Kubernetes deployments

What is a Kubernetes Deployment?

A Kubernetes Deployment is a powerful resource that automates the process of managing and scaling your application. It ensures that the desired state of your application is maintained, including the number of replicas (Pods) and the container version. Essentially, a Deployment allows you to define how your application should run and Kubernetes handles the rest.

A Deployment helps with:

- Scaling applications
- Rolling updates
- Rollbacks to previous versions
- Ensuring high availability and reliability of your app

How Do Deployments Work?

Create a Deployment: You define the desired state of your app, specifying the number of replicas and container configurations (image, ports, etc.).

Replicas Creation: Kubernetes creates a specified number of replicas based on your template. Each replica is an identical copy of the Pod running the container.

Continuous Monitoring: The Deployment controller ensures that the desired number of replicas are always running. If any Pod crashes, a new one is automatically created to replace it.

Rolling Updates: If you need to update your application (e.g., changing the container image), Kubernetes will update your Pods one by one, ensuring minimal downtime and avoiding service interruptions.

Scaling: You can adjust the number of replicas based on traffic demand or load.

Key Benefits of Kubernetes Deployments

1.Automated Replica Management:

- Kubernetes ensures the right number of replicas are always running, making sure your app remains highly available.
- No manual intervention needed for failures; Kubernetes automatically replaces failed Pods.

2.Rolling Updates:

- Allows you to deploy new versions with zero downtime. Kubernetes gradually replaces the old Pods with the new ones.
- Supports version rollbacks if the new deployment fails or causes issues, making it easy to revert to a stable state.

3.Scaling:

- Kubernetes allows you to scale up or down based on demand with simple commands.
- Horizontal scaling adds or removes replicas.
- Vertical scaling increases resources for each Pod (e.g., CPU or memory)

4.Self-Healing:

- Pods that fail or become unresponsive are automatically replaced by new ones.
- Kubernetes continuously monitors the health of Pods through readiness and liveness probes.

5.Declarative Configuration:

- You only need to declare the desired state of your application in YAML or JSON format, and Kubernetes handles all the actions to reach that state.
-

How to Create a Deployment in Kubernetes?

Prerequisites

You should already have the following installed and configured:

- ✓ kops installed and configured
- ✓ kubectl installed and configured
- ✓ AWS CLI (aws) installed and configured with IAM permissions
- ✓ An S3 bucket created and used for Kops state store
- ✓ A Kubernetes cluster created via Kops (1 master + 1 node)

Step 1: Define Your Pod Template

You first create a YAML file that specifies the desired state of your app, including the container image, ports, and any environment variables.

Here’s an example to create a pod:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-web-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-web-app
  template:
    metadata:
      labels:
        app: my-web-app
    spec:
      containers:
      - name: web
        image: nginx:latest
        ports:
        - containerPort: 8080
```

Step 2: Apply the YAML to Create the Deployment

Once the YAML file is ready, use the kubectl command to create the Deployment:

kubectl apply -f my-web-app.yaml

```
root@kiran:~# vi my-web-app.yaml
root@kiran:~# kubectl apply -f my-web-app.yaml
deployment.apps/my-web-app created
root@kiran:~# kubectl get deployment
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
my-web-app   0/3     3            0           12s
```

This will create a Deployment with 3 replicas running the specified image.

Updating a Deployment

To update your application, you modify the Pod template (e.g., changing the container image version) and apply the changes.

Example of a YAML update for a new version:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-web-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-web-app
  template:
    metadata:
      labels:
        app: my-web-app
    spec:
      containers:
      - name: web
        image: nginx
        ports:
        - containerPort: 8080
```

Then, apply the update:

kubectl apply -f my-web-app.yaml

```
root@kiran:~# vi my-web-app.yaml
root@kiran:~# kubectl apply -f my-web-app.yaml
deployment.apps/my-web-app configured
root@kiran:~# kubectl get pod
NAME                          READY   STATUS              RESTARTS   AGE
my-web-app-656d7f5bc4-kw7sv   1/1     Running             0          8s
my-web-app-656d7f5bc4-p824n   1/1     Running             0          5s
my-web-app-656d7f5bc4-tpwfs   0/1     ContainerCreating   0          2s
my-web-app-7b4df479-gsfpx     1/1     Running             0          3m34s
root@kiran:~# kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
my-web-app-656d7f5bc4-kw7sv   1/1     Running   0          23s
my-web-app-656d7f5bc4-p824n   1/1     Running   0          20s
my-web-app-656d7f5bc4-tpwfs   1/1     Running   0          17s
```

Kubernetes will automatically perform a **rolling update**, replacing old Pods with the new version gradually.

Rolling Updates & Rollbacks

Rolling Updates:

- Kubernetes gradually replaces the old Pods with new ones to minimize downtime.
- Customizable: You can configure how many Pods can be unavailable during the update using parameters like maxUnavailable.

Rollback:

- If the new version causes issues, you can roll back to the previous version.

To roll back:

kubectl rollout undo deployment/my-web-app

```
root@kiran:~# kubectl rollout undo deployment/my-web-app
deployment.apps/my-web-app rolled back
```

Scaling a Deployment

You can scale your Deployment based on the traffic and resource needs.

Horizontal Scaling:

To increase or decrease the number of replicas:

kubectl scale deployment/my-web-app --replicas=5

This command will scale the application to 5 replicas.

```
root@kiran:~# kubectl get deployment
NAME         READY   UP-TO-DATE   AVAILABLE
my-web-app   5/5     5            5
```

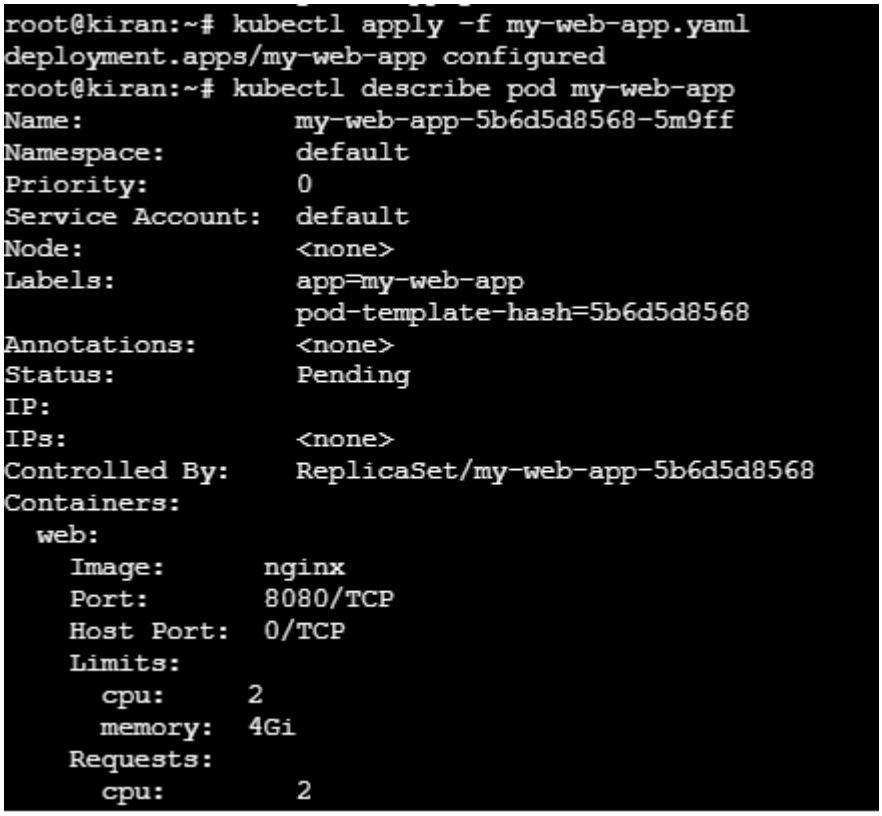
Vertical Scaling:

To change the resources allocated to each Pod (e.g., increasing memory or CPU):

Apply the update:

kubectl apply -f my-web-app.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-web-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-web-app
  template:
    metadata:
      labels:
        app: my-web-app
    spec:
      containers:
      - name: web
        image: nginx
        ports:
        - containerPort: 8080
        resources:
          limits:
            cpu: "2"
            memory: "4Gi"
```



Health Checks & Self-Healing

Kubernetes uses liveness and readiness probes to check the health of Pods:

Liveness Probe: Checks if the app inside the Pod is running correctly. If it fails, Kubernetes restarts the Pod.

Readiness Probe: Checks if the app is ready to receive traffic. If it fails, traffic is stopped from being sent to the Pod.

Example of a liveness probe in YAML:

livenessProbe:

httpGet:

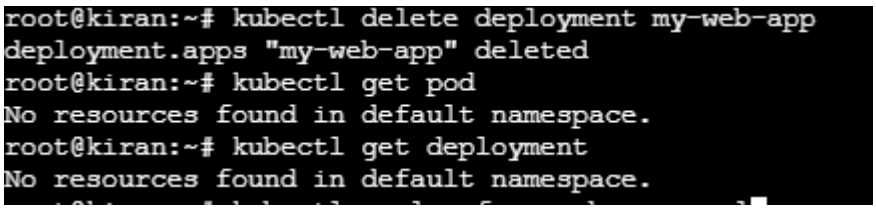
path: /healthz

port: 8080

Kubernetes also supports self-healing, meaning that if a Pod crashes or fails, Kubernetes automatically creates a new replica to replace it.

Command to Delete a Deployment:

kubectl delete deployment my-web-app



Deployment Strategies

1. Rolling Update (Default Strategy)

This is the default and most commonly used deployment method in Kubernetes. It updates Pods incrementally to avoid downtime. As new Pods are brought up, the old ones are terminated step-by-step. You can control the pace using parameters like **maxUnavailable** and **maxSurge**. This approach ensures continuous availability and smooth transitions during application upgrades.

Key Points:

- Gradual replacement of old Pods with new ones.
- No downtime.
- Configurable update strategy for availability control.
- Suitable for high-availability environments.

2. Recreate Strategy

This strategy first deletes all the existing Pods and then starts new ones. It is simpler than rolling updates but causes downtime since the application is unavailable during the transition. It's useful when your application cannot handle multiple versions running at the same time or when shared resources would cause conflicts.

Key Points:

- All existing Pods are shut down before new ones start.
- Causes downtime during deployment.
- Useful when parallel versions are not supported.

3. Blue/Green Deployment

Blue/Green involves maintaining two separate environments: one (Blue) runs the current production version, while the other (Green) runs the new version. After validating the Green version, traffic is switched over from Blue to Green. This method allows full testing in production-like conditions before going live.

Key Points:

- Two environments: current (Blue) and new (Green).
- Seamless switch once the new version is verified.
- Easy rollback by switching traffic back to Blue.
- Requires infrastructure support for dual environments.

4. Canary Deployment

Canary deployment releases the new version to a small subset of users or servers first. Based on the performance and stability, the rollout can be expanded gradually to the rest of the users. It minimizes risk and enables testing in a live environment with limited exposure.

Key Points:

- New version is rolled out to a small percentage initially.
- Observes real user impact before full release.
- Can scale gradually if successful, or stop/rollback if issues arise.
- Helps in performance monitoring and fault isolation.

These deployment strategies provide different levels of control, risk management, and complexity. The choice depends on your application's architecture, tolerance for downtime, and deployment goals.

Summary

Kubernetes Deployments provide a powerful way to automate the management of applications, ensuring that your app is always available, can scale based on demand, and can be updated or rolled back smoothly with zero downtime.

Key features:

- Automated scaling and management of replicas.
- Rolling updates to deploy new versions with no downtime.
- Self-healing to replace failed Pods automatically.
- Declarative configuration: Just define the desired state, and Kubernetes ensures it.
- Easy rollback if something goes wrong.

Deployments simplify many complex tasks, allowing you to focus on building and improving your application.