Guide: How to Build a CI/CD Pipeline with Dynamic Jenkins Agents and Kubernetes

This guide will walk you through creating a professional-grade CI/CD pipeline. We will configure Jenkins to use **dynamic, on-demand Docker agents** to build and test our application, and then deploy it to a **Kubernetes cluster**.

Prerequisites

Before you begin, you will need:

- An **AWS Account** with permissions to create EC2 instances and security groups.
- A **GitHub Account** to host your application code.
- A **Docker Hub Account** to store your container images.
- A terminal with an SSH client to connect to your servers.

Phase 1: Setting Up the Cloud Infrastructure

In this phase, we'll create and configure our three servers on AWS.

1.1 Launch Your EC2 Instances

1. **Jenkins Controller:**
    - **Instance Type:** `t2.micro` (Free Tier eligible)
    - **AMI:** `Amazon Linux 2`
    - **Security Group:** Allow inbound traffic on `TCP port 22` (SSH) and `TCP port 8080` (for Jenkins).
2. **Docker Host:**
    - **Instance Type:** `t2.micro` (Free Tier eligible)
    - **AMI:** `Amazon Linux 2`
    - **Security Group:** Allow inbound traffic on `TCP port 22` (SSH) and `TCP port 4243` (for Docker).
3. **Kubernetes Node:**
    - **Instance Type:** `t2.medium` (**Important:** Requires at least 2 vCPUs)
    - **AMI:** `Amazon Linux 2`
    - **Security Group:** Allow inbound traffic on `TCP port 22` (SSH). We'll add more ports later.

1.2 Configure the Jenkins Controller

- **On the Jenkins Controller instance:**

```
# Install Java 17 and Jenkins
sudo yum install java-17-amazon-corretto -y
sudo wget -O /etc/yum.repos.d/jenkins.repo
https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-
2023.key
sudo yum install jenkins -y
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

- **Complete the Setup Wizard:** Open `http://<Your-Controller-IP>:8080` in your browser and complete the initial Jenkins setup.

1.3 Configure the Docker Host

This server will launch our dynamic agents. We need to configure its Docker service to accept remote commands from Jenkins.

- **On the Docker Host instance:**

```
# Install Docker
sudo yum install docker -y
sudo systemctl start docker
sudo systemctl enable docker

# Expose Docker over TCP on port 4243
sudo mkdir -p /etc/systemd/system/docker.service.d
sudo tee /etc/systemd/system/docker.service.d/override.conf > /dev/null
<<EOF
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://0.0.0.0:4243
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker

# Fix permissions for Docker-in-Docker setup later
sudo chmod 666 /var/run/docker.sock
```

1.4 Configure the Kubernetes Node

This server will run our deployed application.

- **On the Kubernetes Node instance:**

```
# Install Java, Docker, kubectl, and Minikube
sudo yum install java-17-amazon-corretto docker -y
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -aG docker ec2-user # Log out and log back in after this
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

```
curl -Lo minikube
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-
amd64
chmod +x kubectl minikube
sudo mv kubectl minikube /usr/local/bin/

# Start the Kubernetes cluster
minikube start --driver=docker
```

Phase 2: Jenkins Configuration

Now we'll configure Jenkins to manage our infrastructure.

2.1 Connect the Kubernetes Node as a Static Agent

The deployment job will run on this static agent for reliability.

1. In Jenkins, go to **Manage Jenkins > Nodes > New Node**.
2. Create a **Permanent Agent** named `K8s-Agent`.
3. Configure it to connect via **SSH** using your `.pem` key.
4. Give it the label `kubernetes`.
5. Save and wait for it to come online.

2.2 Configure Jenkins for Dynamic Agents

1. **Install the Docker Plugin:** In Jenkins, go to **Manage Jenkins > Plugins > Available plugins** and install the **"Docker"** plugin.
2. **Configure the Docker Cloud:**
    - Go to **Manage Jenkins > Clouds > + Add a new cloud** and select **Docker**.
    - **Name:** `my-docker-host`
    - **Docker Host URI:** `tcp://<Private_IP_of_Docker_Host>:4243`
    - Click **Test Connection**. You should see a success message.
3. **Create the Dynamic Agent Template:**
    - On the same page, click **"Docker Agent templates..." > Add Docker Template**.
    - **Labels:** `docker-agent`
    - **Docker Image:** You will need to create and push a custom image with Docker and kubectl installed.
    - **Connect method:** "Attach Docker container".
    - **Container settings... > Mounts:** Add `type=bind,source=/var/run/docker.sock,target=/var/run/docker.sock`. This is for the Docker-in-Docker functionality.
    - Click **Save**.

Phase 3: Creating the Pipeline Jobs

Create three separate Freestyle jobs for each stage of the pipeline.

1. **`K8s-Build-Job`**:
    - **Restrict to run on label:** `docker-agent`
    - **SCM:** Your GitHub repo.
    - **Build Environment:** Bind your Docker Hub credentials.
    - **Execute Shell:** Use the `build.sh` script to build and push the application image.
    - **Post-build Action:** Trigger the `K8s-Test-Job`.
2. **`K8s-Test-Job`**:
    - **Restrict to run on label:** `docker-agent`
    - Make it **parameterized** to accept the `IMAGE_TAG`.
    - **Build Environment:** Bind your Docker Hub credentials.
    - **Execute Shell:** Use the `test.sh` script to run a smoke test on the container.
    - **Post-build Action:** Trigger the `K8s-Deploy-Job` on success.
3. **`K8s-Deploy-Job`**:
    - **Restrict to run on label:** `kubernetes`
    - **Execute Shell:** Use the `deploy.sh` script to run `kubectl apply` and `kubectl rollout restart`.

Phase 4: Final Automation & Verification

1. **Configure GitHub Webhook:** In your GitHub repo's settings, add a webhook pointing to `http://<Your-Controller-Public-IP>:8080/github-webhook/`.
2. **Expose the Application:**
    - SSH into your `K8s-Node`.
    - Run `kubectl port-forward deployment/my-web-app 8080:80 --address='0.0.0.0'` in a separate, continuous terminal session.
    - Update the `K8s-Node`'s Security Group to allow inbound traffic on `TCP port 8080`.
3. **Verify:** Access your application at `http://<Your-K8s-Node-Public-IP>:8080`.

You now have a fully functional, efficient, and professional-grade CI/CD pipeline!