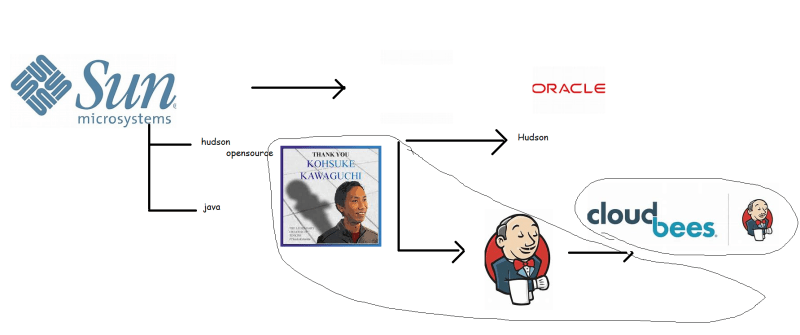
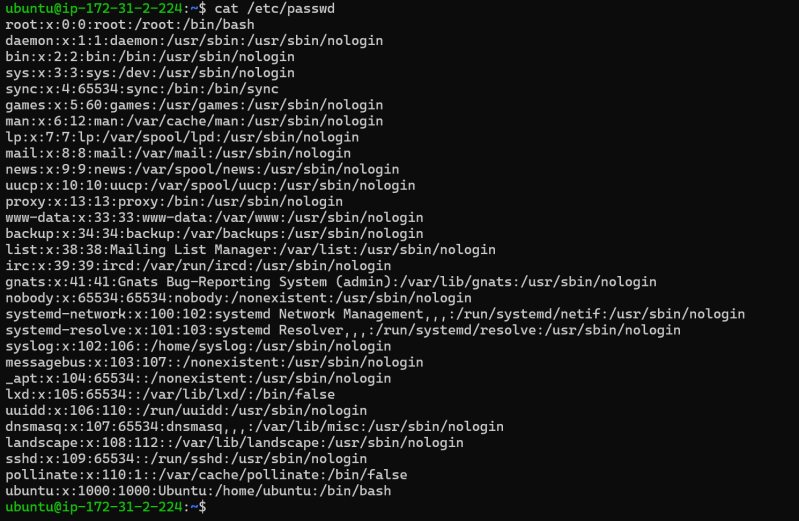
Day-1: Jenkins-1 10/Jul/2021

Exploring Jenkins

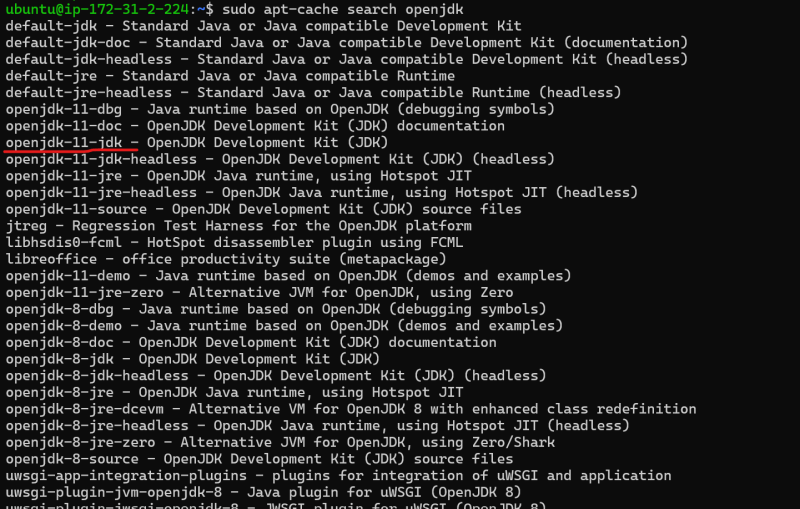
* So let’s get started by installing Jenkins. [Refer Here](https://www.jenkins.io/doc/book/installing/) (<https://www.jenkins.io/doc/book/installing/>)
* Today we will be using a Linux machine to install Jenkins which has Ubuntu distribution
* This setup will be done on AWS with 2 v CPUs and 8 GB of RAM
* Jenkins History



* Installing Jenkins on Ubuntu 18.04:
  + Login into the Jenkins using ssh
  + Let’s see the list of users on the Linux machine

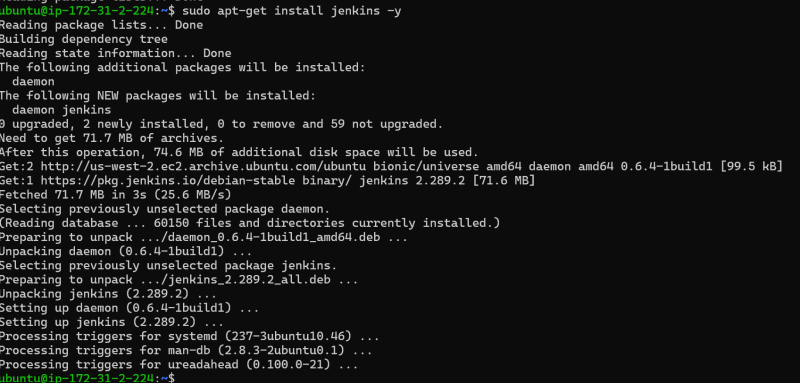


* + To install Jenkins we need to java installed for java requirements [Refer Here](https://www.jenkins.io/doc/administration/requirements/java/) (<https://www.jenkins.io/doc/administration/requirements/java/>)
  + Let’s try to install java 11
  + $Sudo apt-cache search openjdk

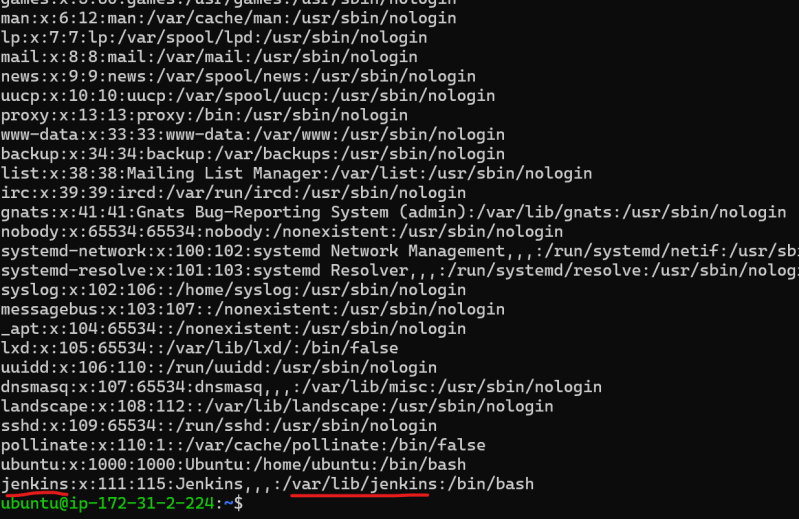


Install by executing the command

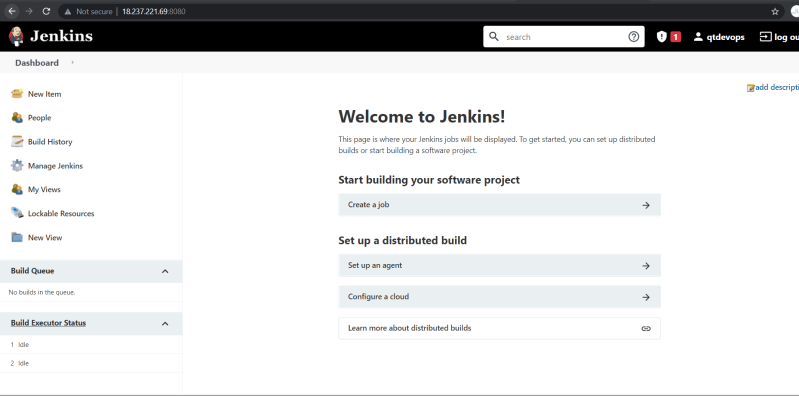
* + sudo apt install openjdk-11-jdk -y
  + Install Jenkins by executing the below commands
* sudo apt update
* wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
* sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > \
* /etc/apt/sources.list.d/jenkins.list'
* sudo apt-get update
* sudo apt-get install jenkins

vm

* + Now let’s see the list of the users



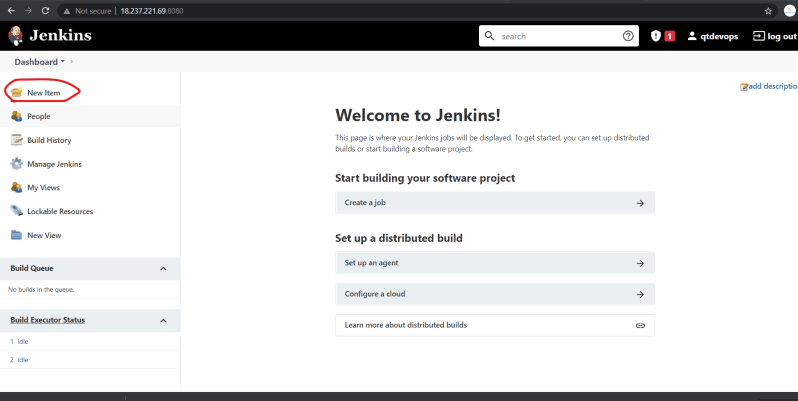
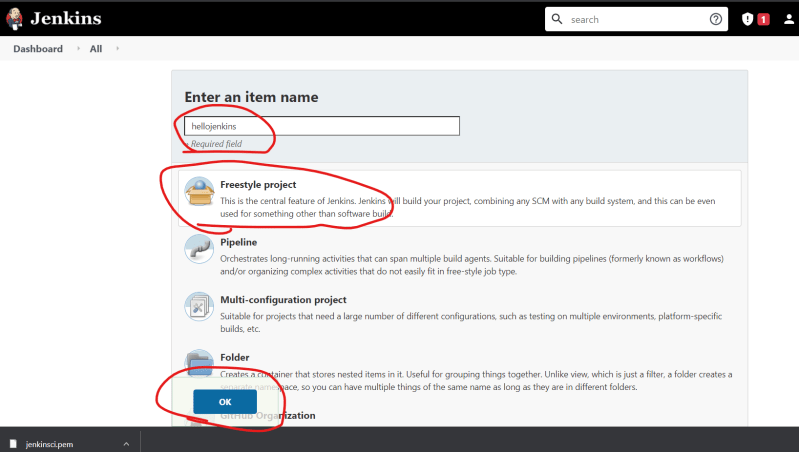
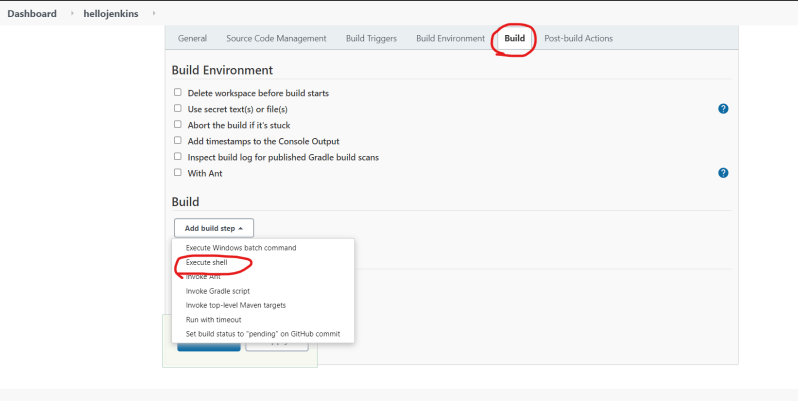
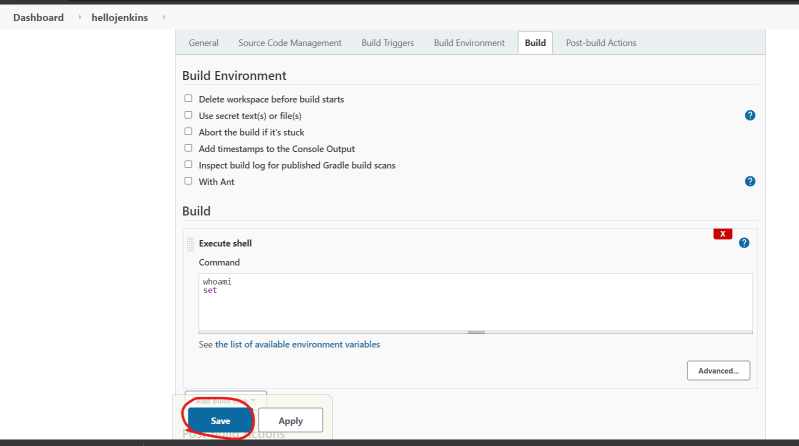
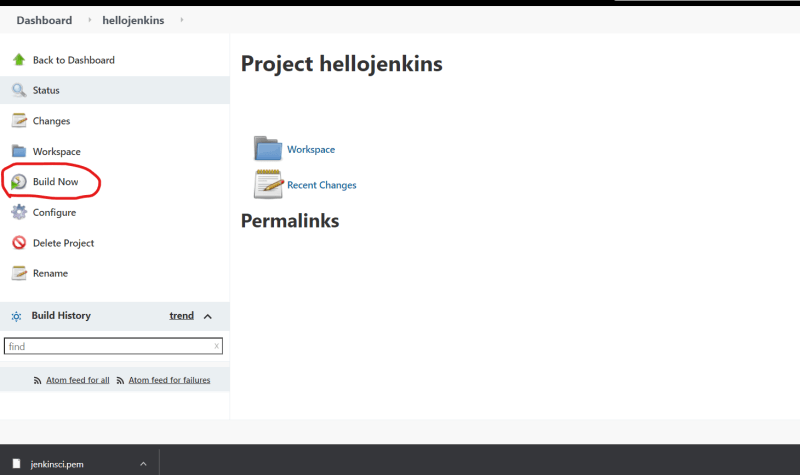
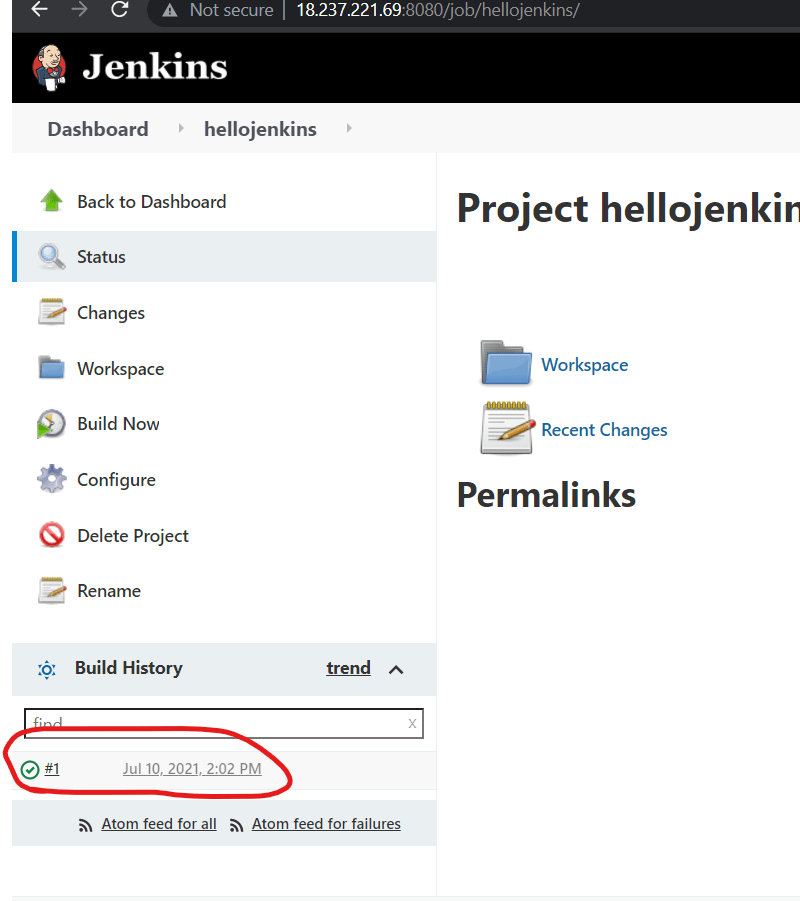
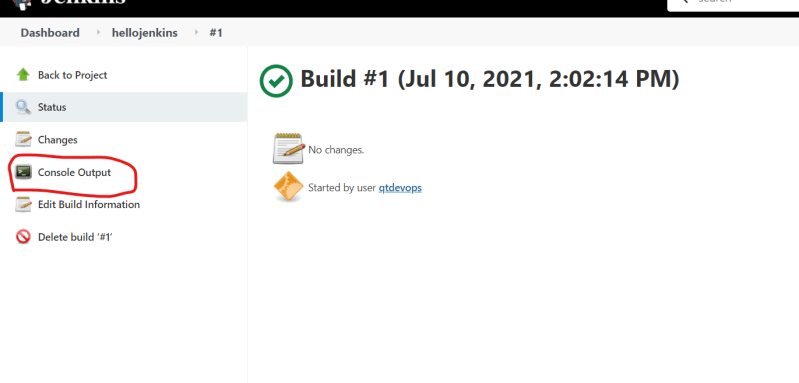
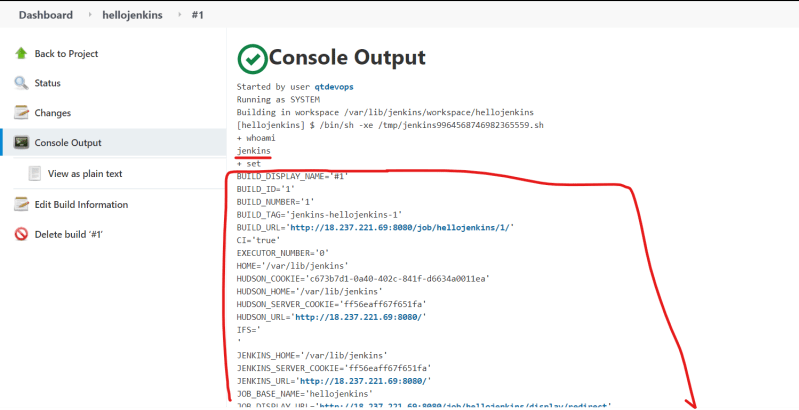
* + A New user called as jenkins is created and the home directory of this user is /var/lib/jenkins
  + Jenkins runs on port 8080 so navigate to http://<public-ip&gt;:8080
  + By using sudo cat we can enable passwd.
  + Follow the installation steps as discussed in the classroom.



Basic Jenkins Terminology

* **Project**/**Job**: This is where we define the steps to be executed.

Exploring & understanding

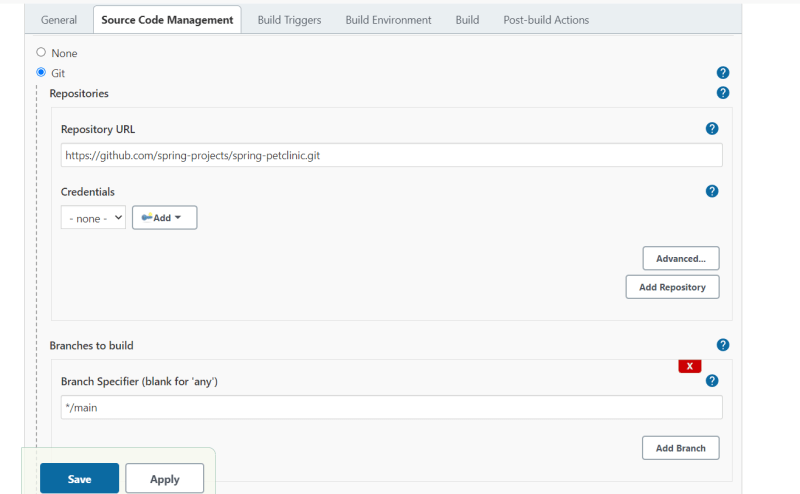
* Steps
*        
* When I run/build a project in Jenkins it is running as a newly created Jenkins user
* So Jenkins is nothing but a scheduler which runs the Linux commands as some Linux user.
* To make the life simple, Jenkins creates some UI components which are friendly towards building a project (CI/CD)

Manual Scenario 1

* Cloning the project from git. For this let’s take a project called as spring pet clinic [Refer Here](https://github.com/spring-projects/spring-petclinic) (<https://github.com/spring-projects/spring-petclinic>)
* To clone a project from git we require
  + git to be installed
  + probably internet connection
  + some disk space
* Create a Jenkins job and in the build step type git clone command.

What is Jenkins Plugin?

Plugin is a UI developed which will translate into low level commands. These UI will help in doing the job easily.

* Let’s try to clone the spring pet clinic repository as mentioned above
* Let’s look at git plugin
* 

Let’s try to build java 11 project

* clone the git repository [Refer Here](https://github.com/neiljbrown/java11-examples) and cd into java-11-projects then execute (<https://github.com/neiljbrown/java11-examples>)

mvn package

* To make this command work install maven

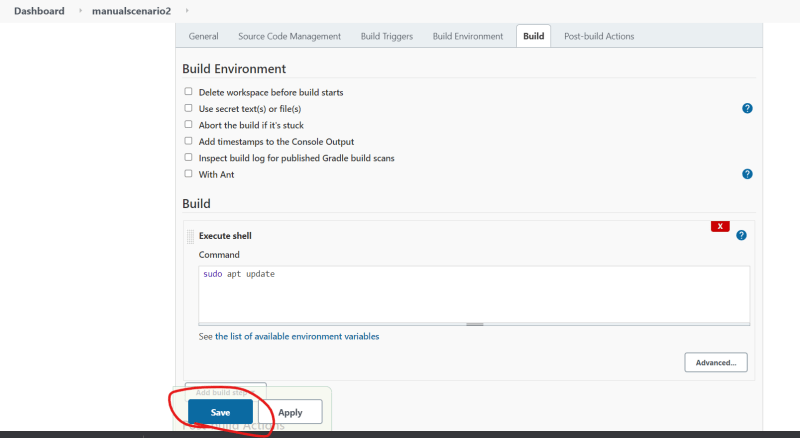
sudo apt install maven -y

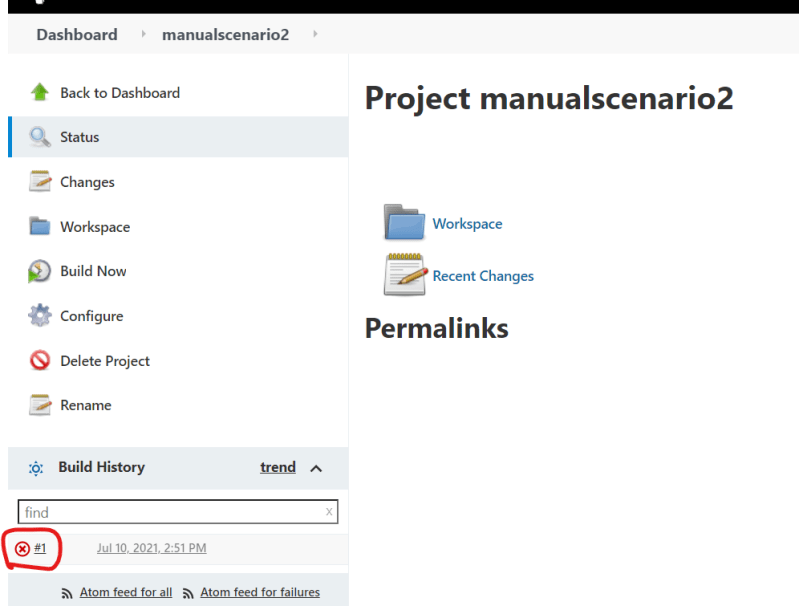
* If we can execute the command manually on the Linux machine, it will work from Jenkins as well.

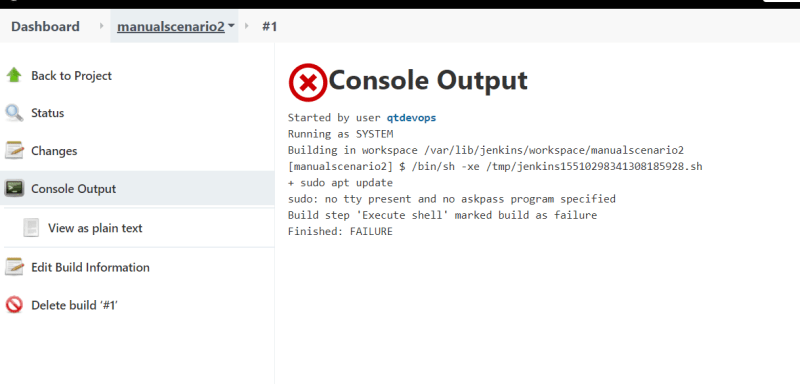
Let’s try to execute sudo commands from Jenkins

* The commands to be executed are

sudo apt update

* Let’s create the Jenkins job
* 
* The Status is





* Just because commands work on the system does not mean they will work from Jenkins. To evaluate whether they work or not we need to work with Jenkins user
* Now added Jenkins user to sudoers file with NOPASSWD option

-----------------------------------------------------------------------------

Day-2: Jenkins-2 **11/Jul/2021**

## Building Java Applications using Maven

* JDK comes with in-built java compiler, which can be used to create the java compiled classes
* JAR means java archai
* War means Web archai
* To convert java files into jar we use command :

Jar cfe jarfilename.jar javafilename

* Ex: jar cfe simple.jar Helloworld
* Build tools in Java help in building the jar/war files from the java code
* In this we would also like to run unit tests
* There was a popular tool called as Ant to perform these operations.
* In Ant we are supposed to write configuration files and in the configuration files we need to define the sequence of instructions that needs to be carried out.
* The instructions are generally written in build.xml file [Refer Here](https://github.com/piona/ant-sample/blob/master/build.xml) for the sample (<https://github.com/piona/ant-sample/blob/master/build.xml>)
* 
* [Refer Here](https://github.com/piona/ant-sample) for the sample java project with ant (<https://github.com/piona/ant-sample>)
* When we write the java code, we use lot of code which is already. This is something which we call as dependency
* So now in this day , we need build tools to
  + build the java code
  + unit test the java code
  + Manage the dependencies.

## Maven

* Maven is a powerful build tool for Java Projects (Java Based Languages).
* Maven believes in Convention over Configuration
* [Refer Here](https://maven.apache.org/) for the website of maven (<https://maven.apache.org/>)
* Let’s use maven to build some java code
* Lab Setup
  + **System** **1**: Create an Ubuntu Linux instance in any cloud/hypervisors and ensure it has internet connection
    - Install Open JDK8
  + sudo apt update
  + sudo apt-cache search openjdk
  + sudo apt install openjdk-8-jdk -y
    - JAVA is installed on the folder **/usr/lib/jvm/java-8-openjdk-amd64**
    - which is considered as JAVA HOME and we need to create an environment variable **JAVA\_HOME** representing home directory of java
  + export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64
    - Add the JAVA\_HOME to either **/etc/environment or ~/.bashrc**
    - Download the latest version of maven
  + cd /tmp
  + wget https://mirrors.estointernet.in/apache/maven/maven-3/3.8.1/binaries/apache-maven-3.8.1-bin.tar.gz
  + tar -xvzf apache-maven-3.8.1-bin.tar.gz
  + sudo mv apache-maven-3.8.1 /var/lib/maven-3.8.1
    - Now we need to create M2\_HOME environment variable which point to /var/lib/maven-3.8.1 and add /var/lib/maven-3.8.1/bin to path
    - Open ~/.bashrc and add the following lines to the end of the file
  + export JAVA\_HOME="/usr/lib/jvm/java-8-openjdk-amd64"
  + export M2\_HOME="/var/lib/maven-3.8.1"
  + export PATH=$PATH:$JAVA\_HOME/bin:$M2\_HOME/bin
    - Now logout and login
  + mvn --version
    - Can we install maven latest version with JDK 11
* On windows to install java and maven. First install chocolaty [Refer Here](https://chocolatey.org/install) (<https://chocolatey.org/install>)

choco install jdk11 -y

choco install maven -y

* Conventions over configurations

| **Item** | **Default** |
| --- | --- |
| source code | ${baseDir}/src/main/java |
| Resources | ${baseDir}/src/main/resources |
| Tests | ${baseDir}/src/test |
| Compile java classes | ${baseDir}/target/classes |
| Jar/War file | ${baseDir}/target |

## 

## Maven POM

* POM stands for Project Object Model.
* This is fundamental unit of work in Maven
* This contains information about the project and various configuration details used by Maven
* Goals:
  + **compile**: will compile the java code
  + **test**: will compile and run the test code
  + **package**: will test and create the jar/war file
  + **install**: will package and copy the package with pom file in local repository
  + **deploy**: will install and deploy the package to maven repository
  + **clean** : will remove the target folder
* In POM we will have the following information
  + **project dependencies**
  + **plugins**
  + **goals**
  + **build profiles**
  + **project version**
* Elements of POM

| **Element** | **Description** |
| --- | --- |
| **ModelVersion** | refers to the schema version of maven and is generally 4.0.0.0 |
| **GroupId** | This refers to the project which you are trying |
| **artifactId** | This refers to id of the project |
| **Version** | This refers to Version of the project if version has SNAPSHOT it means the project is still under development |

* The build package will have the following format

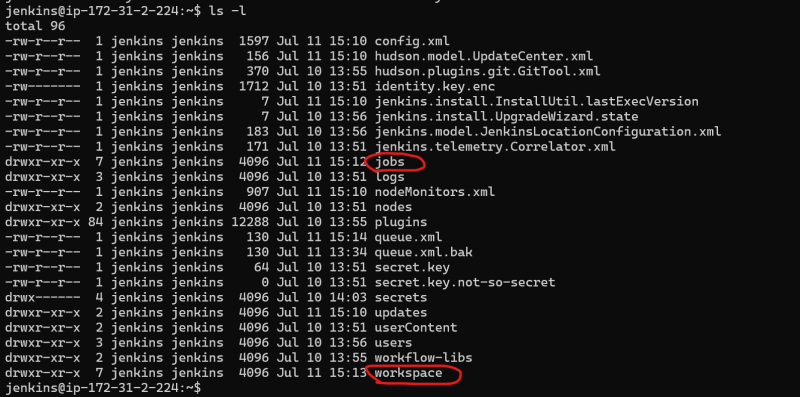
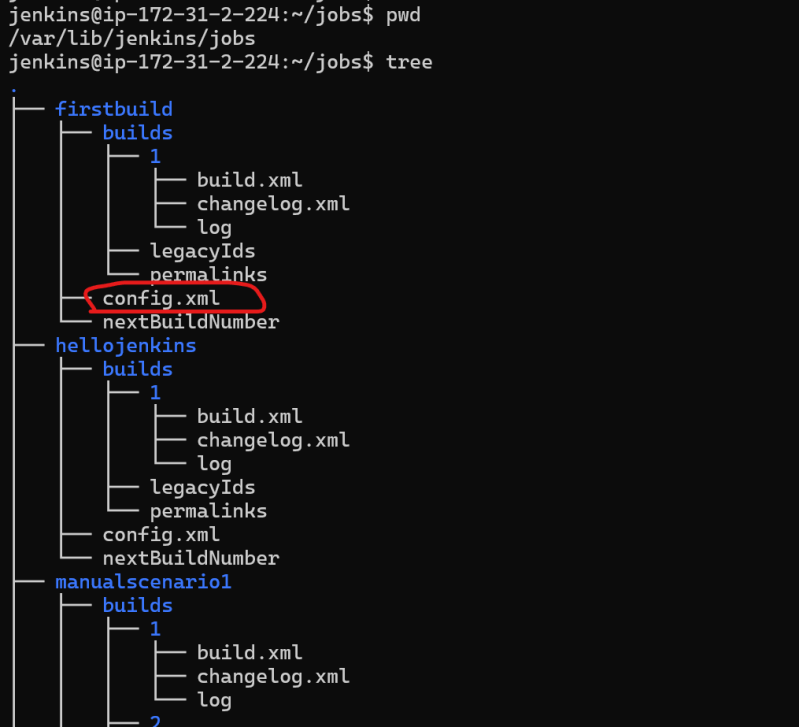
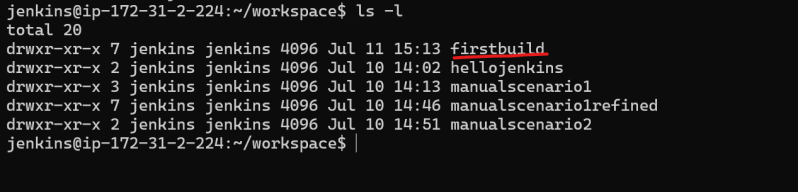
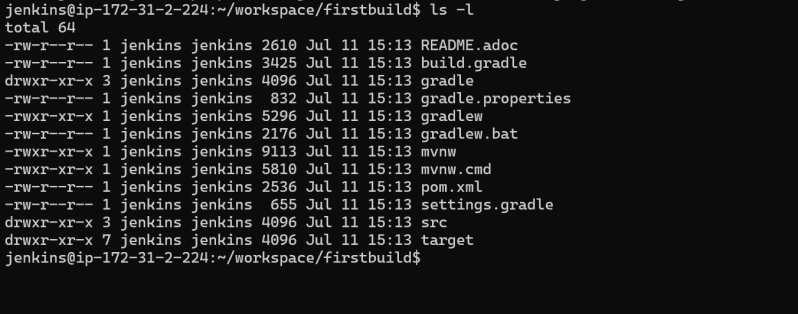
<artifactId>-<version>.<packagingformat>

* **Super POM:**
  + This is Maven’s default POM. All POMs inherit from this parent or default POM
  + Try executing
* mvn help:effective-pom
* What is Build life cycle in maven

Maven is based around the central concept of a build lifecycle. ... There are three built-in build lifecycles: **default, clean and site**. The default lifecycle handles your project deployment, the clean lifecycle handles project cleaning, while the site lifecycle handles the creation of your project's web site

* What are Build profiles
* What are Plugins

## Jenkins continuation

* From Jenkins let’s build a java project and show the jar files to user
* Execute mvn package from Execute shell after configuring git repository of java project [Refer Here](https://github.com/neiljbrown/java11-examples) (<https://github.com/neiljbrown/java11-examples>)
* Now let’s explore how Jenkins is storing the projects
* cd in to the home directory of Jenkins (/var/lib/jenkins) as jenkins user and see the contents
* 
* Now get into jobs/your job name
* 
* The jenkins project created gets stored as xml file in /var/lib/jenkins/jobs/$job-name/config.xml
* Now let’s cd into /var/lib/jenkins/workspace
* 
* Now let’s cd into firstbuild
* 
* Jenkins when a project is created stores an xml file in the $JENKINS\_HOME/jobs/<your-job-name>/config.xml and when the project is executed all the work is done in the workspace folder $JENKINS\_HOME/workspace/<your-job-name>/

Day-3: Jenkins-3

# 13/Jul/2021

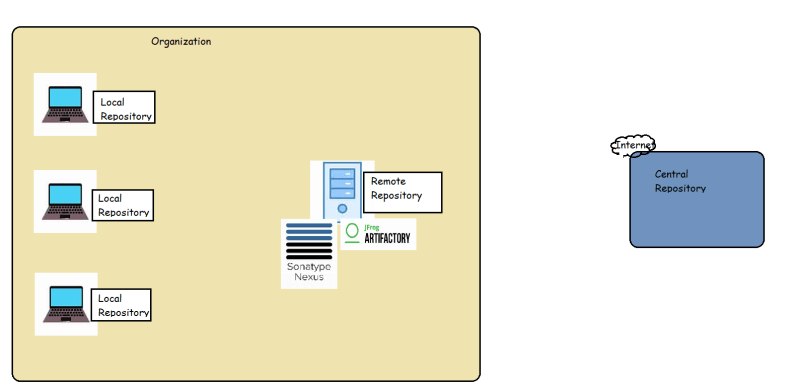
## Maven continued

* **Build Profile:**
* What are Build profiles
  + These are set of configuration values which can be used to set or override the default values
  + Types of Build Profile:
    - Per Project => project’s pom.xml
    - Per User => (HOME-DIRECTORY/.m2/settings.xml)
    - Global => Defined in the M2\_HOME/conf/settings.xml

## Maven Repository:

* What are Repositories in maven?

In Maven, repository is a directory where all the project jars, library or any other project specific artefacts can be stored. Generally we will have dependencies and our project’s jar and pom.xml files stored in repository

* 
* Maven repositories are of three types
  + local
  + remote
  + central
* Central repository: This is repository maintained by Maven community and has large number of commonly used libraries. when maven doesn’t find the dependency in your local repo it will start searching in central repo [Refer Here](https://repo1.maven.org/maven2/) (<https://repo1.maven.org/maven2/>)  and if found downloads to your local repository
* Remote Repository: This repository will be maintained by our organization to have the libraries developed and build daily. All the developers in our organization can start using the libraries developed by other teams/individuals by adding dependencies to the current project pom

<project>

...

<repositories>

<repository>

<id>qt.general</id>

<url>https://download.qt.com/maven2/general</url>

</repository>

</repositories>

</project>

* Skills for DevOps Engineers Required w.r.t build tools
  + How to configure/resolve dependency
  + maven => central repo => pom.xml <dependencies>
  + .net => nuget => packages.json/packages.config in the project
  + node js => npm => npm install
  + python => pip => requirements.txt => pip install -r requirements.txt
  + Get to know the commands
  + How to execute unit tests and where will be the test reports?
  + How to store artefacts into some repository

--------------------------------------------------------------------------------

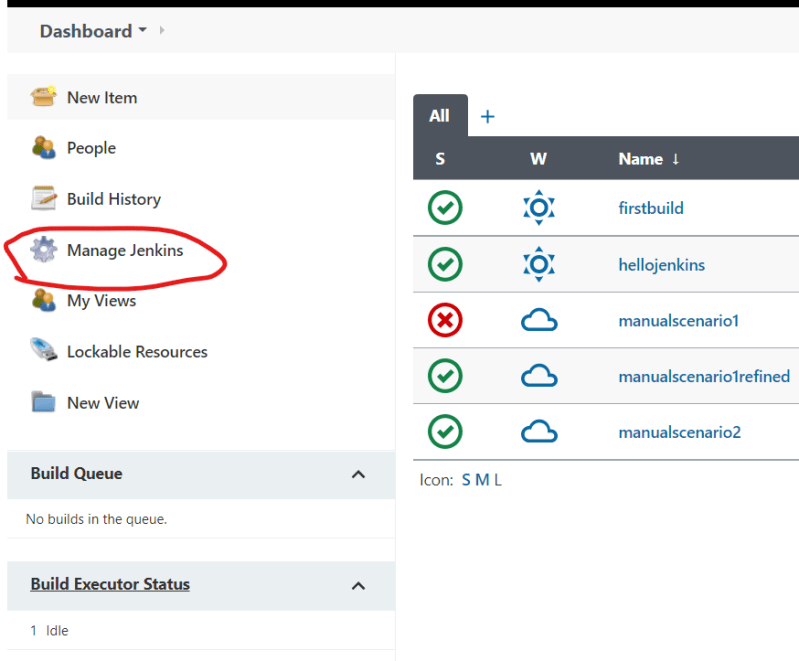
Day-4: Jenkins-4 -14/Jul/2021

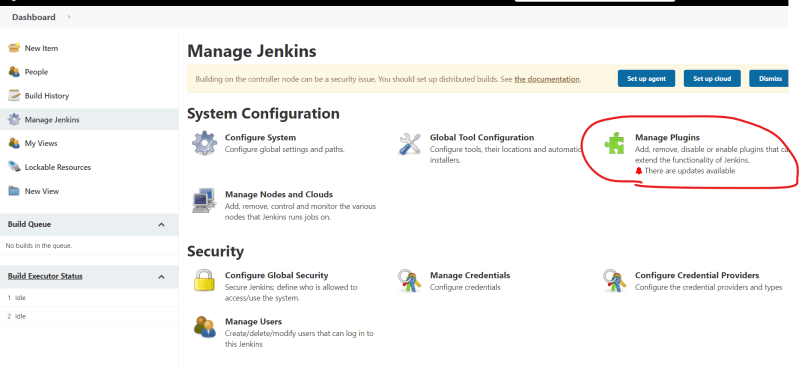
What is the command to see CPU, memory and swap memory utilization?

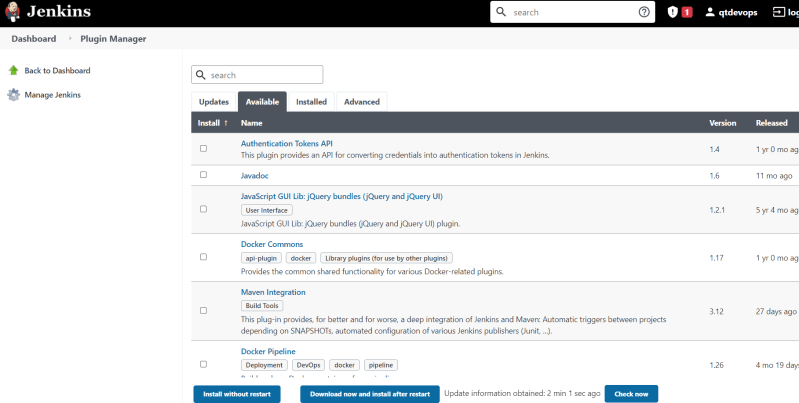
sudo top

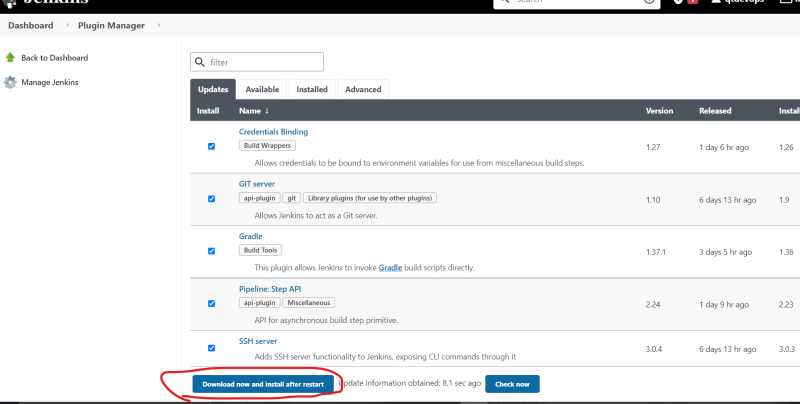
## The Jenkins Plugin Manager

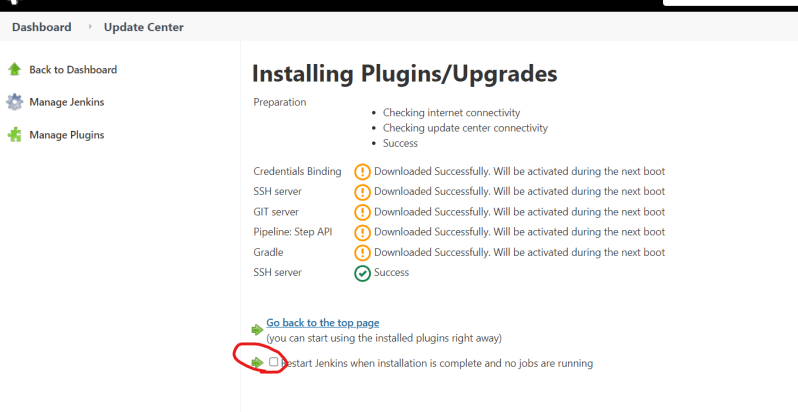
* Jenkins derives most of its power from plugins. A plugin is a piece of software that upon installation will enhance the Jenkins functionality.
* Jenkins is a Plugin Manager

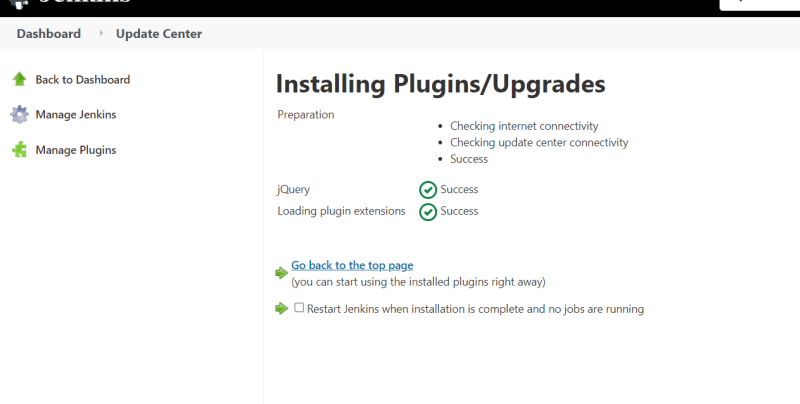
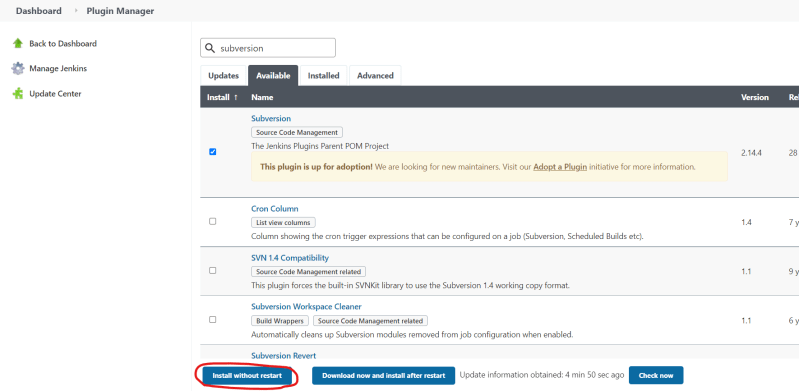
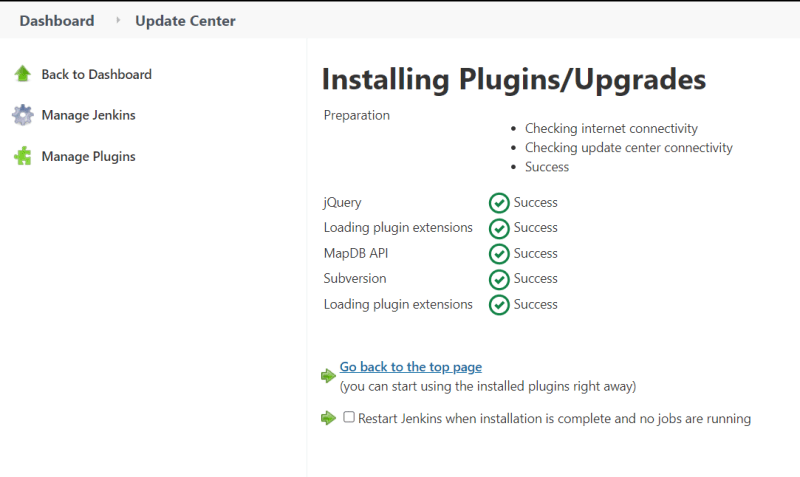
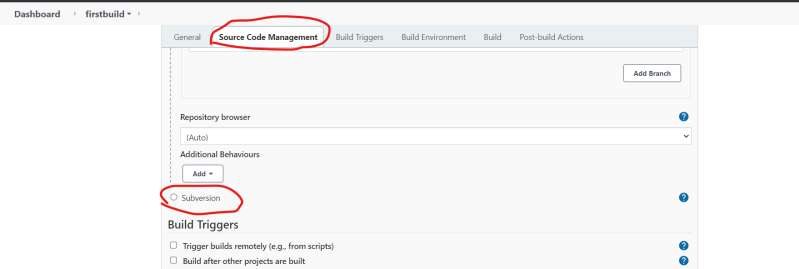


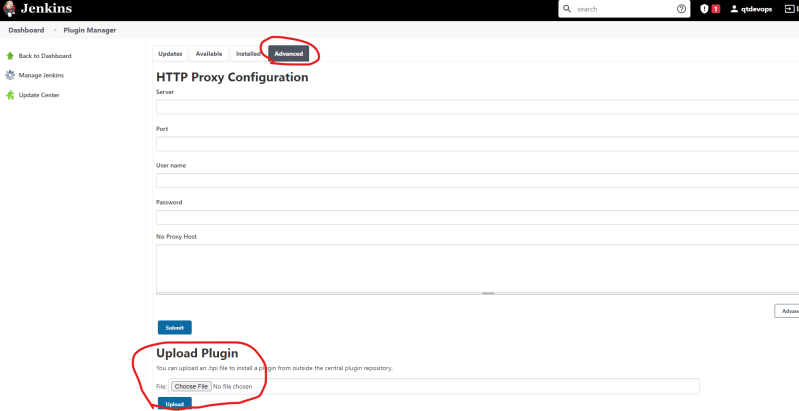




* Updating existing plugins
* 



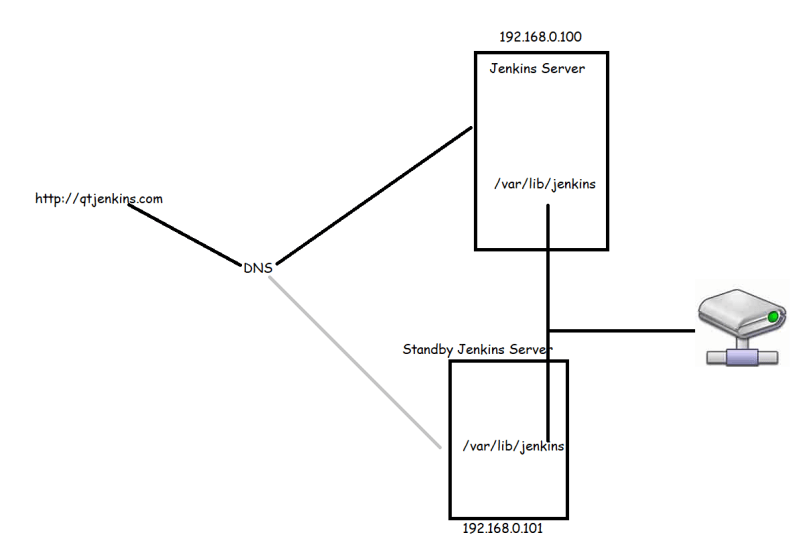
* Installing plugins
* 
* Let’s assume your organization is using subversion as version control system
* 
* 
* 
* Manually installing a Jenkins Plugin:
  + In Some cases we need to install the Jenkins Plugin
  + Jenkins plugins file extension will be of type **.jpi (jenkins plugin interface) or .hpi (hudson plugin interface)**



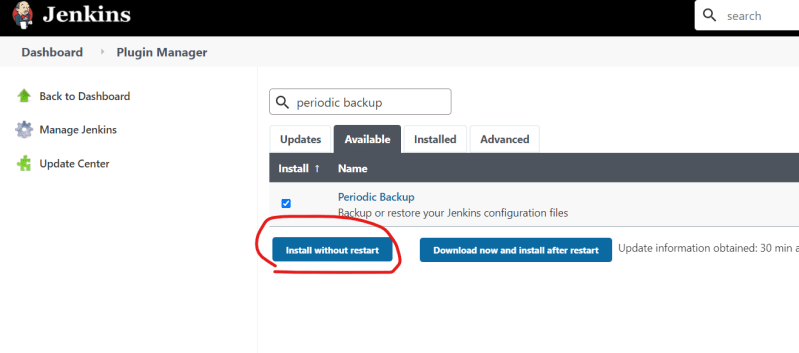
* Jenkins plugin index [Refer Here](https://plugins.jenkins.io/) (<https://plugins.jenkins.io/>)

## Jenkins Backup and restore

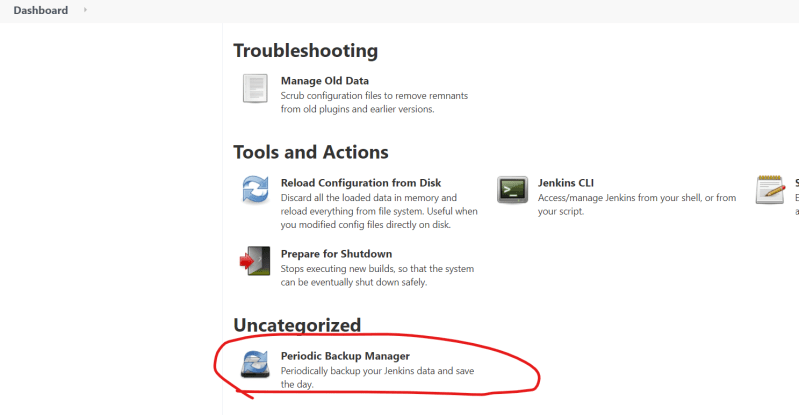
* All of the Jenkins data is stored in the Jenkins home directory
  + **/var/lib/jenkins**
* Most easiest way of taking backup is create a sync of Jenkins home directory in some other system
* Sample Highly Available Jenkins setup



* Periodic Backups of Jenkins using Plugins

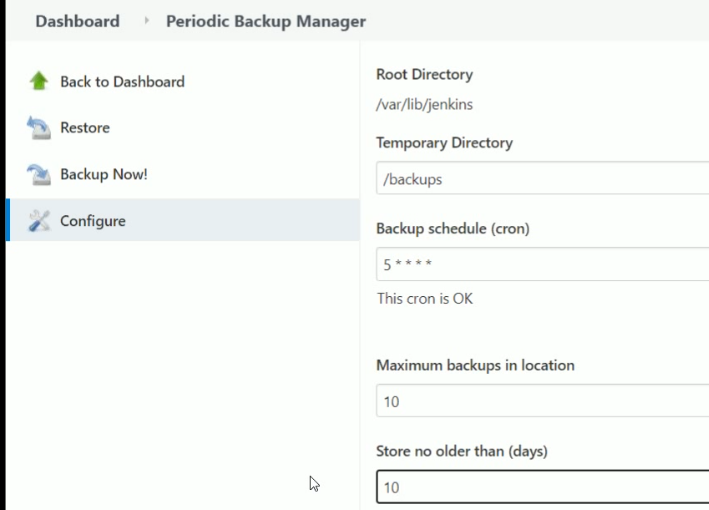
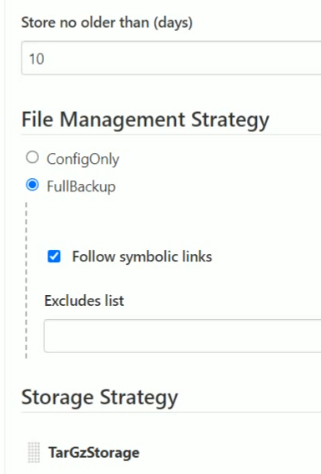


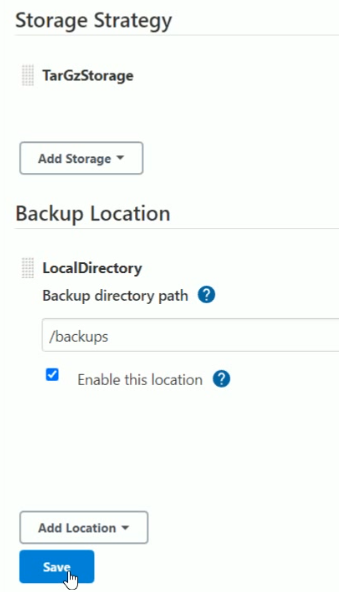
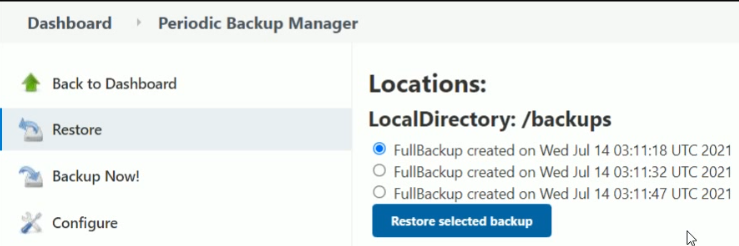
* Configuring periodic backup
  + Manage Jenkins => Periodic Backup Manager



* + Now click configure and ensure we have a folder for backups and provide the information as discussed in the class.
  + For temporary backups create a folder
* $ sudo mkdir /backups
* $sudo chown jenkins /backups

Configure with all required data

==========================================================

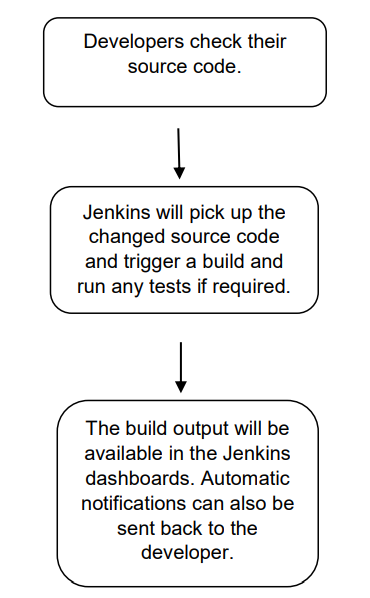
Day-5: Jenkins-5 -23/Jul/2021

Jenkins User Management

* Authentication Methods
  + Delegate to Servlet container
  + Jenkins own user database
  + LDAP (Light Weight Directory Accesses Protocol)
  + Unix user/group database
* Creating users inside Jenkins
* Configuring authorization
  + Logged in users can do anything
  + Matrix Security
  + Project Based Security
  + Role based Security (Plugin needs to be installed)

Why Jenkins?

Jenkins is a software that allows **continuous integration**. Jenkins will be installed on a server where the central build will take place. The following flowchart demonstrates a very simple workflow of how Jenkins works.



Along with Jenkins, sometimes, one might also see the association of Hudson. Hudson is a very popular open-source Java-based continuous integration tool developed by Sun Microsystems which was later acquired by Oracle. After the acquisition of Sun by Oracle, a fork was created from the Hudson source code, which brought about the introduction of Jenkins.

What is Continuous Integration?

Continuous Integration is a development practice that requires developers to integrate code into a shared repository at regular intervals. This concept was meant to remove the problem of finding later occurrence of issues in the build lifecycle. Continuous integration requires the developers to have frequent builds. The common practice is that whenever a code commit occurs, a build should be triggered.

Introduction to Jenkins and its features lets first understand what continuous integration is. CI is one of the most popular application development practices in recent times. Developers integrate bug fix, new feature development, or innovative functionality in code repository. The CI tool verifies the integration process with an automated build and automated test execution to detect issues with the current source of an application, and provide quick feedback. Jenkins is a simple, extensible, and user-friendly open source tool that provides CI services for application development. Jenkins supports SCM tools such as StarTeam, Subversion, CVS, Git, AccuRev and so on. Jenkins can build Freestyle, Apache Ant, and Apache Maven-based projects. The concept of plugins makes Jenkins more attractive, easy to learn, and easy to use. There are various categories of plugins available such as Source code management, Slave launchers and controllers, Build triggers, Build tools, Build notifies, Build reports, other post-build actions, External site/tool integrations, UI plugins, Authentication and user management, Android development, iOS development, .NET development, Ruby development, Library plugins, and so on. Jenkins defines interfaces or abstract classes that model a facet of a build system. Interfaces or abstract classes define an agreement on what needs to be implemented; Jenkins uses plugins to extend those implementations.

Jenkins features:

It is a plugin based tool

Installation and plugin is easy

No need connect any database

Java is the only pre requests for Jenkins

Default port number is 8080

People: It displays all users inside of jenkins.

New Item: to create new build

To create user: Manage jenkins 🡪 manage users 🡪create user

Configure Global Security:

Access control 🡪 security realm:

By default it is in enable mode.

Jenkins has its own user database. By default it stores in workspace program file.

Authorization:

1. Anyone can do anything: it means all users who are inside of Jenkins server will able to access to control and run Jenkins as per their wish if we enable it.
2. Logged-in Users can do anything: It means the users who are just logged in into inside of Jenkins servers they can able to do anything. If we enable allow anonymous read access, it can able to give read access to any user even if he didn’t logged-in into Jenkins server.
3. Matrix-Based Security:

To give a particular specification & permissions to particular user is known as matrix based security.

**Note**: here we need to select at least one user and to need to give full permissions.

1. Project based matrix authorized strategy:

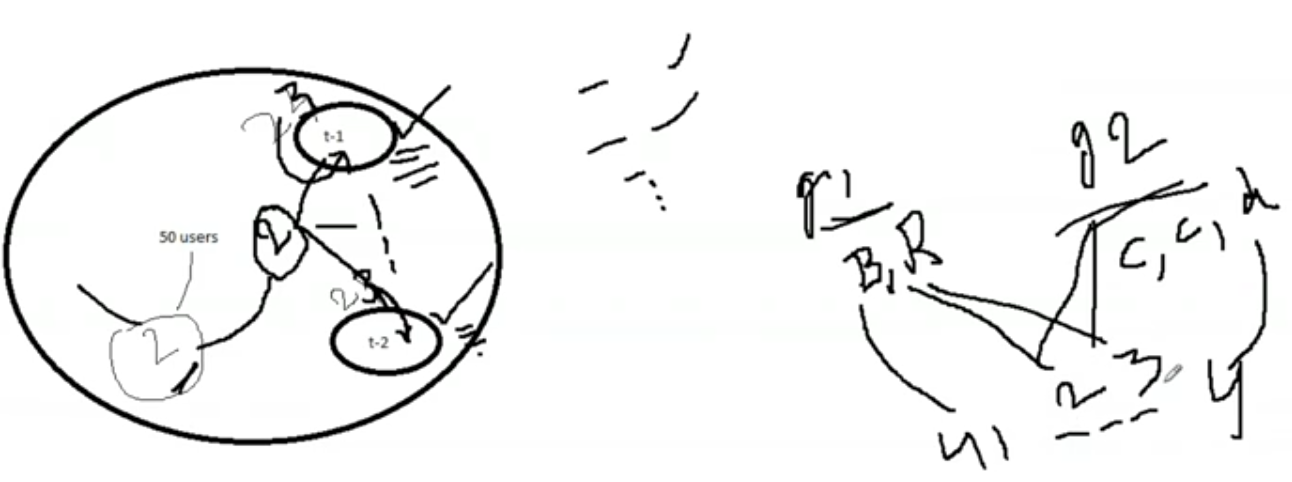
If a specific user or N number of users are assigned to a specific project is known as project based matrix security.

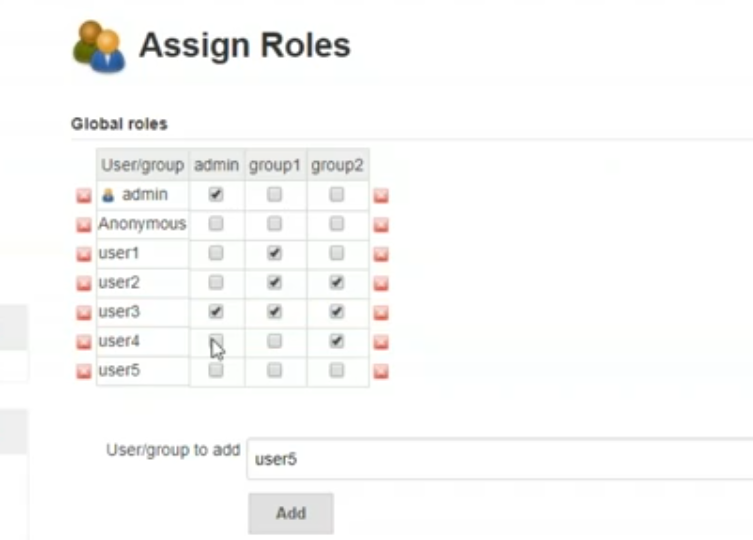
**Note**: in Project based matrix security here we need to select at least one user and to need to give full permissions.

1. Roll based Authorization strategy:

Allocating permissions to particular group, group of people is known as Roll based strategy.

Install plugins and save later go to manage Jenkins🡪Manage and assign roles🡪 we can perform here all roll based strategy tasks.





**Global Tool Configuration**

* Jenkins offers Global tool configurations and also provides some installers for JDK, maven, gradle etc.
* Best Practice:
  + Install the tools manually and configure them in Global tools rather than using automated installers

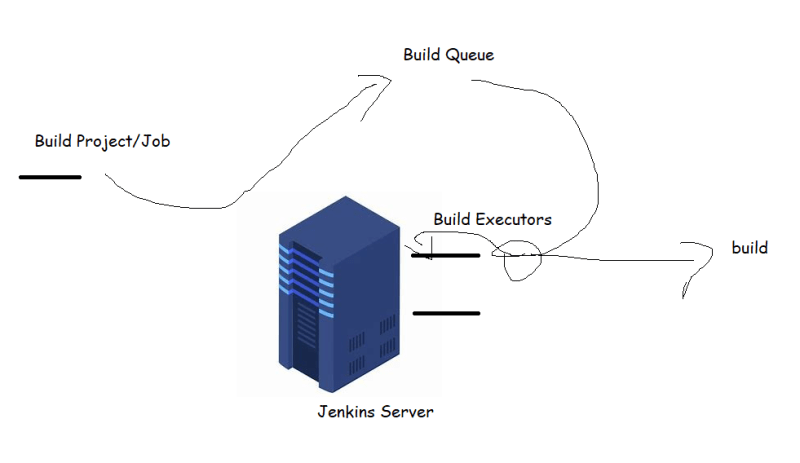
We can configure here like maven, git, jdk, graddle, ANT, Maven, Docker,

To configure JDK we need to give jdk name like jdk8, jdk9 …… and give the variable path of local machine. We can able to add multiple JDKs

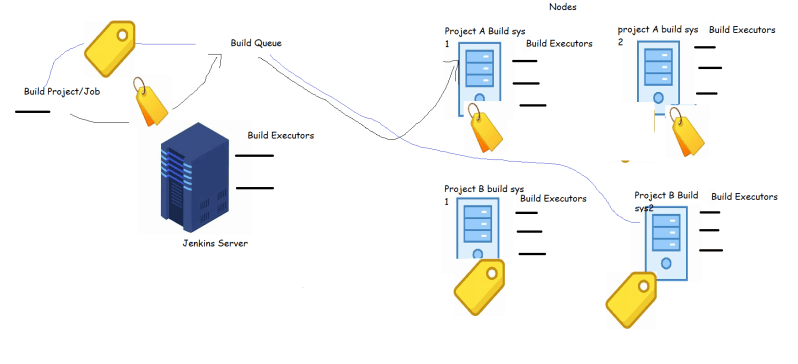
To install automatically from site we have to be enable “install automatically” and select the version of jdk from dropdown.

If we didn’t configure properly we may get fatal errors.

**Build Executors and Queues**



**Distributed Builds**

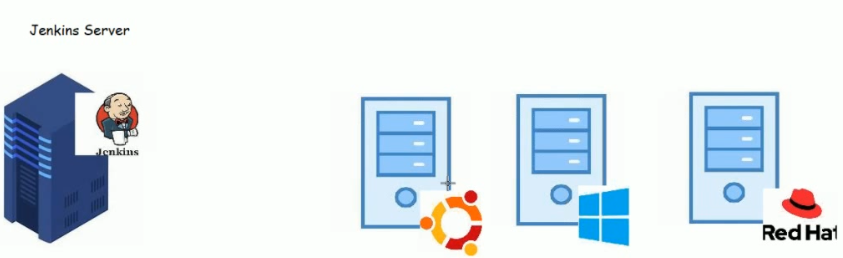
============================================================

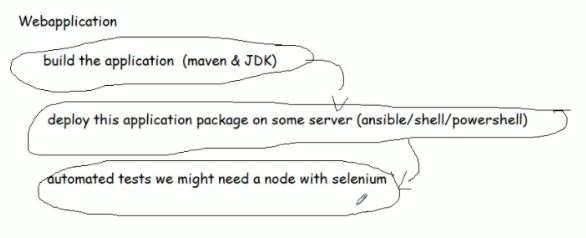
Day-6: Jenkins-6 -24/Jul/2021

[No audio for this video, no problem same stuff explained in next class OR this concept same like ansible node setip]

Jenkins Distributed Builds

* When we try to build a Jenkins job, Jenkins will try to find an **executor**. Jenkins server has by default **2 executor**. This implies we can **build two jobs in parallel**.
* In an enterprise scenario, each project has its set of tools/software’s which might be different, to support this Jenkins has node concept
* A node is a VM or Physical or docker container with preconfigured set of tools installed.
* We need to configure the connection b/w Jenkins Server and node and also configure number of executors. Each node will/should have a label.
* Now once this configuration is done, then we can have our jobs build on different node than Jenkins server
* This gives an option to have various builds configured from one highly available Jenkins.
* Adding nodes to Jenkins server gives us the flexibility to create various environments (QA, Dev, Staging, pre-prod)





How to configure a node to Jenkins server

* Jenkins tries to login into remote node and execute the job for this Jenkins uses an agent from Jenkins called as Jenkins-agent
* On the Jenkins node java 8 or java 11 has to be installed
* For our lab set up lets create a Ubuntu 18 node install java and maven

sudo apt update

sudo apt install openjdk-8-jdk -y

sudo apt install maven -y

* To configure a Linux node we need to configure the ssh communication between Jenkins server and node
  + Jenkins server needs to know the username and password of the node
* Ensure password authentication is enabled by checking **PasswordAuthentication** value in **/etc/ssh/sshd\_config** and change it to **yes** mode
* Ensure a user is created for running builds on Jenkins node

Sudo service sshd restart

sudo adduser jenkins

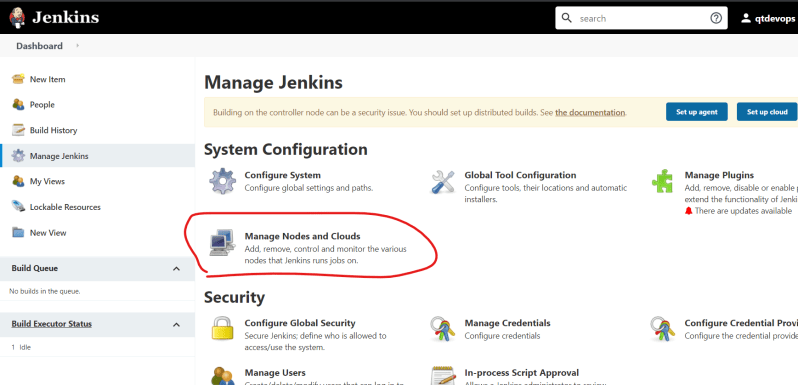
* Give the user necessary permissions

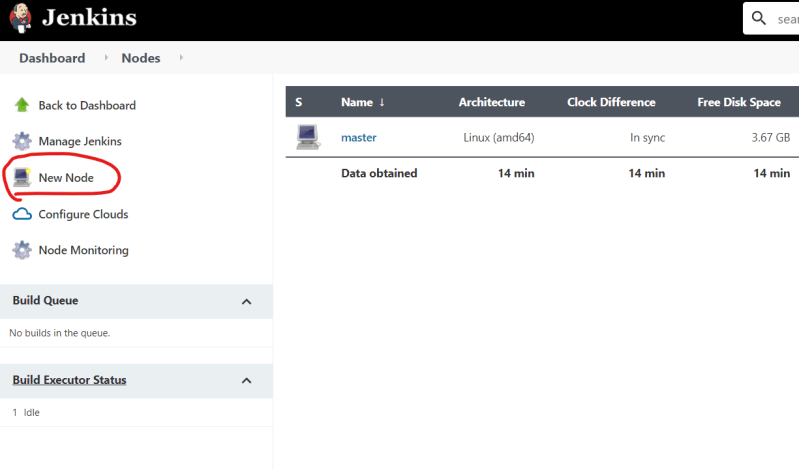
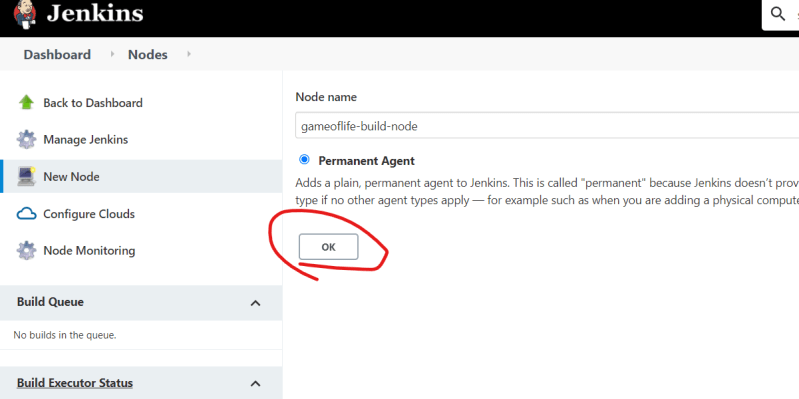
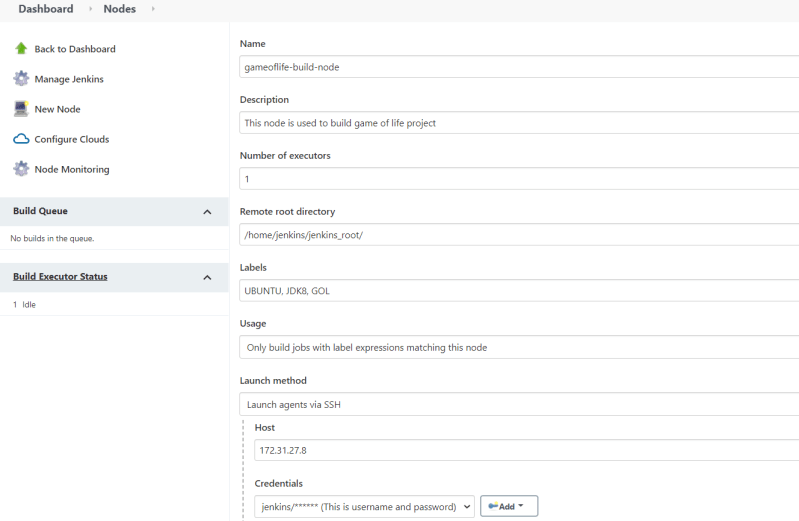
sudo visudo

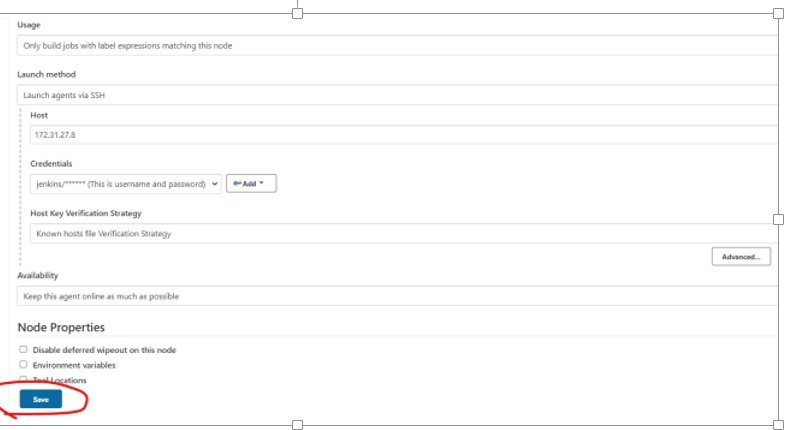
Give sudo permissions to Jenkins with NOPASSWD

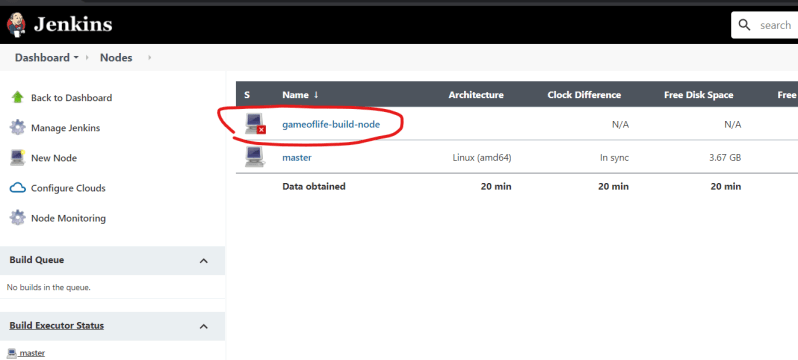
Try to login into Jenkins node from Jenkins server as Jenkins user ssh jenkins@ip

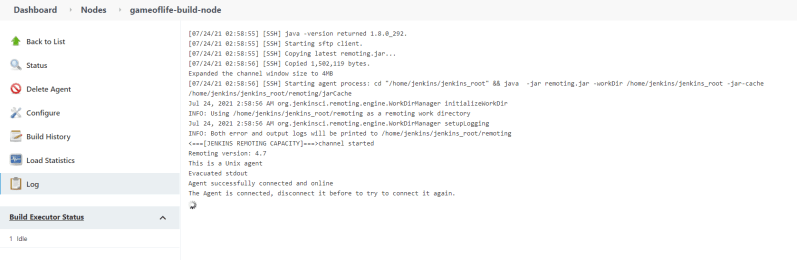
* Now let’s try to configure this node from Jenkins ui



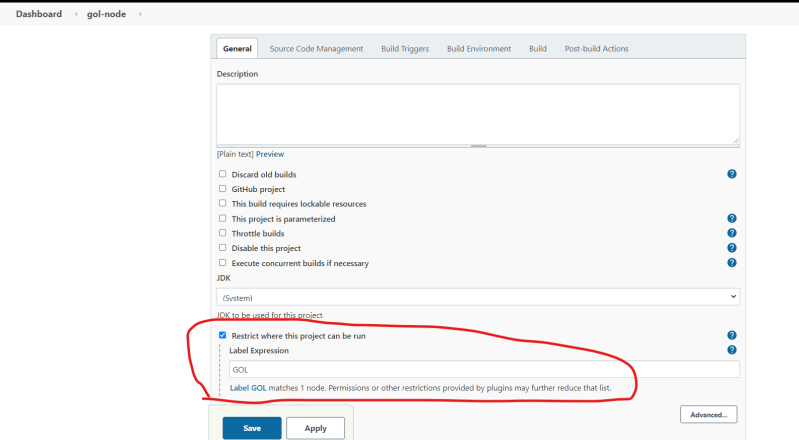
* 
* 
* 
* Host= private IP of node

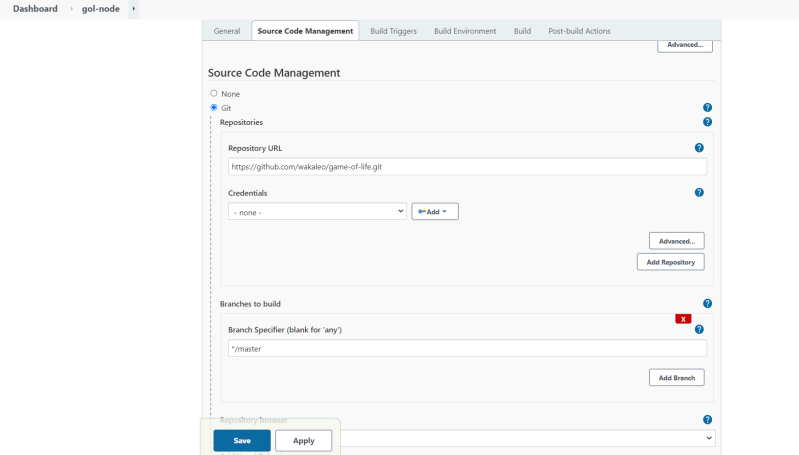


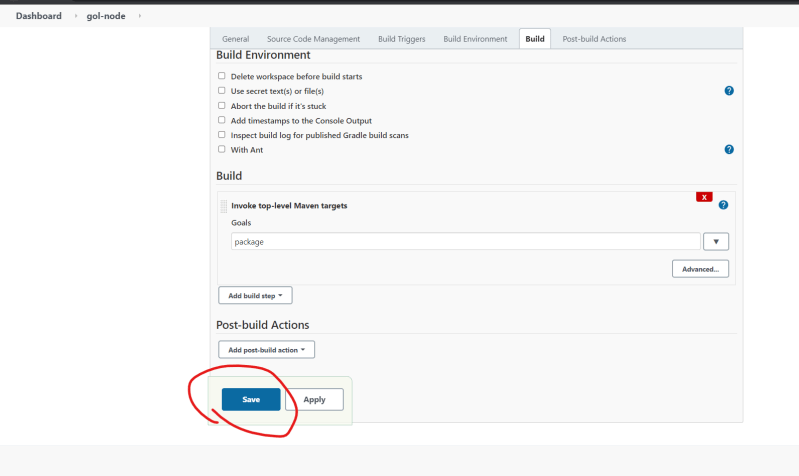
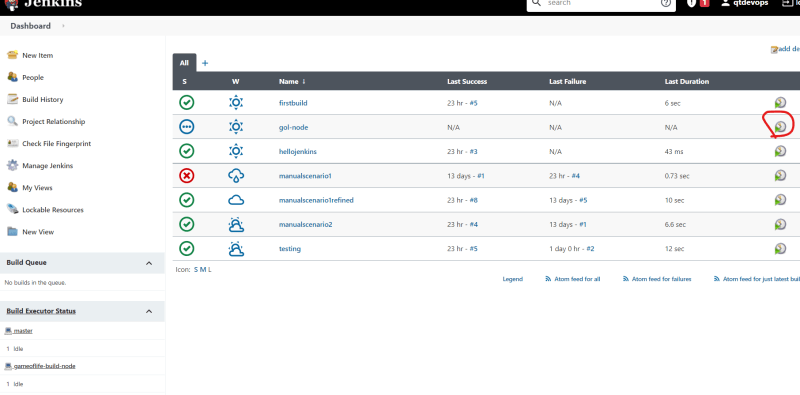
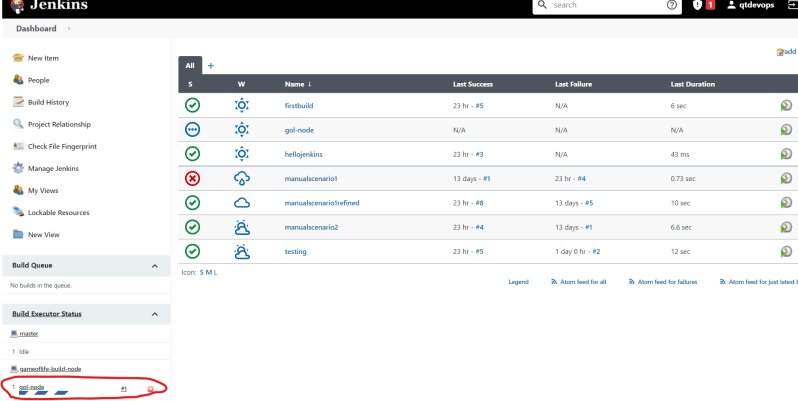


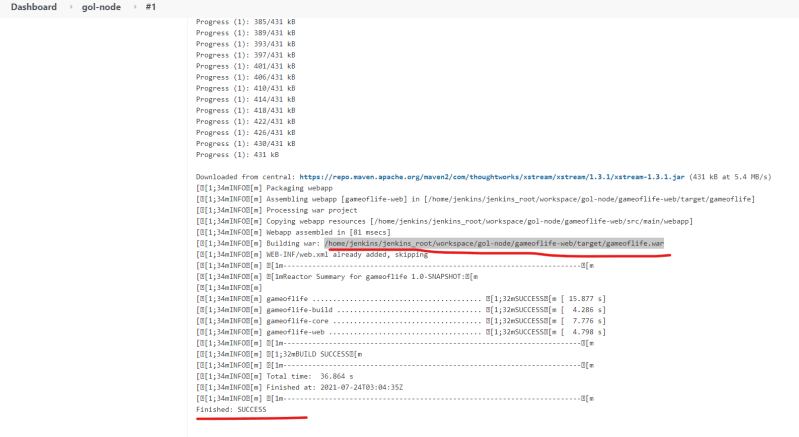


* Now since the node is healthy let’s try to build a game of life on the node







* Next Steps:
  + We have configured a Linux node with ssh username and password, we need to configure with name and ssh-key
  + Other aspects of Jenkins such as periodic builds, mail notifications, and post build actions etc.

====================================

Day-6: Jenkins 7

**24/Jul/2021**

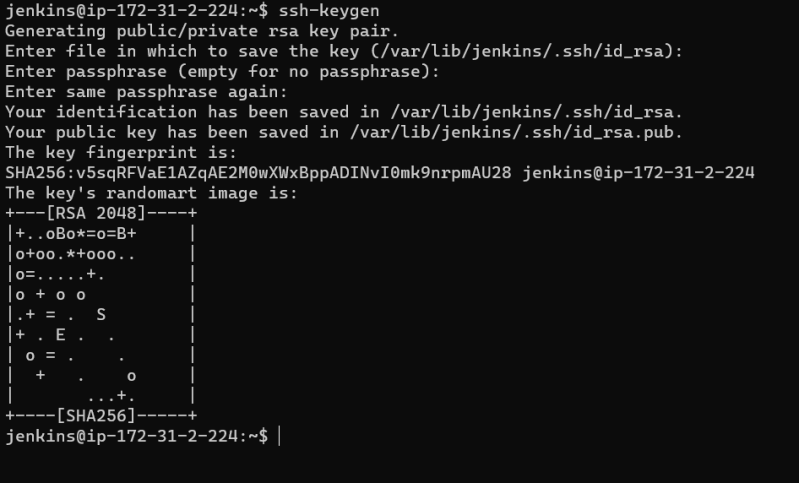
## Configuring Jenkins Node with username and ssh key

* Create a new redhat vm
* Enable password authentication for the redhat vm
* Create a jenkins user with sudo permissions
* Install jdk 11 and maven on redhat vm

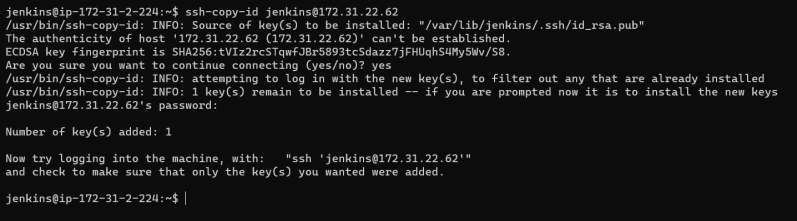
sudo dnf install java-11-openjdk-devel maven –y

* Command to switch the user is **#su username**
* ~ means home directory. To move home directory of user is **#cd ~**
* Now login into jenkins server/master as jenkins user and create a key-pair

ssh-keygen



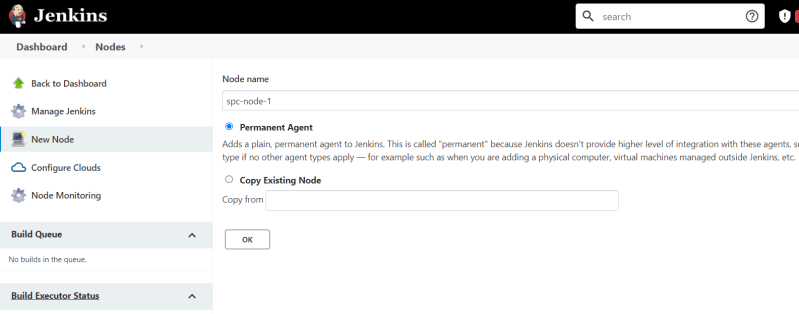
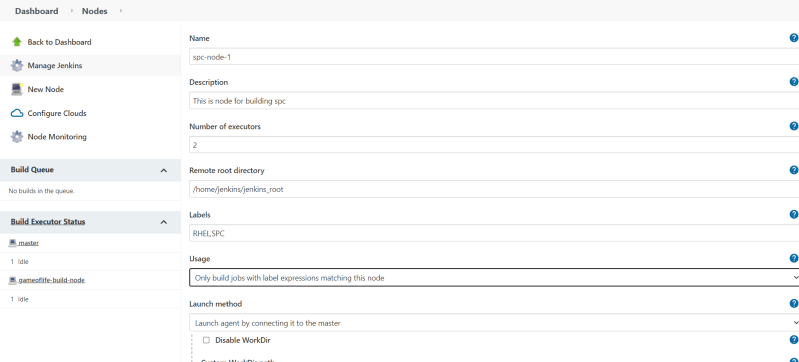
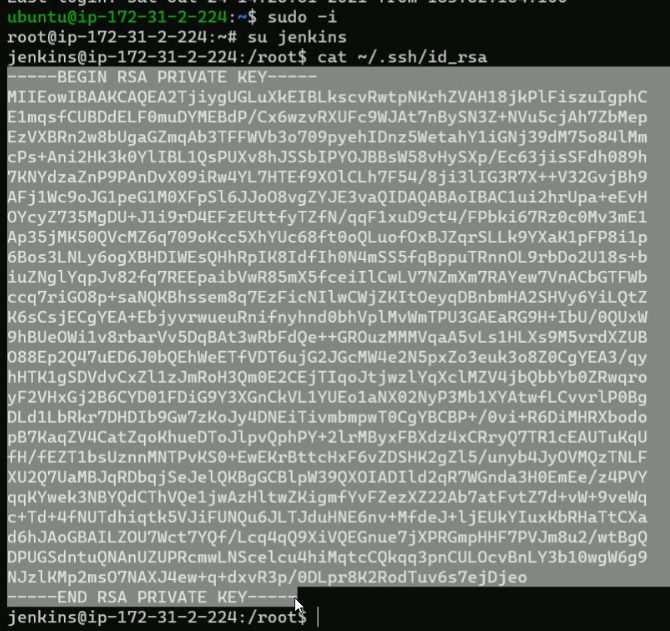
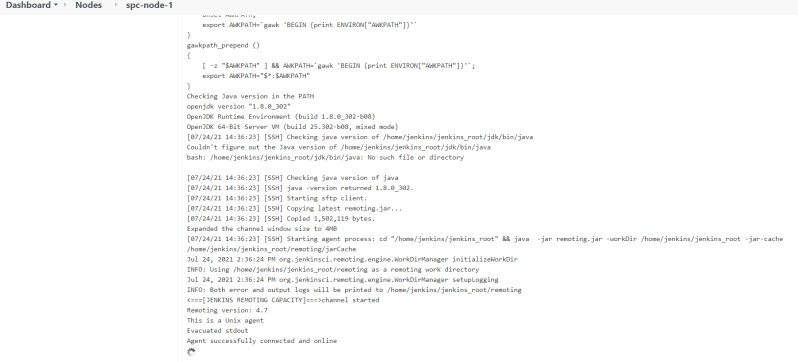
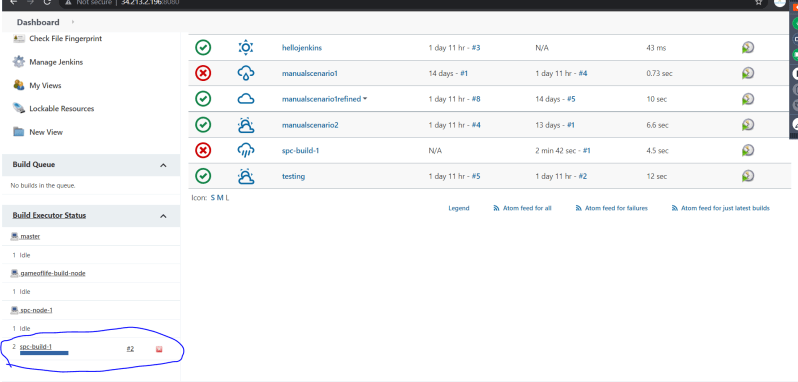
* Now copy the ssh-key (public-key) to redhat node using ssh-copy-id username@pip or private IP



LAB:

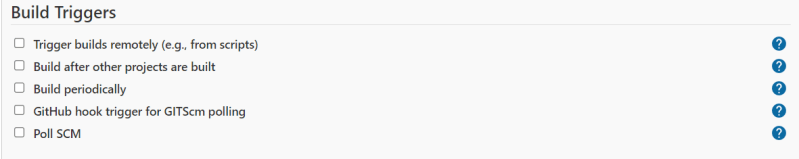
Create a home directory for Jenkins user is **#mkdir jenkins\_root**

To become as root user **#sudo -i**

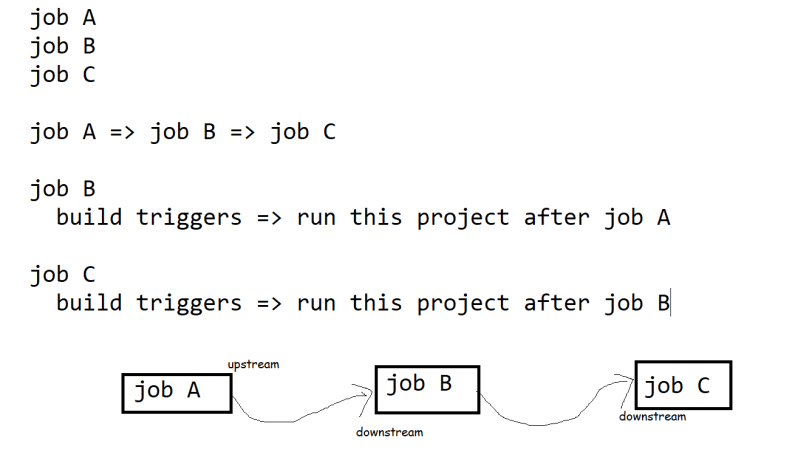
* Now jenkins master can login into redhat node with username
* Now let’s configure the jenkins node (redhat) from manage nodes
*  
* In host place select private IP of node
* In credential block select ssh with key
* 
* 
* 
* Verify, is that jenkins root directory in node having remoting.jar or not?
* 
* 
* Now we know, how to add Linux machines as nodes to the Jenkins server using ssh.

## When to build the project?

* In jenkins we can configure when to build the project using Build Triggers section



* Build after other project is build will lead to upstream and downstream jobs



* **Build periodically** helps us in running Jenkins job periodically like a cron job or scheduler.
* To configure this jenkins periodic builds the use cases might be
  + Build the project once in every hour
  + Build the project on every Friday at 11:00 PM
* To configure Build Periodically we will have a cron sequence

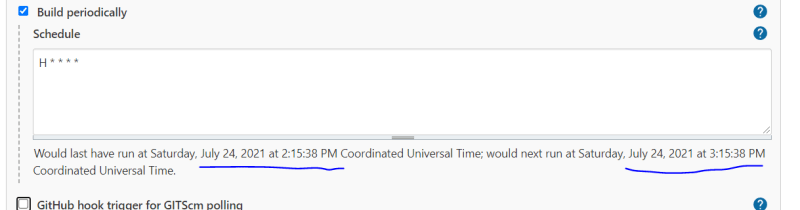
MINUTE HOUR DOM MONTH DOW

\* \* \* \* \* => every minute

For working with cron expression [Refer Here](https://crontab.guru/)

(<https://crontab.guru/>)

* Configure the job to run every hour

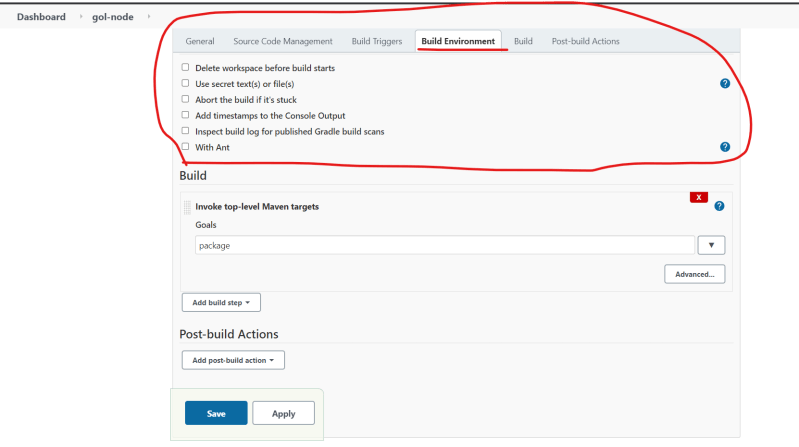
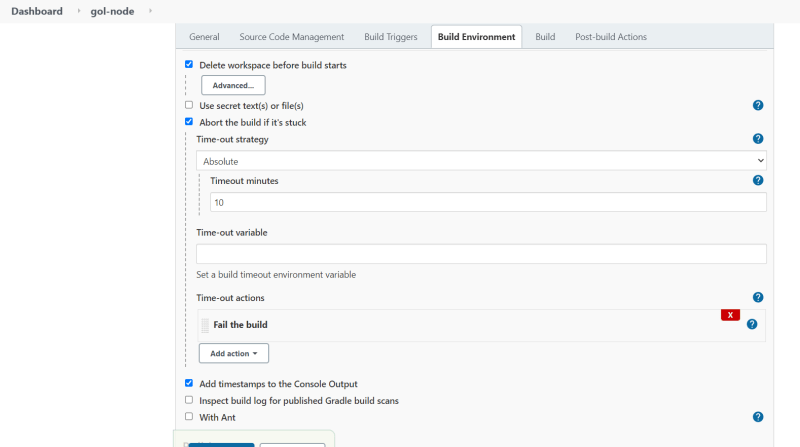
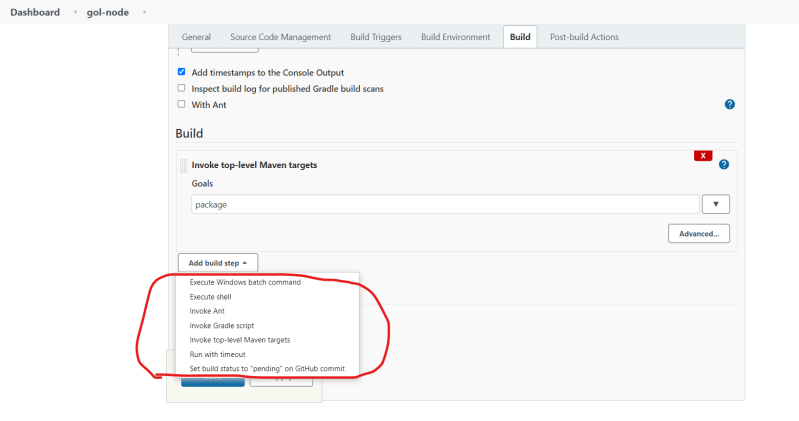


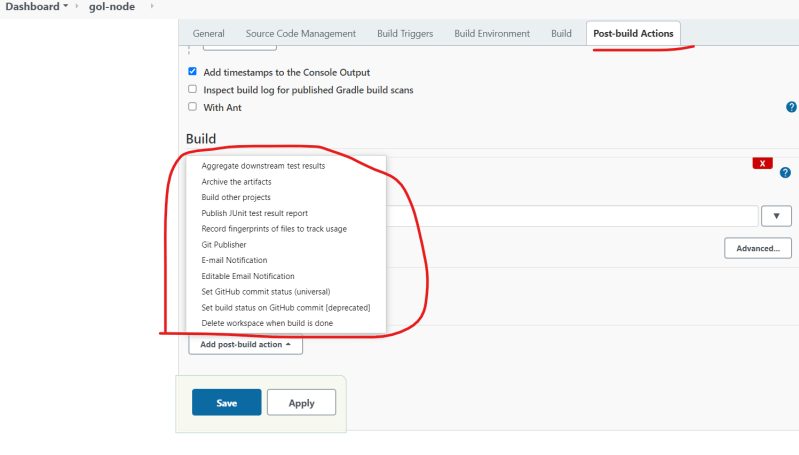
* **Poll SCM** => will build the project when there are some changes in the git repository. To configure poll scm we need to configure how frequently jenkins will poll git repository

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

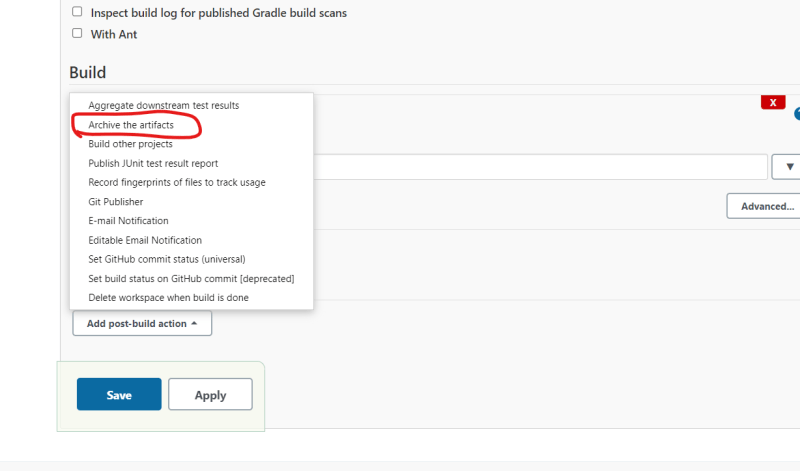
Day-7:Jenkins8- -**25/Jul/2021**

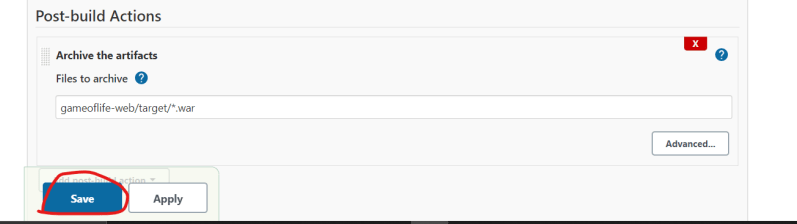
## Build Environment

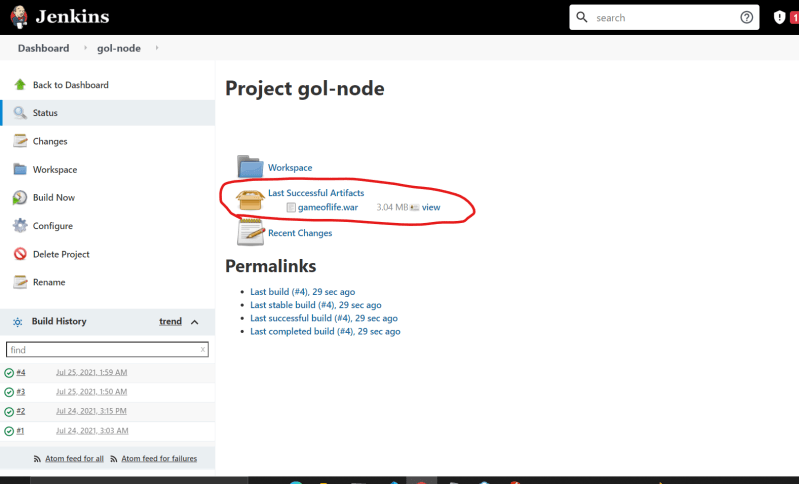
* 
* 
* Build 
* After the build is finished, the steps which we want to perform are called as **post build actions.**
  + Build might be failed
  + Build might be success
  + Build might be aborted

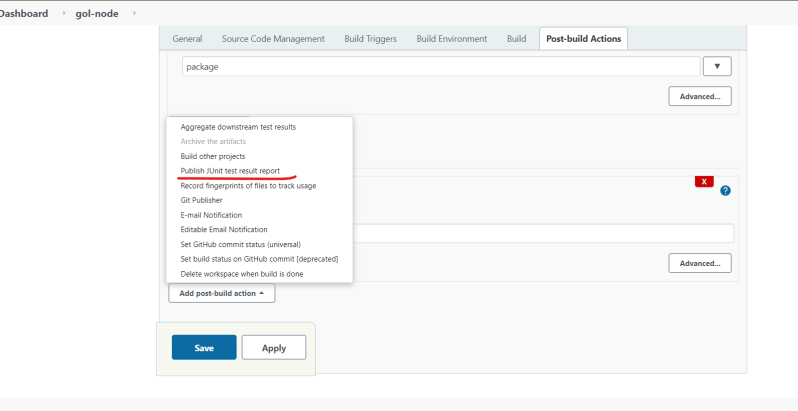
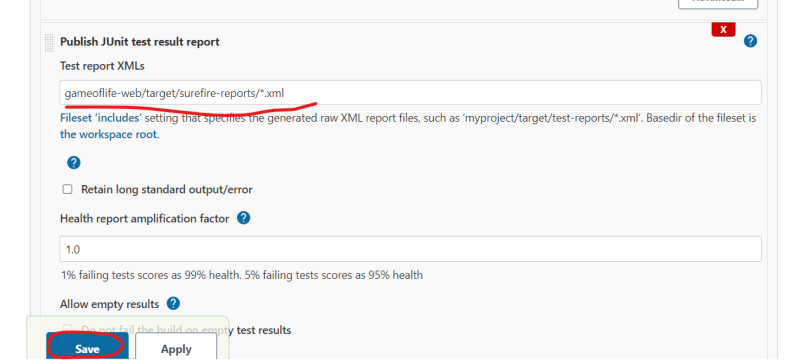
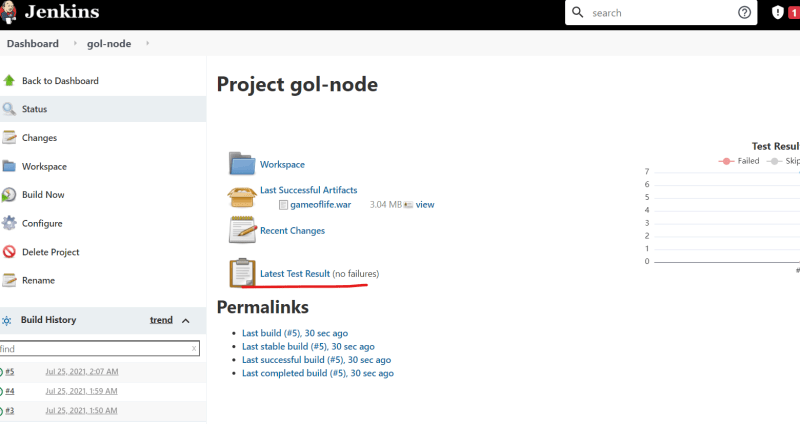


* Archiving the artifacts:
  + Using this we can archive the build artifacts that can be downloaded for the jenkins ui directly

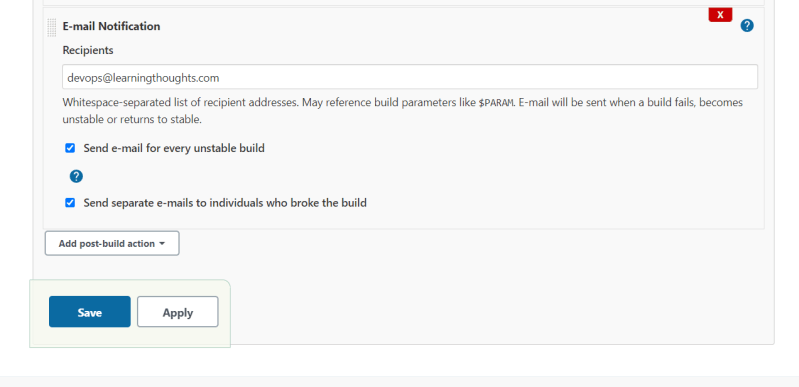






* Publishing the Junit test results:
* 
* 
* 
* 

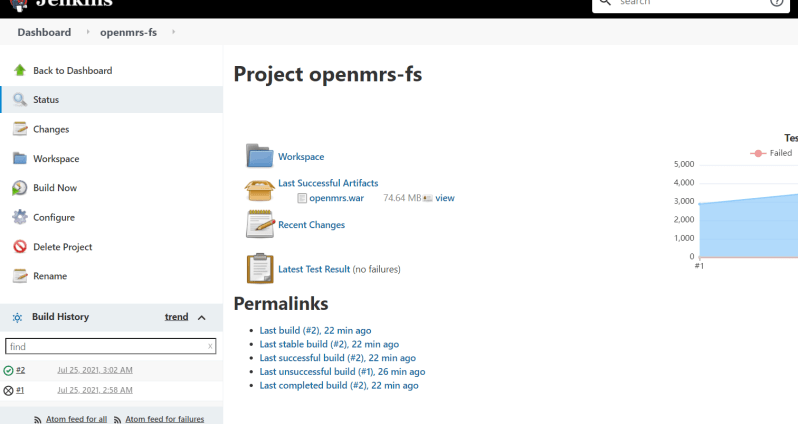
## Configuring Email Notifications in Jenkins

* For the lab setup we will be using mail trap which is simulated fake smtp server [Refer Here](https://mailtrap.io/) (<https://mailtrap.io/>)
* Navigate to Manage Jenkins => Configure System
* 
* 

## Exercise – 1

* Create a Jenkins job to build a java project openmrs [Refer Here](https://github.com/openmrs/openmrs-core). (<https://github.com/openmrs/openmrs-core>)
* You need to archive the openmrs war file and publish junit test results
* Also send email notification when the build is unstable.
* For configuring the test reports from any folder use the following expression for configuring the junit test results

\*\*/TEST-\*.xml



## Problems with Free Style Projects

* The build steps are configured in external jenkins jobs
* Changes in the build steps are not version controlled.

## Jenkins 2

* Jenkins in its newer versions started supporting pipelines-as-code feature.
* We write the build steps or the whole pipeline in a text file generally Jenkinsfile
* This Jenkins file will be part of the code
* Jenkins 2 supports two kinds of pipelines
  + Scripted Pipeline
  + Declarative Pipeline
* With Jenkins 2 the new Job Types are added
  + Pipeline
  + Folder
  + Organization
  + Multibranch Pipeline

## Syntax: Scripted vs Declarative Piplelines

* Scripted referes to the initial way that pipelines-as-code have been done in Jenkins
* Scripted syntax relies heavily on the Groovy Language and Groovy constructs for things like error checkings and dealing with exceptions
* Declarative syntax is the newer option. This is Jenkins DSL

# Scripted Pipeline

node('GOL') {

stage('SCM') {

// clone the code

git 'https://github.com/asquarezone/game-of-life.git'

}

stage('build') {

// build the code

sh 'mvn package'

}

}

# Declarative Pipeline

pipeline {

agent { label 'GOL' }

stages {

stage('SCM') {

steps {

git 'https://github.com/asquarezone/game-of-life.git'

}

}

stage('COMPILE'){

steps {

sh 'mvn package'

}

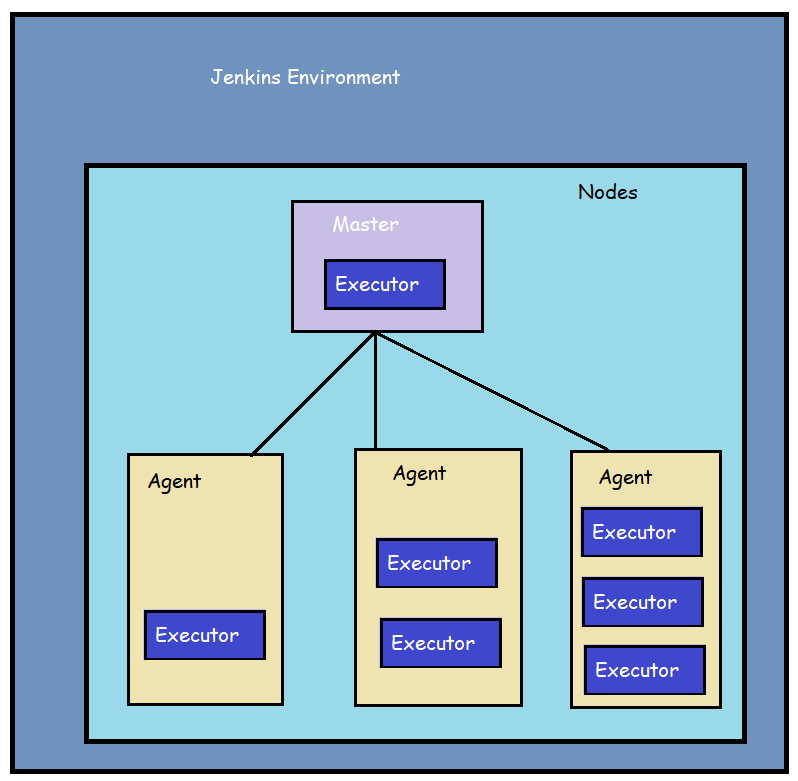
}

}

}

* Advantages of Scripted Pipeline
  + Generally fewer section and less specification needed
  + Capability to use more procedural
  + More like creating a program
  + More flexible to do custom operations if needed
  + Ability to model more complex workflows and pipelines
* Disadavantages of Scripted Pipeline
  + More programming required
  + Syntax checking limited to Groovy Language and environment
  + Further away from traditional Jenkins model
* Advantages of Declarative Pipeline
  + More Structure – close to traditional sections of Jenkins web forms(free style project)
  + More capability to declare what is need, so more readable
  + Can be generated from Blue Ocean Graphical Interface
  + Better syntax check and error identification
* Disadvantages of Declarative Pipeline
  + Less support for iterative logic
  + Still evolving
  + More rigid structure(harder to handle customizations)
  + Not suite for complex pipelines and workflows

## Foundations

* Jenkins Master
* Node
* Agent
* Executor
* 

======================================================================

Day-8:Jenkins9- -26/Jul/2021

Working with Jenkins DSL

* Jenkins DSL (Domain Specific Language) is written in Groovy Language
* Groovy makes easier for Creating DSLs and we can use the DSL without much experience with groovy.
* In this we will be looking at scripted pipeline

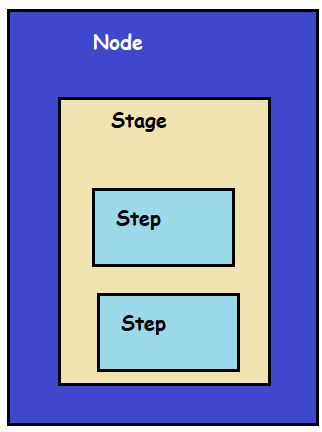
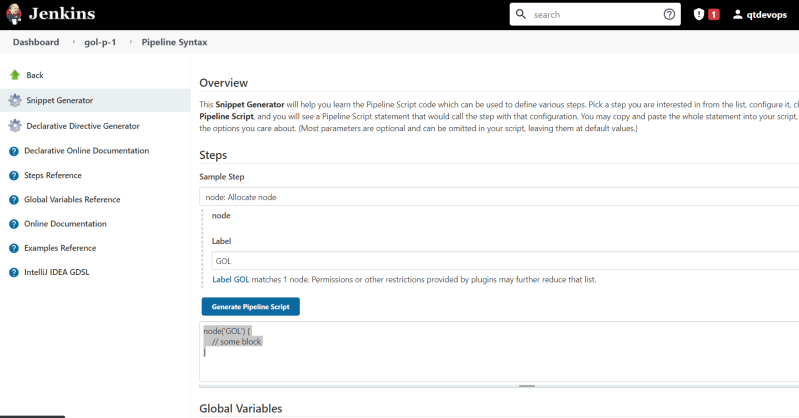
node ('GOL') {

stage('SCM') {

git 'https://github.com/asquarezone/game-of-life.git' # Step

}

}

* The node keyword is used to find the node on which the job can be executed
* stage clouser allows us to group individual steps which contains DSL commands and logic. Every stage needs to have a unique name
* With in the stage we write steps
* Relation b/w node, stage and steps
* 
* Creating Jenkinsfile steps with snippet generator
* 
* We have created a scripted pipeline script to build the game of life

node('GOL') {

stage('scm') {

git 'https://github.com/asquarezone/game-of-life.git'

}

stage('build') {

sh 'mvn clean package'

}

stage('postbuild') {

junit '\*\*/TEST-\*.xml'

archive '\*\*/\*.war'

}

}

* Exercise: Using the snippet generator create the Jenkinsfile for openmrs project
  + Fork the openmrs in your account
  + clone the repository into your system
  + Add a Jenkinsfile
  + IN the Jenkinsfile add scripted pipeline steps by using snippet generator

Day-9:Jenkins10-27/July/2021

## Pipeline Execution Flows

* Declarative pipeline
  + basic structure
* pipeline {
* agent { label 'GOL' }
* stages {
* stage('SCM'){
* //step
* }
* }
* post {
* success {
* }
* failure {
* }
* }
* }
* For all the steps [Refer Here](https://www.jenkins.io/doc/pipeline/steps/) (<https://www.jenkins.io/doc/pipeline/steps/>)
* [Refer Here](https://github.com/asquarezone/game-of-life/commit/4bc43b410f6fd0b8418cdd071033805cda55583c) for the basic declarative pipeline to build game of life (<https://github.com/asquarezone/game-of-life/commit/4bc43b410f6fd0b8418cdd071033805cda55583c>)
* To the jenkinsfile we can add build triggers by using the triggers section [Refer Here](https://www.jenkins.io/doc/book/pipeline/syntax/#triggers). The Jenkinsfile will be as shown below (<https://www.jenkins.io/doc/book/pipeline/syntax/#triggers>)

pipeline {

agent { label 'GOL'}

triggers {

cron('H \* \* \* \*')

pollSCM('\* \* \* \* \*')

}

stages {

stage('scm') {

steps {

git branch: 'master', url: 'https://github.com/asquarezone/game-of-life.git'

}

}

stage('build') {

steps {

sh 'mvn package'

}

}

}

post {

success {

archive '\*\*/gameoflife.war'

junit '\*\*/TEST-\*.xml'

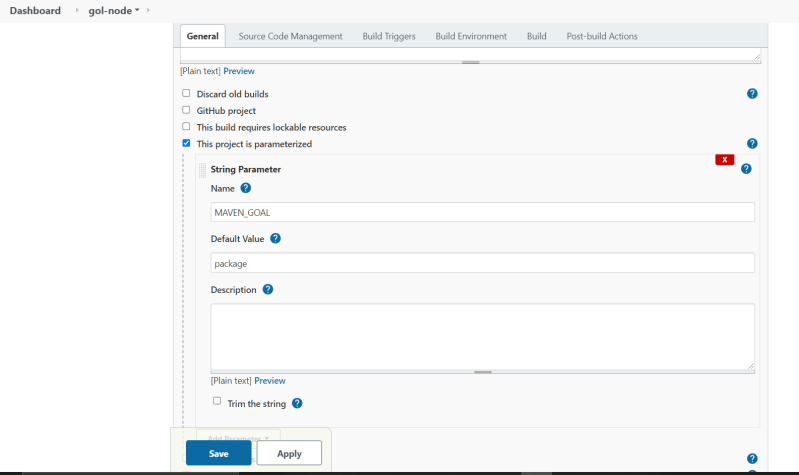
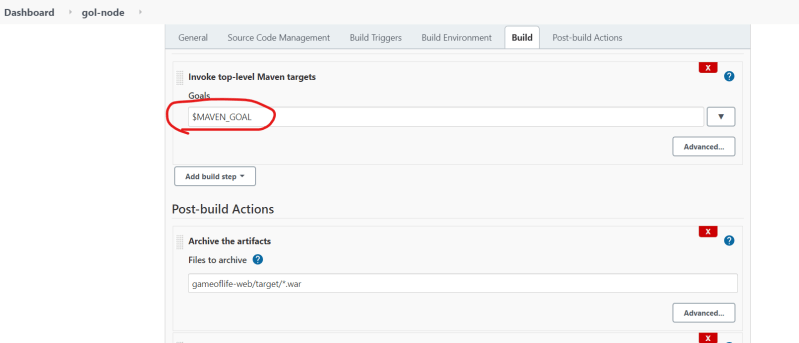
}

}

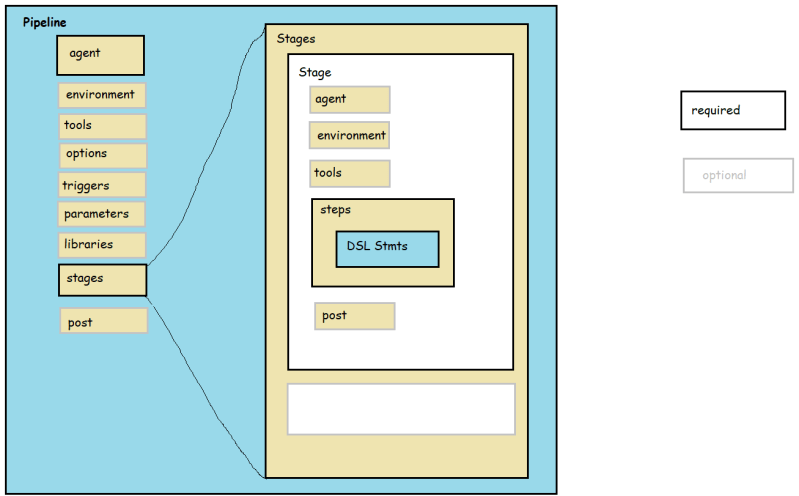
}

* Adding manual inputs from user during build [Refer Here](https://www.jenkins.io/doc/book/pipeline/syntax/#input) (<https://www.jenkins.io/doc/book/pipeline/syntax/#input>)

## Parameters in Jenkins

* A Parameter is used to store a value in variable which can be used in the later build steps.
* Let’s create a simple parameter in the Jenkins Freestyle project 
* 
* Parameters can be created from jenkins pipelines as well [Refer Here](https://www.jenkins.io/doc/book/pipeline/syntax/#parameters) (<https://www.jenkins.io/doc/book/pipeline/syntax/#parameters>)
* [Refer Here](https://github.com/asquarezone/game-of-life/commit/0ae2ffa2b1d591ba80edc4aec0cccfbe65b9ed71) for the parameters added in Jenkins pipeline.

(<https://github.com/asquarezone/game-of-life/commit/0ae2ffa2b1d591ba80edc4aec0cccfbe65b9ed71>)

* Declarative Pipeline structure
* 

Day-10:Jenkins11-27/July/2021