**CICD & GIT (40 & 41 note)**

**CICD**

**CI:-**

**clone**

**build**

**scan the code**

**static scanning --> sonarqube**

**sast --> fortify, coding is based on security standards**

**dast --> web inspect, app will be tested with all vulnerabilities**

**open source scan --> nexus IQ, blackduck scan thrird party libraries**

**image scanning --> ECR scanning, scanning docker images**

**create artifact, generally zip file**

**store the zip file --> nexus, jfrog, s3**

**unit testing --> junit**

**CD:-**

**application infra --> developers or devops engineers**

**-----------------**

**create instance**

**deploy the latest version**

**stop the instance**

**take ami**

**refresh autoscaling**

**do the functional testing --> selenium**

**shift-left**

**------------**

**GIT**

**--------------**

**main/master --> depicts/points the code running currently in production**

**create a copy of file --> do the changes --> test those changes --> then do it main file**

**create another branch from main/master**

**--------------------------------------**

**do the changes here**

**run CICD --> deploy in DEV**

**test the application**

**merge into master --> deploy in PROD**

**we write functions --> functionality**

**input and output**

**code coverage --> how much unit testing covered**

**GIT (41 note)**

**master --> feature**

**PR from feature --> master**

**comments**

**merge/rebase --> extra commit**

**merge commit triggers DEV deployment**

**main**

**feature branches are from main branch**

**we did PR**

**git content tracker**

**SHA code --> 40 char universal unique ID for the content**

**git is a key/value pair**

**commit ID --> content is the value**

**if you change the content commit ID will change**

**echo "" | git hash-object --stdin**

**git cat-file commit-id -p --> you will get total information**

**merge strategies**

**-----------------**

**1. merge**

**-------------**

**when you are merging one branch to another branch**

**feature-1 --> main**

**it will create a merge commit always that have 2 parents.**

**merge will preserve the complete history, it is like chain structure**

**2. rebase**

**-------------**

**1. no extra commit**

**2. commit ids will change, rewrites the history**

**3. it will not preserve the history.**

**a. is this looking like a rebase is created and merged**

**b. is this looking a linear development happened in main branch**

**squash --> mixing them**

**microservices**

**------------**

**2 weeks**

**1 feature is developed by 1 developer**

**a developer can do 60 commits...**

**abcd433 --> project-1 have user module**

**60 commits --> squash into single commit**

**djf788 --> He completed catalogue module**

**dfg67364 --> cart module**

**1. if a branch is developed by multiple developers then**

**prefer merge**

**branching strategy**

**---------------------**

**main**

**feature --> apart from main anything called as feature branch**

**main is long lived branch, feature is short lived branch**

**developers create feature branches, once they complete development they will raise PR and merge/rebase into main branch**

**feature branch, before raising PR... let's do CICD in DEV environment. if success in DEV then you can raise PR**

**then rebase into main...**

**code is same across all environments, but configuration is different**

**configuration should be detached from code, we are using SSM parameter store**

**we got into main branch --> QA, SIT, UAT, PRE-PROD, PERF, PROD**

**feature --> DEV, QA --> success**

**main --> merge --> new commit ID got created**

**when commit id is changed --> code is changed --> so doing once again in QA**

**fast-forward merge --> build once in DEV and run anywhere**

**-------------------**

**feature --> DEV, QA --> success**

**main --> but same commit ID whatever tested in feature branch**

**42 Note**

**merge conflicts**

**what is conflict?**

**if git finds diff code in the same line number**

**main**

**------**

**main --> conflict --> developer-1 created it**

**developer-2**

**-----------------**

**main --> developer-2**

**then he should understand main branch moved forward**

**he need to pull those changes in local, resolve conflicts with other developer-1**

**pull before push**

**Jenkins --> CICD**

**---------------**

**continous integration tool**

**1 person --> 1 acre of agriculture**

**100 acres --> you need lot of resources**

**owner --> employ resources --> distribute the work**

**master --> slave**

**master --> agent**

**1. master asking agent to work**

**2. employee coming to master and asking for work**

**webhook**

**-----------**

**if developer pushes the code to git, then we want the pipeline to run automatically**

**git --> Jenkins**

**43 Note**

**CICD**

**-------**

**Project infra should be ready**

**catalogue app should be in git**

**Jenkinsfile**

**CI**

**-----**

**clone the code**

**unit testing**

**scans**

**sonar scan**

**SAST**

**DAST**

**Open source scanning**

**docker image scanning**

**dependencies install**

**build --> zip file**

**we need to push to Nexus**

**CD**

**----**

**deploy to dev**

**functional test cases**

**publish the results**

**git --> repository --> code --> dont store artifacts**

**nexus --> repository --> artifacts --> dont store code here**

**maven 2 format**

**--------------**

**1000**

**firstname**

**firstname+lastname --> same**

**firstname+lastname+dob --> may be possible**

**project, modules, versions**

**project --> roboshop**

**modules --> catalogue**

**version --> 1.0.0, 1.0.1, 2.0.0**

**group id --> com.roboshop**

**artifact id --> catalogue**

**version --> 10.0.0**

[**http://100.26.147.241:8081/repository/catalogue/**](http://100.26.147.241:8081/repository/catalogue/)

**44 Note**

**1.0.0 --> semantic version**

**http://3.237.181.150:8081/repository/catalogue/**

**http://172.31.5.95/repository/catalogue/com/roboshop/catalogue/1.0.0/catalogue-1.0.0.zip**

**jenkins --> nexus**

**nexus artifact uploader**

**nexus URL, authentication**

**create a server**

**provision it using ansible**

**stop server**

**take AMI**

**create launch template version**

**refresh autoscaling**

**download artifact from nexus**

**download specific version**

**ansible**

**--------**

**nexus location**

**artifact version**

**from terraform you should send artifact version to ansible**

**jenkins have application version, it should send that version to terraform**

**45 Note**

**Types of scanning:-**

**Static source code analysis**

**sonarqube**

**Static application security testing**

**fortify**

**dynamic application security testing**

**web inspect**

**open source library scanning**

**nexus iq**

**docker image scanning**

**twistlock, ECR scanning**

**dependencies installatio**

**build**

**all type of scanning**

**unit testing**

**nexus upload**

**quality gate**

**-------------**

**0 bugs**

**0 code snells**

**security rating A**

**code coverage 80**

**commit-A**

**commit-B**

**commit-B - commit-A = new code**

**git --> multiple branches...multi branch pipeline**

**Language, Deployment platform**

**NodeJs, AWS VM/Docker/Kubernetes/PCF/On-prem VM**

**HDFC --> 200 projects and many microservices**

**nodejsvm pipeline --> don't duplicate the pipeline code**

**reuse, implement best standards and force others to use**

**nodejsvm pipeline --> implement best standards, reuse, centralised**

**jenkins shared library --> treat your pipeline as library, use it wherever you want**

**process**

**-----------**

**1. create a feature branch in catalogue**

**2. configure multi branch pipeline**

**3. configure jenkins shared library**

**1. I created a repo and added pipelines there**

**2. I added in jenkins system configuration, location and name of the library**

**nodejsvm/javavm/pythonvm/govm is a centralised pipeline, we need to send parameters**

**46 Note**

**JIRA/Service now**

**---------------------------**

**project code, app code, version**

**test results attachement**

**scanning results attachement**

**DEV, SIT, UAT --> success**

**jenkins build URLS also here**

**JIRA ID**

**CR number**

**Change management process**

**---------------------------**

**we will raise a CR --> every organisation will have their own tool to manage CR.**

**project code, application code, version, JIRA ID, approvals, date and time, deployment process, revert back process, post deployment testing.**

**NEW --> CR ID**

**Approve --> write some comments**

**Delivery manager --> write some comments and approve**

**Cleint --> approve**

**JIRA --> Jenkins integration**

**trigger PROD --> JIRA will check CR is approved or not, prod trigger time is same as time in CR.**

**Jenkins pipeline**

**DEV and PROD infra**

**DEV will do a change in source code, a pipeline will automatically trigger to deploy the application into DEV..**

**then we will deploy into PROD**

**functional testing**

**------------------**

**microservices**

**1 microservice is handled by 1 person --> Developers are writing functional test cases**

**Once DEV deployment is over, either developer/tester have to provide the scripts and command to perform functional testing,**

**47 Note**

**1. make sure dev and prod infra is ready. get the vpn connected**

**2. catalogue**

**create catalogue repo**

**create jenkinsfile**

**point this jenkinsfile to shared libraries. make sure you have shared libraries configured in jenkins system configuration**

**shared library VM pipeline will be called**

**clone**

**get the version from package.json**

**install dependencies**

**unit tests**

**build**

**scans**

**sonar**

**sast**

**open source libraries scans**

**dast**

**if developer opts for deploy we can deploy**

**catalogue-deploy, it is calling terraform-roboshop-app**

**bootstrap script, we are passing environment, version and component**

**bootstrap script will clone roboshop-ansible-roles-tf and run catalogue role**

**As a DevOps team, we have Nodejs CICD ready... a new project user module is started. How can you setup CICD for it?**

**1. if a new project is started, they should have a repo exist in nexus..**