JENKIN PIPELINE CODE

1. IT'S A DSL – Domain specific language that closely associated with groovy
2. 2 Syntax – Scripted and declarative
3. AGENT , TOOLS , ENVIRONMENT,STAGES (INSTALLATION STEPS AND POST Steps)

HOW TO LAUCH THE JENKINS

1. web search jenkin installation steps –linux and Ubuntu installation steps
2. Copy the commands
3. Note : if we didn't give #! /bin/bash – it will not run from ec2 user data)
4. Ec2 (t2 small) --- Name:Jenkinserver , ubuntu 22 , keypair(pem),sg (new) – Allow (ssh 22 my ip , custom 8080 my ip) , user data
5. Public IP:8080 – open Jenkins
6. git bash
7. ssh downloads key pair…(SSH LOGIN)
8. curl url (to check the user data)
9. systemctl status Jenkins
10. cd /var/lib/Jenkins / ===➔ HOME DIRECTORY OF JENKINS
11. cat password path
12. enter it in password and login in jenkin
13. IP of jenkin changes with respect of ec2 ---so better use elastic ip

TOOLS IN JENKINS

1. Manage tools ---global tools -→

   JDK INSTALLATION

   1. name
   2. ssh the jenkin
   3. sudo apt update
   4. sudo apt install openjdk -8-jdk -y
   5. ls /usr/lib/jvm/ ---- here you have the all the versions of java you installed
   6. get the path **ls /usr/lib/jvm/ jdk 8**
   7. **give it in the java home of jdk and uncheck the automatic install**

   MAVEN INSTALLATION – NAME AND INSTALL AUTOMTTIC

HOW TO DO A SAMPLE PIPELINE - IN JENKINS AFTER DOING THR ABOVE STEPS DO THIS

1. Manage Jenkins -→ Manage plugin --→ Available -→ install
    - Pipeline utility steps
    - Pipeline Maven Integration

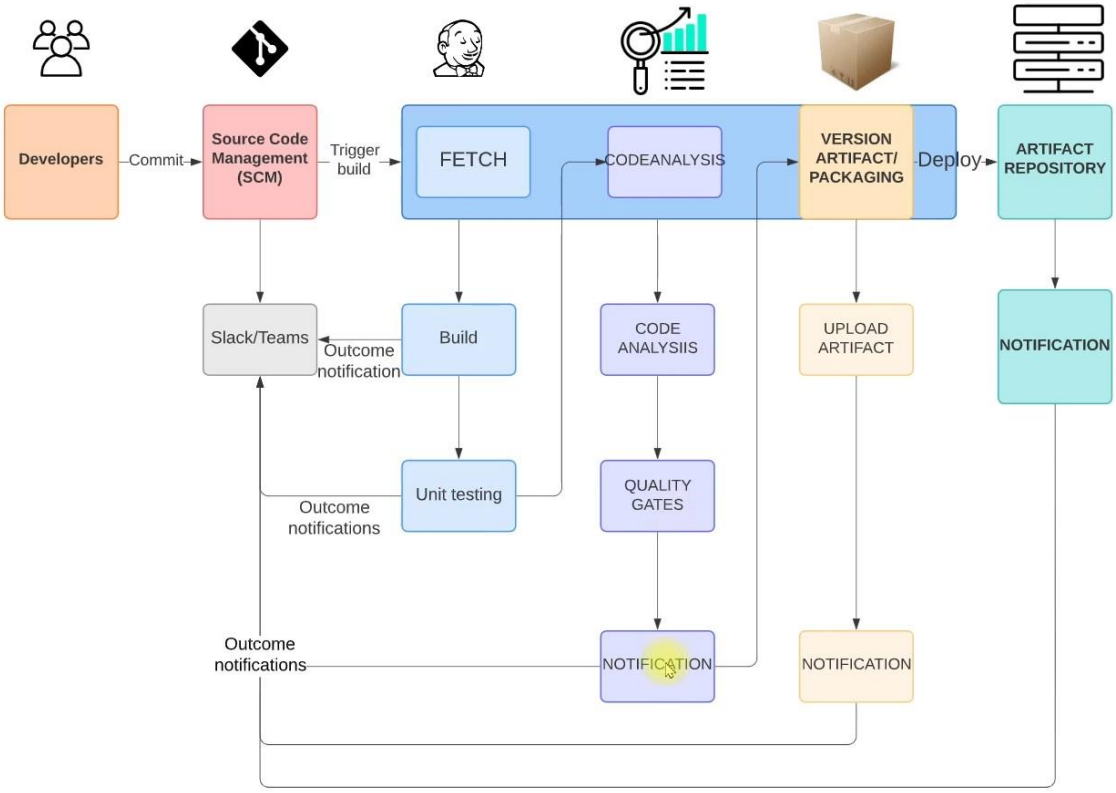
2. Write the sample code
3. New item--→ Pipeline -→ paste the code or Script from scm
4. Build now and by clicking workspace you can see the all files
5. After build we can see the Vprofile.war in target

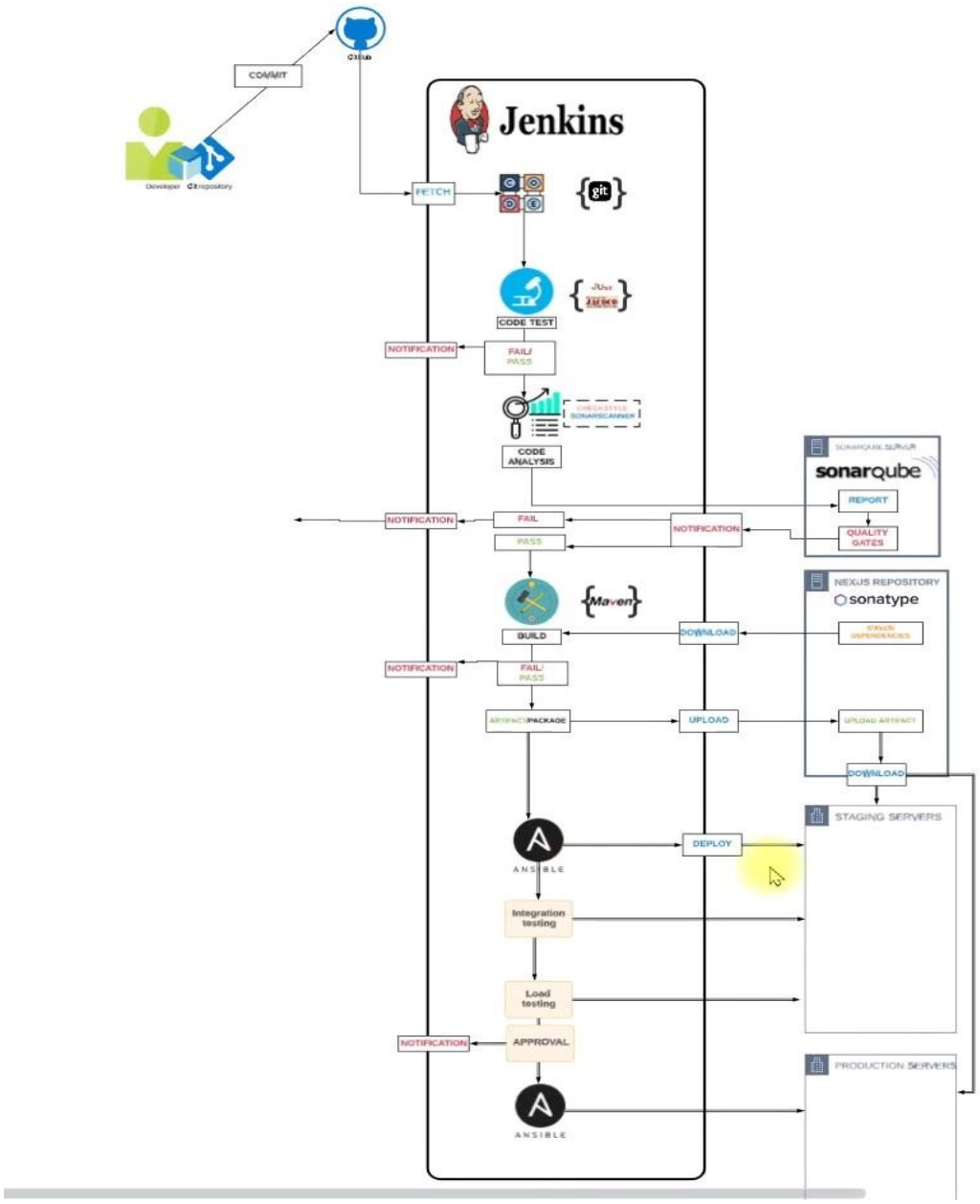
NOTE:

1. Snippet Generator – we can generate script

Tools used:

1. JENKINS – CI SERVER
2. GIT – VERSION CONTROL STSTEM
3. MAVEN – BUILD TOOL
4. CHECKSTYLE – CODE ANALYSIS TOOL
5. SLACK – NOTIFICATION AND EMAIL
6. SONATYPE NEXUS (ARTIFACT /SOFTWARE REPOSITORY) – TO STORE OUR ARTIFACT AND ALSO TO DOWNLOAD DEPENDENCY FOR MAVEN
7. SONARQUBE – CODE ANALYSIS SERVER – CHECK STYLE
8. EC2
9. DOCKER – BUILD ARTIFACT AND PUBLISH IMAGE TO ECR
10. ECR
11. ECS – HOSTING DOCKER CONTAINERS
12. AWS CLI – WHICH WILL RUN FROM JENKINS FETCH the latest image from Amazon ACR and deployed it on ECS Cluster


Developers
Commit
Source Code Management (SCM)
Trigger build
FETCH
CODEANALYSIS
VERSION ARTIFACT/ PACKAGING
Deploy
ARTIFACT REPOSITORY
Slack/Teams
Outcome notification
Build
CODE ANALYSIIS
UPLOAD ARTIFACT
NOTIFICATION
Unit testing
Outcome notifications
QUALITY GATES
Outcome notifications
NOTIFICATION
NOTIFICATION


COMMIT
Jenkins
FETCH
CODE TEST
NOTIFICATION
FAIL/
PASS
CODE
ANALYSIS
SONARQUBE SERVER
sonarqube
REPORT
QUALITY
GATES
NOTIFICATION
FAIL
PASS
NOTIFICATION
NEXUS REPOSITORY
sonatype
BUILD
DOWNLOAD
NOTIFICATION
FAIL/
PASS
ARTIFACT/PACKAGE
UPLOAD
UPLOAD ARTIFACT
DOWNLOAD
STAGING SERVERS
ANSIBLE
DEPLOY
Integration
testing
Load
testing
NOTIFICATION
APPROVAL
PRODUCTION SERVERS
ANSIBLE

FLOW OF STEPS

1. Login into the AWS
2. For Jenkin ,nexus and sonar qube)
   - Create SG
   - Create the key pair
   - Create the instance with userdata
3. Post Installation
   - Jenkin setup and plugins
   - Nexus setup and Repository setup – From where we download the dependencies for maven and upload our artifact
   - Sonarqube login test
4. GIT
   - Create repo and migration code
   - Integrate github with VS code and test it – Write the code for Build
5. Build job with Nexus Integration
6. Git hub Webhook – To trigger your job whenever you make commit in repo
7. Sonarqube Server Integration Stage
8. Nexus Artifact upload stage
9. Slack notification

SG AND KEYPAIR SETUP

Create keypair – vprofilejenci (pem)

SG –

1. **jenkinssgci**
   - **custom TCP/22/MY IP**
   - **Custom TCP/8080/IPV4**
   - **Custom TCP/8080/IPV6**
   - **Custom TCP/8080/sonarsg**
2. **Nexussg**
   - **custom TCP/22/MY IP**
   - **Custom TCP/8081/myip**
   - **Custom TCP/8081/jenkinsg**
3. **Sonarsg**
   - **custom TCP/22/MY IP**
   - **Custom TCP/80/myip**
   - **Custom TCP/80/jenkinsg**

SERVER/EC2 INSTANCE SETUP

1. For instance user data : take from githubrepo (by switching branch to ci Jenkins)
2. Jenkin script explain
    - command to install jenkin and store the repository key
    - With echo command we are going to set the repository URL in the repository file
      So the ec2 will have the access to jenkin repository
3. **Nexus script**
    - URL from which we download
    - Extract it and then we copy that nexus data to /opt/nexus folder
    - we add user nexus and give permission to this folder for nexus user and the group
    - 17 th line – which the system file so we can start and stop nexus service with this command
4. **Sonarqube to access and store its data on postgres database**
5. **Create instance -→**

JENKIN

1. Jenkinsserver
2. Ubuntu 22
3. t2 small (If you change the instance Type Public IP Also changes)
4. Select the key pair created
5. jenkinssg
6. copy the user data of Jenkins
7. launch instance

NEXUS

8. Nexussserver
9. Centos ami 9
10. t2 medium (If you change the instance Type Public IP Also changes)
11. Select the key pair created
12. nexusssg
13. copy the user data of nexus
14. launch instance

SONAR

15. Jenkinsserver
16. Ubuntu 22
17. t2 MEDIUM (If you change the instance Type Public IP Also changes)
18. Select the key pair created
19. Sonarssg
20. copy the user data of Jenkins

21. launch instance

Note

8. **ls /usr/lib/jvm/ ---- here you have the all the versions of java you installed**
9. **/var/lib/Jenkins/ ------ here Jenkins stores the all the data**
10.

POST INSTALLATION STEPS

JENKINS

1. **Copy the Public IP of Jenkins ec2 --→ git bash-→ ssh –I Downloads/ vprofilejenci.pem Ubuntu @ Public IP**

   (File where the Fingerprint of SSH Get stored – is .ssh/known_hosts

2. **sudo –i**
3. **systemctl status Jenkins**
4. **public ip:8080**
5. **cat the path shown in browser and paste the browser and login to Jenkins**

6. **Manage Jenkins --→ plugins --→available plugins -→ install all**
   - **maven integration**
   - **github integration**
   - **nexus artifact uploader**
   - **Sonarqube scanner**
   - **slack notification**
   - **Build timestamp**

NEXUS

1. **ssh –I Downloads/ vprofilejenci.pem ec2-user @ Public IP of nexus**
2. **sudo –i**
3. **systemctl status Jenkins**
4. **public ip:8081**
5. **cat the path shown in browser and paste the browser and login to**
6. **Repository -→ maven hosted -→Name :vprofile-release (Same name as maven configuration) --→ create**
7. **Repository -→ maven proxy-→Name :vpro-maven-central(Same name as maven configuration)----→ BUT WHERE WILL NEXUS WILL DOWLOAD THE DEPENDENCIES FOR THAT --→Url (https://repo1.maven.org/maven2/) --→ create (purpose : to store dependencies- maven will download from here)**
8. **Repository -→ maven hosted -→Name :vprofile-snapshot (Same name as maven configuration) --→version policy (snapshot)---- create**

9. **Repository -→ maven group -→Name :vprofile-maven group (Same name as maven configuration) --→-→add member repository all 3 we created → create**

SONARQUBE

1. **Copy Public IP of sonar ec2 and browse**
2. **login as (admin , pass : admin)**

GIT CODE MIGRATION

1. **Open the git bash and config – To Set access to Github account**
   - **git config - - global user.email "jegan7798@gmail.com"**
   - **git config - - global user.name "jegan7798gmail"**
2. **cat . gitconfig**
3. **ssh login to github account – we need the keys to login**
4. **ssh-keygen (Private key and public key are generated)**
5. **we need to store public key and store it in github account so our private key will match with the public key on github account and give the access**
6. **cat publickey and copy it**
7. **Go to github account ---→ setting ---→ssh and gpg keys --→new ssh keys (givename and paste)**
8. **Click on the fork and create new**
9. **Clone it to the local repository using ssh url of git and push it changes – make sure that branches are correct**

Note

Clone vprofile repo with branch name

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

mv vprofile-project <repositoryname>

cd <repositoryname>

Replace the remote url

git remote set-url origin git@github.com:imnowdevops/<repositoryname>.git

cat .git/config

git branch -c main

git checkout main

git push --all origin

code .

BUILD PIPELINE – TO BUILD ARTIFACT FROM SOURCE CODE

To build artifact we use maven command and maven needs a dependency which is jdk

1. **Manage Jenkins --→ Tools-----→Based on the tools installed its going to show --→ add JDK (OracleJDK11) , Uncheck the installed automatically and -→ add JDK -→ (OracleJDK8)**
2. **Get the Public Ip of Jenkins --→ go to git bash**
3. **sudo –i**
4. **ls /usr/lib/jvm/**
5. **its shows the jdk11**
6. **apt update && apt install openjdk-8-jdk –y**
7. **iis shows the space bar --→ Enter ok ok**
8. **ls /usr/lib/jvm/**
9. **It now shows jdk 8**
10. **/usr/lib/jvm/name of jdk 11 / 8 – copy the path and give it in the jdk11/8 path on Jenkins respectively**
11. **add maven (MAVEN3 , install automatically and save**
12. **Now in our pipeline code , maven is going to download dependencies from nexus and in order to authenticate nexus from maven – we need to add nexus credentials in Jenkins which we will mention in pipeline**
13. **manage Jenkins -→ Credentials --→system-→global credentials -→ add (User name: admin and admin123 , id and description = nexuslogin-----→Create**

WRITE THE FILES

File to be ready – Commit and push

1. **Jenkins file**
2. **Pom.xml**
3. **Setting.xml**

JENKINS PIPELINE

1. **Jenkins --→ New Item (Name:vprofilecipipeline)---→Pipeline ---→Scroll down -→Pipeline Script from SCM -→ Git --→ Give the repo url --→ Copy the shh url of your repo----→ Credentials (Add Jenkins)-→ SSH User name and private key and give the privatekey we created --→ and choose the credentials**

(Note : It still shows the error because ---we need ro ssh to the Jenkins)

2. **ssh –I Downloads/ vprofilejenci.pem UBUNTU @ Public IP of JENKINS**
3. **sudo –i**
4. **su -jenkins**
5. **git ls-remote –h url of ssh of git HEAD (By giving this it stores the identity of github into Jenkins user)**
6. **cat .ssh/known_hosts**
7. **Then go to Jenkins**
8. **In branch give = ci-jenkins and save**
9. **Build now**
10. **MAKE SURE THAT ALL FILES IN VSC CODE ARE SAME VARIABLE**
11. **SUCESSS ----IT DOWNLOADED THE DEPENDENCY FROM NEXUS , BUILD THE ARTIFACT AND IT STORED THOSE DEPENDENCIES IN THE JENKIN ITSELF**
12. **Go to sonarqube and check all the files that are available in groups**

WHENEVER WE MAKE THE COMIT IT SHOULD BE BUILD AUTOMATICALLY

GITHUB WEBHOOK

1. **Make sure the SG of Jenkins – allows port 8080 from anywhere (IP v4 6)**
2. **Copy the URL of Jenkins –(It will Change accordingly EC2 Power off and on – Solution:Use Elastic IP) --→ Go to github ---→ Go to your repository -→ Setting -→ Webhooks --→ add → Paste the url of Jenkins/github.webhook/--→ Content type: JSON (So whenever thre is a commit ,its going to send a JSON Payload and keep on saying commit and based on the event you can select) -→ Just the push event and active -→ Add Webhook**
3. **Click on the web hook and select the recent delivery green tick (Means that is delivered to jenkin and it responded back) and click on Redeliver**
4. **Go to configure : check mark the Github hook trigger for GITscm Poling and save**
5. **Add some lines in Jenkins file (Unit test and code analysis)--→ Commit and push**
6. **Its now automatically trigger and build**
7. **Click on any job and workspaces and see it all there**

CODE ANALYSIS WITH SONAR QUBE

1. **job → Workspace → Target → its shows the check results**
   (In order to Present it in the Human Readable format -→ WE use it)

<u>NOW WE NEED TO WRITE CODE – TO GENERATE REPORTS AND UPLOAD IT TO THE SONRQUBE SERVER</u>

For that we need: in jenkins

1. **Sonar scanner tool**
2. **Sonar qube information in Jenkins – so that Jenkins knows were to upload**
3. **Jenkins -→ Manage Jenkins -→ Global tool Configuration -→ Add Sonar qube scanner (Name: Sonarsanner), Version: 4.7 -→ Save**
4. **Go to sonar server dashboard -→ my account → Security → Give any name and generate token**
5. **Jenkins -→ Manage Jenkins -→ Configure system (We need to store the sonarqube information in Jenkins --→ Sonarqube server (Name:sonarserver, Private IP of sonar server)----→ server auth token -→ Jenkins (Kind: secret text , paste the token)-→Save**
6. **Now write the code → Commit and push--→ its shows results in sonar and Jenkins**

<u>HOW TO CREATE OUR OWN SONAR QUALITY GATE</u>

A quality gate is a milestone in an IT project that requires that predefined criteria be met before the project can proceed to the next phase. Designed to provide benchmarks for quality standards, these gates are commonly used throughout application or software development projects

1. **Sonarqube -→ setting → Quality gate -→ create → (Name: vprofileQG)→ Add conditions -→ on overall code (bugs 100) and add it to the vprofileproject**

Concepts :

Git- webhook -→ Jenkins -→ Jenkins upload the result to sonar

But how the sonarqube knows where the Jenkins and how to send information

1. **Sonarqube -→ vprofile project --→ Project setting -→ Webhook --→ (This is for sending the result to Jenkins) ---→ Create WEbhook-→ (Name:jenkinswebhook , URL : http://private ip of Jenkins:8080 sonarqube-webhook --→ Create**
2. **Write the code and commit push**
3. **Its builds**

LAST STAGE -→ PUBLISH ARTIFACT TO NEXUS REPO

1. **we need to add the code (nexus private IP)**
2. **and we can download the artifact from the sonarqube --→ With the link given**

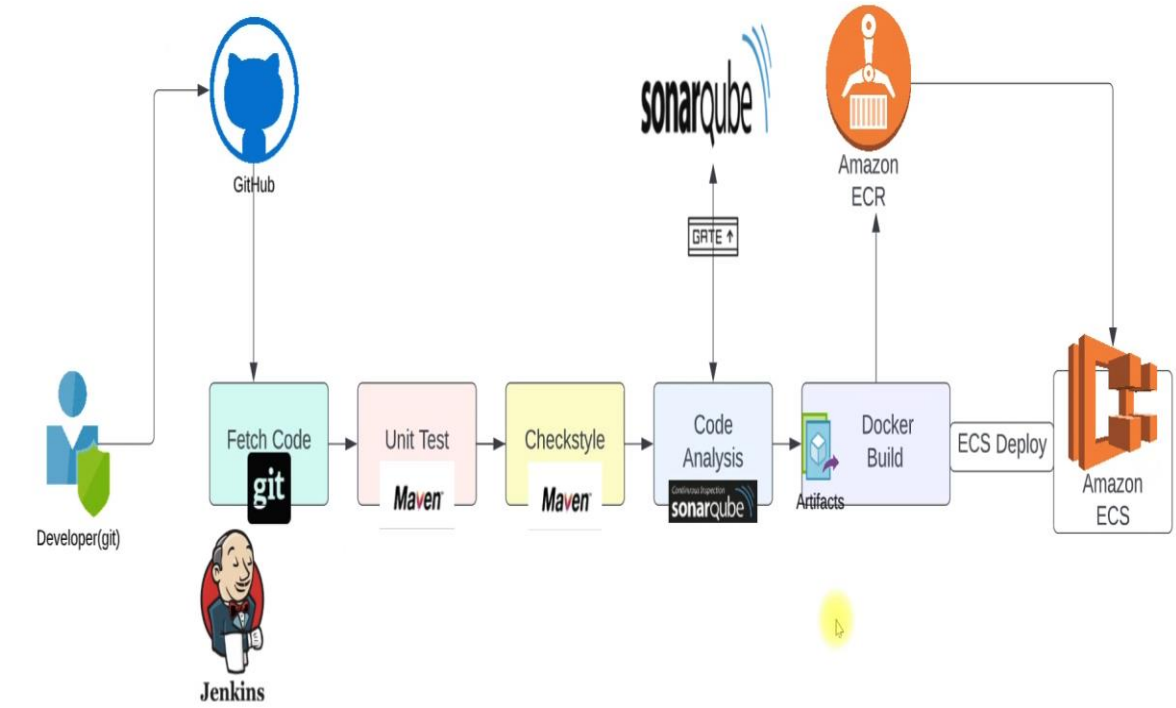
SLACK NOTIFICATION - POPULAR COLLABRATION TOOL

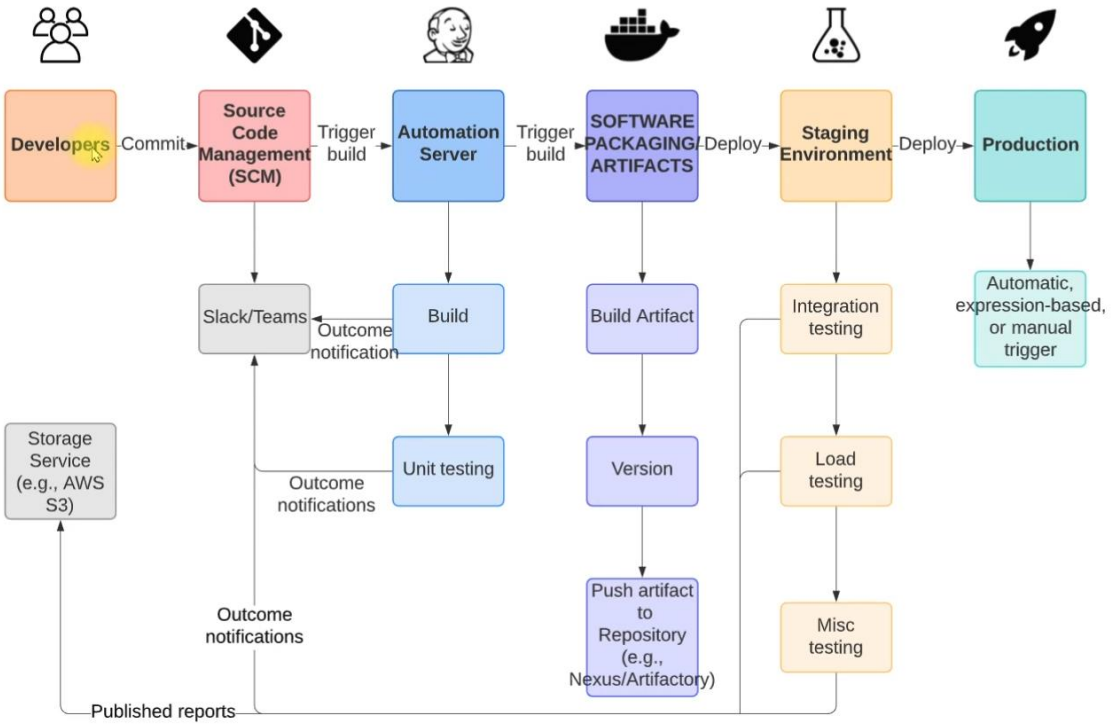
1. **Create the Channel in that like we have groups and then we integrate with Jenkins**
2. **we update the code to send notification**
3. **Search for slack login**
4. **Slack has the workspaces -→ may be a team or company where all are connected to and it has different channels like we have groups like whatsapp**
5. **Create Workspace → Name:Vprofilecicd , devopscicd , add**
6. **open the app -→ Create channel (Name: JenkinsCICD and create)**
7. **In order to Jenkins authenticate to your workspace we need create a token in slack and for that we need to add app in slack account -→ Google; Slackapp -→ search jenkins and add slack and choose the channel and copy the token which shows and save**

JENKINS

1. **manage Jenkins --→ configure system --→ slack and give the work space name and token as secret tesxt and channel name and save**
2. **Now we will add in pipeline code**
3. **Commit and push -→ it build and slack app shows the notification and it gives the URL that we can check in Jenkins**

CONTINEOUS DELIVERY - TEST - PRE READY FOR PRODUCTION


GitHub
sonarqube
Amazon
ECR
GATE
Developer(git)
Fetch Code
git
Unit Test
Maven
Checkstyle
Maven
Code
Analysis
sonarqube
Artifacts
Docker
Build
ECS Deploy
Amazon
ECS
Jenkins



Flow of steps

1. **Update github webhook with new JenkinsIP**
2. **Copy Docker images from vprofile repo to our repo**
3. **Prepare the 2 Separate Jenkin file for staging and production in source code**
4. **AWS**
    - **IAM**
    - **ECR Repo Setup (to Jenkins store Docker images)**
5. **Jenkins Steps**

    Install Plugins

    - **Amazon ECR**
    - **Docker Build and publish Plugin**
    - **Pipeline : AWS Steps**
    - **Credentials**
6. **Install Docker Engine (To build)and AWS CLI (To: When we deploy images on ECS Cluster ,we will run the CLI Commands) on Jenkins**
7. **Write Code For Jenkinsfile for build and publish image to ECR**
8. **Create ECS Setup**
    - **Cluster**
    - **Task Definition**
    - **Service (Going to create the docker container for us , its going to fetch image from ECR and run the container and also provide Load balancer**
9. **Code for Deploy Docker image to ECS**

10. **Repeat it for prod ECS Cluster**

Branches and Webhook

Prerequisite

1. **jenkinssgci**
    - **custom TCP/22/MY IP**
    - **Custom TCP/8080/IPV4**
    - **Custom TCP/8080/IPV6**
    - **Custom TCP/8080/sonarsg**
2. **Nexussg**
    - **custom TCP/22/MY IP**
    - **Custom TCP/8081/myip**
    - **Custom TCP/8081/jenkinsg**
3. **Sonarsg**
    - **custom TCP/22/MY IP**
    - **Custom TCP/80/myip**
    - **Custom TCP/80/jenkinsg**

WEBHOOKS

1. **Copy the URL of Jenkins –(It will Change accordingly EC2 Power off and on – Solution:Use Elastic IP) --→ Go to github ---→ Go to your repository -→ Setting -→ Webhooks --→ add → Paste the url of Jenkins/github.webhook/--→ Content type: JSON (So whenever thre is a commit ,its going to send a JSON Payload and keep on saying commit and based on the event you can select) -→ Just the push event and active -→ Add Webhook**
2. **Click on the web hook and select the recent delivery green tick (Means that is delivered to jenkin and it responded back) and click on Redeliver**

WE NEED TO PREPARE OUR CODE

CREATING PRODUCTION PIPELINE AND STAGING PIPELINE WITH NEW FOLDER CI CD Jenkins

1. We need to add Docker file and we need to arrange the Jenkin file for staging and production environment in our source code
2. Go to vprofile project repo -→ branch:docker--→ download the zip file
3. git clone -b ci-jenkins https://github.com/devopshydclub/vprofile-project.git
4. select the same folder (if u have it its fine)
5. Copy the dockerfile from the zip to here
6. cd vprofileproject/
7. ls

8. git checkout ci-jenkins (switch to ci-jenkins)
9. git checkout –b cicd-jenkins (creating the new branch and switching to that)
10. Copy the dockerfile from the zip to here
11. ls
12. you are now in ci-cd branch
13. mkdir stagePipeline
14. mkdir prodPipeline
15. ls
16. cp Jenkinsfile stagePipeline/
17. cp Jenkinsfile prodPipeline/
18. git rm Jenkinsfile
19. cat .git/config (shows the ci-jenkins)
20. git add.
21. git commit –m "preparing cicd branch"
22. git push origin cicd-Jenkins (new branch is created in githubrepo)

AWS IAM

IAM →user-→Name:cicdjenkins--→ access key acesss-→ SAVE → Download credentials

permis:

- EC2 Container Register full access
- Amazon ECS Full ACCESS

AWS ECR (Store Docker images)

1. Create -→ private → Name:vprofileappimage-→ create --→ it gives url

JENKIN CONFIGURATION

1. **Manage plugin ---→ Available ---→ install**
   - **Docker pipeline**
   - **Cloud bees docker build and publish**
   - **AWS ECR**
   - **pipeline aws steps**
   - **aws credentials**

2. **manage plugin ---→ manage credential --→jenkins(new)==➔global credential --→ Add -→ (AWS Credential ,copy the access key of IAM and give , secret key also -→create**

CLI AND INSTALLING DOCKER ENGINE

1. **ssh to Jenkins , sudo –I ,**
2. **apt update**
3. **apt install awscli –y**
4. **search for docker installation steps on Ubuntu**
5. **install the dependencies (Change accordingly)**

   ```
   # Add Docker's official GPG key:
   sudo apt-get update
   sudo apt-get install ca-certificates curl gnupg
   sudo install -m 0755 -d /etc/apt/keyrings
   curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
   sudo chmod a+r /etc/apt/keyrings/docker.gpg

   # Add the repository to Apt sources:
   echo \
     "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
     "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
     sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
   sudo apt-get update
   ```

6. **sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin (change accordingly)**

7. **Docker image can be only build by root user**
8. **su –jenkins**
9. **docker images**
10. **usemod –aG docker Jenkins**
11. **id Jenkins**
12. **systemctl restart Jenkins / reboot**
13. **logout (from Jenkins)**

TEST THE CI PIPELINE – CONFIRM THAT WORKS

CODE TO BUILD DOCKER IMAGE AND UPLOAD IT TO ECR

1. **cd /f/folder/folder where ci cd /**
2. **git status (To verify)**
3. **code .**
4. **VSC -→ Docker file-→ Dowloaded and make it as jenkin file ----commit and push**
5. **Then we create the new pipeline for the CI CD**
6. **Jenkins -→ Pipeline =➔ Name: Jenkin cicd pipeline-→ create-→ select Git hub hook trigger for GITscm Polling -→ pipeline from scm (GIT,url) and already git credential stored here (so take that)--→Branch: cicd-jenkin (as per)--→Path:** stagePipeline/ Jenkinsfile ---→ create -→ BUILD NOW

AWS ECS SETUP

Where we can deploy the ECR docker image which is ready to deploy – here are the different options

1. **Docker Engine**
2. **Kubernetes**
    - **Standalone**
    - **EKS**
    - **AKS**
    - **GKE**
    - **Openshift**

DEPLOY IT ON ECS

MAKE SURE ON THE SAME REGION AS JENKINS SERVER

1. ECS---→ Create cluster (For staging)
    - Name:vprofilestage
    - choose the all the subnets
    - monitoring the on
    - Tags : Name/vprostage
2. go to created one --→ Task→ Task def → create
    - name:vprotaskstage
    - Container 1 : Vproapp and Imageurl : ecr url
    - port 8080 (Tomcat runs)
    - app env: fargate
    - memory : 1cpu /2 gb

3. **Services -→ deploy -→ deploy**
    - **Launch type**
    - **task def: choose manually and choose created task**
    - **Service name :vproappstage**
    - **create own SG : Name:vprostage**
    - **http/80/anywhere**
    - **Application load balancer and name it**
    - **Targetgroupname:vprosg / protocal : HTTP**
    - **Health check : /login**
    - **Health check grace period : 30**

4. **Ec2 --- need to change the SG 8080**
5. **EC2 -→ Target group (which we created on ECS) --→ Health check -→edit-→Advanced health check → override → 8080**
6. **Ec2-→SG(Created one ECS) ---> (Custom tcp /8080/anyipv4 and also 6)**

NOTE:

1. Services – To expose container to the outside world or for inter communication , provide load balancer and manage container (ITS LIKE ASG FOR CONTAINERS)
2. Task – container

We create the TASK-→ TASK DEFINITION -→ That we use in service to create and manage containers

PIPELINE FOR ECS

1. ECS –OURS---SERVICE –CREATED SERVICE CLICK -→Networking →DNS OPEN
2. ON GOING TO TASK U CAN SEE – Container logs

TO MAKE IT AUTOMATTICALY

1. Copy IP of jenkin instance--→ GIT HUB--→setting → change it

UPDATE

1. change to cd of ours
2. git pull
3. upload the file to the jenkinfile-→ Comit and push -→ jenkin aumattically build and test===→ go to ecs it has new task

FOR PRODUCTION

1. Create ECS CLUSTER FOR PRODUCTION
2. Create task def
3. **EC2 -→ Target group (which we created on ECS) --→ Health check -→edit-→Advanced health check → override → 8080**
4. **Ec2-→SG(Created one ECS) ---> (Custom tcp /8080/anyipv4 and also 6)**
5. Deploy
6. git checkout –b prod (NEW BRANCH)--→ SHOWS IN VSC -→ PUBLISH →IT GOES TO GIT--- > CHANGE THE CONTENT -→ COMMIT AND PUSH
7. JENKIND→ NEW ITEM ➔ NEW PIPELINE