**LOGIN TO AWS ACCOUNT**

**CODE COMMIT**

**1.Create codecommit repo** - vprofile-maven-repo > Public upstream: maven-central-store > This account > Domain: visualpath. We see 2 repositories one for maven-central-store used to store dependencies. Click on maven-central-store > view connection instruction > mvn > It will show you steps 1.Export CodeArtifact authorization token 2.Changes in settings.xml.

From maven-central-store here

**2.Create iam user with codecommit policy** – vprofile-arti-admin > AWSCodeArtifactAdminAccess > Copy Access & Secret Access key. On Gitbash aws configure.

Run command export CODEARTIFACT\_AUTH\_TOKEN=`aws codeartifact get-authentication-token --domain visualpath –domain-owner <account\_id> --query authorizationToken –-output text`. Run echo $CODEARTIFACT\_AUTH\_TOKEN This will generate token which is used to interact CodeArtifact with CodeBuild

Changes in settings.xml see CHANGEHERE.Settings.xml used to configure Maven settings, particularly for repository management and authentication.Total 3 changes.

Changes in pom.xml see CHANGEHERE.pom.xml is Maven POM (Project Object Model) configuration file which contains project's metadata, dependencies, build settings, and repository information for a Java web application.It contains artifactId(artifactName), packaging(war), version, properties, dependencies, build, finalName, plugin, repositories.Added 2 lines of timestamp in properties.Total 2 changes.

Login sonarcloud with github credentials. Generate token: My account > Security > Token name: vprofile-sonar-cloud > copy token. Click plus + symbol > Analyze new project > Create a project manually > Select Organization > Project key: vprofile-repo > set up

We need : 1.Token 2.Project key/name: vprofile-repo 3.URL sonarcloud: <https://sonarcloud.io> 4.Organization name:xyz

AWS > System manager > Parameter store > Create parameter

1.Name: Organization > Value: < organization\_name\_sonarCloud> xyz

2. Name: HOST > Value: <URL\_sonarcloud> https://sonarcloud.io

3. Name: Project > Value: <Project name> vprofile-repo

4.Name: sonartoken > Type: SecureString Value: <Paste token here>

5.Name: codeartifact-token > Type: SecureString Value: <paste token by gitbash echo $CODEARTIFACT\_AUTH\_TOKEN>

AWS > codebuild > Create build project > vprofile-build > Repository: select > Branch: select > Operating system: ubuntu > Runtime: standard > image: 3.0 > Role name: give role name > Insert build command > Paste buildspec.yml with changes below

Buildspec.yml contains : version, env: <environment variables from parameter store>, phases >> **install:** runtime-versions(java: openjdk8), commands(copy settings.xml to /root/.m2/settings.xml) ; **pre-build :** update, install jq checkstyle, wget download maven, extract maven, softlink to maven directory, wget download sonarscanner, unzip sonarscanner, export path for sonar export PATH=$PATH:/sonar-scanner/bin/; **build :** commands: - mvn test, – mvn checkstyle:checkstyle, mvn sonar:sonar with all parameter store values, - sleep 5, - command to create result.json, - cat result.json, - with if condition check quality gate pass or fail

Changes in buildspec.yml : CHANGEHERE

> Logs > Cloudwatch > Group name: vprofile-nvir-buildlogs > stream name: sonarbuildjob > create build project

Build project > vprofile-build > Edit > Environment > copy rolename > go to IAM roles > paste role find it > attach policies > Create policy > service system manager, Actions: DescribeParameters, DescribeDocumentParameters, GetParameters, GetParametersHistory, GetParameter, GetParametersByPath > Policy name: vprofile-sonarparameteraccess > attach policy to role. > start build

In sonarcloud we see result of bugs & everything . Set quality gates on it

Create project with build\_buildspec.yml file content which create artifact Changes in build\_buildspec.yml CODEARTIFACT\_AUTH\_TOKEN: sonartoken. Added extra step artifacts in build\_buildspec.yml

AWS > SNS > Create topic > vprofile-pipeline-notification > Create topic > Create subscription > email > emailid > goto email confirm subscription >

AWS > Codebuild > pipelines > Create pipeline > vprofile-ci-pipeline > Codecommit > Repo > branch > cloudwatch > codebuild > select project 2nd > Skip deploy stage > create pipeline > stop execution > edit > Add stage > Test > Add action group > Sonar-code-analysis > Codenbuild > 1st project > SourceArtifact > Done

Create s3 bucket with pipeline-artifact folder in it

Add stage > Deploy > Deploy-to-s3 > Amazon s3 > BuildArtifact > bucket name > Directory name > check Extract artifact > Done > Save

Git pull repository in local make changes in README file commit & push ci-aws we see pipeline

3.Generate ssh keys locally

4.Exchange keys with IAM user

5.Put source code from github repo to cc repository and push

CODE ARTIFACT

Create an IAM user with code artifact access

Install AWS CLI, configure

Export auth token

Update settings.xml file in source code top level directory with below details

Update pom.xml file with repo details

SONAR CLOUD

Create sonar cloud account

Generate token

Create SSM parameters with sonar details

Create Build project

Update codebuild role to access SSMparameterstore

CREATE NOTIFICATIONS FOR SNS OR SLACK

BUILD PROJECT

Update pom.xml with artifact version with timestamp

Create variables in SSM => parametersore

Create build project

Update codebuild role to access SSMparameterstore

CREATE PIPELINE

Codecommit

Testcode

Build

Deploy to s3 bucket

TEST PIPELINE

12 - Continuous Delivery on AWS Cloud [Java Application]

Choose region with Code Artifact service

Create iam with policy codecommit with All codecommit actions & resource Add arn region & repo name.

Create repo on codecomiit. Access repo by ssh.

Steps to configure codecommit repo by ssh :

1. Install git & awscli

2.Create IAM user with no programmatic access & Create & assign policy of codecommit all actions & resources Add ARN region & codecommit repo name to IAM user.

3.Generate ssh keys by ssh-keygen in .ssh folder with different name like vpro-rsa.

4.Copy public key paste it in IAM user > Security credentials > Upload ssh public key > paste public key

4.Create config file in .ssh & add following lines to it & chmod 600 config

Host git-codecommit.\*.amazonaws.com

User <Paste ssh key id generated from uploading ssh public key above>

IdentityFile ~/.ssh/<private\_key\_name\_only>

5. Test config file is correct or not : ssh –v git-codecommit.<region\_of\_codecommit\_repo>.amazonaws.com & test cloning of repo in /tmp git clone <ssh\_path\_codecommit repo>

6.In some folder git clone <https://github.com/devopshydclub/vprofile-project.git> > git branch –a

list remote branches and filter out those with names "HEAD" and "master”:

git branch -a | grep -v HEAD | cut -d '/' -f3 | grep -v master > /tmp/branches

cat / tmp/branches

for i in `cat /tmp/branches` ; do echo $i ; done

Check that each branch is up to date or doesn't have any uncommitted changes before working on it :

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

git branch –a : see all branches checked out

git fetch --tags

git remote rm origin

git remote add origin <ssh URL of codecommit repo>

cat .git/config : we see codecommit is remote repo instead of github

git push --all origin : push all branches to codecommit repo

git push –tags

Check codecommit repo

7.

Managed Policies:

1. Managed Policies: AdministratorAccess, PowerUserAccess, AmazonS3FullAccess, AmazonEC2FullAccess, AmazonRDSFullAccess, AmazonDynamoDBFullAccess, AmazonSNSFullAccess, AWSLambda\_FullAccess, AWSReadOnlyAccess, SecurityAudit

2. Customer Managed Policies: EC2FullAccess, S3BucketReadWrite, RDSReadOnlyAccess, DynamoDBLimitedAccess, LambdaInvokeOnly, LambdaInvokeOnly,

Inline Policies:

Boundary Policies:

Resource-Based Policies:

Permission Boundaries:

Session Policies:

Organizations Policies:

Service Control Policies (SCPs):

Access Control List (ACL) Policies:

Use single key for 3 instances

jenkins-SG : CustomTCP 22 MyIP, CustomTCP 8080 AnyIPV4(github can connect to jenkins), CustomTCP 8080 AnyIPV6, CustomTCP 8080 sonar-SG(sonar server send reports back to jenkins, for restricted connection)

nexus-SG : CustomTCP 22 MyIP, CustomTCP 8081 MyIP, CustomTCP 8081 jenkins-SG(jenkins can upload artifact to nexus)

sonar-SG : CustomTCP 22 MyIP, CustomTCP 80 MyIP(nginx service listen on 80 in sonar server), CustomTCP 80 jenkins-SG(jenkins will upload test result to sonar server)

Create jenkins, nexus & sonarqube instances with userdata

For nexus userdata : service that can not run through systemctl, we need to create nexus.service file for it

For sonarqube userdata : create system confguration for sonarqube, install postgres & create database & user, sonar.properties have database information, sonarqube.service for systemctl start, install nginx.Nginx listen on port 80 & forward request to port 9000. Nginx as a reverse proxy to forward requests from port 80 to another port 9000 offers flexibility, security, performance optimization, and centralized control over your web server setup.

Plugins – maven integration, github integration, nexus artifact uploader, sonarqube scanner, slack, build timestamp

Login to nexus in browser :

Create repository > maven2 hosted > vprofile-release > create used to upload tested artifact to nexus

Create repository > maven2 proxy > vpro-maven-central > Remote storage : https://repo1.maven.org/maven2/ maven tool download dependencies from this repository

Create repository > maven2 hosted > vprofile-snapshot > Version policy : snapshot used to manage and share versions of software artifacts.

Create repository > maven2 group > vpro-maven-group > add all above created repository used to group all repositories together

Created vprociproject repo on github

$ ssh -T [git@github.com](mailto:git@github.com)

$ git clone -b ci-jenkins <https://github.com/devopshydclub/vprofile-project.git>

mv vprofile-project/ vprociproject

cd vprociproject/

cat .git/config

git remote set-url origin git@github.com:VaibhavB12/vprociproject.git

cat .git/config

git branch -c main

git checkout main

git branch ci-jenkins

git checkout ci-jenkins

git push --all origin

$ git config --global user.email [vaibha.nikalje@gmail.com](mailto:vaibha.nikalje@gmail.com)

$ git config --global user.name "VaibhavB12"

Vi ~/.gitconfig

Code .

Global tool configuration > Install openjdk8 manually OracleJDK8. Paste path /usr/lib/jvm/java-1.8.0-openjdk-amd64 > Install maven MAVEN3

Add nexus credentials > Manage credentials nexus nexuslogin

Check F:\ci-pro\vprociproject\Jenkinsfile in vscode

**PROJECT 12 :** Continuous delivery with Jenkins and Ansible

**Scenario** **:** let's say there is a agile software development team and they're doing a continuous development of the product so they're going to make continuous code changes. This code needs to be packaged and needs to be deployed to servers for testing. If the process is continuous, if the continuous code commit we need to continuously deploy artifact to servers for testing. If all the test checks out fine then we can finally deploy to production.

So after the deployment the software testing and other kinds of testing will be conducted and the test reports will be evaluated so and approval will be given to promote it to production or not. And this is going to be a process of deployment, test, deploy, test because you'll be having different environments dev, QA, UAT, production, staging. So if you have like these multiple environments before even you deployed your production then you will understand there is continuous deployment if there is continuous code commit.

When there is continuous deployment there's also continuous dependency on the operations team so different different teams will be there or different different product owners from different projects will be dependent on ops team for the regular deployments. And every project will have dev QA staging production multiple environments so there will be continuous deployment requests that will be sent to the operations team.

**Problem :** Let's see the problem. If the code deployment process is manual it's going to take a lot of time to promote your changes to production because there are multiple environments where the deployment needs to be done. If it's manual then it's going to take a lot of time doesn't matter how fast you make the code change still the deployment will be slow. And in this process of deployment there will be ticketing systems, there will be approvals, assignments, there will be different owners for different environments. Developer owns the dev environment QA owns the QA environment the software testers and productions are owned by different team monitoring team site reliability engineers. So **there will be too many you know ticketing assignments and approvals going here and there which actually is a bit problematic in an agile process. So too much dependencies on operations and build and release team for the deployment.**

**Solution :** So solution will be to first set **continuous integration process and continuous delivery process.** So for every code commit build test deploy then test now this happens for every commit and **the deployment process should be automated.** So deployment on different different environments and different environments have different owners so you can also notify them if something fails or passes successfully like developer gets notification software testers gets notification. So there will be multiple teams and also there will be multiple tools you need to integrate all the tools all the automation tools together. So continuous integration tool integrated with repositories integrated with automation tools with configuration management tools. So when you do these integrations it will remove human intervention because just writing a script to automate a task is just not enough. So let's say you deployed the artifact to the server then you need to communicate this to software testing team they will conduct their software test execution and there will be again in this process also ticket assignments and feedback and approvals. **So in order to get over from this problem you need to integrate tools together and whenever there is a problem bug in the code or in the deployment process you the process automatically stops and you fix the problem and then the process repeats again and this entire automated process is called as continuous delivery process** and in continuous delivery try to integrate all the tools that that are used in the release management process. So in this project we are going to see varieties of tools integrated together.

**Tools** :

Jenkins as our continuous integration server.

Nexus Sonatype repository to download Maven dependency and to upload our artifact to it.

Sonarqube server for code analysis

Maven to build our artifact

Git as a version control system where we'll also place our Ansible code

Slack for notification

AWS EC2 instance we will use on which we will be running Tomcat server

Ansible which is going to deploy our artifact to Tomcat server.

So Ansible will download artifact from Nexus repository and deploy to Tomcat server.

Objective : We have to do deployment automation so complete end to end automation of delivery. So if anything happens we can even repair things very quickly. If something breaks we can isolate we can find the problem right team gets the notification in the meantime to repair is really become really becomes short here. When you have complete automated release cycle then any requires that comes from the user any code change any bug fix anything can be done very quickly so developer makes a code change automatically the entire pipeline triggers and do the complete delivery and testing it right. And no human intervention so nobody get you know in or not many people get involved while the process is happening so it's going to be less disruptive.

**Architecture :**

Developers are going to make regular code changes as we're talking in an agile development process and they make the code change they commit to the local git repository and then that will be synced with a remote repository. So we're going to use github or source code is in github so whenever the new code is changed whenever there's a commit on github Jenkins will detect that automatically and fetch all the changes and then it's going to do code test it's going to do unit test and whether it's passed or failed a notification will be sent .If it passes it goes to the next level. The next level we'll have code analysis so we are going to use a check style and sonarqube scanner. These tools are going to scan the code and a report will be generated and will be uploaded to sonarqube server sonarqube server will compare it with its quality gates if you pass the quality gate the job continues the next job gets triggered if it's failed then the fail notification will be sent and the pipeline stops but if it's a success it goes to the next level where maven again is going to build the software this time maven is going to build the software in code testing.Maven is going to do unit test here it is going to build the artifact and when it builds the artifact it needs some dependency those dependency will be downloaded from nexus repository so on nexus we're going to create maven central repository so all the dependency will be downloaded when the build process is happening if this fails failure notification is sent pipeline stop but if this passes the packaging of the artifact will be done and the artifact will be uploaded to nexus repository.So now you will have a versioned artifact gone through unit test and code analysis if this is all good then our ansible will get kicked in job of ansible will be to download the artifact from nexus and deploy it to staging server and while ansible is doing that it will also make sure the right packages are installed the right services are running the configurations are correct so we can also make sure this all this happens along with the deployment process also one more thing while the deployment is happening and something fails ansible will put the kind of intelligence that you know it can roll back to the previous state so also we can do auto rollback and all that things we can do with ansible that's why doing deployment with ansible will be much more convenient so once ansible is done by checking everything and deploying to the staging server then our software test suite can get executed and do the software testing on the dev, QA, staging all the environments and then if we get the approval then we can use ansible to deploy the artifact to production server as well and this can be automatic without approval it's pre-approved or it can be with approval also in any case we have a complete automated pipeline

**Flow of execution** **:**

1.Update github webhook with new jenkins IP & resize jenkins volume

Jenkins server which you must have stopped and when you power on you get a new Jenkins IP we need to update that new IP in the github webhook we might also need to resize Jenkins volume now it is pretty simple to resize the Jenkins volume and the reason we will be doing is Jenkins volume EC2 instance has just a GB and it is now almost full or it's already full so we need to increase its size so Jenkins can store more data

*Increase jenkins root volume from 8 GB to 15 GB.In ubuntu 20 modifying volume size will automatrically modify partition size also inside the operating system.For other OS we need to login & resize the partition manually*

2.Launch EC2 instance for app01-staging

next we will launch an EC2 instance where we'll deploy our artifact this EC2 instance will be running Tomcat service which will again set up with Ansible automatically and this will be our app 01 staging

3.Jenkins prerequisites we need to install Ansible in Jenkins, we need to install Ansible plugin, we will save app SSH login credentials in Jenkins. now Jenkins will run Ansible playbook Ansible will do SSH to the app01-staging and to do that SSH we need to save its SSH credential in the Jenkins.Timestamp a variable we will check and will make sure it is correct as per our need this is to version our artifact

4.Create a separate branch for the pipeline in Source code (Git repo)

we will create a separate branch in our github repository where we'll put our Ansible code and also we need to copy the Ansible code from the v profile project repository to our own repository so I've prepared the Ansible code I have kept it in v profile project github repository we just need to download it and copy it in our own repository

5.Create DNS record in route53 for app01-staging

we'll create a DNS record in root 53 for app 01 staging so we will not use app 01 staging IP private IP we will be using its name and that name will be stored in the DNS of root 53 and you will understand why we will do that

6.Create inventory in Ansible code

we need to create inventory file in the Ansible code this inventory file will contain the app01-staging server name and the group name where we need to do the deployment

7. Update security group rules

we need to update some security group rules we need to allow 22 in app security group from Jenkins so Jenkins will do Ansible SSH.so Ansible will do SSH to app01-staging server so we need to allow port 22 from Jenkins security group. We need to also allow port 8081 in Nexus security group from app security group.So our tomcat application server app01-staging will download artifact from Nexus so we need to allow port 8081 in the Nexus security group so app server can download artifact from that

8.Write jenkins file to run ansible playbook from jenkins

after all those things are ready we will write a Jenkins file or we will update our Jenkins file to run our Ansible playbook from Jenkins so we basically prepare the Jenkins file code to run Ansible playbook and

9. Create a pipeline in jenkins & test it

we will create a pipeline in Jenkins run this Jenkins file and test it this is for a staging environment

10. Update ansible code in the prod branch

once we test the staging pipeline we will do the same thing for production we will update the Ansible code in production branch and

11.Launch app01-prod, dns record in route53, create inventory file

similar step as staging we will launch a app01-prod instance we will update the DNS record in root 53 we will create an inventory file for it this is same as we did for staging

12. Create a pipeline in jenkins & test it for prod

again we will create a pipeline for this will be for production create a pipeline in Jenkins and test it and then

13.Test of promoting changes to prod branch by merge

we will see how we are going to promote changes to production by doing the merge