Strategies for Kubernetes Deployments: Balancing Downtime and User Experience

Introduction

In the fast-paced world of software development, staying current with application versions is crucial. Kubernetes, a robust container orchestration system, offers deployment strategies to facilitate seamless updates. This blog explores two primary deployment strategies - Rolling Update and Recreate - and delves into advanced testing strategies like Blue/Green Deployment, Canary Deployment, A/B Testing, and more.

Understanding Rolling Update and Recreate

When updating applications on Kubernetes, two main strategies come into play - Rolling Update and Recreate. Rolling Update ensures continuous availability by updating pods one by one, gradually replacing the old version with the new. Recreate, however, involves terminating all existing pods before launching the new version, causing temporary downtime. While Rolling Update provides 100% uptime, Recreate is faster but entails downtime.

Choosing the Right Strategy

The choice between Rolling Update and Recreate depends on specific use cases. Recreate is advantageous for swift updates, such as changing the base system OS or performing maintenance. If uninterrupted service is paramount, Rolling Update becomes the preferred strategy despite its slower pace.

Advanced Deployment Strategies

As applications become more complex, testing strategies become essential for a smooth transition. Popular testing strategies include Blue/Green Deployment, Canary Deployment, Shadow Deployment, A/B Testing, and Ramped Slow Rollout.

Recreate Strategy

The Recreate strategy involves deleting all existing pods before creating new ones, causing temporary downtime. Here's an example YAML code for a basic deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-app-container
        image: your-image:tag
        ports:
          - containerPort: 80
  strategy:
    type: Recreate
```

Rolling Update

Here's an example YAML file specifying a rolling update strategy for a deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: your-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: your-app
  template:
    metadata:
      labels:
        app: your-app
    spec:
      containers:
        - name: your-container
          image: your-image:latest
          ports:
            - containerPort: 80
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
      minReadySeconds: 5
```

Tools and Kubernetes Tricks

While Kubernetes supports Rolling Update and Recreate, advanced strategies like Canary Deployment are not directly supported. Tools like Istio and the concept of operators can be employed to implement these strategies successfully. A/B Testing helps in making informed decisions about the most user-friendly version.

Blue/Green Deployment

This strategy involves running two identical environments - Blue (current version) and Green (new version), seamlessly switching users between them, minimizing downtime, and enabling easy rollback if issues arise.

Canary Deployment

In Canary Deployment, a small percentage of user traffic is directed to the new version (Canary), allowing for real-time user feedback. If successful, the rollout expands gradually; otherwise, it can be aborted without affecting the entire user base.

```yaml
# Example for Canary Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-canary
spec:
  replicas: 5
  selector:
    matchLabels:
      app: my-app
      version: v1
  template:
    metadata:
      labels:
        app: my-app
        version: v1
    spec:
      containers:
        - name: my-container
          image: my-image:v1

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-v2
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
      version: v2
  template:
    metadata:
      labels:
        app: my-app
        version: v2
    spec:
      containers:
        - name: my-container
          image: my-image:v2
```

To direct traffic based on labels, create a NodePort service:

```
# Example NodePort Service
apiVersion: v1
kind: Service
metadata:
  name: my-app-service
spec:
  type: NodePort
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Scale the deployment:kubectl scale deployment my-app-canary -replicas=6

Blue-Green Deployments: Seamlessly Transitioning Environments

Blue-Green Deployments offer a risk-free method for rolling out updates by maintaining two identical environments. Follow this guide:

Creating Duplicate Environments

Set up two identical environments - 'Blue' for the current production version and 'Green' for the new version.

```
# Example Service for Blue Environment
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app-green
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Apply the service switch:

```
kubectl apply -f service-green.yaml
```

Validation and Rollback

Thoroughly validate the 'Green' environment. If issues arise, rolling back is as simple as redirecting traffic back to 'Blue.'

```
kubectl apply -f service-blue.yaml
```

Traffic Switching

Once the 'Green' environment is ready, switch traffic seamlessly using Kubernetes' service abstraction. Example YAML code for service switch:

```
# Example Service Switch
# Apply this YAML after validating the 'Green' environment
kubectl apply -f service-switch.yaml
```

Implement these strategies judiciously based on your deployment needs, adjusting parameters as required. Kubernetes provides a powerful platform for orchestrating deployments, ensuring a balance between downtime and user experience.

Check_Out_Detailed_Blog:-
https://medium.com/@srivastavayushmaan1347/strategies-for-kubernetes-deployments-balancing-downtime-and-user-experience-part5-c17b647f4253