**Agenda**

**DEMO: ClusterIP Service in Kubernetes**

**Step 1: Access & Inspect the Kubernetes Cluster**

Before creating and applying deployments and services, we first need to check and inspect the cluster to gather information.

1. **List all available API resources in the Kubernetes cluster:**

kubectl api-resources

This command will show you all the available API resources that can be used in Kubernetes, such as Pods, Services, Deployments, etc.

1. **List all the API versions in the cluster:**

kubectl api-versions

This will show you all the API versions supported in your cluster, which is useful for ensuring that you are using the correct API version in your manifests.

1. **Get cluster information:**

kubectl cluster-info

This command provides details about the Kubernetes master and services running within the cluster, such as the Kubernetes dashboard URL.

1. **List the Pods running in the kube-system namespace:**

kubectl get pods -n kube-system

This will display the pods running in the kube-system namespace, which contains system-critical components of your Kubernetes cluster.

1. **Get nodes and their detailed information:**

kubectl get nodes -o wide

This will show you the nodes in your cluster with additional details like IP addresses and the status of the nodes.

1. **Explain the Deployment resource:**

kubectl explain deployment

This command provides an explanation of the Deployment resource, including its fields and usage.

**Step 2: Create and Apply Kubernetes Deployment using YAML Manifest**

1. **Deployment YAML manifest:**

In this step, we will create a Deployment to manage our web server pods.

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**labels:**

**app: webserver**

**name: webserver**

**spec:**

**replicas: 2**

**selector:**

**matchLabels:**

**app: webserver**

**template:**

**metadata:**

**labels:**

**app: webserver**

**spec:**

**containers:**

**- image: httpd**

**name: httpd**

1. **Apply the Deployment:**

Save the YAML manifest as webser\_vig\_deploy.yaml and apply it using:

kubectl apply -f webser\_vig\_deploy.yaml

This will create a deployment named webserver with 2 replicas, each running the httpd image.

**Step 3: Create and Apply ClusterIP Service using YAML Manifest**

1. **Service YAML manifest:**

Next, create a ClusterIP service to expose the webserver pods within the cluster.

**apiVersion: v1**

**kind: Service**

**metadata:**

**labels:**

**app: webserver**

**name: clusterip-service**

**spec:**

**ports:**

**- name: 8080-80**

**port: 8080**

**protocol: TCP**

**targetPort: 80**

**selector:**

**app: webserver**

**type: ClusterIP**

1. **Apply the Service:**

Save the YAML manifest as webser\_vig\_cip.yaml and apply it using:

kubectl apply -f webser\_vig\_cip.yaml

This will create a ClusterIP service named clusterip-service, which will route traffic on port 8080 to port 80 on the webserver pods.

**Step 4: Access ClusterIP Service from Within the Cluster**

Now, we will test access to the ClusterIP service from within the cluster.

1. **Create a test pod for accessing the service:**

Use the following command to create a test pod that you can use to access the service inside the cluster:

kubectl run test-pod --rm -it --image=alpine:latest -- /bin/sh

1. **Install curl in the test pod:**

Inside the test-pod, install curl to make the HTTP request to the ClusterIP service:

**apk add --no-cache curl**

1. **Test the connection to the service:**

Once curl is installed, you can test the service by making a request to the ClusterIP address of the service:

**curl -v 10.104.192.35:8080**

Replace 10.104.192.35 with the actual IP of your service. This should give you the response from the httpd container running in your pods.

**Step 5: Access ClusterIP Service from Outside the Cluster**

To test access from outside the Kubernetes cluster (e.g., from your local machine or EC2 instance), you need to ensure that the IP 10.104.192.35 (ClusterIP) is accessible to your external client. However, ClusterIP services are not directly accessible from outside the cluster.

**Alternative approach:**

1. **Access the service through an external tool (e.g., kubectl port-forward):**

Since ClusterIP is internal, one way to access the service externally is to use kubectl port-forward:

**kubectl port-forward service/clusterip-service 8080:8080**

This will forward port 8080 from the clusterip-service to your local machine's port 8080.

1. **Test with curl:**

Now you can access the service externally:

curl -v http://localhost:8080

This should return the response from the httpd server running in the pods.