Demo Protect: CD - Deploy Application from Jenkins Pipeline to EC2 Instance (automatically with docker)

This guide demonstrates how to:

1. Prepare an AWS EC2 instance for deployment.
2. Configure Jenkins to connect to the EC2 instance using SSH.
3. Extend a Jenkins CI pipeline to deploy a Docker image on a remote EC2 instance.
4. Verify the deployment and ensure the application is publicly accessible.

Step 1: Install SSH Agent Plugin and Create SSH Credentials

1. Install SSH Agent Plugin:

1. Go to **Jenkins Dashboard → Manage Jenkins → Plugins → Available Plugins**.
2. Search for **SSH Agent** and install it.
    - **Purpose**: This plugin allows Jenkins to use SSH credentials for securely accessing remote servers.

2. Add SSH Credentials for EC2 Instance:

1. Go to the **Multibranch Pipeline → Credentials → Add Credentials**.
    - **Note**: Multibranch pipeline credentials are scoped to this specific pipeline. Under **Stores scoped to aws-multibranch-pipeline**
2. Configure the following:
    - **Kind**: `SSH Username with Private Key`.
    - **Username**: `ec2-user`.
    - Go to **Private Key → Enter directly → Add**: Copy the contents of your `.pem` file:

      `cat ~/.ssh/docker-server.pem`
    - **ID**: `ec2-ssh-key`.

Step 2: Add SSH Agent Configuration to Jenkinsfile

1. Generate SSH Pipeline Syntax:

1. Navigate to **Multibranch Pipeline → Pipeline Syntax → SSH Agent**.
2. Configure the SSH Agent and click **Generate Pipeline Script**.

2. Switch to the Appropriate Branch:

- Use the `feature/payment` branch for initial testing. This branch contains a simplified Jenkinsfile.

3. Update the Jenkinsfile:

1. Add the following snippet to test SSH connectivity:

```
#!/usr/bin.env groovy

pipeline {
    agent any
    stages {
        stage("test") {
            steps {
                script {
                    echo "Testing the application..."

                }
            }
        }
        stage("build") {
            steps {
                script {
                    echo "Building the application..."
                }
            }
        }
        stage("deploy") {
            steps {
                script {
                    def dockerCmd = 'docker run -p 3080:3080 -d eduardobautistamaciel/demo-
app:1.0'
                    sshagent(['ec2-server-key']) {
                       sh "ssh -o StrictHostKeyChecking=no ec2-user@52.71.72.116 ${dockerCmd}"
                    }
                }
            }
        }
    }
}
```

   - Note: **Suppressing SSH Prompts**: The option `o StrictHostKeyChecking=no` bypasses the SSH fingerprint confirmation to prevent build interruptions.

2. Commit and push the updated Jenkinsfile to the repository.

Step 3: Test SSH Agent Plugin

1. Prepare EC2 for Testing:

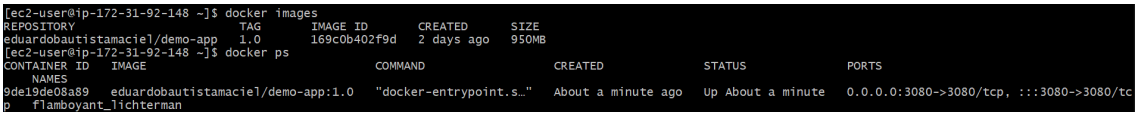
- Ensure Docker is installed and running on the EC2 instance.
- Log in to the EC2 instance:
  - `ssh -i ~/.ssh/docker-server.pem ec2-user@<public-ip-address>`
  - `docker login -u <username> -p <password>`

2. Update AWS Security Group:

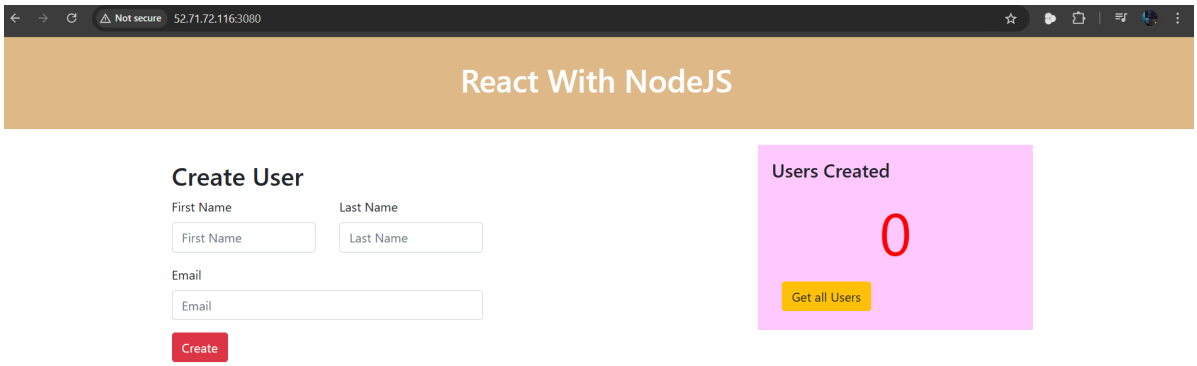
- Allow SSH access from the Jenkins server IP address on port 22.

3. Run the Pipeline:

1. **Trigger the pipeline** in Jenkin:
   - Go to the Jenkins Multibranch Pipeline dashboard.
   - Select the `feature/payment` branch and click **Build Now**.
2. Monitor the pipeline and output logs.
3. SSH into the EC2 instance and verify:
   - Docker image was pulled: `docker images`



   - Container is running: `docker ps`
4. Access the application: `http://<ec2-public-ip>:3080`



Step 4: Executing Complete Pipeline

1. Switch to the Appropriate Branch:

- Use the `jenkins-jobs` branch. This branch contains the Jenkins Shared Library for Maven build and Docker image creation.
- **Note**: The Java Maven application will run on port 8080.

2. Updated Jenkinsfile:

Include deployment commands:

```
def dockerCmd = 'docker run -p 8080:8080 -d eduardobautistamaciel/demo-app:1.0'
```

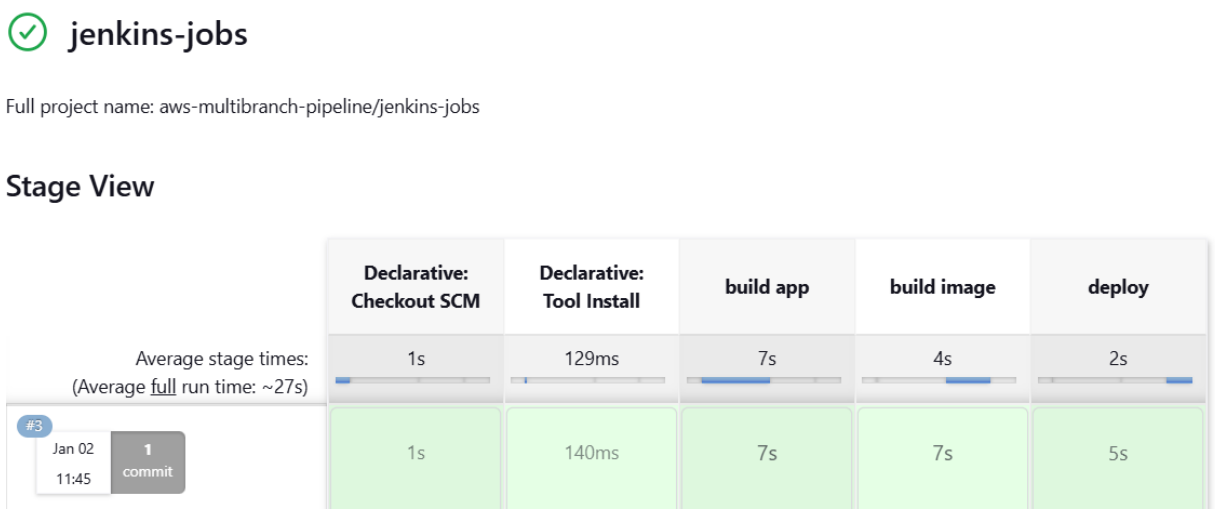
```groovy
#!/usr/bin/env groovy

library identifier: 'jenkins-shared-library@main', retriever: modernSCM([
    $class: 'GitSCMSource',
    remote: 'https://gitlab.com/twn-devops-projects/aws/jenkins-shared-library',
    credentialsId: 'gitlab-credentiales'
])


pipeline {
    agent any
    tools {
        maven 'maven-3.9'
    }
    environment {
        IMAGE_NAME = 'eduardobautistamaciel/demo-app:java-maven-1.0'
    }
    stages {
        stage('build app') {
            steps {
                echo 'building application jar...'
                buildJar()
            }
        }
        stage('build image') {
            steps {
                script {
                    echo 'building the docker image...'
                    buildImage(env.IMAGE_NAME)
                    dockerLogin()
                    dockerPush(env.IMAGE_NAME)
                }
            }
        }
        stage("deploy") {
            steps {
                script {
                    def dockerCmd = "docker run -p 8080:8080 -d ${IMAGE_NAME}"
                       sshagent(['ec2-server-key']) {
                       sh "ssh -o StrictHostKeyChecking=no ec2-user@52.71.72.116 ${dockerCmd}"
                    }
                }
            }
        }
    }
}
```

3. Run the Jenkins Pipeline:

- **Trigger the pipeline**:
  - Go to the Jenkins Multibranch Pipeline dashboard.
  - Select the `jenkins-jobs` branch and click **Build Now**.

3. Confirm pipeline and Verify Deployment on EC2

1. Confirm successful pipeline and output logs in Jenkins.



2. Verify Deployment on EC2

- **Confirm the Docker Image**:
  - List the Docker images to confirm the application image is present: `docker images`

    Example Output:

```
[ec2-user@ip-172-31-92-148 ~]$ docker images
REPOSITORY                          TAG              IMAGE ID       CREATED
 SIZE
eduardobautistamaciel/demo-app      java-maven-1.0   623fd49f7079   8 minutes ago
 150MB
eduardobautistamaciel/demo-app      1.0              169c0b402f9d   2 days ago
 950MB
```

- **Verify the Running Container**:
  - Check the running containers to confirm the application is live: `docker ps`

    Example Output:

```
[ec2-user@ip-172-31-92-148 ~]$ docker ps
CONTAINER ID   IMAGE                                    COMMAND                  CREATED          STATUS          PORTS
     NAMES
2355b341e1b2   eduardobautistamaciel/demo-app:java-maven-1.0   "/bin/sh -c 'java -j…"   4 minutes ago    Up 3 minutes    0.0.0.0:8080->8080/tcp, :::8080->8080/t
cp   wizardly_pasteur
9de19de08a89   eduardobautistamaciel/demo-app:1.0              "docker-entrypoint.s…"   3 hours ago      Up 3 hours      0.0.0.0:3080->3080/tcp, :::3080->3080/t
cp   flamboyant_lichterman
```