CI/CD pipelines - Jenkins pipelines with Docker and Kubernetes

Setting up and troubleshooting a **CI/CD pipeline** with **Jenkins**, **Docker**, and **Kubernetes** is a powerful and scalable solution for automating the build, test, and deployment process. Here's a guide to help you set it up and troubleshoot common issues.

1. Prerequisites

- **Jenkins** installed and running (can be done via Docker or a native installation).
- **Docker** installed on the Jenkins server.
- **Kubernetes** cluster (either local with Minikube or cloud-based like AWS EKS, GKE, or Azure AKS).
- **Kubectl** installed on the Jenkins server to interact with the Kubernetes cluster.

---

2. Step-by-Step Setup

Step 1: Set Up Jenkins with Docker

1. **Install Jenkins on Docker**: You can run Jenkins inside a Docker container:
   `docker run -d -p 8080:8080 -p 50000:50000 --name jenkins -v jenkins_home:/var/jenkins_home jenkins/jenkins:lts`

   This will expose Jenkins on port `8080`.

2. **Install Docker Plugin**:
   In Jenkins, go to **Manage Jenkins** > **Manage Plugins**, search for the **Docker** plugin, and install it.
3. **Configure Docker**:
   To allow Jenkins to use Docker, ensure Jenkins is in the Docker group:
   `sudo usermod -aG docker jenkins`
4. **Test Docker in Jenkins**:
   Set up a simple pipeline that runs a Docker command to check if Docker is working:

```
pipeline {

agent any
stages {
```

```
        stage('Test Docker') {
            steps {
                script {
                    sh 'docker --version'
                }
            }
        }
    }
}
```

Step 2: Integrate Jenkins with Kubernetes

1. **Install Kubernetes Plugin**: In Jenkins, go to **Manage Jenkins** > **Manage Plugins**, search for **Kubernetes** plugin, and install it.
2. **Configure Kubernetes**:
    - **Jenkins** > **Manage Jenkins** > **Configure System** > **Cloud** section.
    - Add a new **Kubernetes Cloud** and set the **Kubernetes URL** (this is your cluster's API endpoint).
    - Add Jenkins credentials that allow it to communicate with Kubernetes. You can create a `ServiceAccount` for Jenkins and provide its token here.
3. **Configure Kubernetes Pods for Jenkins Agents**: Jenkins will use Kubernetes to dynamically spin up agents. In the **Pod Templates** section, you can configure what containers should be inside the agent pods.
4. **Test Kubernetes Agent**: Create a simple pipeline that runs on a Kubernetes agent:

```
pipeline {

    agent {
        kubernetes {
            yaml """
            apiVersion: v1
            kind: Pod
            spec:
              containers:
              - name: jnlp
                image: jenkins/inbound-agent
                args: ['\$(JENKINS_SECRET)', '\$(JENKINS_NAME)']
```

```
              - name: docker
                image: docker
                command:
                - cat
            """
        }
    }
    stages {
        stage('Run on K8s') {
            steps {
                sh 'docker --version'
            }
        }
    }
}
```

Step 3: Deploy to Kubernetes using Jenkins

1. **Docker Build and Push**: First, ensure Jenkins can build a Docker image and push it to a Docker registry:

```
pipeline {
    agent any
    stages {
        stage('Build Docker Image') {
            steps {
                script {
                    sh 'docker build -t
myapp:${BUILD_NUMBER} .'
                    sh 'docker tag myapp:${BUILD_NUMBER}
myregistry/myapp:${BUILD_NUMBER}'
                    sh 'docker push
myregistry/myapp:${BUILD_NUMBER}'
                }
            }
        }
```

```
        }
    }
```

2. **Kubernetes Deployment**: Use `kubectl` to deploy the Docker image to the Kubernetes cluster. Make sure you configure `kubectl` correctly on Jenkins by copying the `kubeconfig` or by using a `ServiceAccount` for authentication:

```
    pipeline {
    agent any
    stages {
        stage('Deploy to Kubernetes') {
            steps {
                script {
                    sh 'kubectl set image deployment/myapp
myapp=myregistry/myapp:${BUILD_NUMBER}'
                }
            }
        }
    }
}
```

---

3. Common Troubleshooting Tips

Issue 1: Docker Command Fails in Jenkins

- Ensure that Jenkins has permission to run Docker commands. Check that the Jenkins user is in the `docker` group and has sufficient permissions.
- Check Docker Daemon status on the Jenkins server using:

```
systemctl status docker
```

Issue 2: Jenkins Kubernetes Plugin Can't Connect to Cluster

- Verify that the Kubernetes API URL is correct.
- Ensure the credentials (service account or token) are valid and have proper permissions to manage pods.

- Use `kubectl` on the Jenkins server manually to check the connection:

  ```
  kubectl get pods
  ```

Issue 3: Kubernetes Agents Are Not Spinning Up

- Ensure that Jenkins is configured to use the Kubernetes cloud.
- Check the Jenkins controller logs to see if it's able to request Kubernetes pods.
- Make sure the pod template is correctly configured with proper containers and arguments.

Issue 4: Unable to Push Docker Image to Registry

- Check if Jenkins has access to the Docker registry (credentials are correctly configured).
- Ensure the Jenkins environment has access to the internet if using a cloud registry.

Issue 5: Kubectl Command Fails

- Ensure `kubectl` is installed on the Jenkins server and that the correct `kubeconfig` or service account is being used.
- Try running the `kubectl` command directly on the Jenkins server to debug the issue.

Issue 6: Pipeline Build is Slow

- If builds are slow, consider using caching mechanisms for Docker layers or increasing the resource limits for Kubernetes agents.
- Check for network issues between Jenkins and the Docker registry or Kubernetes cluster.

---

4. Best Practices

- **Use Declarative Pipelines**: Jenkins declarative pipelines are easier to maintain and debug.
- **Separate Build and Deploy Pipelines**: Consider separating build and deploy pipelines for better management and control over different environments.

- **Use Jenkinsfiles**: Store your pipeline as code in `Jenkinsfile` within your repository to maintain versioning.

By following these steps, you can effectively set up a CI/CD pipeline that builds Docker images, pushes them to a registry, and deploys them to a Kubernetes cluster, with Jenkins acting as the orchestrator.