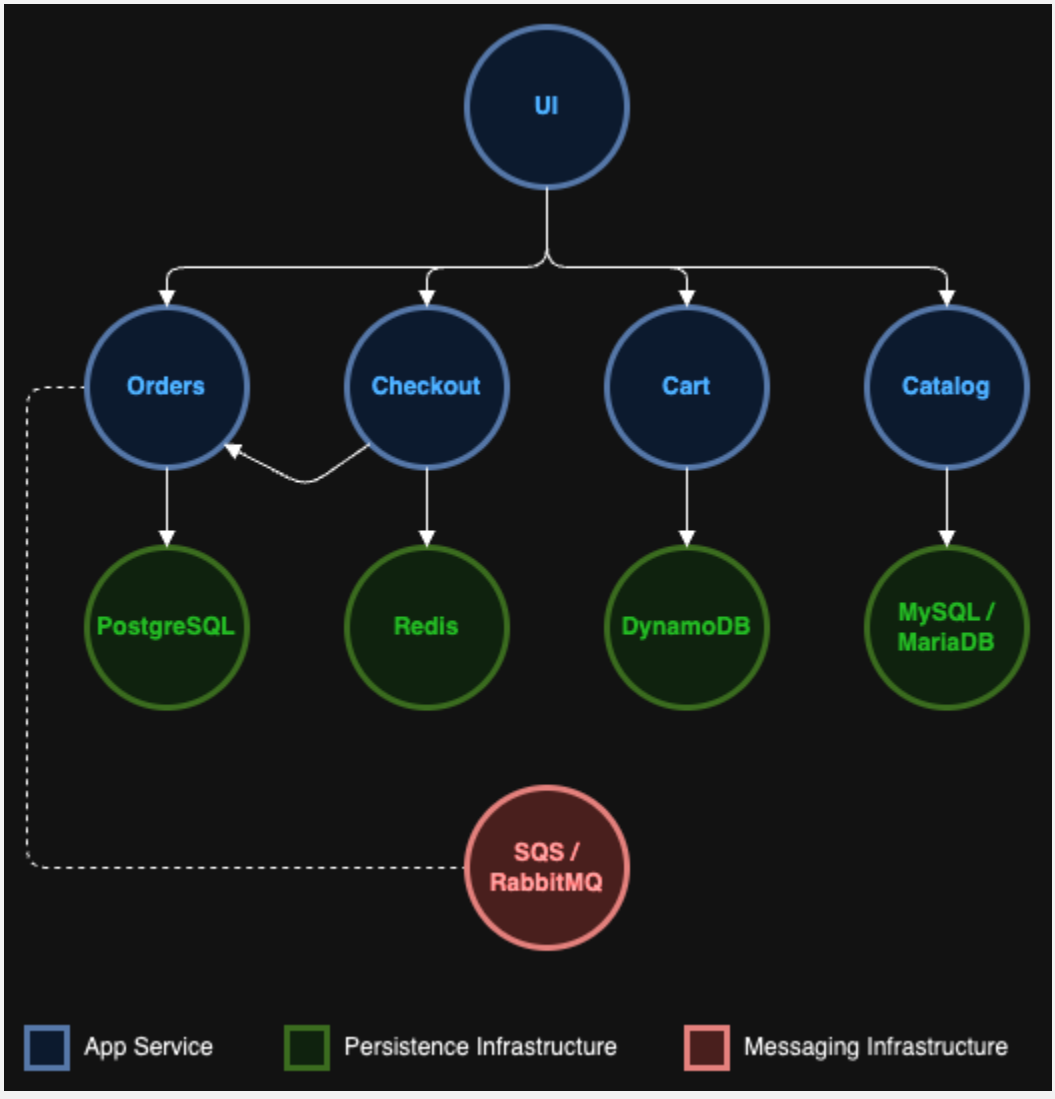
Deploying Retail Store Sample App on Kubernetes

Objective

The AWS Retail Store Sample App is a microservices-based cloud-native application representing an online retail store. It includes multiple services such as UI, Catalog, Cart, Orders, etc.

It explains each step in detail from setup to accessing the app to help learners and professionals gain hands-on experience with kubernetes deployment and service management.



Prerequisites

Before deploying, ensure:

- Kubernetes cluster is running (cloud or local)
- Kubectl command-line tool is configured to access the cluster

Deployment Steps Followed

Step1: Check kubectl version

kubectl version --client

```
controlplane:~$ kubectl version --client
Client Version: v1.34.1
Kustomize Version: v5.7.1
controlplane:~$
controlplane:~$
```

This confirms that kubectl (Kubernetes command-line-tool) is installed correctly on the system. It shows the client version being used to interact with the Kubernetes cluster.

Step 2: Verify Kubernetes cluster connection

kubectl cluster-info

```
controlplane:~$ kubectl cluster-info
Kubernetes control plane is running at https://172.30.1.2:6443
CoreDNS is running at https://172.30.1.2:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
controlplane:~$
```

This checks if the kubernetes cluster is properly configured and reachable. If successful, it displays details of the kubernetes master and DNS services. This ensures kubectl is connected to the correct cluster before deployment.

Step 3: Deploy the Retail Store application

Applies the official manifest file which contains all Kubernetes objects required for the Retail app, including:

- Deployments (UI, catalog, cart, orders…)
- Services (ClusterIP / LoadBalancer)
- Configuration for communication between microservices

This automatically pulls the required images from public container repositories.

https://github.com/obomble67/Deploying-Retail-Store-Sample-App-on-Kubernetes/blob/main/kubernetes.yaml

```
controlplane:~$ kubectl apply -f https://github.com/aws-containers/retail-store-sample-app/releases/latest/download/kubernetes.yaml
kubectl wait --for=condition=available deployments --all
serviceaccount/catalog created
secret/catalog-db created
configmap/catalog created
service/catalog-mysql created
service/catalog created
deployment.apps/catalog created
statefulset.apps/catalog-mysql created
serviceaccount/carts created
configmap/carts created
service/carts-dynamodb created
service/carts created
deployment.apps/carts created
deployment.apps/carts-dynamodb created
serviceaccount/orders created
secret/orders-rabbitmq created
secret/orders-db created
configmap/orders created
service/orders-postgresql created
service/orders-rabbitmq created
service/orders created
deployment.apps/orders created
statefulset.apps/orders-postgresql created
statefulset.apps/orders-rabbitmq created
serviceaccount/checkout created
configmap/checkout created
```

Step 4: Check Pods

kubectl get pods

```
controlplane:~$ kubectl get pods
NAME                              READY   STATUS              RESTARTS     AGE
carts-5d4d59f667-j9hfg            0/1     Running             0            41s
carts-dynamodb-7d79bdf955-n4qct   1/1     Running             0            41s
catalog-5f44667799-t6xtr          0/1     CrashLoopBackOff    2 (7s ago)   42s
catalog-mysql-0                   0/1     ContainerCreating   0            42s
checkout-5c78b886b4-knl5t         0/1     ContainerCreating   0            39s
checkout-redis-5ffd844f7-vfpwn    0/1     ContainerCreating   0            39s
orders-6549fc9484-r9ll2           0/1     Running             0            41s
orders-postgresql-0               0/1     ContainerCreating   0            41s
orders-rabbitmq-0                 0/1     ContainerCreating   0            40s
ui-7d45fc58bf-j6mjw               0/1     ContainerCreating   0            38s
controlplane:~$
```

This shows the status of each microservice container during startup Initially, they appear as:

ContainerCreating - Image are being downloaded

CreashLoopBackOff - may require dependency readiness

This helps confirm the deployment process has started correctly.

Check Pods again

After a short wait, all pods should reach the Running status, meaning all services are healthy and ready to accept traffic.

```
controlplane:~$ kubectl get pods
NAME                              READY   STATUS    RESTARTS      AGE
carts-5d4d59f667-j9hfg            1/1     Running   0             3m21s
carts-dynamodb-7d79bdf955-n4qct   1/1     Running   0             3m21s
catalog-5f44667799-t6xtr          1/1     Running   5 (93s ago)   3m22s
catalog-mysql-0                   1/1     Running   0             3m22s
checkout-5c78b886b4-knl5t         1/1     Running   0             3m19s
checkout-redis-5ffd844f7-vfpwn    1/1     Running   0             3m19s
orders-6549fc9484-r9ll2           0/1     Running   1 (51s ago)   3m21s
orders-postgresql-0               1/1     Running   0             3m21s
orders-rabbitmq-0                 1/1     Running   0             3m20s
ui-7d45fc58bf-j6mjw               0/1     Running   0             3m18s
controlplane:~$
```

Step 5: Check Kubernetes Services

kubectl get svc

```
controlplane:~$ kubectl get svc
NAME                TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)              AGE
carts               ClusterIP      10.109.152.226   <none>        80/TCP               4m10s
carts-dynamodb      ClusterIP      10.100.116.151   <none>        8000/TCP             4m10s
catalog             ClusterIP      10.103.198.100   <none>        80/TCP               4m10s
catalog-mysql       ClusterIP      10.96.218.146    <none>        3306/TCP             4m10s
checkout            ClusterIP      10.104.62.38     <none>        80/TCP               4m8s
checkout-redis      ClusterIP      10.103.18.240    <none>        6379/TCP             4m8s
kubernetes          ClusterIP      10.96.0.1        <none>        443/TCP              16d
orders              ClusterIP      10.99.167.162    <none>        80/TCP               4m9s
orders-postgresql   ClusterIP      10.105.109.198   <none>        5432/TCP             4m9s
orders-rabbitmq     ClusterIP      10.107.10.152    <none>        5672/TCP,15672/TCP   4m9s
ui                  LoadBalancer   10.100.106.136   <pending>     80:31699/TCP         4m7s
controlplane:~$
```

Lists all services created by the deployment.

This includes the ui service, which is exposed through a LoadBalancer or NodePort depending on the environment.

Services allow internal and external communication between microservices and the outside world.

Step 6: View UI Service Details

kubectl get svc ui

```
controlplane:~$ kubectl get svc ui
NAME   TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
ui     LoadBalancer   10.100.106.136   <pending>     80:31699/TCP   4m57s
controlplane:~$
controlplane:~$
```

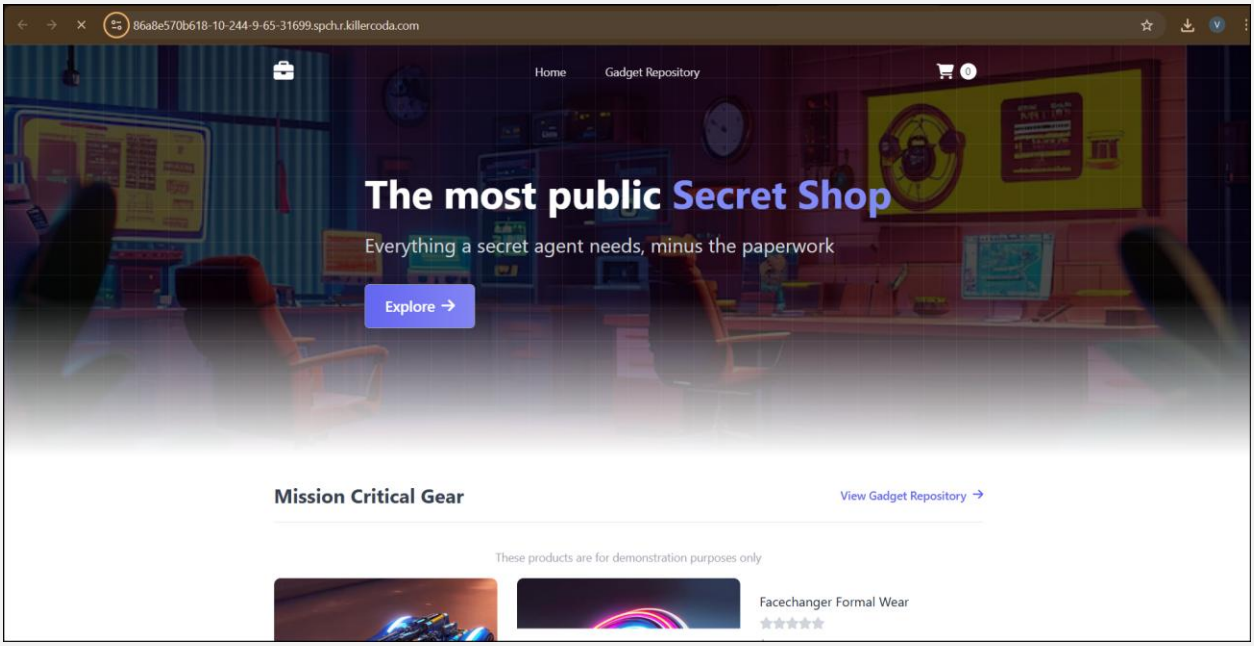
This specifically shows the UI service, including:

TYPE	How It's exposed (LoadBalancer / NodePort)
PORT(S)	Port of the application (usually 8080)
EXTERNAL-IP	Public IP for user access

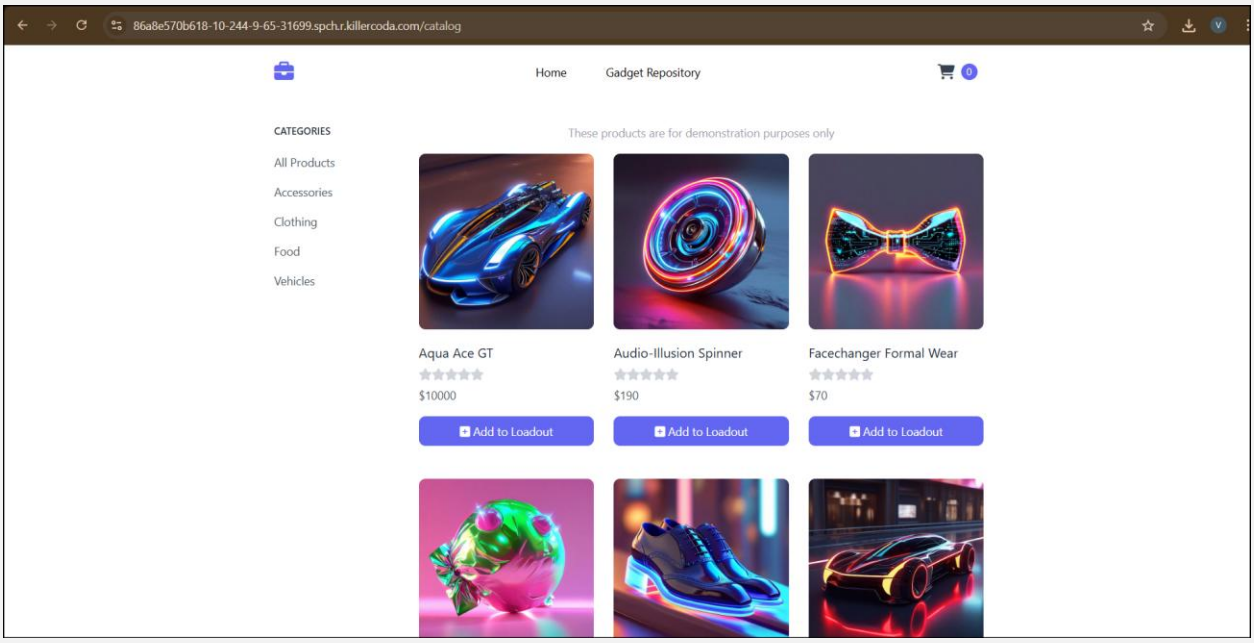
Step 7: Access the Home Page of the Application

Using the LoadBalancer port

http://<ip-of-node>:31699



View Gadget Repository



Conclusion:

- Successfully deployed the AWS Retail Store app on Kubernetes.
- Verified cluster setup and application status using kubectl.
- Accessed the app externally via LoadBalancer/NodePort