**1. Basic Deployment**

**Scenario:** Deploy a simple Nginx web server.

**YAML Definition**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 3 # Number of replicas to run

selector:

matchLabels:

app: nginx # Label selector to match pods

template:

metadata:

labels:

app: nginx # Label that will be applied to the pods

spec:

containers:

- name: nginx

image: nginx:latest # Image to run

ports:

- containerPort: 80 # Port exposed by container

**kubectl Command to Create the Deployment**

kubectl apply -f nginx-deployment.yaml

Explanation:

* kubectl apply -f nginx-deployment.yaml is used to create the deployment from the YAML file.
* The deployment will ensure 3 replicas of Nginx pods are running, each listening on port 80.

**Check the Deployment Status**

kubectl get deployments

* This command shows the current state of the deployments in the cluster.

**Get Pods Managed by the Deployment**

kubectl get pods

* Displays the list of pods created under the nginx-deployment.

**Check the Details of the Deployment**

kubectl describe deployment nginx-deployment

* This command gives detailed information about the deployment, such as the number of pods, strategy used, and the status of the pods.

**Rollout the Update (Nginx Version Change)**

Change the image to a new version and apply the updated deployment YAML:

image: nginx:1.21.0

**Apply the changes:**

kubectl apply -f nginx-deployment.yaml

**To track the update rollout status:**

kubectl rollout status deployment/nginx-deployment

* This command shows whether the update is successful or if there's a failure.

**2. Deployment with Resource Requests and Limits**

**Scenario:** Deploy a web server with resource requests and limits.

**YAML Definition**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 2

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

resources:

requests:

memory: "64Mi"

cpu: "250m"

limits:

memory: "128Mi"

cpu: "500m"

**kubectl Command to Create the Deployment with Resource Limits**

kubectl apply -f nginx-resource-deployment.yaml

Explanation:

* **Requests**: These are the resources that Kubernetes will guarantee to the container.
* **Limits**: These are the maximum resources a container can use. If it tries to use more than the specified limit, it will be throttled or killed.

**Check Resource Usage**

kubectl top pods

* This will show the current resource usage of the pods.

**3. Deployment with Environment Variables**

**Scenario:** Add environment variables to the deployment.

**YAML Definition**

apiVersion: apps/v1

kind: Deployment

metadata:

name: web-app-deployment

spec:

replicas: 2

selector:

matchLabels:

app: web-app

template:

metadata:

labels:

app: web-app

spec:

containers:

- name: web-app

image: my-web-app:latest

ports:

- containerPort: 8080

env:

- name: APP\_MODE

value: "production"

- name: DB\_HOST

value: "db.example.com"

**kubectl Command to Apply the Deployment with Environment Variables**

kubectl apply -f web-app-deployment.yaml

Explanation:

* The env section adds environment variables (APP\_MODE and DB\_HOST) that will be available to the container during runtime.

**Check the Environment Variables Inside the Pod**

To check the environment variables:

kubectl exec -it <pod-name> -- printenv

* This command will print the environment variables for the container in the pod.

**4. Deployment with Persistent Storage**

**Scenario:** Create a deployment that uses a persistent volume (PV) and persistent volume claim (PVC) for data storage.

**Persistent Volume and Persistent Volume Claim Definition**

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv-storage

spec:

capacity:

storage: 1Gi

accessModes:

- ReadWriteOnce

hostPath:

path: "/mnt/data"

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: pvc-storage

spec:

resources:

requests:

storage: 1Gi

accessModes:

- ReadWriteOnce

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: app-with-storage

spec:

replicas: 1

selector:

matchLabels:

app: app-with-storage

template:

metadata:

labels:

app: app-with-storage

spec:

containers:

- name: web-app

image: my-web-app:latest

volumeMounts:

- mountPath: "/data"

name: storage-volume

volumes:

- name: storage-volume

persistentVolumeClaim:

claimName: pvc-storage

**kubectl Command to Create the Deployment with Persistent Storage**

kubectl apply -f app-with-storage.yaml

kubectl describe pod <podname>

Explanation:

* The persistent volume (pv-storage) provides storage at /mnt/data on the node.
* The persistent volume claim (pvc-storage) ensures the pod has access to the storage and mounts it inside the container at /data.

**5. Deployment with Rolling Update Strategy**

**Scenario:** Update a deployment using a rolling update strategy.

**YAML Definition**

apiVersion: apps/v1

kind: Deployment

metadata:

name: rolling-update-deployment

spec:

replicas: 3

strategy:

type: RollingUpdate

rollingUpdate:

maxSurge: 1 # Max number of pods that can be created above the desired number of pods

maxUnavailable: 1 # Max number of pods that can be unavailable during the update

selector:

matchLabels:

app: rolling-app

template:

metadata:

labels:

app: rolling-app

spec:

containers:

- name: app-container

image: my-app:v2

ports:

- containerPort: 8080

**kubectl Command to Create the Rolling Update Deployment**

kubectl apply -f rolling-update-deployment.yaml

**Track the Rollout**

kubectl rollout status deployment/rolling-update-deployment

**Describe the Deployment**

kubectl describe deployment <deployment name>

Explanation:

* **maxSurge**: Defines the maximum number of pods that can be scheduled above the desired count during an update.
* **maxUnavailable**: Defines the maximum number of pods that can be unavailable during the update.
* This ensures the deployment update happens with no downtime.

**6. Rollback Deployment**

**Scenario:** Rollback to the previous version of a deployment.

**Rollback Command**

kubectl rollout undo deployment/nginx-deployment

Explanation:

* This command rolls back the deployment to the previous version in case of any issues with the current deployment.

**Summary of kubectl Commands**

1. **Create Deployment from YAML:**

kubectl apply -f <deployment-file>.yaml

1. **View Deployment Status:**

kubectl get deployments

kubectl describe deployment <deployment-name>

1. **Get Pods Managed by a Deployment:**

kubectl get pods

1. **Track Rollout Progress:**

kubectl rollout status deployment/<deployment-name>

1. **Rollback Deployment:**

kubectl rollout undo deployment/<deployment-name>

1. **Exec into a Pod:**

kubectl exec -it <pod-name> -- /bin/sh

1. **Check Resource Usage:**

kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

kubectl top pods