Deploying and Managing Microservices in a Cloud-Native Environment

Objective:

This project aims to migrate a monolithic application to a Kubernetes-based microservices architecture to achieve better scalability, high availability, and employment flexibility. As a DevOps Engineer at ABVC Solutions, the focus is on setting up Kubernetes infrastructure, containerizing services, and implementing cloud-native deployment practices.

Infrastructure and Tools Used:

- Kubernetes Cluster: Provisioned using EKS (Elastic Kubernetes Service)
- Containerization: Docker (with images stored on Docker Hub)
- CI/CD & CLI Tools: kubectl, docker, hey (load generation), curl
- Monitoring/Scaling: Horizontal Pod Autoscaler (HPA)
- Storage: Kubernetes Persistent Volumes (PV) and Persistent Volume Claims (PVC)

Implementation Steps:

1. Kubernetes Cluster Setup

- Created Kubernetes cluster using EKS on AWS
- Installed and configured kubectl to interact with the cluster:
  aws eks --region <region> update-kubeconfig --name <cluster-name>
  kubectl get nodes

2. Containerization & Docker

- Built a sample microservices application with three services:
  1. User Service
  2. Product Service
  3. Order Service
- Created separate Dockerfiles for each service
- Built and pushed images to Docker Hub:
  docker build -t <username>/user-service:latest .
  docker push <username>/user-service:latest

3. Kubernetes Deployments

- Created Deployment YAML files for each service with:

1. Container image
2. Replicas
3. Ports
4. Env variables

- Deployed with:

  kubectl apply -f deployments/

4. Service Discovery & Load Balancing

- Defined Kubernetes Service resources:
  1. Used ClusterIP for internal microservice communication
  2. Used NodePort for external access during development
- Verified communication using internal DNS and curl requests

5. Scaling with HPA

- Enabled autoscaling for Product Service using HPA:

  apiVersion: autoscaling/v2
  kind: HorizontalPodAutoscaler
  ...
- Simulated load with:

  hey -z 1m -c 50 http://<node-ip>:<port>/products
- Observed scaling activity with:

  kubectl get hpa

6. Persistent Storage

- Created Persistent Volume (PV) and Persistent Volume Claim (PVC)
- Configured Order Service to store and read data from mounted volume
- Verified that data persisted even after pod restarts

Key Points :

1. Cluster Setup

- Used EKS for managed Kubernetes
- Configured access with AWS CLI and kubectl

2. Deployment & Testing

- Dockerized and deployed 3 services
- Set up internal/external service exposur
- Verified inter-service communication

3. Tools & Technologies

- AWS EKS, Docker, Kubernetes, Helm (if used), HPA, PV/PVC, kubectl, curl, hey

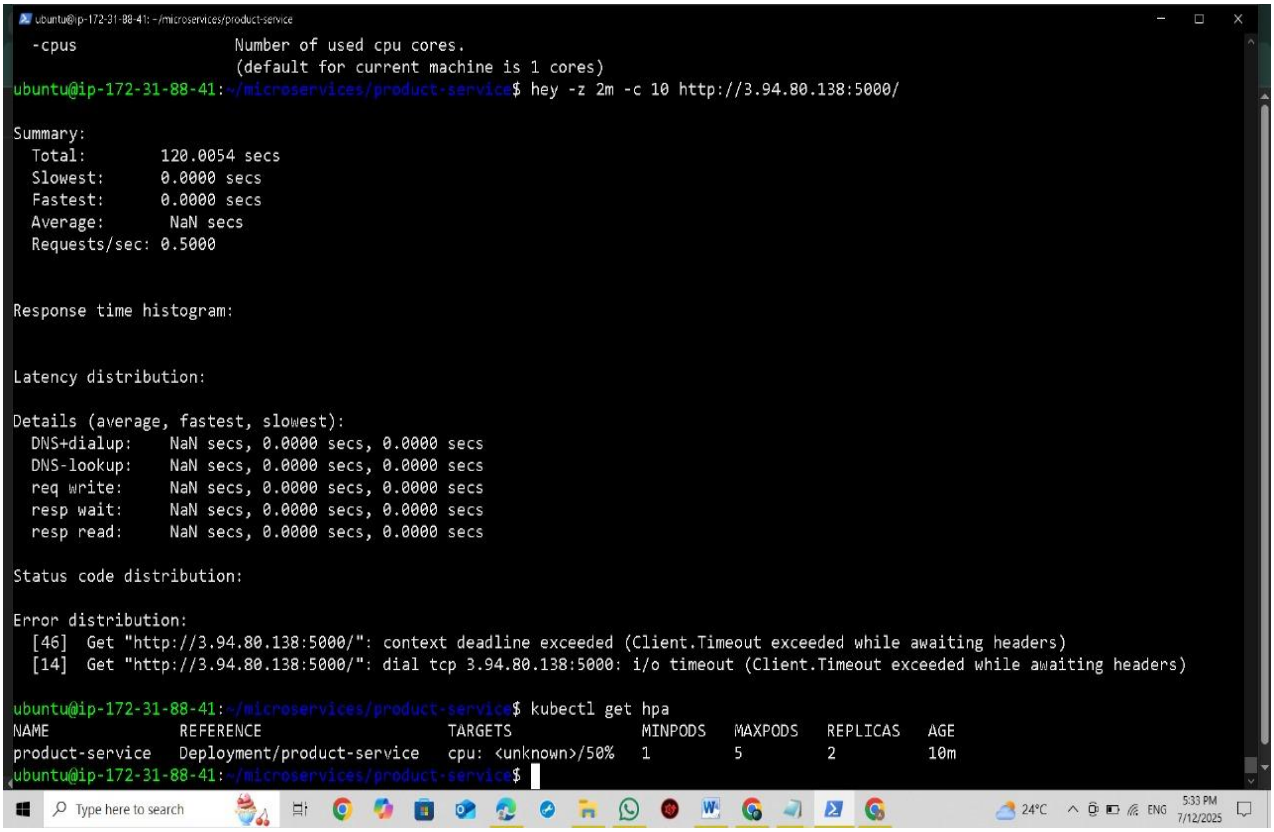
4. Challenges Faced

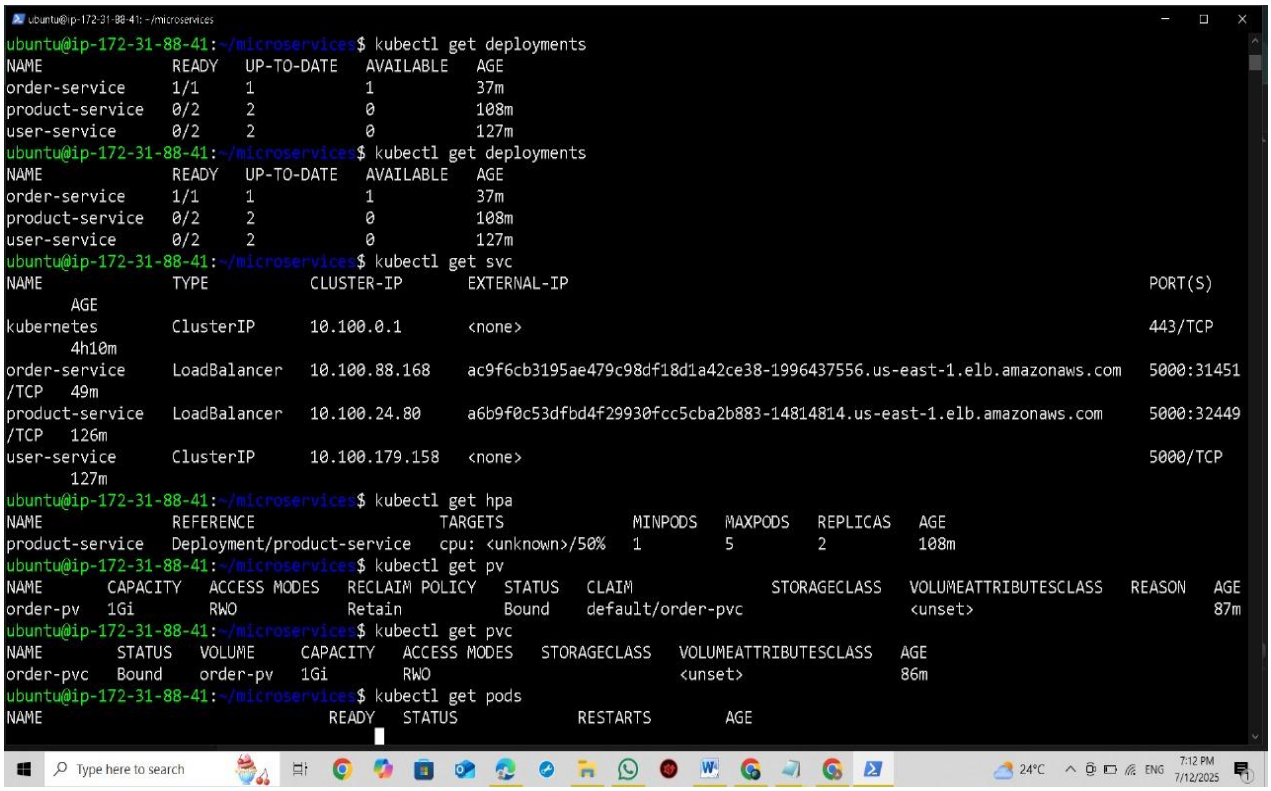
Challenge	Solution Implemented
Docker image size optimization	Used multi-stage builds in Dockerfiles
Inter-service DNS issues	Ensured proper service names and namespace usage
Autoscaler not triggering	Increased CPU load using stress/load tools
Data loss after restart	Implemented PVC and volume mounts correctly in yaml

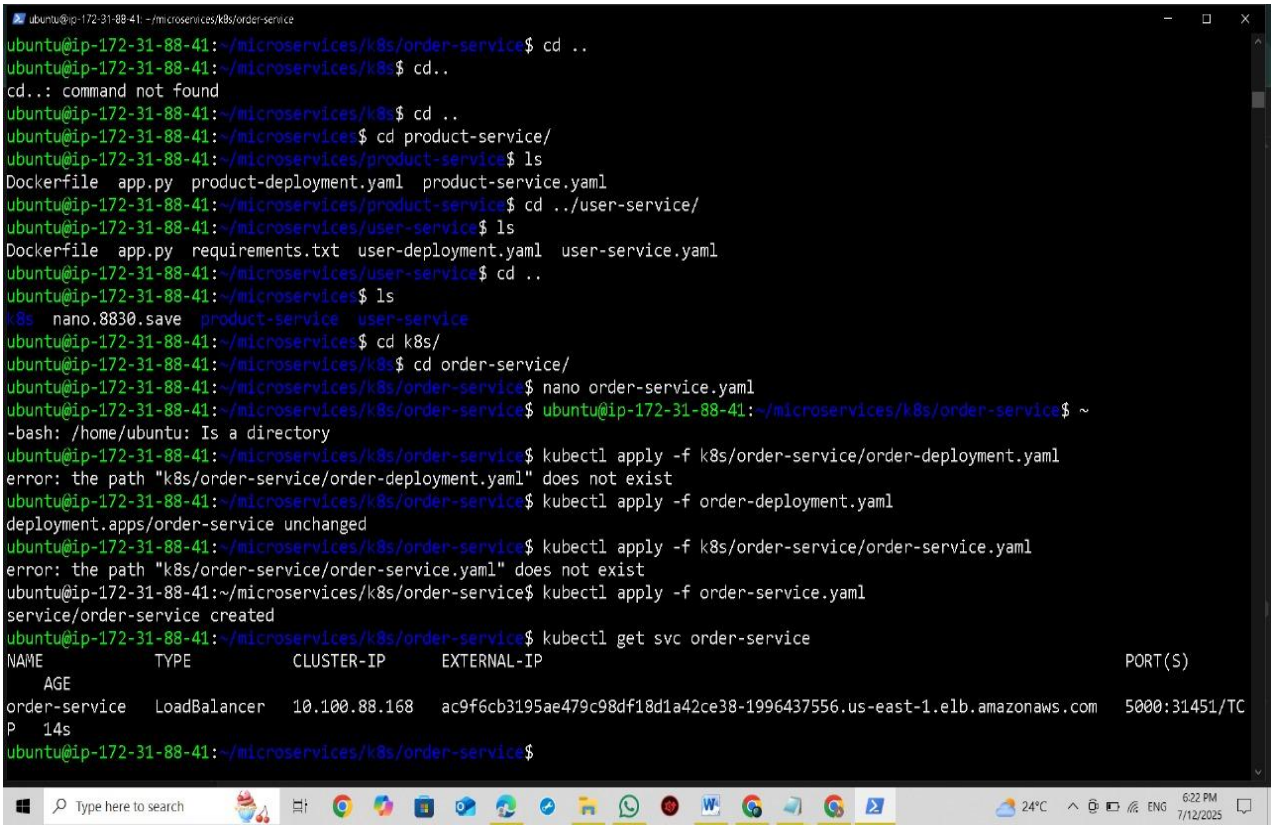
Outcome:

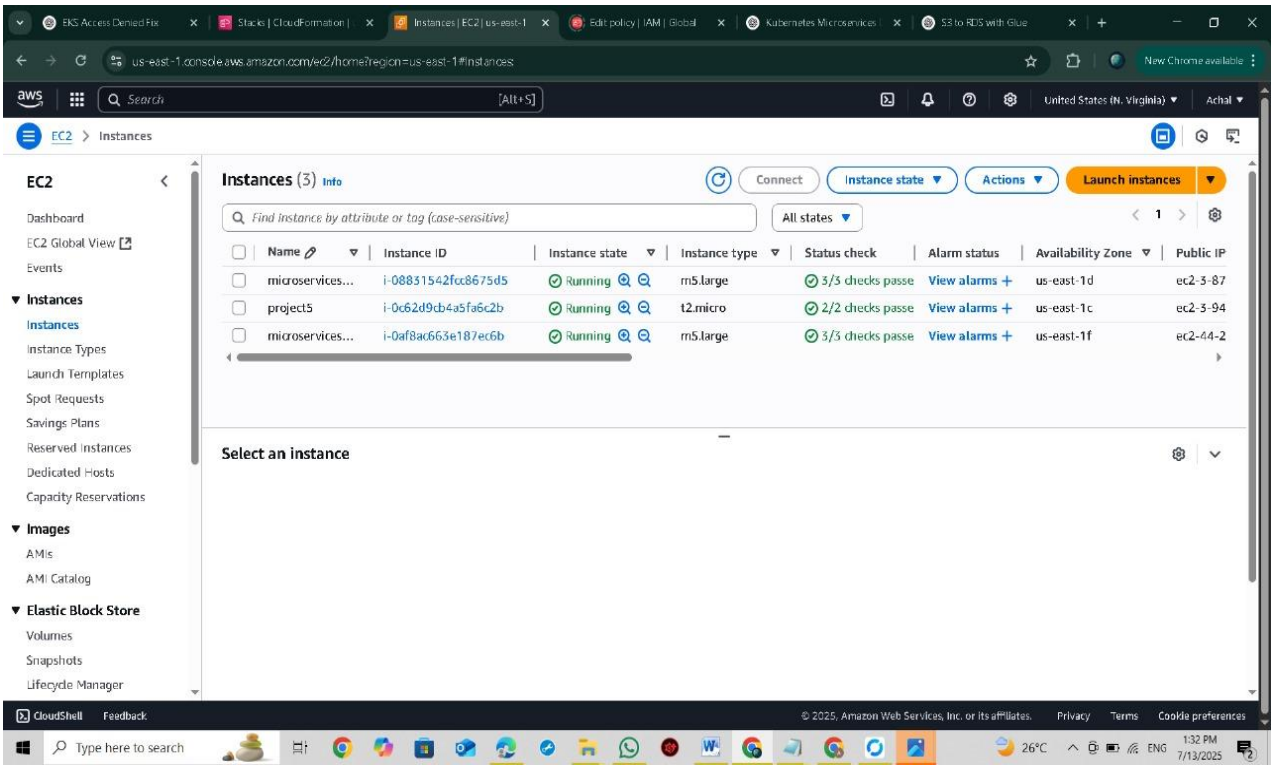
The project successfully demonstrated:

- End-to-end deployment of containerized microservices
- Kubernetes concepts such as services, scaling, and persistent storage
- Real-world microservice orchestration and fault tolerance


ubuntu@ip-172-31-88-41: ~/microservices/product-service
-cpus Number of used cpu cores.
(default for current machine is 1 cores)
ubuntu@ip-172-31-88-41:~/microservices/product-service$ hey -z 2m -c 10 http://3.94.80.138:5000/
Summary:
Total: 120.0054 secs
Slowest: 0.0000 secs
Fastest: 0.0000 secs
Average: NaN secs
Requests/sec: 0.5000
Response time histogram:
Latency distribution:
Details (average, fastest, slowest):
DNS+dialup: NaN secs, 0.0000 secs, 0.0000 secs
DNS-lookup: NaN secs, 0.0000 secs, 0.0000 secs
req write: NaN secs, 0.0000 secs, 0.0000 secs
resp wait: NaN secs, 0.0000 secs, 0.0000 secs
resp read: NaN secs, 0.0000 secs, 0.0000 secs
Status code distribution:
Error distribution:
[46] Get "http://3.94.80.138:5000/": context deadline exceeded (Client.Timeout exceeded while awaiting headers)
[14] Get "http://3.94.80.138:5000/": dial tcp 3.94.80.138:5000: i/o timeout (Client.Timeout exceeded while awaiting headers)
ubuntu@ip-172-31-88-41:~/microservices/product-service$ kubectl get hpa
NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS AGE
product-service Deployment/product-service cpu: <unknown>/50% 1 5 2 10m
ubuntu@ip-172-31-88-41:~/microservices/product-service$
Type here to search
24°C
ENG
5:33 PM
7/12/2025


ubuntu@ip-172-31-88-41: ~/microservices
ubuntu@ip-172-31-88-41:~/microservices$ kubectl get deployments
NAME READY UP-TO-DATE AVAILABLE AGE
order-service 1/1 1 1 37m
product-service 0/2 2 0 108m
user-service 0/2 2 0 127m
ubuntu@ip-172-31-88-41:~/microservices$ kubectl get deployments
NAME READY UP-TO-DATE AVAILABLE AGE
order-service 1/1 1 1 37m
product-service 0/2 2 0 108m
user-service 0/2 2 0 127m
ubuntu@ip-172-31-88-41:~/microservices$ kubectl get svc
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S)
AGE
kubernetes ClusterIP 10.100.0.1 <none> 443/TCP
4h10m
order-service LoadBalancer 10.100.88.168 ac9f6cb3195ae479c98df18d1a42ce38-1996437556.us-east-1.elb.amazonaws.com 5000:31451
/TCP 49m
product-service LoadBalancer 10.100.24.80 a6b9f0c53dfbd4f29930fcc5cba2b883-14814814.us-east-1.elb.amazonaws.com 5000:32449
/TCP 126m
user-service ClusterIP 10.100.179.158 <none> 5000/TCP
127m
ubuntu@ip-172-31-88-41:~/microservices$ kubectl get hpa
NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS AGE
product-service Deployment/product-service cpu: <unknown>/50% 1 5 2 108m
ubuntu@ip-172-31-88-41:~/microservices$ kubectl get pv
NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS VOLUMEATTRIBUTESCLASS REASON AGE
order-pv 1Gi RWO Retain Bound default/order-pvc <unset> 87m
ubuntu@ip-172-31-88-41:~/microservices$ kubectl get pvc
NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS VOLUMEATTRIBUTESCLASS AGE
order-pvc Bound order-pv 1Gi RWO <unset> 86m
ubuntu@ip-172-31-88-41:~/microservices$ kubectl get pods
NAME READY STATUS RESTARTS AGE
Type here to search
24°C
ENG
7:12 PM
7/12/2025


ubuntu@ip-172-31-88-41: ~/microservices/k8s/order-service
ubuntu@ip-172-31-88-41:~/microservices/k8s/order-service$ cd ..
ubuntu@ip-172-31-88-41:~/microservices/k8s$ cd..
cd..: command not found
ubuntu@ip-172-31-88-41:~/microservices/k8s$ cd ..
ubuntu@ip-172-31-88-41:~/microservices$ cd product-service/
ubuntu@ip-172-31-88-41:~/microservices/product-service$ ls
Dockerfile app.py product-deployment.yaml product-service.yaml
ubuntu@ip-172-31-88-41:~/microservices/product-service$ cd ../user-service/
ubuntu@ip-172-31-88-41:~/microservices/user-service$ ls
Dockerfile app.py requirements.txt user-deployment.yaml user-service.yaml
ubuntu@ip-172-31-88-41:~/microservices/user-service$ cd ..
ubuntu@ip-172-31-88-41:~/microservices$ ls
k8s nano.8830.save product-service user-service
ubuntu@ip-172-31-88-41:~/microservices$ cd k8s/
ubuntu@ip-172-31-88-41:~/microservices/k8s$ cd order-service/
ubuntu@ip-172-31-88-41:~/microservices/k8s/order-service$ nano order-service.yaml
ubuntu@ip-172-31-88-41:~/microservices/k8s/order-service$ ubuntu@ip-172-31-88-41:~/microservices/k8s/order-service$ ~
-bash: /home/ubuntu: Is a directory
ubuntu@ip-172-31-88-41:~/microservices/k8s/order-service$ kubectl apply -f k8s/order-service/order-deployment.yaml
error: the path "k8s/order-service/order-deployment.yaml" does not exist
ubuntu@ip-172-31-88-41:~/microservices/k8s/order-service$ kubectl apply -f order-deployment.yaml
deployment.apps/order-service unchanged
ubuntu@ip-172-31-88-41:~/microservices/k8s/order-service$ kubectl apply -f k8s/order-service/order-service.yaml
error: the path "k8s/order-service/order-service.yaml" does not exist
ubuntu@ip-172-31-88-41:~/microservices/k8s/order-service$ kubectl apply -f order-service.yaml
service/order-service created
ubuntu@ip-172-31-88-41:~/microservices/k8s/order-service$ kubectl get svc order-service
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S)
AGE
order-service LoadBalancer 10.100.88.168 ac9f6cb3195ae479c98df18d1a42ce38-1996437556.us-east-1.elb.amazonaws.com 5000:31451/TC
P 14s
ubuntu@ip-172-31-88-41:~/microservices/k8s/order-service$
Type here to search
24°C
ENG
6:22 PM
7/12/2025


us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:
aws
Search
[Alt+S]
United States (N. Virginia)
Achal
EC2 > Instances
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Instances (3) Info
Connect
Instance state
Actions
Launch instances
Find instance by attribute or tag (case-sensitive)
All states
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IP
microservices...
i-08831542fcc8675d5
Running
m5.large
3/3 checks passe
View alarms +
us-east-1d
ec2-3-87
project5
i-0c62d9cb4a5fa6c2b
Running
t2.micro
2/2 checks passe
View alarms +
us-east-1c
ec2-3-94
microservices...
i-0af8ac663e187ec6b
Running
m5.large
3/3 checks passe
View alarms +
us-east-1f
ec2-44-2
Select an instance
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
26°C
ENG
1:32 PM
7/13/2025