DevOps Shack

Different CICD Pipelines For Different Deployment Stratgeies

1. Rolling Deployment

Description

A rolling deployment gradually replaces instances of the previous version of an application with instances of the new version until all instances are updated.

GitHub Actions Workflow

```
name: Rolling Deployment

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '14'

    - name: Install dependencies
      run: npm install

    - name: Run tests
      run: npm test
```

```
  deploy:
    needs: build
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Deploy to Kubernetes
      uses: azure/k8s-deploy@v3
      with:
        namespace: default
        manifests: |
          manifests/deployment.yaml
        images: |
          my-app:latest
        strategy: rolling
```

2. Blue-Green Deployment

Description

A blue-green deployment involves running two identical production environments, only one of which (the green environment) serves live production traffic. The blue environment is used to stage the new version of the application, which is then switched to production.

GitHub Actions Workflow

```
name: Blue-Green Deployment

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '14'

    - name: Install dependencies
      run: npm install

    - name: Run tests
      run: npm test
```

```
  deploy:
    needs: build
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Deploy to Blue environment
      run: kubectl apply -f manifests/blue-deployment.yaml

    - name: Switch traffic to Blue environment
      run: kubectl apply -f manifests/blue-service.yaml
```

3. Canary Deployment

Description

A canary deployment involves deploying the new version of the application to a small subset of users initially, then gradually increasing the number of users over time.

GitHub Actions Workflow

```
name: Canary Deployment

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '14'

    - name: Install dependencies
      run: npm install

    - name: Run tests
      run: npm test

  deploy:
    needs: build
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2
```

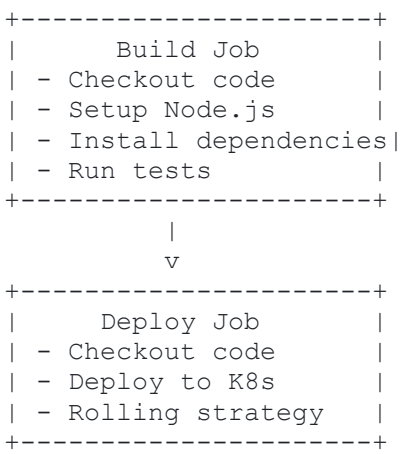
```
    - name: Deploy Canary version
      run: kubectl apply -f manifests/canary-deployment.yaml

    - name: Monitor Canary deployment
      run: ./scripts/monitor_canary.sh

    - name: Scale up Canary deployment
      run: kubectl scale deployment my-app-canary --replicas=10

    - name: Promote Canary to stable
      run: |
        kubectl delete deployment my-app-stable
        kubectl apply -f manifests/canary-deployment.yaml -f
manifests/stable-service.yaml
```

Explanations and Diagrams

Rolling Deployment

- **Build Job**: Checks out the code, sets up Node.js, installs dependencies, and runs tests.
- **Deploy Job**: Needs the build job to complete, checks out the code again, and deploys the new version using a rolling update strategy.

Diagram:



Blue-Green Deployment

- **Build Job**: Same as the rolling deployment.
- **Deploy Job**: Deploys the new version to the blue environment and switches traffic to it.

Diagram:

```
+----------------------+
|      Build Job       |
| - Checkout code      |
```

```
| - Setup Node.js      |
| - Install dependencies|
| - Run tests          |
+----------------------+
          |
          v
+----------------------+
|     Deploy Job       |
| - Checkout code      |
| - Deploy to Blue     |
| - Switch traffic     |
+----------------------+
```

Canary Deployment

- **Build Job**: Same as the rolling deployment.
- **Deploy Job**: Deploys a canary version, monitors it, scales it up, and finally promotes it to stable.

Diagram:

```
+----------------------+
|      Build Job       |
| - Checkout code      |
| - Setup Node.js      |
| - Install dependencies|
| - Run tests          |
+----------------------+
          |
          v
+-----------------------------+
|         Deploy Job          |
| - Checkout code             |
| - Deploy Canary version     |
| - Monitor Canary deployment |
| - Scale up Canary deployment|
| - Promote Canary to stable  |
+-----------------------------+
```

4. A/B Testing Deployment

Description

A/B testing deployment involves running two versions of the application simultaneously and directing a subset of users to each version to compare performance and user satisfaction.

GitHub Actions Workflow

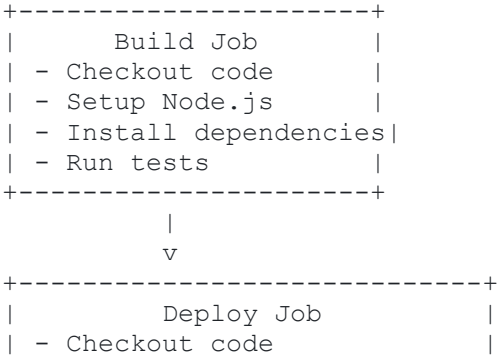
```
name: A/B Testing Deployment

on:
```

```
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '14'

    - name: Install dependencies
      run: npm install

    - name: Run tests
      run: npm test

  deploy:
    needs: build
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Deploy A version
      run: kubectl apply -f manifests/deployment-a.yaml

    - name: Deploy B version
      run: kubectl apply -f manifests/deployment-b.yaml

    - name: Configure A/B traffic split
      run: kubectl apply -f manifests/service-ab.yaml
```

Explanation

- **Build Job**: Same as previous examples.
- **Deploy Job**: Deploys both versions (A and B) and configures the traffic split to direct users to both versions.

Diagram:



```
| - Deploy A version          |
| - Deploy B version          |
| - Configure traffic split   |
+-----------------------------+
```

5. Shadow Deployment

Description

Shadow deployment involves deploying the new version of the application alongside the old version, but only the old version handles live traffic. The new version processes the same requests without serving the responses to test its performance.

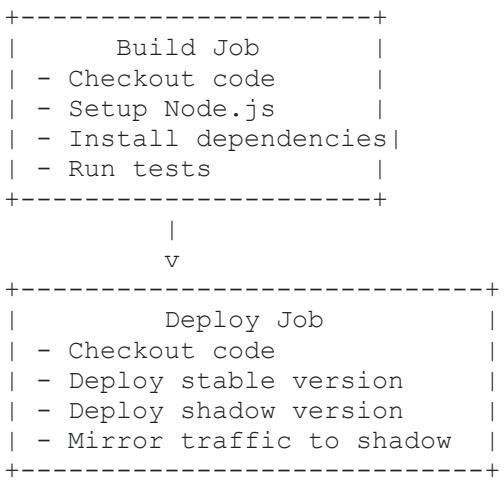
GitHub Actions Workflow

```
name: Shadow Deployment

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '14'

    - name: Install dependencies
      run: npm install

    - name: Run tests
      run: npm test

  deploy:
    needs: build
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Deploy stable version
      run: kubectl apply -f manifests/stable-deployment.yaml

    - name: Deploy shadow version
      run: kubectl apply -f manifests/shadow-deployment.yaml

    - name: Mirror traffic to shadow
      run: kubectl apply -f manifests/traffic-mirroring.yaml
```

Explanation

- **Build Job**: Same as previous examples.
- **Deploy Job**: Deploys the stable and shadow versions and configures traffic mirroring.

Diagram:



6. Recreate Deployment

Description

A recreate deployment involves taking down all instances of the previous version of the application and then deploying the new version.

GitHub Actions Workflow

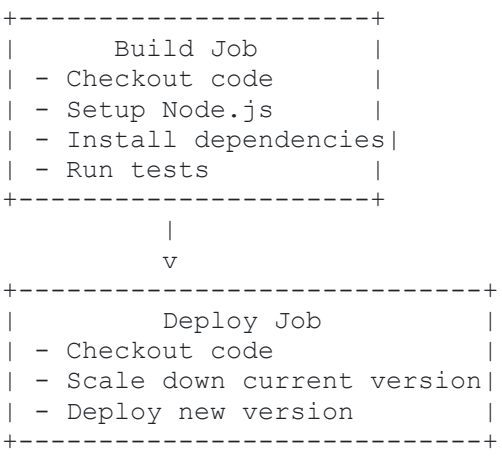
```
name: Recreate Deployment

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '14'

    - name: Install dependencies
      run: npm install
```

```
    - name: Run tests
      run: npm test

  deploy:
    needs: build
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Scale down current version
      run: kubectl scale deployment my-app --replicas=0

    - name: Deploy new version
      run: kubectl apply -f manifests/deployment.yaml
```

Explanation

- **Build Job**: Same as previous examples.
- **Deploy Job**: Scales down the current version to zero instances, then deploys the new version.

Diagram:



7. Rolling Update with Helm

Description

This deployment strategy uses Helm to manage Kubernetes applications, allowing a smooth rolling update of the application.

GitHub Actions Workflow

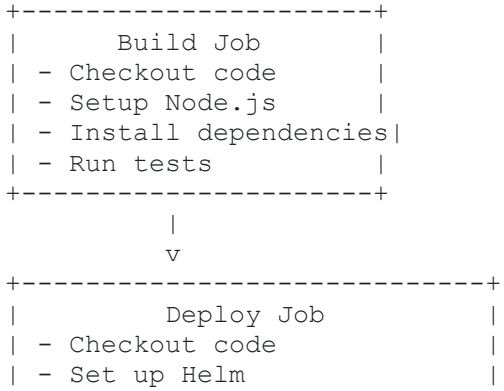
```
name: Rolling Update with Helm

on:
```

```
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '14'

    - name: Install dependencies
      run: npm install

    - name: Run tests
      run: npm test

  deploy:
    needs: build
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Helm
      run: |
        curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-
helm-3 | bash

    - name: Deploy with Helm
      run: helm upgrade --install my-app ./helm-chart
```

Explanation

- **Build Job**: Same as previous examples.
- **Deploy Job**: Sets up Helm and deploys the application using Helm for a rolling update.

Diagram:



```
| - Deploy with Helm          |
+-----------------------------+
```

8. Serverless Deployment

Description

Deploying a serverless application, such as an AWS Lambda function, using GitHub Actions.

GitHub Actions Workflow

```
name: Serverless Deployment

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '14'

    - name: Install dependencies
      run: npm install

    - name: Run tests
      run: npm test

  deploy:
    needs: build
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Serverless Framework
      run: npm install -g serverless

    - name: Deploy to AWS Lambda
      run: serverless deploy
      env:
        AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
        AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
```

Explanation

- **Build Job**: Same as previous examples.
- **Deploy Job**: Sets up the Serverless Framework and deploys the application to AWS Lambda.

Diagram: