University of Maryland - College Park

Cloud Computing Final Project

EKS AND MONITORING WITH OPENTELEMETRY

Group 11

Project Files - GitHub

Video Demo

Jayesh Pamnani
Manav Gupta
Chetan Rajesh
Sagar Wagh

Contents

Phase Assignment and Contribution Breakdown

	Group/Individual	Assignee
Phase 1 → 20%	Group	Group Effort
Phase 2 → 20%	Group	Group Effort
Phase 3	Individual	Manav Gupta
Phase 4	Individual	Jayesh Pamnani
Phase 5	Individual	Chetan Rajesh
Phase 6	Individual	Sagar Wagh
Phase 7 → 35%	Group	Group Effort

Important Links:

Project Files - GitHub

Video Demo

Infrastructure Automation Using AWS CloudFormation

To ensure an efficient, scalable, and reliable setup of our Kubernetes environment in AWS, we leveraged **AWS CloudFormation** to automate the creation and management of critical infrastructure components. CloudFormation allows us to define infrastructure as code (IaC), providing numerous advantages such as automation, consistency, scalability, version control and efficiency.

In this project, we used CloudFormation to provision the following components:

1. VPC and Networking
2. Security Groups
3. EKS Cluster
4. EKS Nodes Group
5. Client EC2 Instance

VPC and Networking CloudFormation Template:

This CloudFormation template sets up a **multi-AZ VPC environment** tailored for the EKS cluster. It provides a robust and scalable network foundation with high availability and fault tolerance. The main components provisioned are:

1. **VPC**:
    - A virtual private cloud with a CIDR block of 10.0.0.0/16.
    - DNS support and hostnames enabled for seamless resource resolution.
2. **Subnets**:
    - **Two public subnets** for internet-accessible resources such as NAT Gateways.
    - **Two private subnets** for secure communication and resource isolation of EKS worker nodes.
    - Subnets are distributed across multiple Availability Zones (AZs) for high availability.
3. **Internet Gateway**:
    - Enables internet connectivity for resources in the public subnets.
4. **NAT Gateways**:
    - Two **NAT Gateways** in different public subnets to allow outbound internet access for resources in private subnets while maintaining their security.
5. **Route Tables**:
    - Separate route tables for public and private subnets:
        - Public route table with a route to the internet gateway.
        - Private route tables with routes to respective NAT Gateways.

Outputs:

- **VPC ID**: Reference to the created VPC.
- **Public Subnets**: List of public subnet IDs.
- **Private Subnets**: List of private subnet IDs.

This template ensures secure and highly available networking infrastructure that supports the EKS cluster and its workloads.

Security Group CloudFormation Template:

This template provisions **security groups** that enforce network traffic control for the EKS cluster and its worker nodes, ensuring secure communication within the cluster.

1. **Cluster Security Group**:
    - Provides security for the EKS control plane.
    - Allows inbound traffic on port **443 (HTTPS)** from any IP (for simplicity, could be restricted in production).
2. **Node Security Group**:
    - Facilitates communication between worker nodes and the control plane.
    - Inbound rules include:
        - Port **443** and ephemeral ports (**1025-65535**) from the control plane for secure communication.
        - Full internal communication within the VPC CIDR for unrestricted node-to-node communication.

Outputs:

- **Cluster Security Group ID**: Security Group ID for the EKS control plane.
- **Node Security Group ID**: Security Group ID for the worker nodes.

This template ensures the cluster's security by isolating external access and enabling secure internal communication.

EKS Cluster CloudFormation Template:

This template sets up the **EKS cluster control plane**, which manages Kubernetes workloads. It ensures the cluster is securely integrated with the VPC and allows API access through defined security rules.

1. **EKS Cluster Role**:
    - An IAM role assigned to the EKS cluster for accessing AWS resources.
    - Attached with AmazonEKSClusterPolicy.
2. **EKS Cluster**:
    - Creates an EKS cluster named **otel-demo**.
    - Configured with multiple **subnets** for high availability.
    - Integrated with a **security group** for controlled network access.
    - Latest Kubernetes version (1.31) is deployed for compatibility and performance.

Outputs:

- **Cluster Name**: Name of the EKS cluster.
- **Cluster Endpoint**: API endpoint for cluster communication.
- **Cluster Role ARN**: Role ARN associated with the cluster.

This template ensures that the EKS control plane is highly available and securely managed.

EKS Node Group CloudFormation Template:

This template provisions **managed worker nodes** for the EKS cluster. The nodes are auto-scaled and designed to run containerized workloads efficiently.

1. **Node IAM Role**:
    - An IAM role assigned to worker nodes with policies for EKS operations:
        - AmazonEKSWorkerNodePolicy
        - AmazonEC2ContainerRegistryReadOnly
        - AmazonEKS_CNI_Policy
2. **Node Group**:
    - Managed node group connected to the EKS cluster.
    - Configured with **public subnets** for deployment.
    - **Instance type**: t3.large, suitable for moderate workloads.
    - **Scaling configuration**:
        - Minimum: 2 nodes
        - Maximum: 5 nodes
        - Desired: 2 nodes
    - SSH access enabled using a **KeyPair** for administration.
    - Tags included for environment tracking (Project: otel-demo, Environment: production).

Outputs:

- **Node Group Name**: Identifier for the node group.
- **Node Role ARN**: IAM Role ARN assigned to the nodes.

This ensures auto-scaled worker nodes are seamlessly integrated with the cluster, capable of scaling based on workload.

EC2 Client Instance CloudFormation Template:

This template provisions an **EC2 instance** with pre-installed tools for managing the EKS cluster, serving as a client machine for Kubernetes administration.

1. **EC2 Instance**:
    - Instance type: **t3.micro** (or other supported types for flexibility).
    - Connected to a **public subnet** for internet access.
    - Configured with an **IAM Role** providing access to EKS, ECR, and other necessary AWS services.
    - **Security Group** allows inbound traffic on ports **22 (SSH)**, **80 (HTTP)**, and **443 (HTTPS)** for remote access and administration.
2. **Pre-installed Tools**:
    - **AWS CLI** for managing AWS services.
    - **kubectl** for Kubernetes management.
    - **eksctl** for simplified EKS operations.
3. **User Data Script**:
    - Installs and configures tools upon instance launch.
    - Automatically updates kubeconfig for EKS cluster management.

Outputs:

- **Client Instance ID**: Identifier for the EC2 instance.
- **Client Public IP**: Public IP for accessing the instance.
- **Client Security Group ID**: Security group associated with the instance.

This setup provides an administrator-friendly environment for seamless management and monitoring of the EKS cluster and workloads

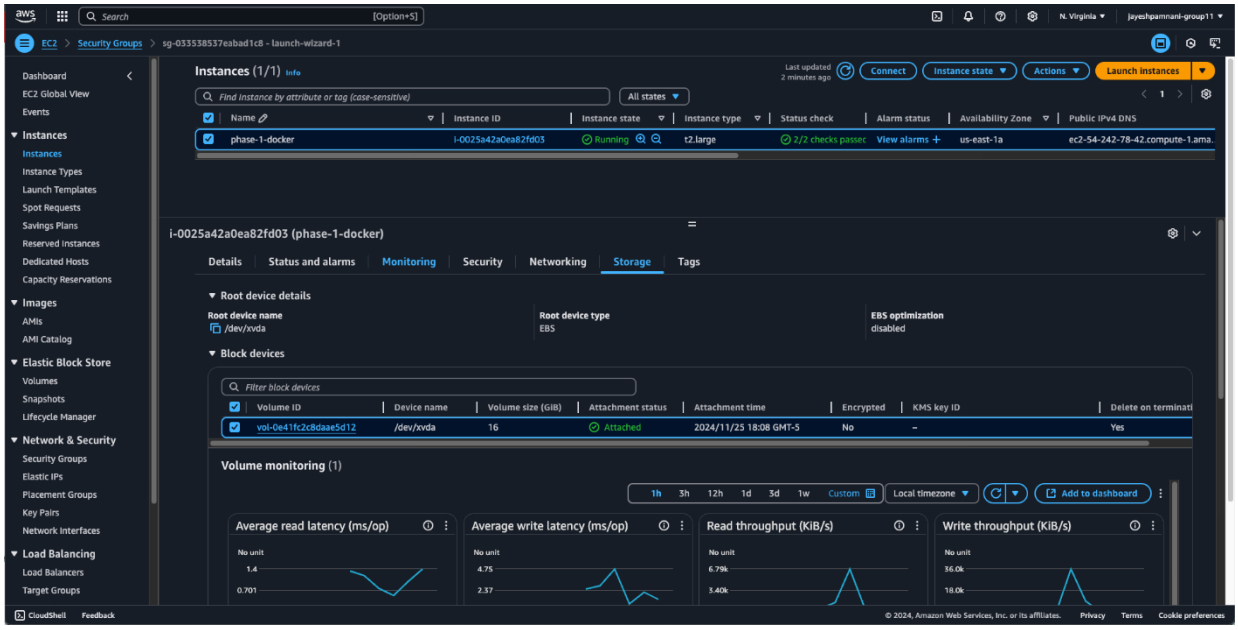
1. Environment and Initial Application Setup

1.1 Docker Deployment

1.1.1 Deployment Environment Setup

Provisioned an **EC2 instance** with the following specifications:

- **Instance Type**: t3.large (or equivalent).
- **Storage**: 16 GB attached to the instance.



1.1.2 Application Deployment

Installed and configured **Docker** and **Docker Compose** on the instance using the following steps:

- Install Docker:
  - sudo yum install docker -y
- Start and enable Docker:
  - sudo systemctl start docker
  - sudo systemctl enable docker
- Add ec2-user to the Docker group:
  - sudo usermod -aG docker ec2-user
  - newgrp docker
- Download and Install Docker Compose
  - sudo curl -L "https://github.com/docker/compose/releases/download/v2.20.0/docker-compose-$(uname -s)-$(uname -m)" -o
  - /usr/local/bin/docker-compose

- sudo chmod +x /usr/local/bin/docker-compose

- Verify the installation:
  - docker-compose –version
- Deploy Application Using Docker Compose
  - Clone the repository containing the application:
    - git clone https://github.com/open-telemetry/opentelemetry-demo.git
    - cd opentelemetry-demo
- Run Docker Compose
  - docker-compose up -d

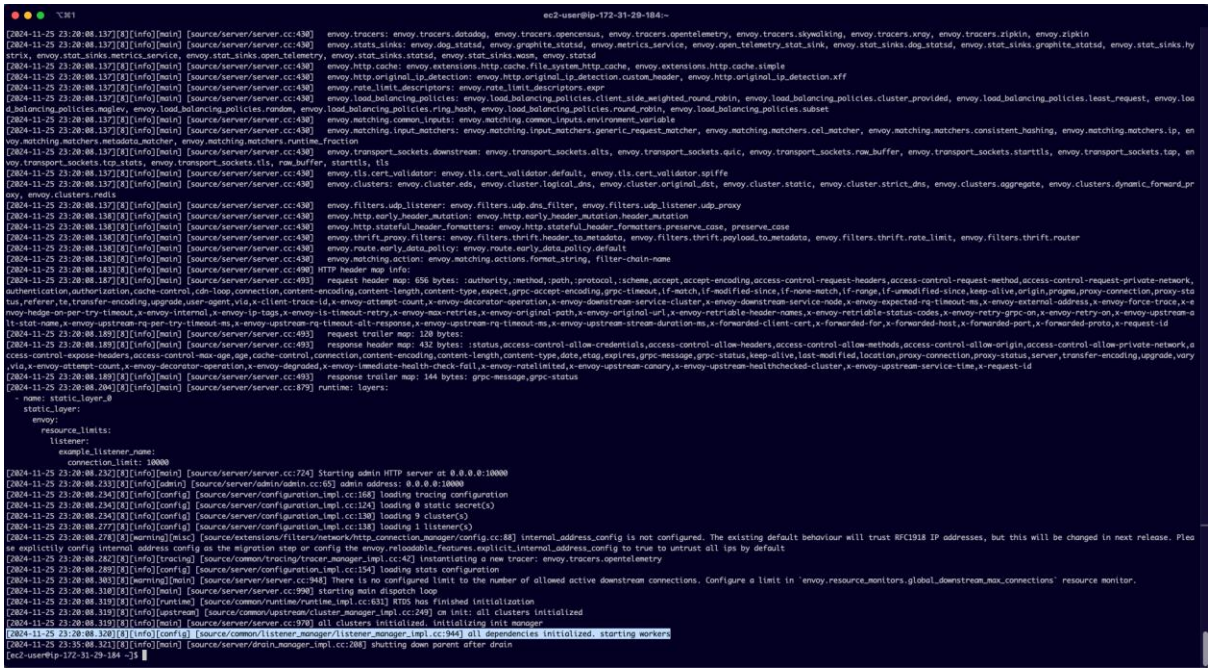
1.1.3 Validating Deployment

Verified service using docker ps and captured logs for each service to confirm successful startup using commands:

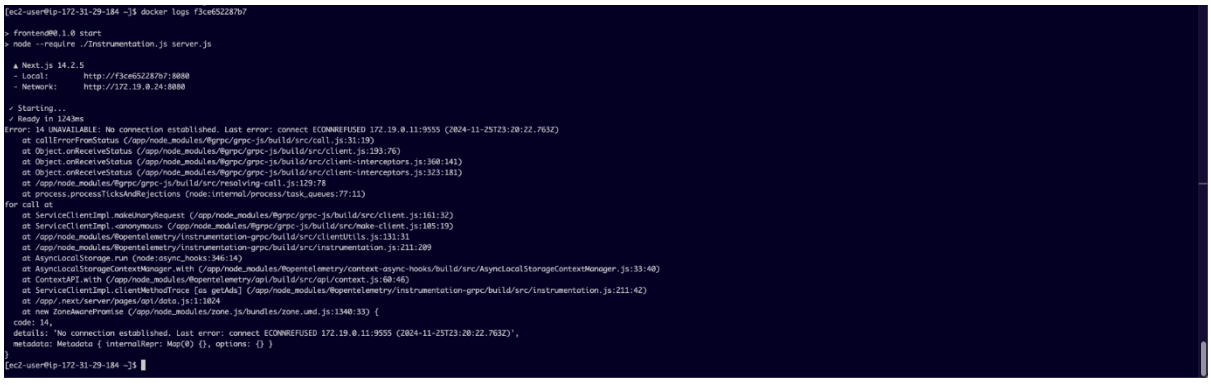
- docker ps

- docker-compose logs

FrontendProxy Logs:



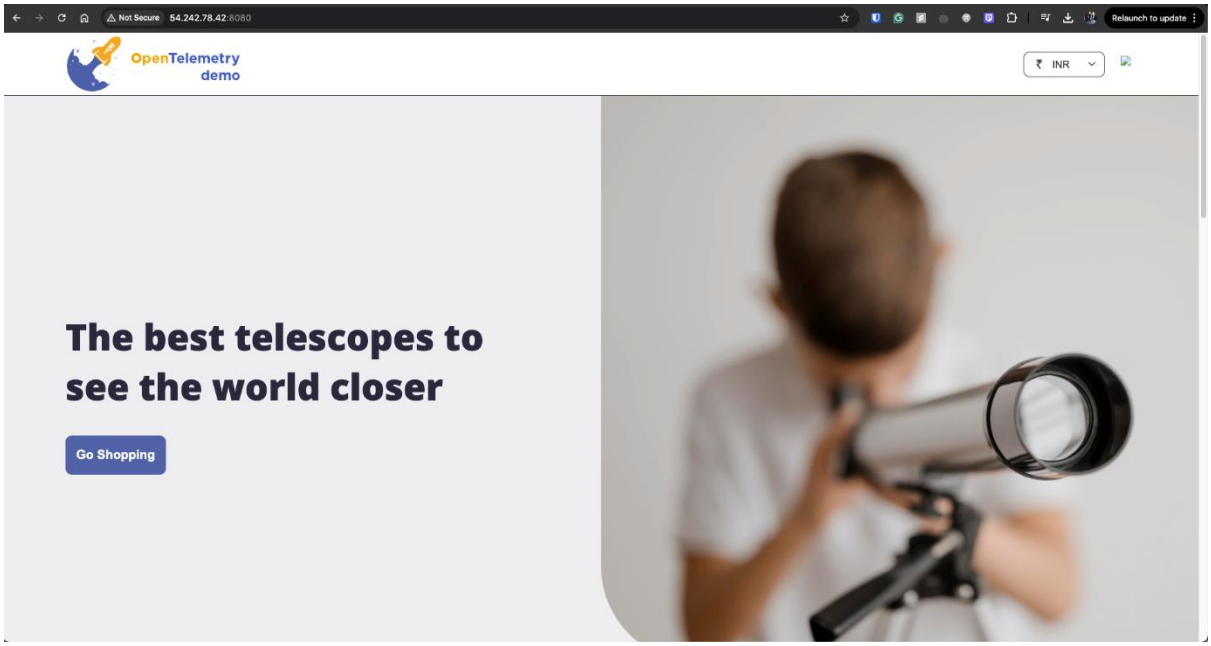
Docker application logs for Frontend



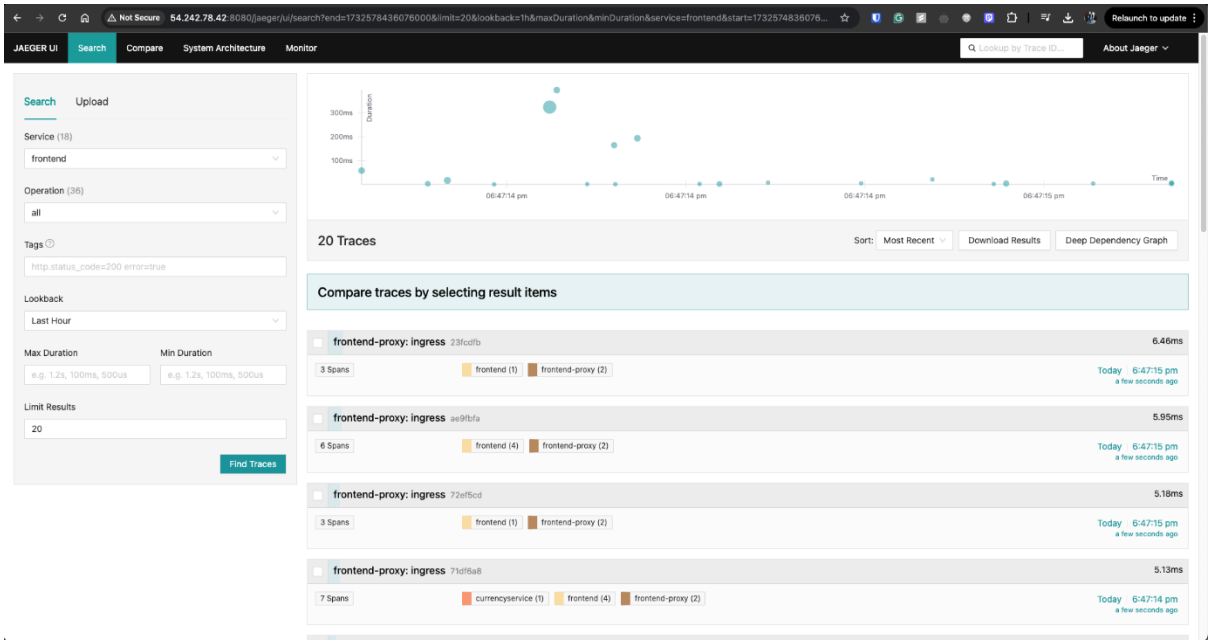
1.1.4 Accessing the Application

Accessed the application endpoints exposed by the containers to ensure proper functionality (e.g., frontend, backend, database, or other services) at http://<public-ip>:8080.

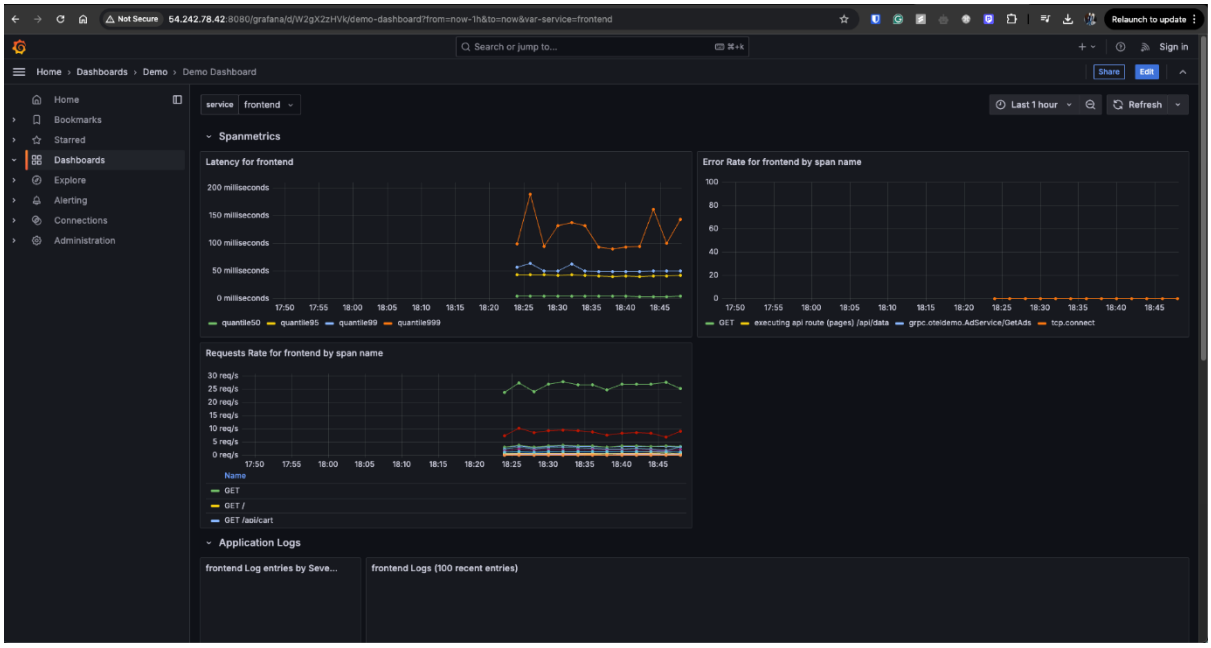
Application @8080 Port



Application Jaeger UI



Application Grafana



1.2 Kubernetes Setup Tasks

The setup utilized AWS CloudFormation templates for automation and consistency.

1.2.1 EKS Cluster Setup

Created an EKS Cluster using a CloudFormation template to automate resource provisioning:

- Control Plane in a specified VPC and subnets.
- Worker Nodes:
  - At least 2 nodes.
  - Instance Type: t3.large.
- Deployed and configured cluster resources with required IAM roles.

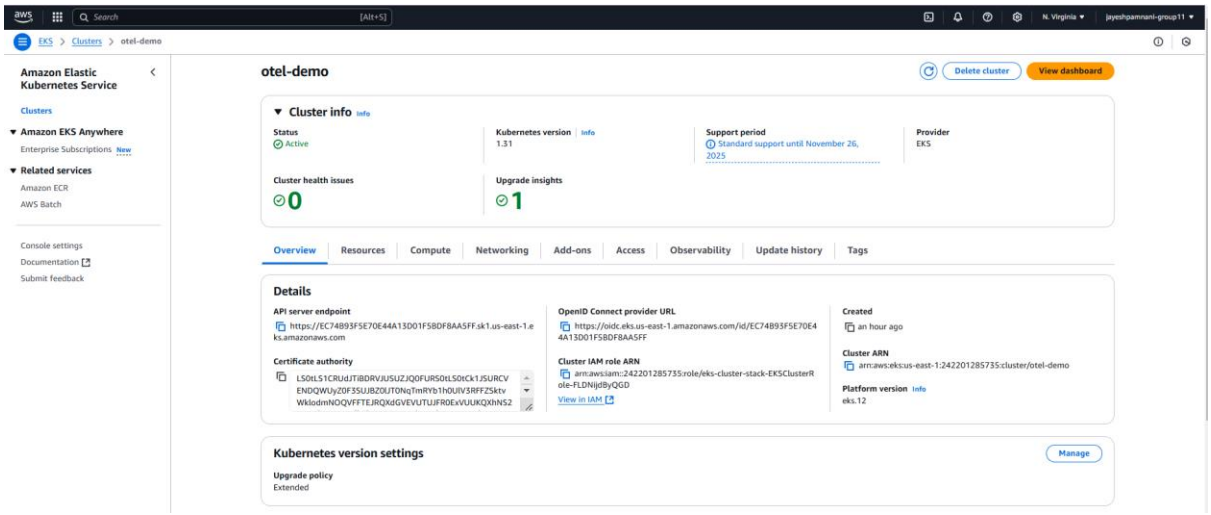
CloudFormation Template Reference: EKS Cluster CloudFormation Template, EKS Node Groups CloudFormation Template

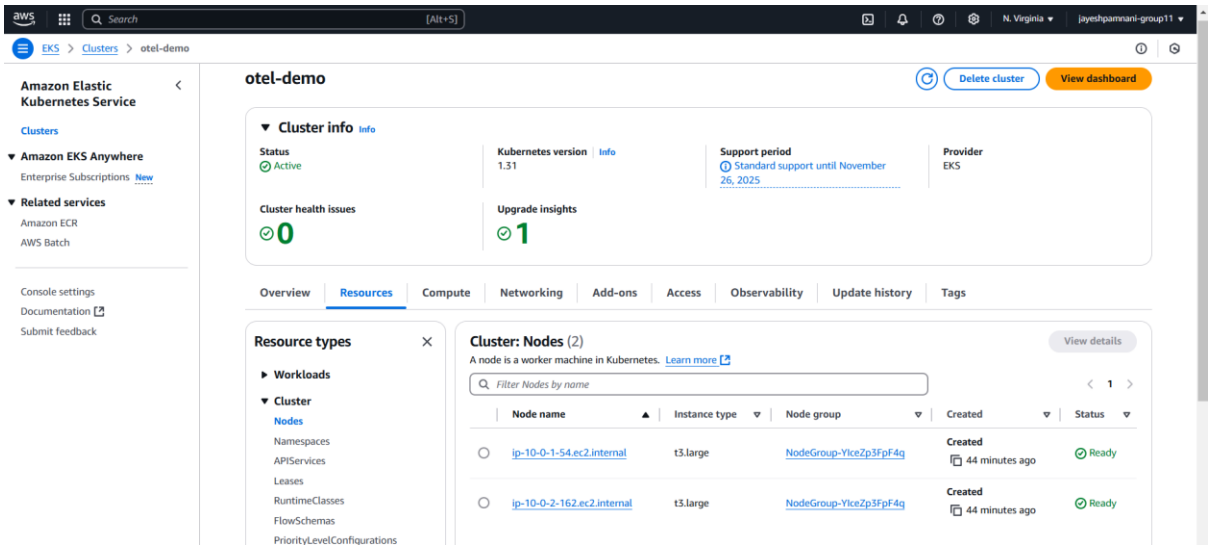
Cluster Details:

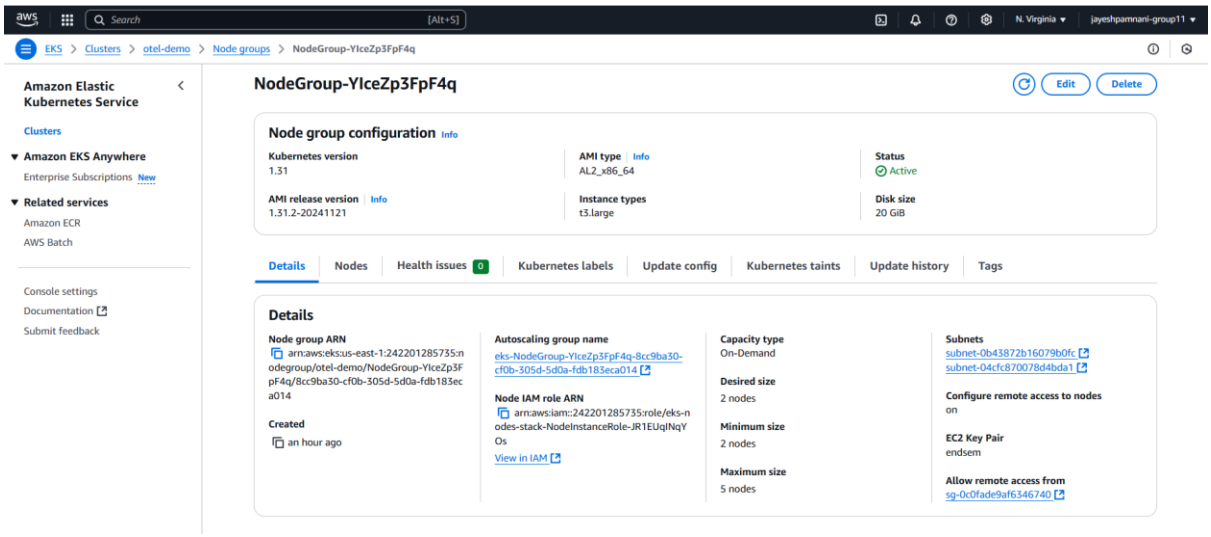
- **Cluster Name**: otel-demo
- **Kubernetes Version**: 1.31
- **Cluster Role ARN**: arn:aws:iam::242201285753:role/eks-cluster-stack-EKSClusterRole-JTREuJqNgY (visible in IAM link)
- **Cluster ARN**: arn:aws:eks:us-east-1:242201285753:cluster/otel-demo
- **Security Group**: otel-demo-cluster-sg

Node Group Details:

- **Node Group Name**: NodeGroup-YIe2z3FpF4q
- **Instance Type**: t3.large
- **Node Group ARN**: arn:aws:eks:us-east-1:242201285753:nodegroup/otel-demo/NodeGroup-YIe2z3FpF4q/8c8a3e90-cf0b-305d-a0b6-fb8183c8e014
- **Node IAM Role ARN**: arn:aws:iam::242201285735:role/eks-cluster-stack-EKSNodeInstanceRole-JREUqINqYOs


otel-demo
Delete cluster
View dashboard
Cluster info
Status
Active
Kubernetes version
1.31
Support period
Standard support until November 26, 2025
Provider
EKS
Cluster health issues
0
Upgrade insights
1
Overview
Resources
Compute
Networking
Add-ons
Access
Observability
Update history
Tags
Details
API server endpoint
https://EC74B93F5E70E44A13D01F5BDF8AA5FF.sk1.us-east-1.eks.amazonaws.com
OpenID Connect provider URL
https://oidc.eks.us-east-1.amazonaws.com/id/EC74B93F5E70E44A13D01F5BDF8AA5FF
Created
an hour ago
Certificate authority
Cluster IAM role ARN
arn:aws:iam::242201285735:role/eks-cluster-stack-EKSClusterRole-FLDNjdByQGD
Cluster ARN
arn:aws:eks:us-east-1:242201285735:cluster/otel-demo
View in IAM
Platform version
eks.12
Kubernetes version settings
Manage
Upgrade policy
Extended


otel-demo
Delete cluster
View dashboard
Cluster info
Status
Active
Kubernetes version
1.31
Support period
Standard support until November 26, 2025
Provider
EKS
Cluster health issues
0
Upgrade insights
1
Overview
Resources
Compute
Networking
Add-ons
Access
Observability
Update history
Tags
Resource types
Workloads
Cluster
Nodes
Namespaces
APIServices
Leases
RuntimeClasses
FlowSchemas
PriorityLevelConfigurations
Cluster: Nodes (2)
A node is a worker machine in Kubernetes.
Learn more
View details
Filter Nodes by name
Node name
Instance type
Node group
Created
Status
ip-10-0-1-54.ec2.internal
t3.large
NodeGroup-YIceZp3FpF4q
Created
44 minutes ago
Ready
ip-10-0-2-162.ec2.internal
t3.large
NodeGroup-YIceZp3FpF4q
Created
44 minutes ago
Ready


NodeGroup-YIceZp3FpF4q
Edit
Delete
Node group configuration
Kubernetes version
1.31
AMI type
AL2_x86_64
Status
Active
AMI release version
1.31.2-20241121
Instance types
t3.large
Disk size
20 GiB
Details
Nodes
Health issues
0
Kubernetes labels
Update config
Kubernetes taints
Update history
Tags
Details
Node group ARN
arn:aws:eks:us-east-1:242201285735:nodegroup/otel-demo/NodeGroup-YIceZp3FpF4q/8cc9ba30-cf0b-305d-5d0a-fdb183eca014
Autoscaling group name
eks-NodeGroup-YIceZp3FpF4q-8cc9ba30-cf0b-305d-5d0a-fdb183eca014
Capacity type
On-Demand
Subnets
subnet-0b43872b16079b0fc
subnet-04cfc870078d4bda1
Created
an hour ago
Node IAM role ARN
arn:aws:iam::242201285735:role/eks-nodes-stack-NodeInstanceRole-JR1EUqINqYOs
View in IAM
Desired size
2 nodes
Minimum size
2 nodes
Maximum size
5 nodes
Configure remote access to nodes
on
EC2 Key Pair
endsem
Allow remote access from
sg-0c0fade9af6346740

1.2.2 EKS Client Instance Configuration

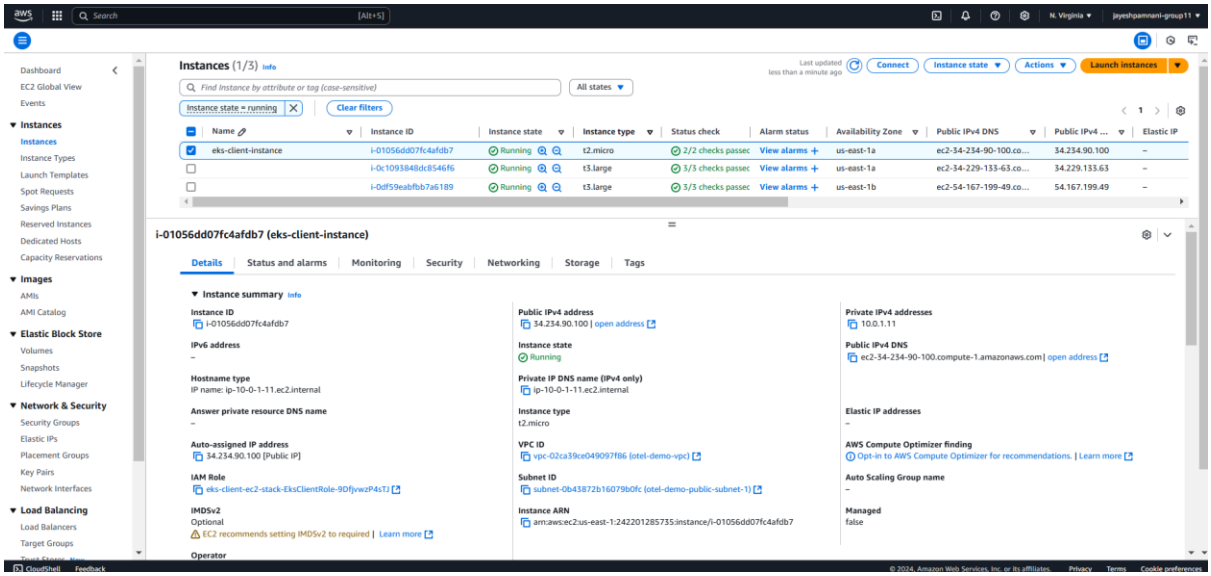
Provisioned a dedicated EC2 instance using a CloudFormation template:

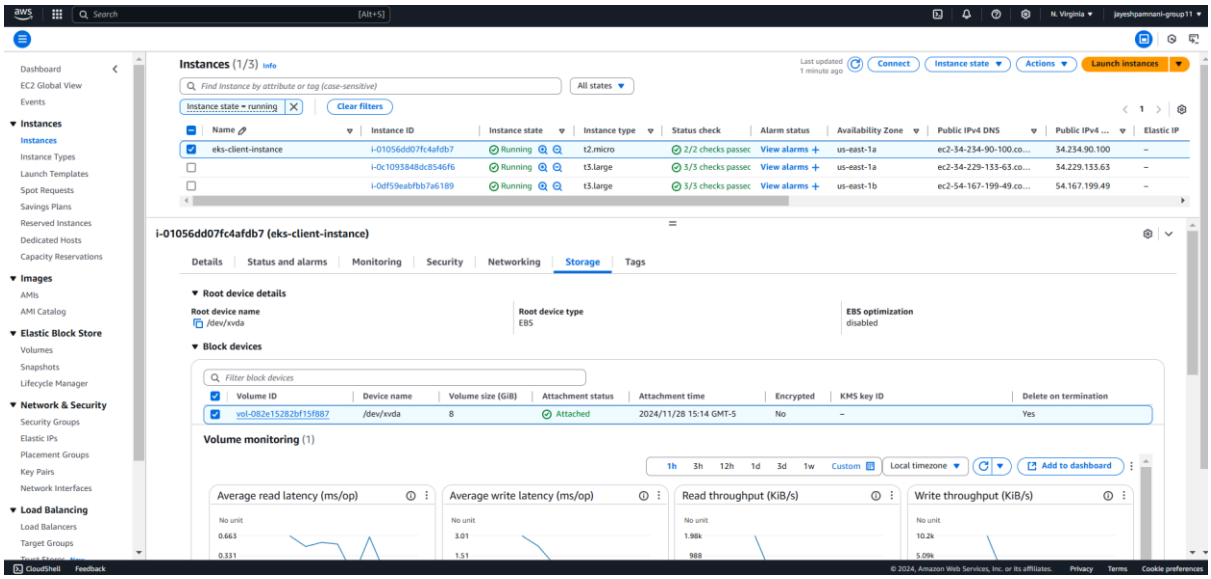
- Instance Type: t3.micro
- Storage: 16 GB attached.

Installed and configured tools:

- kubectl
- AWS CLI
- Eksctl

CloudFormation Template Reference: [EC2 Client Instance CloudFormation Template]





1.2.3 Application Deployment

Used the **opentelemetry-demo.yaml** Kubernetes manifest for deploying the OpenTelemetry Demo application.

- git clone https://github.com/open-telemetry/opentelemetry-demo.git
- cd opentelemetry-demo/Kubernetes
- kubectl apply --namespace otel-demo -f opentelemetry-demo.yaml

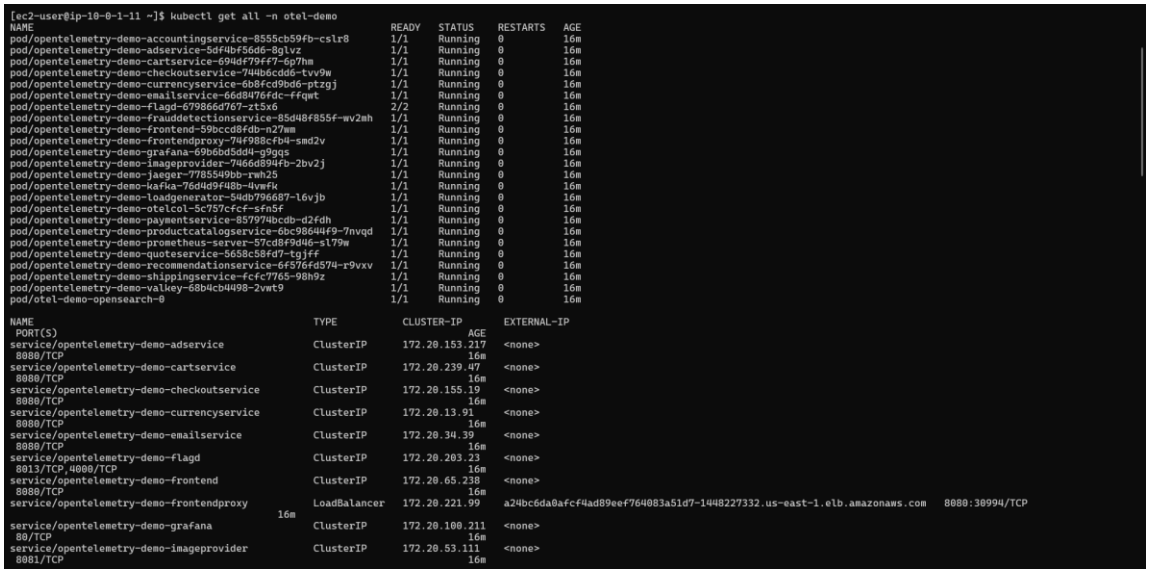
Validated the deployment:

- kubectl get all -n otel-demo

Validation:

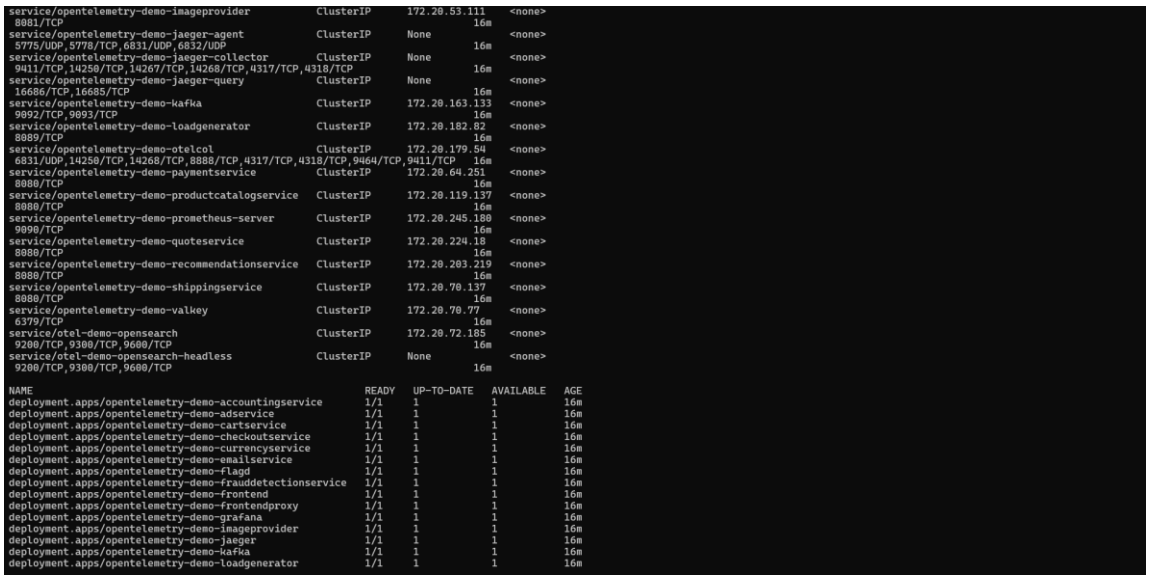
Ensured all resources in the otel-demo namespace were running as expected. Accessed application endpoints via port-forwarding or LoadBalancer.

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



```
[ec2-user@ip-10-0-1-11 ~]$ kubectl get all -n otel-demo
NAME                                                  READY   STATUS    RESTARTS   AGE
pod/opentelemetry-demo-accountingservice-8555cb59fb-cslr8      1/1     Running   0          16m
pod/opentelemetry-demo-adservice-5df4bf56d6-8glvz              1/1     Running   0          16m
pod/opentelemetry-demo-cartservice-694df79ff7-6p7hm            1/1     Running   0          16m
pod/opentelemetry-demo-checkoutservice-744b6cdd6-tvv9w         1/1     Running   0          16m
pod/opentelemetry-demo-currencyservice-6b8fcd9bd6-ptzgj        1/1     Running   0          16m
pod/opentelemetry-demo-emailservice-66d8476fdc-ffqwt           1/1     Running   0          16m
pod/opentelemetry-demo-flagd-679866d767-zt5x6                  2/2     Running   0          16m
pod/opentelemetry-demo-frauddetectionservice-85d48f855f-wv2mh  1/1     Running   0          16m
pod/opentelemetry-demo-frontend-59bccd8fdb-n27wm               1/1     Running   0          16m
pod/opentelemetry-demo-frontendproxy-74f988cfb4-smd2v          1/1     Running   0          16m
pod/opentelemetry-demo-grafana-69b6bd5dd4-g9ggs                1/1     Running   0          16m
pod/opentelemetry-demo-imageprovider-7466d894fb-2bv2j          1/1     Running   0          16m
pod/opentelemetry-demo-jaeger-7785549bb-rwh25                  1/1     Running   0          16m
pod/opentelemetry-demo-kafka-76d4d9f48b-4vmfk                  1/1     Running   0          16m
pod/opentelemetry-demo-loadgenerator-54db796687-l6vjb          1/1     Running   0          16m
pod/opentelemetry-demo-otelcol-5c757cfcf-s6fn5f                1/1     Running   0          16m
pod/opentelemetry-demo-paymentservice-857974bcdb-d2fdh         1/1     Running   0          16m
pod/opentelemetry-demo-productcatalogservice-6bc98644f9-7nvqd  1/1     Running   0          16m
pod/opentelemetry-demo-prometheus-server-57cd8f9d46-sl79w      1/1     Running   0          16m
pod/opentelemetry-demo-quoteservice-5658c58fd7-tgjff           1/1     Running   0          16m
pod/opentelemetry-demo-recommendationservice-6f576fd574-r9vxv  1/1     Running   0          16m
pod/opentelemetry-demo-shippingservice-fcfc7765-98h9z          1/1     Running   0          16m
pod/opentelemetry-demo-valkey-68b4cb4498-2vwt9                 1/1     Running   0          16m
pod/otel-demo-opensearch-0                                     1/1     Running   0          16m

NAME                                          TYPE           CLUSTER-IP       EXTERNAL-IP
 PORT(S)                                                  AGE
service/opentelemetry-demo-adservice          ClusterIP      172.20.153.217   <none>
 8080/TCP                                                 16m
service/opentelemetry-demo-cartservice        ClusterIP      172.20.239.47    <none>
 8080/TCP                                                 16m
service/opentelemetry-demo-checkoutservice    ClusterIP      172.20.155.19    <none>
 8080/TCP                                                 16m
service/opentelemetry-demo-currencyservice    ClusterIP      172.20.13.91     <none>
 8080/TCP                                                 16m
service/opentelemetry-demo-emailservice       ClusterIP      172.20.34.39     <none>
 8080/TCP                                                 16m
service/opentelemetry-demo-flagd              ClusterIP      172.20.203.23    <none>
 8013/TCP,4000/TCP                                        16m
service/opentelemetry-demo-frontend           ClusterIP      172.20.65.238    <none>
 8080/TCP                                                 16m
service/opentelemetry-demo-frontendproxy      LoadBalancer   172.20.221.99    a24bc6da0afcf4ad89eef764083a51d7-1448227332.us-east-1.elb.amazonaws.com   8080:30994/TCP
                                    16m
service/opentelemetry-demo-grafana            ClusterIP      172.20.100.211   <none>
 80/TCP                                                   16m
service/opentelemetry-demo-imageprovider      ClusterIP      172.20.53.111    <none>
 8081/TCP                                                 16m
```



```
service/opentelemetry-demo-imageprovider          ClusterIP      172.20.53.111    <none>
 8081/TCP                                                 16m
service/opentelemetry-demo-jaeger-agent           ClusterIP      None             <none>
 5775/UDP,5778/TCP,6831/UDP,6832/UDP                      16m
service/opentelemetry-demo-jaeger-collector       ClusterIP      None             <none>
 9411/TCP,14250/TCP,14267/TCP,14268/TCP,4317/TCP,4318/TCP 16m
service/opentelemetry-demo-jaeger-query           ClusterIP      None             <none>
 16686/TCP,16685/TCP                                      16m
service/opentelemetry-demo-kafka                  ClusterIP      172.20.163.133   <none>
 9092/TCP,9093/TCP                                        16m
service/opentelemetry-demo-loadgenerator          ClusterIP      172.20.182.82    <none>
 8089/TCP                                                 16m
service/opentelemetry-demo-otelcol                ClusterIP      172.20.179.54    <none>
 6831/UDP,14250/TCP,14268/TCP,8888/TCP,4317/TCP,4318/TCP,9464/TCP,9411/TCP   16m
service/opentelemetry-demo-paymentservice         ClusterIP      172.20.64.251    <none>
 8080/TCP                                                 16m
service/opentelemetry-demo-productcatalogservice  ClusterIP      172.20.119.137   <none>
 8080/TCP                                                 16m
service/opentelemetry-demo-prometheus-server      ClusterIP      172.20.245.180   <none>
 9090/TCP                                                 16m
service/opentelemetry-demo-quoteservice           ClusterIP      172.20.224.18    <none>
 8080/TCP                                                 16m
service/opentelemetry-demo-recommendationservice  ClusterIP      172.20.203.219   <none>
 8080/TCP                                                 16m
service/opentelemetry-demo-shippingservice        ClusterIP      172.20.70.137    <none>
 8080/TCP                                                 16m
service/opentelemetry-demo-valkey                 ClusterIP      172.20.70.77     <none>
 6379/TCP                                                 16m
service/otel-demo-opensearch                      ClusterIP      172.20.72.185    <none>
 9200/TCP,9300/TCP,9600/TCP                               16m
service/otel-demo-opensearch-headless             ClusterIP      None             <none>
 9200/TCP,9300/TCP,9600/TCP                               16m

NAME                                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/opentelemetry-demo-accountingservice      1/1     1            1           16m
deployment.apps/opentelemetry-demo-adservice              1/1     1            1           16m
deployment.apps/opentelemetry-demo-cartservice            1/1     1            1           16m
deployment.apps/opentelemetry-demo-checkoutservice        1/1     1            1           16m
deployment.apps/opentelemetry-demo-currencyservice        1/1     1            1           16m
deployment.apps/opentelemetry-demo-emailservice           1/1     1            1           16m
deployment.apps/opentelemetry-demo-flagd                  1/1     1            1           16m
deployment.apps/opentelemetry-demo-frauddetectionservice  1/1     1            1           16m
deployment.apps/opentelemetry-demo-frontend               1/1     1            1           16m
deployment.apps/opentelemetry-demo-frontendproxy          1/1     1            1           16m
deployment.apps/opentelemetry-demo-grafana                1/1     1            1           16m
deployment.apps/opentelemetry-demo-imageprovider          1/1     1            1           16m
deployment.apps/opentelemetry-demo-jaeger                 1/1     1            1           16m
deployment.apps/opentelemetry-demo-kafka                  1/1     1            1           16m
deployment.apps/opentelemetry-demo-loadgenerator          1/1     1            1           16m
```

```
deployment.apps/opentelemetry-demo-otelcol                1/1     1            1           16m
deployment.apps/opentelemetry-demo-paymentservice         1/1     1            1           16m
deployment.apps/opentelemetry-demo-productcatalogservice  1/1     1            1           16m
deployment.apps/opentelemetry-demo-prometheus-server      1/1     1            1           16m
deployment.apps/opentelemetry-demo-quoteservice           1/1     1            1           16m
deployment.apps/opentelemetry-demo-recommendationservice  1/1     1            1           16m
deployment.apps/opentelemetry-demo-shippingservice        1/1     1            1           16m
deployment.apps/opentelemetry-demo-valkey                 1/1     1            1           16m

NAME                                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/opentelemetry-demo-accountingservice-8555cb59fb       1         1         1       16m
replicaset.apps/opentelemetry-demo-adservice-5df4bf56d6               1         1         1       16m
replicaset.apps/opentelemetry-demo-cartservice-694df79ff7             1         1         1       16m
replicaset.apps/opentelemetry-demo-checkoutservice-744b6cdd6          1         1         1       16m
replicaset.apps/opentelemetry-demo-currencyservice-6b8fcd9bd6         1         1         1       16m
replicaset.apps/opentelemetry-demo-emailservice-66d8476fdc            1         1         1       16m
replicaset.apps/opentelemetry-demo-flagd-679866d767                   1         1         1       16m
replicaset.apps/opentelemetry-demo-frauddetectionservice-85d48f855f   1         1         1       16m
replicaset.apps/opentelemetry-demo-frontend-59bccd8fdb                1         1         1       16m
replicaset.apps/opentelemetry-demo-frontendproxy-74f988cfb4           1         1         1       16m
replicaset.apps/opentelemetry-demo-grafana-69b6bd5dd4                 1         1         1       16m
replicaset.apps/opentelemetry-demo-imageprovider-7466d894fb           1         1         1       16m
replicaset.apps/opentelemetry-demo-jaeger-7785549bb                   1         1         1       16m
replicaset.apps/opentelemetry-demo-kafka-76d4d9f48b                   1         1         1       16m
replicaset.apps/opentelemetry-demo-loadgenerator-54db796687           1         1         1       16m
replicaset.apps/opentelemetry-demo-otelcol-5c757cfcf                  1         1         1       16m
replicaset.apps/opentelemetry-demo-paymentservice-857974bcdb          1         1         1       16m
replicaset.apps/opentelemetry-demo-productcatalogservice-6bc98644f9   1         1         1       16m
replicaset.apps/opentelemetry-demo-prometheus-server-57cd8f9d46       1         1         1       16m
replicaset.apps/opentelemetry-demo-quoteservice-5658c58fd7            1         1         1       16m
replicaset.apps/opentelemetry-demo-recommendationservice-6f576fd574   1         1         1       16m
replicaset.apps/opentelemetry-demo-shippingservice-fcfc7765           1         1         1       16m
replicaset.apps/opentelemetry-demo-valkey-68b4cb4498                  1         1         1       16m

NAME                                     READY   AGE
statefulset.apps/otel-demo-opensearch    1/1     16m
```

1.2.4 Application Validation

Accessed application to validate functionality:

- Used port-forwarding or LoadBalancer to expose services.
- Confirmed endpoints were operational.

Logs from Key Application Pods to confirm successful deployment:

Frontend

```
[ec2-user@ip-10-0-1-11 ~]$ kubectl logs pod/opentelemetry-demo-frontend-59bccd8fdb-n27wm -n otel-demo

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

  ▲ Next.js 14.2.5
  - Local:        http://opentelemetry-demo-frontend-59bccd8fdb-n27wm:8080
  - Network:      http://10.0.1.185:8080

 ✓ Starting...
 ✓ Ready in 597ms
```

Grafana

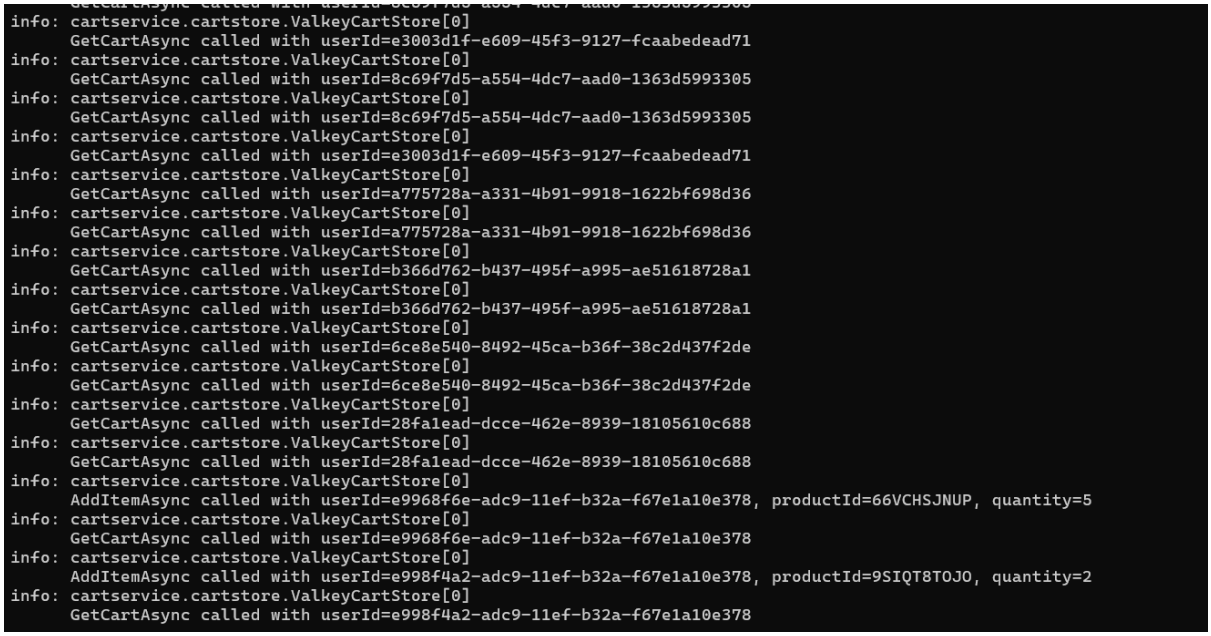
```
[ec2-user@ip-10-0-1-11 ~]$ kubectl logs pod/opentelemetry-demo-grafana-69b6bd5dd4-g9gqs -n otel-demo
✔Downloaded and extracted grafana-opensearch-datasource v2.22.0 zip successfully to /var/lib/grafana/plugins/grafana-opensearch-datasource

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

logger=settings t=2024-11-28T20:23:08.287133715Z level=info msg="Starting Grafana" version=11.3.0 commit=d9455ff7db73b694db7d412e49a68bec767f2b5a branch=HEAD compiled=202
4-11-28T20:23:08Z
logger=settings t=2024-11-28T20:23:08.287627272Z level=info msg="Config loaded from" file=/usr/share/grafana/conf/defaults.ini
logger=settings t=2024-11-28T20:23:08.287643126Z level=info msg="Config loaded from" file=/etc/grafana/grafana.ini
logger=settings t=2024-11-28T20:23:08.287650211Z level=info msg="Config overridden from command line" arg="default.paths.data=/var/lib/grafana/"
logger=settings t=2024-11-28T20:23:08.287656581Z level=info msg="Config overridden from command line" arg="default.paths.logs=/var/log/grafana"
logger=settings t=2024-11-28T20:23:08.287662941Z level=info msg="Config overridden from command line" arg="default.paths.plugins=/var/lib/grafana/plugins"
logger=settings t=2024-11-28T20:23:08.287669396Z level=info msg="Config overridden from command line" arg="default.paths.provisioning=/etc/grafana/provisioning"
logger=settings t=2024-11-28T20:23:08.287676866Z level=info msg="Config overridden from command line" arg="default.log.mode=console"
logger=settings t=2024-11-28T20:23:08.287683528Z level=info msg="Config overridden from Environment variable" var="GF_PATHS_DATA=/var/lib/grafana/"
logger=settings t=2024-11-28T20:23:08.287689793Z level=info msg="Config overridden from Environment variable" var="GF_PATHS_LOGS=/var/log/grafana"
logger=settings t=2024-11-28T20:23:08.287695597Z level=info msg="Config overridden from Environment variable" var="GF_PATHS_PLUGINS=/var/lib/grafana/plugins"
logger=settings t=2024-11-28T20:23:08.287701888Z level=info msg="Config overridden from Environment variable" var="GF_PATHS_PROVISIONING=/etc/grafana/provisioning"
logger=settings t=2024-11-28T20:23:08.287708055Z level=info msg="Config overridden from Environment variable" var="GF_SECURITY_ADMIN_USER=admin"
logger=settings t=2024-11-28T20:23:08.287714237Z level=info msg="Config overridden from Environment variable" var="GF_SECURITY_ADMIN_PASSWORD=*********"
logger=settings t=2024-11-28T20:23:08.287720824Z level=info msg="Target target=[all]
logger=settings t=2024-11-28T20:23:08.287733582Z level=info msg="Path Home" path=/usr/share/grafana
logger=settings t=2024-11-28T20:23:08.287739109Z level=info msg="Path Data" path=/var/lib/grafana/
logger=settings t=2024-11-28T20:23:08.287744491Z level=info msg="Path Logs" path=/var/log/grafana
logger=settings t=2024-11-28T20:23:08.287750084Z level=info msg="Path Plugins" path=/var/lib/grafana/plugins
logger=settings t=2024-11-28T20:23:08.287755787Z level=info msg="Path Provisioning" path=/etc/grafana/provisioning
logger=settings t=2024-11-28T20:23:08.287761492Z level=info msg="App mode production"
logger=featuremgmt t=2024-11-28T20:23:08.288353071Z level=info msg=FeatureToggles cloudWatchCrossAccountQuerying=true addFieldFromCalculationStatFunctions=true cloudWatch
NewLabelParsing=true dashboardSceneForViewers=true transformationsVariableSupport=true transformationsRedesign=true dashboardSceneSolo=true ssoSettingsApi=true accessCont
rolOnCall=true managedPluginsInstall=true groupToNestedTableTransformation=true promQLScope=true lokiQuerySplitting=true prometheusMetricEncyclopedia=true formatString=tr
ue logsInfiniteScrolling=true correlations=true alertingInsights=true autoMigrateXYChartPanel=true prometheusConfigOverhaulAuth=true publicDashboards=true dataplaneFronte
ndFallback=true alertingSimplifiedRouting=true dashboardScene=true alertingNoDataErrorExecution=true lokiMetricDataplane=true publicDashboardsScene=true dashgpt=true nest
edFolders=true annotationPermissionUpdate=true cloudWatchRoundUpEndTime=true lokiQueryHints=true prometheusAzureOverrideAudience=true topnav=true awsAsyncQueryCaching=tru
e lokiStructuredMetadata=true pinNavItems=true influxdbBackendMigration=true tlsMemcached=true notificationBanner=true exploreMetrics=true angularDeprecationUI=true logsC
```

Payment Service

Cartservice



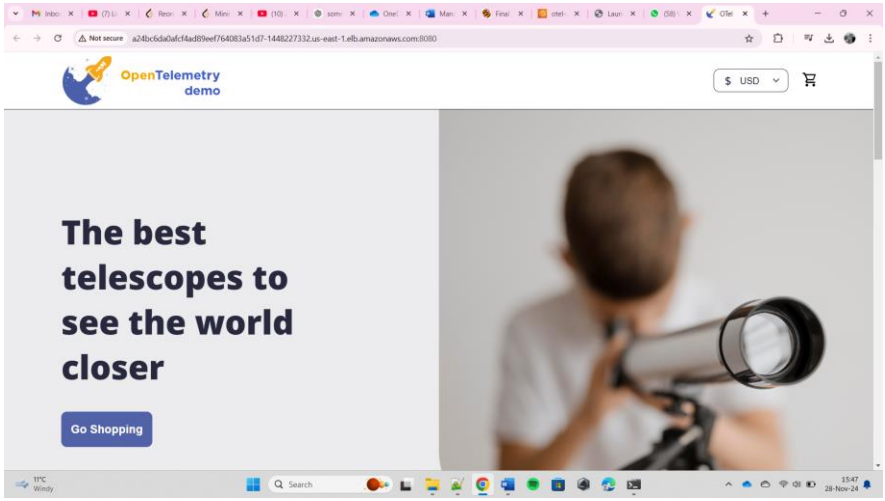
Shipping Service

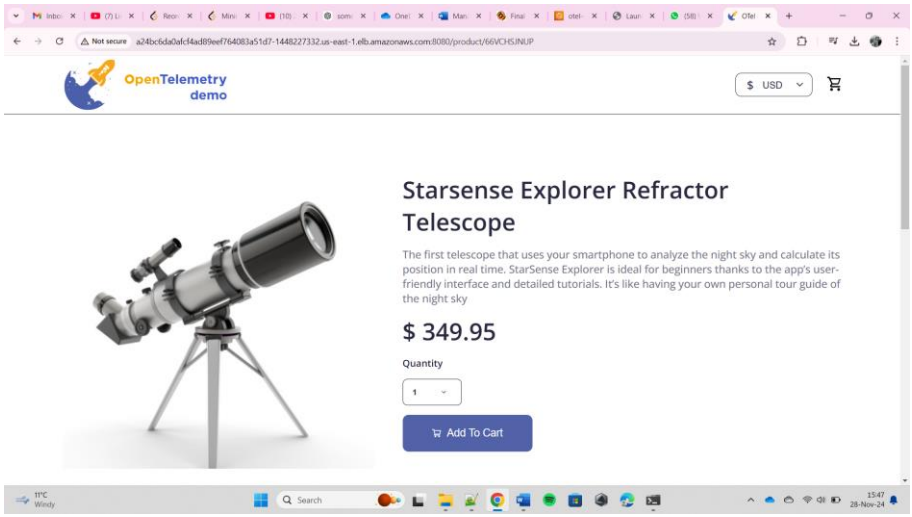
```
[ec2-user@ip-10-0-1-11 ~]$ kubectl logs pod/opentelemetry-demo-shippingservice-fcfc7765-98h9z -n otel-demo
20:22:56 [INFO] OTel pipeline created
20:22:56 [INFO] listening on 0.0.0.0:8080
20:24:23 [INFO] Received quote: "1225.6"
20:24:23 [INFO] Sending Quote: 1225.59
20:24:23 [INFO] Received quote: "996.6"
20:24:23 [INFO] Sending Quote: 996.60
20:24:23 [INFO] Tracking ID Created: 81221474-b112-4f6d-956d-a83f45770553
20:24:23 [INFO] Tracking ID Created: f0c388dd-6f60-46cc-920f-9e1d04934d1c
20:24:26 [INFO] Received quote: "281.6"
20:24:26 [INFO] Sending Quote: 281.60
20:24:26 [INFO] Tracking ID Created: 7bc9d672-851f-4c0f-9f50-62d6442f23fd
20:24:47 [INFO] Received quote: "267"
20:24:47 [INFO] Sending Quote: 267.0
20:24:47 [INFO] Tracking ID Created: 45c2a75a-b4c5-4233-9097-3112f8d65c14
20:24:56 [INFO] Received quote: "380"
20:24:56 [INFO] Sending Quote: 380.0
20:24:56 [INFO] Tracking ID Created: 7a6f1368-8a1f-403c-aaf3-19abf2a39b36
20:25:17 [INFO] Received quote: "339.6"
20:25:17 [INFO] Sending Quote: 339.60
20:25:17 [INFO] Tracking ID Created: 008c5a5c-30b9-4200-b1a1-da07edf11610
20:25:44 [INFO] Received quote: "128.4"
20:25:44 [INFO] Sending Quote: 128.40
20:25:44 [INFO] Tracking ID Created: 21eb88df-23d1-4908-963f-3e0198b21c0b
20:25:55 [INFO] Received quote: "203.1"
20:25:55 [INFO] Sending Quote: 203.10
20:25:55 [INFO] Tracking ID Created: 4e50d228-4392-42ce-9162-652bbad4683d
20:26:47 [INFO] Received quote: "1580"
20:26:47 [INFO] Sending Quote: 1580.0
20:26:47 [INFO] Tracking ID Created: 6fcb7c64-8075-401b-87ee-18adddbd6e3c
20:26:49 [INFO] Received quote: "441"
20:26:49 [INFO] Sending Quote: 441.0
20:26:49 [INFO] Tracking ID Created: 77d9ca7a-f4a8-423b-b526-9c40522091b3
20:26:51 [INFO] Received quote: "324.4"
20:26:51 [INFO] Sending Quote: 324.39
20:26:51 [INFO] Tracking ID Created: c83b515d-900d-4a10-80f5-709c59b4f471
20:27:19 [INFO] Received quote: "455"
20:27:19 [INFO] Sending Quote: 455.0
20:27:19 [INFO] Tracking ID Created: 264892d3-d88e-424c-9e51-558ce0e2e3ba
```

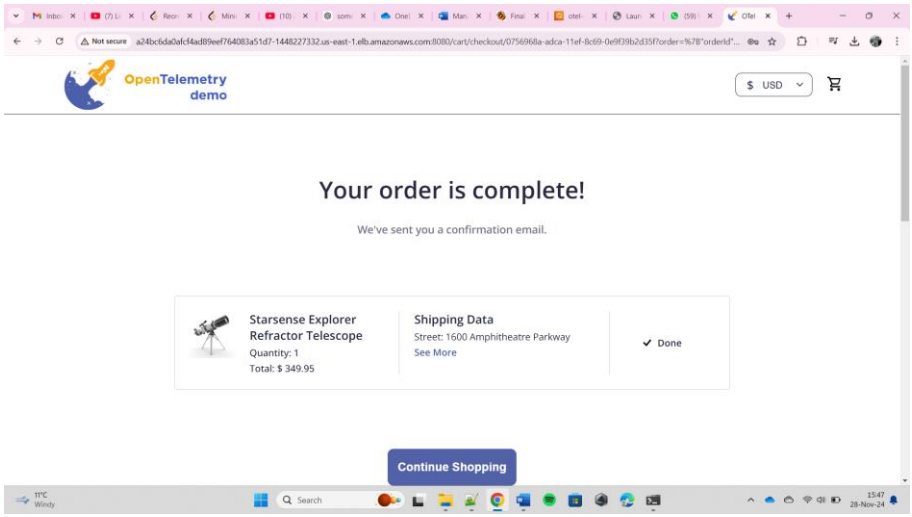
1.2.5 Application Accessibility

```
[ec2-user@ip-10-0-1-11 kubernetes]$ kubectl get svc opentelemetry-demo-frontendproxy -n otel-demo
[ec2-user@ip-10-0-1-11 kubernetes]$
 AGE
opentelemetry-demo-frontendproxy   LoadBalancer   172.20.221.99   a24bc6da0afcf4ad89eef764083a51d7-1448227332.us-east-1.elb.amazonaws.com   8080:30994/TCP   36m
[ec2-user@ip-10-0-1-11 kubernetes]$
```

Accessing the application via the LoadBalancer's EXTERNAL-IP:8080




OpenTelemetry demo
$ USD
Starsense Explorer Refractor Telescope
The first telescope that uses your smartphone to analyze the night sky and calculate its position in real time. StarSense Explorer is ideal for beginners thanks to the app's user-friendly interface and detailed tutorials. It's like having your own personal tour guide of the night sky
$ 349.95
Quantity
1
Add To Cart


OpenTelemetry demo
$ USD
Your order is complete!
We've sent you a confirmation email.
Starsense Explorer Refractor Telescope
Quantity: 1
Total: $ 349.95
Shipping Data
Street: 1600 Amphitheatre Parkway
See More
Done
Continue Shopping

2. YAML Splitting and Modular Deployment

In this phase, we try to modularize the application deployment by splitting the monolithic opentelemetry-demo.yaml file into smaller, logical components. By dividing the configuration into distinct, service-specific YAML files, we can ensure better deployment flexibility, ease of debugging, and more manageable updates. This also allows for deploying resources independently based on their requirements, reducing the risk of affecting other services.

2.1 Original State: Monolithic opentelemetry-demo.yaml

Initially, the deployment process used a single, monolithic opentelemetry-demo.yaml file containing all the Kubernetes resources for the entire application.

2.1.1 The Problem with a Single opentelemetry-demo.yaml File

While convenient for a quick, one-step deployment, it introduced several issues:

- **Hard to Debug**: When one resource failed (e.g., a pod didn't start or a service wasn't accessible), identifying and resolving the issue was challenging since the entire file had to be reapplied.
- **Resource Dependency Problems**: Some resources (like ConfigMaps and Secrets) are prerequisites for Deployments, so a misconfiguration or race condition could cause failures.
- **High Risk of Unintended Changes**: Reapplying the entire file to fix a single issue would potentially affect resources that were already functioning correctly, causing unnecessary disruptions.

2.1.2 Why We Need to Split the YAML File

The need for splitting the original YAML file arises from several challenges:

- **Simplified Debugging**: By splitting the YAML file, each resource can be deployed or redeployed individually, allowing for easier identification of issues.
- **Targeted Deployment**: In complex environments with multiple interdependent services, resources can be applied selectively. If a failure occurs, only the affected resources need to be redeployed, instead of restarting everything.
- **Clearer Resource Management**: With the separation of resources, each component can be updated and versioned independently, enhancing maintainability and tracking.

2.2 First Approach: Dividing by Resource Types

The initial approach involved dividing the YAML files by resource type, such as:

- 01-namespaces/
- 02-services/
- 03-deployments/
- 04-configmaps/
- 05-secrets/

While this strategy worked in some cases, it had its shortcomings:

- **Scattered Context**: Resources related to a single service (e.g., Grafana) were placed in multiple folders (e.g., ServiceAccount in services/, ConfigMap in configmaps/, etc.). This made it difficult to track and manage all the components related to a single service.

- **Deployment Complexity**: To deploy all resources for a service, it required navigating across multiple folders. This added complexity and risked missing dependencies.

2.3 Second Approach: Dividing by Services

A more logical and efficient approach was adopted by grouping resources based on the service they pertain to. For example:

- 03-opentelemetry-collector/
  - Contains YAML files for deploying the OpenTelemetry collector, such as service account, config map, cluster roles, and deployment.
- 04-jaeger/
  - Contains files for the Jaeger service, including service account, service, and deployment.
- 06-grafana/
  - All Grafana-related files, including service account, secrets, config map, role bindings, deployment, etc.

2.3.1 Advantages of this Approach

This approach allowed the following benefits:

- **Service-Specific Grouping**: All resources related to a specific service are now located in a single folder, making it easier to manage, debug, and understand.

- **Simpler Deployment**: By grouping resources for each service together, it is easier to deploy them as a cohesive unit, reducing the need to jump between multiple folders.

2.3.2 Namespace Management and Deployment Order

A critical aspect of deploying resources in Kubernetes is ensuring the correct order of operations, especially when dependencies exist between resources. Kubernetes validates and applies resources in a specific order:

1. **Namespace and Service Accounts**: These are the first resources to be deployed, as they establish the namespace and required access control.

2. **Core Services**: Resources that provide foundational services (e.g., OpenTelemetry Collector, Jaeger) are deployed next.

3. **Visualization Tools**: Tools like Grafana and Prometheus are deployed after core services, as they rely on the services being operational.

We adopted a numbered pattern and naming convention to organize the YAML files and ensure a clear, logical order for resource deployment. This system helps ensure that Kubernetes resources are applied in the correct order, respecting dependencies between different components.

Below is a more detailed explanation of the naming convention and numbering pattern:

A) Numbered Folder Structure

Each folder in the otel-demo-yaml directory has been numbered sequentially, starting from 01-08, to represent the order in which they should be deployed. This numbering system is not only intuitive but also prevents mistakes when applying the YAML files in a large, interconnected system.

```
otel-demo-yaml/
├── 01-namespaces            (Namespace setup comes first)
├── 02-core                  (Core resources like service accounts)
├── 03-opentelemetry-collector (OpenTelemetry Collector resources)
├── 04-jaeger                (Jaeger deployment)
├── 05-prometheus            (Prometheus setup)
├── 06-grafana               (Grafana resources)
├── 07-opensearch            (OpenSearch deployment)
└── 08-application           (Application-related services and deployments)
```

B) Numbered Files Within Folders

Inside each folder, the YAML files are also numbered sequentially (e.g., 01, 02, 03, etc.). This ensures that resources within each folder are applied in the correct order as well.

Why Number the Files?

- **Handling Dependencies Between Resources**: Some Kubernetes resources depend on others. For example, a service account (e.g., 01-serviceaccount.yaml) must exist before a deployment (e.g., 02-deployment.yaml) that uses that service account can be applied. By numbering the files, we can ensure that Kubernetes applies them in the correct sequence.

- **Logical Flow**: The order of resource application is logical and helps avoid conflicts or race conditions in the cluster. Each resource that is listed first is assumed to be foundational or required by the next resource.

Example in the 03-opentelemetry-collector/ folder:

```
03-opentelemetry-collector/
├── 01-otelcol-serviceaccount.yaml    (Service account is created first)
├── 02-otelcol-configmap.yaml         (ConfigMap for the collector)
├── 03-otelcol-clusterrole.yaml       (ClusterRole for permissions)
├── 04-otelcol-clusterrolebinding.yaml (Binding the role to the service account)
├── 05-otelcol-service.yaml           (OpenTelemetry Collector Service)
└── 06-otelcol-deployment.yaml        (Finally, the deployment of OpenTelemetry Collector)
```

C) Deployment Flow Based on Numbering

When deploying the resources, the numbering pattern ensures that Kubernetes checks the folder and file sequence, and applies them in a structured way:

1. **Namespaces** are created first, followed by any foundational resources like service accounts or roles.

2. Then, **core services** such as OpenTelemetry Collector, Jaeger, and Prometheus are deployed to provide essential telemetry and monitoring.

3. **Visualization and storage services**, such as Grafana and OpenSearch, are deployed after the core monitoring components are set up.

4. Finally, **application-related services** (e.g., frontend, cartservice, paymentservice, etc.) are deployed. These are dependent on the monitoring and telemetry services, so they are deployed after those foundational resources are established.

D) Dependencies Between Resources

The numbering system also helps manage dependencies in Kubernetes resources.
For instance:

- A service account must exist before a deployment using that service account.
- A ConfigMap must be created before it can be referenced by a deployment or pod.
- A role and rolebinding must be applied before any pods can make use of them.

```
03-opentelemetry-collector/
├── 01-otelcol-serviceaccount.yaml    --> Service account needed before any pods use it
├── 02-otelcol-configmap.yaml         --> ConfigMap for application configuration
├── 03-otelcol-clusterrole.yaml       --> Role permissions
├── 04-otelcol-clusterrolebinding.yaml --> Binding the role to the service account
├── 05-otelcol-service.yaml           --> Service definition for communication
└── 06-otelcol-deployment.yaml        --> Deployment referencing above resources
```

This numbering pattern creates a natural progression for the resources, ensuring that all dependencies are met before the next step in the deployment process.

The split approach resulted in:

- **28 Directories**
- **67 Files** in total, each representing a discrete Kubernetes resource.

The final structure grouped the resources in a logical, hierarchical way. For example, all files related to OpenTelemetry Collector were grouped under 03-opentelemetry-collector, and all services for individual application components (e.g., frontend, backend, payment service) were grouped under 08-application, each with their own subfolder.

E) Explanation of the file structure and explaining the purpose of each resource:

The original single YAML file opentelemetry-demo.yaml was broken down into several smaller files, each representing a distinct Kubernetes resource. These files were then organized into folders based on the type of resource they define.

Folder Structure and Total Files

- **Root Directory**: otel-demo-yaml/
    - **Total Number of Files**: 67
    - **Total Number of Directories**: 28

Folder Breakdown:

1. **01-namespaces**
    - **Files**:
        - 01-otel-demo-namespace.yaml
    - **Purpose**: Defines the otel-demo namespace.

2. **02-core**
    - **Files**:
        - serviceaccount.yaml
    - **Purpose**: Defines core resources like service accounts and base permissions.

3. **03-opentelemetry-collector**
    - **Files**:
        - 01-otelcol-serviceaccount.yaml
        - 02-otelcol-configmap.yaml
        - 03-otelcol-clusterrole.yaml
        - 04-otelcol-clusterrolebinding.yaml
        - 05-otelcol-service.yaml
        - 06-otelcol-deployment.yaml
    - **Purpose**: Resources related to the OpenTelemetry Collector service.

4. **04-jaeger**
    - **Files**:
        - 01-jaeger-serviceaccount.yaml
        - 02-jaeger-service.yaml
        - 03-jaeger-deployment.yaml
    - **Purpose**: Resources related to Jaeger for distributed tracing.

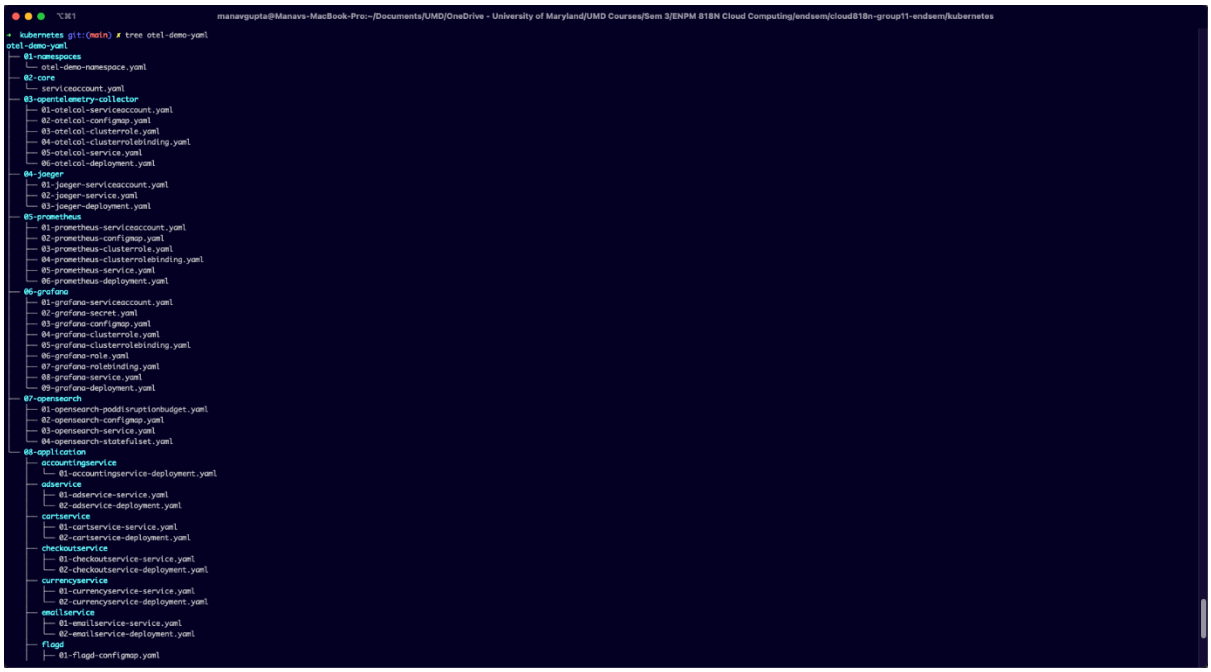
5. **05-prometheus**
    - **Files**:
        - 01-prometheus-serviceaccount.yaml
        - 02-prometheus-configmap.yaml
        - 03-prometheus-clusterrole.yaml
        - 04-prometheus-clusterrolebinding.yaml
        - 05-prometheus-service.yaml
        - 06-prometheus-deployment.yaml
    - **Purpose**: Resources related to Prometheus for monitoring.

6. **06-grafana**
    - **Files**:
        - 01-grafana-serviceaccount.yaml
        - 02-grafana-secret.yaml
        - 03-grafana-configmap.yaml
        - 04-grafana-clusterrole.yaml\
        - 05-grafana-clusterrolebinding.yaml
        - 06-grafana-role.yaml
        - 07-grafana-rolebinding.yaml
        - 08-grafana-service.yaml
        - 09-grafana-deployment.yaml
    - **Purpose**: Resources related to Grafana for visualization and monitoring.

7. **07-opensearch**
    - **Files**:
        - 01-opensearch-poddisruptionbudget.yaml
        - 02-opensearch-configmap.yaml
        - 03-opensearch-service.yaml
        - 04-opensearch-statefulset.yaml
    - **Purpose**: Resources related to OpenSearch for logs and data storage.

8. **08-application**
    - **Files**: A collection of files for various application services such as accountingservice, adservice, cartservice, etc. These include service and deployment files for each microservice.
    - **Purpose**: Each subfolder contains the deployment and service definitions for a specific application service.

Screenshots of the created folder structure containing the split YAML –



2.3.3 Applying Resources Individually or Recursively

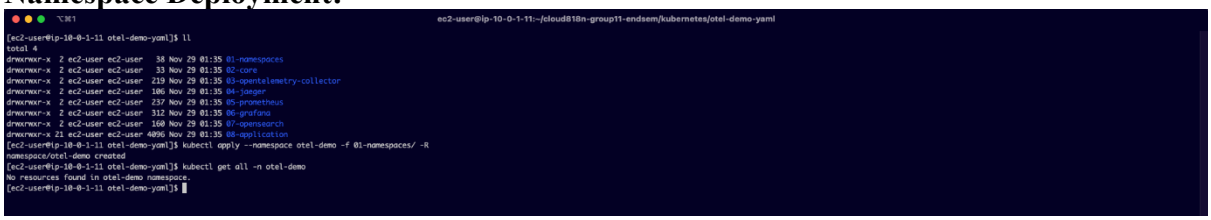
The goal here is to apply the split YAML files, either individually or recursively, to deploy the resources in a controlled manner, ensuring that each resource is deployed correctly before moving on to the next. This method also ensures that if a service encounters an issue, it can be troubleshooted and redeployed without affecting other parts of the system.

2.3.3.1 Individually Applying Resources

We began the deployment process by applying the namespace resources using --namespace flag to ensure that all subsequent resources were applied to the correct namespace (otel-demo).

Command Used:
kubectl apply--namespace otel-demo -f 01-namespaces/otel-demo-namespace.yaml

Namespace Deployment:



After ensuring that the namespace was set up, we moved on to apply core resources which has service accounts that are foundational and must be applied before any application-specific components like services and deployments.

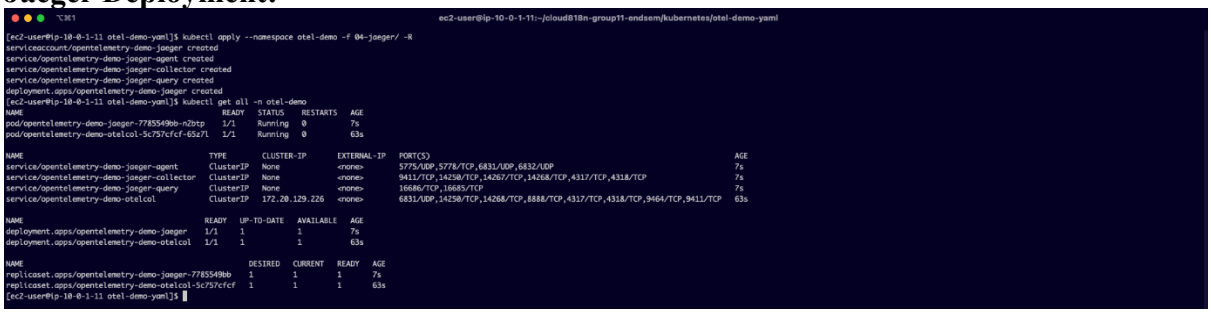
Command Used:
kubectl apply--namespace otel-demo -f 02-core/ -R

Core Deployment:

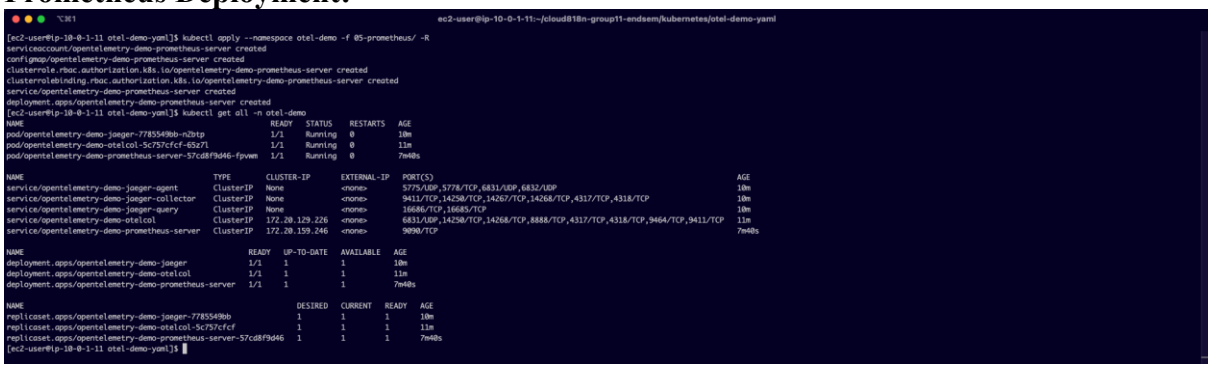
We then proceeded to apply resources for each service. Each service (e.g., OpenTelemetry Collector, Prometheus, Grafana, etc.) was organized into its own folder. These folders contained all relevant resources such as service accounts, secrets, config maps, roles, services, and deployments. Finally, we applied resources for the individual application services (e.g., frontend, paymentservice, adservice, etc.). Each of these services had its own folder and contained all related Kubernetes resources (services, deployments, etc.).

Otecol Deployment:

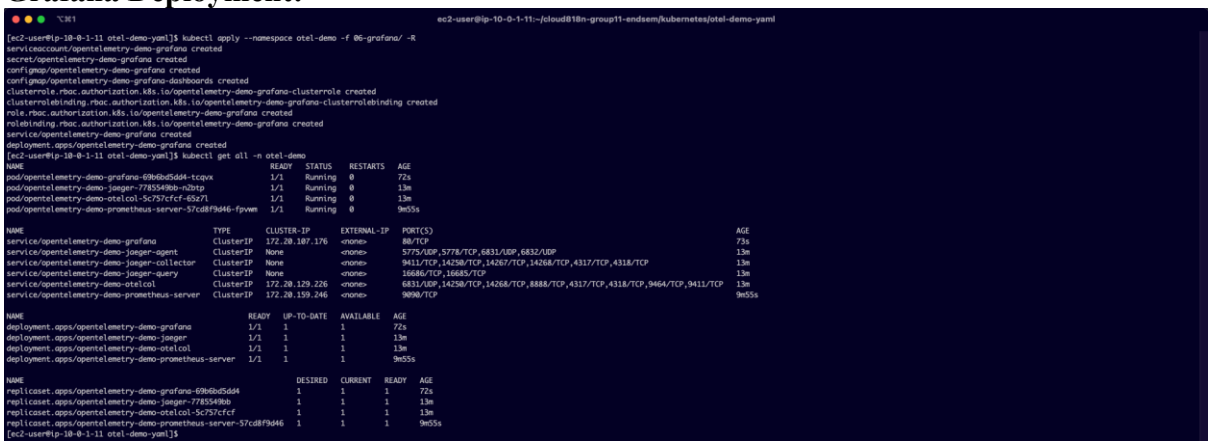
Jaeger Deployment:



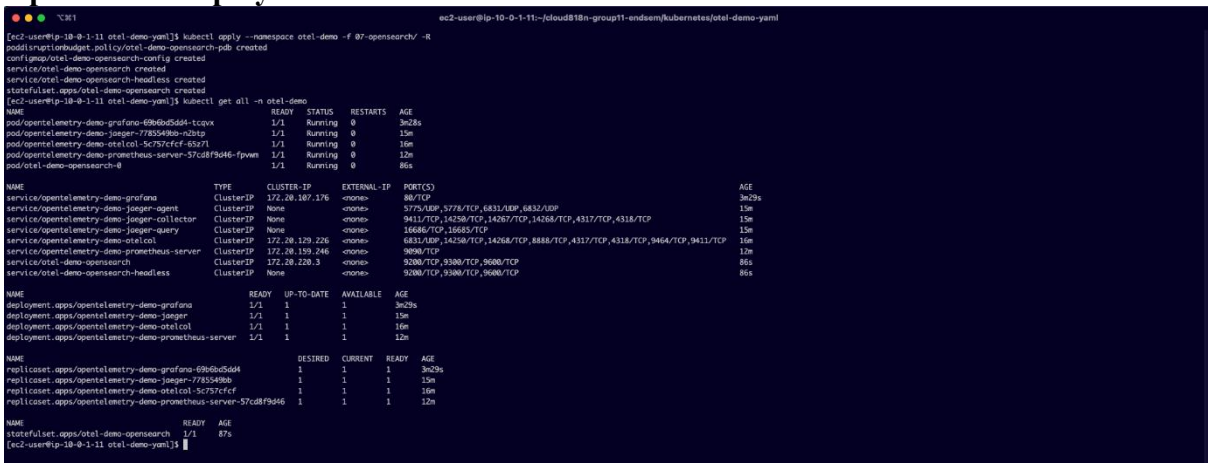
Prometheus Deployment:



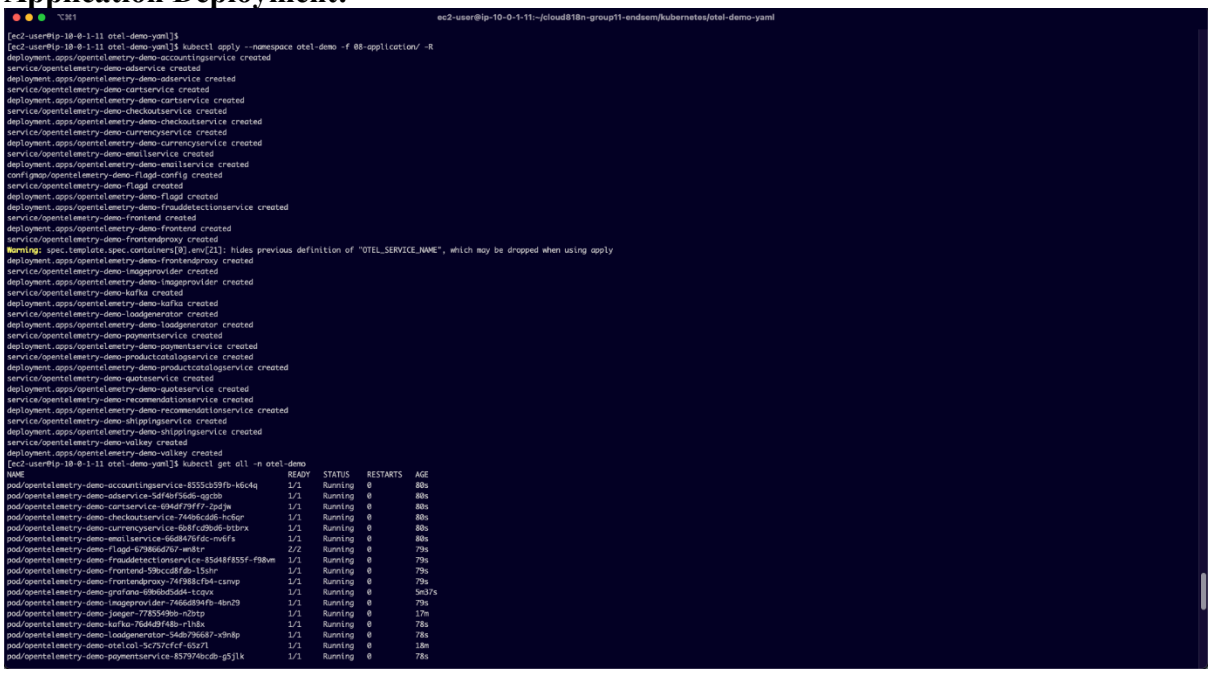
Grafana Deployment:



Opensearch Deployment:



Application Deployment:

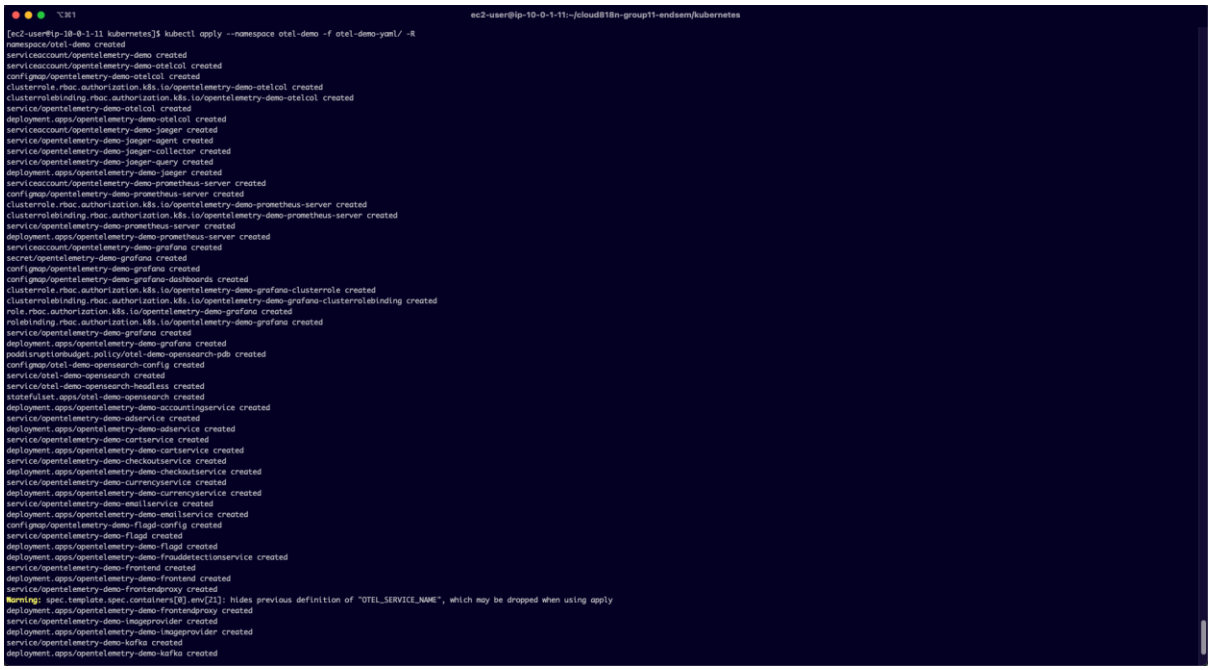


2.3.3.2 Recursive Application of the Entire Structure

After confirming that all services had been applied successfully, we applied the entire project recursively, starting from the root folder of the project to deploy all the services in the correct order.

Command Used:

kubectl apply--namespace otel-demo -f otel-demo-yaml/ -R

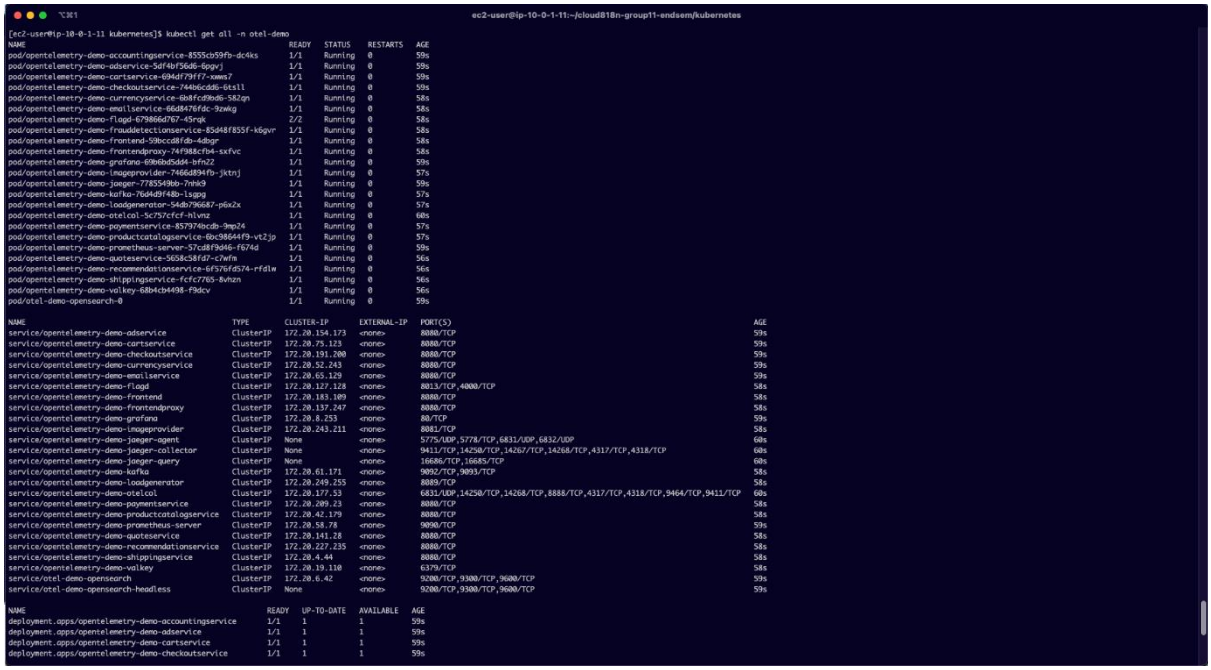


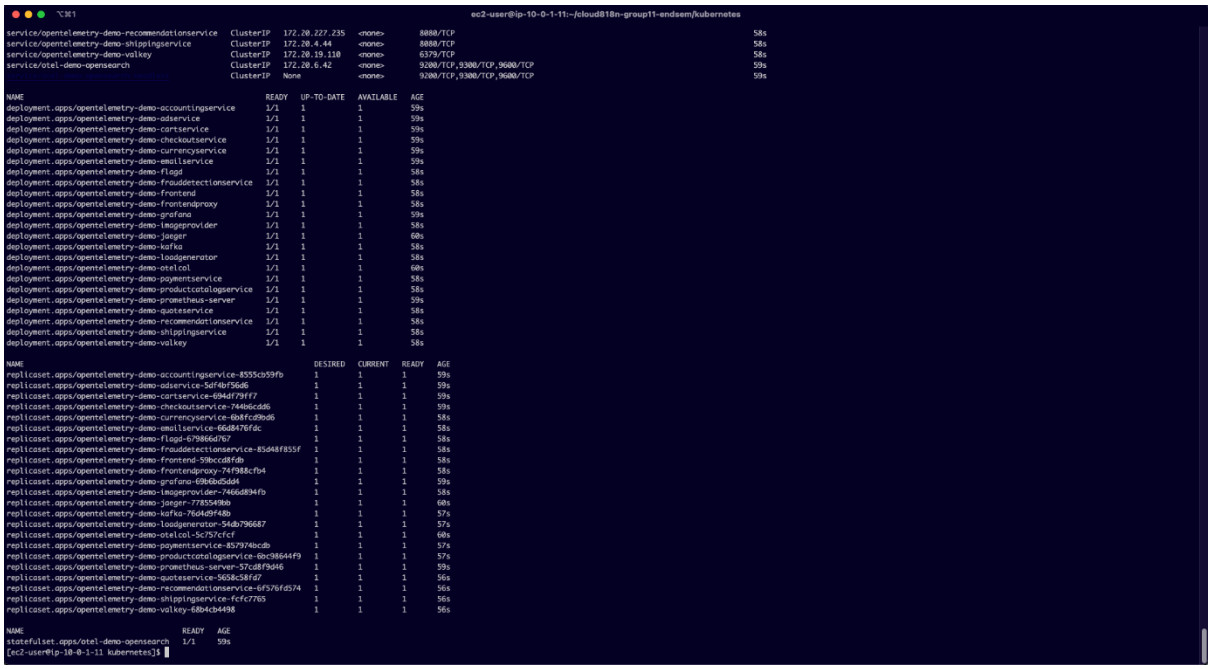
2.3.4 Validate Resource Deployment

2.3.4.1 Checking Pod and Service Status

After applying the resources, the next step was to validate the deployment by checking the status of the pods, services, and other resources. This ensures that everything was deployed correctly and is functioning as expected.

We used the `kubectl get all -n otel-demo` command to check the status of all resources within the otel-demo namespace. This command lists all the pods, services, deployments, and other resources, making it easy to confirm that they were all running correctly.



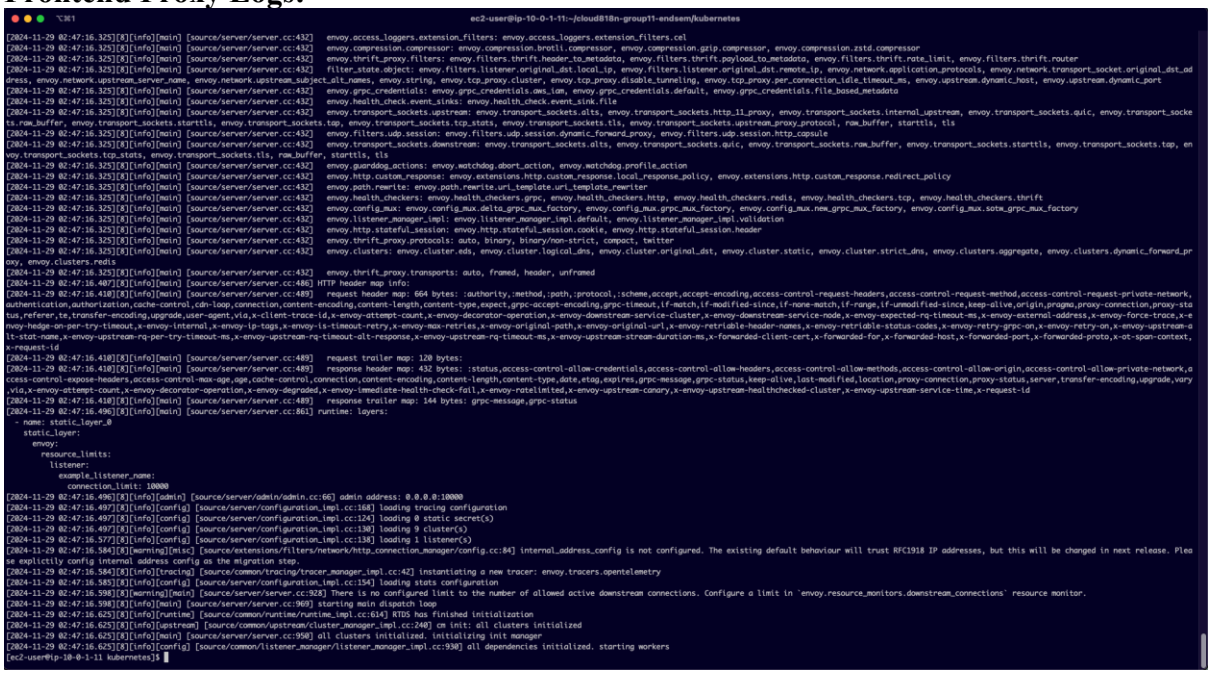


2.3.4.2 Reviewing Pod Logs

If any pods were not running or showed errors, we reviewed the logs for those specific pods using the command kubectl logs <pod-name>-n otel-demo. We checked the logs to identify any issues related to service misconfigurations, missing dependencies, or Kubernetes errors. We also checked the running application and took some screenshots.

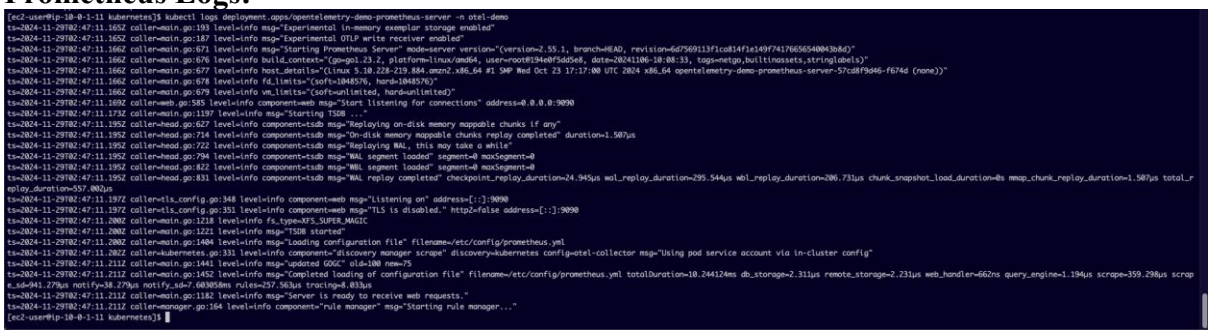
Frontend Logs:

Frontend Proxy Logs:



Jaeger Logs:

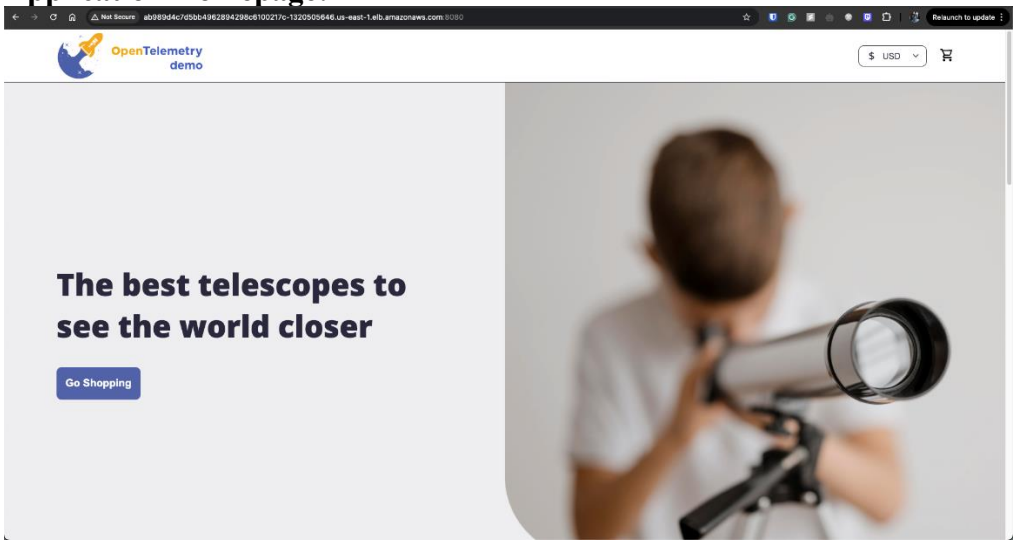
Prometheus Logs:



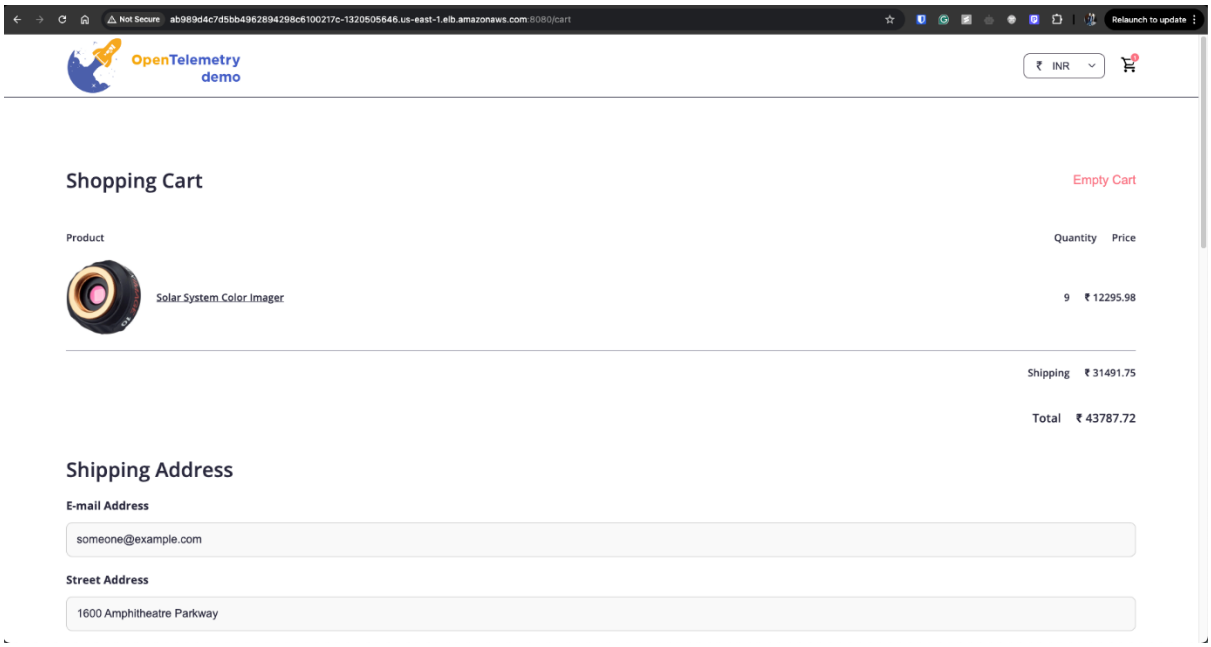
Grafana Logs:

ProductCatalogService Logs:

Application Homepage:



Cart Service View:



Checkout Service: