Section-11

A white background with black text

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A diagram of a company

Description automatically generated

**Pre-requisites:**

1. **Code in local**
2. **Aws account mit repository in code commit**
3. **Iam user account to push code to codecommit. This user having permissions to push to codecommit.**
4. **Now create codecommit repo**
5. **Connect the local code with codecommit repos using ssh. Later you need to upload public key in iam user policy.**
6. **Now after proper configuration you can push local code to codecommit with below commands**

ssh-keygen

vim ~/.ssh/config

ssh git-codecommit.us-east-1.amazonaws.com

git clone https://github.com/hkhcoder/vprofile-project.git

git branch -a | grep remotes | grep -v HEAD |cut -d / -f3 > /tmp/br

for i in `cat /tmp/br`; do git checkout $i; done

git remote add origin ssh://git-codecommit.us-east-1.amazonaws.com/v1/repos/rakesh-codecommit

git push origin --all

Aws code artifact:

Select maven central store

Created a repo in code artifact ( vprofile code artifact and maven central store)

Later in sonar cloud -> created organistion ,then in security generate a token, then new project

Then based on sonar buildspec.yml file some attributes values are stored in ssm param store

AWS code build:

* Now in code artifact maven central repo copy the url(in connection instructions) by selecting mvn .
* The above copied link should be modified in pom.xml, settings.xml
* Now copy from same codeartifact maven central repo the auth token(connection instructions->mvn)
* Change the sonar\_buildspec.yml to buildspec.yml and put in it root folder
* Copied token should be put in buildspec.yml, under commands
* And push all modifed code to aws codecommit

Now configure all necessary things in aws code build(also point to write branch). Create the new role in that configuration of codebuild. For this code build role you need to attach policies

Such as access to code artifact and and custom policy for reading the ssm param store values

Later ready to build and test application

77) Build jar file and store in s3:

* Initially only code build configuration is done. Create another codebuild project with path for buildspec in aws-files( here u need to replace with the auth token from mvn central)

And for the role you selected in aws code build, we need to attach awscodeartifact permission.

Now if you build , the jar file should be ready

**Code pipeline: this pipeline you configure initially codebuild(which builds the jar file from earlier step) and another stage for codeanalysis build(from previous buid) finally the deploy part is configured for deploying to s3. Also configure the sns notifications for email.**

Pipeline stages: source -> codeanalysis with sonar (codebuil)-> build jar (codebuild) -> deploytos3 (input jar will be from previous codebuild)

**Section-12 ( cd of java web application)**

**A computer screen shot of a computer

Description automatically generated**

**A diagram of software

Description automatically generated**

**A blue screen with white text

Description automatically generated**

**A blue screen with white text

Description automatically generated**

First, create security groups for nexus(port: 8081, port 22, it will allow traffic from Jenkins)

Create sg for sonar(port: 80, port:22(for ssh)) , it will allow traffic from Jenkins)

Then create sg for Jenkins(port: 8080, port:22(for ssh), it will allow traffic from sonar)

Then create ec2 instances for sonar(t2 medium), nexus(t2medium) and Jenkins(t2 small)

**Eg for ssh:**

ssh -i "vprofile-prod-key-virginia.pem" [ec2-user@ec2-54-146-78-153.compute-1.amazonaws.com](mailto:ec2-user@ec2-54-146-78-153.compute-1.amazonaws.com)

* After you ssh into Jenkins server, you can switch to root user and see
* Systemctl status Jenkins
* Now copy publicip:portnumber in browser jenkisn will open
* Later on install all plugins (nexus,github,sonar,timestamps,slack,)

Ssh in to nexus;

* Login as admin
* 1)Create repository -- “vprofile-release”(choose maven hosted)
* 2) Create repo-(choose mavenproxy)- name: Vpro-maven-central

Remote storage: (if nexus needs dependencies to download it will use this ) = https://repo1.maven.org/maven2/

3) create repository( choose maven hosted) : (name: vprofile-snapshot)

4) create repository (choose maven group) : all the above repos u put it as members

For this group

Sonarqube :

Paste directly the publicip in browser sonarqube will open

Do ssh config for his github where his repo is forked. And now commit something to his forked repo in your account, then you can go further

Now go to Jenkins ssh terminal: install jdk 8

apt update && apt install openjdk-8-jdk -y

**later do :** ls /usr/lib/jvm/

**Go to Jenkins ui:**

**now in Jenkins tools add jdk in java\_home:**

/usr/lib/jvm/ java-1.8.0-openjdk-amd64

/usr/lib/jvm/ java-1.11.0-openjdk-amd64

Now in manage Jenkins -> tools-> choose maven -> select 3.9.5 version -> and save

Now in credentials-> system -> choose id nexuslogin-> and admin as username and pwd

Now in local code in branch ci -jenkins:

Add below jenkinfile code

pipeline {  
  
 agent any  
  
 tools {  
 maven "MAVEN3"  
 jdk "OracleJDK8"  
 }  
  
 environment {  
  
 *SNAP\_REPO* = 'vprofile-snapshot'  
 *NEXUS\_USER* = 'admin'  
 *NEXUS\_PASS* = 'admin'  
 *RELEASE\_REPO* = 'vprofile-release'  
 *CENTRAL\_REPO* = 'vpro-mven-central'  
 *NEXUSIP* = '172.31.47.217'  
 *NEXUSPORT* = '8081'  
 *NEXUS\_GRP\_REPO* = 'vpro-maven-group'  
 *NEXUS\_LOGIN* = 'nexuslogin'  
 }  
  
 stages{  
  
 stage('BUILD'){  
 steps {  
 sh 'mvn -s settings.xml -DskipTests install'  
 }  
  
  
 }  
 }  
  
  
  
}

the you push the updated code to the forked branch

Now go to Jenkins dashboard:

1. Create item
2. Choose pipeline
3. Select pipeline script from scm
4. Scm: git
5. Repo url: copy ssh url
6. A screenshot of a computer

   Description automatically generated
7. A screenshot of a computer

   Description automatically generated
8. The below is the command to get private key from sshgen

cat ~/.ssh/id\_rsa

)

1. Do ssh in to Jenkins and do this

su - jenkins

git ls-remote -h git@github.com:rakeshallampally657/vprofile-project.git HEAD

cat .ssh/known\_hosts

1. Choose the right branch and do build

**66) Git hub webhook:**

payloadUrl: <http://54.162.2.36:8080/github-webhook/> (public ip)

select the push event

add webhook

now go to Jenkins ui-> vprofile pipeline -> configuration

select the “GitHub hook trigger for GITScm polling[”](http://54.162.2.36:8080/job/vprofile-ci-pipeline/configure)

in Jenkins file add additional stages and do build again

pipeline {  
  
 agent any  
  
 tools {  
 maven "MAVEN3"  
 jdk "OracleJDK8"  
 }  
  
 environment {  
  
 *SNAP\_REPO* = 'vprofile-snapshot'  
 *NEXUS\_USER* = 'admin'  
 *NEXUS\_PASS* = 'admin'  
 *RELEASE\_REPO* = 'vprofile-release'  
 *CENTRAL\_REPO* = 'vpro-mven-central'  
 *NEXUSIP* = '172.31.60.58'  
 *NEXUSPORT* = '8081'  
 *NEXUS\_GRP\_REPO* = 'vpro-maven-group'  
 *NEXUS\_LOGIN* = 'nexuslogin'  
 }  
  
 stages{  
  
  
 stage('BUILD'){  
 steps {  
 sh 'mvn -s settings.xml -DskipTests install'  
 }  
  
 post{  
 success {  
 echo "Now archiving"  
 archiveArtifacts *artifacts*: '\*\*/\*.war'  
 }}  
 }  
  
 stage('Test'){  
 steps{  
 sh 'mvn test'}  
 }  
  
 stage('Checkstyle analysis'){  
 steps{  
 sh 'mvn checkstyle:checkstyle'}  
 }  
  
  
  
  
 }  
  
  
  
}

**code analysis with sonarqube**:

1)go to tools in manage Jenkins -> name: sonnarscanner -> version : 4.7

2) manage jenkisn -> system -> name: sonarserver , sever: <http://privateipaddress>ofsonarserver ,

soanr token: you need first generate from its ui and copy it.

Now in manage Jenkins-> server auth token -> kind: secret text ; id: sonartoken

3) modify jenkinsfile add sonaranalysis part

pipeline {  
  
 agent any  
  
 tools {  
 maven "MAVEN3"  
 jdk "OracleJDK8"  
 }  
  
 environment {  
  
 *SNAP\_REPO* = 'vprofile-snapshot'  
 *NEXUS\_USER* = 'admin'  
 *NEXUS\_PASS* = 'admin'  
 *RELEASE\_REPO* = 'vprofile-release'  
 *CENTRAL\_REPO* = 'vpro-mven-central'  
 *NEXUSIP* = '172.31.60.58'  
 *NEXUSPORT* = '8081'  
 *NEXUS\_GRP\_REPO* = 'vpro-maven-group'  
 *NEXUS\_LOGIN* = 'nexuslogin'  
 *SONARSERVER* = 'sonarserver'  
 *SONARSCANNER* = 'sonarscanner'  
 }  
  
 stages{  
  
  
 stage('BUILD'){  
 steps {  
 sh 'mvn -s settings.xml -DskipTests install'  
 }  
  
 post{  
 success {  
 echo "Now archiving"  
 archiveArtifacts *artifacts*: '\*\*/\*.war'  
 }}  
 }  
  
 stage('Test'){  
 steps{  
 sh 'mvn -s settings.xml test'}  
 }  
  
 stage('Checkstyle analysis'){  
 steps{  
 sh 'mvn -s settings.xml checkstyle:checkstyle'}  
 }  
  
 stage('CODE ANALYSIS with SONARQUBE') {  
  
 environment {  
 scannerHome = tool "${SONARSCANNER}"  
 }  
  
 steps {  
 withSonarQubeEnv("${SONARSERVER}") {  
 sh '''${scannerHome}/bin/sonar-scanner -Dsonar.projectKey=vprofile \  
 -Dsonar.projectName=vprofile \  
 -Dsonar.projectVersion=1.0 \  
 -Dsonar.sources=src/ \  
 -Dsonar.java.binaries=target/test-classes/com/visualpathit/account/controllerTest/ \  
 -Dsonar.junit.reportsPath=target/surefire-reports/ \  
 -Dsonar.jacoco.reportsPath=target/jacoco.exec \  
 -Dsonar.java.checkstyle.reportPaths=target/checkstyle-result.xml'''  
 }  
  
  
 }  
 }  
  
  
  
}

**Quality gate for sonar:**

1. First in sonar ui create a quality gate which sets bugs limit to 25
2. Now for the project you attach this quality gate you created
3. Create a webhook for the same project

Name: jenkinswebhook

url: <http://172.31.47.217:8080/sonarqube-webhook>

stage("quality gate"){  
 steps{  
 timeout(*time*: *10*, *unit*: 'MINUTES') {  
 waitForQualityGate *abortPipeline*: *true* }  
 }  
}

ADD the above stage to pipeline and run

**69) publish artifact to nexus repo**

1) go to manage Jenkins -> system -> build timestamp: choose format of time stamp

2) add new stage to pipeline

stage("upload artifact to nexus"){  
 steps {  
 nexusArtifactUploader(  
 *nexusVersion*: 'nexus3',  
 *protocol*: 'http',  
 *nexusUrl*: "${NEXUSIP}:${NEXUSPORT}",  
 *groupId*: 'QA',  
 *version*: "${env.BUILD\_ID}-${env.BUILD\_TIMESTAMP}",  
 *repository*: "${RELEASE\_REPO}",  
 *credentialsId*: "${NEXUS\_LOGIN}",  
 *artifacts*: [  
 [*artifactId*: 'vproapp',  
 *classifier*: '',  
 *file*: 'target/vprofile-v2.war',  
 *type*: 'war']  
 ]  
 )  
  
 }  
}

**Section: 12**

<https://github.com/devopshydclub/vprofile-project/tree/docker>

* checkout out the docker branch and download the above zipped repo.
* From the previous repo branch ci-jenkins you need to checkout to the new branch as

Cicd-jenkins.

* Now copy docker-files folder from above downloaded repo to the branch of cicd-jenkins.
* Now create 2 folders ProdPipeline, StagePipeline
* And copy Jenkinsfile in to the above created folder and remove from the old location
* Create iam user which has 2 permissions (1. [AmazonEC2ContainerRegistryFullAccess](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonEC2ContainerRegistryFullAccess)

2. [AmazonECS\_FullAccess](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonECS_FullAccess) )

* Create ecr repo
* Go to Jenkins ui -> manage plugins -> and install following plugins:

[Docker Pipeline,](https://plugins.jenkins.io/docker-workflow) [Docker Build and PublishVersion1.4.0](https://plugins.jenkins.io/docker-build-publish) , Amazon ecr, [Pipeline: AWS StepsVersion1.45](https://plugins.jenkins.io/pipeline-aws)

* Manage Jenkins - > credentials -> system -> global credentials ->

Kind: AWS credentials ,

A screenshot of a computer

Description automatically generated

* Ssh to jenkins terminal:
* apt update && apt install awscli -y (run this command on cmd )
* then install docker in ssh terminal of Jenkins server
* now do “su - jenkins” but you see you don’t have paermission to view docker images
* so do “usermod -aG docker Jenkins”
* id jenkins ( will show groups and uids)
* now go to jenkinsfile in the stagepipeline:
* add new stages to existing pipeline such : build dockerimage, upload image to ecr.

pipeline {  
  
 agent any  
  
 tools {  
 maven "MAVEN3"  
 jdk "OracleJDK8"  
 }  
  
 environment {  
  
 *SNAP\_REPO* = 'vprofile-snapshot'  
 *NEXUS\_USER* = 'admin'  
 *NEXUS\_PASS* = 'admin'  
 *RELEASE\_REPO* = 'vprofile-release'  
 *CENTRAL\_REPO* = 'vpro-mven-central'  
 *NEXUSIP* = '172.31.60.58'  
 *NEXUSPORT* = '8081'  
 *NEXUS\_GRP\_REPO* = 'vpro-maven-group'  
 *NEXUS\_LOGIN* = 'nexuslogin'  
 *SONARSERVER* = 'sonarserver'  
 *SONARSCANNER* = 'sonarscanner'  
 registryCredential = 'ecr:us-east-1:awscreds'  
 appRegistry = "891377113487.dkr.ecr.us-east-1.amazonaws.com/vprofile-ecr-rakesh"  
 fileRegistry = "https://891377113487.dkr.ecr.us-east-1.amazonaws.com"  
 }  
  
 stages{  
  
  
 stage('BUILD'){  
 steps {  
 sh 'mvn -s settings.xml -DskipTests install'  
 }  
  
 post{  
 success {  
 echo "Now archiving"  
 archiveArtifacts *artifacts*: '\*\*/\*.war'  
 }}  
 }  
  
 stage('Test'){  
 steps{  
 sh 'mvn -s settings.xml test'}  
 }  
  
 stage('Checkstyle analysis'){  
 steps{  
 sh 'mvn -s settings.xml checkstyle:checkstyle'}  
 }  
  
 stage('CODE ANALYSIS with SONARQUBE') {  
  
 environment {  
 scannerHome = tool "${SONARSCANNER}"  
 }  
  
 steps {  
 withSonarQubeEnv("${SONARSERVER}") {  
 sh '''${scannerHome}/bin/sonar-scanner -Dsonar.projectKey=vprofile \  
 -Dsonar.projectName=vprofile \  
 -Dsonar.projectVersion=1.0 \  
 -Dsonar.sources=src/ \  
 -Dsonar.java.binaries=target/test-classes/com/visualpathit/account/controllerTest/ \  
 -Dsonar.junit.reportsPath=target/surefire-reports/ \  
 -Dsonar.jacoco.reportsPath=target/jacoco.exec \  
 -Dsonar.java.checkstyle.reportPaths=target/checkstyle-result.xml'''  
 }  
  
  
 }  
 }  
  
 stage("quality gate"){  
 steps{  
 timeout(*time*: *10*, *unit*: 'MINUTES') {  
 waitForQualityGate *abortPipeline*: *true* }  
 }  
 }  
  
 stage("upload artifact to nexus"){  
 steps {  
 nexusArtifactUploader(  
 *nexusVersion*: 'nexus3',  
 *protocol*: 'http',  
 *nexusUrl*: "${NEXUSIP}:${NEXUSPORT}",  
 *groupId*: 'QA',  
 *version*: "${env.BUILD\_ID}-${env.BUILD\_TIMESTAMP}",  
 *repository*: "${RELEASE\_REPO}",  
 *credentialsId*: "${NEXUS\_LOGIN}",  
 *artifacts*: [  
 [*artifactId*: 'vproapp',  
 *classifier*: '',  
 *file*: 'target/vprofile-v2.war',  
 *type*: 'war']  
 ]  
 )  
  
 }  
 }  
  
 stage('build app image'){  
 steps{  
 script {  
 dockerImage = docker.build(appRegistry + ":$BUILD\_NUMBER" , "./Docker-files/app/multistage/")  
 }  
 }}  
  
 stage('upload app image to ecr'){  
 steps{  
 script {  
 docker.withRegistry(vprofileRegistry, registryCredential){  
 dockerImage.push("$BUILD\_NUMBER")  
 dockerImage.push('latest')  
 }  
 }  
 }  
 }  
 }  
  
  
  
}

**85**: **deploy to ecs**

A diagram of a software

Description automatically generated

1. create a ecs cluster
2. then create a task definition. Here you give reference to the ecr image you want to deploy and also configure other stuff such as cpu etc
3. then create a service, here you choose the task definition you created. Then here you create a load balancer.
4. Go to target group , and go to health checks -> advance health checks -> override: port 8080
5. Later on add new stage to deploy to ecs with cli command “aws ecs updateservice”

87) **promote to production :**

1) create a new cluster again, new task definition(with image ref), create service

2) create a new pipeline for prod.

3) create a new branch, and use this time Jenkins file in prodpipeline folder. Now clean the jenksfile for prod env. Keep only “deploy to ecs” stage. And push it.

4) in Jenkins create a new pipeline for prod. Use branch as : prod.