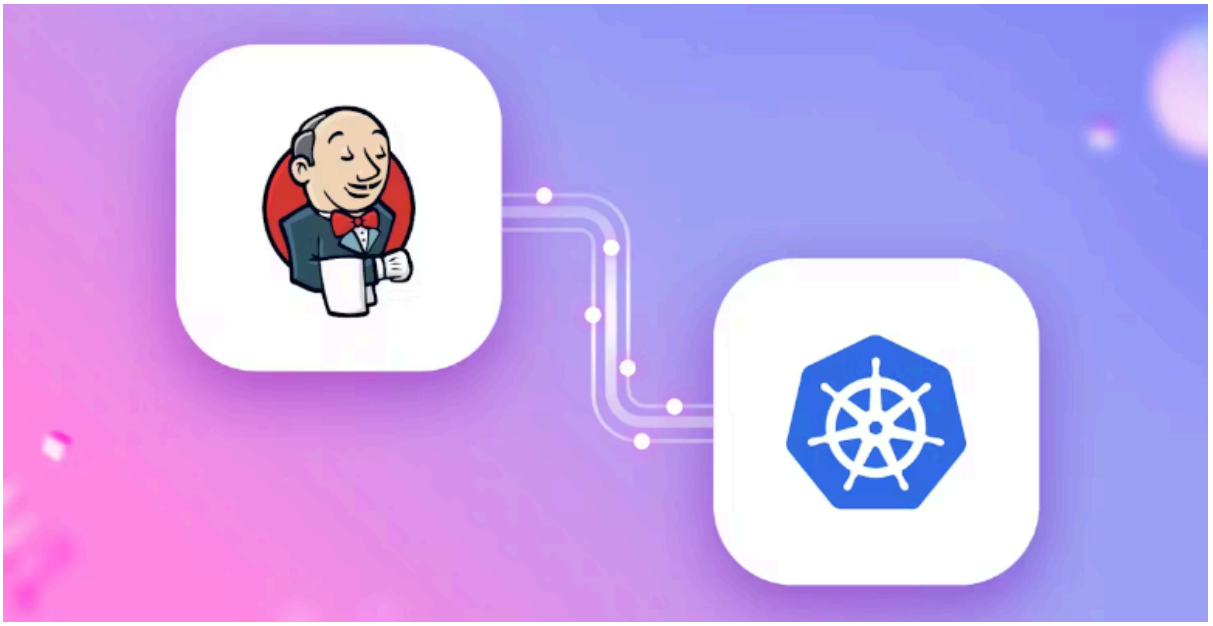
Setting Up a CI/CD Pipeline with Jenkins on Kubernetes

Written by Zayan Ahmed | 5 min read

Goal

Set up a Jenkins CI/CD pipeline to automatically build, test, and deploy a Node.js application to a Kubernetes cluster whenever code is pushed to a Git repository.



Prerequisites

1. **Kubernetes Cluster**: A running Kubernetes cluster with `kubectl` configured to access it.
2. **Jenkins Installed**: Jenkins installed either on the Kubernetes cluster or as a separate server.
3. **Docker Hub Account**: Docker Hub credentials to push images.
4. **Node.js Application**: A simple Node.js application with a Dockerfile, Kubernetes manifests, and a Jenkinsfile.

Step 1: Prepare Your Node.js Application

1. **Create a `Dockerfile`**:

```
# Use an official Node.js runtime as the base image
FROM node:14
```

```
# Create and set the application directory
WORKDIR /usr/src/app

# Copy package.json and install dependencies
COPY package*.json ./
RUN npm install

# Bundle app source
COPY . .

# Expose port and start application
EXPOSE 8080
CMD ["node", "app.js"]
```

2. **Kubernetes Manifests** (`k8s/deployment.yaml` and `k8s/service.yaml`):

```
# Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
 name: node-app
spec:
 replicas: 2
 selector:
   matchLabels:
     app: node-app
 template:
   metadata:
     labels:
       app: node-app
   spec:
     containers:
       - name: node-app
         image: <DOCKER_HUB_USERNAME>/node-app:latest
         ports:
           - containerPort: 8080

# Service
apiVersion: v1
kind: Service
metadata:
 name: node-app-service
spec:
```

```
 selector:
   app: node-app
 ports:
   - protocol: TCP
     port: 80
     targetPort: 8080
 type: LoadBalancer
```

3. **Jenkinsfile**:

```
pipeline {
    agent any
    environment {
        DOCKER_HUB_REPO = "<DOCKER_HUB_USERNAME>/node-app"
        DOCKER_HUB_CREDENTIALS = "docker-hub-credentials"
    }
    stages {
        stage('Build') {
            steps {
                script {
                    docker.build(DOCKER_HUB_REPO)
                }
            }
        }
        stage('Test') {
            steps {
                sh 'npm test'
            }
        }
        stage('Push to Docker Hub') {
            steps {
                script {

docker.withRegistry('https://registry.hub.docker.com',
DOCKER_HUB_CREDENTIALS) {
                        docker.image(DOCKER_HUB_REPO).push("latest")
                    }
                }
            }
        }
        stage('Deploy to Kubernetes') {
            steps {
                sh 'kubectl apply -f k8s/deployment.yaml'
```

```
                sh 'kubectl apply -f k8s/service.yaml'
            }
        }
    }
}
```

Step 2: Set Up Docker Hub Credentials in Jenkins

1. Go to **Jenkins Dashboard > Manage Jenkins > Manage Credentials**.
2. Add new **Username and Password** credentials with ID `docker-hub-credentials`, using your Docker Hub username and password.

Step 3: Configure the Jenkins Pipeline Job

1. On the Jenkins Dashboard, go to **New Item** and select **Pipeline**.
2. Name the job and scroll down to the **Pipeline** section.
3. Select **Pipeline script from SCM** and choose **Git**.
4. Add your Node.js application repository URL.
5. In the **Script Path** field, enter `Jenkinsfile`.

Step 4: Set Up Webhooks for Git Repository (GitHub or GitLab)

1. Go to your Git repository (GitHub or GitLab) and navigate to **Settings > Webhooks**.
2. Add a webhook URL pointing to `http://<YOUR_JENKINS_URL>/github-webhook/` or `http://<YOUR_JENKINS_URL>/gitlab-webhook/`.
3. Set the content type to **application/json**.

Step 5: Configure Jenkins to Run Jobs on Push

1. Go back to your Jenkins job and open **Configure**.
2. Under **Build Triggers**, select **GitHub hook trigger for GITScm polling** or **Build when a change is pushed to GitLab**.
3. Save the configuration.

Step 6: Run the Pipeline

1. Commit changes to the Git repository to trigger the Jenkins pipeline.

2. Check the Jenkins console to see each stage run:
    - **Build**: Jenkins will use Docker to build the image.
    - **Test**: Runs the application tests.
    - **Push to Docker Hub**: Pushes the image to Docker Hub.
    - **Deploy to Kubernetes**: Applies the Kubernetes manifests to the cluster.

Step 7: Verify the Deployment

1. Run the following command to check the status of the deployment:

```
kubectl get deployments
```

2. Check if the service is exposed:

```
kubectl get services
```

3. Access the application using the external IP provided by the LoadBalancer service.

Step 8: Monitor and Scale the Application

1. **Scaling the Deployment**:

```
kubectl scale deployment node-app --replicas=3
```

2. **Monitor Logs**:

```
kubectl logs -f <POD_NAME>
```

3. Monitor Performance (Optional): Use monitoring tools like **Prometheus** and **Grafana** for application metrics.

Troubleshooting Tips

- **Docker Push Failures**: Check Docker Hub credentials in Jenkins.
- **Kubernetes Deployment Issues**: Review Kubernetes manifests, especially the `image` field, for any configuration errors.
- **Pipeline Failures**: View logs for each stage in Jenkins to diagnose issues.

Conclusion:

This setup provides a basic CI/CD pipeline that can be extended with additional stages (e.g., security scans, performance testing) or integration with tools like Slack for deployment notifications.

Follow me on **LinkedIn** for more 😊