Terraform Configuration

```
provider "aws" {
 region = "us-east-1"
}

resource "aws_instance" "demo-server" {
 ami           = "ami-053b0d53c279acc90"
 instance_type = "t2.micro"
 key_name      = "dpp"
 //security_groups = [ "demo-sg" ]
 vpc_security_group_ids = [aws_security_group.demo-sg.id]
 subnet_id              = aws_subnet.dpp-public-subnet-01.id
 for_each               = toset(["jenkins-master", "build-slave",
"ansible"])
 tags = {
   Name = "${each.key}"
 }
}

resource "aws_security_group" "demo-sg" {
 name        = "demo-sg"
 description = "SSH Access"
 vpc_id      = aws_vpc.dpp-vpc.id

 ingress {
   description = "SHH access"
   from_port   = 22
   to_port     = 22
   portocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
 }

 ingress {
   description = "Jenkins port"
```

```
   from_port   = 8080
   to_port     = 8080
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
 }

 egress {
   from_port        = 0
   to_port          = 0
   protocol         = "-1"
   cidr_blocks      = ["0.0.0.0/0"]
   ipv6_cidr_blocks = ["::/0"]
 }

 tags = {
   Name = "ssh-port"

 }
}

resource "aws_vpc" "dpp-vpc" {
 cidr_block = "10.1.0.0/16"
 tags = {
   Name = "dpp-vpc"
 }

}

resource "aws_subnet" "dpp-public-subnet-01" {
 vpc_id                  = aws_vpc.dpp-vpc.id
 cidr_block              = "10.1.1.0/24"
 map_public_ip_on_launch = "true"
 availability_zone       = "us-east-1a"
 tags = {
   Name = "dpp-public-subent-01"
 }
```

```
}

resource "aws_subnet" "dpp-public-subnet-02" {
 vpc_id                  = aws_vpc.dpp-vpc.id
 cidr_block              = "10.1.2.0/24"
 map_public_ip_on_launch = "true"
 availability_zone       = "us-east-1b"
 tags = {
   Name = "dpp-public-subent-02"
 }
}

resource "aws_internet_gateway" "dpp-igw" {
 vpc_id = aws_vpc.dpp-vpc.id
 tags = {
   Name = "dpp-igw"
 }
}

resource "aws_route_table" "dpp-public-rt" {
 vpc_id = aws_vpc.dpp-vpc.id
 route {
   cidr_block = "0.0.0.0/0"
   gateway_id = aws_internet_gateway.dpp-igw.id
 }
}

resource "aws_route_table_association" "dpp-rta-public-subnet-01" {
 subnet_id      = aws_subnet.dpp-public-subnet-01.id
 route_table_id = aws_route_table.dpp-public-rt.id
}

resource "aws_route_table_association" "dpp-rta-public-subnet-02" {
 subnet_id      = aws_subnet.dpp-public-subnet-02.id
 route_table_id = aws_route_table.dpp-public-rt.id
}
```

Ansible Configuration

1. SSH onto ansible machine and install ansible:

   ```
   sudo apt update
   sudo apt install software-properties-common
   sudo add-apt-repository --yes --update ppa:ansible/ansible
   sudo apt install ansible
   ```

2. scp key to ansible machine:

   scp -i dpp.pem dpp.pem ubuntu@34.229.208.50:/home/ubuntu/

   Place key in /opt/ directory

   Note: Ensure permissions on key does not have write permissions i.e. use chmod 400.

3. Create hosts file in /opt directory for jenkins master and slave machines using private IPs.

   ```
   [jenkins-master]
   10.1.1.189

   [jenkins-master:vars]
   ansible_user=ubuntu
   ansible_ssh_private_key_file=/opt/dpp.pem

   [jenkins-slave]
   10.1.1.105

   [jenkins-slave:vars]
   ansible_user=ubuntu
   ansible_ssh_private_key_file=/opt/dpp.pem
   ```

4. Check connectivity from ansible machine to jenkins master and slave:
   ansible all -i hosts -m ping

5. **Copy the following file jenkins-master-setup.yaml config to /opt directory:**

```
---
```

```yaml
- hosts: jenkins-master
 become: true
 tasks:
 - name: add jenkins key
   apt_key:
     url:  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
     state: present

 - name: add jenkins repo
   apt_repository:
     repo: 'deb https://pkg.jenkins.io/debian-stable binary/'
     state: present

 - name: install java
   apt:
     name: openjdk-11-jre
     state: present

 - name: install jenkins
   apt:
     name: jenkins
     state: present

 - name: start jenkins service
   service:
     name: jenkins
     state: started

 - name: enable jenkins to start at boot time
   service:
     name: jenkins
     enabled: yes
```

Run the playbook to install Jenkins and Java on the jenkins master machine:

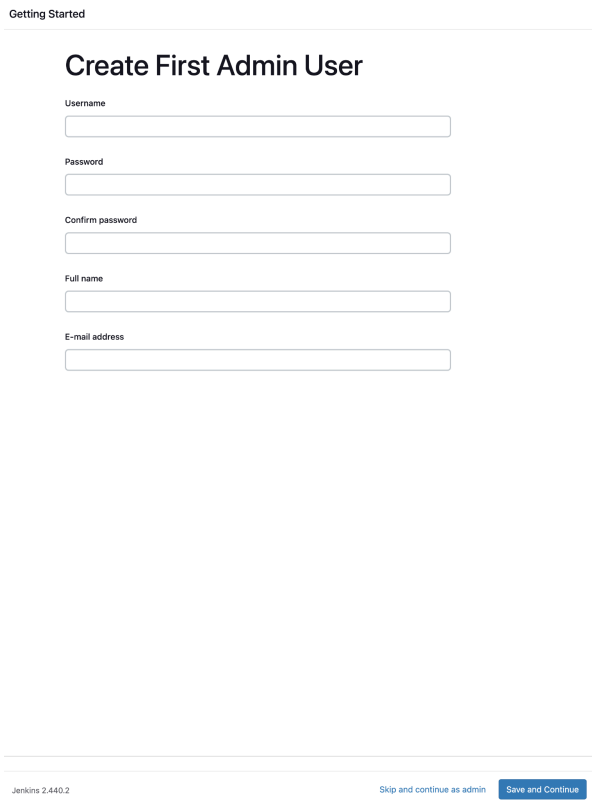
ansible-playbook -i /opt/hosts jenkins-master-setup.yaml

Go to http://52.91.238.126:8080/ and input initialAdminPassword located /var/lib/jenkins/secrets

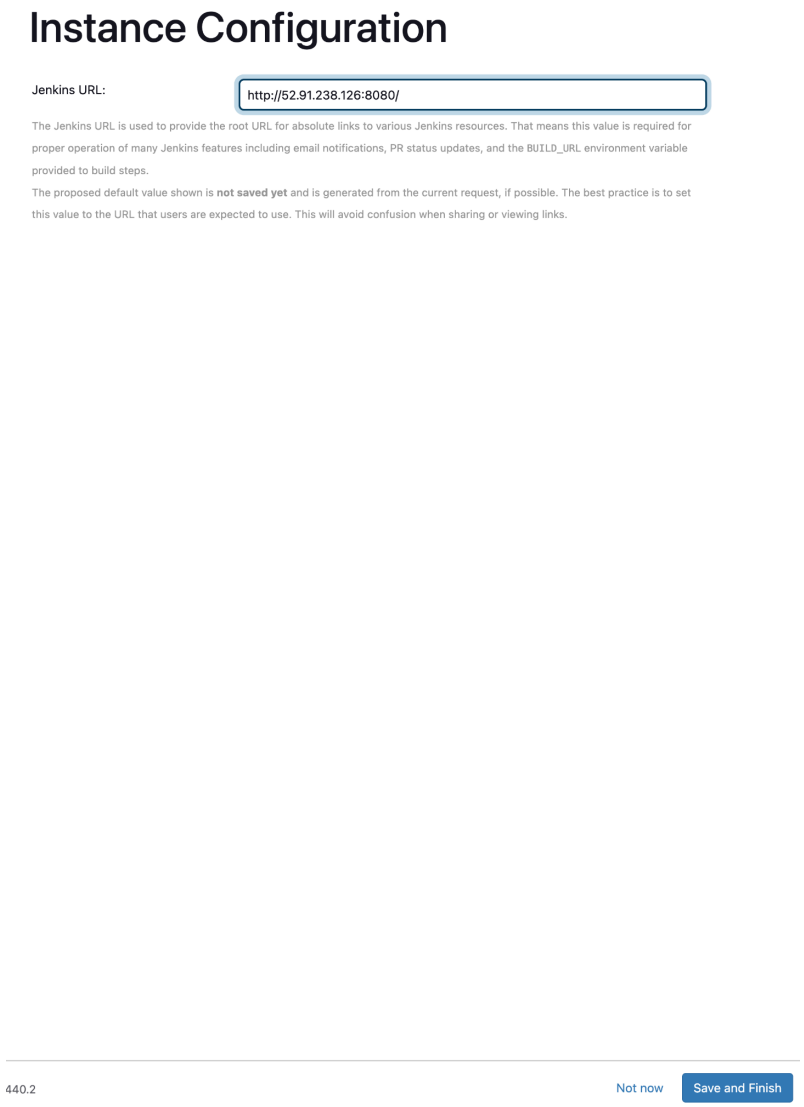
dc98e6c2417a4231a1eb7542ee0a9f77

Select “Install suggested plugins”

We don’t need to create a new user so we will skip this step and simply use the existing Admin user.



Under “Instance Configuration” page select Not Now since this is a public IP so will constantly keep on changing.



6. Copy the following into a file called jenkins-slave-setup.yaml and place it in the /opt directory on the ansible server.

```yaml
---
- hosts: jenkins-slave
 become: true
 tasks:
 - name: update ubuntu repo and cache
   apt:
     update_cache: yes
     cache_valid_time: 3600
```

```
 - name: install java
   apt:
     name: openjdk-11-jre
     state: present

 - name: download maven packages
   get_url:
     url:
https://dlcdn.apache.org/maven/maven-3/3.9.6/binaries/apache-maven-3.9.6-bin.tar.gz
     dest: /opt

 - name: extract maven packages
   unarchive:
     src: /opt/apache-maven-3.9.2-bin.tar.gz
     dest: /opt
     remote_src: yes
```

This will install Java and maven on the Jenkins slave machine. Note: We are using the latest version of Maven at the time of writing this v3.9.6 so you may need to change the version in the ansible script if this has changed by going to https://maven.apache.org/download.cgi

Run the ansible playbook on the ansible machine:

ansible-playbook -i /opt/hosts jenkins-slave-setup.yaml

Check maven is correctly installed by going to directory /opt/apache-maven-3.9.6/bin and run the command:

./mvn --version

7. Configuring Jenkins slave and master

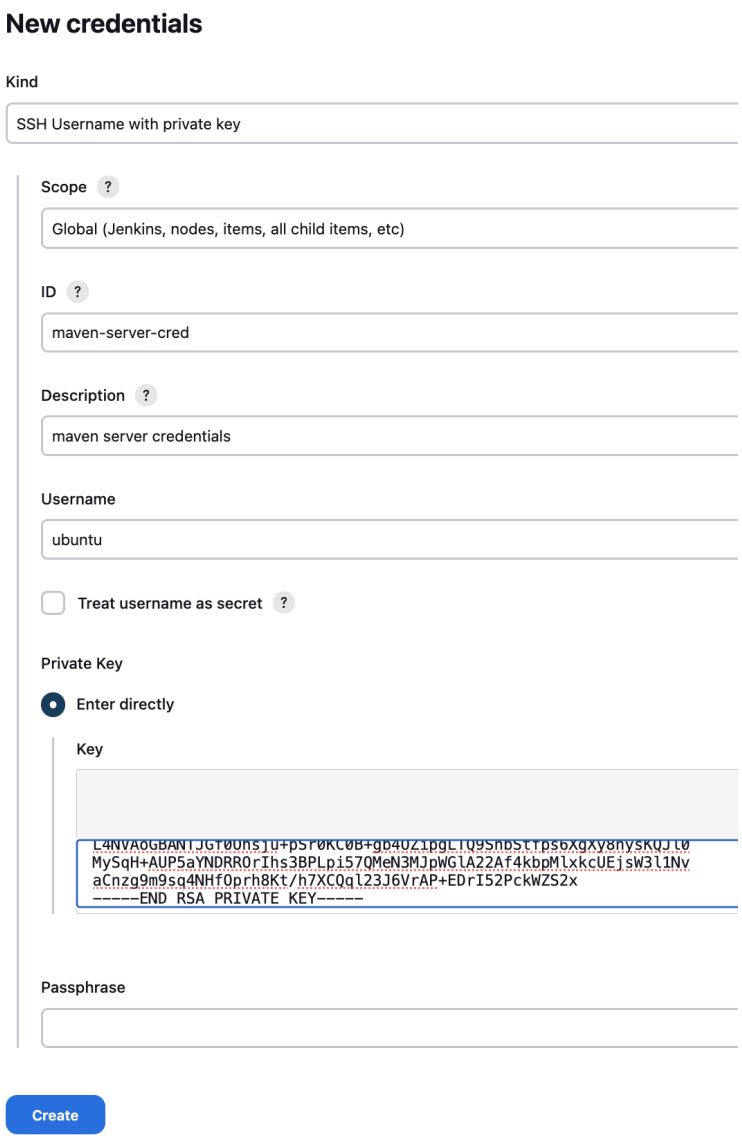
On jenkins master change admin password to something simple e.g. admin123

We need to add credentials so that master can communicate with slave.

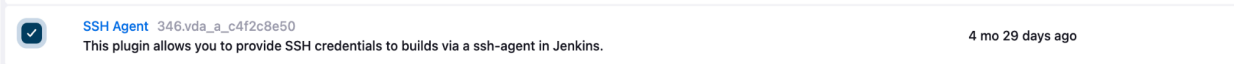
Add credentials by going to Dashboard -> Manage Jenkins -> Security -> Credentials -> System -> Global credentials (unrestricted) -> Add credentials

```
Under New credentials enter the following:
Kind: SSH Username with private key
Scope: Global
ID:maven-server-cred
Description: Maven server credentials
Username: ubuntu
Private: Enter dpp.pem private key
Pass phrase: N/A
```

See screenshot:



8. **Install SSH Agent plugin**



Now we need to add the slave as a node:

Dashboard -> Manage Jenkins -> Nodes -> New node

Node name: maven-slave
Permanent agent: yes

Enter the following:
Number of executors: 3
Remote root directory: /home/ubuntu/jenkins
Labels: maven
Usage: Use this node as much as possible
Launch method: Launch agents via SSH
Host: <Private_IP_of_Slave>
Credentials: <Jenkins_Slave_Credentials>
Host Key Verification Strategy: Non verifying Verification Strategy
Availability: Keep this agent online as much as possible

Note: Test jenkins slave is working properly by creating a small jenkins job which simply writes a text file using shell.

9. **Clone app code repo and create Jenskins file and push to app repo**

Add following Jenkinsfile file to source code repo:

```
pipeline {
   agent {
       node {
           label 'maven'
       }
   }

   stages {
       stage('Clone code') {
           steps {
               git branch: 'main', url:
'https://github.com/rachrafi/tweet_trend_new.git'
```

```
            }
        }
    }
}
```

10. **Create personal access token in GitHub then create new Jenkins credentials using the token.**

    Settings → Developer Settings → Personal access tokens → Tokens (classic)

    Generate token and give it full privileges.

    In Jenkins create new credentials using the token as follows:

New credentials

Kind

Username with password

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Username ?

rachrafi

Treat username as secret ?

Password ?

••••••••••••••••••••••••••••••••••••••

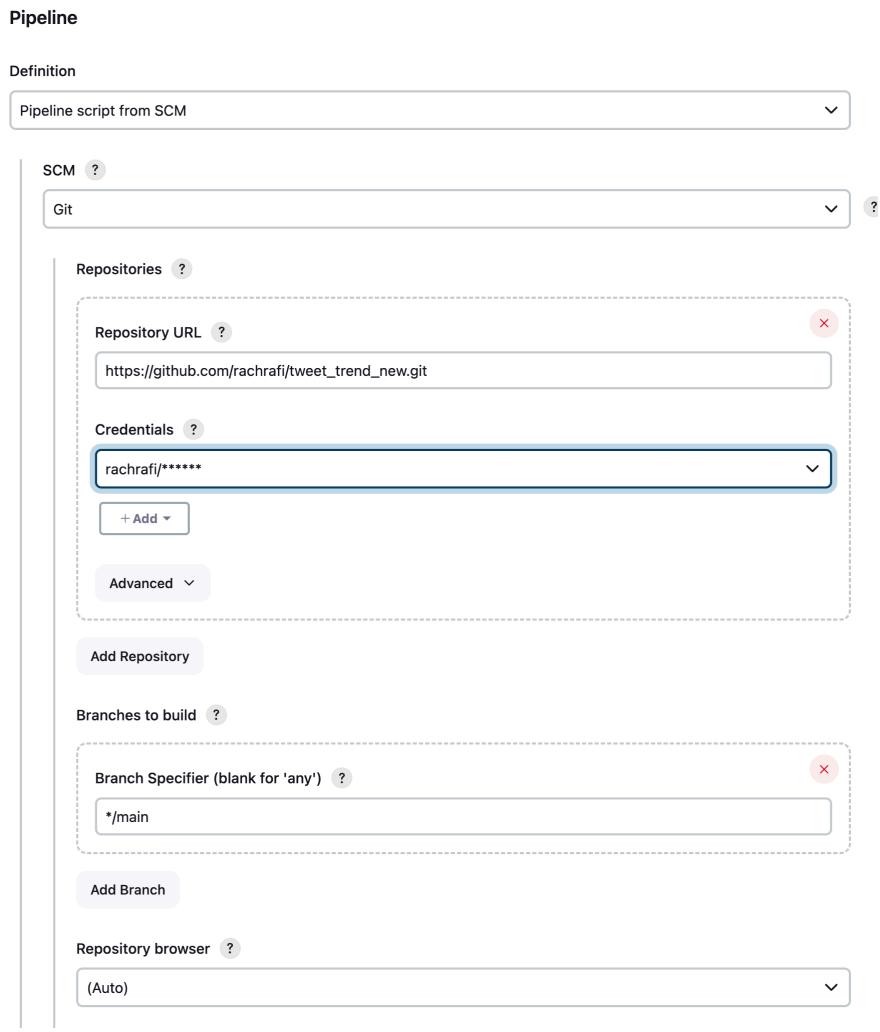
ID ?

GitHub_Cred

Description ?

Create

Add credentials to the Jenkins Jobs:



11. **Create multibranch pipeline.**

Whenever you trigger a job in Jenkins it will check if there are other branches to build not just main.

12. **Setup GitHub webhook so Jenkins is notified when there there are any changes made to the GitHub repo**

Install "multibranch scan webhook trigger" plugin
From dashboard --> manage jenkins --> manage plugins --> Available Plugins
Search for "Multibranch Scan webhook Trigger" plugin and install it.

Install	Name ↓	Released
☐	Multibranch Scan Webhook Trigger 1.0.11 Trigger that can receive any HTTP request and trigger a multibranch job scan when token matched.	3 mo 14 days ago

Once installed enable “scan by webhook” within the multibranch pipeline job and enter a random token name.

Trigger URL will look something like:

http://54.221.65.236:8080/multibranch-webhook-trigger/invoke?token=dpp-token

Now add Webhook to GitHub repo:

1. Github repo --> settings --> webhooks --> Add webhook
   Payload URI:
   <jenkins_IP>:8080/multibranch-webhook-trigger/invoke?token=<token_name>
   Content type: application/json
   Which event would you like to trigger this webhook: just the push event

13. SonarQube Configuration

1. Create Sonar cloud account on https://sonarcloud.io
2. Generate an Authentication token on SonarQube Account --> my account --> Security --> Generate Tokens
3. On Jenkins create credentials Manage Jenkins --> manage credentials --> system --> Global credentials --> add credentials - Credentials type: Secret text - ID: sonarqube-key
4. Install SonarQube plugin Manage Jenkins --> Available plugins Search for sonarqube scanner
5. Configure sonarqube server Manage Jenkins --> Configure System --> sonarqube server Add Sonarqube server - Name: sonar-server - Server URL: https://sonarcloud.io/ - Server authentication token: sonarqube-key
6. Configure sonarqube scanner Manage Jenkins --> Global Tool configuration --> Sonarqube scanner Add sonarqube scanner - Sonarqube scanner: sonar-scanner

14. Update jococo to version 0.8.4 in pom file

15. Upgraded instance type from t2.micro to t3.medium for jenkins slave and master.

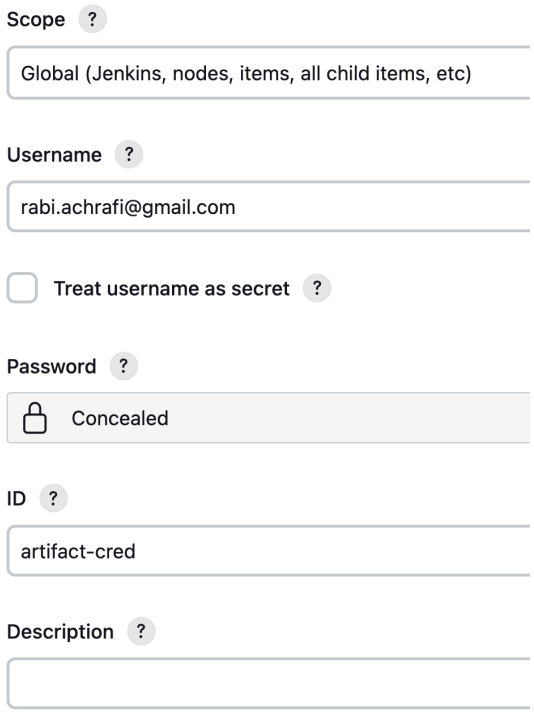
16. Create quality gate on SonarQube. Set number of bugs and code smells thresholds. Then attach quality gate to twittertrend project

17. Create quality gate stage in Jenkinsfile. This allows SonarQube to inform Jenkinsfile whether job passed quality gate or not.

Artifactory

19. Install Jfrog artifactory plugin

20. Create JFrog artifactory account. Create token in JFrog and use it to create Jenkins credentials.



21. Update Jenkinsfile with jar publish stage

```
def registry = 'https://valaxy01.jfrog.io'
        stage("Jar Publish") {
       steps {
           script {
                  echo '<--------------- Jar Publish Started --------------->'
                   def server = Artifactory.newServer url:registry+"/artifactory" ,
credentialsId:"artifactory_token"
                   def properties = "buildid=${env.BUILD_ID},commitid=${GIT_COMMIT}";
                   def uploadSpec = """{
                        "files": [
```

```
                {
                  "pattern": "jarstaging/(*)",
                  "target": "libs-release-local/{1}",
                  "flat": "false",
                  "props" : "${properties}",
                  "exclusions": [ "*.sha1", "*.md5"]
                }
              ]
            }"""
            def buildInfo = server.upload(uploadSpec)
            buildInfo.env.collect()
            server.publishBuildInfo(buildInfo)
            echo '<--------------- Jar Publish Ended --------------->'

        }
    }
}
```

Docker

22. Update ansible script for jenkins slave to install Docker:

```
---
- hosts: jenkins-slave
 become: true
 tasks:
 - name: update ubuntu repo and cache
   apt:
     update_cache: yes
     cache_valid_time: 3600

 - name: install java
   apt:
     name: openjdk-17-jre
     state: present

 - name: download maven packages
   get_url:
```

```
    url:
https://dlcdn.apache.org/maven/maven-3/3.9.6/binaries/apache-maven-3.9.6-b
in.tar.gz
    dest: /opt

 - name: extract maven packages
   unarchive:
     src: /opt/apache-maven-3.9.6-bin.tar.gz
     dest: /opt
     remote_src: yes

 - name: install docker
   apt:
     name: docker.io
     state: present

 - name: start docker services
   service:
     name: docker
     state: started
  - name: give 777 permissions on /var/run/docker.sock
   file:
     path: /var/run/docker.sock
     state: file
     mode: 0777

 - name: start docker on boot time
   service:
     name: docker
     enabled: yes
```

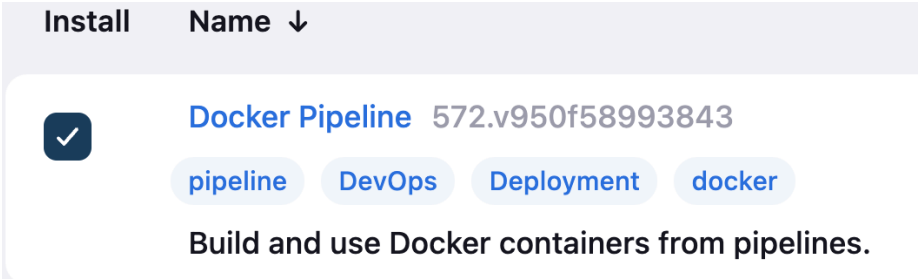
Run this script on ansible machine:
ansible-playbook -i hosts v2-jenkins-slave-setup.yaml

23. Add following docker file Dockerfile to the app repo:

```
   FROM openjdk:8
   ADD jarstaging/com/valaxy/demo-workshop/2.1.2/demo-workshop-2.1.2.jar
demo-workshop.jar
   ENTRYPOINT ["java", "-jar", "demo-workshop.jar"]
```

24. Create repo in Artifactory to store docker files

25. Install Docker Jenkins plugin called: Docker Pipeline



26. Add Docker build and publish stages to Jenkins file:

```
def registry = 'https://rachrafi.jfrog.io'
def imageName = 'rachrafi.jfrog.io/rachrafi-docker-local/ttrend'
def version   = '2.1.2'

pipeline {
   agent {
       node {
           label 'maven'
       }
   }
environment {
   PATH = "/opt/apache-maven-3.9.2/bin:$PATH"
}
   stages {
       stage("build"){
           steps {
                echo "----------- build started ----------"
               sh 'mvn clean deploy -Dmaven.test.skip=true'
```

```
                echo "----------- build complted ----------"
            }
        }
        stage("test"){
            steps{
                echo "----------- unit test started ----------"
                sh 'mvn surefire-report:report'
                 echo "----------- unit test Complted ----------"
            }
        }

   stage('SonarQube analysis') {
   environment {
     scannerHome = tool 'valaxy-sonar-scanner'
   }
   steps{
   withSonarQubeEnv('valaxy-sonarqube-server') { // If you have configured
more than one global server connection, you can specify its name
     sh "${scannerHome}/bin/sonar-scanner"
   }
   }
 }
 stage("Quality Gate"){
   steps {
       script {
       timeout(time: 1, unit: 'HOURS') { // Just in case something goes
wrong, pipeline will be killed after a timeout
   def qg = waitForQualityGate() // Reuse taskId previously collected by
withSonarQubeEnv
   if (qg.status != 'OK') {
     error "Pipeline aborted due to quality gate failure: ${qg.status}"
   }
 }
}
   }
 }
```

```
     stage("Jar Publish") {
       steps {
           script {
                   echo '<--------------- Jar Publish Started
--------------->'
                    def server = Artifactory.newServer
url:registry+"/artifactory" ,  credentialsId:"artfiact-cred"
                    def properties =
"buildid=${env.BUILD_ID},commitid=${GIT_COMMIT}";
                    def uploadSpec = """{
                         "files": [
                           {
                             "pattern": "jarstaging/(*)",
                             "target": "libs-release-local/{1}",
                             "flat": "false",
                             "props" : "${properties}",
                             "exclusions": [ "*.sha1", "*.md5"]
                           }
                        ]
                    }"""
                    def buildInfo = server.upload(uploadSpec)
                    buildInfo.env.collect()
                    server.publishBuildInfo(buildInfo)
                    echo '<--------------- Jar Publish Ended
--------------->'

           }
       }



   }

   stage(" Docker Build ") {
     steps {
       script {
          echo '<--------------- Docker Build Started --------------->'
```

```
            app = docker.build(imageName+":"+version)
            echo '<--------------- Docker Build Ends --------------->'
        }
      }
    }

    stage (" Docker Publish "){
        steps {
            script {
                echo '<--------------- Docker Publish Started
--------------->'
                 docker.withRegistry(registry, 'artifact-cred'){
                     app.push()
                 }
                echo '<--------------- Docker Publish Ended
--------------->'
            }
        }
    }
}
}
```

On Jenkins slave run container as a test:

docker run -dt --name demo-workshop -p 8000:8000
rachrafi.jfrog.io/rachrafi-docker-local/demo-workshop:2.1.2

Test app by going to jenkins slave public IP on port 8000: http://18.232.81.202:8000/

Kubernetes

27. Create Kubernetes cluster - see code

Instances are run behind ASG so new instances will be deployed and removed based on load automatically.

28. Install kubectl on Jenkins slave server:

https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/

Make sure the version is the same as the EKS version on AWS.

```
curl -LO https://dl.k8s.io/release/v1.29.2/bin/linux/amd64/kubectl
```

chmod +x ./kubectl
mv ./kubectl /usr/local/bin
kubectl version

29. Install AWS CLI on Jenkins slave

sudo ./aws/install

30. Configure AWS cli

aws configure

31. Download Kubernetes credentials and cluster configuration (.kube/config file) from the cluster

aws eks update-kubeconfig --region us-east-1 --name devops-workshop-eks-01

Now we can manage the nodes/pods e.g.
kubectl get nodes

32. Kubernetes manifest files

namespace: rachrafi
service: rachrafi-service

Create kubernetes folder under /opt on Jenkins slave and copy manifest files namespace.yaml and deployment.yaml

33. Create JFrog account used to pull images in Kubernetes
Username: docker-cred
Pwd: Rachrafi1978!

On jenkins slave test original JFrog login using Jfrog account:
Enter the command: docker login https://rachrafi.jfrog.io

Username: rabi.achrafi@gmail.com
Pwd: Rachrafi1978!

34. Copy secrets and services manifest file to Jenkins slave in /opt/kubernetes folder and apply the file

35. Move kubernetes folder from /opt to /home/ubuntu and set permissions on the folder and contents to ubuntu since we are using the ubuntu on Jenkins so we need to use the same account so Jenkins can run the job to create the K8 cluster

mv kubernetes /home/ubuntu/
chown -R ubuntu:ubuntu /home/ubuntu/

Now run aws configure command under the ubuntu user
Execute cluster config credentials command:
aws eks update-kubeconfig --region us-east-1 --name devops-workshop-eks-01

36. Copy the deploy.sh script into the kubernetes folder on Jenkins slave and add execute permisions on it

37. Add deploy stage to Jenkinsfile and commit to Git

38. Make sure /var/run/docker.sock has 777 permissions on the jenkins slave server

39. Whenever making changes to code we need to update the versions in the following files:
- Dockerfile
- Jenkinsfile
- deployment.yaml
- pom.xml

So if the current version is 2.1.2 we need to update the version to 2.1.3 in the above 4 files.

Helm Configuration

Commiting new Code changes.

- Delete the existing deployment using helm:
    - helm uninstall devops-workshop
- helm create devops-workshop

- delete all file in templates folder and replace with our manifest files:

  deployment.yaml, (update version)
  service.yaml,
  secret.yaml,
  namespace.yaml

- Update "version" and "appVersion" in file Chart.yaml

version = is the chart version and should be updated each time you make changes to the chart and its templates, including the app version.
appVersion = this is the version of the application being deployed.

- Now create helm package which will create a .tgz file
  - helm package devops-workshop

Update versions in the following files:

- Dockerfile
- Jenkinsfile
- pom.xml

If cluster is newly provisioned you must do the following:

- Uncomment sgs and eks modules and provision cluster
- Ensure /var/run/docker.sock has 777 permissions on the jenkins slave server
- Create security group rule on one of the K8 pods to allow access to port 30082 on the pod IP to test app
- **Execute cluster config credentials command to update kube config on Jenkins slave:**
  - aws eks update-kubeconfig --region us-east-1 --name devops-workshop-eks-01

Configure Prometheus and Grafana

Note: Worth remember prometheus uses port 9090

Create a dedicated namespace for prometheus
kubectl create namespace monitoring

Add prometheus helm chat repo
helm repo add prometheus-community
https://prometheus-community.github.io/helm-charts

Update helm chart repo
helm repo update

helm repo list

Install prometheus and grafana stack
helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring

We can't access prometheus since it's using ClusterIP so we need to change it to load balancer type. We can do this using on the fly edit command:

kubectl edit service prometheus-kube-prometheus-operator -n monitoring

Change "ClusterIP" to "LoadBalancer"

We can view updated details including what the DNS name is by viewing the K8 services:
kubectl get service -n monitoring

To access Prometheus simply access the DNS name followed by port 9090 e.g.

http://a8113c599100b4bb099a75e2316e4141-1502529474.us-east-1.elb.amazonaws.com:9090

Grafana

To access Grafana we need to do similar process that is change the grafana pod from using ClusterIP to LoadBalancer.

Run command and change "ClusterIP" to "LoadBalancer"
kubectl edit service prometheus-grafana -n monitoring

Access Grafana using the URL provided by the output of above command:
http://ad7329212e92f4806b6c4a686770d1f6-839587635.us-east-1.elb.amazonaws.com

Login details:
username: admin
password: prom-operator

Terminate Kubernetes Cluster

1. Remove load balancers from prometheus and grafana by changing LoadBalancer back to ClusterIP by editing their services:
   kubectl edit service prometheus-kube-prometheus-prometheus -n monitoring
   kubectl edit service prometheus-grafana -n monitoring
2.

React Deployment Lab

1. Create key pair
2. Create VPC terraform code
3. SSH onto ansible machine and install ansible:

   ```
   sudo apt update
   sudo apt install software-properties-common
   sudo add-apt-repository --yes --update ppa:ansible/ansible
   sudo apt install ansible
   ```

4. scp key to ansible machine:

   scp -i dpp.pem dpp.pem ubuntu@34.229.208.50:/home/ubuntu/

   Place key in /opt/ directory

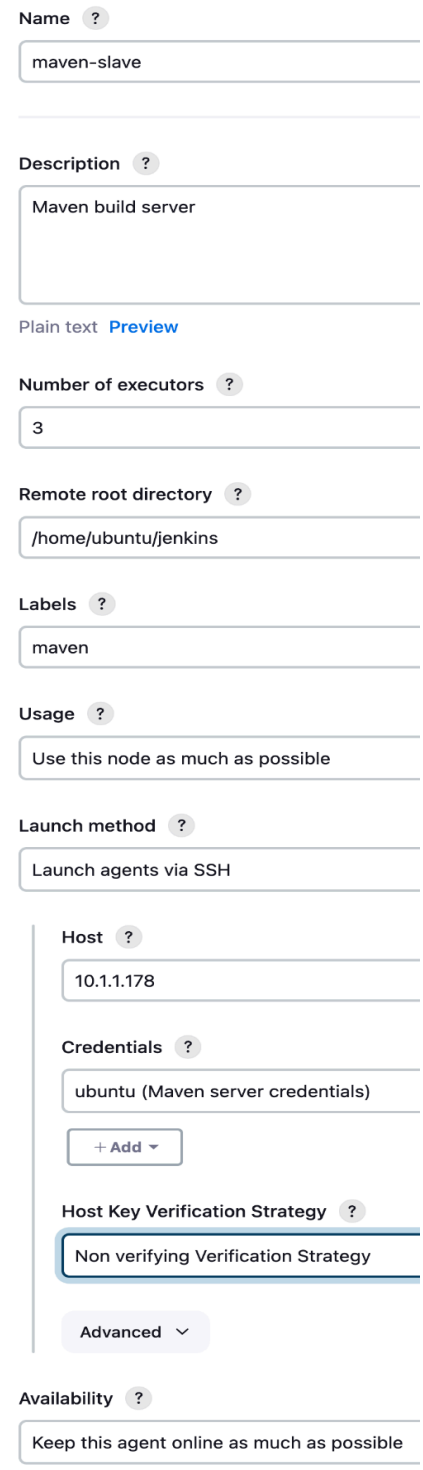
   Note: Ensure permissions on key does not have write permissions i.e. use chmod 400.

   Do this for slave and master machines.

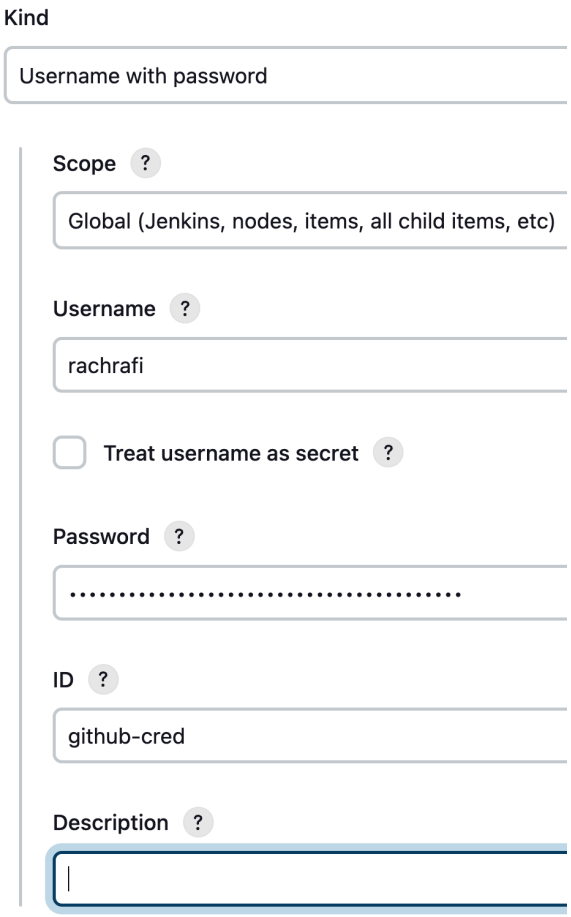
5. Copy the ansible master and slave scripts to /opt folder on ansible machine
   Run jenkins master playbook:
   ansible-playbook -i /opt/hosts jenkins-master-setup.yaml

   Run the jenkins slave playbook:
   ansible-playbook -i /opt/hosts jenkins-slave-setup.yaml
6. Log into Jenkins and change password

7. create maven server credentials using the key pair we created in AWS aka **react-lab.pem.** Call it maven-server-credentials
8. Install SSH Agent Plugin
9. Add maven slave as a node in Jenkins



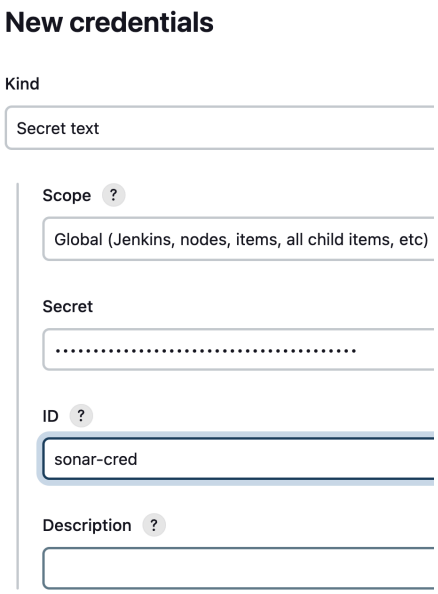
10. Create github token and use that to create a new Jenkins credential called github-cred:



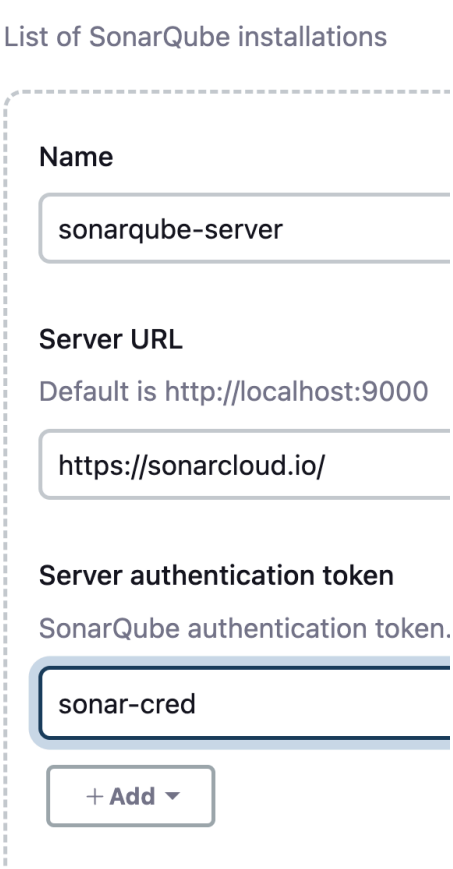
11. Create multibranch pipeline: react-multibranch-pipeline
12. Install multibranch scan webhook trigger plugin
13. Add token “react-token” to scan by webhook option on the multibranch pipeline
14. Add webhook to Github
    “http://54.196.163.69:8080/multibranch-webhook-trigger/invoke?token=react-token” as Github webhook
15. Add distribution management to Pom file:

```
<distributionManagement>
        <repository>
            <id>dev</id>
            <url>file://jarstaging</url>
        </repository>
    </distributionManagement>
```

16. Configure SonarQube and Sonar Scanner
17. Create SonarQube account in https://sonarcloud.io
18. Create token in SonarQube called “jenkins-token”
19. Use this token to create a new credential in Jenkins:

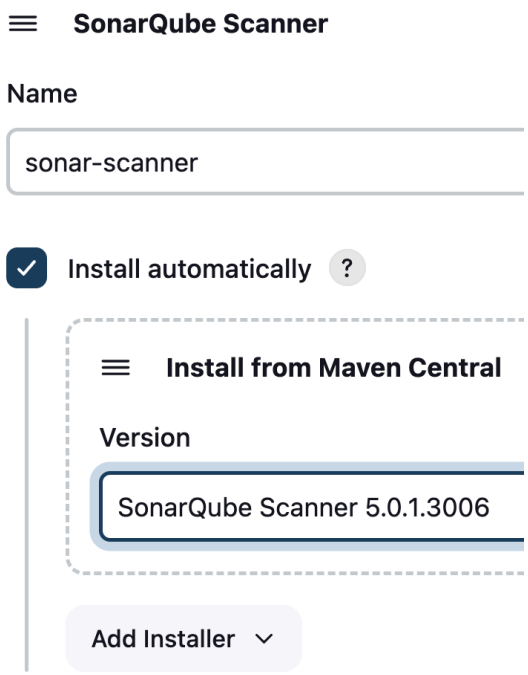


20. Install SonarQube scanner plugin on Jenkins
21. Add SonarQube server to Jenkins under System:



Note: Remove the slash after .io

22. Add SonarQube Scanner in Jenkins under Tools:



23. Create SonarQube Properties file
24. Create SonarQube Jenkins stage
25. Create SonarQube quality gate
26. Needed to add slight delay as there seemed to be race condition affecting QualityGate.

Artifactory Configuration

27. Create Artifactory Token and create credentials in Jenkins called artifact-cred
28. Install Artifactory plugin
29. In