Jenkins Pipeline Configuration Guide

This guide will walk you through setting up and configuring a Jenkins pipeline that builds, tests, and deploys Docker images to Docker Hub using Kubernetes.

Table of Contents

Installing Jenkins

Jenkins can be installed on Kubernetes using Helm:

```bash
# Add the Jenkins Helm repository if not already added
helm repo add jenkins https://charts.jenkins.io
helm repo update

# Install Jenkins using Helm
helm install my-jenkins jenkinsci/jenkins
```

For more details or custom configurations, refer to: https://artifacthub.io/packages/helm/jenkinsci/jenkins

Accessing Jenkins

Since a LoadBalancer is not available in your environment, you'll need to use port-forwarding to access Jenkins:

```bash
kubectl --namespace default port-forward svc/my-jenkins 8080:8080
```

This will forward your local port 8080 to the Jenkins service's port 8080. Access Jenkins by opening a browser and navigating to `http://localhost:8080`.

Login Credentials

- Username: Admin
- Password: Retrieve using the following command:

```bash
echo Password: $(kubectl get secret --namespace default my-jenkins -o jsonpath="{.data.jenk
```

Installing Required Plugins

After logging in, install the following plugins via "Manage Jenkins" > "Manage Plugins" > "Available":

1. Docker Plugin
2. Kubernetes Plugin (ensure all related Kubernetes plugins are installed)

Setting Up Credentials

Adding Docker Hub Credentials

1. Navigate to "Manage Jenkins" > "Manage Credentials"
2. Click on "Jenkins" under "Stores scoped to Jenkins"
3. Click on "Global credentials (unrestricted)"
4. Click "Add Credentials" and fill in:
   - Kind: Username with password
   - Scope: Global
   - Username: Your Docker Hub username
   - Password: Your Docker Hub password
   - ID: docker-hub-credentials (this must match the ID in your Jenkinsfile)
   - Description: Docker Hub Credentials

Adding GitHub Credentials

1. Generate a GitHub token at: https://github.com/settings/tokens
   - Ensure it has appropriate permissions (repo, workflow)
2. Navigate to "Manage Jenkins" > "Manage Credentials"
3. Click "Add Credentials" and fill in:

- Kind: Username with password
- Scope: Global
- Username: Your GitHub username
- Password: Your GitHub token
- ID: github-credentials
- Description: GitHub Token

Understanding the Jenkinsfile

The Jenkinsfile defines your CI/CD pipeline using declarative syntax. Let's break down each section:

Agent Configuration

groovy

```groovy
agent {
    kubernetes {
        yaml '''
apiVersion: v1
kind: Pod
metadata:
  labels:
    jenkins/my-jenkins-jenkins-agent: "true"
spec:
  containers:
  - name: jnlp
    image: jenkins/inbound-agent:3309.v27b_9314fd1a_4-1
    resources:
      limits:
        memory: "512Mi"
        cpu: "512m"
      requests:
        memory: "512Mi"
        cpu: "512m"
  - name: docker
    image: docker:latest
    command:
    - cat
    tty: true
    volumeMounts:
    - name: docker-sock
      mountPath: /var/run/docker.sock
  - name: kubectl
    image: bitnami/kubectl:latest
    command:
    - cat
    tty: true
  volumes:
  - name: docker-sock
    hostPath:
      path: /var/run/docker.sock
'''
        defaultContainer 'jnlp'
    }
}
```

This section defines a Kubernetes pod with three containers:

- `jnlp`: The Jenkins agent
- `docker`: Container with Docker CLI for building and pushing images
- `kubectl`: Container with kubectl for Kubernetes operations

It also mounts the Docker socket to allow building Docker images.

Environment Variables

```groovy
environment {
    // Define environment variables
    DOCKER_HUB_REPO = 'grinbaum/eran-repo'  // Based on your log, this is the repo you're using
    APP_NAME = 'eran-app'
    NAMESPACE = 'default'
}
```

These variables define:

- `DOCKER_HUB_REPO`: Your Docker Hub repository (username/repository)
- `APP_NAME`: Name of your application
- `NAMESPACE`: Kubernetes namespace for deployment

Pipeline Stages

1. **Checkout**: Fetches your code from the configured SCM (Git)
2. **Build Docker Image**: Builds and tags Docker images from your code
3. **Test**: Runs a simple test within the Docker container
4. **Push to Docker Hub**: Authenticates with Docker Hub and pushes the images

Post Actions

The `post` section handles cleanup and notifications after the pipeline runs:

- On success: Displays a success message
- On failure: Displays a failure message
- Always: Cleans up local Docker images to prevent disk space issues

Creating a Pipeline Job

1. From the Jenkins dashboard, click "New Item"
2. Enter a name for your pipeline
3. Select "Pipeline" and click "OK"
4. In the configuration page:
   - Under "Pipeline" section, select "Pipeline script from SCM"
   - Select "Git" as SCM
   - Enter your repository URL
   - Specify the branch to build (e.g., `*/main`)
   - Set "Script Path" to "Jenkinsfile" (or the path to your Jenkinsfile)
   - Click "Save"

Customizing for Your Docker Hub Account

To use the Jenkinsfile with your Docker Hub account, you need to modify the following:

1. Change the `DOCKER_HUB_REPO` variable to your Docker Hub username and repository name:

```groovy
environment {
    DOCKER_HUB_REPO = 'YOUR_USERNAME/YOUR_REPO'  // Replace with your details
    APP_NAME = 'YOUR_APP_NAME'
    NAMESPACE = 'default'
}
```

For example, if your Docker Hub username is "johndoe" and you want to create a repository called "my-app", change it to:

```groovy
DOCKER_HUB_REPO = 'johndoe/my-app'
```

2. Ensure you've created the Docker Hub credentials in Jenkins with the ID `docker-hub-credentials` as explained in the Setting Up Credentials section.
3. Make sure your repository exists on Docker Hub. If it doesn't exist yet, it will be created when you first push an image to it.

Troubleshooting

Common Issues

1. **Docker build fails**:
   - Check if your Dockerfile is in the root of your repository
   - Verify the Docker daemon is running and accessible
2. **Docker push fails**:
   - Verify your Docker Hub credentials are correct
   - Check if you have permission to push to the repository
3. **Pod creation fails**:
   - Ensure your Kubernetes cluster has enough resources
   - Check if the service account has appropriate permissions
4. **Port forwarding not working**:
   - Verify the Jenkins service is running (`kubectl get svc my-jenkins`)
   - Ensure no other services are using port 8080 on your local machine

Viewing Logs

To diagnose issues, check the Jenkins build logs and Kubernetes pod logs:

```
bash

# Get pod name
kubectl get pods | grep jenkins-agent

# View pod logs
kubectl logs <pod-name> -c <container-name>
```

This guide provides a comprehensive overview of setting up a Jenkins pipeline for Docker image building and deployment. Adapt the configurations as needed for your specific environment and requirements.