Kubernetes Deployment for Zero-Downtime Updates

When developers update a file in a pod, such as upgrading from one version to a newer version, we need to manage the update seamlessly. The **Deployment object** in Kubernetes helps us achieve zero downtime while upgrading the application. It provides the ability to roll back, roll forward, and perform a rolling update to avoid downtime when introducing new features.

Steps to Write a Deployment Manifest File:

To perform such updates, we create a **deployment manifest file** in YAML format. Below is an example:

yaml
Copy code

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deploy
  labels:
    app: facebook
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-deploy
  template:
    metadata:
      name: my-pod
      labels:
        app: my-deploy
    spec:
      containers:
        - name: deployment
          image: <your-image>
```

Key Concepts:

- **Replicas**: Defines the number of pods you want to run. In this case, we are creating 3 replicas.
- **Selectors**: The selector ensures that the pods created by this deployment match the labels specified.
- **Rolling Update**: Kubernetes will gradually replace old pods with new ones to ensure minimal downtime during the upgrade process.

Deployment Strategies:

1. **Rolling Update**: This is the default strategy, where Kubernetes gradually replaces old pods with new ones. It's important for achieving zero downtime during application updates.
   - **maxSurge**: Specifies the maximum number of pods that can be created above the desired number of pods during an update.
   - **maxUnavailable**: Specifies the maximum number of pods that can be unavailable during the update.

Example for a rolling update:
yaml
Copy code

```
strategy:
  rollingUpdate:
    maxSurge: 1
    maxUnavailable: 0
```

2.
3. **Blue-Green Deployment**: In this strategy, two environments (blue and green) are used to ensure that a fully functional version of the application is always available to users.
4. **Canary Deployment**: This strategy allows you to deploy a new version to a small subset of users before fully rolling it out. It helps to test the new version in production with minimal risk.

Managing Deployment with kubectl Commands:
To **apply** the deployment:
bash
Copy code

```
kubectl apply -f deployment.yaml
```

-

To **check the status** of deployments:
bash
Copy code

```
kubectl get deployments
```

-

To **undo** a rollout (e.g., if something goes wrong):
bash
Copy code

```
kubectl rollout undo deployment fb-deploy
```

-
- **Port mapping**: You can expose your service to the outside world by specifying ports in the manifest file.

Zero-Downtime Upgrade:

Kubernetes handles rolling updates in a way that ensures zero downtime. When upgrading, Kubernetes will create new pods, and as the new pods become ready, it will gradually terminate the old ones, ensuring that the service remains available.

Best Practices:

- **Manifest File Structure**: Use proper indentation, especially in the spec and containers sections.
- **Versioning**: Keep the container image version updated to ensure smooth transitions during upgrades.
- **Port Mapping**: Ensure the application is accessible via correct ports when defining the deployment.

By using **Deployment objects**, we ensure our Kubernetes pods are upgraded seamlessly without affecting the users, even during production updates.