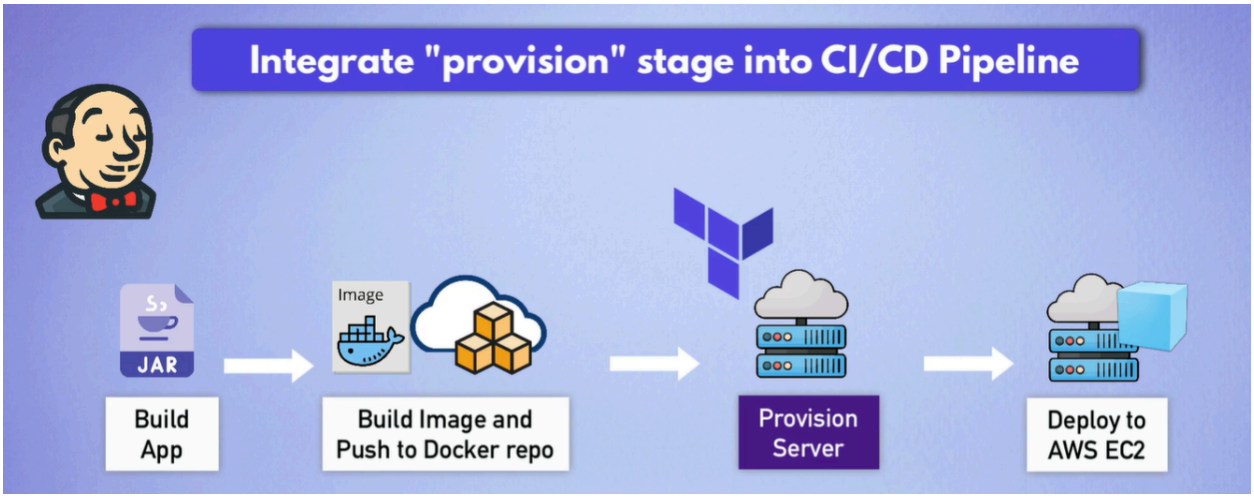
Demo Project: Complete CI/CD with Terraform

This guide outlines how to integrate a provisioning stage into a CI/CD pipeline to automate server provisioning and deployment using Jenkins and Terraform.

Stages

- **CI Step:** Build the artifact for a Java Maven application.
- **CI Step:** Build and push a Docker image.
- **CD Step:** Provision an EC2 instance using Terraform.
- **CD Step:** Deploy the application using Docker Compose.

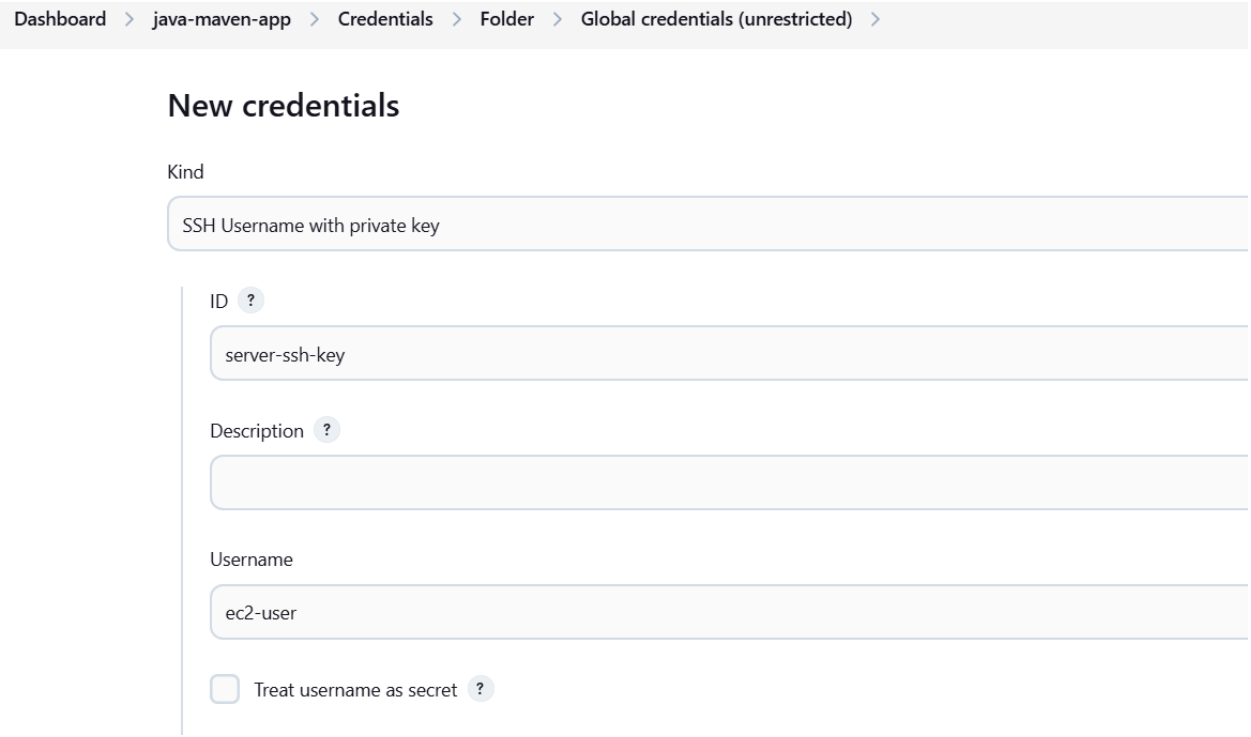


Troubleshooting

Issue: `wget: command not found`

Step 1: Create SSH key-pair

1. Navigate to the AWS EC2 Key Pair service.
2. Create a new key pair named `myapp-key-pair`.
3. Download the `.pem` file.
4. Add the key to Jenkins:
    - Open the Jenkins UI.
    - Go to **Manage Jenkins** > **Credentials**.
    - Select **Kind:** SSH Username with private key.
    - Username: `ec2-user`.
    - Private Key: Paste the content from `cat myapp-key-pair.pem`.



Step 2: Install Terraform in Jenkins container

1. SSH into the Jenkins host: `ssh root@<host-ip>`
2. Access the Jenkins container as root: `docker exec -it -u 0 <container-id> bash`
3. Check the Operating system being used: `cat /etc/os-release`

   Example:

   ```
   PRETTY_NAME="Debian GNU/Linux 12 (bookworm)"
   NAME="Debian GNU/Linux"
   VERSION_ID="12"
   VERSION="12 (bookworm)"
   VERSION_CODENAME=bookworm
   ID=debian
   HOME_URL="https://www.debian.org/"
   SUPPORT_URL="https://www.debian.org/support"
   BUG_REPORT_URL="https://bugs.debian.org/"
   ```

4. Install Terraform

   Download Terraform from https://developer.hashicorp.com/terraform/install based on your operating system.

   Remeber not to use sude since we are login as root user.

   - `wget -O - https://apt.releases.hashicorp.com/gpg | gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg`
   - `echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(grep VERSION_CODENAME /etc/os-release | cut -d= -f2) main" | tee /etc/apt/sources.list.d/hashicorp.list`
   - `apt update && apt install terraform`

5. Verify instalattion: `terraform -v`

Step 3: Add Terraform Configuration to Application Repository

1. In your application's Git repository, create a `terraform/` directory.
2. Add `main.tf` with the Terraform configuration.
3. Create a `variables.tf` file in the same directory to define variables.

Step 4: Add Provision Stage to Jenkinsfile

Add a new stage to the Jenkinsfile:

```
stage("provision server") {
  environment {
    AWS_ACCESS_KEY_ID = credentials('jenkins_aws_access_key_id')
    AWS_SECRET_ACCESS_KEY = credentials('jenkins-aws_secret_access_key'
    TF_VAR_env_prefix = 'test'// Override variable
  }
  steps {
    script {
      dir('terraform') {
        sh "terraform init"
        sh "terraform apply --auto-approve"
        EC2_PUBLIC_IP = sh(
          script: "terraform output ec2-public_ip",
          returnStdout: true
        ).trim()
      }
    }
  }
}
```

Step 5: Add Deploy Stage to Jenkinsfile

Ensure the deployment stage dynamically retrieves the EC2 public IP. We don't want to hard-code the EC2 public IP since Terraform outputs it into an environment variable, `EC2_PUBLIC_IP`. Access it as shown below:

```
stage("deploy") {
  environment {
    DOCKER_CREDS = credentials('docker-hub-repo')
  }
  steps {
    script {
      echo "waiting for EC2 server to initialize"
      sleep(time: 90, unit: "SECONDS")

      echo 'deploying docker image to EC2...'
      echo "${EC2_PUBLIC_IP}"

      def shellCmd = "bash ./server-cmds.sh ${IMAGE_NAME} ${DOCKER_CRED
      def ec2Instance = "ec2-user@${EC2_PUBLIC_IP}"

      sshagent(['server-ssh-key']) {
        sh "scp -o StrictHostKeyChecking=no server-cmds.sh ${ec2Instance}:/h
        sh "scp -o StrictHostKeyChecking=no docker-compose.yaml ${ec2Instar
        sh "ssh -o StrictHostKeyChecking=no ${ec2Instance} ${shellCmd}"
      }
    }
  }
```

Step 6: Create `server-cmds.sh`

This script will log in to Docker and start the application. The script takes three arguments:

`IMAGE_NAME`, `DOCKER_CREDS_USR`, and `DOCKER_CREDS_PSW`

```
#!/usr/bin/env bash

export IMAGE=$1
export DOCKER_USER=$2
export DOCKER_PWD=$3
echo $DOCKER_PWD | docker login -u $DOCKER_USER --password-stdin
docker-compose -f docker-compose.yaml up --detach
echo "success"
```

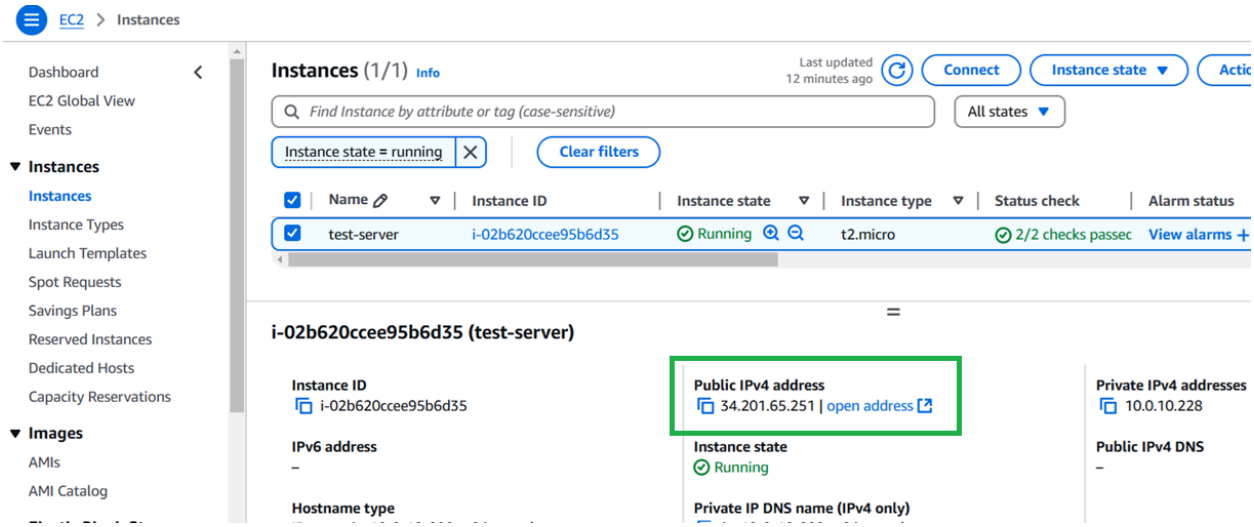
In Jenkinsfile:

```
...
def shellCmd = "bash ./server-cmds.sh ${IMAGE_NAME} ${DOCKER_CREDS_US
...
```

Step 7: Run CI/CD Pipeline

1. Commit and push changes to the Git repository.
2. Check the Jenkins pipeline and ensure all stages complete successfully.

3. Retrieve the public IP of the provisioned EC2 instance:



4. SSH into the instance:

```
chmod 400 myapp-key-pair.pem
ssh -i /path/to/myapp-key-pair.pem ec2-user@<ec2-public-ip>
```

5. Verify the running containers: `docker ps`

   Example output:

```
[ec2-user@ip-10-0-10-228 ~]$ docker ps
CONTAINER ID   IMAGE                                   COMMAND                  CREATED
844f5b915927   postgres:15                              "docker-entrypoint.s..."   12 mi
e70fbe389cbf   eduardobautistamaciel/demo-app:java-maven-2.0   "/bin/sh -c 'j
```

```
$ ssh -i myapp-key-pair.pem ec2-user@34.201.65.251
The authenticity of host '34.201.65.251 (34.201.65.251)' can't be established.
ECDSA key fingerprint is SHA256:4dH+EkzL6lBxHWcAgJvRTA10NUMFbvqhD9CVrDeu/U4.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '34.201.65.251' (ECDSA) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2026-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-10-0-10-228 ~]$ docker ps
CONTAINER ID   IMAGE                                          COMMAND                  CREATED
 STATUS          PORTS                                       NAMES
844f5b915927   postgres:15                                     "docker-entrypoint.s…"   12 minutes ago
  Up 12 minutes   0.0.0.0:5432->5432/tcp, :::5432->5432/tcp   ec2-user-postgres-1
e70fbe389cbf   eduardobautistamaciel/demo-app:java-maven-2.0   "/bin/sh -c 'java -j…"   12 minutes ago
  Up 12 minutes   0.0.0.0:8080->8080/tcp, :::8080->8080/tcp   ec2-user-java-maven-app-1
```

Step 8: Clean Up Resources

1. Navigate to the Terraform directory: `cd /path/to/terraform`
2. Destroy the provisioned resources:
   - `terraform init`
   - `terraform destroy --auto-approve`

Troubleshooting

Issue: `wget: command not found`

Reason: `wget` is not installed.

Solution: Install `wget`:

```
sudo apt update
sudo apt install wget
```

Note: Remove `sudo` if you are the root user.