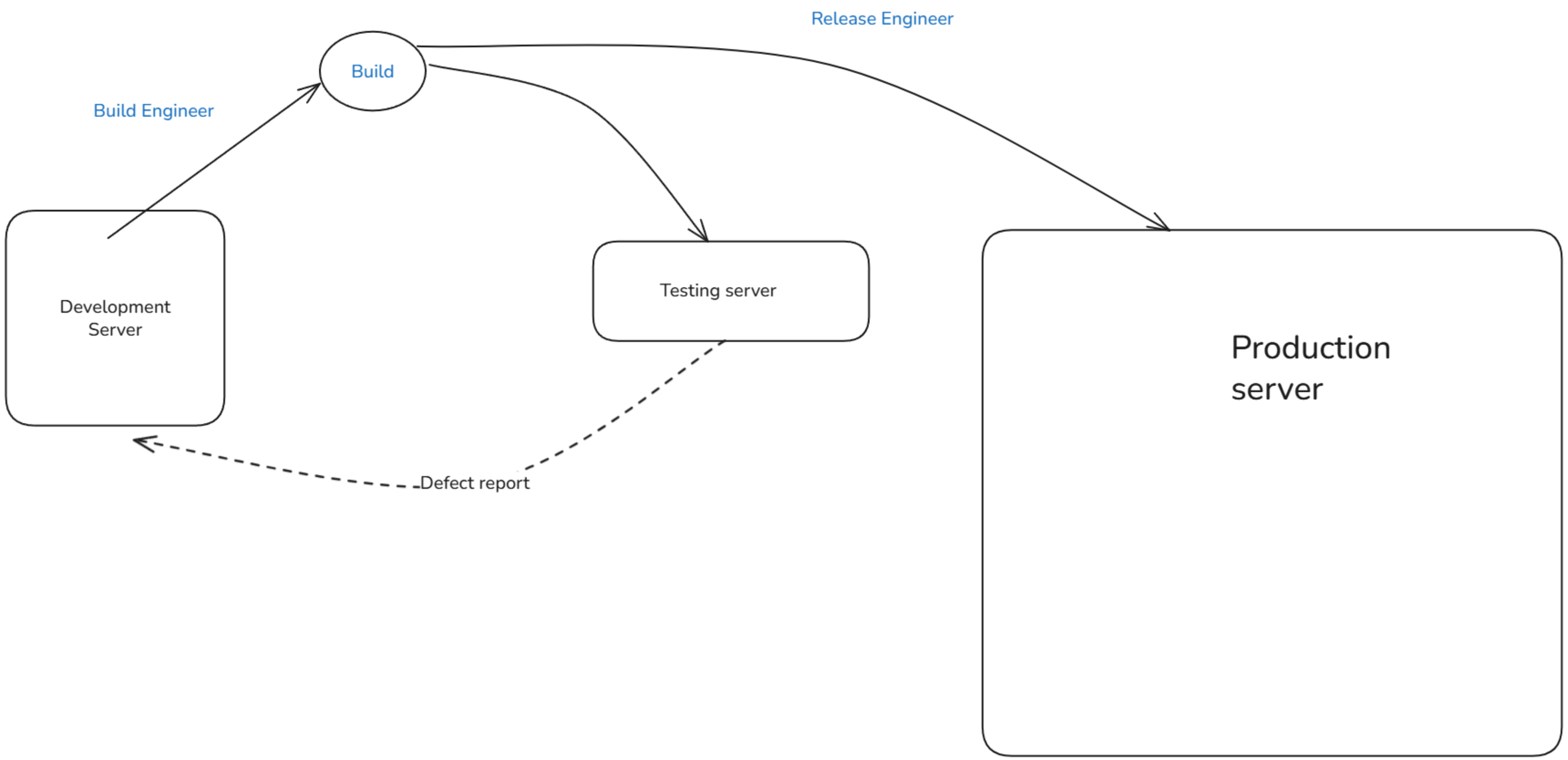
JENKINS

1. Traditional build and release process
2. Introduction to Jenkins
   - What is Jenkins?
   - What is CI, CD?
3. History and features of Jenkins
4. Alternatives tools for Jenkins
5. Installation of Jenkins
6. Dashboard Explanation of Jenkins
7. Jobs and It's type
8. Running job in windows
9. Build triggers in Windows
   - Build periodically
   - PollSCM
   - Github webhook
10. Role based Autorization
11. Installation of Jenkins in AWS
12. Jobs
13. Build Triggers
14. Run a shell batch commands
15. Pipeline and its type
   - Declarative pipeline
   - Scripted pipeline
16. Master slave Architecture

Traditional Build and Release process:



Drawbacks:

1. It is time consuming process
2. It is less efficient
3. Manual work is more
4. Less productivity

To overcome all this drawbacks we are using CI/CD tool called "Jenkins"

Introduction to Jenkins:

Jenkins:

It is free and open source CI/CD tool to automate the process
of building, testing and deployment of any software/application.

It is developed by Java language

CI(Continuous Integration):

Continuously collecting the source code and
generating the build. This process is called as
continuous integration.

CD(Continuous Delivery):

Continuously delivering the build form Development server
to Testing server is called as CD

- Preparing the build ready to be installed in production server.

CD(Continuous Deployment):

Continuously installing build into production server
is called as CD(Continuous deployment)

These process are automated

History of Jenkins:

2004: "Kohsuke Kawaguchi", he created a project called "Hudson"
to automate the process of building and testing

He was working in Sun Microsystem

2010: Oracle company acquired Sun Microsystem, so it created a conflict between
Hudson and orcale

2011: Hudson got renamed as "Jenkins"

Features of Jenkins:

1. It is free and open source
2. It is automated CI/CD tool
3. It also support extensible plugins(1800+)
4. It supports pipeline as a code
5. It is user friendly
6. It is scalable for master slave architecture.
7. It supports authentication and authorization
8. It can easily integrate with third party application.

Alternative tools for Jenkins:

1. Github action

   Powerful feature inside Github, that enables automate workflows for various actions like building, testing, deploying.

   - Here to automate the process we will be using YAML file.

   - It is not suitable for complex projects.

2. GitLab CI/CD:

   It is integrated within GitLab
   - Uses YAML based pipeline configuration.

3. TeamCity:

   It is a CI/CD server provided by "JetBrains"

4. Circle CI:

   - Cloud based CI/CD platform that automates the building, testing, and deployment process.
   - It allows developers to build, test and deploy the applications
   - It is paid version

5. Bamboo:

6. Travis CI:

Job:

It is a simple task or multiple set of task that Jenkins will execute

Jenkins are used to perform the task like building, testing and deployment

Types of jobs:

1. Freestyle project:

- It is a simplest type of job suitable for simple/basic task
- It provides an easy to use GUI

How to create a freestyle job

1. Go to dashboard
2. Click on new item
3. Give name of the project(job)
4. Select "freestyle job"
5. click on ok
6. Give the description of project
7. Set github repository URL in "source code management" by giving repo url and default branch name
8. Go to build steps
   - select "execute windows batch commands"
9. Give the commands for basic execute
   - echo "hello world"
10. click on apply and save
11. click on build now to generate the build

Generate Build using maven in free-style project

-> install "maven integration" plugin
-> need to configure maven tool in jenkins

To install maven plugins

- Go to dashboard
- click on manage jenkins
- select plugins under system configuration
- click on available plugins
- search for "maven integration" select it and click on install
- click on "Restart jenkins after installation".

To configure maven tool:

- go to dashboard and click on "manage jenkins"
- click on "Tools" under system configuration
- scroll down till the end, click on "Add Maven"
- Give the name for maven tool
- Give the path where maven is installed
(you can find the path in environmental variable)
- click on apply and save

Steps to integrate maven with freestyle project:

- make sure maven plugin is installed
- configure maven tool in jenkins
- Click on dashboard and click on "new item"
- Create a job
    - give project name
    - select job type as "freestyle project"
- scroll down to build steps
- click on build steps and select "execute windows batch command"
- write commands to generate build and "save"
- click on "Build now"
- you can see the output by clicking on "console output"

take other project and do git clone => freestyle

maven - take spring petclinic repo url and generate build
using freestyle project

2. Maven Project:

-Integrating maven with jenkins to generate builds, test, deploy

Steps to create maven project:

1. Go to jenkins dashboard
2. click on new item, and create a job
    - give project name
    - select "Maven project" under item type
    - click on "ok"
3. click on git in SCM
    - give the url of your repo
    - specify branch name
4. Build:
    - Root pom:
        pom.xml
    - Goals and object:
        clean package
5. Click on apply and "save"
6. Click on build now

Practice:

- installing jenkins in ubuntu instance
- creating freestyle and maven project
- integrating maven in freestyle project
- Build triggers and post build triggers

3. Pipeline:

Pipelines are series of stages which helps to automate the process of building, testing, deployment.

- Here pipeline scripts are written in groovy language

We have 2 types of pipeline:

1. Declarative pipeline
2. Scripted pipeline

1. Declarative pipeline:

This pipeline offers human friendly syntax and easy to understand.
It Involves some sections such as:

1. Pipeline
2. Stages
3. Stage
4. Steps

- the template name is "hello world"

Syntax:

```
Pipeline
{
agent name
Stages
 {
    stage('Stage-name')
    {
     steps{
          -----
          -----
     }
    }
 }
}
```

```
Pipeline
{
agent name
Stages
  {
    stage('Stage-name')
    {
      steps{
           -----
           -----
      }
    }
    stage('Stage-name')
    {
      steps{
           -----
           -----
      }
    }
  }
}
```

Stage 1

stage 2

2. Scripted pipeline:

It is a type of pipeline, where it is more flexible way to define CI/CD workflow by using "Groovy Scripting"

- the template name is "Scripted pipeline"

Syntax:

```
node{
  stage('stage-name'){
    -----
    ------
  }
}
```

Build Triggers:

1. Build periodically

2. Poll SCM

3. Build after other projects are build

1. Build periodically:

trigger build based on a schedule

trigger the build every 5 minutes => */5 * * * *

cron: It will help to schedule a job within the specified time interval

Syntax: cron(minute hour day-of-month month day-of-week)

(0-59)

(0-23)

(1-31)

(1-12)

(0-7)

0 and 7 are "sunday"

1 -> mon
2 -> tue

Note: Irrespective of the changes in GitHub source code, the job will get generated on a specified interval in build periodically

steps:
1. create job
2. click on "build periodically" under "triggers"
3. schedule a time using cron expression
   -(eg: */5 * * * * => to generate build every 5 mins)
4. Click on apply and save

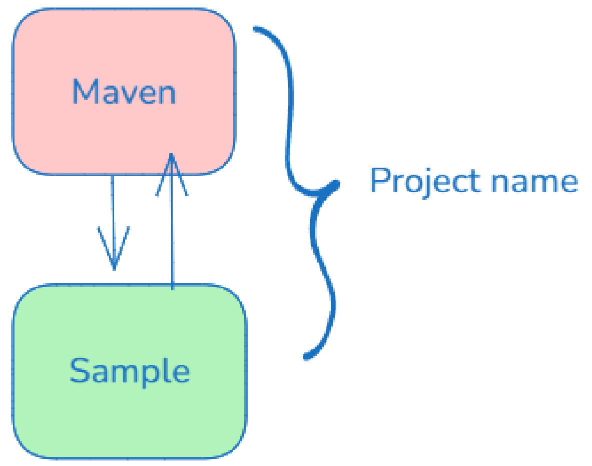
2. Poll SCM:

Build will get generated when there is a changes in the github repo and it will generate the build } Based on the time interval specified

*/5 * * * *

steps for poll SCM:
1. create job
2. click on "Poll SCM" under "triggers"
3. schedule a time using cron expression
   -(eg: */5 * * * * => to check SCM(github) every 5 mins)
4. Click on apply and save

3. Build after other projects are built:



steps for poll SCM:

1. create job
2. click on "Build after other projects are built" under "triggers"
3. In "project to watch" add the name of the upstream project
4. click on apply and save

Steps to install jenkins in AWS(Ubuntu instance)

1. launch an instance with 16GB config of instance type t2.medium
2. connect the instance to EC2 terminal
3. update the machine and install java, jenkins

```
    - > sudo apt update
       > sudo apt install openjdk-17-jdk -y
       > sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc \
         https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
         echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc]" \
         https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
         /etc/apt/sources.list.d/jenkins.list > /dev/null
        sudo apt-get update
        sudo apt-get install jenkins
       > systemctl status jenkins
```

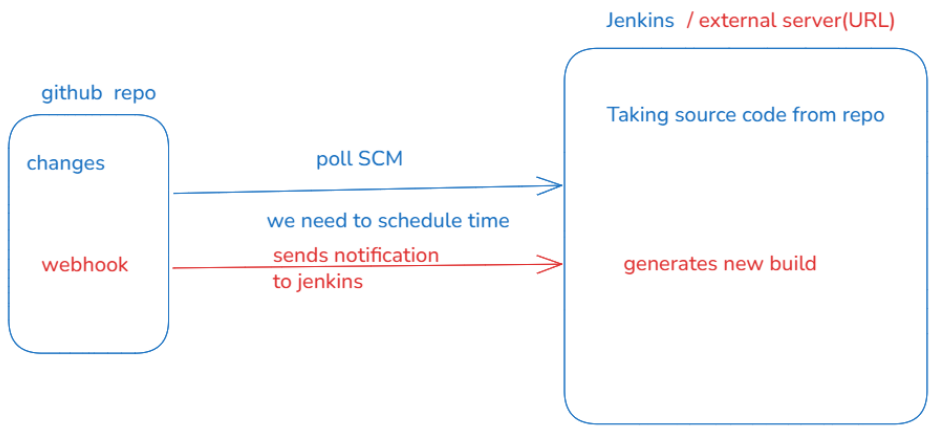
4. Copy paste the ip address of instance in browser
5. Paste the path to get password from jenkins tab

```
       > sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

http://publi-ip:8080

- copy the path for initial admin password
- paste the password which you copied from instance
- click on "install suggested plugins"
- complete profile settings(username, password, e-mail)
- click on "start jenkins"
- create a new job and start using jenkins.

Webhook


Jenkins / external server(URL)
github repo
changes
webhook
poll SCM
we need to schedule time
sends notification to jenkins
Taking source code from repo
generates new build

- Webhooks provide a way for notifications to be delivered to an external web server whenever certain event occur on GitHub(repository)

Functionality:

- Sends real-time data to a specific URL when selected events occur in your github repo
- Acts like a listener or alert system when something changes in GitHub notifies Jenkins

Working:

1. Configure webhook with specified URL(jenkins)
2. Choose events type
3. When event occurs github send a POST request to your URL with details
4. your server will handle the data and perform task (deploying code, running a script)

Where do we use it:

1. To trigger CI/CD pipelines
2. Auto-deploy to servers

Steps:

1. Launch instance and configure with jenkins
2. in jenkins => create pipeline job
3. add git checkout stage("configure" inside job)
    - click on generate pipeline syntax
    - sample step -> checkout: check out from version control
        - scm => git
        - add repo URL
        - create credential => click on add => jenkins
            - kind => username and password
            - username => name of github account
            - password
                => goto github and click on profile
                => settings => developer settings => personal access token(token classic)
                => give note (for jenkins) => select scope(select all)
            - copy the generated token
            - paste the token as password while creating credential
        - give id and description and create
4. In jenkins:
    - select credential
    - give branch name
5. click "generate"
6. copy the generated script and add it to stage

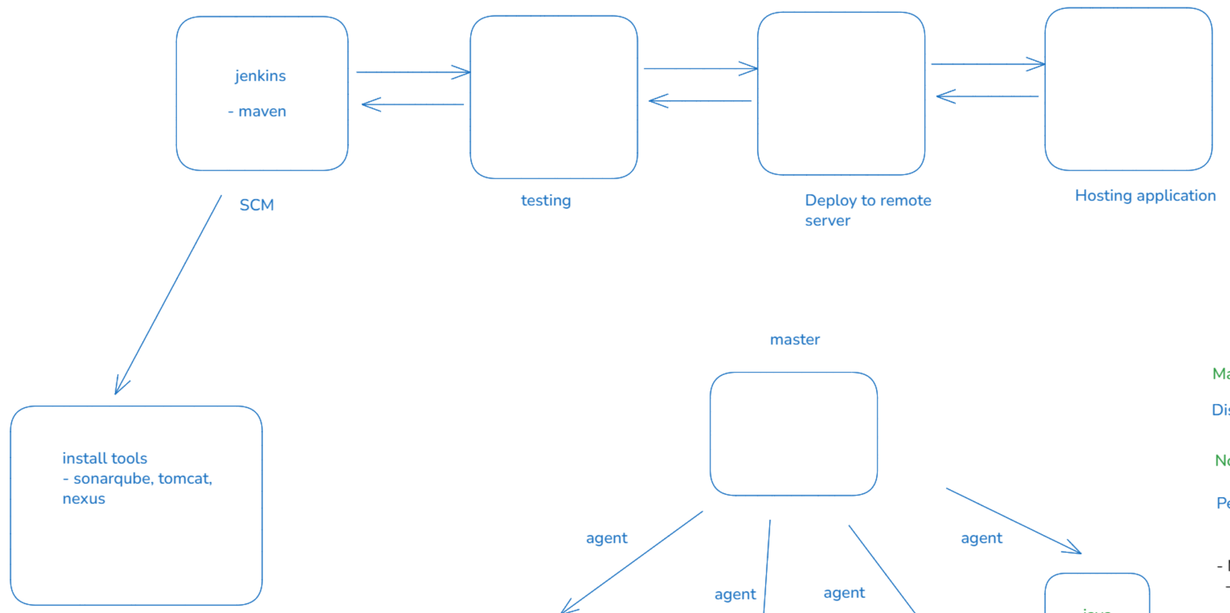
```
node {

     stage('git checkout'){

          checkout scmGit(branches: [[name: '*/main']], extensions: [],
          userRemoteConfigs: [[credentialsId: 'github-token', url:
          'https://github.com/spring-petclinic/spring-framework-petclinic.git']])
     }


}
```

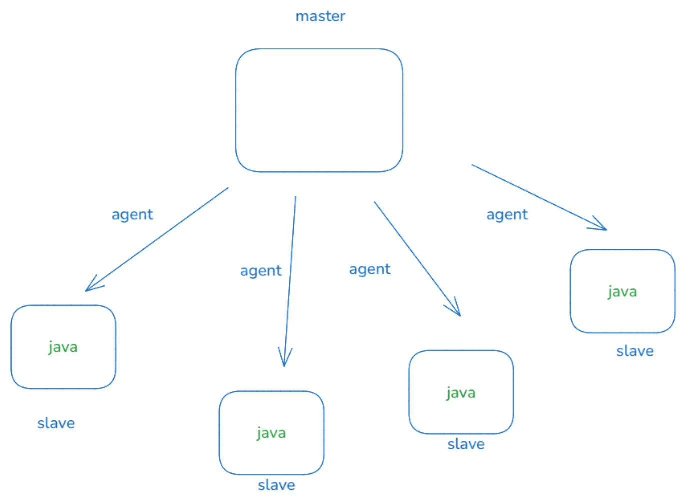
7. webhook: (go to github repository)
    - select the repository
    - click on settings
    - click on webhook
    - add webhook
        - payload URl(url of jenkins)
            eg: "http://<public-ip>:8080/github-webhook/"
        - select content type(application/json)
        - select event(push request or send me everything)
        - create webhook
8. Go to jenkins tab
    => select the job => configure =>
        select trigger as "GitHub hook trigger for GITScm polling"
9. Save
10. bulild now

NOTE: to observe/check webhook make some changes in github repoitory

Master-Slave Architecture:

Distributing workload in different machine





Master:

Distributes workloads among multiple slave or nodes or server

Nodes(Slave):

Performs the task assigned by master

- Master and slaves are connected by using agents
  -> ssh agent

slave => task => build(label)
slave2 => task => test (label)=> sonarqube
slave3 => deploy artifact => deploy(label)=> nexus

Implementation:

Master server: Install java, jenkins, maven

t2.medium (20GB)

Node-1: Install java

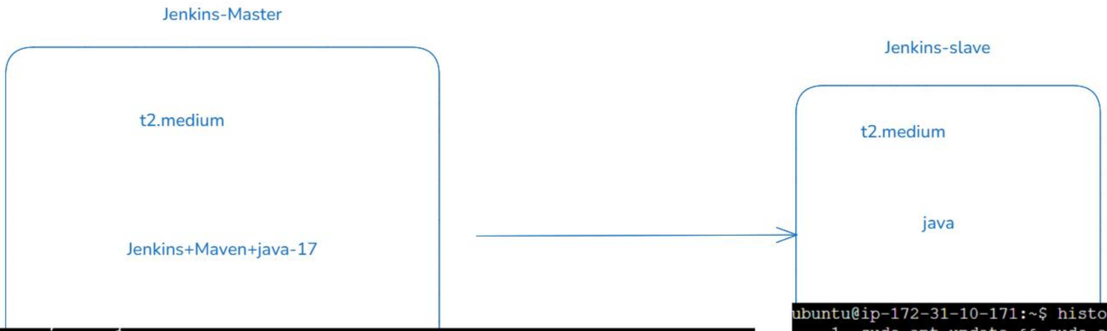
t2.micro (8GB)

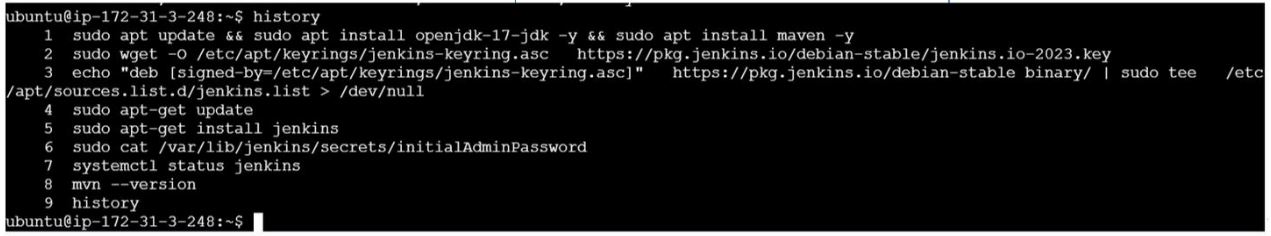
Steps: In master

- Launch instance of type "t2.medium/t3.medium"
- Configure the storage as "10GB-20GB"
- Connect this instance and install java, jenkins
-

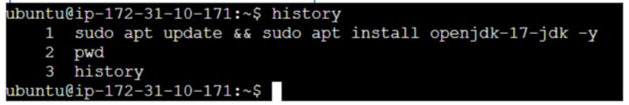
steps in slave instance:

- launch instance of type "t2.micro/t3.micro"
- storage is "8GB"
- connect this instance and install java
- create a directory
  - => mkdir project
  - => cd project
  - => pwd
    - > /home/ubuntu/project

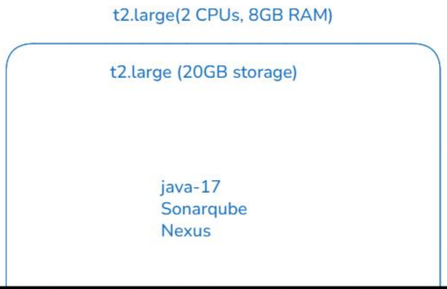


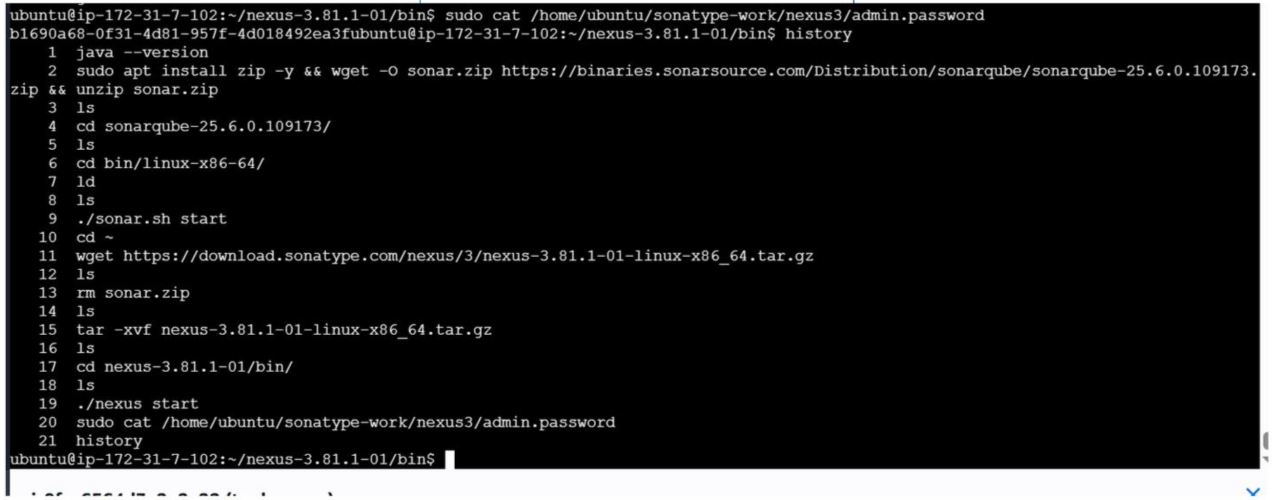


```
ubuntu@ip-172-31-3-248:~$ history
    1  sudo apt update && sudo apt install openjdk-17-jdk -y && sudo apt install maven -y
    2  sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc   https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
    3  echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc]"   https://pkg.jenkins.io/debian-stable binary/ | sudo tee   /etc
/apt/sources.list.d/jenkins.list > /dev/null
    4  sudo apt-get update
    5  sudo apt-get install jenkins
    6  sudo cat /var/lib/jenkins/secrets/initialAdminPassword
    7  systemctl status jenkins
    8  mvn --version
    9  history
ubuntu@ip-172-31-3-248:~$
```



```
ubuntu@ip-172-31-10-171:~$ history
    1  sudo apt update && sudo apt install openjdk-17-jdk -y
    2  pwd
    3  history
ubuntu@ip-172-31-10-171:~$
```





```
ubuntu@ip-172-31-7-102:~/nexus-3.81.1-01/bin$ sudo cat /home/ubuntu/sonatype-work/nexus3/admin.password
b1690a68-0f31-4d81-957f-4d018492ea3fubuntu@ip-172-31-7-102:~/nexus-3.81.1-01/bin$ history
    1  java --version
    2  sudo apt install zip -y && wget -O sonar.zip https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-25.6.0.109173.
zip && unzip sonar.zip
    3  ls
    4  cd sonarqube-25.6.0.109173/
    5  ls
    6  cd bin/linux-x86-64/
    7  ld
    8  ls
    9  ./sonar.sh start
   10  cd ~
   11  wget https://download.sonatype.com/nexus/3/nexus-3.81.1-01-linux-x86_64.tar.gz
   12  ls
   13  rm sonar.zip
   14  ls
   15  tar -xvf nexus-3.81.1-01-linux-x86_64.tar.gz
   16  ls
   17  cd nexus-3.81.1-01/bin/
   18  ls
   19  ./nexus start
   20  sudo cat /home/ubuntu/sonatype-work/nexus3/admin.password
   21  history
ubuntu@ip-172-31-7-102:~/nexus-3.81.1-01/bin$
```

- Install the plugins
    - maven integration, pipeline maven integration
    - nexus artifact uploader
    - sonarqube scanner
- Create credential for
    - sonarqube
    - nexus
- configure tool(maven)
    - click on add maven
    - give maven name
    - check "install automatically"
- Configure system (sonarqube)
    - give sonarqube url and credential

Steps to slave:

-> click on dashboard -> manage jenkins -> Nodes
-> new node
-> Give node name and click on "permanent type"
-> Give name and description
-> Select the number of executor(2)
-> Give the path of remote root directory(created directory path of slave)
    - "/home/ubuntu"
-> Label (test or build or deploy)
-> usage
    - use this node as much as possible
-> launch method via SSH
    - host
        "public-ip-of-slave-instance"
    - create credential for slave using private key file
        - kind (SSH username with private file)
        - Give id and credeentaial
        - add private key file
            -> enter directly
            -> add
        - create
-> select credential
-> Host Key Verification Strategy
    - non verifying verification strategy
-> Availability
    - keep this online as much as possible
-> apply and save

```
node('sample') {        // sample is label name

        stage ('git checkout'){
                git branch: 'main', url: 'https://github.com/Sangeetha-j-QSP/spring-framework-petclinic.git'

        }
         stage('Build generation'){

                withMaven(maven: 'apache_maven') {
                        sh 'mvn clean package'

                }

         }
         stage ('sonar analysis'){
                 withSonarQubeEnv(installationName: 'sonarqube',credentialsId: 'sonarqube-token') {

                        withMaven(maven: 'apache_maven') {

                                sh 'mvn sonar:sonar'

                        }

                 }
         }

          stage ('nexus Artifact'){

                  nexusArtifactUploader artifacts:

                  [[artifactId: 'spring-framework-petclinic',

                    classifier: '', file: 'target/petclinic.war', type: 'war']],
                    credentialsId: 'nexus-credential',

                    groupId: 'org.springframework.samples',

                    nexusUrl: '54.183.40.108:8081',

                     nexusVersion: 'nexus3',

                     protocol: 'http',

                     repository: 'petclinic',

                      version: '1.0-SNAPSHOT'
          }


}
```

Role Based Authorization:

- Install role based authorization strategy plugin
- Go to manage jenkins => security
    - authorization => role based strategy
- create users:

    steps to create users:

    - manage jenkins => users
    - give username, password, email
    - apply and save

- To manage and assign the roles
    - manage roles:
        - add roles and select permissions to that role:
            - add some roles under global role
                - give a role name under "role to add"
                    - developer
                - click on add
            - after adding role select the type of permission(read, view, run, job)
    - Assign roles:
        - add existing user to the roles created
            - click on "add user"
            - give existing user name
            - select the role for the user(developer, admin)

Generate maven project and deploy it in tomcat server
- use master-slave architecture

Mock Interview

Write a Jenkinsfile that:

-Pulls code from Git

-Runs Maven build

-Publishes artifacts

-Post-Build Action – Deploy to Tomcat