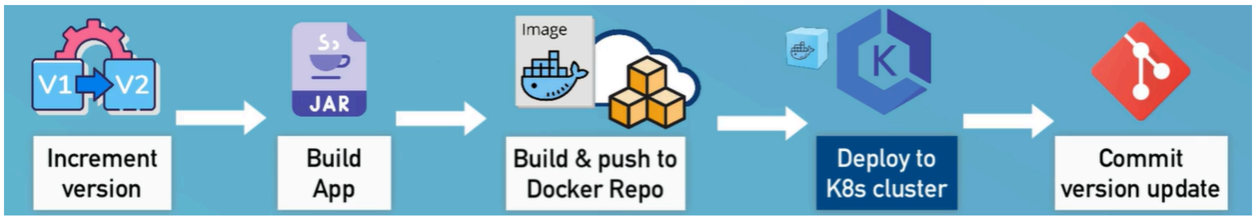
Demo Project: Complete CI/CD Pipeline with EKS and private DockerHub registry

This guide outlines how to build the Java Maven application, push the Docker image to the private DockerHub registry, and deploy the updated application to an AWS EKS cluster via a Jenkins pipeline.



Step 1: Create Kubernetes Manifests for the Application

1. **Deployment Manifest (`kubernetes/deployment.yaml`):**

   Create a file (or update an existing one) with the following content. This template uses environment variable substitution via `envsubst`:

   ```
   apiVersion: apps/v1
   kind: Deployment
   ```

```
metadata:
  name: $APP_NAME
  labels:
    app: $APP_NAME
spec:
  replicas: 2
  selector:
    matchLabels:
      app: $APP_NAME
  template:
    metadata:
      labels:
        app: $APP_NAME
    spec:
      containers:
        - name: $APP_NAME
          image: xxx
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
```

2. **Service Manifest (`kubernetes/service.yaml`):**

   Create or update the service file with:

```
apiVersion: v1
kind: Service
metadata:
  name: $APP_NAME
spec:
  selector:
    app: $APP_NAME
  ports:
    - protocol: TCP
    port: 80
    targetPort: 8080
```

Step 2: Adjust the Jenkinsfile to Deploy to EKS

1. **Checkout the Appropriate Branch:**
   - From your repository (e.g., `java-maven-app`), checkout the branch that contains the Jenkins pipeline configuration: `git checkout jenkins-jobs`
2. **Update the Jenkinsfile's Deploy Stage:**

   Add the following deploy stage to your Jenkinsfile:

```
...
stage('deploy') {
    environment {
        AWS_ACCESS_KEY_ID = credentials('jenkins_aws_access_key_id')
        AWS_SECRET_ACCESS_KEY = credentials('jenkins-aws_secret_access_key')
        APP_NAME = 'java-maven-app'
    }
    steps {
        script {
            echo 'deploying docker image...'
            sh 'envsubst < kubernetes/deployment.yaml | kubectl apply -f -'
            sh 'envsubst < kubernetes/service.yaml | kubectl apply -f -'
        }
    }
}
```

Note: To use `envsubst`, ensure that the `gettext-base` package is installed on your Jenkins server (see Step 3).

Step 3: Install “gettext-base” tool (for `envsubst`) on the Jenkins Server

1. **SSH into the Jenkins Server:** `ssh root@<jenkins-ip>`
2. **Identify the Jenkins Docker Container:** `docker ps`
3. **Enter the Jenkins Container as Root:** `docker exec -u 0 -it <container-id> bash`
4. **Install `gettext-base`:** `apt-get update && apt-get install -y gettext-base`
5. **Verify Installation:** `envsubst --version`
6. **Exit the Container:** `exit`

Step 4: Create Secret for DockerHub credentials

For Kubernetes to be able to fetch the image from a private repo from DockerHub, we need authentication from inside the Kubernetes cluster.

Important: Ensure that the secret is created in the same namespace where your application will be deployed (or create it in the appropriate namespace).

1. **From Your Local Machine, Create the Secret:**

```
kubectl create secret docker-registry my-registry-key \
--docker-server=docker.io \
--docker-username=eduardobautistamaciel \
--docker-password=<password>
```

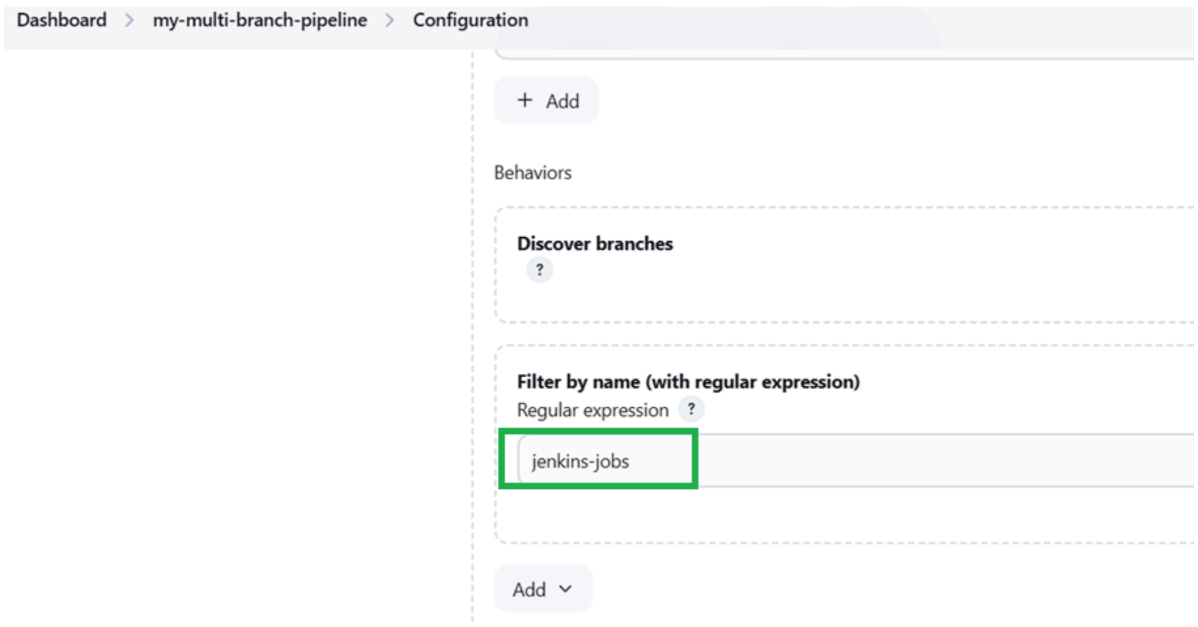
2. **Verify the secret:** `kubectl get secret`
3. In the **Deployment Manifest (`kubernetes/deployment.yaml`): add the secret for example:**

```
spec:
  imagePullSecrets:
  - name: my-registry-key
```

4. **Commit and Push Changes:**
   - Commit your updated Jenkinsfile to the `jenkins-jobs` branch and push it to your repository.

Step 5: Execute Jenkins Pipeline

1. **Configure the Multibranch Pipeline in Jenkins:**
   - Update the pipeline configuration to trigger only the branch jenkins-jobs:



Dashboard > my-multi-branch-pipeline > Configuration

Behaviors

Discover branches

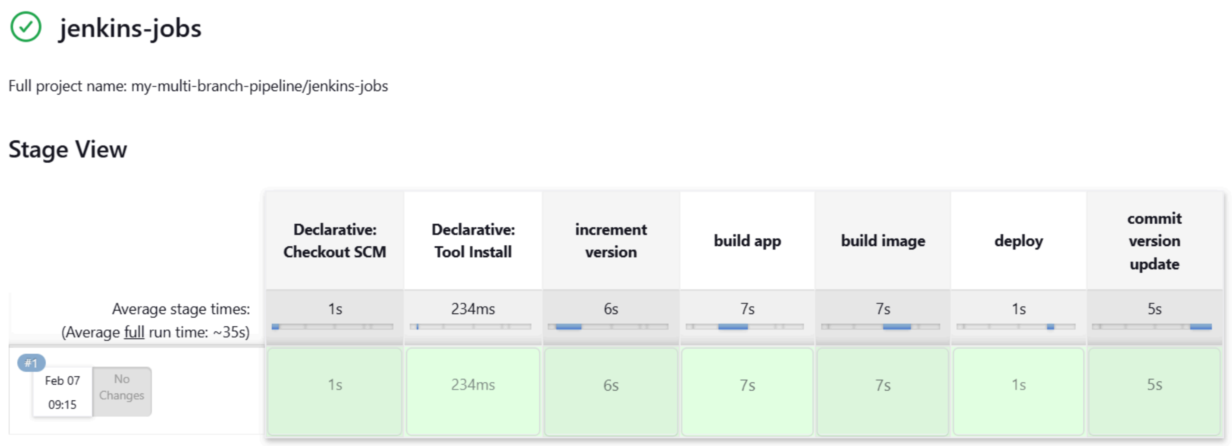
Filter by name (with regular expression)

Regular expression: jenkins-jobs

   - Save the changes in Jenkins.

2. **Trigger the Pipeline:**
   - Confirm that no pods are running: `kubectl get pod`
   - Manually trigger the pipeline from the Jenkins UI.

- Monitor the console output to ensure that all stages (build, image build, deploy) complete successfully.



jenkins-jobs

Full project name: my-multi-branch-pipeline/jenkins-jobs

Stage View

	Declarative: Checkout SCM	Declarative: Tool Install	increment version	build app	build image	deploy	commit version update
Average stage times: (Average full run time: ~35s)	1s	234ms	6s	7s	7s	1s	5s
#1 Feb 07 09:15 No Changes	1s	234ms	6s	7s	7s	1s	5s

3. **Verify Deployment on EKS:**
   - Use the following command to check the 2 pods in the EKS cluster: `kubectl get pod`
   - Confirm that the deployment is running and using the correct image.: `kubectl describe pod <pod name>`
   - Confirm the deployment created: `kubectl get deployment` "java-maven-app" is shown
   - Confirm the services created: `kubectl get service`

Step 6: Clean Up

- When finished, you can clean up your resources:

1. **Delete the Deployment** (if needed): `kubectl delete deployment <deployment-name>`
2. **Optionally, Delete the EKS Cluster**: `eksctl delete cluster --name <your-cluster-name> --region <your-region>`
