ secrets.AWS_EC2_SSH_KEY }}
          script: |
            # Backup current config file
            if [ -f /home/ubuntu/kubernetes/demo-app-config.yaml ]; then
              cp /home/ubuntu/kubernetes/demo-app-config.yaml
/home/ubuntu/kubernetes/demo-app-config-backup.yaml
            fi

            # Deploy new config to the cluster
            KUBECONFIG="/home/ubuntu/.kube/config" kubectl apply --namespace
demo-app -f /home/ubuntu/kubernetes/demo-app-config.yaml

            # Wait for deployment completion
            KUBECONFIG="/home/ubuntu/.kube/config" kubectl rollout status
deployment/demo-app-backend -n demo-app --timeout=30s
```

Conclusion:

This project has highlighted the tremendous value of automation in software development, especially through CI/CD pipelines. We gained practical experience automating key stages of the application lifecycle, from building container images to running tests and deploying the application to a Kubernetes cluster. Setting up a rollback mechanism taught us the critical role of ensuring high availability and minimizing downtime in production environments. We also emphasized the importance of securely managing sensitive data within the pipeline, safeguarding secrets and credentials. Additionally, we deepened our understanding of how tools like GitHub Actions, container registries, and Kubernetes integrate to streamline the development workflow. As a result, this experience has enhanced our ability to build robust, scalable, and reliable deployment pipelines, equipping us with the skills to adapt to changes quickly while ensuring the stability of our applications.

REFERENCES

https://docs.aws.amazon.com/eks/

https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/

https://kubernetes.io/docs/home/

https://kubernetes.io/docs/concepts/workloads/pods/

https://helm.sh/docs/

https://prometheus.io/docs/

https://grafana.com/docs/

https://docs.docker.com/

https://kubernetes.io/docs/tasks/debug/debug-cluster/prometheus/

https://docs.aws.amazon.com/codepipeline/latest/userguide/pipelines.html

https://www.eksworkshop.com/

https://kubernetes.io/docs/concepts/

https://helm.sh/docs/topics/charts/

https://prometheus-operator.dev/docs/

https://grafana.com/docs/grafana-cloud/monitor-infrastructure/kubernetes-monitoring/

https://docs.docker.com/desktop/kubernetes/

https://aws.amazon.com/blogs/devops/

https://kubernetes.io/docs/reference/kubernetes-api/

https://helm.sh/docs/chart_best_practices/

https://grafana.com/docs/grafana-cloud/monitor-infrastructure/kubernetes-monitoring/configuration/config-other-methods/prometheus/remote-write-helm-prometheus/

https://semaphoreci.com/blog/prometheus-grafana-kubernetes-helm

https://github.com/prometheus-community/helm-charts/blob/main/charts/kube-prometheus-stack/README.md

https://thelinuxnotes.com/index.php/how-to-set-up-prometheus-and-grafana-on-kubernetes-with-helm-charts/

https://dev.to/prodevopsguytech/devops-project-cicd-pipeline-for-a-microservices-based-application-on-kubernetes-1ba8

https://www.slingacademy.com/article/how-to-set-up-ci-cd-pipeline-in-kubernetes/

https://devtron.ai/blog/ci-cd-pipeline-for-kubernetes/

https://komodor.com/blog/ci-cd-pipelines-for-kubernetes-best-practices-and-tools/

https://www.codingdrills.com/tutorial/docker-and-kubernetes-tutorial/k8s-ci-cd-pipelines

https://spacelift.io/blog/kubernetes-ci-cd

https://thenewstack.io/kubernetes-ci-cd-pipelines-explained/

https://dev.to/akshat_gautam/building-an-enterprise-cicd-pipeline-with-jenkins-docker-trivy-and-gke-5fcg

https://dzone.com/articles/building-a-cicd-pipeline-with-kubernetes

https://dev.to/mrcaption49/cicd-terraform-docker-and-kubernetes-2kl4

https://devopstraininginpune.com/integrating-docker-with-kubernetes-for-efficient-ci-cd-pipelines/

https://circleci.com/blog/deploy-web-apps-on-kubernetes-with-ci/