UNIVERSITY OF MARYLAND

ENPM 818N : CLOUD COMPUTING

GROUP 9 Final Project Report

SPRING 2025

Link to Project Files : GitHub Link

Bolla Sai Saketh

Fahad Shaker

Shiv Ramolia

Swaraj M Rao

Table of Contents

Project Overview

This AWS-based observability platform was designed to demonstrate a scalable, resilient, and production-grade deployment of the OpenTelemetry Demo Application using Amazon EKS. The project aimed to integrate modern DevOps workflows, Kubernetes-native monitoring, and cloud infrastructure best practices to emulate telemetry pipelines.

To begin, a highly available Virtual Private Cloud (VPC) was provisioned using AWS CloudFormation. The VPC architecture included public and private subnets across two Availability Zones, an Internet Gateway, and dual NAT Gateways for fault-tolerant internet access from private resources. Subnets were strategically assigned to support EKS control plane communication, frontend ingress, and internal monitoring services.

Security was established using a layered approach. Dedicated security groups were created for EKS control plane, worker nodes, admin EC2 access, frontend ingress, and monitoring dashboards. These groups restricted network access to essential ports and CIDR ranges, ensuring secure communication paths between services while allowing external access where needed (e.g., HTTP/HTTPS for the frontend).

Docker was used initially on a standalone EC2 instance to validate the application setup using the provided docker-compose.yaml. This step confirmed container orchestration, internal service connectivity, and basic functionality of endpoints like /, /cart, and /products.

The validated setup was then transitioned to Kubernetes. An Amazon EKS cluster was deployed with two t3.xlarge worker nodes, spread across private subnets for high availability. A separate EC2 instance was configured with kubectl, eksctl, and AWS CLI to serve as the cluster administrator node. Application resources were applied using the provided OpenTelemetry manifest file, and successful pod deployments and service accessibility were verified within the otel-demo namespace.

To enhance maintainability and upgradeability, Helm was introduced in the next phase. The OpenTelemetry Helm repository was added, and the application was re-deployed using Helm charts into a separate namespace. This allowed for parameter overrides, simplified upgrades, and rollbacks. Helm's package management capabilities also made it easier to manage complex sets of Kubernetes resources as a unified release.

Monitoring and alerting were implemented using Prometheus and Grafana, deployed within the cluster. Custom Prometheus alert rules were configured to trigger based on pod restart thresholds. Alert notifications were integrated with email using SMTP to notify operators in real time about service-level anomalies.

To close the DevOps loop, a CI/CD pipeline was built using GitHub Actions. The pipeline automated Docker image builds, pushed them to a container registry, and applied changes to the Kubernetes cluster upon successful test completion. Failure handling was addressed using Helm rollback functionality built into the workflow, ensuring the system could safely revert to a stable state if a deployment failed. Secrets were securely managed using GitHub Secrets for tokens, credentials, and kubeconfig files.

Overall, this project effectively demonstrated how a real-world, microservices-based observability platform can be deployed and managed on AWS using EKS, Helm, Prometheus, and CI/CD pipelines. Each phase of the project reflected best practices in infrastructure-as-code, secure cloud deployment, and Kubernetes-native application lifecycle management.

Link to Project Files: GitHub Link

Infrastructure Setup

This report outlines the planning, deployment, and validation of a production-grade observability platform using OpenTelemetry on AWS. The project focused on leveraging cloud-native technologies to build a scalable, resilient, and secure microservices architecture, emphasizing best practices in infrastructure automation, Kubernetes operations, and performance monitoring.

Key AWS services included Amazon EKS for container orchestration, CloudFormation for infrastructure provisioning, and EC2 for administrative control and testing. Helm was used to streamline Kubernetes resource management, while Prometheus and Grafana enabled robust observability and alerting. The system was further enhanced with a CI/CD pipeline built on GitHub Actions to automate application delivery and rollback. Together, these components provided a reliable, extensible, and cost-conscious framework suitable for real-world telemetry workloads.

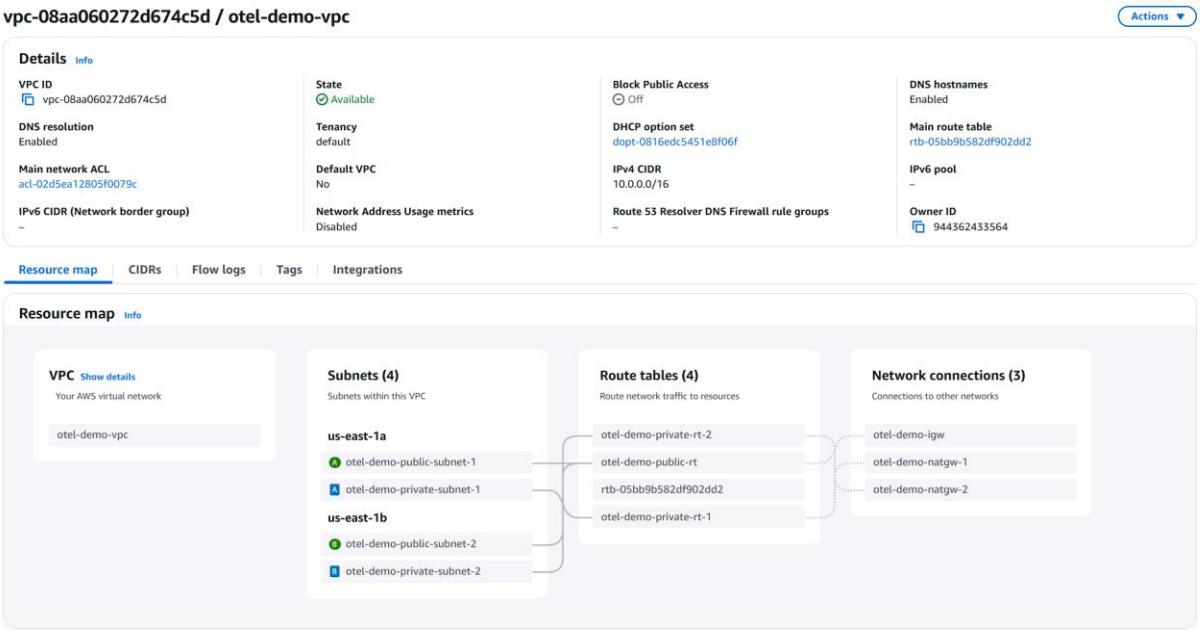
Overview of the Infrastructure and Implementation

This architecture is designed to support a secure, scalable, and resilient observability platform using Kubernetes on AWS. It leverages a suite of AWS services to manage networking, compute resources, application orchestration, monitoring, and security. Core components include a highly available VPC, an Amazon EKS cluster with worker nodes distributed across multiple subnets, an EC2-based admin node for cluster control, and secure connectivity managed via custom security groups. Additional layers include Helm-based deployment automation, Prometheus-Grafana for observability, and GitHub Actions CI/CD for continuous delivery—each contributing to the reliability, traceability, and maintainability of the system.

1. VPC and Networking Structure

To ensure high availability, fault tolerance, and secure segmentation of services, the foundational network infrastructure for the OpenTelemetry EKS project was provisioned entirely using AWS CloudFormation. The design includes a multi-AZ VPC, route tables, NAT gateways, and fine-tuned security groups to enforce controlled communication paths between components.

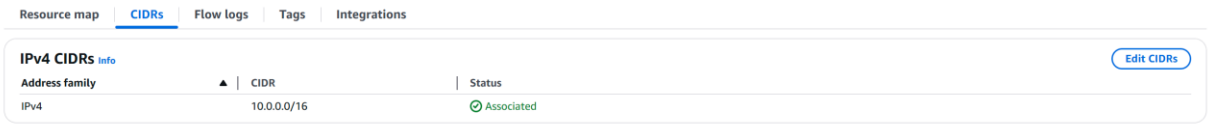
Link to the VPC CloudFormation files : GitHub Link



1.1 CIDR Block

The entire architecture is encapsulated within a dedicated Virtual Private Cloud (VPC) with the CIDR block:

- **10.0.0.0/16 —** This large address range **allows for multiple subnets, scalable node groups, and future network expansion across availability zones**.



1.2 Multi-AZ VPC Design

High availability is achieved by distributing subnets and workloads across two Availability Zones (AZs). The architecture includes:

- **Public Subnet 1**: 10.0.1.0/24 in **AZ1**
- **Public Subnet 2**: 10.0.2.0/24 in **AZ2**
- **Private Subnet 1**: 10.0.3.0/24 in **AZ1**
- **Private Subnet 2**: 10.0.4.0/24 in **AZ2**

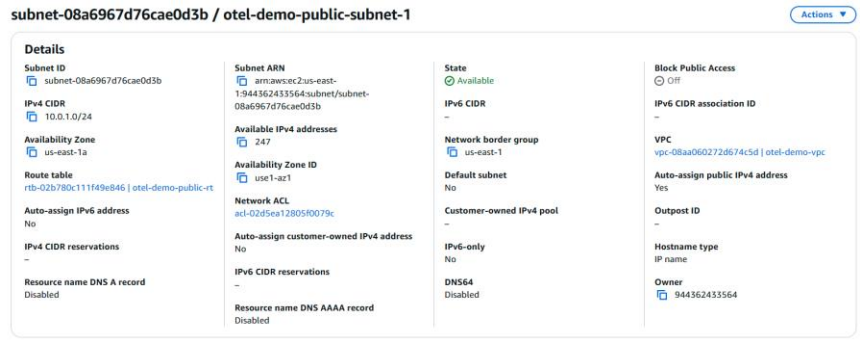
This ensures workload distribution for services like EKS worker nodes and public-facing resources such as NAT Gateways or Load Balancers.

1.3 Subnets

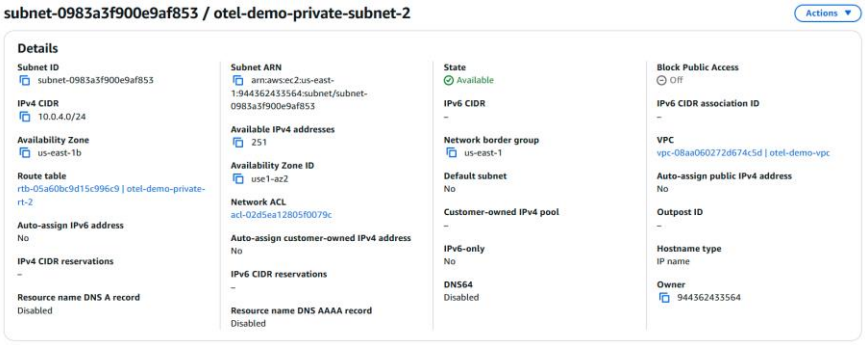
The subnets were designed as follows:

- **Public Subnets** are configured with **MapPublicIpOnLaunch**: true to allow resources (e.g., NAT Gateways) to be publicly accessible.
- **Private Subnets** host sensitive components like **EKS worker nodes** and monitoring tools, **keeping them isolated from the public internet**.

All subnets are tagged for identification and grouped by functionality.



Public Subnet



Private Subnet

1.4 Route Tables

Custom route tables were configured and associated with the subnets to define the traffic flow:

- **Public Route Table** routes **0.0.0.0/0 traffic through an Internet Gateway**, attached to the VPC.
- **Private Route Tables** (1 and 2) route outbound internet traffic **via corresponding NAT Gateways**, allowing private instances to access external services securely without being exposed directly.

Each subnet is associated with its appropriate route table to ensure correct routing.

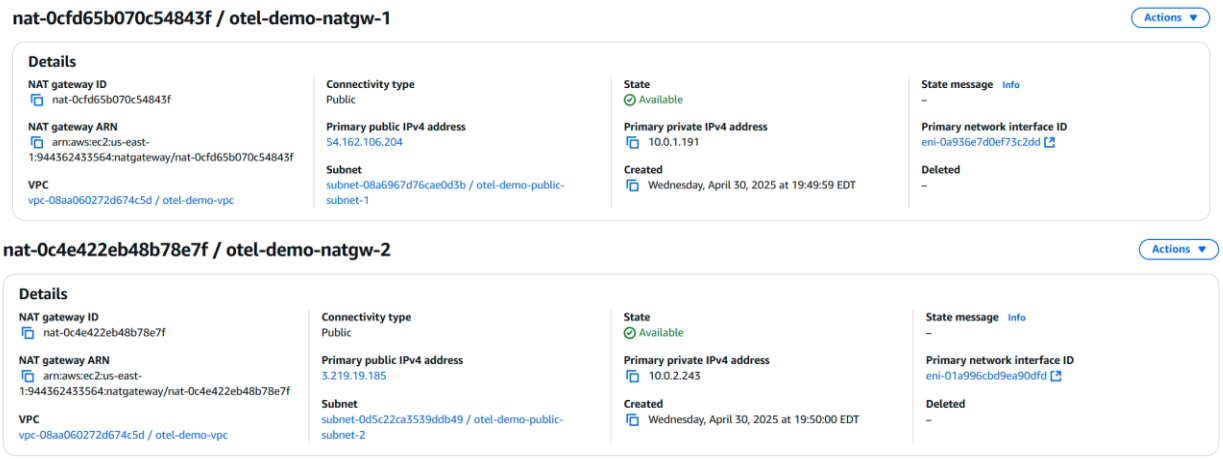
☐	otel-demo-private-rt-1	rtb-057e0f897dbbaf922	subnet-0c2b65966e115d...	–	No	vpc-08aa060272d674c5d | otel...	944362433564
☐	otel-demo-private-rt-2	rtb-05a60bc9d15c996c9	subnet-0983a3f900e9af8...	–	No	vpc-08aa060272d674c5d | otel...	944362433564
☐	otel-demo-public-rt	rtb-02b780c111f49e846	2 subnets	–	No	vpc-08aa060272d674c5d | otel...	944362433564

1.5 NAT Gateways and Elastic Ips

To support outbound internet access from private subnets:

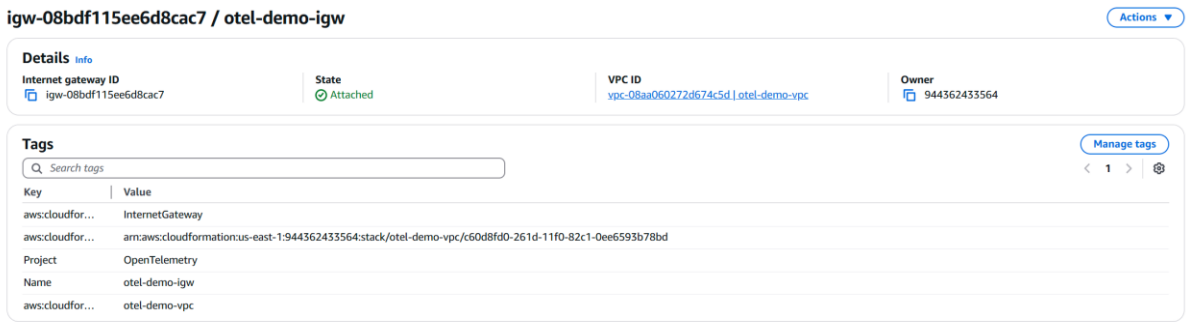
- **Two NAT Gateways** were provisioned, one **in each public subnet**.
- Each NAT Gateway uses an **Elastic IP allocated** in advance for stable public addressing.

This setup guarantees continued internet access in case of AZ-specific outages.



1.6 Internet Gateway

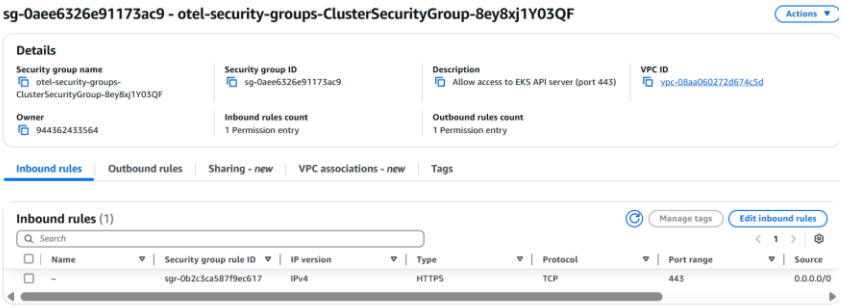
An Internet Gateway was attached to the VPC to enable internet connectivity for resources in the public subnets, such as NAT Gateways and load balancer endpoints.



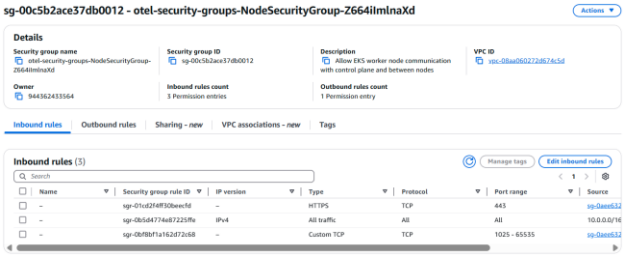
1.7 Security Groups

Security was enforced using purpose-specific Security Groups, provisioned via a separate **CloudFormation template**. These include:

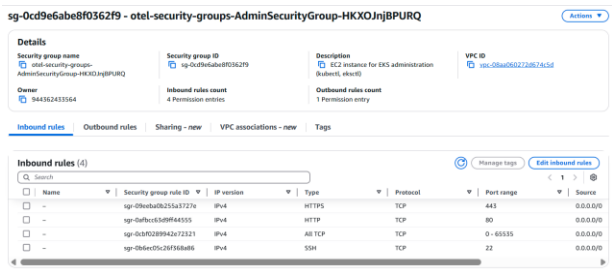
- **EKS Cluster Security Group**: Allows secure access to the Kubernetes API server on port 443.



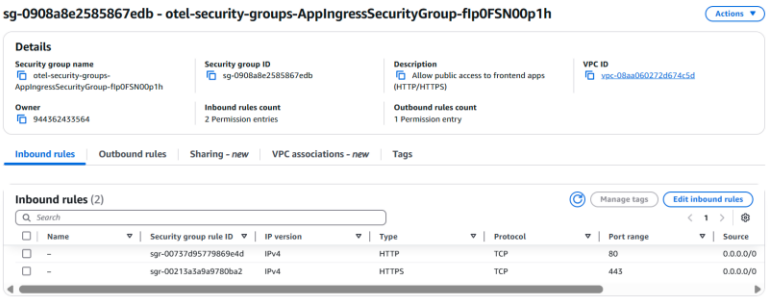
- **Node Security Group**: Enables inter-node communication and traffic from the cluster control plane.



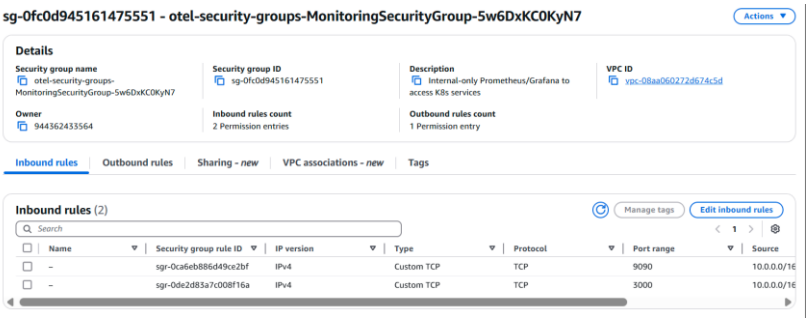
- **Admin EC2 Security Group**: Grants SSH (22) and HTTPS (443) access for the EKS administrator instance.



- **Application Ingress Security Group**: Allows HTTP (80) and HTTPS (443) traffic to frontend services.



- **Monitoring Security Group**: Internal-only access to Prometheus (9090) and Grafana (3000) dashboards from within the VPC.



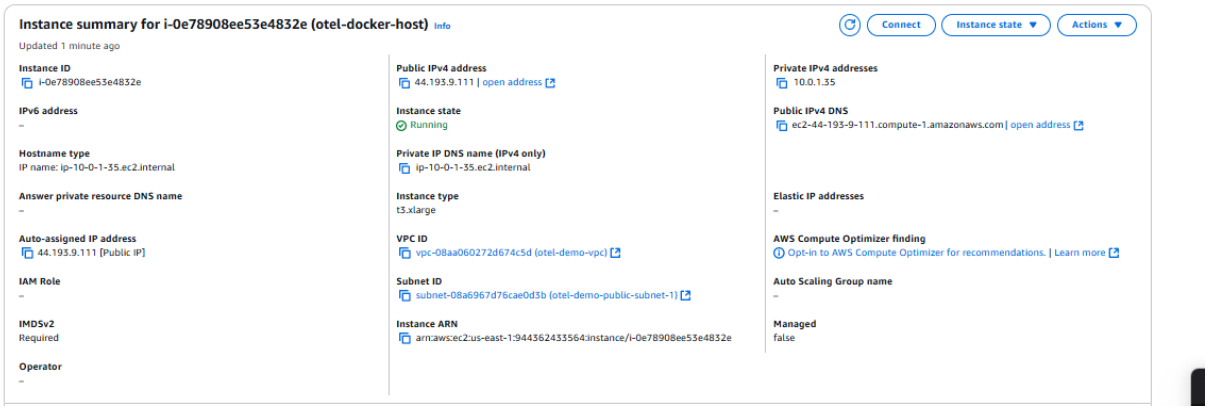
Phase 1

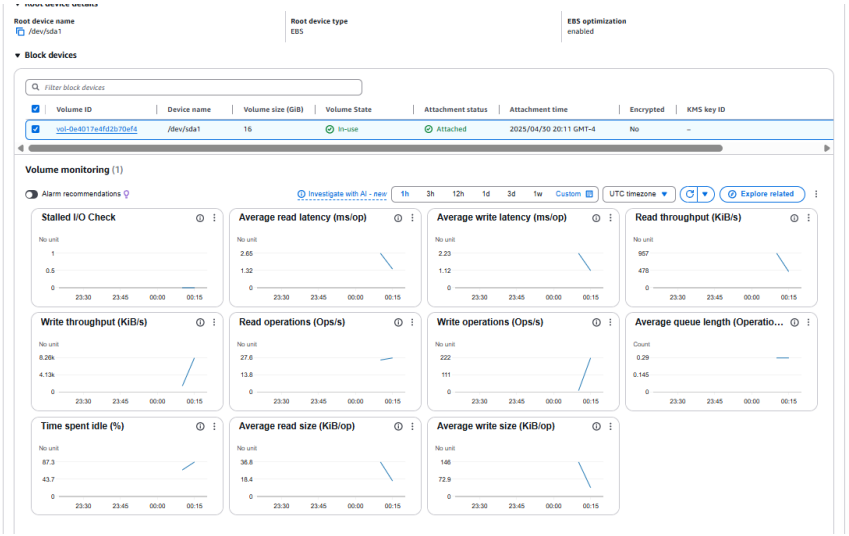
Part 1 : Docker Deployment

As a first step in validating the OpenTelemetry Demo Application, a standalone EC2 instance was provisioned to deploy the application using Docker. This approach enabled isolated testing of all services defined in the official docker-compose.yaml prior to Kubernetes integration.

1.1 EC2 Instance Setup

To begin the project, a dedicated EC2 instance was provisioned on AWS to host the OpenTelemetry demo application using Docker. The instance used was of type **t3.large**, meeting the requirement with **4 vCPU and 16.0 GiB's of memory**, with an **EBS volume attached and configured with at least 16GB of storage**. For operating system Ubuntu was selected(**Ubuntu Server 24.04 LTS**), and Docker along with Docker Compose were installed. The current user was added to the docker group, allowing Docker commands to be executed without elevated privileges.





1.2 Application Deployment with Docker Compose

The OpenTelemetry demo application was deployed using the docker-compose.yml file from the official OpenTelemetry GitHub repository.

Deployment steps included:

- Cloning the repository using:

  git clone https://github.com/open-telemetry/opentelemetry-demo.git

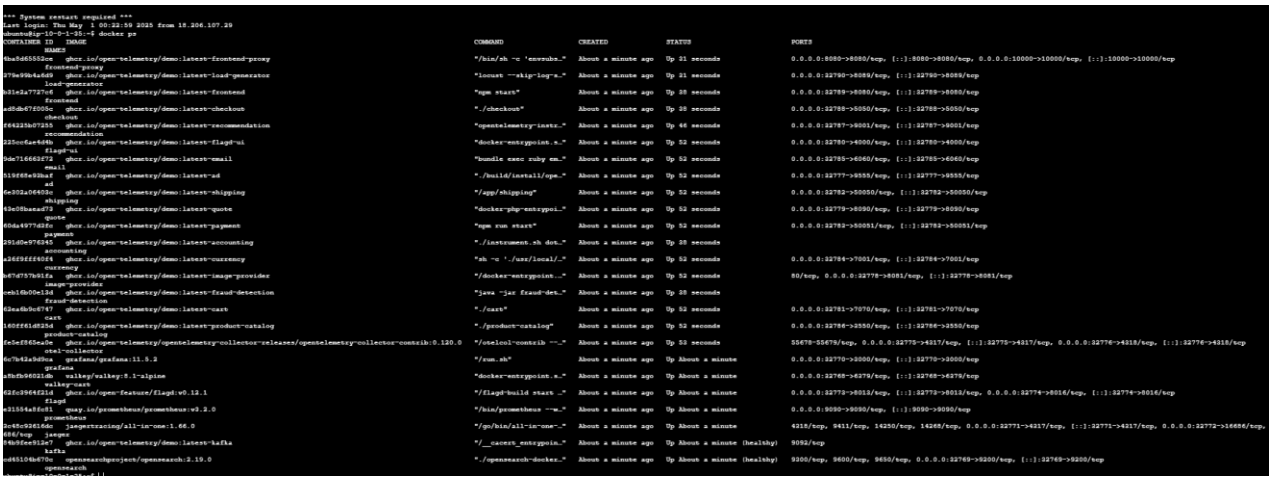
- Navigating to the directory and deploying the application stack using :

  cd opentelemetry-demo

  docker compose up -d

- This brought up all essential services including:
  - frontend
  - loadgen
  - checkoutservice
  - productcatalogservice
  - otel-collector
  - jaeger
  - grafana
  - several backend microservices

Container health and status were verified using: **docker ps**

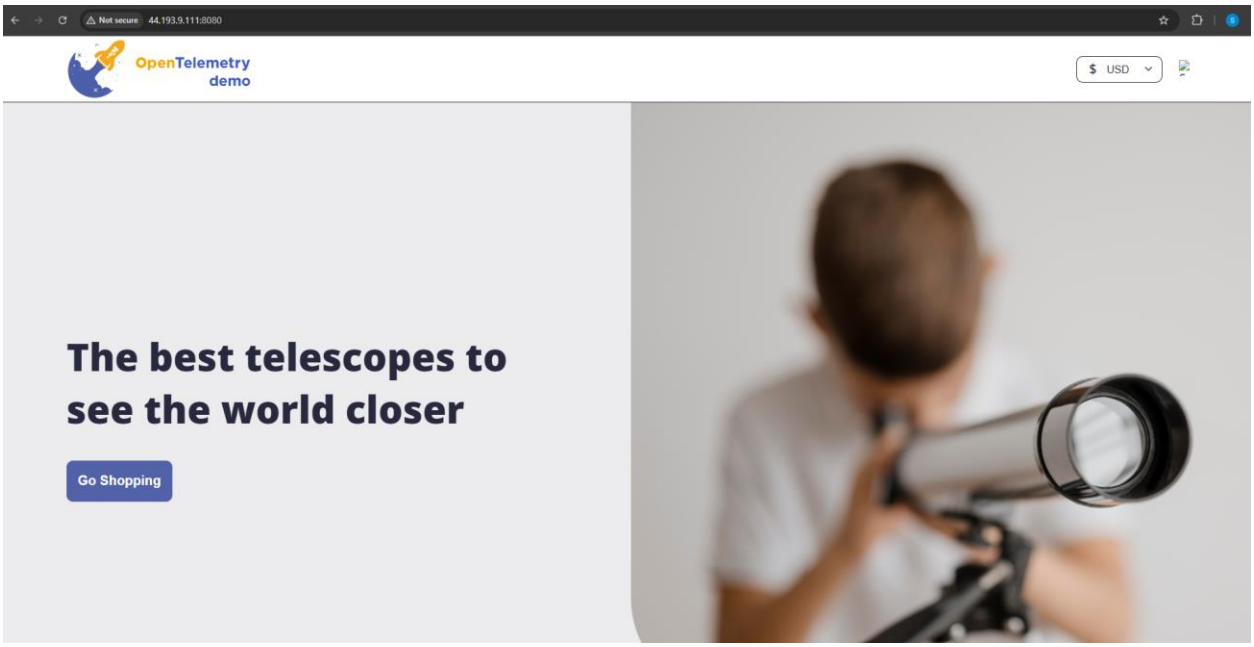


1.3 Validation and Observability

To ensure the services had started correctly and without error, the docker-compose logs command was used to view the logs from the application's startup sequence. The logs showed that services initialized in the proper order, confirming that dependencies were resolved and telemetry components (like metrics and traces) were being generated and collected.

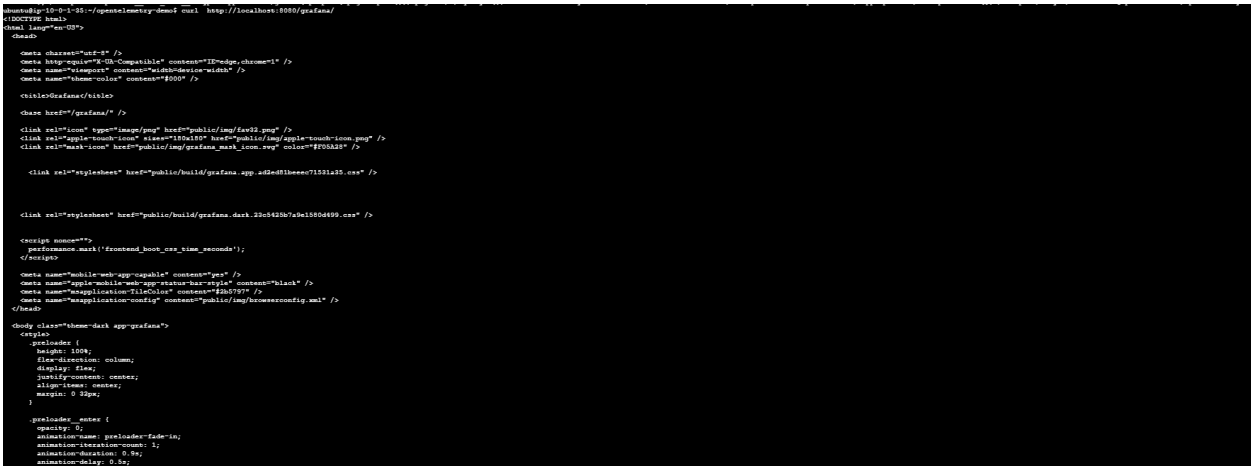
Once all services were verified to be running, the application was accessed through its exposed endpoints to validate user-facing and monitoring components. The frontend application was reachable at http://<EC2_PUBLIC_IP>:8080/, confirming that the user interface loaded correctly and responded to user interaction.

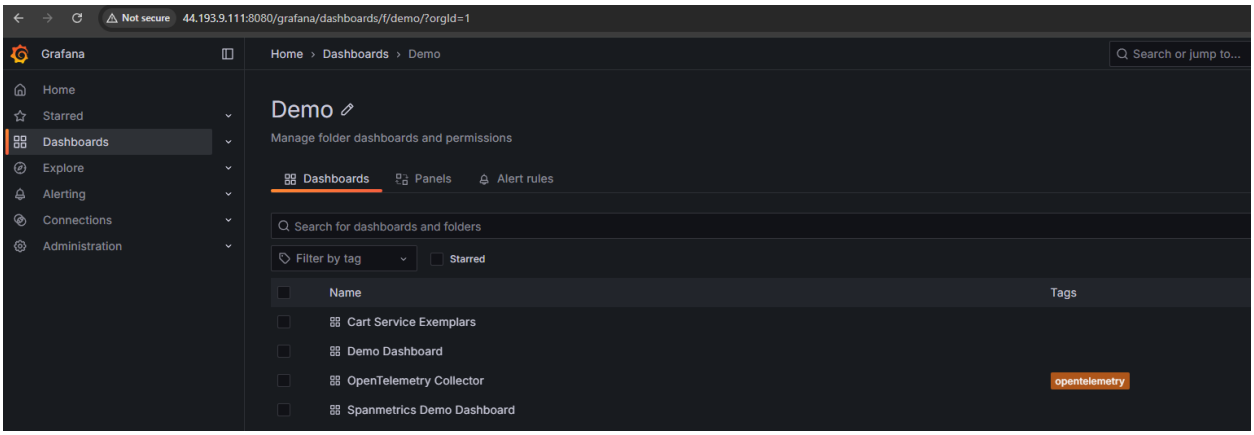
Web store



Grafana

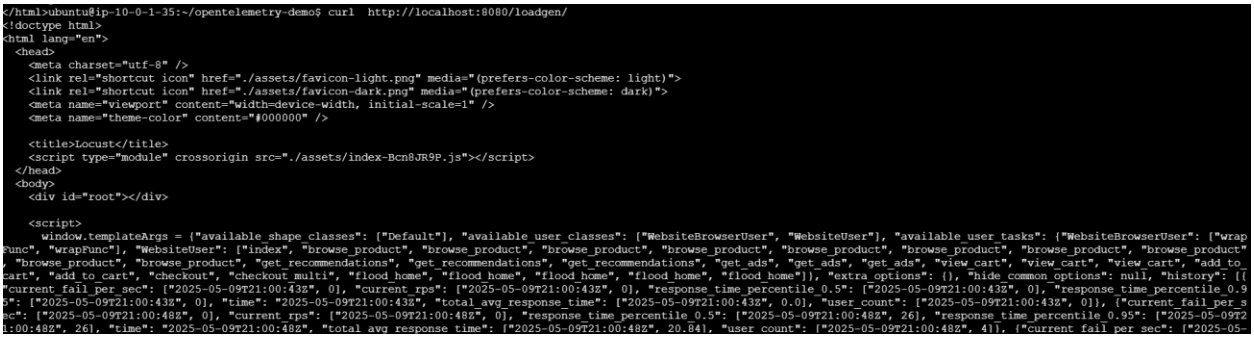
Additionally, the observability interface (Grafana) was tested at http://<EC2_PUBLIC_IP>:8080/grafana/, where dashboards and metrics visualizations were available.

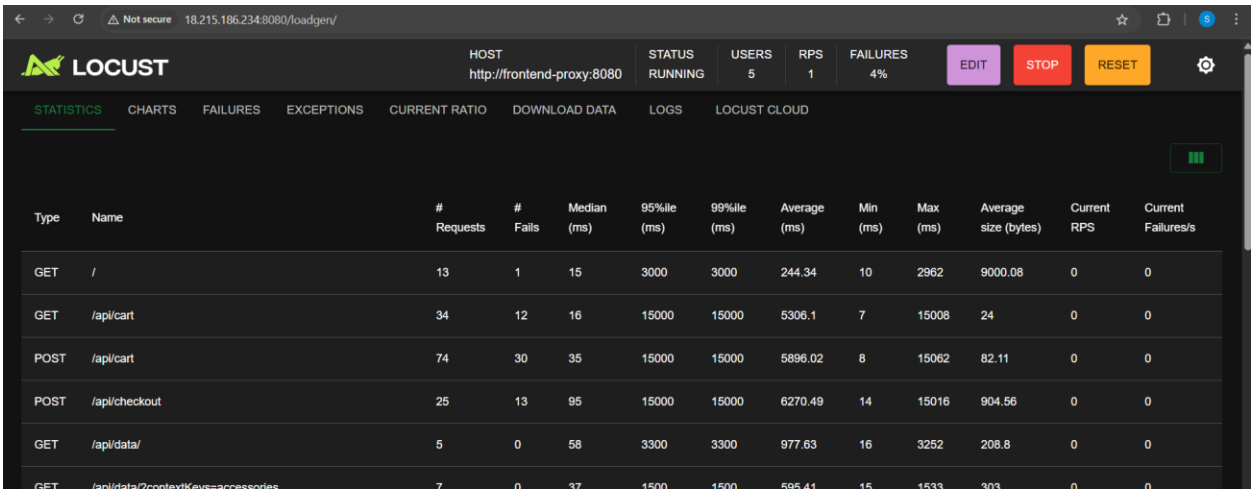




Additional screenshots were captured from the related UIs to show the working telemetry features, such as traces, logs, and metrics. These included views of the load generator service (loadgen) and services such as the product catalog and checkout microservices. Traces shown in the dashboards validated that the telemetry backend was receiving data and that distributed tracing was properly implemented.

Loadgen

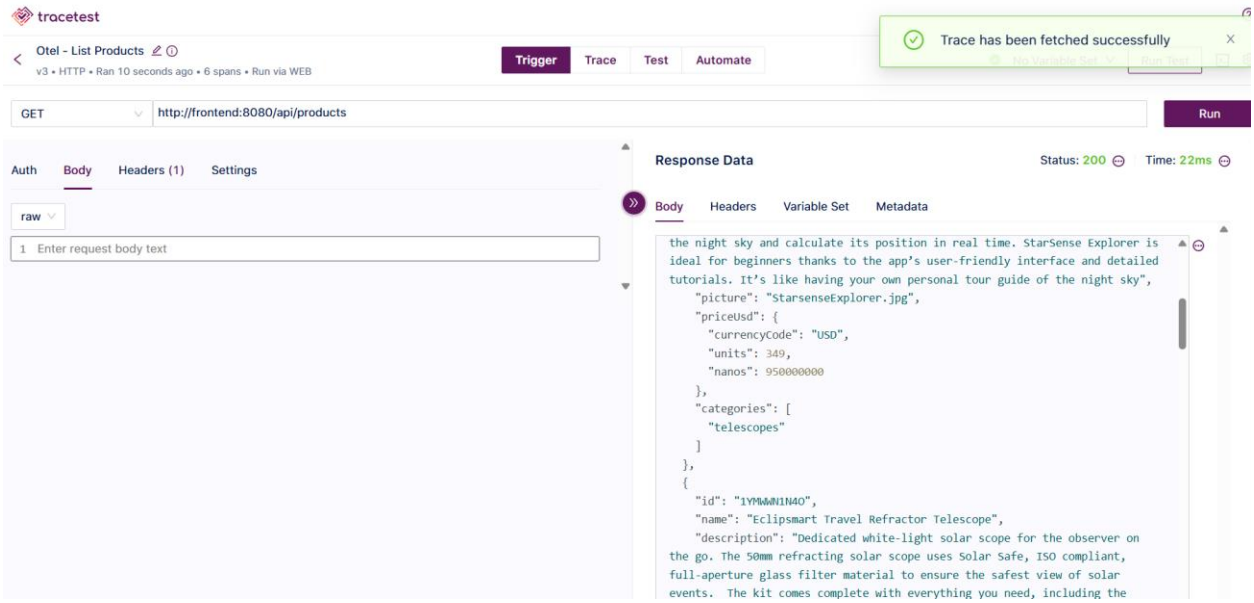




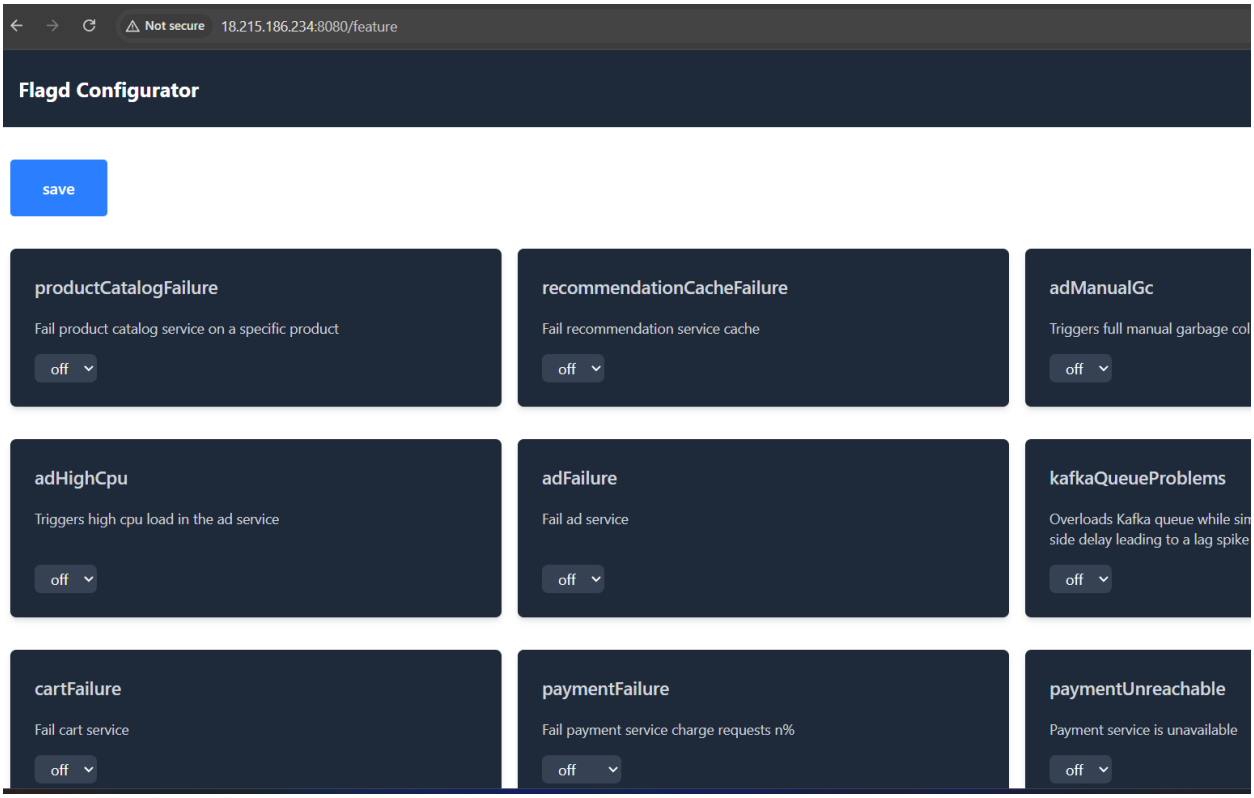
Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	13	1	15	3000	3000	244.34	10	2962	9000.08	0	0
GET	/api/cart	34	12	16	15000	15000	5306.1	7	15008	24	0	0
POST	/api/cart	74	30	35	15000	15000	5896.02	8	15062	82.11	0	0
POST	/api/checkout	25	13	95	15000	15000	6270.49	14	15016	904.56	0	0
GET	/api/data/	5	0	58	3300	3300	977.63	16	3252	208.8	0	0
GET	/api/data/?contextKeys=accessories	7	0	37	1500	1500	595.41	15	1533	303	0	0

Tracing

```
ubuntu@ip-10-0-1-35:~/opentelemetry-demo$ docker compose -f docker-compose-tests.yml run traceBasedTests
WARN[0000] Found orphan containers ([opentelemetry-demo-traceBasedTests-run-26ed345d89ca]) for this project. If you removed or re
command with the --remove-orphans flag to clean it up.
[+] Creating 22/22
 ✓ Container tracetest-postgres          Running
 ✓ Container cd45104b670c_opensearch     Running
 ✓ Container 84b9fee912e7_kafka          Running
 ✓ Container a8bfb96021db_valkey-cart    Running
 ✓ Container 3c48c93616dc_jaeger         Running
 ✓ Container 62fc3964f21d_flagd          Running
 ✓ Container otel-collector              Running
 ✓ Container accounting                  Running
 ✓ Container ad                          Running
 ✓ Container cart                        Running
 ✓ Container payment                     Running
 ✓ Container email                       Running
 ✓ Container shipping                    Running
 ✓ Container tracetest-server            Running
 ✓ Container image-provider              Running
 ✓ Container product-catalog             Running
 ✓ Container recommendation              Running
 ✓ Container quote                       Running
 ✓ Container currency                    Running
 ✓ Container checkout                    Running
 ✓ Container frontend                    Running
 ✓ Container fraud-detection             Running
[+] Running 3/3
 ✓ Container cd45104b670c_opensearch     Healthy
 ✓ Container 84b9fee912e7_kafka          Healthy
 ✓ Container tracetest-postgres          Healthy
Starting tests...
Running trace-based tests for: ad cart currency checkout frontend email payment product-catalog recommendation shipping ...
```



Feature



1.4 Cleanup

After validating the deployment, the EC2 resources were deprovisioned. The stack was shut down using: **docker compose down.** The EC2 instance was terminated to avoid incurring ongoing costs.

Part 2: Kubernetes Deployment on Amazon EKS

After validating the Docker-based deployment, the application was migrated to a Kubernetes environment using Amazon EKS (Elastic Kubernetes Service). This phase focused on establishing a secure and scalable container orchestration platform to manage OpenTelemetry's distributed microservices at scale.

2.1 EKS Cluster Provisioning

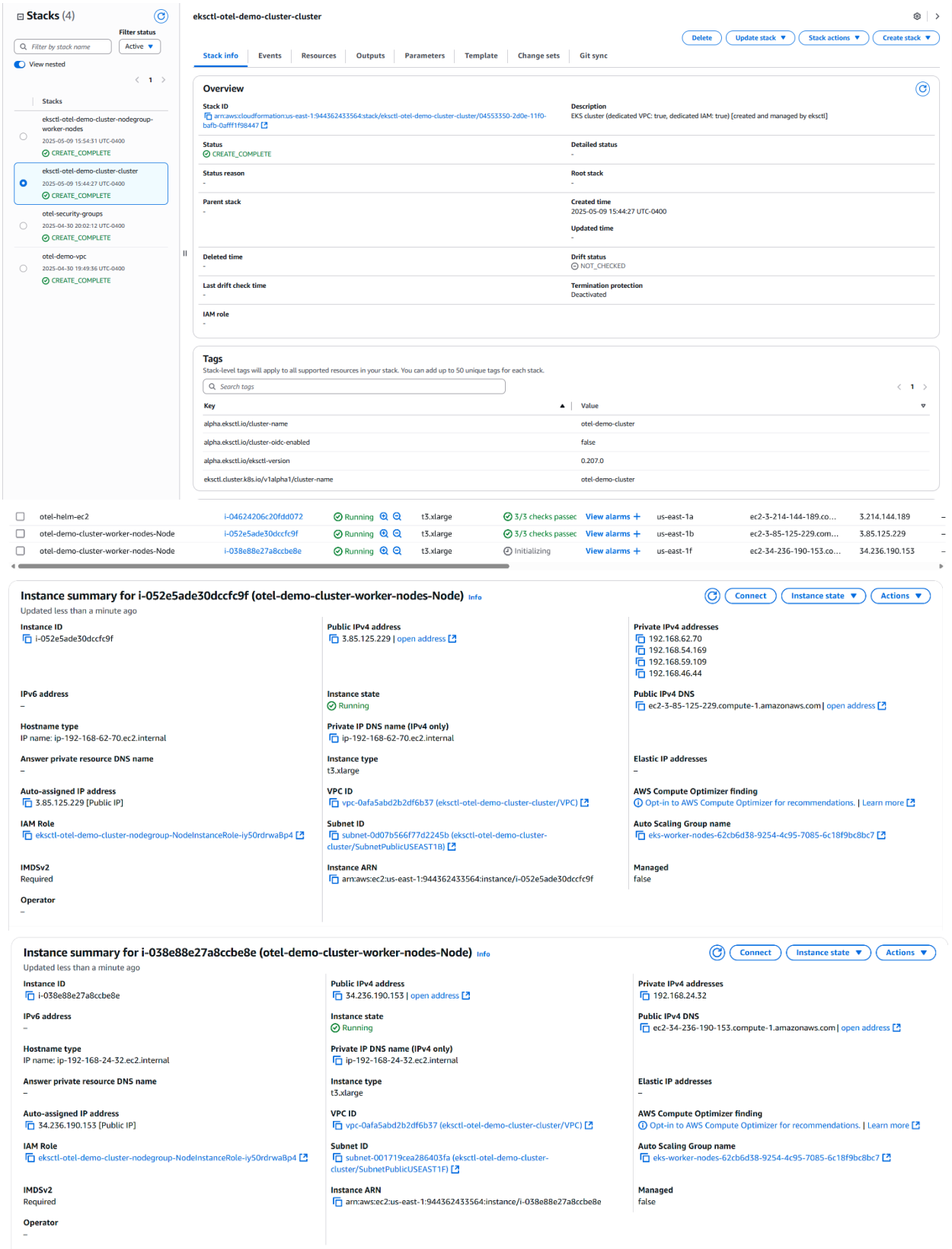
The Amazon EKS cluster was created using eksctl, a CLI tool that simplifies EKS management.

- **Cluster Configuration:**
  - Cluster Name: **otel-demo-cluster**
  - Region: **us-east-1**
  - Node Group Name: **worker-nodes**
  - Node Type: **t3.xlarge**
  - Desired Nodes: **2**
  - Min/Max Nodes: **2**
  - Node Management: **Managed Node Group**

Command Used to Set up an EKS Cluster in AWS (run from otel-kubernetes-eks EC2 instance) :

```
eksctl create cluster \
 --name otel-demo-cluster \
 --region us-east-1 \
 --nodegroup-name worker-nodes \
 --node-type t3.xlarge \
 --nodes 2 \
 --nodes-min 2 \
 --nodes-max 2 \
 --managed
```

The cluster was provisioned into the previously created VPC and private subnets to isolate workloads from direct internet exposure.



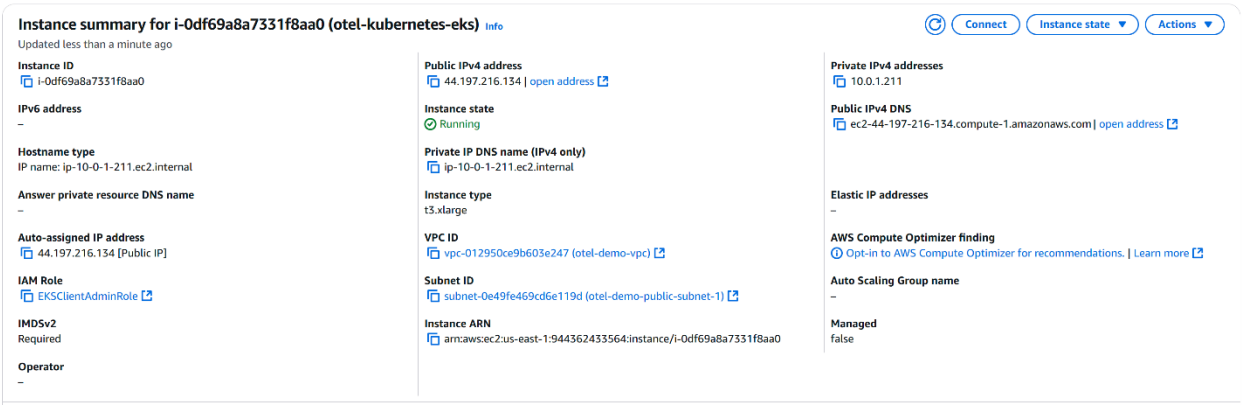
EKS Cluster Provisioning and Node Deployment Overview

2.2 Admin Node Setup

A separate EC2 admin node (Ubuntu 24.04) was provisioned to serve as the Kubernetes client. The following tools were installed:

- kubectl – for interacting with the Kubernetes cluster
- eksctl – for cluster management
- aws-cli – for authentication and role-based access

The kubeconfig was automatically configured by eksctl, allowing kubectl to execute commands on the cluster.



Screenshot of the EC2 instance used as the EKS client

2.3 Application Deployment to EKS

The OpenTelemetry demo application was deployed using the provided opentelemetry-demo.yaml manifest. This file defines multiple Kubernetes resources, including Deployments, Services, ConfigMaps, and persistent volumes for the demo environment.

The original opentelemetry-demo.yaml file provided by the OpenTelemetry Demo repository contains the full definition of all Kubernetes resources needed to deploy the application. However, **the file was too large for Kubernetes to process directly**, and applying it triggered an error related to resource limits—specifically around ConfigMap size exceeding the Kubernetes annotation limit (262144 bytes).

To resolve this and improve maintainability, **the file was systematically broken down into multiple smaller, logically grouped manifest files, each representing a distinct part of the application stack**. This modularization not only addressed the deployment issue but also aligned with best practices in infrastructure management and GitOps workflows.

The modified/split yaml files can be found here : GitHub Link

Description of the broken down files :

Category	Description	File(s)
Namespace	Defines the isolated otel-demo namespace for all resources.	namespaces.yaml
RBAC (Access Control)	Grants roles and permissions to components like Grafana, Jaeger, and Prometheus.	roles.yaml, rolebindings.yaml, clusterroles.yaml, clusterrolebindings.yaml
Service Accounts	Declares dedicated service accounts for telemetry services.	serviceaccounts.yaml
Secrets	Stores sensitive credentials, such as Grafana admin credentials.	secrets.yaml
ConfigMaps	Holds configurations for observability tools (Prometheus, OTEL Collector, OpenSearch, etc.).	prometheus.yaml, otel-collector.yaml, opensearch-config.yaml, flagd-config.yaml, product-catalog-products.yaml
Deployments	Stateless workloads for frontend UI, load generation, tracing, and collector services.	deployments.yaml
StatefulSets	Manages OpenSearch pods that require stable network IDs and persistent storage.	statefulsets.yaml
PodDisruptionBudgets (PDBs)	Ensures workload availability during node upgrades or maintenance.	poddisruptionbudgets.yaml
Services	Provides internal and cluster-wide access to deployed services via stable DNS.	services.yaml
Dashboards	Prebuilt Grafana dashboards for service metrics, tracing, and RED analysis.	grafana_dashboard1.yaml to grafana_dashboard4.yaml
Grafana Configuration	Sets up Grafana deployment, service exposure, and credentials.	grafana.yaml

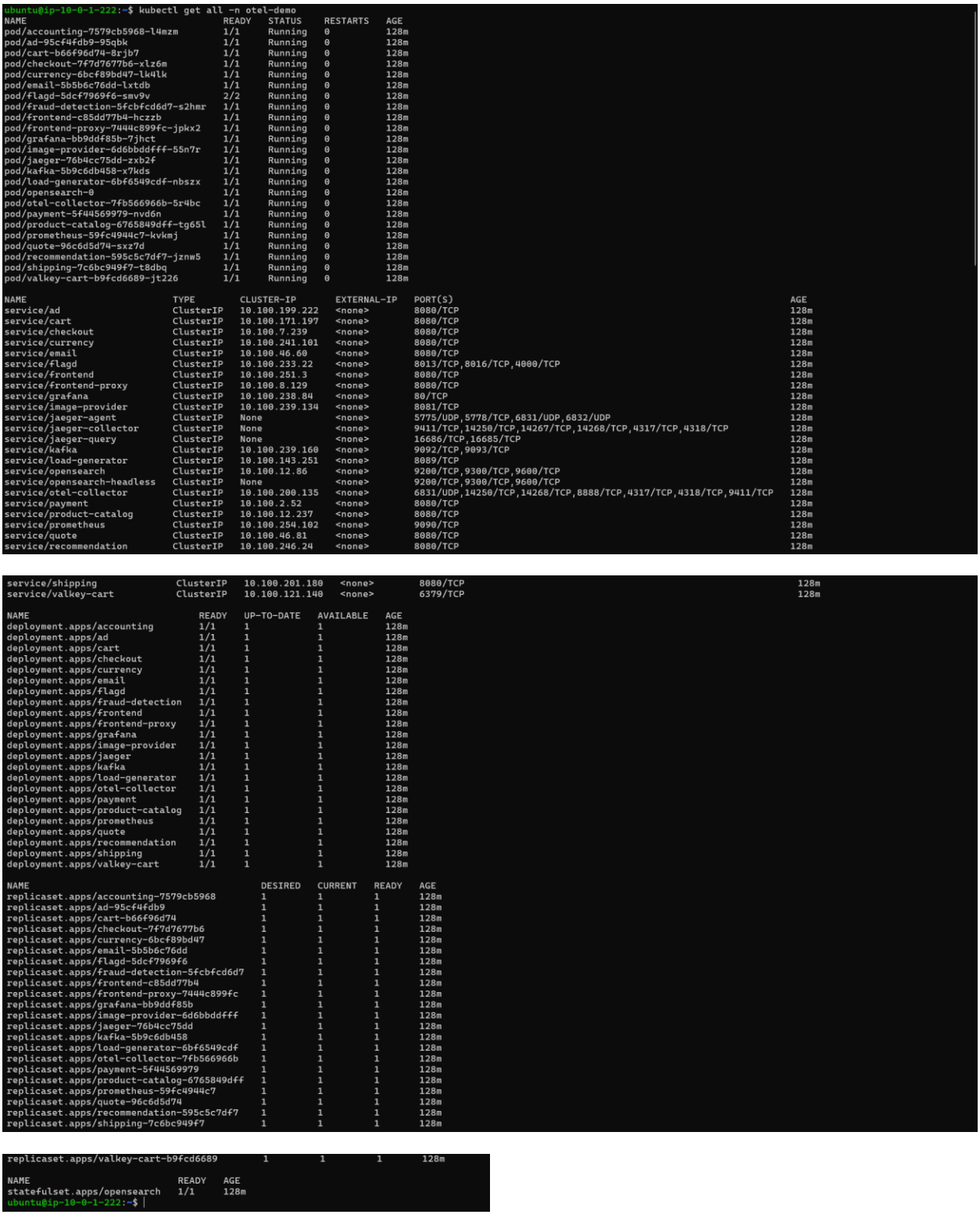
After splitting the opentelemetry-demo.yaml into modular manifest files, the resources were applied in a structured and namespace-aware order to ensure proper initialization of all components.

1. Namespace creation using : **kubectl apply -f namespaces.yaml**

2. All other decomposed Kubernetes resources (RBAC, ConfigMaps, Deployments, Services, etc.) were organized into a directory named kuber_custom/. These were applied within the otel-demo namespace using:

kubectl apply -f kuber_custom/ -n otel-demo

This modular deployment sequence ensured that resources were registered correctly under the appropriate namespace and resolved issues stemming from the oversized monolithic manifest.



Screenshot of kubectl get all -n otel-demo showing the status of pods, services, and deployments

Logs from Key Application Pods to confirm successful deployment:

Frontend

```
ubuntu@ip-10-0-1-222:~$ kubectl logs pod/frontend-c85dd77b4-nsgkd -n otel-demo

> frontend@0.1.0 start
> node --require ./Instrumentation.js server.js

   ▲ Next.js 15.2.1
   - Local:        http://frontend-c85dd77b4-nsgkd:8080
   - Network:      http://frontend-c85dd77b4-nsgkd:8080

 ✓ Starting...
 ✓ Ready in 969ms
ubuntu@ip-10-0-1-222:~$
```

Grafana

```
ubuntu@ip-10-0-1-222:~$ kubectl logs pod/grafana-bb9ddf85b-qm8gz -n otel-demo
✓Downloaded and extracted grafana-opensearch-datasource v2.26.0 zip successfully to /var/lib/grafana/plugins/grafana-opensearch-datasource

Please restart Grafana after installing or removing plugins. Refer to Grafana documentation for instructions if necessary.

logger=settings t=2025-05-13T00:55:17.085529688Z level=info msg="Starting Grafana" version=11.5.2 commit=598e0338d537d6bc404b02a58094132c5eeceb8 branch=HEAD compiled=2025-05-13T00:55:17
Z
logger=settings t=2025-05-13T00:55:17.086123988Z level=info msg="Config loaded from" file=/usr/share/grafana/conf/defaults.ini
logger=settings t=2025-05-13T00:55:17.086140026Z level=info msg="Config loaded from" file=/etc/grafana/grafana.ini
logger=settings t=2025-05-13T00:55:17.086146973Z level=info msg="Config overridden from command line" arg="default.paths.data=/var/lib/grafana/"
logger=settings t=2025-05-13T00:55:17.086153815Z level=info msg="Config overridden from command line" arg="default.paths.logs=/var/log/grafana"
logger=settings t=2025-05-13T00:55:17.086160676Z level=info msg="Config overridden from command line" arg="default.paths.plugins=/var/lib/grafana/plugins"
logger=settings t=2025-05-13T00:55:17.086167458Z level=info msg="Config overridden from command line" arg="default.paths.provisioning=/etc/grafana/provisioning"
logger=settings t=2025-05-13T00:55:17.086174404Z level=info msg="Config overridden from command line" arg="default.log.mode=console"
logger=settings t=2025-05-13T00:55:17.086183007Z level=info msg="Config overridden from Environment variable" var="GF_PATHS_DATA=/var/lib/grafana/"
logger=settings t=2025-05-13T00:55:17.086191168Z level=info msg="Config overridden from Environment variable" var="GF_PATHS_LOGS=/var/log/grafana"
logger=settings t=2025-05-13T00:55:17.086197695Z level=info msg="Config overridden from Environment variable" var="GF_PATHS_PLUGINS=/var/lib/grafana/plugins"
logger=settings t=2025-05-13T00:55:17.086203505Z level=info msg="Config overridden from Environment variable" var="GF_PATHS_PROVISIONING=/etc/grafana/provisioning"
logger=settings t=2025-05-13T00:55:17.086211435Z level=info msg="Config overridden from Environment variable" var="GF_SECURITY_ADMIN_USER=admin"
logger=settings t=2025-05-13T00:55:17.086217838Z level=info msg="Config overridden from Environment variable" var="GF_SECURITY_ADMIN_PASSWORD=*********"
logger=settings t=2025-05-13T00:55:17.086225055Z level=info msg=Target target=[all]
logger=settings t=2025-05-13T00:55:17.086239475Z level=info msg="Path Home" path=/usr/share/grafana
logger=settings t=2025-05-13T00:55:17.086245939Z level=info msg="Path Data" path=/var/lib/grafana/
logger=settings t=2025-05-13T00:55:17.086251809Z level=info msg="Path Logs" path=/var/log/grafana
logger=settings t=2025-05-13T00:55:17.086258018Z level=info msg="Path Plugins" path=/var/lib/grafana/plugins
logger=settings t=2025-05-13T00:55:17.086263525Z level=info msg="Path Provisioning" path=/etc/grafana/provisioning
logger=settings t=2025-05-13T00:55:17.086268982Z level=info msg="App mode production"
logger=featuremgmt t=2025-05-13T00:55:17.091339181Z level=info msg=FeatureToggles dataplaneFrontendFallback=true logsInfiniteScrolling=true useSessionStorageForRedirection=true cloudWatc
hNewLabelParsing=true logsExploreTableVisualisation=true nestedFolders=true kubernetesPlaylists=true promQLScope=true unifiedRequestLog=true cloudWatchCrossAccountQuerying=true lokiLabel
NamesQueryApi=true azureMonitorEnableUserAuth=true alertingSimplifiedRouting=true tlsMemcached=true lokiQueryHints=true angularDeprecationUI=true dashboardSceneSolo=true exploreMetrics=t
rue logsContextDatasourceUi=true recoveryThreshold=true newPDFRendering=true openSearchBackendFlowEnabled=true cloudWatchRoundUpEndTime=true alertingNoDataErrorExecution=true influxdbBac
kendMigration=true publicDashboardsScene=true logRowsPopoverMenu=true pinNavItems=true lokiQuerySplitting=true zipkinBackendMigration=true groupToNestedTableTransformation=true awsAsyncQ
ueryCaching=true recordedQueriesMulti=true managedPluginsInstall=true newFiltersUI=true preinstallAutoUpdate=true transformationsRedesign=true addFieldFromCalculationStatFunctions=true a
lertingApiServer=true annotationPermissionUpdate=true reportingUseRawTimeRange=true ssoSettingsApi=true prometheusAzureOverrideAudience=true accessControlOnCall=true dashboardSceneForVie
wers=true dashboardScene=true transformationsVariableSupport=true dashgpt=true alertingInsights=true alertingUIOptimizeReducer=true userStorageAPI=true accessActionSets=true onPremToClou
dMigrations=true formatString=true correlations=true lokiStructuredMetadata=true newDashboardSharingComponent=true panelMonitoring=true
logger=sqlstore t=2025-05-13T00:55:17.091471825Z level=info msg="Connecting to DB" dbtype=sqlite3
logger=sqlstore t=2025-05-13T00:55:17.091492668Z level=info msg="Creating SQLite database file" path=/var/lib/grafana/grafana.db
```

Payment Service

```
ubuntu@ip-10-0-1-222:~$ kubectl logs pod/payment-5f44569979-tcrsh -n otel-demo

> start
> node --require ./opentelemetry.js index.js

{"level":"info","time":1747097710682,"pid":17,"hostname":"payment-5f44569979-tcrsh","service.name":"payment","msg":"payment gRPC ser
ver started on port 8080"}
{"level":"info","time":1747097815110,"pid":17,"hostname":"payment-5f44569979-tcrsh","trace_id":"5bcb076bc505489da690e802a76609f4","s
pan_id":"864087662c647635","trace_flags":"01","service.name":"payment","request":{"amount":{"currencyCode":"USD","units":{"low":2276
,"high":0,"unsigned":false},"nanos":849999983},"creditCard":{"creditCardNumber":"4916-0816-6217-7968","creditCardCvv":397,"creditCar
dExpirationYear":2039,"creditCardExpirationMonth":5}},"msg":"Charge request received."}
```

Cartservice

```
ubuntu@ip-10-0-1-222:~$ kubectl logs pod/cart-b66f96d74-vtttt -n otel-demo
Defaulted container "cart" out of: cart, wait-for-valkey-cart (init)
info: cart.cartstore.ValkeyCartStore[0]
      Successfully connected to Redis
warn: Microsoft.AspNetCore.Hosting.Diagnostics[15]
      Overriding HTTP_PORTS '8080' and HTTPS_PORTS ''. Binding to values defined by URLS instead 'http://*:8080
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://[::]:8080
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /usr/src/app
info: cart.cartstore.ValkeyCartStore[0]
      GetCartAsync called with userId=
info: cart.cartstore.ValkeyCartStore[0]
      GetCartAsync called with userId=ba1db1ee-a2e1-48f5-bbb2-11e74e1e1c21
info: cart.cartstore.ValkeyCartStore[0]
      GetCartAsync called with userId=ba1db1ee-a2e1-48f5-bbb2-11e74e1e1c21
info: cart.cartstore.ValkeyCartStore[0]
      GetCartAsync called with userId=01a3166e-b341-4746-87b2-90228c8256bd
info: cart.cartstore.ValkeyCartStore[0]
      GetCartAsync called with userId=01a3166e-b341-4746-87b2-90228c8256bd
info: cart.cartstore.ValkeyCartStore[0]
      GetCartAsync called with userId=6d9b6975-1dc7-4533-b48c-fd02c1a00a8c
info: cart.cartstore.ValkeyCartStore[0]
```

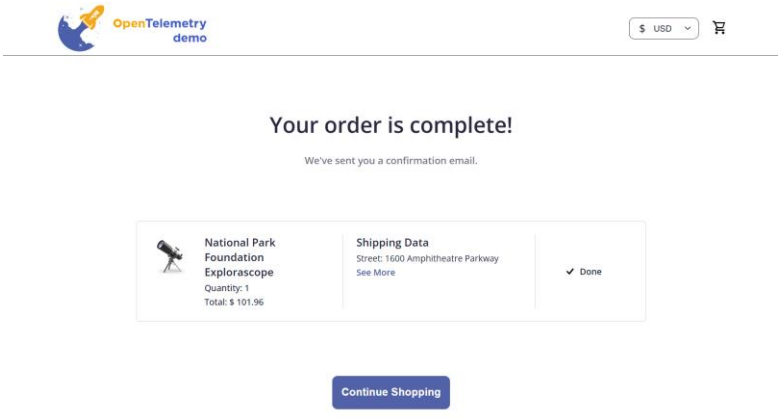
```
info: cart.cartstore.ValkeyCartStore[0]
      AddItemAsync called with userId=27d11090-2f95-11f0-913d-d2be14ea593a, productId=66VCHSJNUP, quantity=10
info: cart.cartstore.ValkeyCartStore[0]
      GetCartAsync called with userId=27d11090-2f95-11f0-913d-d2be14ea593a
info: cart.cartstore.ValkeyCartStore[0]
      GetCartAsync called with userId=260d0de5-f080-489a-a463-1f16ced627f9
info: cart.cartstore.ValkeyCartStore[0]
      GetCartAsync called with userId=b2ad60f4-7451-426c-968c-3b916e75d556
info: cart.cartstore.ValkeyCartStore[0]
      GetCartAsync called with userId=260d0de5-f080-489a-a463-1f16ced627f9
info: cart.cartstore.ValkeyCartStore[0]
      GetCartAsync called with userId=29d9143e-1a00-4541-9d24-6ade6db0e561
info: cart.cartstore.ValkeyCartStore[0]
      GetCartAsync called with userId=29d9143e-1a00-4541-9d24-6ade6db0e561
info: cart.cartstore.ValkeyCartStore[0]
      GetCartAsync called with userId=260d0de5-f080-489a-a463-1f16ced627f9
info: cart.cartstore.ValkeyCartStore[0]
      AddItemAsync called with userId=29886eec-2f95-11f0-913d-d2be14ea593a, productId=OLJCESPC7Z, quantity=2
info: cart.cartstore.ValkeyCartStore[0]
```

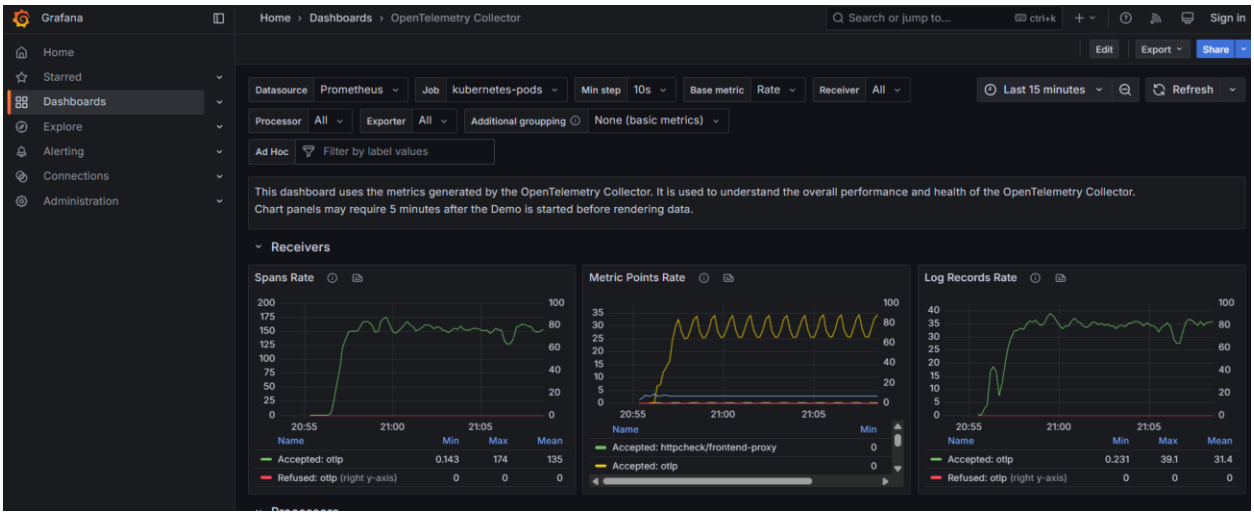
Shipping Service

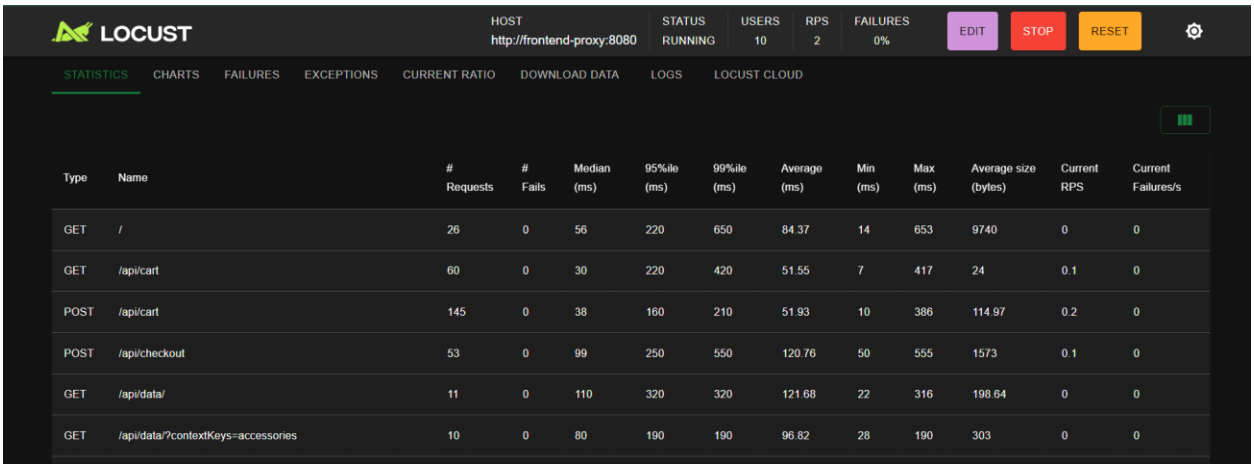
```
ubuntu@ip-10-0-1-222:~$ kubectl logs pod/shipping-7c6bc949f7-zwl7t -n otel-d
00:54:59 [INFO] OTel pipeline created
00:54:59 [INFO] listening on 0.0.0.0:8080
00:56:55 [INFO] Received quote: "1417.6"
00:56:55 [INFO] Sending Quote: 1417.60
00:56:55 [INFO] Tracking ID Created: 1235ea52-6f80-4855-a0a3-b46ff20994b2
00:57:01 [INFO] Received quote: "1824.1"
00:57:01 [INFO] Sending Quote: 1824.10
00:57:01 [INFO] Tracking ID Created: 6f837571-ffea-4472-951f-4e2c7f45bd5e
00:57:54 [INFO] Received quote: "553.8"
00:57:54 [INFO] Sending Quote: 553.79
00:57:54 [INFO] Tracking ID Created: 96735d6d-7389-41a6-b686-41dcf04c8e09
00:58:06 [INFO] Received quote: "276.6"
00:58:06 [INFO] Sending Quote: 276.60
00:58:06 [INFO] Tracking ID Created: 67faa92b-0e2e-4d93-b3bb-f1efe7fedd21
00:58:07 [INFO] Received quote: "1411.8"
00:58:07 [INFO] Sending Quote: 1411.80
00:58:07 [INFO] Tracking ID Created: ae848a34-1c3a-4b95-9d6e-5e665b51d06e
00:58:10 [INFO] Received quote: "283.8"
00:58:10 [INFO] Sending Quote: 283.80
00:58:10 [INFO] Tracking ID Created: 7e068d5a-b278-4a7d-8ac3-0d76f92009c0
00:58:13 [INFO] Received quote: "511"
00:58:13 [INFO] Sending Quote: 511.0
00:58:13 [INFO] Tracking ID Created: 85dcc260-604d-47ad-b8c2-cdec05954d56
00:58:26 [INFO] Received quote: "66"
00:58:26 [INFO] Sending Quote: 66.0
```

Screenshot of accessible application

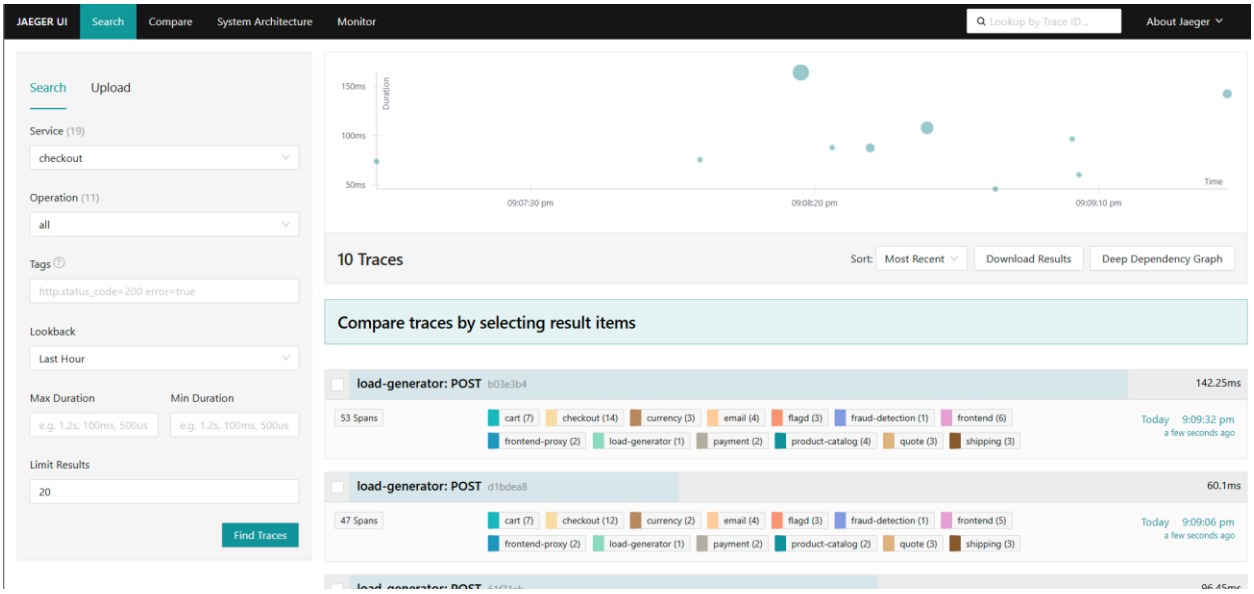
Port forwarding from the local machine to the frontend-proxy service on port 8080 was successfully established using kubectl. This allows local browser access to the OpenTelemetry demo UI for validation and testing purposes







Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	26	0	56	220	650	84.37	14	653	9740	0	0
GET	/api/cart	60	0	30	220	420	51.55	7	417	24	0.1	0
POST	/api/cart	145	0	38	160	210	51.93	10	386	114.97	0.2	0
POST	/api/checkout	53	0	99	250	550	120.76	50	555	1573	0.1	0
GET	/api/data/	11	0	110	320	320	121.68	22	316	198.64	0	0
GET	/api/data/?contextKeys=accessories	10	0	80	190	190	96.82	28	190	303	0	0



Phase 2: Integrating Helm for Deployment

To streamline the management of Kubernetes resources and adopt a more maintainable and scalable deployment strategy, Helm was integrated in this phase. Helm enables templated configuration, versioned releases, and simplified rollbacks, which are critical in managing complex microservice environments like the OpenTelemetry Demo.

2.1 Adding and Updating the Helm Repository

The official OpenTelemetry Helm charts were sourced from the OpenTelemetry Helm Chart Repository. The Helm client was first installed on the EKS admin EC2 node, followed by repository configuration.

The following commands were run:

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

helm repo update

```
ubuntu@ip-10-0-1-222:~$ helm repo add open-telemetry https://open-telemetry.
github.io/opentelemetry-helm-charts
"open-telemetry" already exists with the same configuration, skipping
```

```
ubuntu@ip-10-0-1-222:~$ helm search repo open-telemetry
NAME                                    CHART VERSION   APP VERSION     DESCRIPTION
open-telemetry/opentelemetry-collector  0.122.5         0.123.1         OpenTelemetry Collector Helm chart for Kubernetes
open-telemetry/opentelemetry-demo       0.37.1          2.0.2           opentelemetry demo helm chart
open-telemetry/opentelemetry-ebpf       0.1.6           v0.10.2         OpenTelemetry eBPF Helm chart for Kubernetes
open-telemetry/opentelemetry-kube-stack 0.5.2           0.120.0         OpenTelemetry Quickstart chart for Kubernetes. ...
open-telemetry/opentelemetry-operator   0.88.6          0.124.0         OpenTelemetry Operator Helm chart for Kubernetes
ubuntu@ip-10-0-1-222:~$
```

```
ubuntu@ip-10-0-1-222:~$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "open-telemetry" chart repository
Update Complete. ⎈Happy Helming!⎈
ubuntu@ip-10-0-1-222:~$
```

2.2 Helm-Based Deployment into a Separate Namespace

To isolate the Helm-managed resources from the manually applied manifests in Phase 1, a new namespace otel-demo-helm was created using this command

kubectl create namespace otel-demo-helm

```
ubuntu@ip-10-0-1-222:~$ kubectl create namespace otel-demo-helm
namespace/otel-demo-helm created
ubuntu@ip-10-0-1-222:~$
```

```
ubuntu@ip-10-0-1-222:~$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   11m
kube-node-lease   Active   11m
kube-public       Active   11m
kube-system       Active   11m
otel-demo-helm    Active   21s
ubuntu@ip-10-0-1-222:~$
```

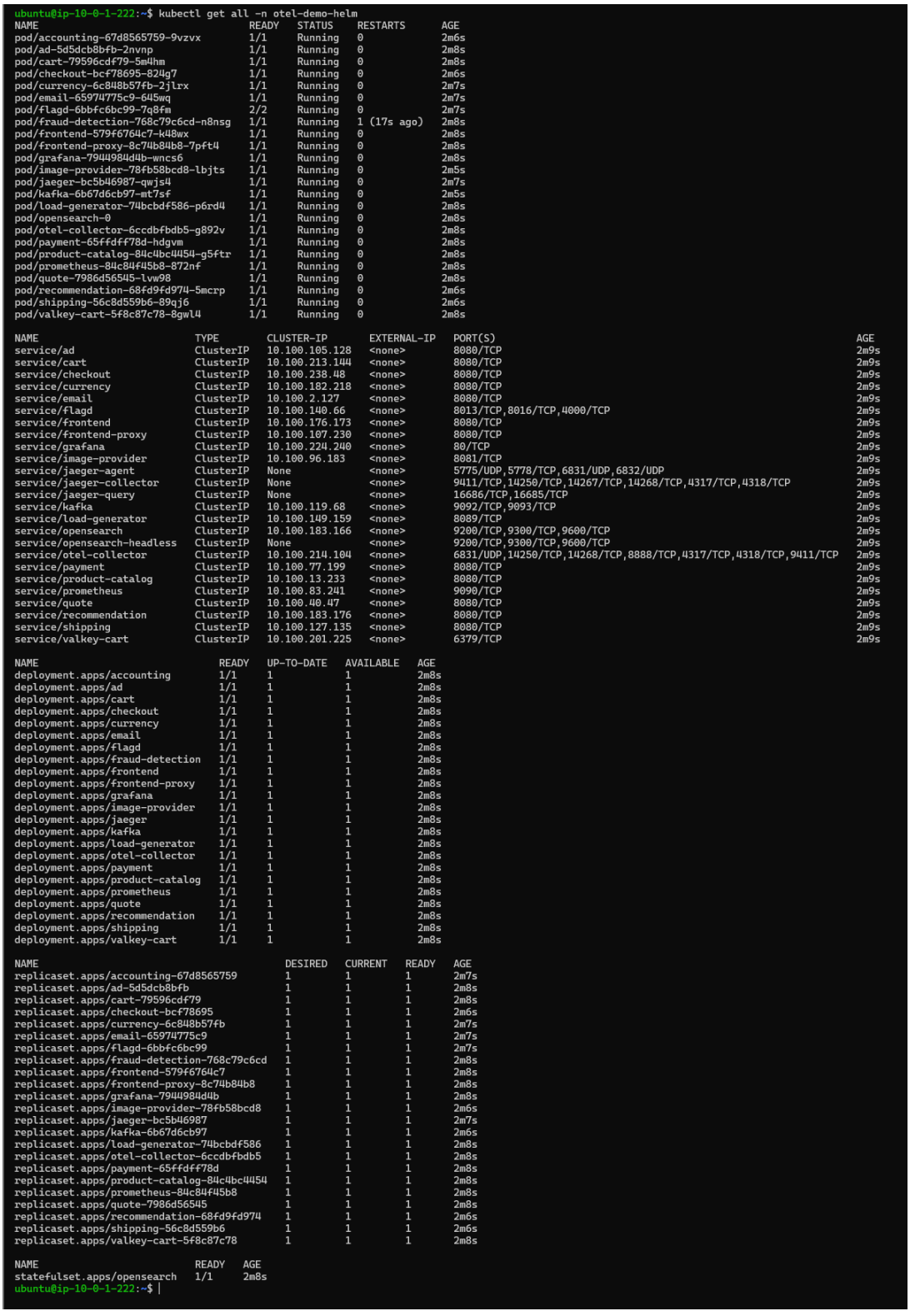
The Helm chart was deployed using the following command:

helm install otel-demo open-telemetry/opentelemetry-demo -n otel-demo-helm

```
ubuntu@ip-10-0-1-222:~$ helm install otel-demo open-telemetry/opentelemetry-demo -n otel-demo-helm
NAME: otel-demo
LAST DEPLOYED: Tue May 13 02:20:29 2025
NAMESPACE: otel-demo-helm
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
=======================================================================================

- All services are available via the Frontend proxy: http://localhost:8080
  by running these commands:
     kubectl --namespace otel-demo-helm port-forward svc/frontend-proxy 8080:8080

  The following services are available at these paths after the frontend-proxy service is exposed with port forwardi
ng:
  Webstore             http://localhost:8080/
  Jaeger UI            http://localhost:8080/jaeger/ui/
  Grafana              http://localhost:8080/grafana/
  Load Generator UI    http://localhost:8080/loadgen/
  Feature Flags UI     http://localhost:8080/feature/
ubuntu@ip-10-0-1-222:~$
```

This deployed all required components — including the frontend, backend microservices, OTEL Collector, Prometheus, Grafana, and Jaeger — as a unified Helm release. Custom values can be overridden using a values.yaml file, if needed.

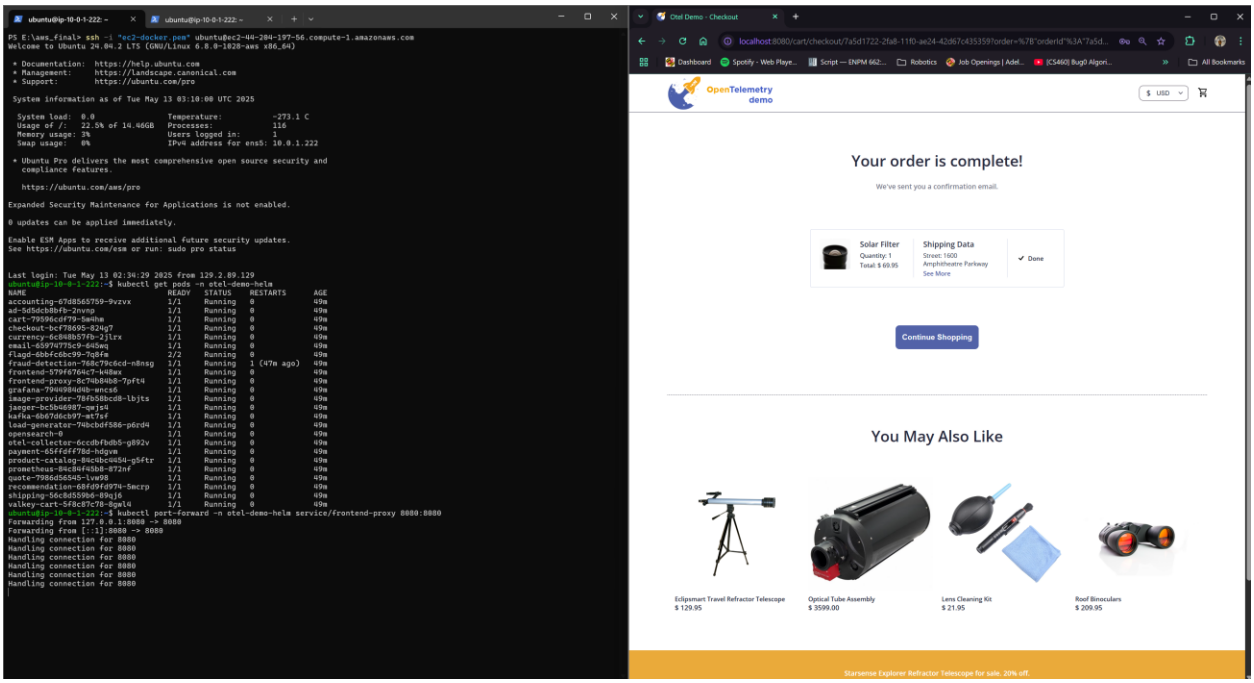
Screenshot of all pods and services running under **otel-demo-helm namespace**



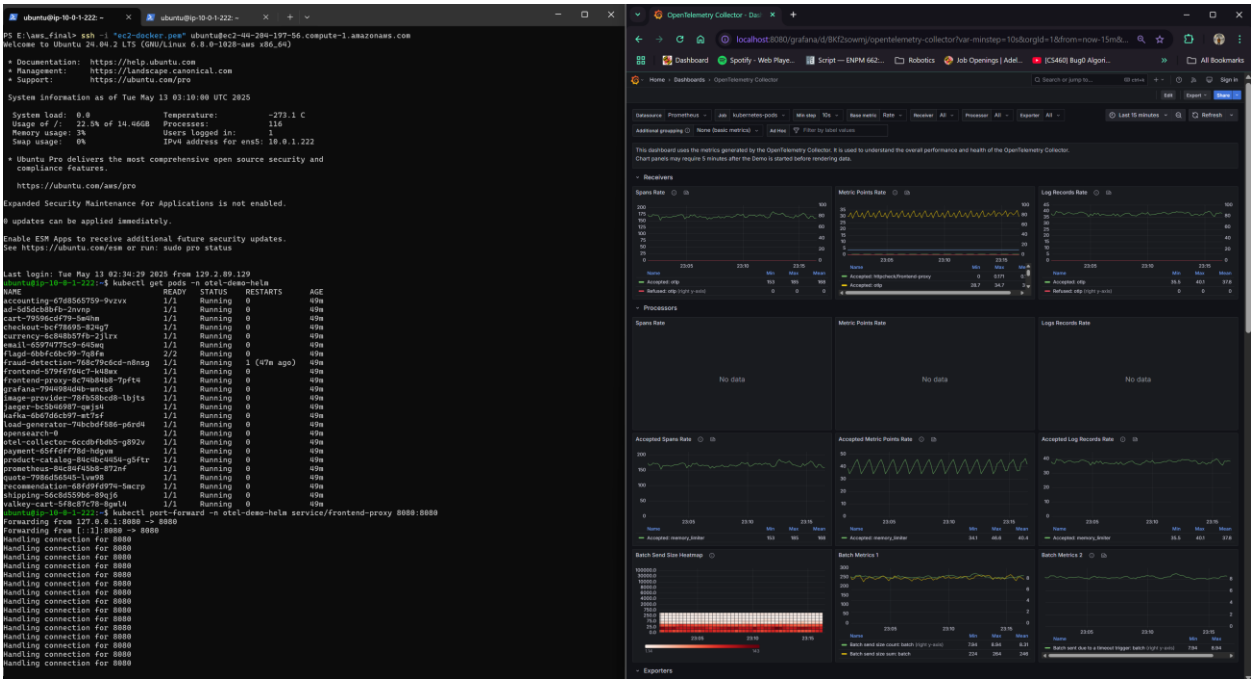
```
ubuntu@ip-10-0-1-222:~$ kubectl get all -n otel-demo-helm
NAME                                      READY   STATUS    RESTARTS      AGE
pod/accounting-67d8565759-9vzvx           1/1     Running   0             2m6s
pod/ad-5d5dcb8bfb-2nvnp                   1/1     Running   0             2m8s
pod/cart-79596cdf79-5m4hm                 1/1     Running   0             2m8s
pod/checkout-bcf78695-824g7               1/1     Running   0             2m6s
pod/currency-6c848b57fb-2jlrx             1/1     Running   0             2m7s
pod/email-65974775c9-645wq                1/1     Running   0             2m7s
pod/flagd-6bbfc6bc99-7q8fm                2/2     Running   0             2m7s
pod/fraud-detection-768c79c6cd-n8nsg      1/1     Running   1 (17s ago)   2m8s
pod/frontend-579f6764c7-k48wx             1/1     Running   0             2m8s
pod/frontend-proxy-8c74b84b8-7pft4        1/1     Running   0             2m8s
pod/grafana-7944984d4b-wncs6              1/1     Running   0             2m8s
pod/image-provider-78fb58bcd8-lbjts       1/1     Running   0             2m5s
pod/jaeger-bc5b46987-qwjs4                1/1     Running   0             2m7s
pod/kafka-6b67d6cb97-mt7sf                1/1     Running   0             2m5s
pod/load-generator-74bcbdf586-p6rd4       1/1     Running   0             2m8s
pod/opensearch-0                          1/1     Running   0             2m8s
pod/otel-collector-6ccdbfbdb5-g892v       1/1     Running   0             2m8s
pod/payment-65ffdff78d-hdgvm              1/1     Running   0             2m8s
pod/product-catalog-84c4bc4454-g5ftr      1/1     Running   0             2m8s
pod/prometheus-84c84f45b8-872nf           1/1     Running   0             2m8s
pod/quote-7986d56545-lvw98                1/1     Running   0             2m8s
pod/recommendation-68fd9fd974-5mcrp       1/1     Running   0             2m6s
pod/shipping-56c8d559b6-89qj6             1/1     Running   0             2m6s
pod/valkey-cart-5f8c87c78-8gwl4           1/1     Running   0             2m8s

NAME                           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                                                                    AGE
service/ad                     ClusterIP   10.100.105.128   <none>        8080/TCP                                                                   2m9s
service/cart                   ClusterIP   10.100.213.144   <none>        8080/TCP                                                                   2m9s
service/checkout               ClusterIP   10.100.238.48    <none>        8080/TCP                                                                   2m9s
service/currency               ClusterIP   10.100.182.218   <none>        8080/TCP                                                                   2m9s
service/email                  ClusterIP   10.100.2.127     <none>        8080/TCP                                                                   2m9s
service/flagd                  ClusterIP   10.100.140.66    <none>        8013/TCP,8016/TCP,4000/TCP                                                 2m9s
service/frontend               ClusterIP   10.100.176.173   <none>        8080/TCP                                                                   2m9s
service/frontend-proxy         ClusterIP   10.100.107.230   <none>        8080/TCP                                                                   2m9s
service/grafana                ClusterIP   10.100.224.240   <none>        80/TCP                                                                     2m9s
service/image-provider         ClusterIP   10.100.96.183    <none>        8081/TCP                                                                   2m9s
service/jaeger-agent           ClusterIP   None             <none>        5775/UDP,5778/TCP,6831/UDP,6832/UDP                                        2m9s
service/jaeger-collector       ClusterIP   None             <none>        9411/TCP,14250/TCP,14267/TCP,14268/TCP,4317/TCP,4318/TCP                   2m9s
service/jaeger-query           ClusterIP   None             <none>        16686/TCP,16685/TCP                                                        2m9s
service/kafka                  ClusterIP   10.100.119.68    <none>        9092/TCP,9093/TCP                                                          2m9s
service/load-generator         ClusterIP   10.100.149.159   <none>        8089/TCP                                                                   2m9s
service/opensearch             ClusterIP   10.100.183.166   <none>        9200/TCP,9300/TCP,9600/TCP                                                 2m9s
service/opensearch-headless    ClusterIP   None             <none>        9200/TCP,9300/TCP,9600/TCP                                                 2m9s
service/otel-collector         ClusterIP   10.100.214.104   <none>        6831/UDP,14250/TCP,14268/TCP,8888/TCP,4317/TCP,4318/TCP,9411/TCP           2m9s
service/payment                ClusterIP   10.100.77.199    <none>        8080/TCP                                                                   2m9s
service/product-catalog        ClusterIP   10.100.13.233    <none>        8080/TCP                                                                   2m9s
service/prometheus             ClusterIP   10.100.83.241    <none>        9090/TCP                                                                   2m9s
service/quote                  ClusterIP   10.100.40.47     <none>        8080/TCP                                                                   2m9s
service/recommendation         ClusterIP   10.100.183.176   <none>        8080/TCP                                                                   2m9s
service/shipping               ClusterIP   10.100.127.135   <none>        8080/TCP                                                                   2m9s
service/valkey-cart            ClusterIP   10.100.201.225   <none>        6379/TCP                                                                   2m9s

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/accounting           1/1     1            1           2m8s
deployment.apps/ad                   1/1     1            1           2m8s
deployment.apps/cart                 1/1     1            1           2m8s
deployment.apps/checkout             1/1     1            1           2m8s
deployment.apps/currency             1/1     1            1           2m8s
deployment.apps/email                1/1     1            1           2m8s
deployment.apps/flagd                1/1     1            1           2m8s
deployment.apps/fraud-detection      1/1     1            1           2m8s
deployment.apps/frontend             1/1     1            1           2m8s
deployment.apps/frontend-proxy       1/1     1            1           2m8s
deployment.apps/grafana              1/1     1            1           2m8s
deployment.apps/image-provider       1/1     1            1           2m8s
deployment.apps/jaeger               1/1     1            1           2m8s
deployment.apps/kafka                1/1     1            1           2m8s
deployment.apps/load-generator       1/1     1            1           2m8s
deployment.apps/otel-collector       1/1     1            1           2m8s
deployment.apps/payment              1/1     1            1           2m8s
deployment.apps/product-catalog      1/1     1            1           2m8s
deployment.apps/prometheus           1/1     1            1           2m8s
deployment.apps/quote                1/1     1            1           2m8s
deployment.apps/recommendation       1/1     1            1           2m8s
deployment.apps/shipping             1/1     1            1           2m8s
deployment.apps/valkey-cart          1/1     1            1           2m8s

NAME                                            DESIRED   CURRENT   READY   AGE
replicaset.apps/accounting-67d8565759           1         1         1       2m7s
replicaset.apps/ad-5d5dcb8bfb                   1         1         1       2m8s
replicaset.apps/cart-79596cdf79                 1         1         1       2m8s
replicaset.apps/checkout-bcf78695               1         1         1       2m6s
replicaset.apps/currency-6c848b57fb             1         1         1       2m7s
replicaset.apps/email-65974775c9                1         1         1       2m7s
replicaset.apps/flagd-6bbfc6bc99                1         1         1       2m7s
replicaset.apps/fraud-detection-768c79c6cd       1         1         1       2m8s
replicaset.apps/frontend-579f6764c7             1         1         1       2m8s
replicaset.apps/frontend-proxy-8c74b84b8        1         1         1       2m8s
replicaset.apps/grafana-7944984d4b              1         1         1       2m8s
replicaset.apps/image-provider-78fb58bcd8       1         1         1       2m6s
replicaset.apps/jaeger-bc5b46987                1         1         1       2m7s
replicaset.apps/kafka-6b67d6cb97                1         1         1       2m6s
replicaset.apps/load-generator-74bcbdf586       1         1         1       2m8s
replicaset.apps/otel-collector-6ccdbfbdb5       1         1         1       2m8s
replicaset.apps/payment-65ffdff78d              1         1         1       2m8s
replicaset.apps/product-catalog-84c4bc4454      1         1         1       2m8s
replicaset.apps/prometheus-84c84f45b8           1         1         1       2m8s
replicaset.apps/quote-7986d56545                1         1         1       2m8s
replicaset.apps/recommendation-68fd9fd974       1         1         1       2m6s
replicaset.apps/shipping-56c8d559b6             1         1         1       2m6s
replicaset.apps/valkey-cart-5f8c87c78           1         1         1       2m8s

NAME                                 READY   AGE
statefulset.apps/opensearch          1/1     2m8s
ubuntu@ip-10-0-1-222:~$
```

Application deployed successfully using Helm:

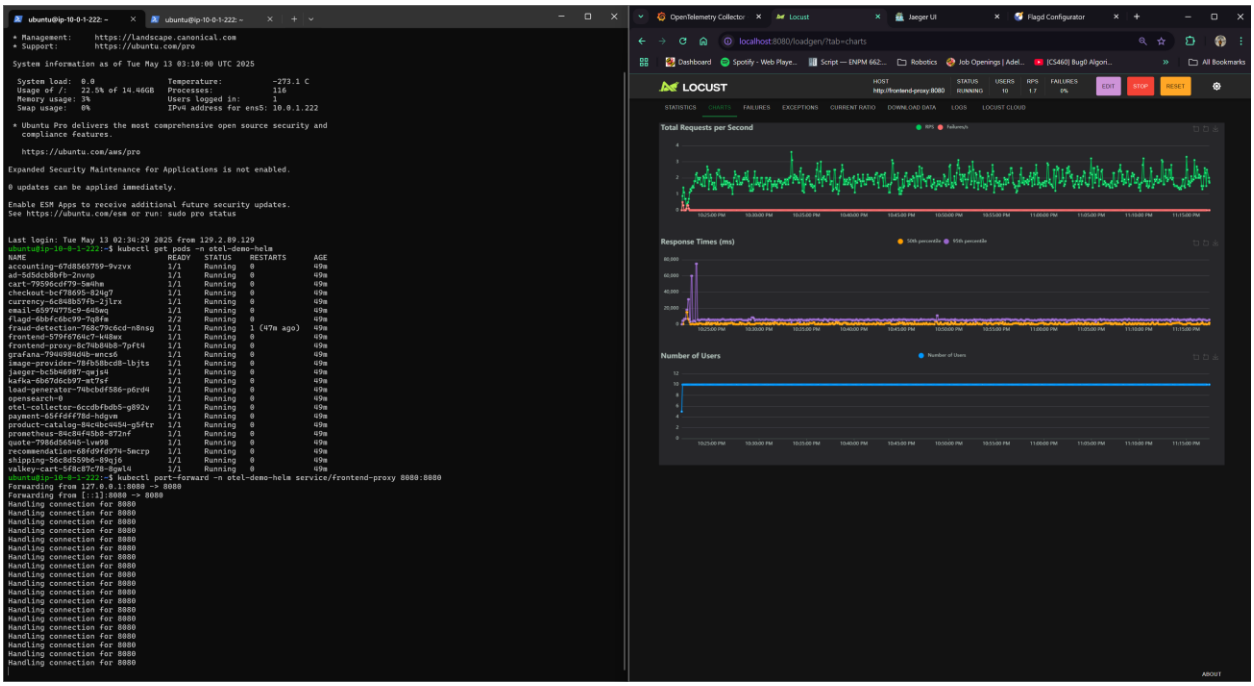
Web store:



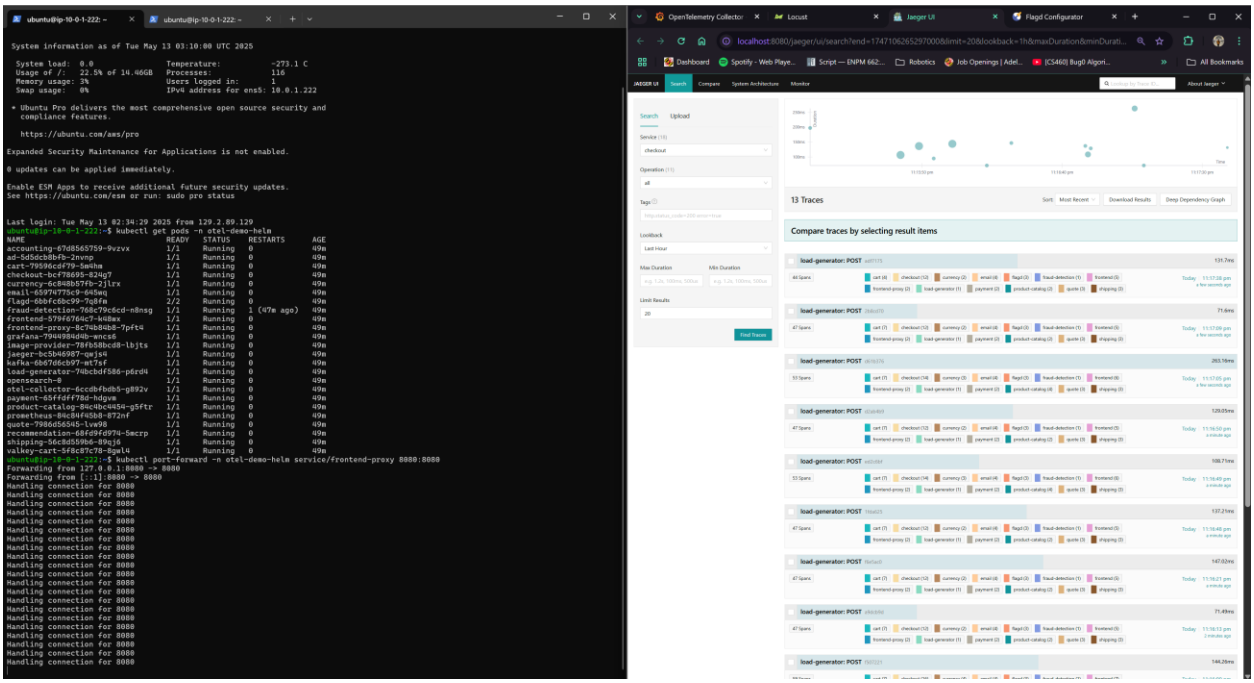
Grafana:



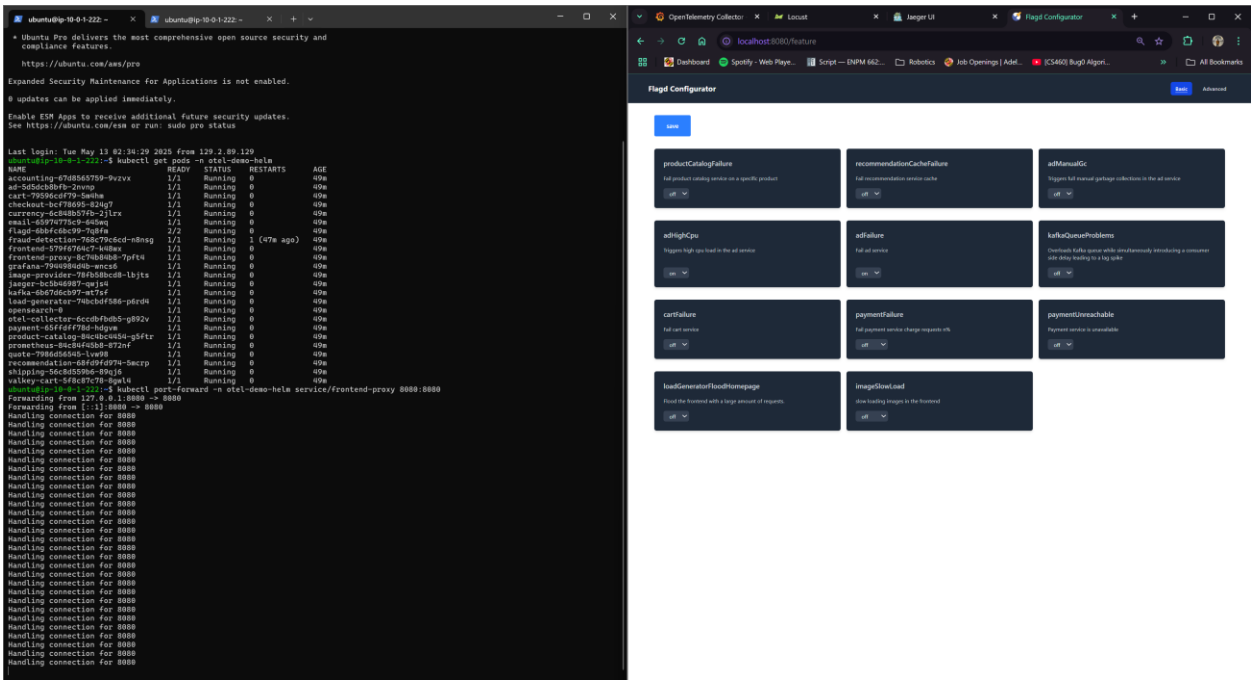
Load Generator UI:



Jaeger UI:



Flagd configurator UI:



2.3 Upgrade and Rollback Simulation

To test Helm's upgrade and rollback features, we simulated a configuration change by increasing the number of Jaeger replicas from 1 to 2. This type of upgrade is useful in real-world scenarios for scaling observability components based on load or redundancy requirements.

Initially, only a single pod for Jaeger was deployed as shown below (Output shows one Jaeger pod running.)

```
ubuntu@ip-10-0-1-222:~$ kubectl get pods -n otel-demo-helm
NAME                                 READY   STATUS    RESTARTS      AGE
accounting-67d8565759-9vzvx          1/1     Running   0             14m
ad-5d5dcb8bfb-2nvnp                  1/1     Running   0             14m
cart-79596cdf79-5m4hm                1/1     Running   0             14m
checkout-bcf78695-824g7              1/1     Running   0             14m
currency-6c848b57fb-2jlrx            1/1     Running   0             14m
email-65974775c9-645wq               1/1     Running   0             14m
flagd-6bbfc6bc99-7q8fm               2/2     Running   0             14m
fraud-detection-768c79c6cd-n8nsg     1/1     Running   1 (12m ago)   14m
frontend-579f6764c7-k48wx            1/1     Running   0             14m
frontend-proxy-8c74b84b8-7pft4       1/1     Running   0             14m
grafana-7944984d4b-wncs6             1/1     Running   0             14m
image-provider-78fb58bcd8-lbjts      1/1     Running   0             14m
jaeger-bc5b46987-qwjs4               1/1     Running   0             14m
kafka-6b67d6cb97-mt7sf               1/1     Running   0             14m
load-generator-74bcbdf586-p6rd4      1/1     Running   0             14m
opensearch-0                         1/1     Running   0             14m
otel-collector-6ccdbfbdb5-g892v      1/1     Running   0             14m
payment-65ffdff78d-hdgvm             1/1     Running   0             14m
product-catalog-84c4bc4454-g5ftr     1/1     Running   0             14m
prometheus-84c84f45b8-872nf          1/1     Running   0             14m
quote-7986d56545-lvw98               1/1     Running   0             14m
recommendation-68fd9fd974-5mcrp      1/1     Running   0             14m
shipping-56c8d559b6-89qj6            1/1     Running   0             14m
valkey-cart-5f8c87c78-8gwl4          1/1     Running   0             14m
ubuntu@ip-10-0-1-222:~$
```

Performing the Upgrade:

Before making any changes to the Helm release, we extracted the currently deployed configuration using:

helm get values -n otel-demo-helm otel-demo --all > values.yaml

This command retrieves all user-supplied values currently applied to the otel-demo release, including default and custom configurations. The output is redirected into a values.yaml file which serves two key purposes:

- Baseline Reference: It allows us to understand and compare the existing deployment configuration before applying any updates.
- Safe Upgrade Practice: Having a snapshot of the current values ensures we can safely roll back or re-apply the original settings if the upgrade fails or causes unexpected issues.

This is considered a best practice when performing upgrades with Helm, especially in production-like environments, as it preserves state and enables reproducibility.

```
ubuntu@ip-10-0-1-222:~$ helm upgrade otel-demo open-telemetry/opentelemetry-demo -n otel-demo-helm -f values.yaml
Release "otel-demo" has been upgraded. Happy Helming!
NAME: otel-demo
LAST DEPLOYED: Tue May 13 02:56:43 2025
NAMESPACE: otel-demo-helm
STATUS: deployed
REVISION: 2
TEST SUITE: None
NOTES:
================================================================================

 ____  ____  ____  ____     ____  ____  __  __  ____
(  _ \(_  _)( ___)(  _ \   (  _ \( ___)(  \/  )(  _ \
 )(_) )_)(_  )__)  )   /    )(_) ))__)  )    (  )___/
(____/(____)(____)(_)\_)   (____/(____)(_/\/\_)(__)

- All services are available via the Frontend proxy: http://localhost:8080
  by running these commands:
     kubectl --namespace otel-demo-helm port-forward svc/frontend-proxy 8080:8080

  The following services are available at these paths after the frontend-proxy service is exposed with port forwarding:
  Webstore            http://localhost:8080/
  Jaeger UI           http://localhost:8080/jaeger/ui/
  Grafana             http://localhost:8080/grafana/
  Load Generator UI   http://localhost:8080/loadgen/
  Feature Flags UI    http://localhost:8080/feature/
ubuntu@ip-10-0-1-222:~$
```

After performing the Helm upgrade using the modified values.yaml file, we validated that the configuration was applied successfully.

```
ubuntu@ip-10-0-1-222:~$ kubectl get pods -n otel-demo-helm
NAME                                  READY   STATUS    RESTARTS      AGE
accounting-67d8565759-9vzvx           1/1     Running   0             37m
ad-5d5dcb8bfb-2nvnp                   1/1     Running   0             37m
cart-79596cdf79-5m4hm                 1/1     Running   0             37m
checkout-bcf78695-824g7               1/1     Running   0             37m
currency-6c848b57fb-2jlrx             1/1     Running   0             37m
email-65974775c9-645wq                1/1     Running   0             37m
flagd-6bbfc6bc99-7q8fm                2/2     Running   0             37m
fraud-detection-768c79c6cd-n8nsg      1/1     Running   1 (35m ago)   37m
frontend-579f6764c7-k48wx             1/1     Running   0             37m
frontend-proxy-8c74b84b8-7pft4        1/1     Running   0             37m
grafana-7944984d4b-wncs6              1/1     Running   0             37m
image-provider-78fb58bcd8-lbjts       1/1     Running   0             37m
jaeger-bc5b46987-qwjs4                1/1     Running   0             37m
jaeger-bc5b46987-sw4v2                1/1     Running   0             78s
kafka-6b67d6cb97-mt7sf                1/1     Running   0             37m
load-generator-74bcbdf586-p6rd4       1/1     Running   0             37m
opensearch-0                          1/1     Running   0             37m
otel-collector-6ccdbfbdb5-g892v       1/1     Running   0             37m
payment-65ffdff78d-hdgvm              1/1     Running   0             37m
product-catalog-84c4bc4454-g5ftr      1/1     Running   0             37m
prometheus-84c84f45b8-872nf           1/1     Running   0             37m
quote-7986d56545-lvw98                1/1     Running   0             37m
recommendation-68fd9fd974-5mcrp       1/1     Running   0             37m
shipping-56c8d559b6-89qj6             1/1     Running   0             37m
valkey-cart-5f8c87c78-8gwl4           1/1     Running   0             37m
```

As shown in the screenshot above, the output confirms that two Jaeger pods (jaeger-bc5b46987-qwjs4 and jaeger-bc5b46987-swv42) are now running in the otel-demo-helm namespace. This change demonstrates that Helm correctly interpreted the updated replicas: 2 configuration in the values file and triggered a rolling update for the Jaeger Deployment.

Each Jaeger pod is in the Running state with 1/1 containers ready, indicating a healthy rollout. The second pod (-swv42) is newer (AGE: 78s) compared to the other (AGE: 37m), clearly showing that it was created as a result of the upgrade operation.

This outcome validates that:

- Helm accurately applies changes from the values.yaml file.

- Kubernetes dynamically scales deployments based on the updated Helm release.
- Stateful components like Jaeger remain fully functional and horizontally scalable.

This successful upgrade sets the stage for demonstrating Helm's rollback capability by simulating a failure next. After confirming the upgrade was successful, we used Helm's built-in version control to roll back the deployment to its previous state.

Checking Helm History:

```
ubuntu@ip-10-0-1-222:~$ helm history -n otel-demo-helm otel-demo
REVISION        UPDATED                         STATUS          CHART                           APP VERSION     DESCRIPTION
1               Tue May 13 02:20:29 2025        superseded      opentelemetry-demo-0.37.1       2.0.2           Install complete
2               Tue May 13 02:56:43 2025        deployed        opentelemetry-demo-0.37.1       2.0.2           Upgrade complete
ubuntu@ip-10-0-1-222:~$
```

As seen in the output, two revisions existed:

- Revision 1: The initial install (Install complete)
- Revision 2: The updated release with Jaeger replicas = 2 (Upgrade complete)

Performing the Rollback

To revert the changes and restore the original configuration (with a single Jaeger replica), we ran:

```
ubuntu@ip-10-0-1-222:~$ helm rollback otel-demo -n otel-demo-helm
Rollback was a success! Happy Helming!
```

Post-Rollback Verification:

```
ubuntu@ip-10-0-1-222:~$ kubectl get pods -n otel-demo-helm
NAME                                 READY   STATUS    RESTARTS      AGE
accounting-67d8565759-9vzvx          1/1     Running   0             49m
ad-5d5dcb8bfb-2nvnp                  1/1     Running   0             49m
cart-79596cdf79-5m4hm                1/1     Running   0             49m
checkout-bcf78695-824g7              1/1     Running   0             49m
currency-6c848b57fb-2jlrx            1/1     Running   0             49m
email-65974775c9-645wq               1/1     Running   0             49m
flagd-6bbfc6bc99-7q8fm               2/2     Running   0             49m
fraud-detection-768c79c6cd-n8nsg     1/1     Running   1 (47m ago)   49m
frontend-579f6764c7-k48wx            1/1     Running   0             49m
frontend-proxy-8c74b84b8-7pft4       1/1     Running   0             49m
grafana-7944984d4b-wncs6             1/1     Running   0             49m
image-provider-78fb58bcd8-lbjts      1/1     Running   0             49m
jaeger-bc5b46987-qwjs4               1/1     Running   0             49m
kafka-6b67d6cb97-mt7sf               1/1     Running   0             49m
load-generator-74bcbdf586-p6rd4      1/1     Running   0             49m
opensearch-0                         1/1     Running   0             49m
otel-collector-6ccdbfbdb5-g892v      1/1     Running   0             49m
payment-65ffdff78d-hdgvm             1/1     Running   0             49m
product-catalog-84c4bc4454-g5ftr     1/1     Running   0             49m
prometheus-84c84f45b8-872nf          1/1     Running   0             49m
quote-7986d56545-lvw98               1/1     Running   0             49m
recommendation-68fd9fd974-5mcrp      1/1     Running   0             49m
shipping-56c8d559b6-89qj6            1/1     Running   0             49m
valkey-cart-5f8c87c78-8gwl4          1/1     Running   0             49m
ubuntu@ip-10-0-1-222:~$
```

Only **one Jaeger pod** (jaeger-bc5b46987-qwjs4) **remained**, indicating that the system successfully returned to the original state.

Phase 3: Alerting Service and Notifications

Prometheus was installed as the required monitoring solution using Helm with the following command:

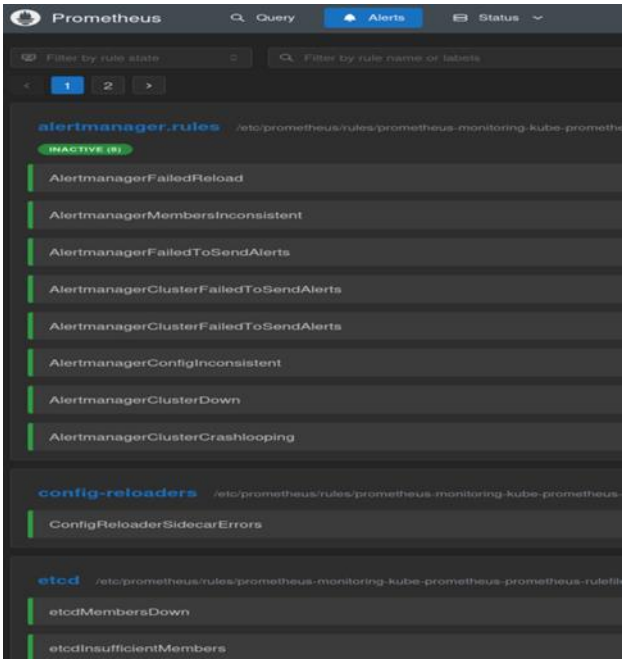
- **Helm install monitoring prometheus-community/kube-prometheus-stack --namespace monitoring**

```
ubuntu@ip-10-0-1-30:~$ kubectl get pods -n monitoring
NAME                                                     READY   STATUS             RESTARTS          AGE
alertmanager-monitoring-kube-prometheus-alertmanager-0   2/2     Running            0                 13m
crashloop-pod                                            0/1     CrashLoopBackOff   16 (4m37s ago)    61m
monitoring-grafana-5c548d845b-njmkr                      3/3     Running            0                 133m
monitoring-kube-prometheus-operator-d756c74bd-qwl7j      1/1     Running            0                 133m
monitoring-kube-state-metrics-7c4bdd9c57-6jv75           1/1     Running            0                 133m
monitoring-prometheus-node-exporter-pz6tf                1/1     Running            0                 133m
monitoring-prometheus-node-exporter-r42kc                1/1     Running            0                 133m
prometheus-monitoring-kube-prometheus-prometheus-0       2/2     Running            0                 133m
```

We confirmed that Prometheus, Alertmanager, Grafana, and supporting components such as kube-state-metrics and node-exporter were all running successfully. To further verify the installation, we used port-forwarding to access the Prometheus UI from localhost:

- **kubectl port-forward svc/monitoring-kube-prometheus-prometheus -n monitoring 9090:9090**

Navigating to http://localhost:9095 allowed us to view the Prometheus dashboard.



To enable automatic detection of abnormal behavior in the cluster, we configured Prometheus alerting rules. A custom alert was defined to monitor pod restart frequency. This alert triggers when any container within a pod restarts more than twice in a five-minute period — a common sign of crash loops or unstable workloads.

The custom alert rule was written using Prometheus' expression language and applied using the following file:

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: pod-restart-rules
  namespace: monitoring
spec:
  groups:
    - name: pod-restart-alerts
      rules:
        - alert: PodHighRestartCount
          expr: increase(kube_pod_container_status_restarts_total[5m]) > 2
          for: 1m
          labels:
            severity: warning
          annotations:
            summary: "Pod is restarting frequently"
            description: "Pod {{ $labels.pod }} in namespace {{ $labels.namespace }} restarted more than 2 times in 5 minutes."
```

This file was applied to the cluster with:

- **kubectl apply -f pod-restart-alert.yaml**

To test the alert, we created a pod that continuously fails using the following manifest:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: crashloop-pod
  namespace: monitoring
spec:
  restartPolicy: Always
  containers:
    - name: crashloop
      image: busybox
      command: ["/bin/sh", "-c", "exit 1"]
```

Email Notifications

To receive alerts externally, we configured Alertmanager to send email notifications via Gmail's SMTP service. This required enabling 2-Step Verification on the Gmail account and generating a 16-character App Password, which was used in place of a regular Gmail password.

```
global:
  smtp_smarthost: 'smtp.gmail.com:587'
  smtp_from: 'f.shaker02@gmail.com'
  smtp_auth_username: 'f.shaker02@gmail.com'
  smtp_auth_password: 'rvuohtjbtdxwpzru'

route:
  receiver: gmail-alerts
  group_by: ['alertname']
  group_wait: 10s
  group_interval: 5m
  repeat_interval: 1h

receivers:
  - name: gmail-alerts
    email_configs:
      - to: 'shaker.55@outlook.com'
        send_resolved: true
```

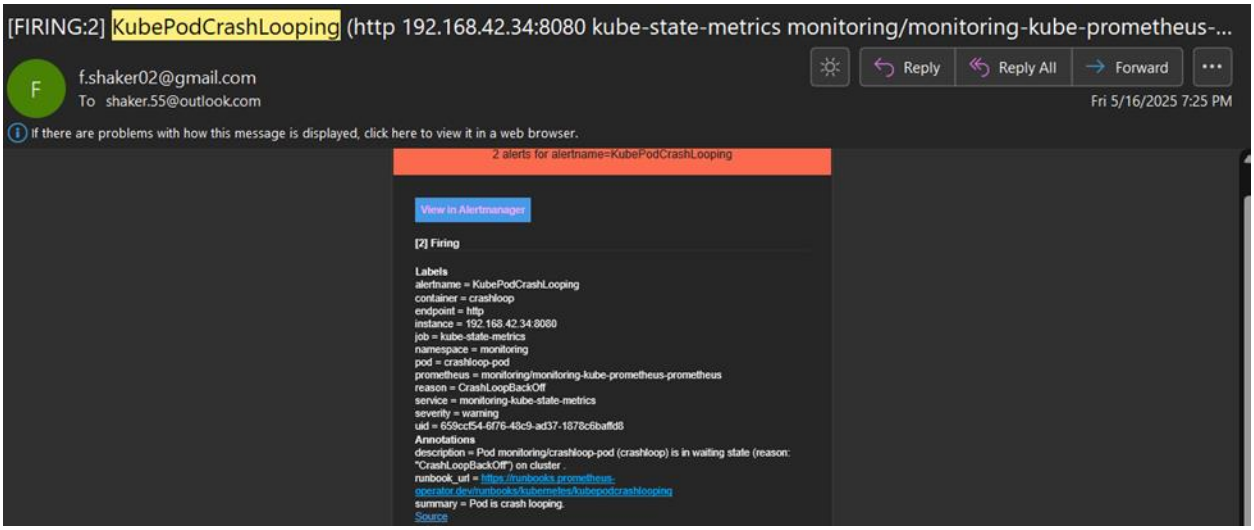
This file was base64-encoded and inserted into the running Kubernetes environment via the Alertmanager secret:

- **kubectl edit secret alertmanager-monitoring-kube-prometheus-alertmanager -n monitoring**

After saving the changes, the Alertmanager pod was restarted to apply the new configuration:

- **kubectl delete pod -l app.kubernetes.io/name=alertmanager -n monitoring**

The following screenshot confirms that the email alert was successfully triggered and delivered to the configured Gmail account when the crashloop-pod began restarting continuously:


[FIRING:2] KubePodCrashLooping (http 192.168.42.34:8080 kube-state-metrics monitoring/monitoring-kube-prometheus-...
f.shaker02@gmail.com
To shaker.55@outlook.com
Reply
Reply All
Forward
Fri 5/16/2025 7:25 PM
If there are problems with how this message is displayed, click here to view it in a web browser.
2 alerts for alertname=KubePodCrashLooping
View in Alertmanager
[2] Firing
Labels
alertname = KubePodCrashLooping
container = crashloop
endpoint = http
instance = 192.168.42.34:8080
job = kube-state-metrics
namespace = monitoring
pod = crashloop-pod
prometheus = monitoring/monitoring-kube-prometheus-prometheus
reason = CrashLoopBackOff
service = monitoring-kube-state-metrics
severity = warning
uid = 659cc154-6f76-48c9-ad37-1878c6baffd8
Annotations
description = Pod monitoring/crashloop-pod (crashloop) is in waiting state (reason: "CrashLoopBackOff") on cluster .
runbook_url = https://runbooks.prometheus-operator.dev/runbooks/kubernetes/kubepodcrashlooping
summary = Pod is crash looping.
Source

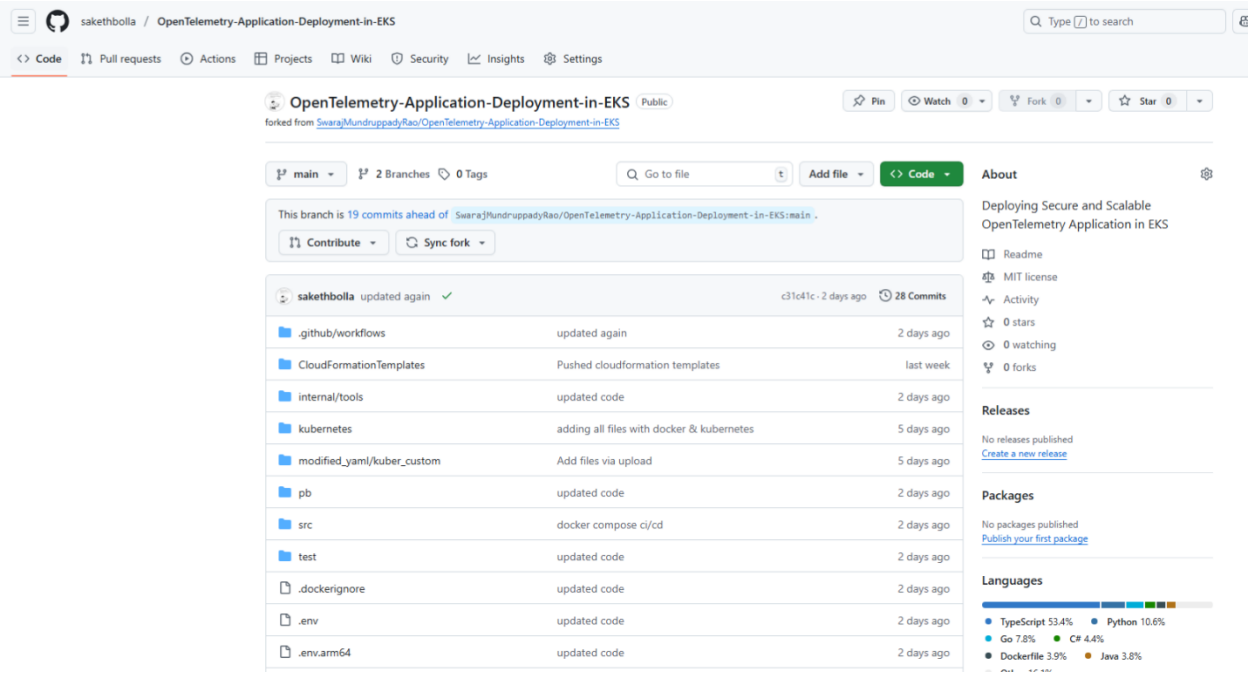

kubernetes-apps
KubePodCrashLooping
max_over_time(kube_pod_container_status_waiting_reason{job="kube-state-metrics",namespace=~".*",reason="CrashLoopBackOff"}[5m]) >= 1
severity="warning"
description Pod {{ $labels.namespace }}/{{ $labels.pod }} ({{ $labels.container }}) is in waiting state (reason: "CrashLoopBackOff") on cluster {{ $labels.cluster }}.
runbook_url https://runbooks.prometheus-operator.dev/runbooks/kubernetes/kubepodcrashlooping
summary Pod is crash looping.
Alert labels
State
Active Since
Value
FIRING
40m 16.963s
description Pod monitoring/crashloop-pod (crashloop) is in waiting state (reason: "CrashLoopBackOff") on cluster .
runbook_url https://runbooks.prometheus-operator.dev/runbooks/kubernetes/kubepodcrashlooping
summary Pod is crash looping.
FIRING
23m 46.963s
description Pod otel-demo-helm/prometheus-84c64f45b8-s7l5 (prometheus-server) is in waiting state (reason: "CrashLoopBackOff") on cluster .
runbook_url https://runbooks.prometheus-operator.dev/runbooks/kubernetes/kubepodcrashlooping
summary Pod is crash looping.

Phase 4: CI/CD Integration

4.1 CI/CD pipeline using GitHub Actions

To *build, deploy, test, rollback* the Open Telemetry demo app, we have utilized GitHub actions – GitHub's ubuntu env, GitHub Container registry, GitHub secrets, aws auth (for EKS) and Custom Helm Charts.

This is our repository that holds all the Workflow, manifest, docker compose files.



Pipeline is created in the path **.github/workflows/ci-cd-frontend.yml**

This CI/CD pipeline uses Docker Compose to build images for all services and pushes them to GitHub Container Registry (GHCR). These images are then used to create pods in EKS using Helm, which helps keep the deployment simple and consistent.

4.2 Security within workflow & deployment

For authentication, we used a **GitHub Personal Access Token (GPAT)**.

For security, we created **GitHub secrets** and used them in the workflow code as shown below:

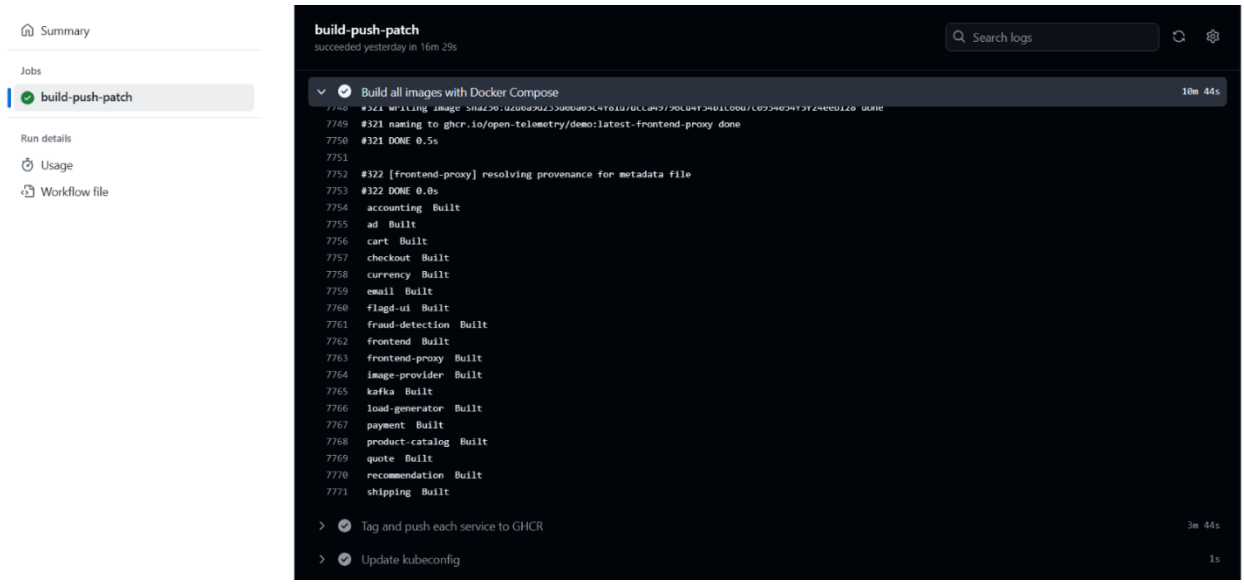
```
aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
```

```
aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
```

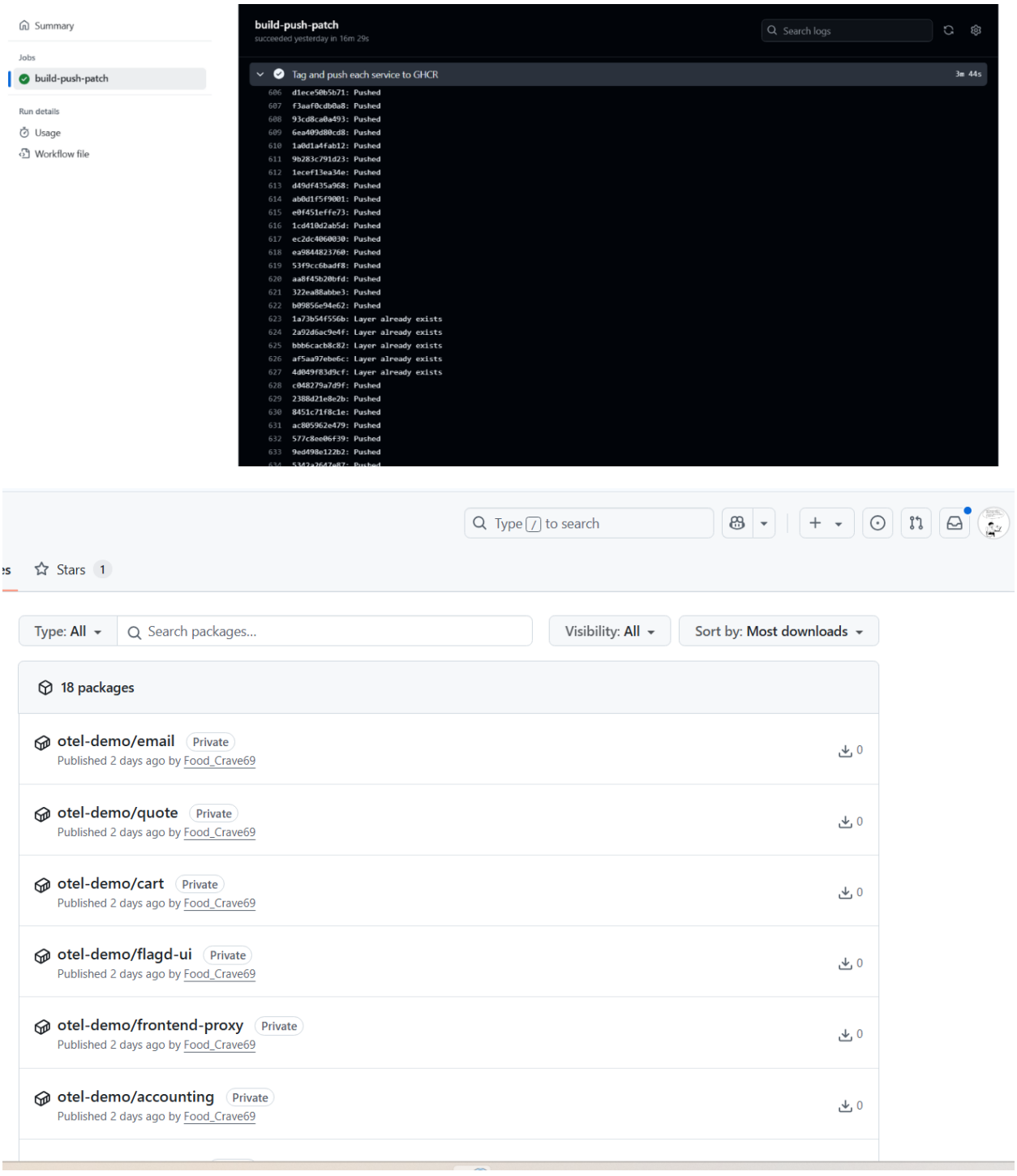
These secrets were referenced directly in the GitHub Actions workflow to keep credentials secure.

```
30        - name: Log in to GHCR
31          run: echo "${{ secrets.GHCR_PAT }}" | docker login ghcr.io -u ${{ github.actor }} --password-stdin
32
33        - name: Build all images with Docker Compose
34          run: docker compose -f $COMPOSE_FILE build
35
36        - name: Tag and push each service to GHCR
37          run: |
38            SERVICES=$(docker compose -f $COMPOSE_FILE config --services)
39            for SERVICE in $SERVICES; do
40              LOCAL_IMAGE=$(docker compose -f $COMPOSE_FILE config | awk "/^services:/,/^networks:/" | awk "/^  $SERVICE:/{flag=1;next}/^  [^ ]/{flag=0}flag && /image:/ {print \$2}")
41
42              if [ -z "$LOCAL_IMAGE" ]; then
43                echo "❌ Skipping $SERVICE — image not found"
44                continue
45              fi
46
47              GHCR_IMAGE="$REPO_BASE/$SERVICE:latest"
48              echo "📦 Tagging $LOCAL_IMAGE → $GHCR_IMAGE"
49              docker tag "$LOCAL_IMAGE" "$GHCR_IMAGE" || { echo "❌ Failed to tag $LOCAL_IMAGE"; continue; }
50
51              echo "🚀 Pushing $GHCR_IMAGE"
52              docker push "$GHCR_IMAGE" || echo "⚠️ Push failed — continuing"
53            done
54
```

The pipeline logs below indicate building images from the docker-compose.yml file.

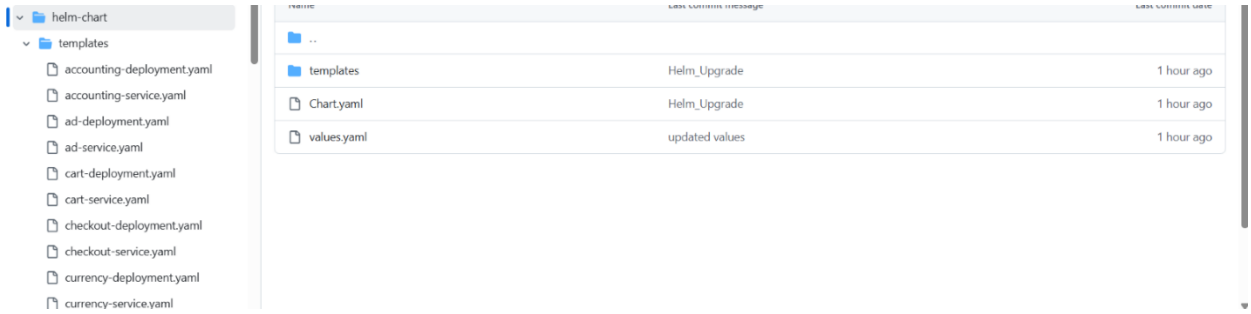


The next step in the workflow ensures that the docker Images are pushed to **personal GHCR**.

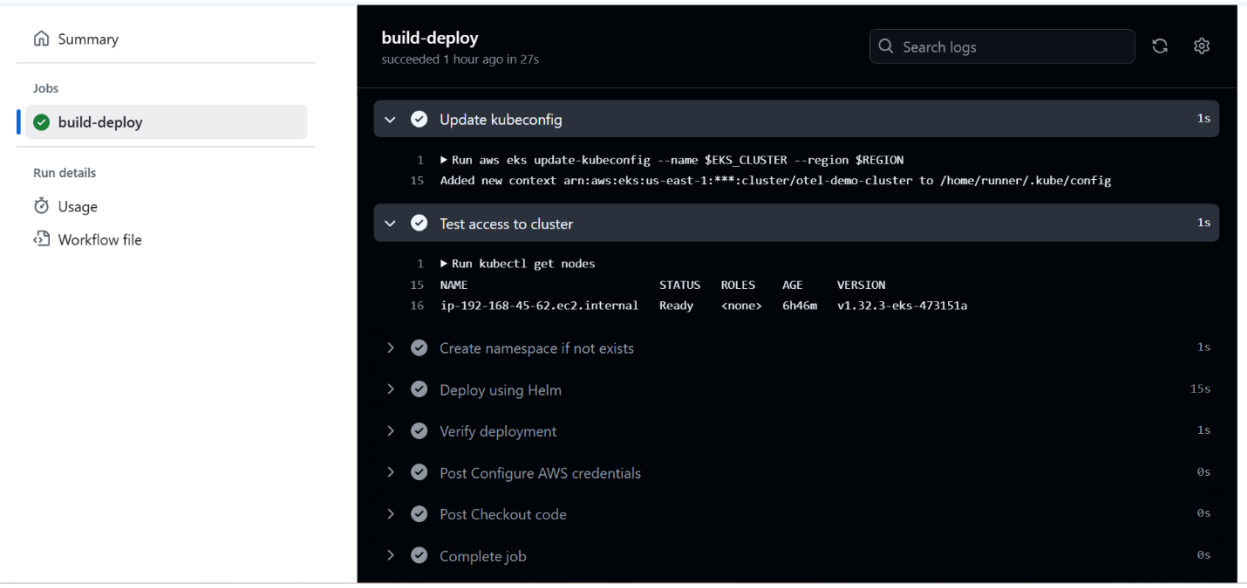


Since these are new docker images in personal GHCR, we had to create a new helm chart.

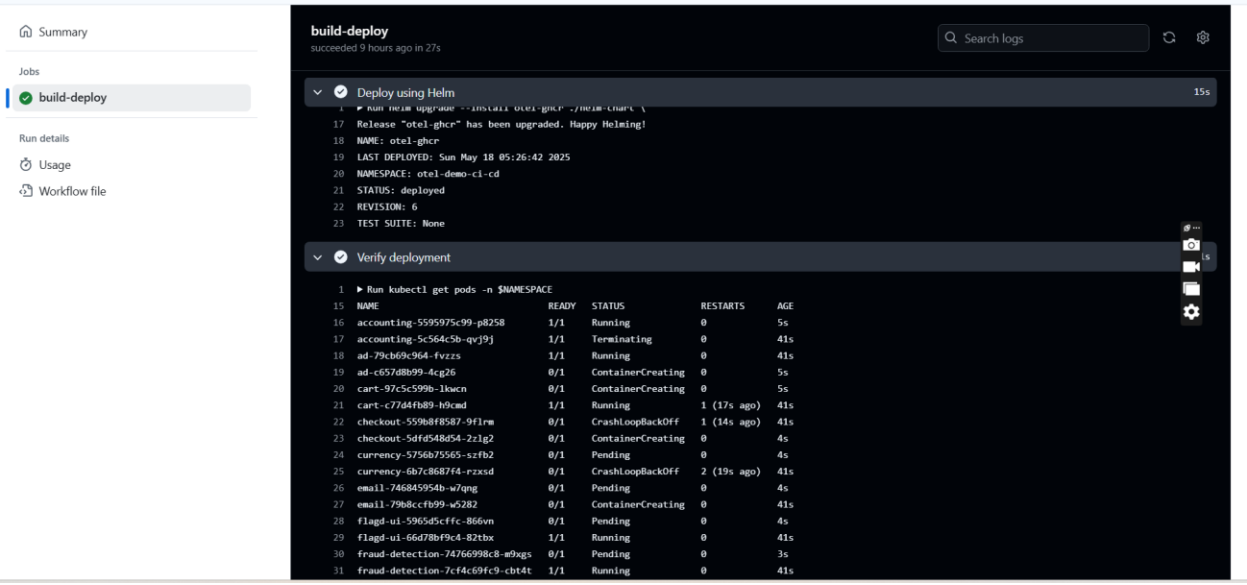
we created a custom helm-structure with the chart.yaml , value.yaml , templates –- deployment, service.yaml



After pushing the images, the workflow connects to Kubernetes EKS cluster.

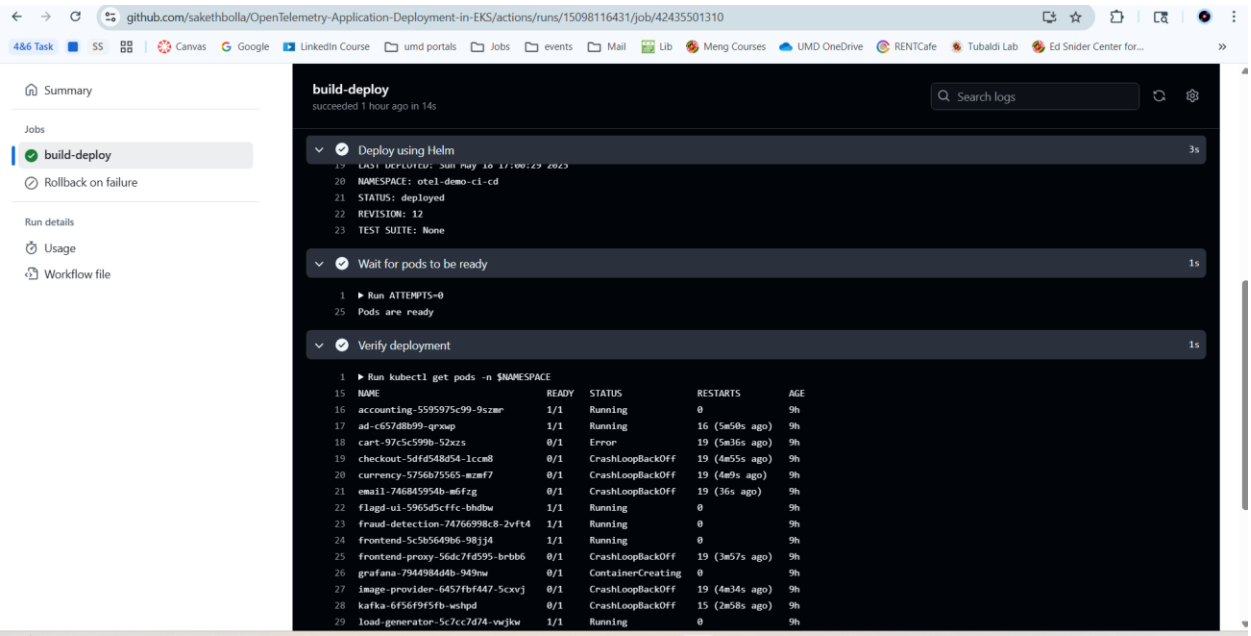


Then, its deployed using helm upgrade command **helm upgrade --install otel-ghcr ./helm-chart \ --namespace $NAMESPACE \ --create-namespace**



4.3 Rollback strategies

The reason many pods are giving CrashLoopBackOff and error is, Services like checkout, payment, product-catalog require DB connection strings or REDIS_HOST, KAFKA_BROKER, etc. Which weren't present due to custom Helm upgrade, and custom docker images.



This should not affect the application, so we have integrated a simple rollback command in our pipeline to revert our application back whenever we face any issue.

To test out rollback, I performed.

Failed build

4.4 Testing Stage in CI/CD Pipeline

The testing stage in our CI/CD pipeline ensures the application components are functionally verified after deployment but before they are promoted as stable. This acts as a gatekeeper to catch deployment issues like service crashes, Helm misconfigurations, or failing pods before the change reaches production.

What We Test (Testing Pipeline**)**

1. Helm Tests: We run built-in Helm release tests to validate Kubernetes objects are healthy and responsive.
2. Smoke Tests: These validate basic application functionality post-deployment, such as service reachability and response codes.
3. Pod Health & Readiness: Kubernetes native kubectl get pods and kubectl describe pod help ensure all containers are in Running or Completed state with zero restarts or error statuses.
4. Manual Port-Forward Verification: We expose services like frontend-proxy locally.

Successful build

Conclusion

Through the successful deployment of the OpenTelemetry Demo Application on AWS, this project validated a range of advanced cloud-native technologies and DevOps practices in a real-world context. The resulting architecture is not only scalable and observable but also demonstrates key operational principles required for modern distributed systems.

A major outcome of this project was the deep integration of automated infrastructure, security, monitoring, and CI/CD. Fine-grained AWS networking, including multi-AZ VPC layouts and strict security group policies, established a secure operational baseline. The use of modular Kubernetes resource definitions and Helm chart templating enabled rapid iteration and reproducibility across environments.

The monitoring stack—leveraging Prometheus and Grafana—provided actionable observability, with custom alerting and real-time notifications closing the feedback loop for operational awareness. The automation of build, test, deployment, and rollback processes via GitHub Actions ensured rapid, reliable delivery, while also illustrating how to integrate Helm and Kubernetes best practices into a CI/CD workflow.

Key technical lessons learned include:

- Modularization and Automation: Breaking down large manifests and using Helm charts streamlined complex deployments, simplified upgrades, and reduced operational risk.
- Observability by Design: Embedding telemetry, alerting, and dashboards at every layer provided immediate feedback and made troubleshooting more efficient.
- Resilience and Recovery: Simulating failures and rollbacks tested the platform's ability to self-heal and maintain service continuity under adverse conditions.
- Security and Compliance: Automating credential management and restricting network access minimized exposure and aligned with least-privilege principles.

This platform is now positioned for further extension—whether by scaling workloads, integrating additional telemetry sources, or adapting the CI/CD pipeline to new application requirements. The project's architecture and automation patterns serve as a blueprint for future microservices-based observability solutions on AWS or similar cloud environments.