

Kubernetes Retail Application Deployment Documentation

Step 1: Check Kubernetes Cluster

Command:

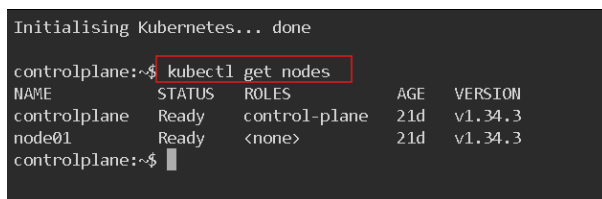
```
kubectl get nodes
```

Explanation:

This command checks whether the Kubernetes cluster is running properly. All worker nodes must show the status as **Ready** before proceeding.

If nodes are not ready, the deployments may fail because Kubernetes will not schedule pods on unhealthy nodes.

Image:



```
Initialising Kubernetes... done
controlplane:~$ kubectl get nodes
NAME                STATUS    ROLES    AGE   VERSION
controlplane        Ready     control-plane   21d   v1.34.3
node01              Ready     <none>         21d   v1.34.3
controlplane:~$
```

Screenshot showing all nodes in Ready state.

Step 2: Create Namespace

Command:

```
kubectl create namespace yash-ns
kubectl get ns
```

Explanation:

We create a namespace called **yash-ns** to logically separate the Retail application resources from other applications running in the cluster. After creating it, we verify that it appears in the namespace list.

Image:

```
controlplane:~$ kubectl create namespace yash-ns
namespace/yash-ns created
controlplane:~$ kubectl get ns
NAME                STATUS   AGE
default             Active   21d
kube-node-lease     Active   21d
kube-public         Active   21d
kube-system         Active   21d
local-path-storage  Active   21d
yash-ns             Active   8s
controlplane:~$
```

Screenshot showing yash-ns in namespace list.

Step 3: Create Working Directory

Command:

```
mkdir k8s-manifest-files
```

```
navigate to directory cd k8s-manifest-files
```

```
pwd
```

Note: If using the Git clone method, manual directory creation is not required.

Explanation:

We create a dedicated folder named **k8s-manifest-files** to store all Kubernetes YAML configuration files related to the Retail project. This keeps our project structured and organized.

Image:

```
controlplane:~$ mkdir k8s-manifest-files
controlplane:~$ cd k8s-manifest-files
controlplane:~/k8s-manifest-files$ pwd
/root/k8s-manifest-files
controlplane:~/k8s-manifest-files$
```

Screenshot showing directory creation and current path.

YAML Files Creation and Deployment Order

The files are created and applied in the following correct order:

1. ConfigMap
2. Secret
3. MongoDB Deployment
4. MongoDB Service
5. Retail App Deployment
6. Retail App Service

This order is important because deployments depend on ConfigMap and Secret.

Step 4: Create ConfigMap

Command:

```
vi configmap.yaml
```

YAML Content:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: retail-app-config
  namespace: yash-ns
data:
  MONGODB_URI: "mongodb://mongodb:27017/myDatabase"
  SESSION_SECRET: "1234"
  PORT: "3130"
  MONGO_INITDB_DATABASE: "myDatabase"
```

Save and verify: Esc → :wq

```
cat configmap.yaml
```

Explanation:

ConfigMap stores non-sensitive configuration values such as:

- Database connection string
- Application port
- Session secret

It separates configuration from application code, which is a best practice in Kubernetes.

Image:

```
controlplane:~/k8s-manifest-files$ vi configmap.yaml
controlplane:~/k8s-manifest-files$ cat configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: retail-app-config
  namespace: yash-ns
data:
  MONGODB_URI: "mongodb://mongodb:27017/myDatabase"
  SESSION_SECRET: "1234"
  PORT: "3130"
  MONGO_INITDB_DATABASE: "myDatabase"
controlplane:~/k8s-manifest-files$
```

Apply ConfigMap

```
kubectl apply -f configmap.yaml -n yash-ns
```

```
kubectl get cm -n yash-ns
```

Image:

```
controlplane:~/k8s-manifest-files$ kubectl apply -f configmap.yaml -n yash-ns
configmap/retail-app-config created
controlplane:~/k8s-manifest-files$ kubectl get cm -n yash-ns
NAME              DATA  AGE
kube-root-ca.crt  1      3m2s
retail-app-config  4      13s
controlplane:~/k8s-manifest-files$
```

Screenshot showing ConfigMap created successfully.

Step 5: Create Secret

Before creating the Secret file, sensitive values must be encoded.

Encode Secret Values

Command:

```
echo -n "your_email_user" | base64
echo -n "your_email_password" | base64
echo -n "chagantyteja2502@gmail.com" | base64
echo -n "yxoq bjuK rdnt alzp" | base64
```

Explanation:

- echo -n prints the value without adding a newline.
- base64 converts the value into encoded format.
- Kubernetes Secret requires Base64 encoded values.

Example:

```
echo -n "chagantyteja2502@gmail.com" | base64
```

Output:

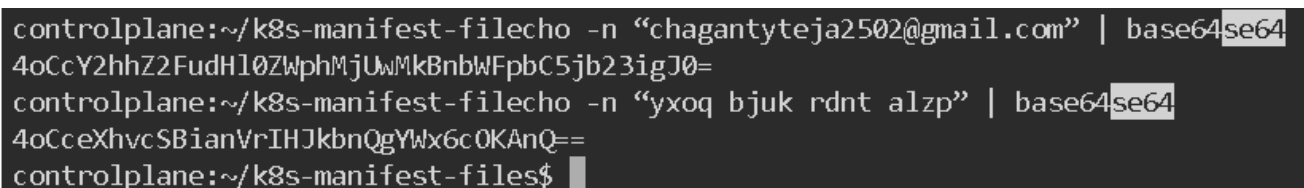
```
4oCcY2hhZ2FudHl0ZWphMjUwMkBnbWFpbC5jb23igJ0=
```

```
echo -n "yxoq bjuK rdnt alzp" | base64
```

Output:

```
4oCceXhvcSBianVrIHJkbnQgYWx6cOKAnQ==
```

Image:



```
controlplane:~/k8s-manifest-fileecho -n "chagantyteja2502@gmail.com" | base64se64
4oCcY2hhZ2FudHl0ZWphMjUwMkBnbWFpbC5jb23igJ0=
controlplane:~/k8s-manifest-fileecho -n "yxoq bjuK rdnt alzp" | base64se64
4oCceXhvcSBianVrIHJkbnQgYWx6cOKAnQ==
controlplane:~/k8s-manifest-files$
```

Create Secret File

Command:

```
vi secret.yaml
```

YAML Content:

```
apiVersion: v1
```

```
kind: Secret
```

```
metadata:
```

```
  name: retail-app-secret
```

```
  namespace: yash-ns
```

```
type: Opaque
```

```
data:
```

```
  EMAIL_USER:
```

```
4oCcY2hhZ2FudHl0ZWphMjUwMkBnbWFpbC5jb23igJ0=
```

```
  EMAIL_PASS: 4oCceXhvcSBianVrIHJkbnQgYWx6cOKAnQ==
```

Save and verify:

```
Esc → :wq
```

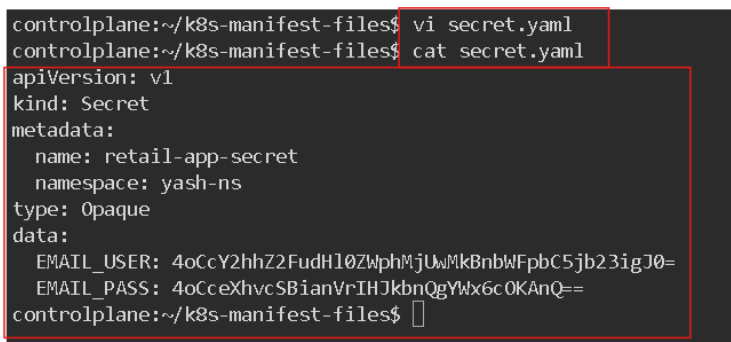
```
cat secret.yaml
```

Explanation:

The Secret file securely stores sensitive information such as email credentials.

The application will read these values securely during runtime.

Image:



```
controlplane:~/k8s-manifest-files$ vi secret.yaml
controlplane:~/k8s-manifest-files$ cat secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: retail-app-secret
  namespace: yash-ns
type: Opaque
data:
  EMAIL_USER: 4oCcY2hhZ2FudHl0ZWphMjUwMkBnbWFpbC5jb23igJ0=
  EMAIL_PASS: 4oCceXhvcSBianVrIHJkbnQgYWx6cOKAnQ==
controlplane:~/k8s-manifest-files$
```

Apply Secret

```
kubectl apply -f secret.yaml -n yash-ns
```

```
kubectl get secret -n yash-ns
```

Image:

```
controlplane:~/k8s-manifest-files$ kubectl apply -f secret.yaml -n yash-ns
secret/retail-app-secret created
controlplane:~/k8s-manifest-files$ kubectl get secret -n yash-ns
NAME                TYPE      DATA   AGE
retail-app-secret    Opaque    2       13s
controlplane:~/k8s-manifest-files$
```

Screenshot showing Secret created.

Step 6: Create MongoDB Deployment

Command:

```
vi retail-mongodb-deployment.yaml
```

YAML Content:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: retail-mongodb
  namespace: yash-ns
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
```

```
containers:
- name: mongodb
  image: mongo:latest
  ports:
  - containerPort: 27017
  env:
  - name: MONGO_INITDB_DATABASE
    valueFrom:
      configMapKeyRef:
        name: retail-app-config
        key: MONGO_INITDB_DATABASE
```

Save and verify:

Esc → :wq

cat retail-mongodb-deployment.yaml

Explanation:

This file defines the MongoDB deployment.
It creates one MongoDB pod inside the namespace.
It also reads the database name from the ConfigMap.

Image:

```
controlplane:~/k8s-manifest-files$ vi retail-mongodb-deployment.yaml
controlplane:~/k8s-manifest-files$ cat retail-mongodb-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: retail-mongodb
  namespace: yash-ns
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - name: mongodb
        image: mongo:latest
        ports:
        - containerPort: 27017
        env:
        - name: MONGO_INITDB_DATABASE
          valueFrom:
            configMapKeyRef:
              name: retail-app-config
              key: MONGO_INITDB_DATABASE
controlplane:~/k8s-manifest-files$
```


Apply MongoDB Deployment

kubectl apply -f retail-mongodb-deployment.yaml -n yash-ns

Explanation:

Kubernetes creates:

- Deployment
- ReplicaSet
- Pod

Image:

```
controlplane:~/k8s-manifest-files$ kubectl apply -f retail-mongodb-deployment.yaml -n yash-ns
deployment.apps/retail-mongodb created
```

Screenshot showing MongoDB pod created

Step 7: Create MongoDB Service

Command:

vi retail-mongodb-svc.yaml

YAML Content:

```
apiVersion: v1
kind: Service
metadata:
  name: mongodb
  namespace: yash-ns
spec:
  selector:
    app: mongodb
  ports:
    - port: 27017
```

```
targetPort: 27017
protocol: TCP
```

Save and verify:

Esc → :wq

```
cat retail-mongodb-svc.yaml
```

Explanation:

This service exposes MongoDB internally within the cluster. Other pods can connect to MongoDB using the service name `mongodb`.

Image:

```
controlplane:~/k8s-manifest-files$ vi retail-mongodb-svc.yaml
controlplane:~/k8s-manifest-files$ cat retail-mongodb-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: mongodb
  namespace: yash-ns
spec:
  selector:
    app: mongodb
  ports:
    - port: 27017
      targetPort: 27017
      protocol: TCP
controlplane:~/k8s-manifest-files$
```

Apply MongoDB Service

```
kubectl apply -f retail-mongodb-svc.yaml -n yash-ns
```

```
kubectl get svc -n yash-ns
```

Image:

```
controlplane:~/k8s-manifest-files$ kubectl apply -f retail-mongodb-svc.yaml -n yash-ns
service/mongodb created
controlplane:~/k8s-manifest-files$ kubectl get svc -n yash-ns
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)    AGE
mongodb      ClusterIP   10.101.157.71 <none>       27017/TCP  12s
controlplane:~/k8s-manifest-files$
```

Screenshot showing MongoDB service created.

Step 8: Create Retail App Deployment

Command:

```
vi retail-app-deployment.yaml
```

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: yash-deployment
```

```
  namespace: yash-ns
```

```
spec:
```

```
  replicas: 4
```

```
  selector:
```

```
    matchLabels:
```

```
      app: retail
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

```
        app: retail
```

```
    spec:
```

```
      containers:
```

```
        - name: retail-container
```

```
          image: saiteja2502/userprofileretail:latest
```

```
          ports:
```

```
            - containerPort: 3130
```

```
          env:
```

- name: MONGODB_URI
valueFrom:
configMapKeyRef:
 - name: retail-app-config
 - key: MONGODB_URI
- name: SESSION_SECRET
valueFrom:
configMapKeyRef:
 - name: retail-app-config
 - key: SESSION_SECRET
- name: PORT
valueFrom:
configMapKeyRef:
 - name: retail-app-config
 - key: PORT
- name: EMAIL_USER
valueFrom:
secretKeyRef:
 - name: retail-app-secret
 - key: EMAIL_USER
- name: EMAIL_PASS
valueFrom:
secretKeyRef:
 - name: retail-app-secret

key: EMAIL_PASS

Save and verify:

Esc → :wq

cat retail-app-deployment.yaml

Explanation:

This deployment creates 4 replicas of the Retail application.

It reads:

- Database connection from ConfigMap
- Email credentials from Secret

Since both already exist, pods will start successfully.

Image:

```
controlplane:~/k8s-manifest-files$ vi retail-app-deployment.yaml
controlplane:~/k8s-manifest-files$ cat retail-app-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: yash-deployment
  namespace: yash-ns
spec:
  replicas: 4
  selector:
    matchLabels:
      app: retail
  template:
    metadata:
      labels:
        app: retail
    spec:
      containers:
        - name: retail-container
          image: saiteja2502/userprofileretail:latest
          ports:
            - containerPort: 3130
          env:
            - name: MONGODB_URI
              valueFrom:
                configMapKeyRef:
                  name: retail-app-config
                  key: MONGODB_URI
            - name: SESSION_SECRET
              valueFrom:
                configMapKeyRef:
                  name: retail-app-config
                  key: SESSION_SECRET
            - name: PORT
              valueFrom:
                configMapKeyRef:
                  name: retail-app-config
```

```
- name: PORT
  valueFrom:
    configMapKeyRef:
      name: retail-app-config
      key: PORT
- name: EMAIL_USER
  valueFrom:
    secretKeyRef:
      name: retail-app-secret
      key: EMAIL_USER
- name: EMAIL_PASS
  valueFrom:
    secretKeyRef:
      name: retail-app-secret
      key: EMAIL_PASS
```

Apply Retail App Deployment

`kubectl apply -f retail-app-deployment.yaml -n yash-ns`

Image:

```
controlplane:~/k8s-manifest-files$ kubectl apply -f retail-app-deployment.yaml -n yash-ns
deployment.apps/yash-deployment created
```

Screenshot showing Deployment Created

Step 9: Create Retail Application Service

Command:

`vi retail-app-svc.yaml`

`apiVersion: v1`

`kind: Service`

`metadata:`

`name: retail-service`

`namespace: yash-ns`

`spec:`

type: LoadBalancer

ports:

- port: 3130

targetPort: 3130

protocol: TCP

selector:

app: retail

Save and verify:

Esc → :wq

cat retail-app-svc.yaml

Explanation:

This service exposes the Retail application externally using a LoadBalancer.

Image:

```
controlplane:~/k8s-manifest-files$ vi retail-app-svc.yaml
controlplane:~/k8s-manifest-files$ cat retail-app-svc.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  name: retail-service
  namespace: yash-ns
spec:
  type: LoadBalancer
  ports:
    - port: 3130
      targetPort: 3130
      protocol: TCP
  selector:
    app: retail
```

Apply Retail App Service

```
kubectl apply -f retail-app-svc.yaml -n yash-ns
```

```
kubectl get svc -n yash-ns
```

Explanation:

Kubernetes creates a LoadBalancer service resource. However, the EXTERNAL-IP remains <pending> because Killercode is not integrated with a cloud provider. Therefore, Kubernetes cannot provision an external load balancer.

As a result, the application cannot be accessed using a public IP in Killercode.

Image:

```
controlplane:~/k8s-manifest-files$ kubectl apply -f retail-app-svc.yaml -n yash-ns
service/retail-service created
controlplane:~/k8s-manifest-files$ kubectl get svc -n yash-ns
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
mongodb	ClusterIP	10.101.157.71	<none>	27017/TCP	9m12s
retail-service	LoadBalancer	10.99.157.118	<pending>	3130:31547/TCP	10s

Final Verification

Command:

```
kubectl get all -n yash-ns
```

Explanation:

This command displays:

- Pods
- Services
- Deployments
- ReplicaSets

All pods should be in **Running** state.

Image:

```
controlplane:~/k8s-manifest-files$ kubectl get all -n yash-ns
NAME                                READY   STATUS    RESTARTS   AGE
pod/retail-mongodb-58ffcb7cf9-79klp 1/1     Running   0           13m
pod/yash-deployment-64c485cbd7-5ztnc 1/1     Running   0           4m42s
pod/yash-deployment-64c485cbd7-n6zk2 1/1     Running   0           4m42s
pod/yash-deployment-64c485cbd7-tg2rt 1/1     Running   0           4m42s
pod/yash-deployment-64c485cbd7-vzgpv 1/1     Running   0           4m42s

NAME                                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service/mongodb                     ClusterIP     10.101.157.71 <none>        27017/TCP        10m
service/retail-service               LoadBalancer 10.99.157.118 <pending>     3130:31547/TCP   64s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/retail-mongodb      1/1     1             1           13m
deployment.apps/yash-deployment     4/4     4             4           4m42s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/retail-mongodb-58ffcb7cf9 1         1         1       13m
replicaset.apps/yash-deployment-64c485cbd7 4         4         4       4m42s
controlplane:~/k8s-manifest-files$
```

Verify Node Details

Command:

```
kubectl get nodes -o wide
```

Explanation:

This command displays detailed information about all nodes in the Kubernetes cluster.

Image:

```
controlplane:~/k8s-manifest-files$ kubectl get nodes -o wide
NAME        STATUS   ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
controlplane Ready    control-plane 21d   v1.34.3   172.30.1.2    <none>        Ubuntu 24.04.3 LTS   6.8.0-94-generic containerd://1.7.28
node01      Ready    <none>     21d   v1.34.3   172.30.2.2    <none>        Ubuntu 24.04.3 LTS   6.8.0-94-generic containerd://1.7.28
```

LoadBalancer will Not Work in Killercoda

In this project, we initially created the Retail service using:

type: LoadBalancer

A LoadBalancer service works only when Kubernetes is integrated with a cloud provider such as:

- AWS
- Azure
- Google Cloud

These cloud providers automatically create an external load balancer and assign a public IP address.

However, Killercoda is a lab-based container environment. It does not have integration with any cloud provider.

Because of this:

- EXTERNAL-IP remains <pending>
- No public IP is generated
- The application cannot be accessed using LoadBalancer

That is why LoadBalancer did not give correct web access in Killercoda.

It Works in AWS EC2 (Real Cloud Environment)

To Access Application in Browser

Command:

```
kubectl get svc -n yash-ns
```

Copy the EXTERNAL-IP and open:

<http://<EXTERNAL-IP>:3130>