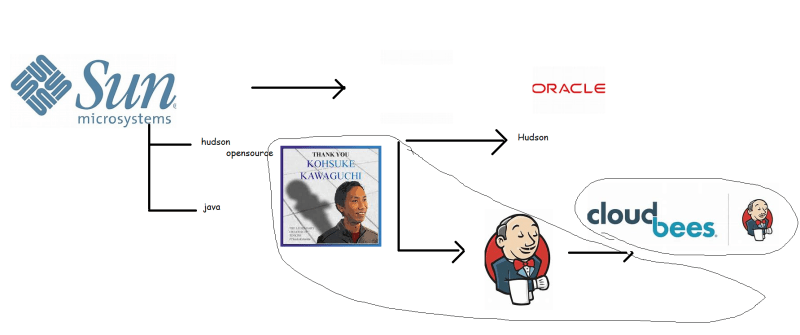
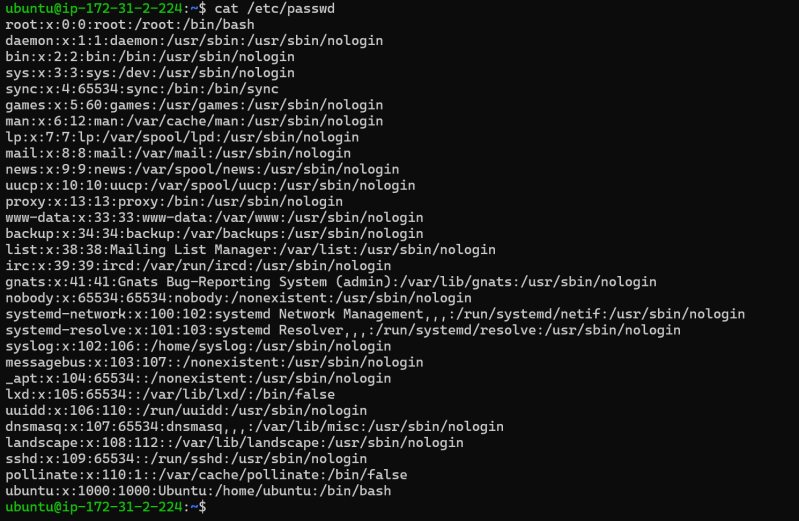
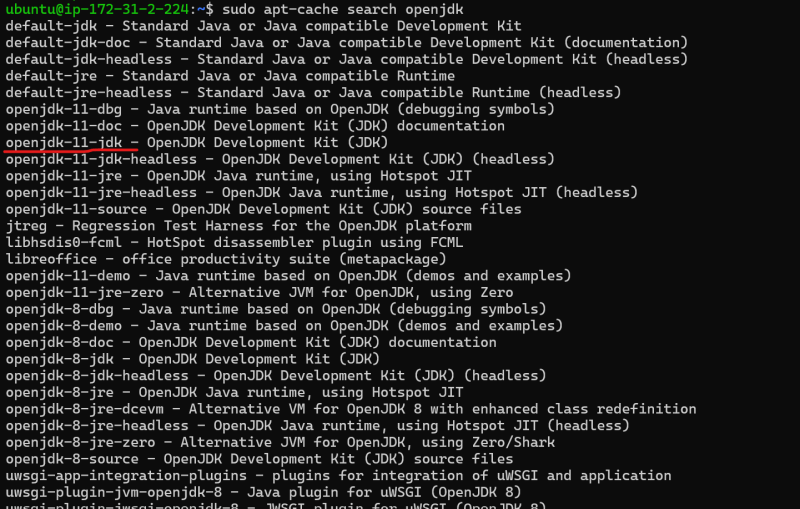
JULY 10, 2021

DevOps Classroom Series – 10/Jul/2021

Exploring Jenkins

* So lets get started by installing jenkins. [Refer Here](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 vCPUs and 8 GB of RAM
* Jenkins History 
* Installing Jenkins on Ubuntu 18.04:
  + Login into the jenkins using ssh
  + Lets 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/)
  + Lets try to install java 11 
  + Install by executing the command sudo apt install openjdk-11-jdk -y

Install jenkins by executing the below commands

sudo apt update

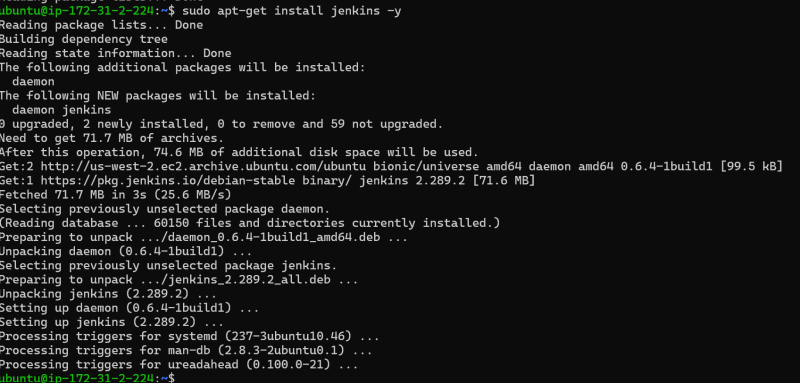
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -

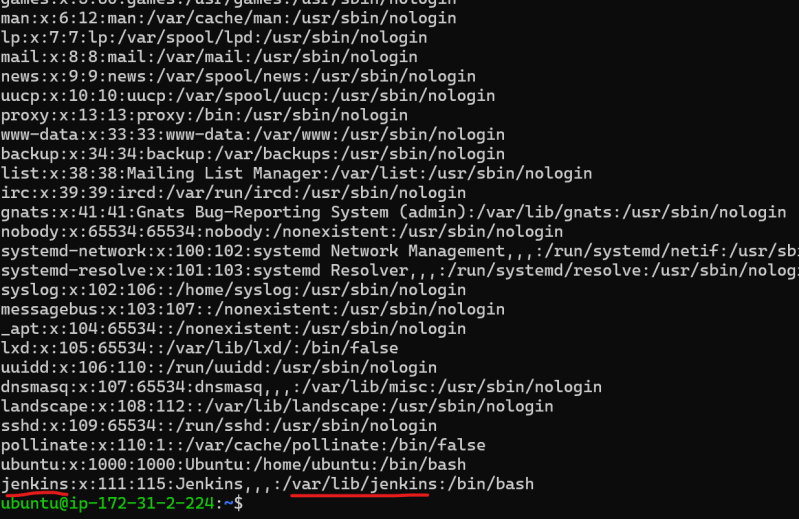
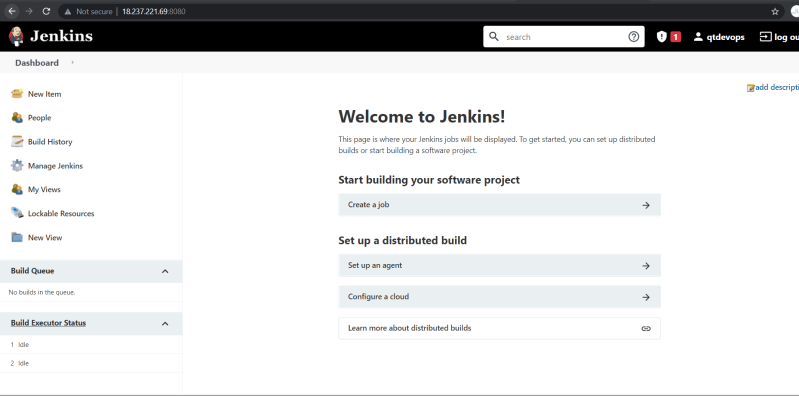
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > \

/etc/apt/sources.list.d/jenkins.list'

sudo apt-get update

sudo apt-get install jenkins

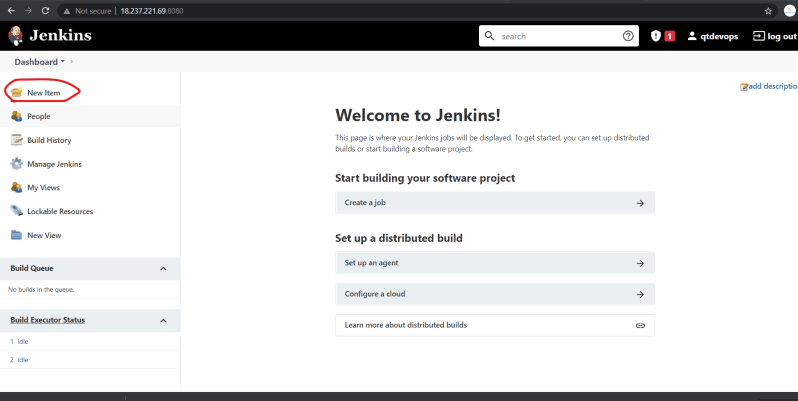
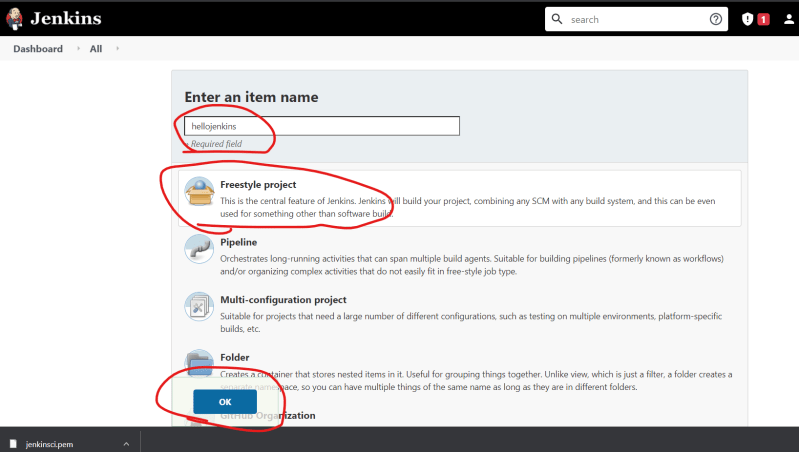
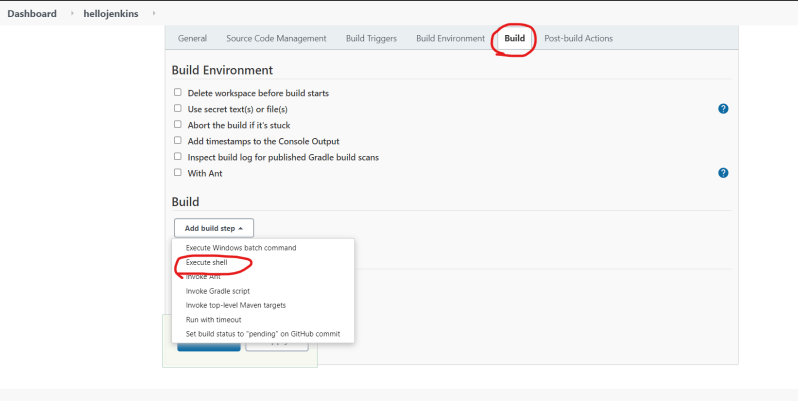
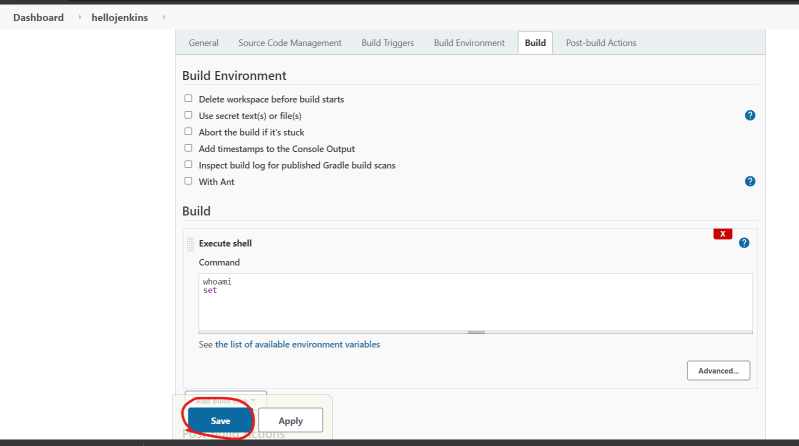
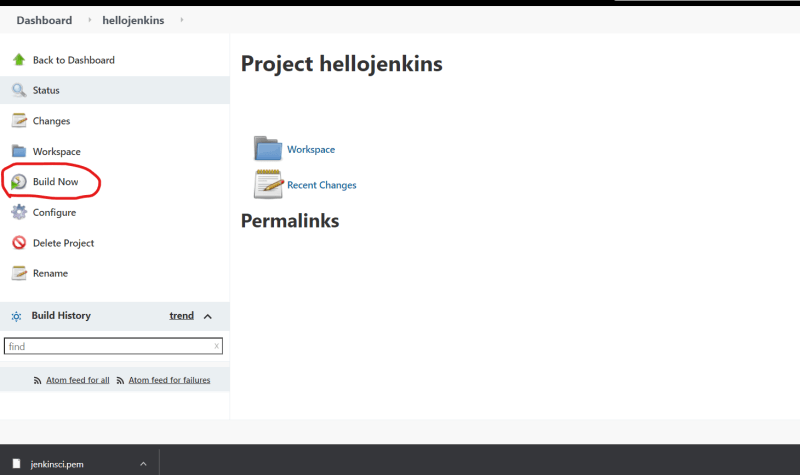
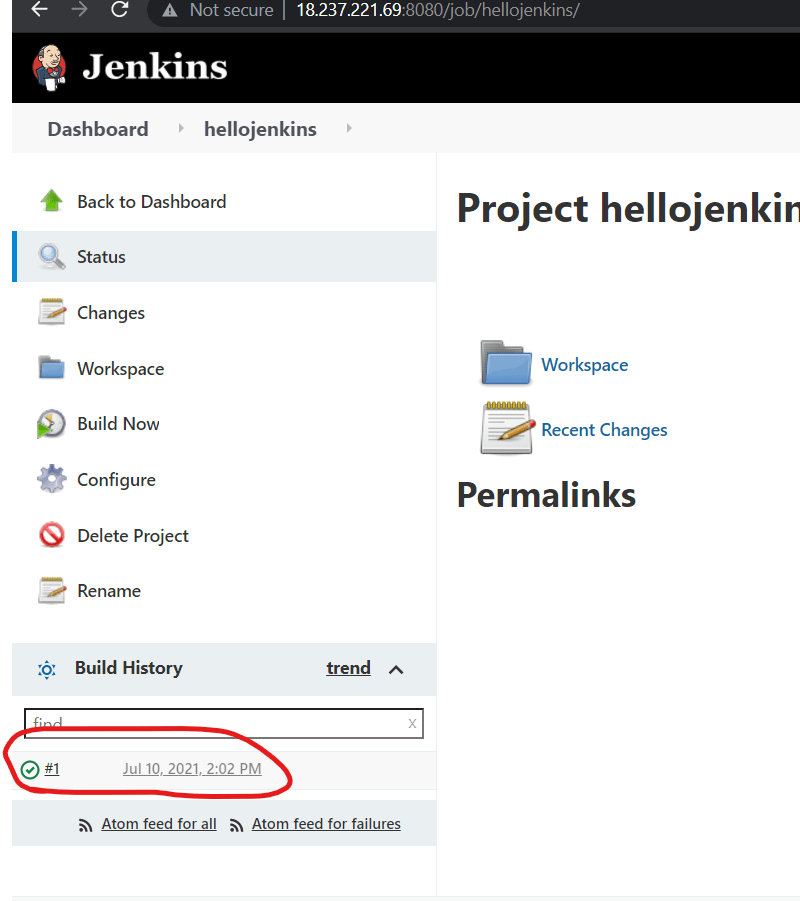
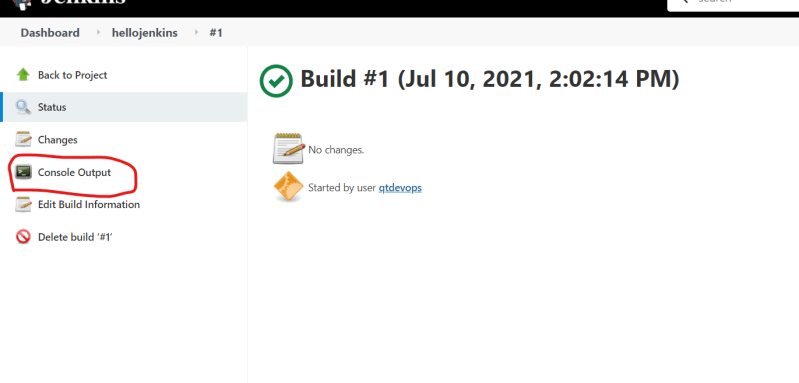
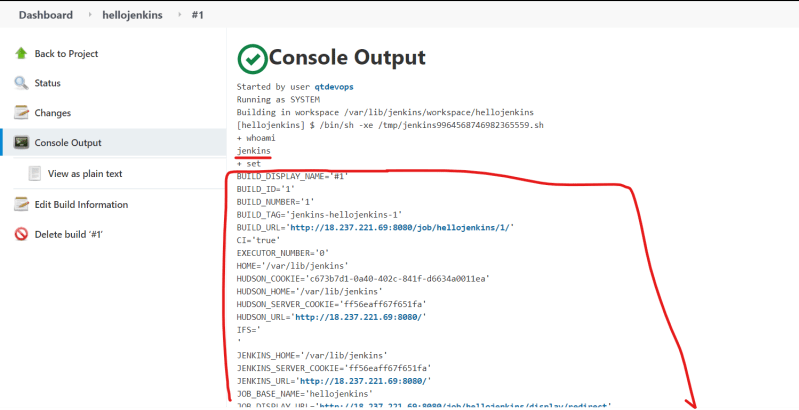


* + Now lets see the list of the users 
  + A New user called as jenkins is created and the home directory of this user is /var/lib/jenkins
  + Jenkins runs on port 8080 so navigate to http://<public-ip&gt;:8080
  + Follow the installation steps as discussed in the classroom. 

Basic Jenkins Terminology

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

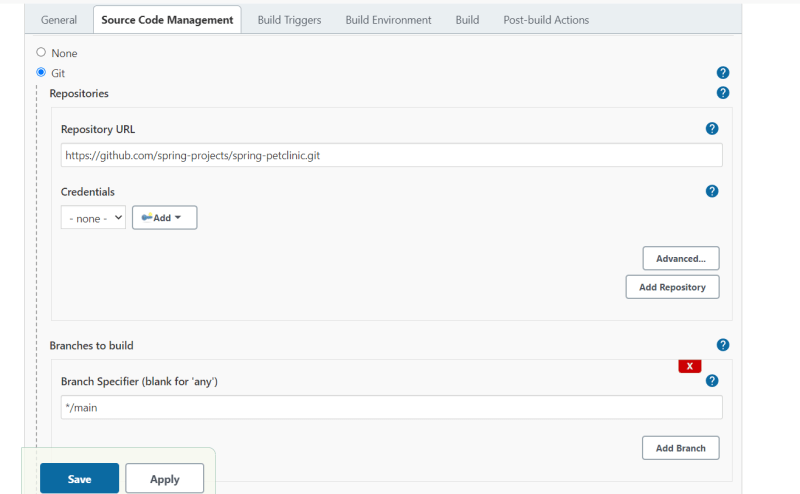
Exploring & understanding

* 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 lets take a project called as spring pet clinic [Refer Here](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

* Lets try to clone the spring pet clinic repository as mentioned above
* Lets look at git plugin 
* plugin is a UI developed which will translate into low level commands. These UI will help in doing the job easily.

Lets 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

mvn package

* To make this command work install maven

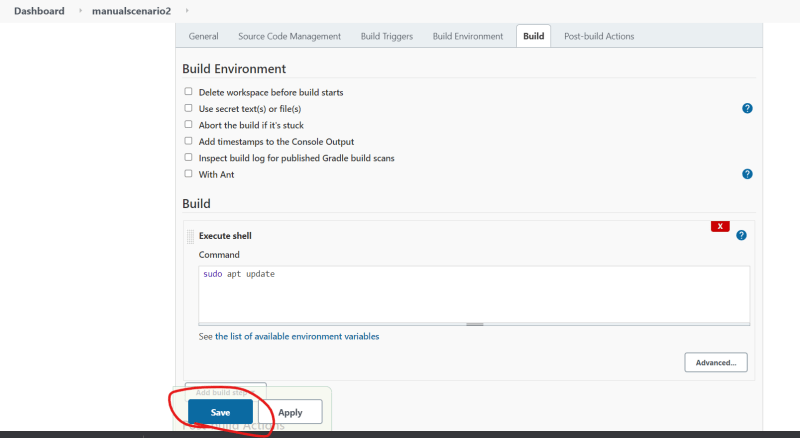
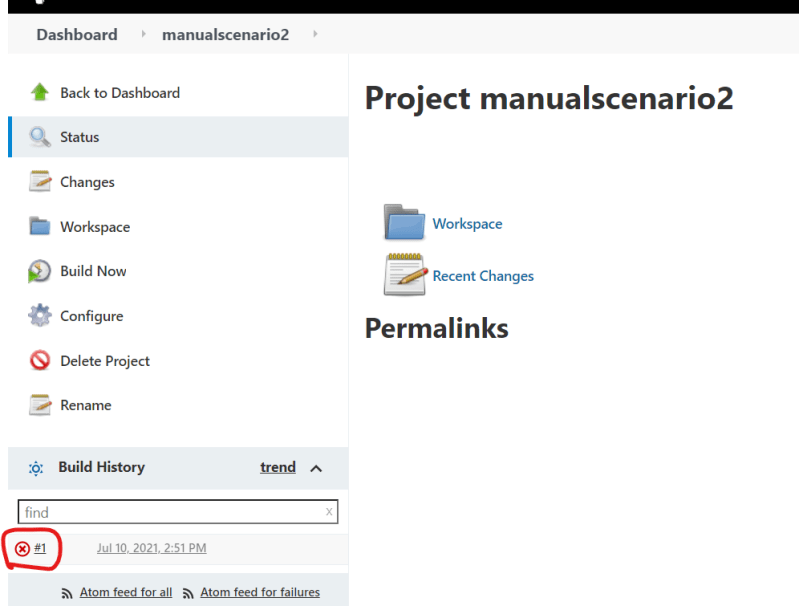
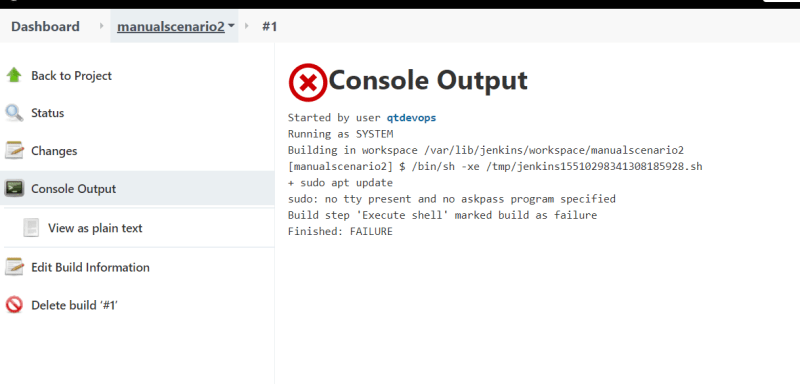
sudo apt install maven -y

* If we can execute the command manually on the linux machine it will work from jenkins as well.

Lets try to execute sudo commands from jenkins

* The commands to be executed are

sudo apt update

* Lets create the jenkins job 
* The Status is  
* Just because commands work on the system doesnot 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

**Share this:**

JULY 12, 2021

# DevOps Classroom Series – 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
* 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
* [Refer Here](https://github.com/piona/ant-sample) for the sample java project with ant
* 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
* Lets 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 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

* + - 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 chocolatey [Refer Here](https://chocolatey.org/install)

choco install jdk11 -y

choco install maven -y

* On Mac install homebrew [Refer Here](https://brew.sh/)
  + Java [Refer Here](https://mkyong.com/java/how-to-install-java-on-mac-osx/)
  + maven [Refer Here](https://formulae.brew.sh/formula/maven)
* 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

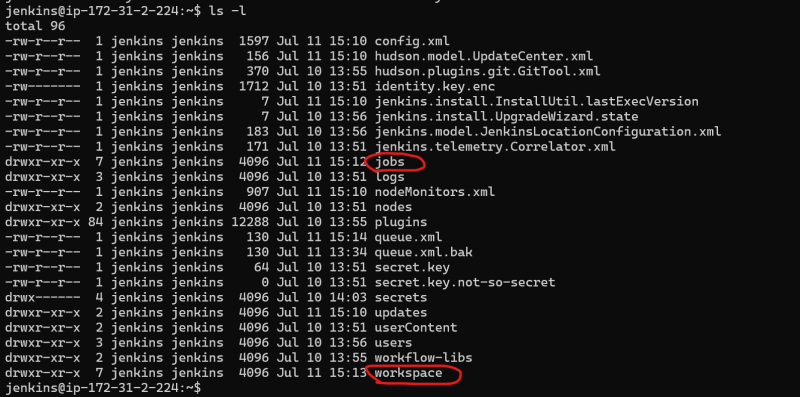
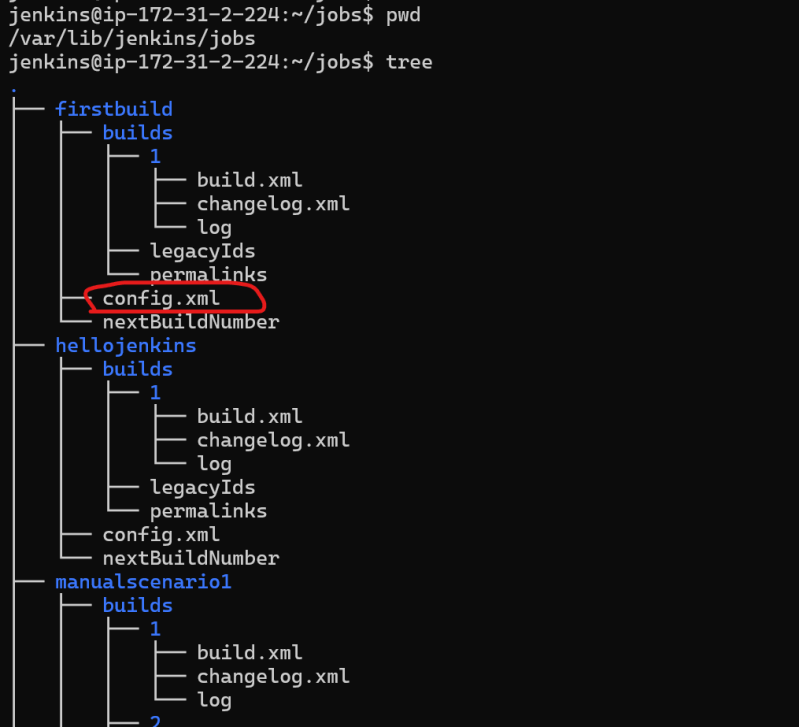
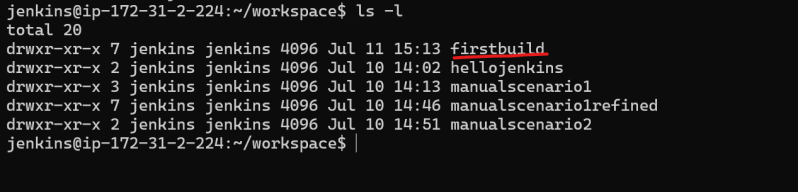
<artifactId>-<version>.<packagingformat>

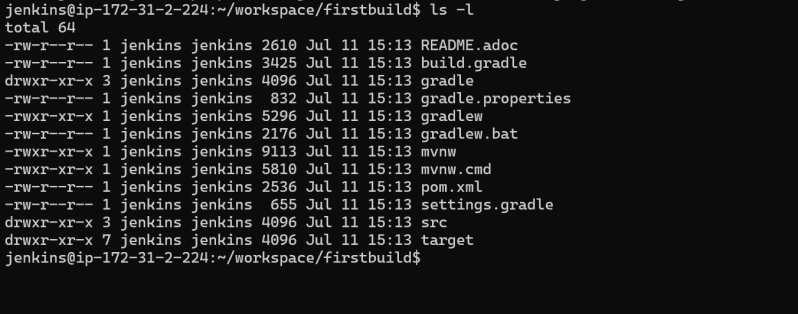
* Super POM:
  + This is Maven’s default POM. All POMs inherit from this parent or default POM
  + Try executing

mvn help:effective-pom

* What are Repositories in maven?
* What is Build life cycle in maven
* What are Build profiles
* What are Plugins

## Jenkins contd

* From Jenkins lets 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)
* Now lets 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 first build 

* 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>/

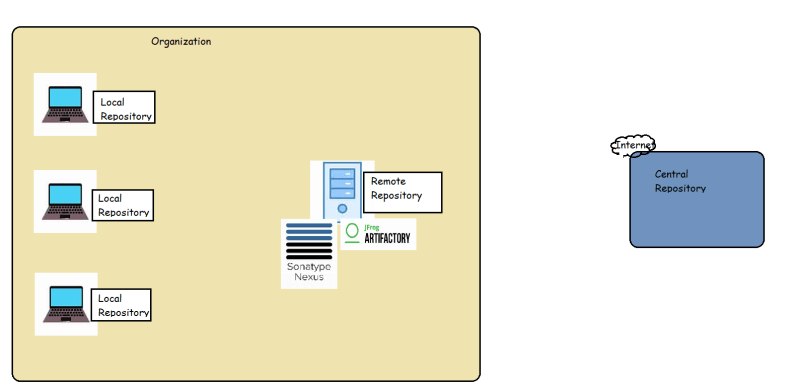
JULY 13, 2021

# DevOps Classroom Series – 13/Jul/2021

## Maven continued

* Build Profile:
  + 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 th M2\_HOME/conf/settings.xml

## Maven Repository

* In Maven, repository is a directory where all the project jars, library or any other project specific artifacts 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 does not find the dependency in your local repo it will start searching in central repo [Refer Here](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>

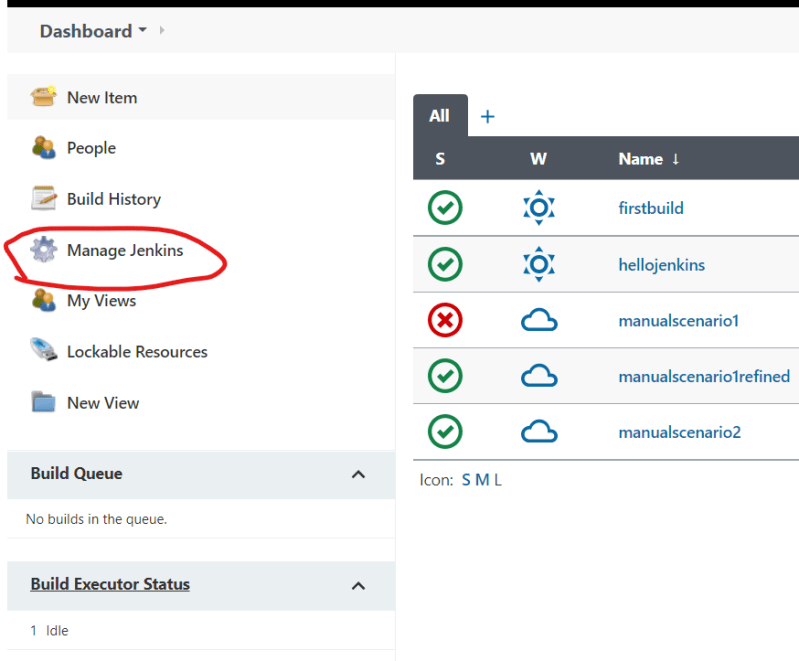
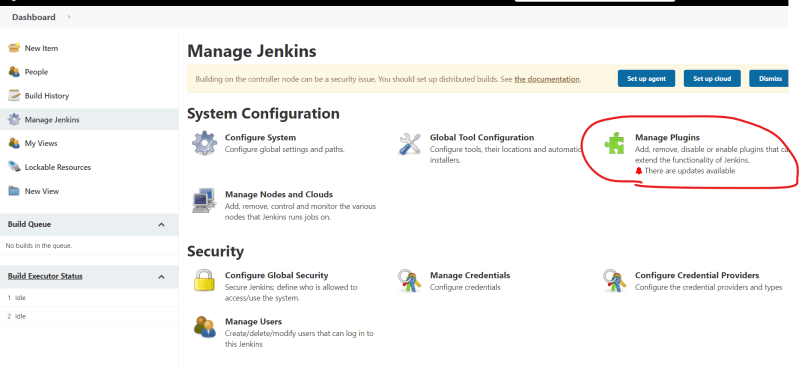
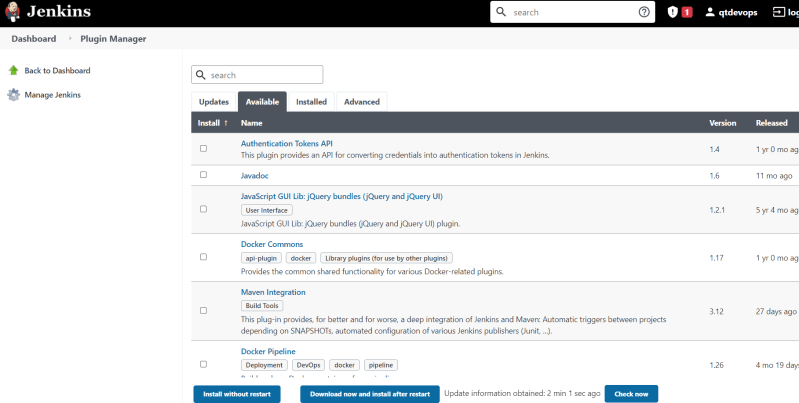
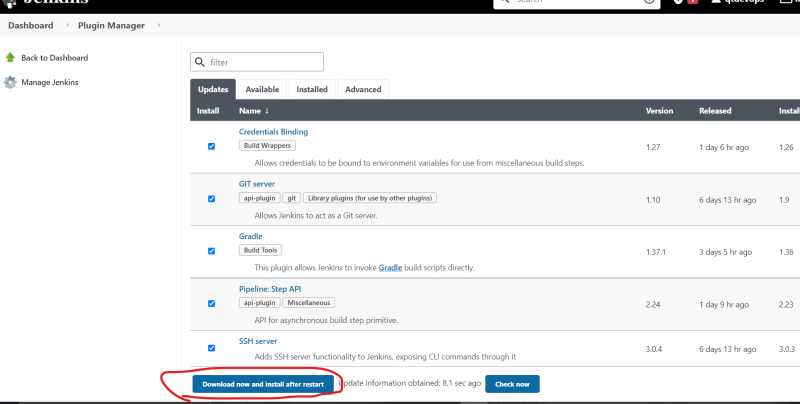
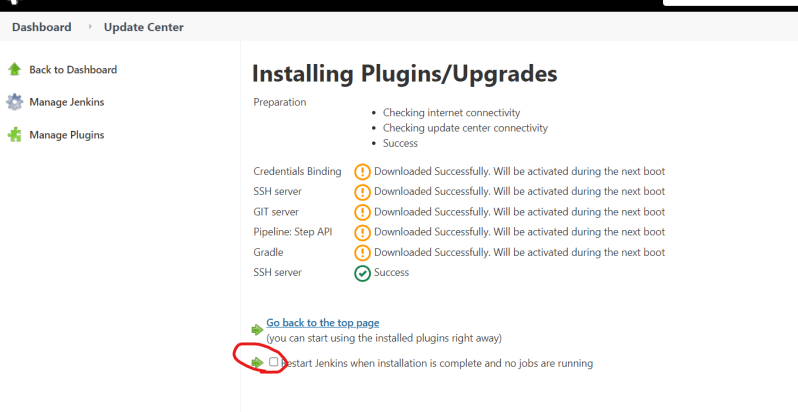
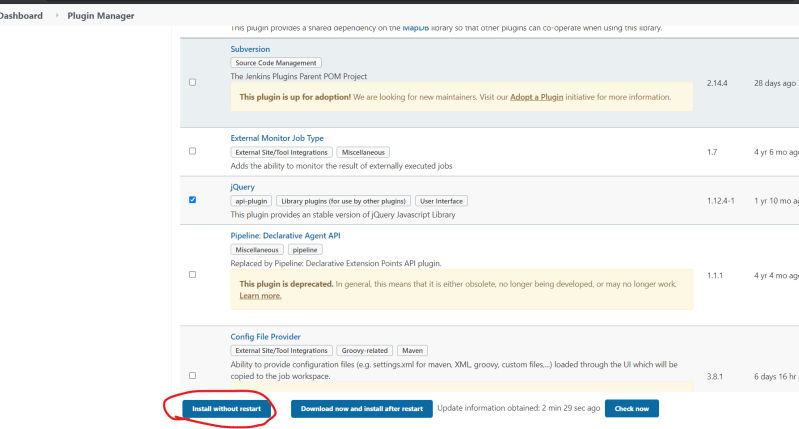
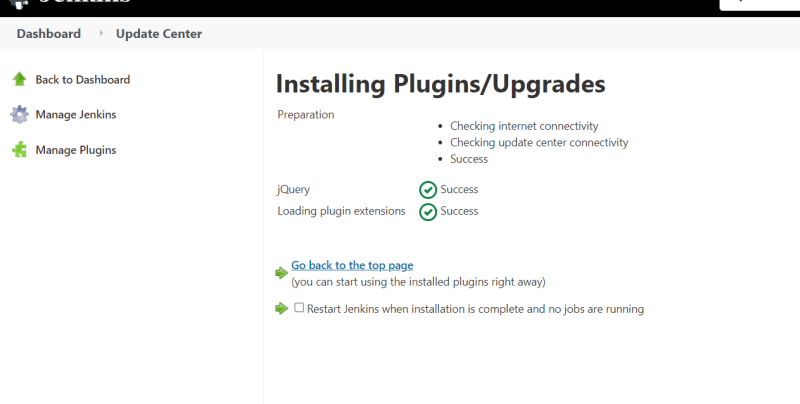
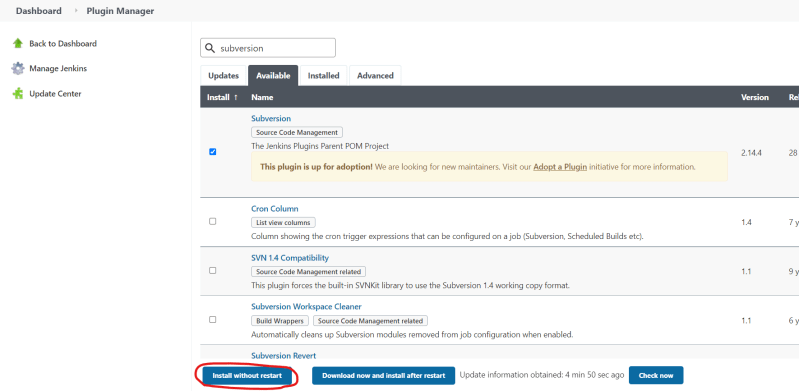
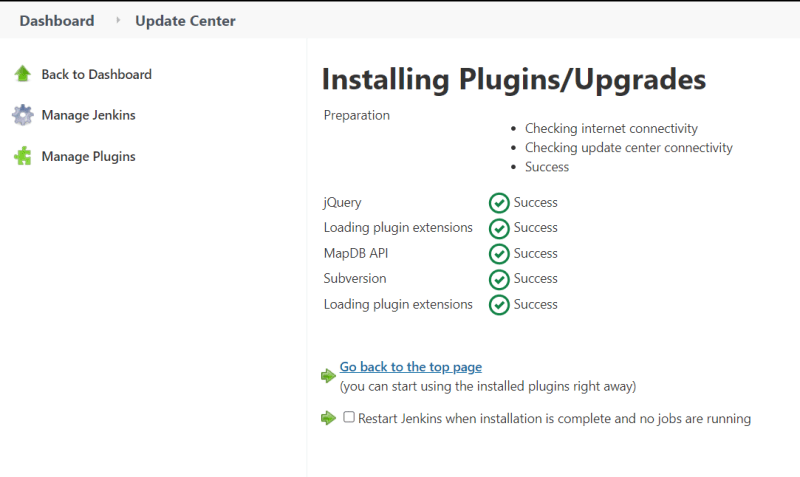
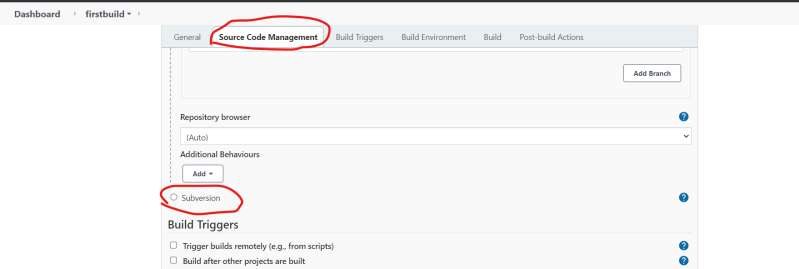
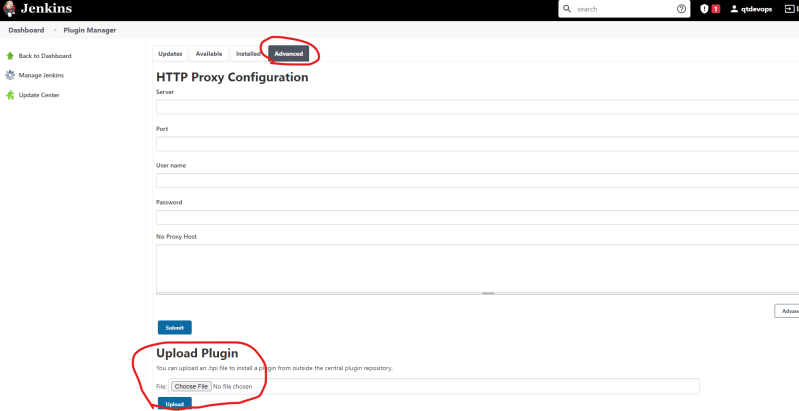
</project>

* 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 artifacts into some repository

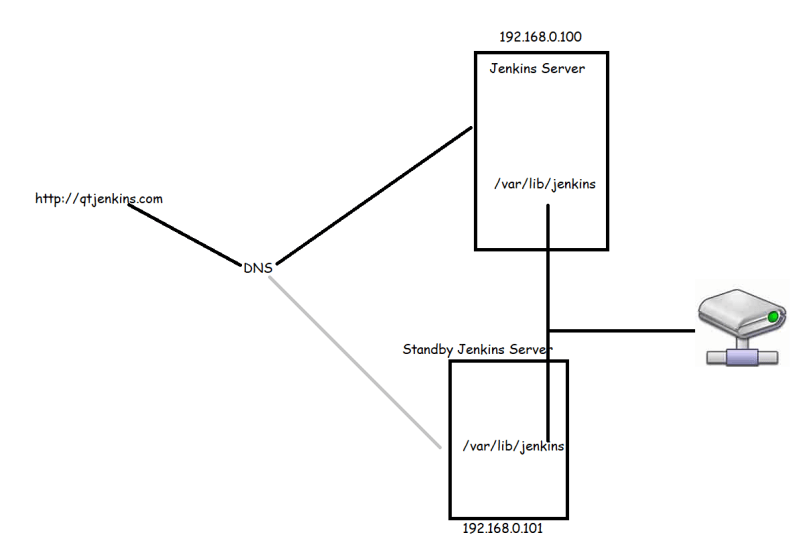
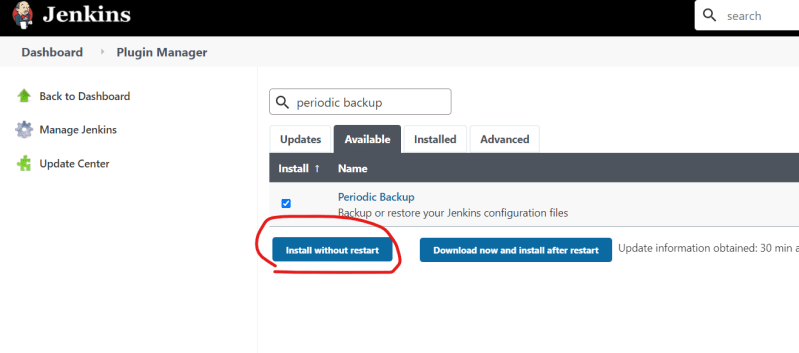
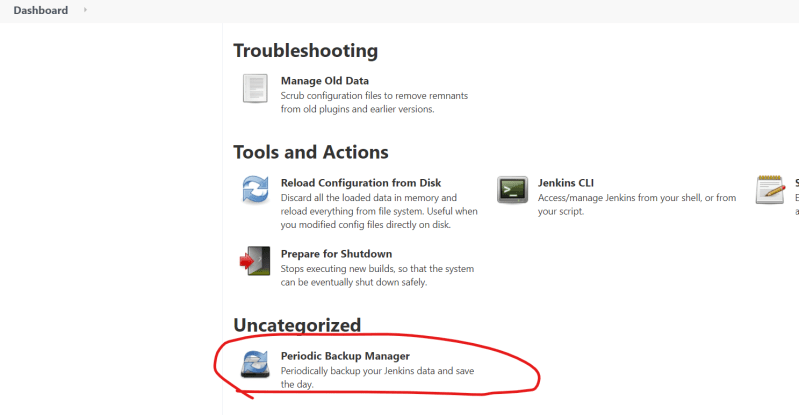
JULY 14, 2021

# DevOps Classroom Series – 14/Jul/2021

## 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 Plugin Manager   
* Updating existing plugins  
* Installing plugins  
* Lets 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/)

## 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.

### Share this:

JULY 23, 2021

DevOps Classroom Series – 23/Jul/2021

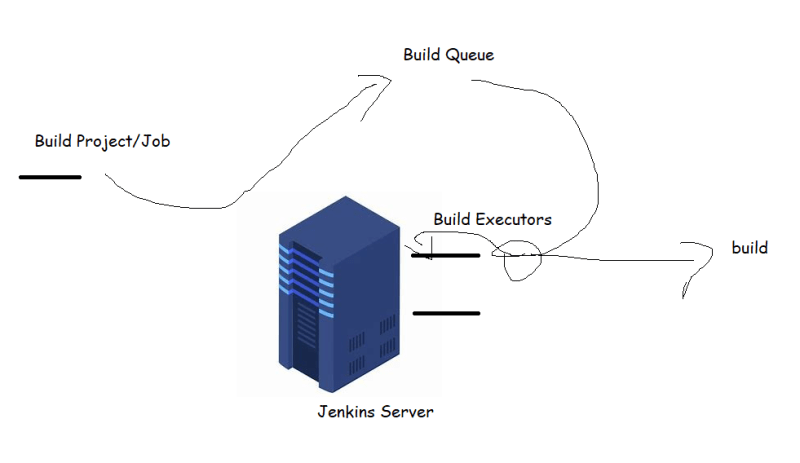
Jenkins User Management

* Authentication Methods
  + Delegate to Servlet container
  + Jenkins own user database
  + LDAP
  + 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)

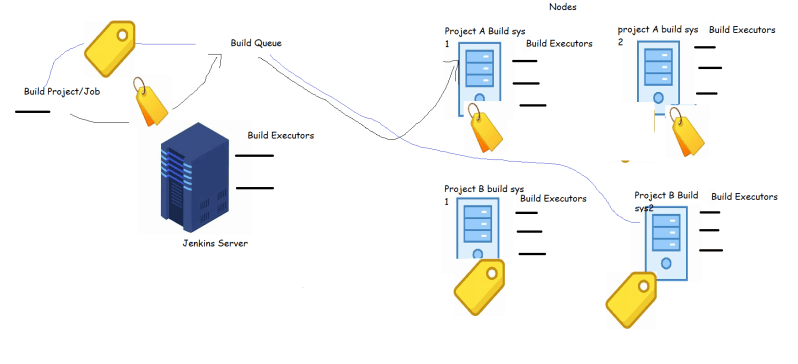
Global Tool Configuration

* 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

Build Executors and Queues



Distributed Builds



JULY 24, 2021

DevOps Classroom Series – 24/Jul/2021

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/softwares 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

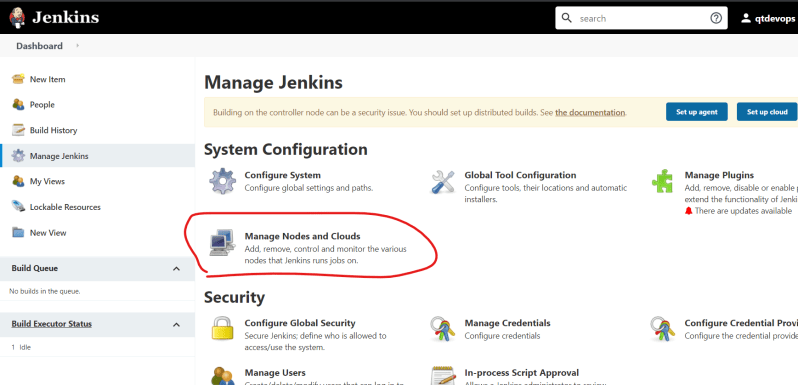
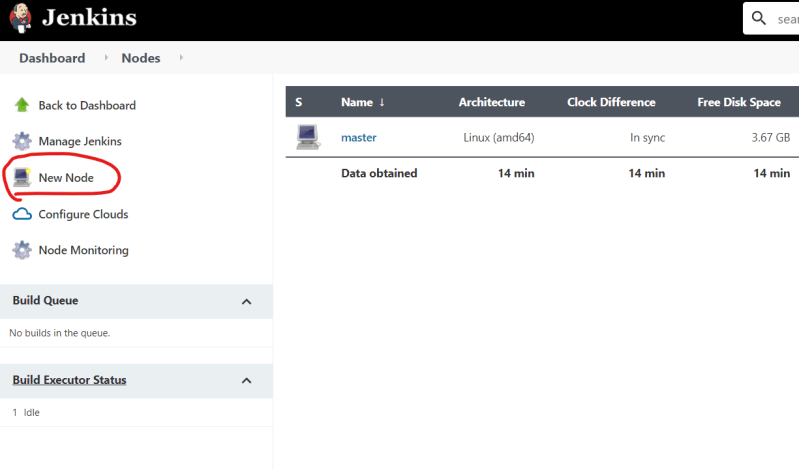
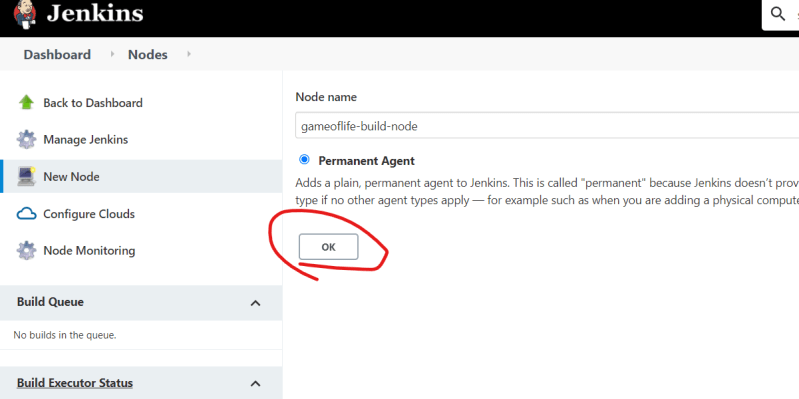
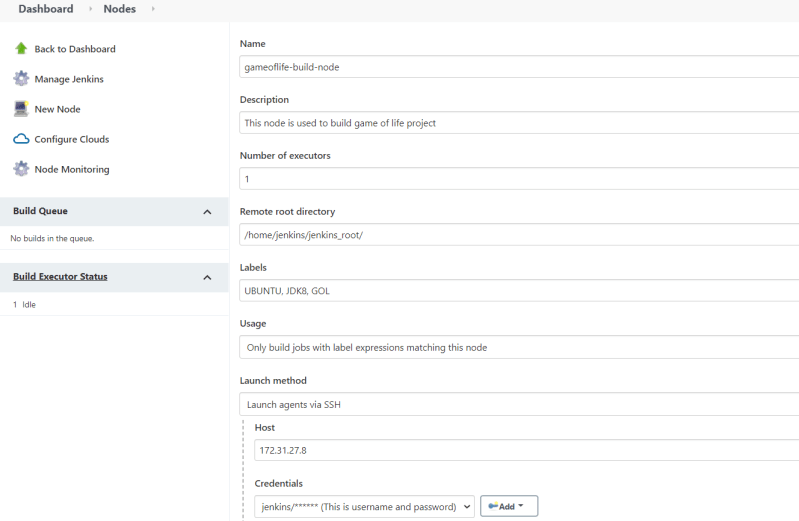
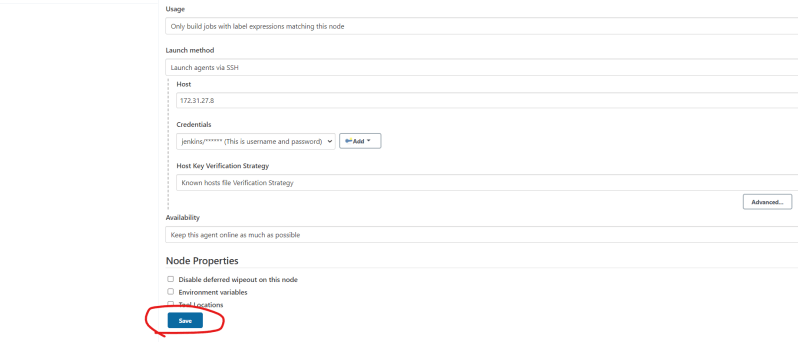
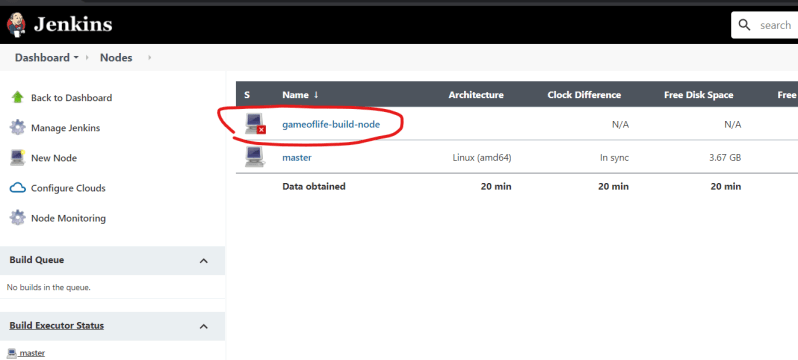
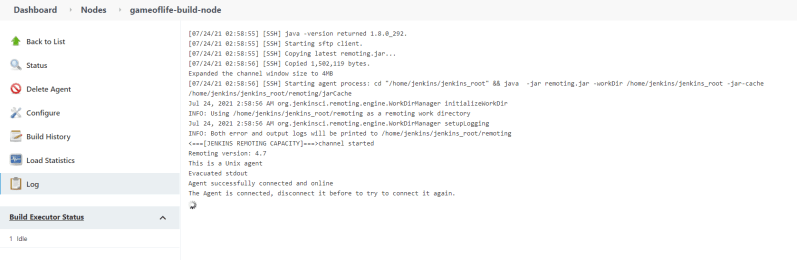
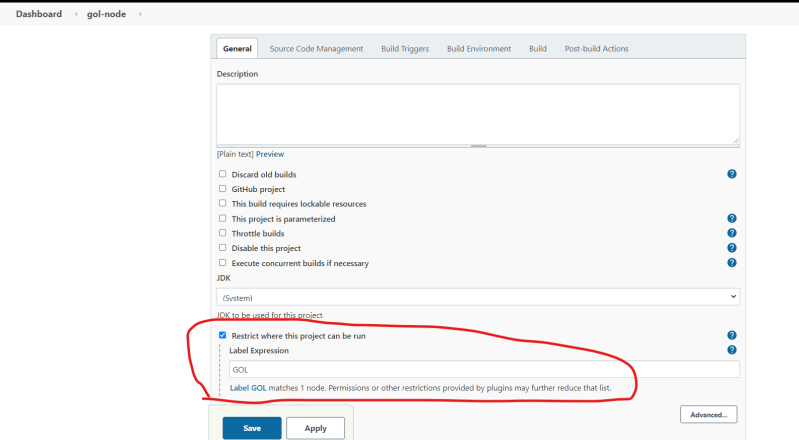
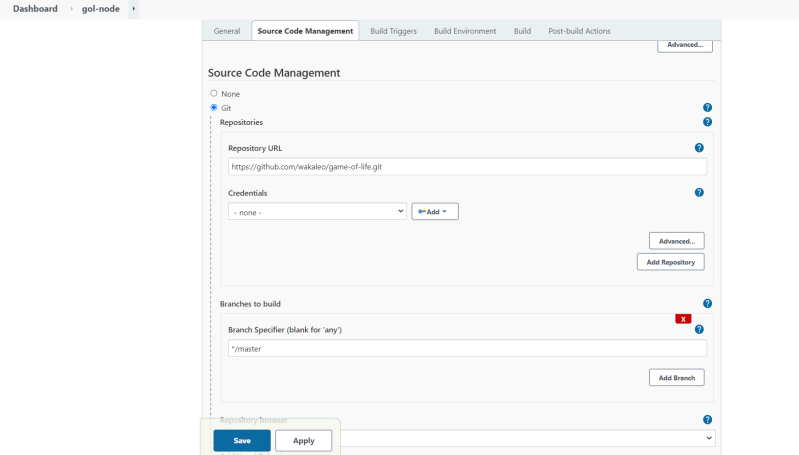
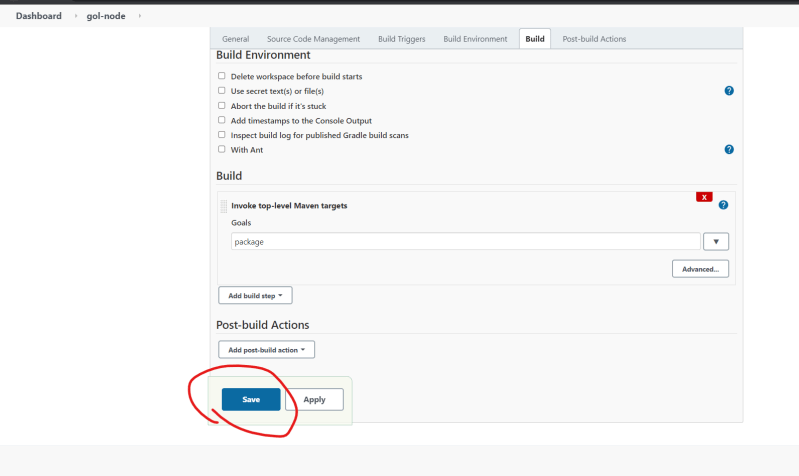
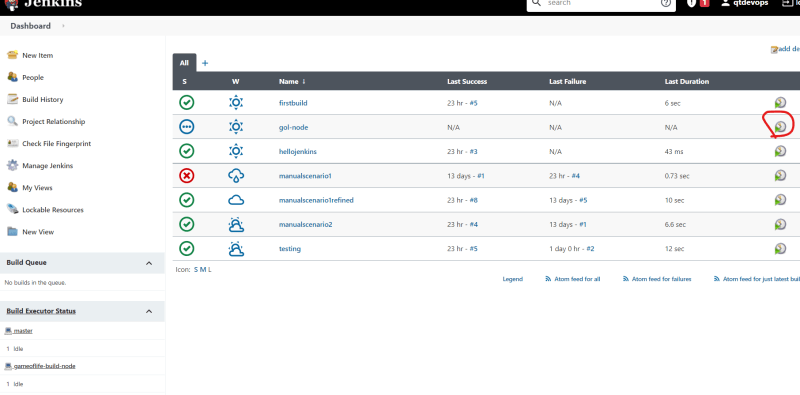
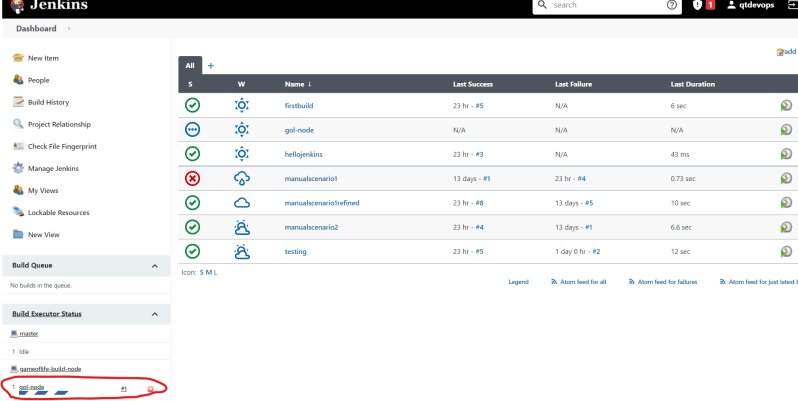
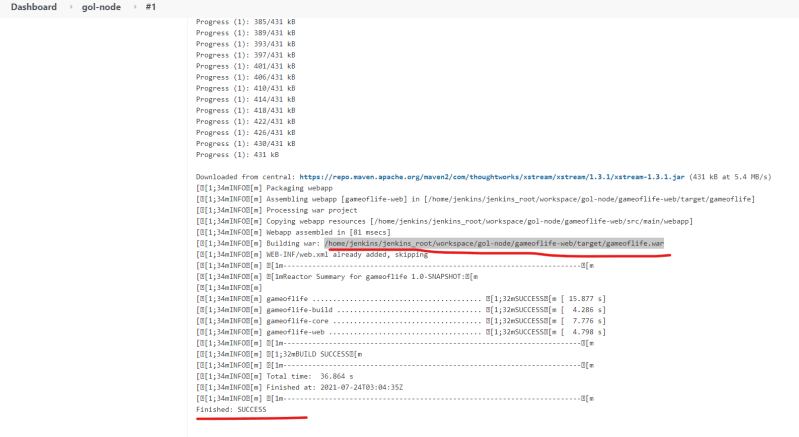
sudo apt update

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 Password Authentication value in /etc/ssh/sshd\_config
* Ensure a user is created for running builds on jenkins node

sudo adduser jenkins

* Give the user necessary permissions
* Try to login into jenkins node from jenkins server as jenkins user
* Now let’s try to configure this node from jenkins ui       
* 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, post build actions etc..

JULY 25, 2021

# DevOps Classroom Series – 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

* 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
* Now jenkins master can login into redhat node with username
* Now lets configure the jenkins node (redhat) from manage nodes
* 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

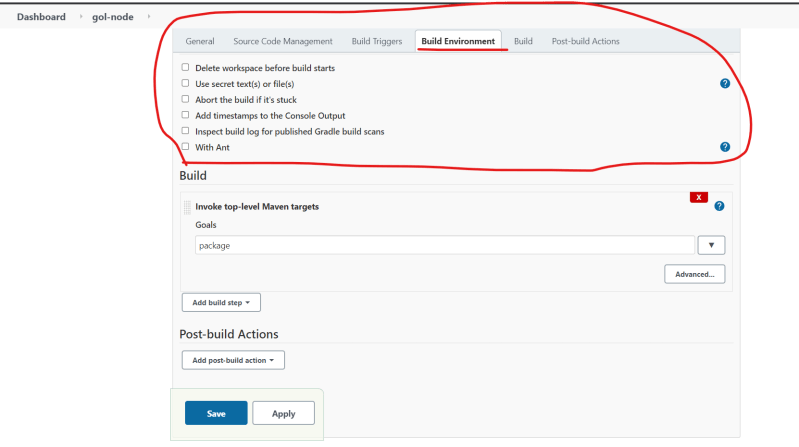
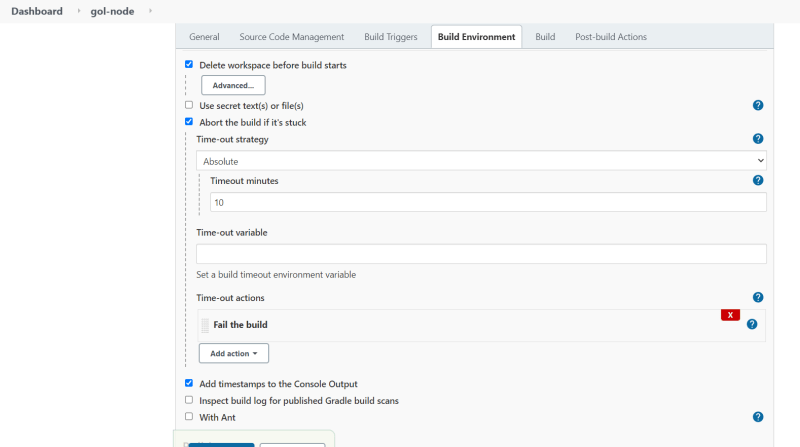
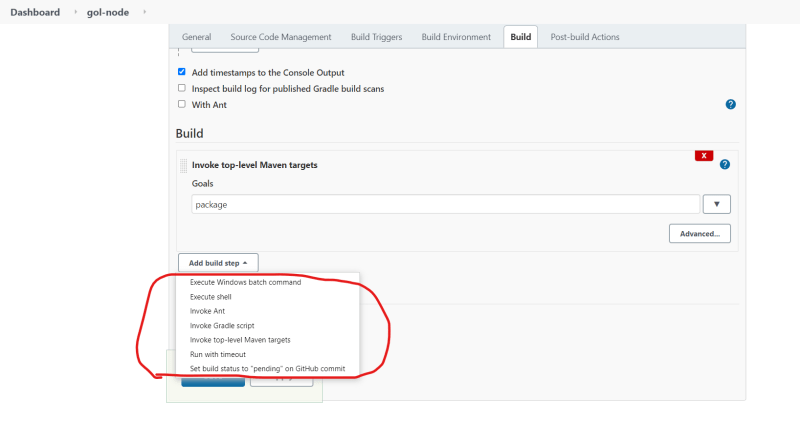
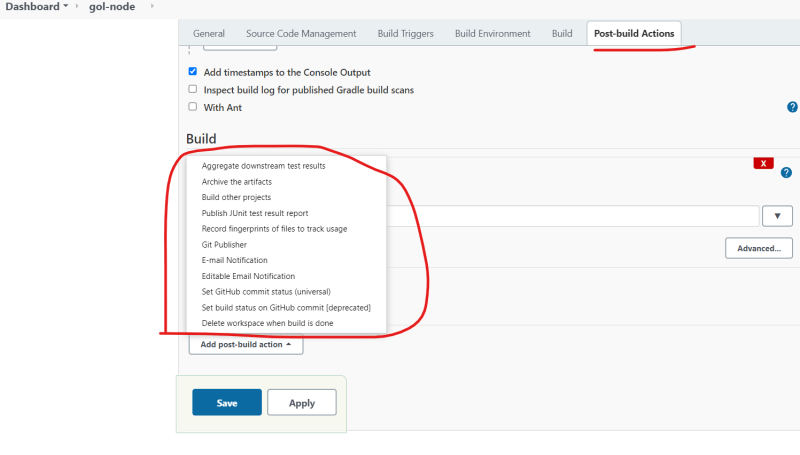
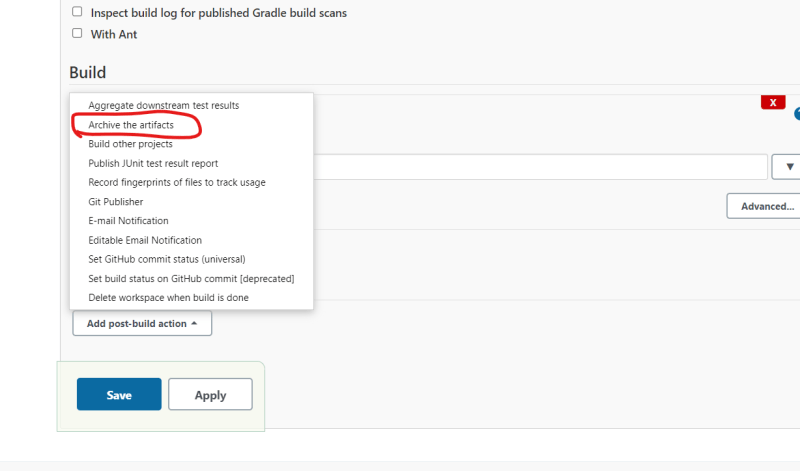
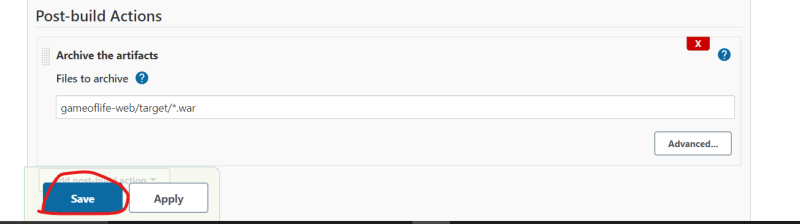
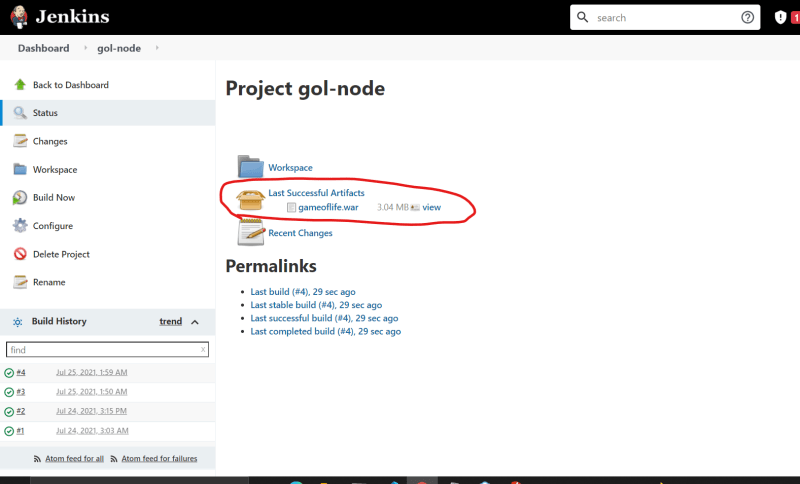
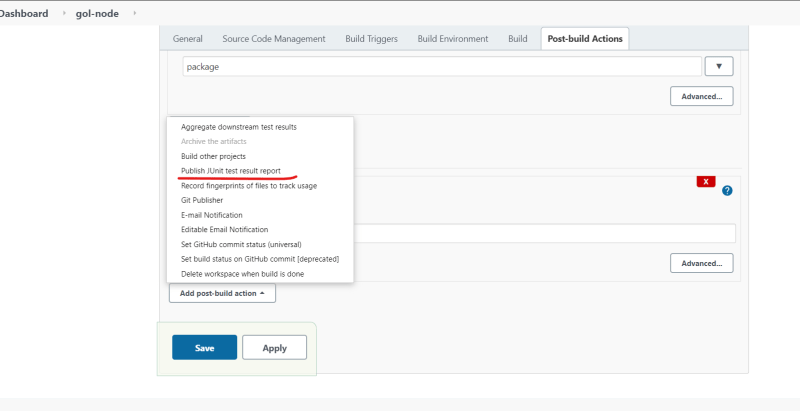
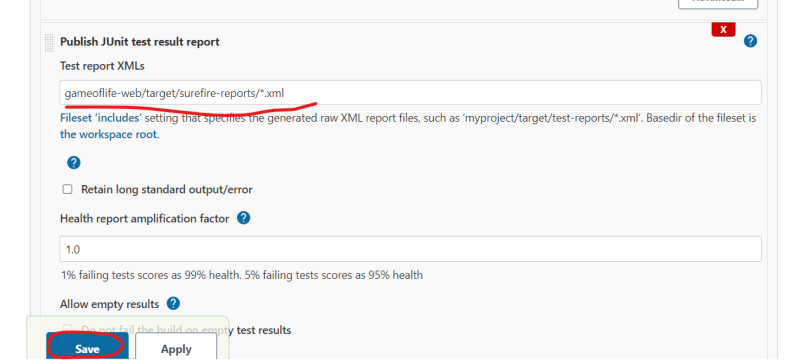
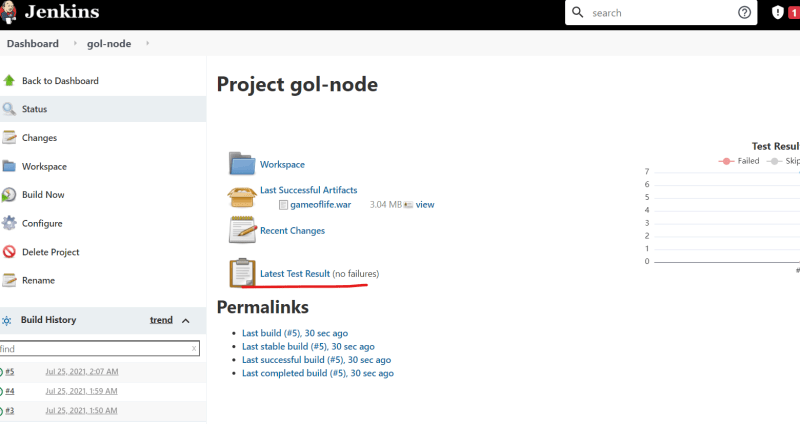
\* \* \* \* \* => every minute

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

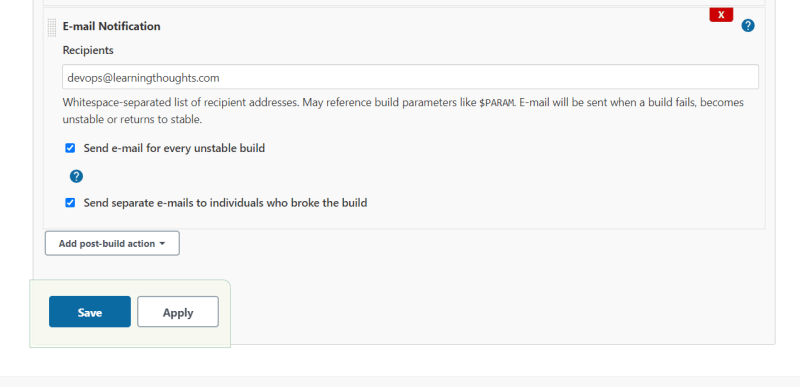
JULY 25, 2021

# DevOps Classroom Series – 25/Jul/2021

## Jenkins Continued

* 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 dowloaded for the jenkins ui directly   
* Publishing the Junit test results:    

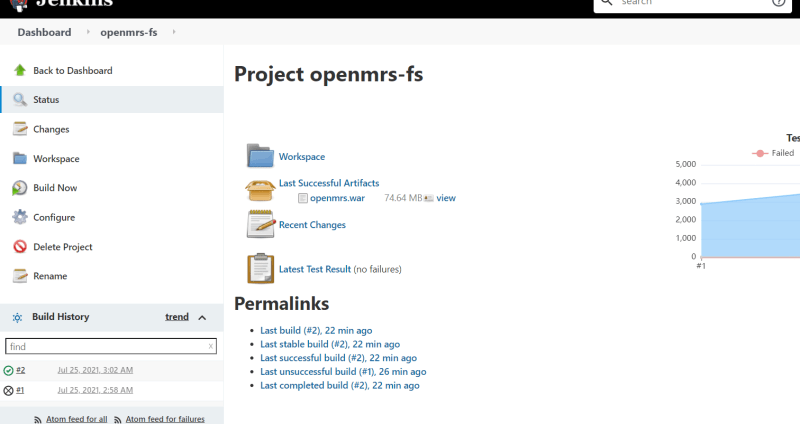
## Configuring Email Notifications in Jenkins

* For the lab setup we will be using mail trap which is simulated smtp server [Refer Here](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).
* 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
  + Multi branch 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'

}

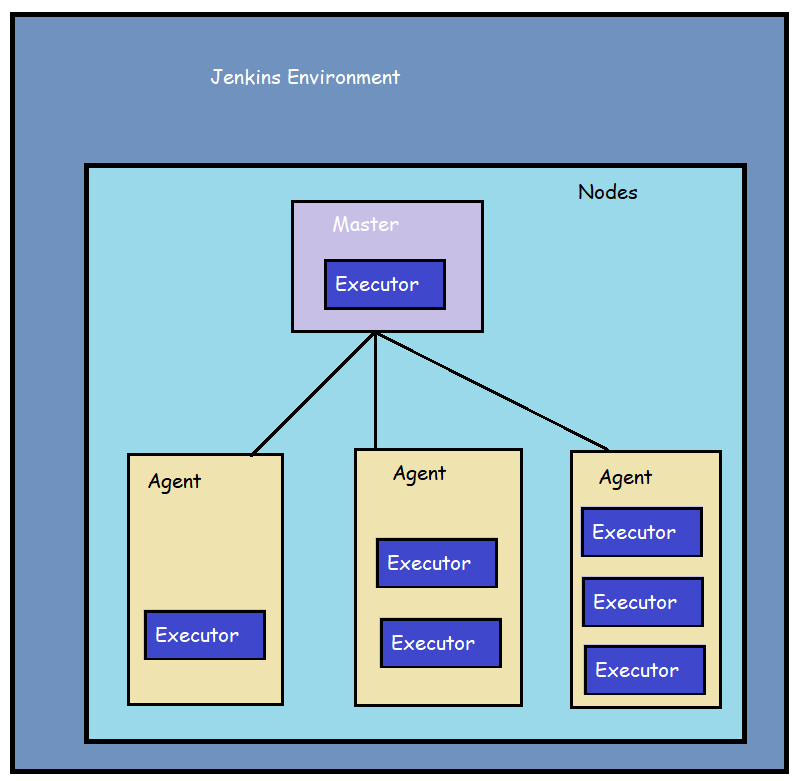
}

}

}

* 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
* Dis adavantages 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 

### Share this:

JULY 26, 2021

DevOps Classroom Series – 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

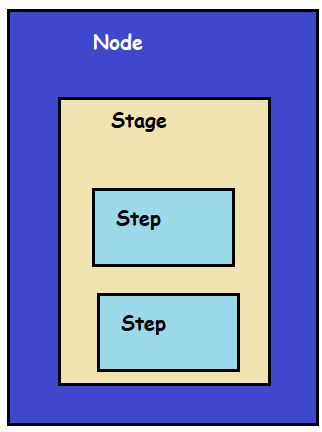
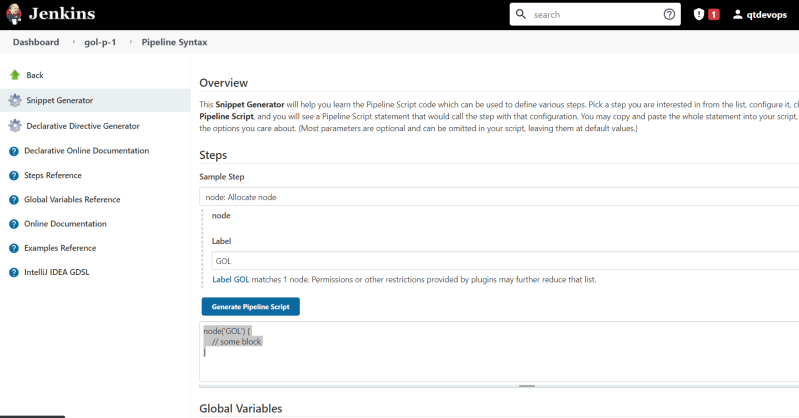
node ('GOL') {

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

JULY 27, 2021

# DevOps Classroom Series – 27/Jul/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/)
* [Refer Here](https://github.com/asquarezone/game-of-life/commit/4bc43b410f6fd0b8418cdd071033805cda55583c) for the basic declarative pipeline to build game of life
* 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

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'

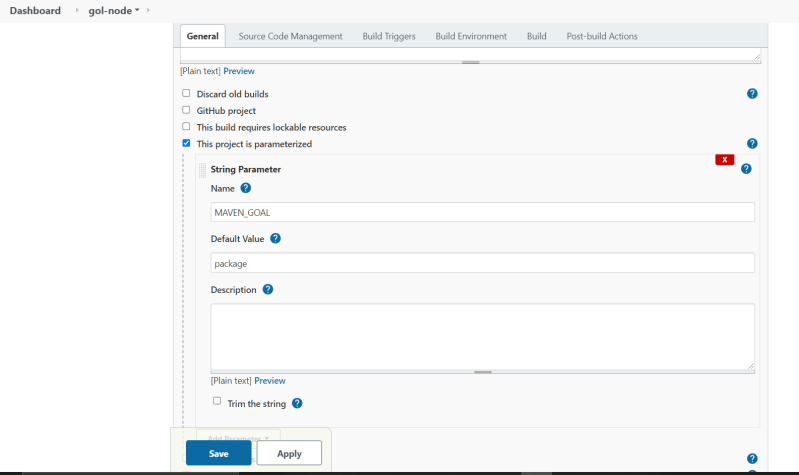
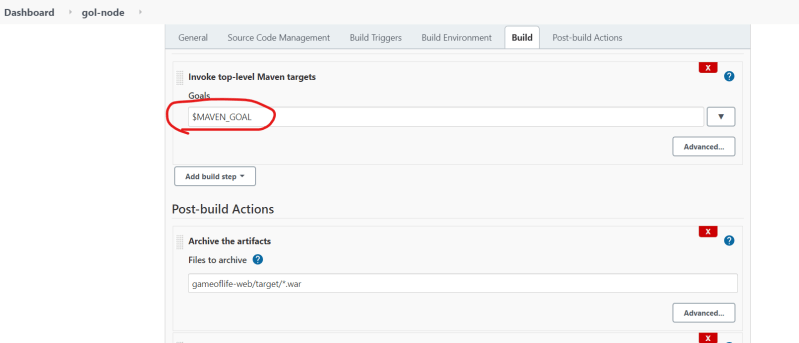
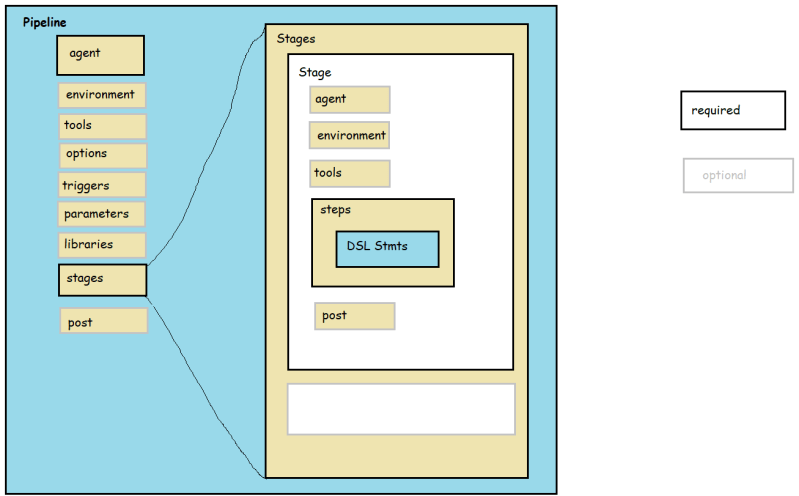
}

}

}

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

## Parameters in Jenkins

* A Parameter is use 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)
* [Refer Here](https://github.com/asquarezone/game-of-life/commit/0ae2ffa2b1d591ba80edc4aec0cccfbe65b9ed71) for the parameters added in jenkins pipeline.
* Declarative Pipeline structure 

# DevOps Classs room Series – 28/Jul/2021

## Jenkins Pipeline continued

* The following variables are available to shell scripts
  + BRANCH\_NAME: For a multi branch project, this will be set to the name of the branch being built, for example in case you wish to deploy to production from master but not from feature branches; if corresponding to some kind of change request, the name is generally arbitrary (refer to CHANGE\_ID and CHANGE\_TARGET).
  + BRANCH\_IS\_PRIMARY: For a multibranch project, if the SCM source reports that the branch being built is a primary branch, this will be set to "true"; else unset. Some SCM sources may report more than one branch as a primary branch while others may not supply this information.
  + CHANGE\_ID: For a multibranch project corresponding to some kind of change request, this will be set to the change ID, such as a pull request number, if supported; else unset.
  + CHANGE\_URL: For a multibranch project corresponding to some kind of change request, this will be set to the change URL, if supported; else unset.
  + CHANGE\_TITLE: For a multibranch project corresponding to some kind of change request, this will be set to the title of the change, if supported; else unset.
  + CHANGE\_AUTHOR: For a multibranch project corresponding to some kind of change request, this will be set to the username of the author of the proposed change, if supported; else unset.
  + CHANGE\_AUTHOR\_DISPLAY\_NAME: For a multibranch project corresponding to some kind of change request, this will be set to the human name of the author, if supported; else unset.
  + CHANGE\_AUTHOR\_EMAIL: For a multibranch project corresponding to some kind of change request, this will be set to the email address of the author, if supported; else unset.
  + CHANGE\_TARGET: For a multibranch project corresponding to some kind of change request, this will be set to the target or base branch to which the change could be merged, if supported; else unset.
  + CHANGE\_BRANCH: For a multibranch project corresponding to some kind of change request, this will be set to the name of the actual head on the source control system which may or may not be different from BRANCH\_NAME. For example in GitHub or Bitbucket this would have the name of the origin branch whereas BRANCH\_NAME would be something like PR-24.
  + CHANGE\_FORK: For a multibranch project corresponding to some kind of change request, this will be set to the name of the forked repo if the change originates from one; else unset.
  + TAG\_NAME: For a multibranch project corresponding to some kind of tag, this will be set to the name of the tag being built, if supported; else unset.
  + TAG\_TIMESTAMP: For a multibranch project corresponding to some kind of tag, this will be set to a timestamp of the tag in milliseconds since Unix epoch, if supported; else unset.
  + TAG\_UNIXTIME: For a multibranch project corresponding to some kind of tag, this will be set to a timestamp of the tag in seconds since Unix epoch, if supported; else unset.
  + TAG\_DATE: For a multibranch project corresponding to some kind of tag, this will be set to a timestamp in the format as defined by java.util.Date#toString() (e.g., Wed Jan 1 00:00:00 UTC 2020), if supported; else unset.
  + JOB\_DISPLAY\_URL: URL that will redirect to a Job in a preferred user interface
  + RUN\_DISPLAY\_URL: URL that will redirect to a Build in a preferred user interface
  + RUN\_ARTIFACTS\_DISPLAY\_URL: URL that will redirect to Artifacts of a Build in a preferred user interface
  + RUN\_CHANGES\_DISPLAY\_URL: URL that will redirect to Changelog of a Build in a preferred user interface
  + RUN\_TESTS\_DISPLAY\_URL: URL that will redirect to Test Results of a Build in a preferred user interface
  + CI: Statically set to the string "true" to indicate a "continuous integration" execution environment.
  + BUILD\_NUMBER: The current build number, such as "153".
  + BUILD\_ID: The current build ID, identical to BUILD\_NUMBER for builds created in 1.597+, but a YYYY-MM-DD\_hh-mm-ss timestamp for older builds.
  + BUILD\_DISPLAY\_NAME: The display name of the current build, which is something like "#153" by default.
  + JOB\_NAME: Name of the project of this build, such as "foo" or "foo/bar".
  + JOB\_BASE\_NAME: Short Name of the project of this build stripping off folder paths, such as "foo" for "bar/foo".
  + BUILD\_TAG: String of "jenkins-${JOB\_NAME}-${BUILD\_NUMBER}". All forward slashes ("/") in the JOB\_NAME are replaced with dashes ("-"). Convenient to put into a resource file, a jar file, etc for easier identification.
  + EXECUTOR\_NUMBER: The unique number that identifies the current executor (among executors of the same machine) that’s carrying out this build. This is the number you see in the "build executor status", except that the number starts from 0, not 1.
  + NODE\_NAME: Name of the agent if the build is on an agent, or "master" if run on master.
  + NODE\_LABELS: Whitespace-separated list of labels that the node is assigned.
  + WORKSPACE: The absolute path of the directory assigned to the build as a workspace.
  + WORKSPACE\_TMP: A temporary directory near the workspace that will not be browsable and will not interfere with SCM checkouts. May not initially exist, so be sure to create the directory as needed (e.g., mkdir -p on Linux). Not defined when the regular workspace is a drive root.
  + JENKINS\_HOME: The absolute path of the directory assigned on the master node for Jenkins to store data.
  + JENKINS\_URL: Full URL of Jenkins, like <http://server:port/jenkins/> (note: only available if Jenkins URL set in system configuration).
  + BUILD\_URL: Full URL of this build, like <http://server:port/jenkins/job/foo/15/> (Jenkins URL must be set).
  + JOB\_URL: Full URL of this job, like <http://server:port/jenkins/job/foo/> (Jenkins URL must be set).
  + GIT\_COMMIT: The commit hash being checked out.
  + GIT\_PREVIOUS\_COMMIT: The hash of the commit last built on this branch, if any.
  + GIT\_PREVIOUS\_SUCCESSFUL\_COMMIT: The hash of the commit last successfully built on this branch, if any.
  + GIT\_BRANCH: The remote branch name, if any.
  + GIT\_LOCAL\_BRANCH: The local branch name being checked out, if applicable.
  + GIT\_CHECKOUT\_DIR: The directory that the repository will be checked out to. This contains the value set in Checkout to a sub-directory, if used.
  + GIT\_URL: The remote URL. If there are multiple, will be GIT\_URL\_1, GIT\_URL\_2, etc.
  + GIT\_COMMITTER\_NAME: The configured Git committer name, if any, that will be used for FUTURE commits from the current workspace. It is read from the Global Config user.name Value field of the Jenkins Configure System page.
  + GIT\_AUTHOR\_NAME: The configured Git author name, if any, that will be used for FUTURE commits from the current workspace. It is read from the Global Config user.name Value field of the Jenkins Configure System page.
  + GIT\_COMMITTER\_EMAIL: The configured Git committer email, if any, that will be used for FUTURE commits from the current workspace. It is read from the Global Config user.email Value field of the Jenkins Configure System page.
  + GIT\_AUTHOR\_EMAIL: The configured Git author email, if any, that will be used for FUTURE commits from the current workspace. It is read from the Global Config user.email Value field of the Jenkins Configure System page.
  + SVN\_REVISION: Subversion revision number that’s currently checked out to the workspace, such as "12345"
  + SVN\_URL: Subversion URL that’s currently checked out to the workspace.
* Refer the below jenkinsfile to see the creation and usage of existing environment variables

pipeline {

agent { label 'GOL'}

triggers {

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

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

}

parameters {

string(name: 'BRANCH', defaultValue: 'master', description: 'Branch to build' )

}

environment {

CI\_ENV = 'DEV'

}

stages {

stage('scm') {

environment {

DUMMY = 'FUN'

}

steps {

git branch: "${params.BRANCH}", url: 'https://github.com/asquarezone/game-of-life.git'

//input message: 'Continue to next stage? ', submitter: 'qtaws,qtazure'

echo env.CI\_ENV

echo env.DUMMY

}

}

stage('build') {

steps {

echo env.GIT\_URL

sh 'mvn package'

}

}

}

post {

success {

archive '\*\*/gameoflife.war'

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

}

}

}

## Flow Control Options

* timeout: This step allows you to limit the amount of your script spends waiting for action to happen [Refer Here](https://www.jenkins.io/doc/pipeline/steps/workflow-basic-steps/#timeout-enforce-time-limit)
* sleep: This is basic delay step [Refer Here](https://www.jenkins.io/doc/pipeline/steps/workflow-basic-steps/#sleep-sleep)

sleep time: 7, unit: 'MINUTES'

* retry: This closure retires the process n times if an exception [Refer Here](https://www.jenkins.io/doc/pipeline/steps/workflow-basic-steps/#retry-retry-the-body-up-to-n-times)

retry(3){

sh 'mvn clean package'

}

* For notification on build failures we can use the mail step [Refer Here](https://www.jenkins.io/doc/pipeline/steps/workflow-basic-steps/#mail-mail)
* Refer the below Jenkins file with notifications and other additions as discussed in the class

pipeline {

agent { label 'GOL'}

triggers {

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

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

}

parameters {

string(name: 'BRANCH', defaultValue: 'master', description: 'Branch to build' )

choice(name: 'GOAL', choices: ['package', 'clean package', 'install'], description: 'maven goals')

}

options {

timeout(time: 1, unit: 'HOURS')

retry(2)

}

environment {

CI\_ENV = 'DEV'

}

stages {

stage('scm') {

environment {

DUMMY = 'FUN'

}

steps {

mail subject: 'BUILD Started '+env.BUILD\_ID, to: 'devops@qt.com', from: 'jenkins@qt.com', body: 'EMPTY BODY'

git branch: "${params.BRANCH}", url: 'https://github.com/asquarezone/game-of-life.git'

//input message: 'Continue to next stage? ', submitter: 'qtaws,qtazure'

echo env.CI\_ENV

echo env.DUMMY

}

}

stage('build') {

steps {

echo env.GIT\_URL

timeout(time:10, unit: 'MINUTES') {

sh "mvn ${params.GOAL}"

}

}

}

}

post {

success {

archive '\*\*/gameoflife.war'

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

mail subject: 'BUILD Completed Successfully '+env.BUILD\_ID, to: 'devops@qt.com', from: 'jenkins@qt.com', body: 'EMPTY BODY'

}

failure {

mail subject: 'BUILD Failed '+env.BUILD\_ID+'URL is '+env.BUILD\_URL, to: 'devops@qt.com', from: 'jenkins@qt.com', body: 'EMPTY BODY'

}

always {

echo "Finished"

}

changed {

echo "Changed"

}

unstable {

mail subject: 'BUILD Unstable '+env.BUILD\_ID+'URL is '+env.BUILD\_URL, to: 'devops@qt.com', from: 'jenkins@qt.com', body: 'EMPTY BODY'

}

}

}

* Next Steps:
  + Jenkins
    - Understanding Stash in Jenkins
    - Measuring Code Quality
    - Storing Artifacts into Jfrog Artifactory
    - Creating a ci/cd pipeline for gameoflife/openmrs
  + Git:
    - Stash
    - Hooks
    - Tags
    - Submodules
    - Config & aliases
    - GitHUB:
      * WebHooks
      * Pull Requests
    - Azure Source Repos
    - AWS Code Commit
  + Branching Strategy
  + Build for .net project using msbuild
  + CI/CD using Azure DevOps (VSTS)

### Share this:

# DevOps Classroom Series – 29/Jul/2021

## Stash and unstash in Jenkins Pipeline

* Stash can be used to copy any stuff from stage to stage (even on different node)
* As an example, lets build a game of life and then stash gameoflife.war file from node 1 and unstash to the node 2 where the file will be made available
* For sample usage of stash and unstash refer below

pipeline {

agent { label 'GOL'}

triggers {

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

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

}

parameters {

string(name: 'BRANCH', defaultValue: 'master', description: 'Branch to build' )

choice(name: 'GOAL', choices: ['package', 'clean package', 'install'], description: 'maven goals')

}

options {

timeout(time: 1, unit: 'HOURS')

retry(2)

}

environment {

CI\_ENV = 'DEV'

}

stages {

stage('scm') {

environment {

DUMMY = 'FUN'

}

steps {

mail subject: 'BUILD Started '+env.BUILD\_ID, to: 'devops@qt.com', from: 'jenkins@qt.com', body: 'EMPTY BODY'

git branch: "${params.BRANCH}", url: 'https://github.com/asquarezone/game-of-life.git'

//input message: 'Continue to next stage? ', submitter: 'qtaws,qtazure'

echo env.CI\_ENV

echo env.DUMMY

}

}

stage('build') {

steps {

echo env.GIT\_URL

timeout(time:10, unit: 'MINUTES') {

sh "mvn ${params.GOAL}"

}

stash includes: '\*\*/gameoflife.war', name: 'golwar'

}

}

stage('devserver'){

agent { label 'RHEL,'}

steps {

unstash name: 'golwar'

}

}

}

post {

success {

archive '\*\*/gameoflife.war'

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

mail subject: 'BUILD Completed Successfully '+env.BUILD\_ID, to: 'devops@qt.com', from: 'jenkins@qt.com', body: 'EMPTY BODY'

}

failure {

mail subject: 'BUILD Failed '+env.BUILD\_ID+'URL is '+env.BUILD\_URL, to: 'devops@qt.com', from: 'jenkins@qt.com', body: 'EMPTY BODY'

}

always {

echo "Finished"

}

changed {

echo "Changed"

}

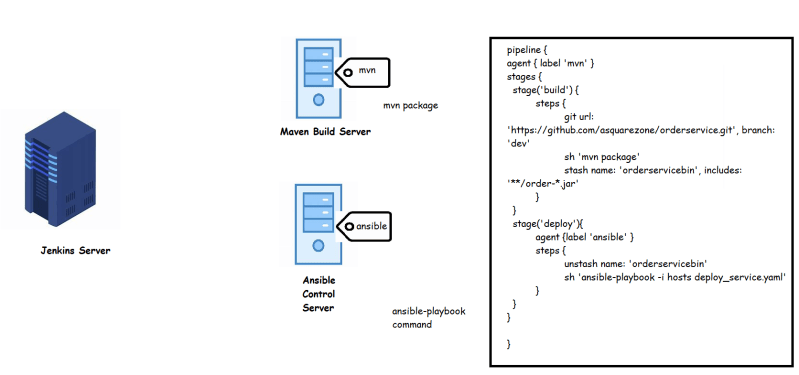
unstable {

mail subject: 'BUILD Unstable '+env.BUILD\_ID+'URL is '+env.BUILD\_URL, to: 'devops@qt.com', from: 'jenkins@qt.com', body: 'EMPTY BODY'

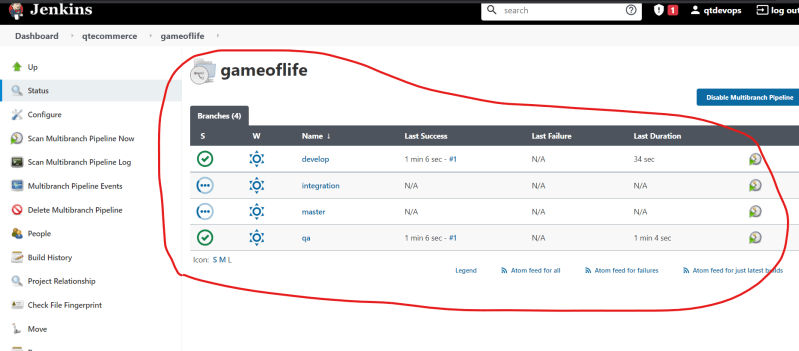
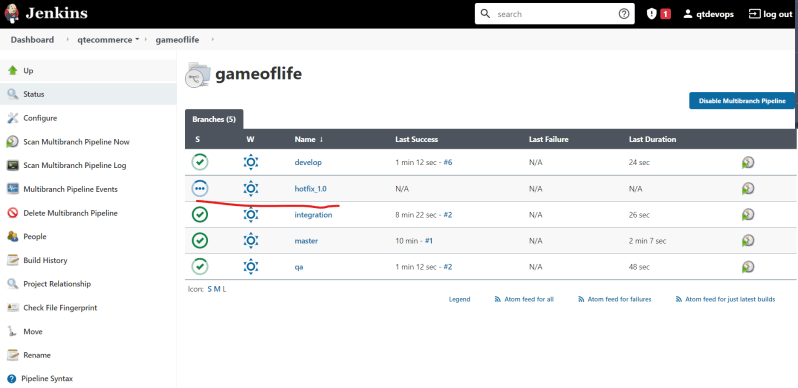
}

}

}

* Scenarios:
  + Create a CI/CD pipeline where we need to build the application and deploy the application using Ansible 

## Jenkins Project Types

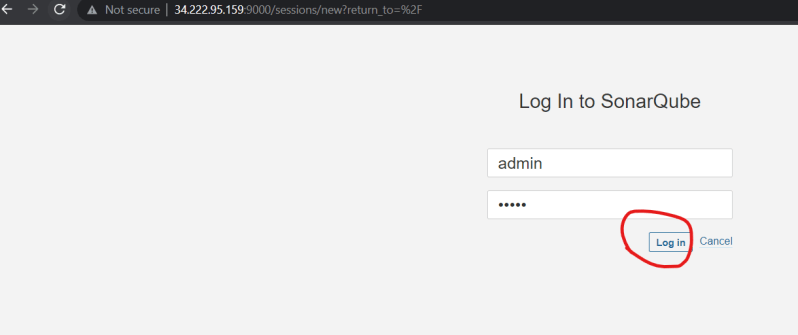
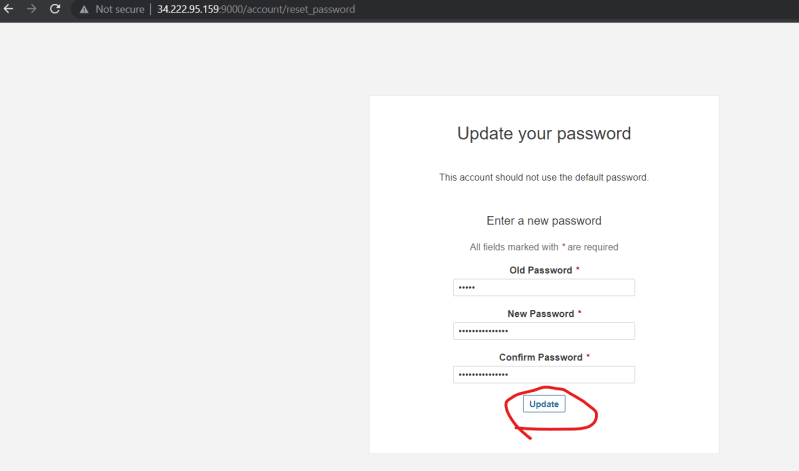
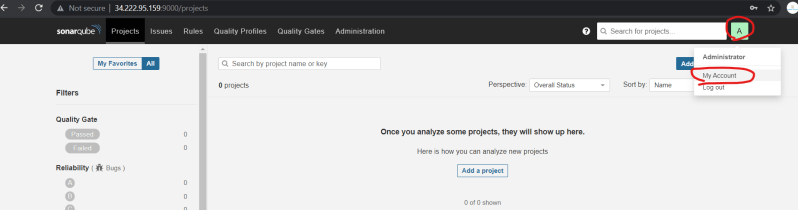
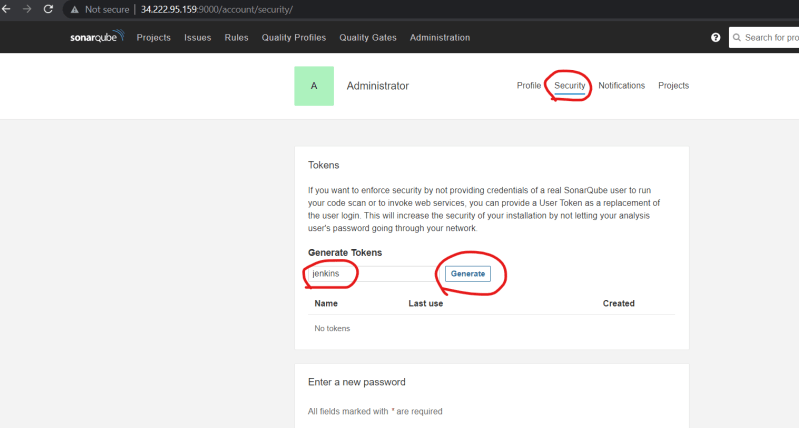
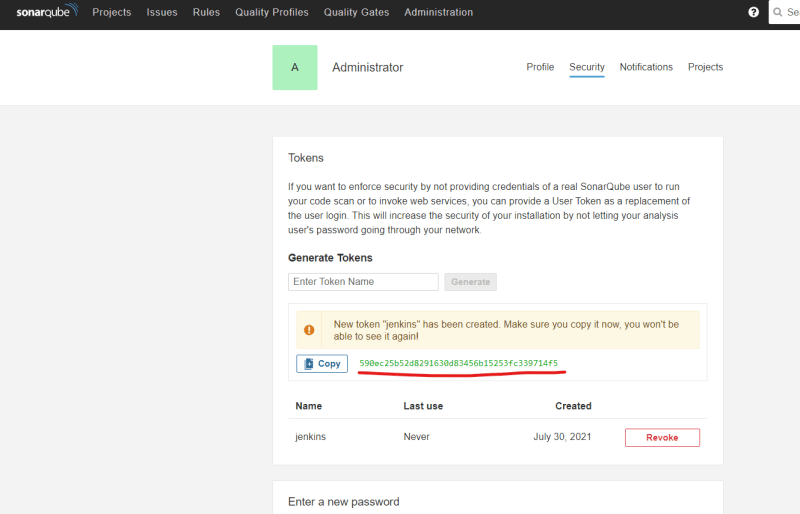
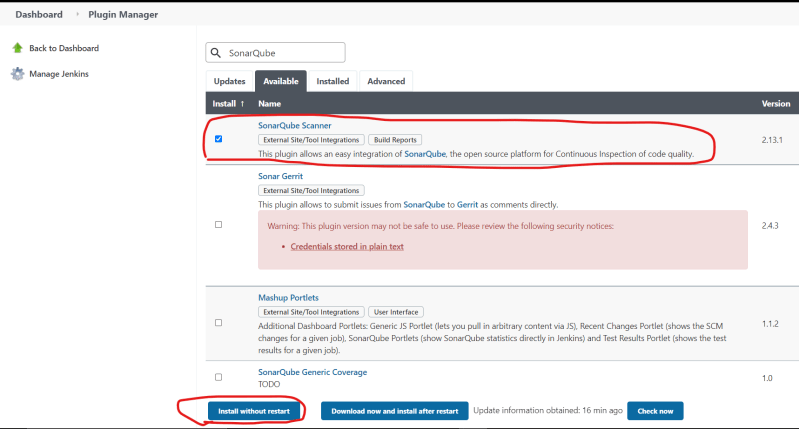
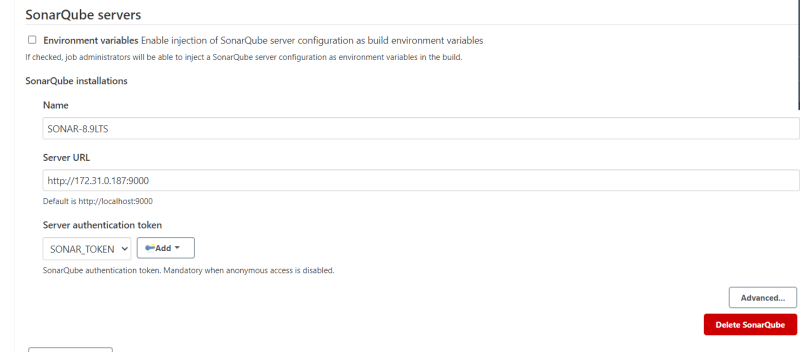
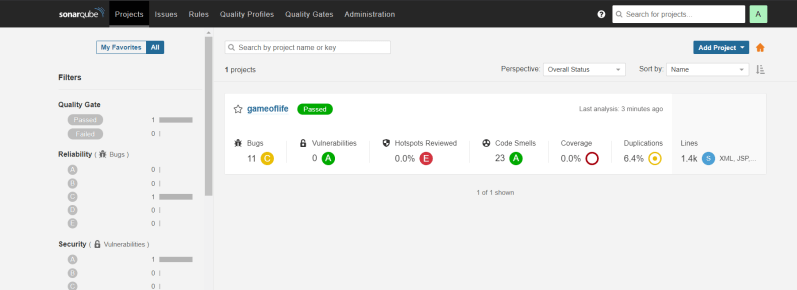
* Freestyle
* Pipeline Project Type
* Multi configuration project type
* Folder
* Multi branch pipeline
* Exercise:
  + Fork the game o file project from [Refer Here](https://github.com/asquarezone/game-of-life.git)
  + Now clone the project into your local system
  + Create the following branches in your git repository
    - develop
    - qa
    - integration
  + Now we will have total of 4 branches
  + Let’s push all the branches to git repo
  + Now lets create a multi branch project 
  + If we create a new branch then we need to scan multibranch pipeline manually to get the new branch added to build 
  + For every branch we can have different configuration like build triggers, agents etc written in Jenkinsfile.

## Code Quality Scanning

* Sonar Qube is an open source platform for managing code quality in several software areas like
  + Architecture and Design
  + Comments
  + Coding rules
  + Potential Bugs
  + Duplications
  + Unit tests
  + Complexity
* Note:
  + Installation of Sonar Qube will be shared as video
  + Installation of Jfrog artifactory will be shared as video

# DevOps Classroom Notes 30/Jul/2021

## Sonarqube Configuration in Jenkins

* [Refer Here](https://directdevops.blog/2021/07/30/installing-sonarqube-8-9-lts-on-ubuntu/) for sonarqube installation
* [Refer Here](https://directdevops.blog/2019/01/05/sonarqube/) for sonarqube configuration
* After install sonarqube Navigate to sonarqube using http://<public-ip&gt;:9000 using credentials username: admin password: admin  
* Generate a token in sonarqube for jenkins to connect to sonarqube   
* Now install Sonarqube scanner plugin for jenkins 
* Now Navigate Manage Jenkins => Configure System => Sonarqube Servers 
* [Refer Here](https://github.com/asquarezone/game-of-life/commit/a7b1931ff792faff8dd7af46abab468f8e3fa70f) for the changes in Jenkinsfile 
* [Refer Here](https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-jenkins/) to integrate jenkins with SonarQube
* [Refer Here](https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-azure-devops/) to integrate azure devops with SonarQube

## Artifactory

* Install Artifactory using the following commands

# To determine your distribution, run lsb\_release -c or cat /etc/os-release

# Example:echo "deb https://releases.jfrog.io/artifactory/artifactory-pro-debs xenial main" | sudo tee -a /etc/apt/sources.list;

wget -qO - https://releases.jfrog.io/artifactory/api/gpg/key/public | sudo apt-key add -;

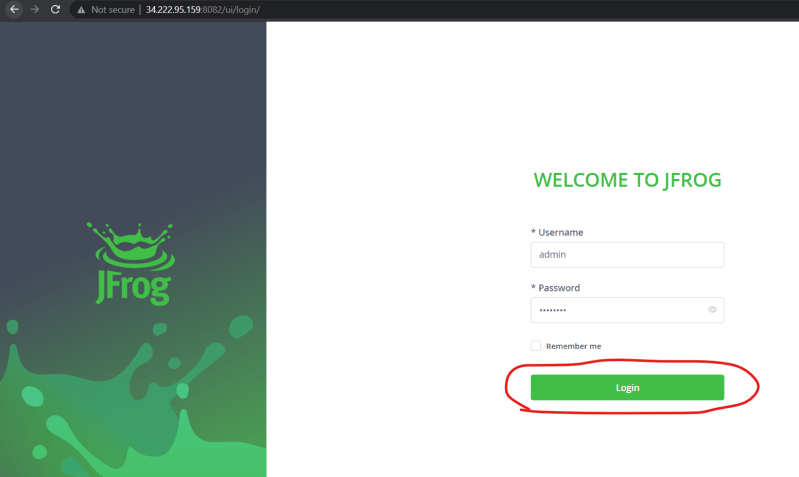
echo "deb https://releases.jfrog.io/artifactory/artifactory-debs {distribution} main" | sudo tee -a /etc/apt/sources.list;

sudo apt-get update && sudo apt-get install jfrog-artifactory-oss

sudo systemctl start artifactory.service

sudo systemctl enable artifactory.service

sudo systemctl status artifactory.service

* [Refer Here](https://jfrog.com/open-source/) for the download of artifactory opensource
* Now navigate to artifactory using url http://<public-ip&gt;:8081 
* Configuring artifactory from jenkins [Refer Here](https://directdevops.blog/2019/10/17/artifactory-configuration/) for the documentation
* Once the artifacts are uploaded we can download the artifacts from any connected node by using simple

curl -u "admin:MOTHERindia@123" -O http://34.222.95.159:8081/artifactory/lt-local/com/wakaleo/gameoflife/gameoflife-web/1.0-SNAPSHOT/gameoflife-web-1.0-20210730.032620-1.war

* In some cases your organization might set the API Key

curl -H 'X-JFROG-Art-API:<API\_KEY>' -O 'http(s)://<ARTIFACTORY\_URL>/<REPO>/<PATH>'

* Exercise:
  + Create a Declarative pipeline to push artifacts to artifactory
  + How to download the latest version of a jar/war file using curl command from artificatory

# Installing SonarQube 8.9 LTS on Ubuntu

## Installing SonarQube in ubuntu

* Perform a system update and install unzip

sudo apt update

sudo apt install unzip -y

* Install Openjdk11

sudo apt install openjdk-11-jdk -y

* Install and Configure Postgres

sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb\_release -cs`-pgdg main" >> /etc/apt/sources.list.d/pgdg.list'

wget -q https://www.postgresql.org/media/keys/ACCC4CF8.asc -O - | sudo apt-key add -

sudo apt-get -y install postgresql postgresql-contrib

* Enable and Start Postgresql

sudo systemctl enable postgresql

sudo systemctl start postgresql

* Change the passwd for postgres user

sudo passwd postgres

* Switch to postgres user and create a user called sonar

su - postgres

createuser sonar

psql

* Set a password for the newly created user for SonarQube database and create a database for Postgresql database

ALTER USER sonar WITH ENCRYPTED password 'P@ssword';

CREATE DATABASE sonar OWNER sonar;

* Exit the psql shell and switch back to the user by running exit comand

\q

exit

* Download Sonarqube

wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-8.9.1.44547.zip

* Unzip the sonarqube using following command

sudo unzip sonarqube-8.9.1.44547.zip -d /opt

* Rename the directory

sudo mv /opt/sonarqube-8.9.1.44547 /opt/sonarqube

* Create a non sudo linux user

sudo adduser sonarq

* Assign permissions to sonarqube directory

sudo chown -R sonarq:sonarq /opt/sonarqube/

* Sonarqube uses the elastic search service so increase vm max map

sudo sysctl -w vm.max\_map\_count=262144

* Open the Sonarqube properties file sudo nano /opt/sonarqube/conf/sonar.properties and change the following properties

sonar.jdbc.username=sonar

sonar.jdbc.password=P@ssword

sonar.jdbc.url=jdbc:postgresql://localhost/sonar

sonar.web.javaAdditionalOpts=-server

* Configure Sonarqube as service

sudo nano /etc/systemd/system/sonar.service

* Add the following content to sonar.service

[Unit]

Description=SonarQube service

After=syslog.target network.target

[Service]

Type=forking

ExecStart=/opt/sonarqube/bin/linux-x86-64/sonar.sh start

ExecStop=/opt/sonarqube/bin/linux-x86-64/sonar.sh stop

User=sonarq

Group=sonarq

Restart=always

[Install]

WantedBy=multi-user.target

* Now enable and start sonarqube

sudo systemctl enable sonar

sudo systemctl start sonar

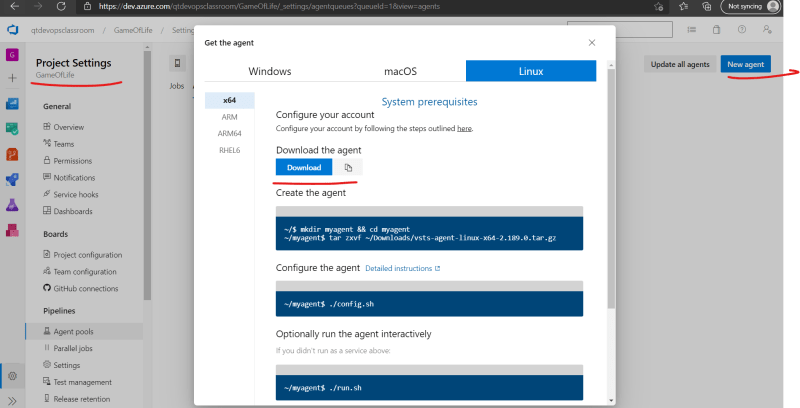
sudo systemctl status sonar

* Now access the sonarqube with the ip address of the server http://<ipaddress&gt;:9000. Login into sonarqube with default credentails username: admin and password: admin

AUGUST 1, 2021

# DevOps Classroom Series – 31/Jul/2021

## Adding Agents to Azure DevOps

* For configuring self host agents go through the classroom video
* Create a linux agent (ubuntu 20)
  + Install java 8
  + install maven 
* With the azure-pipelines.yaml as shown below and with the self hosted agent

---

trigger:

- master

variables:

MVN\_GOAL: 'package'

pool:

name: Default

steps:

- task: Maven@3

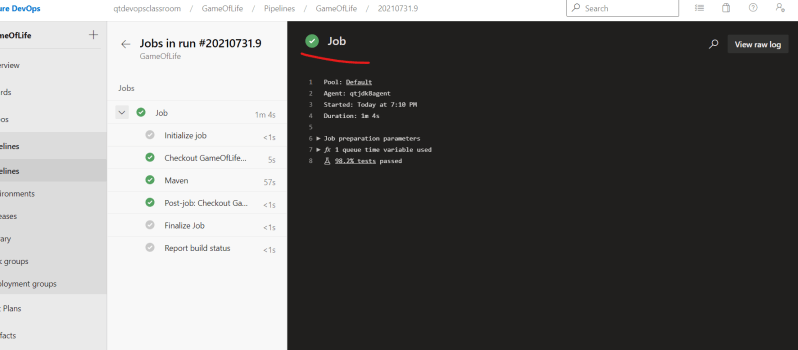
inputs:

mavenPomFile: 'pom.xml'

goals: $(MVN\_GOAL)

jdkVersionOption: 1.8

testResultsFiles: '\*\*/surefire-reports/TEST-\*.xml'



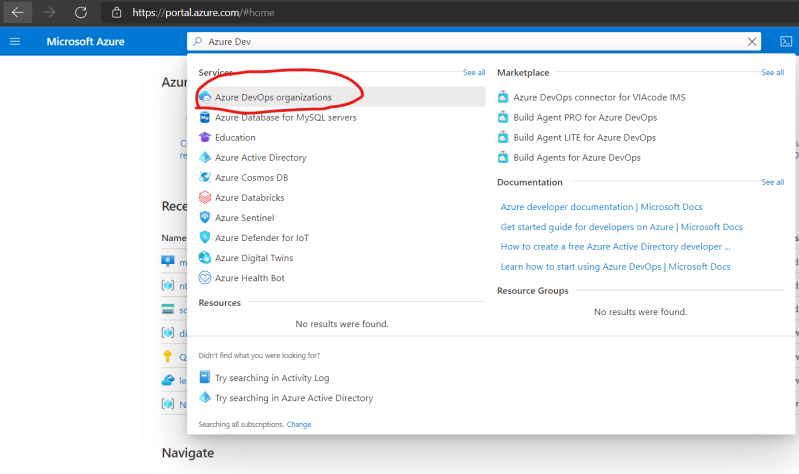
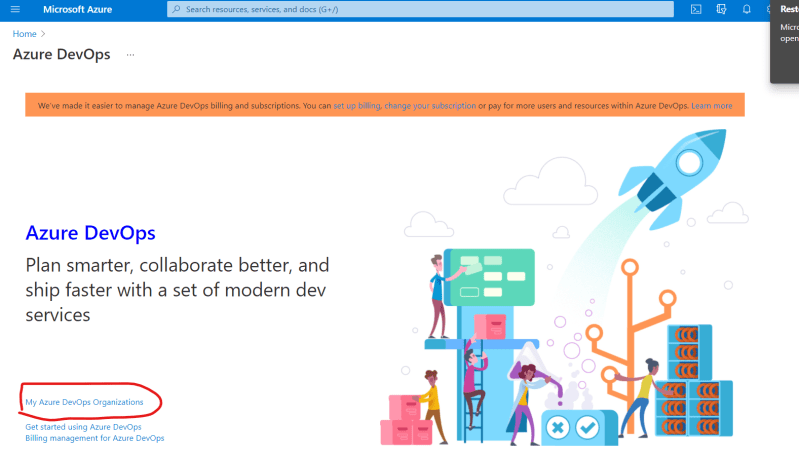
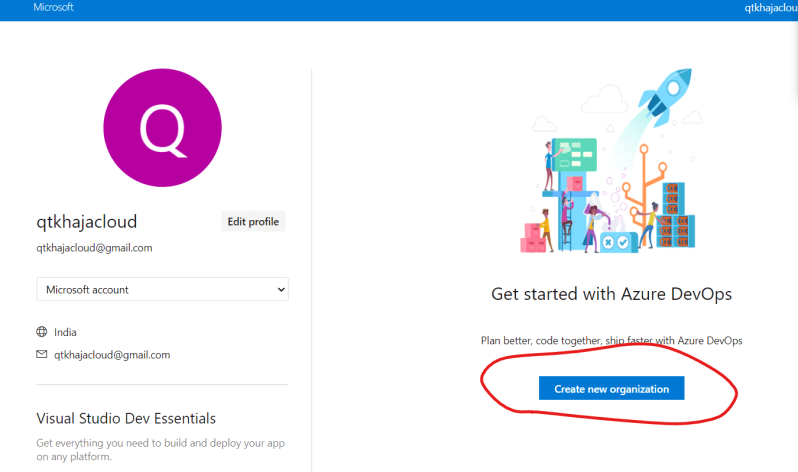
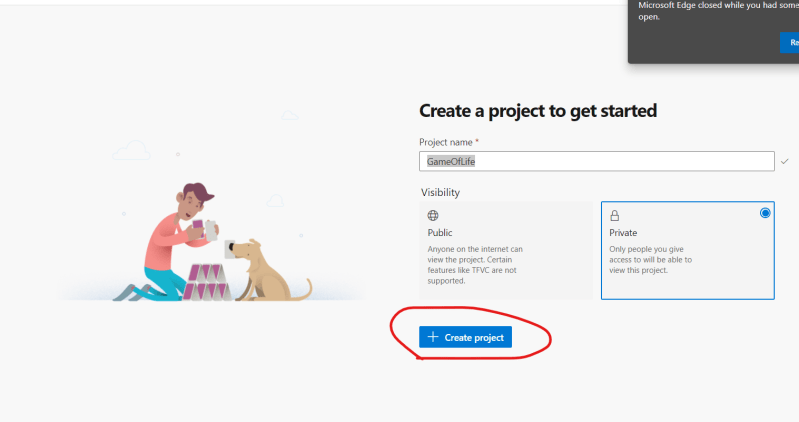
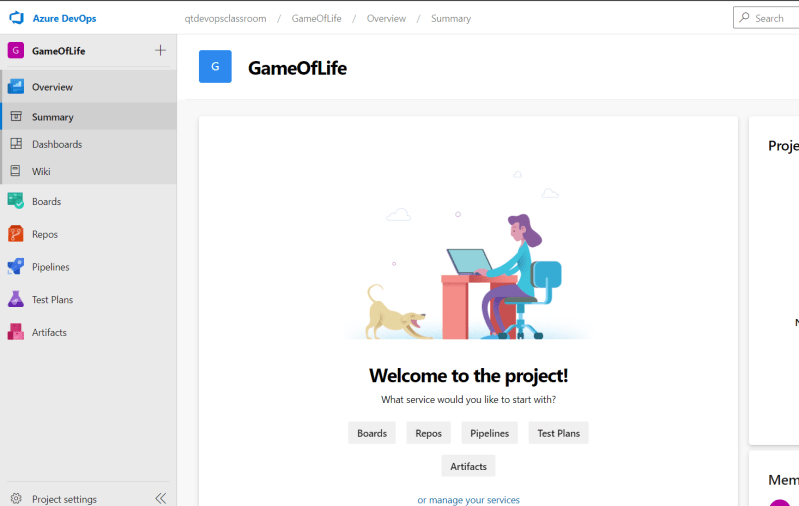
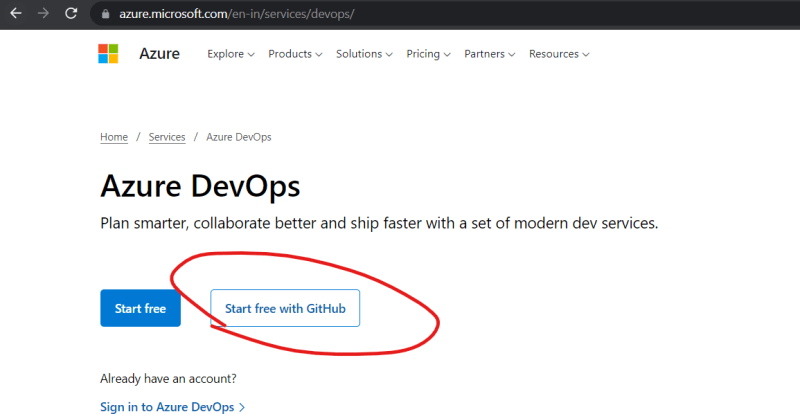
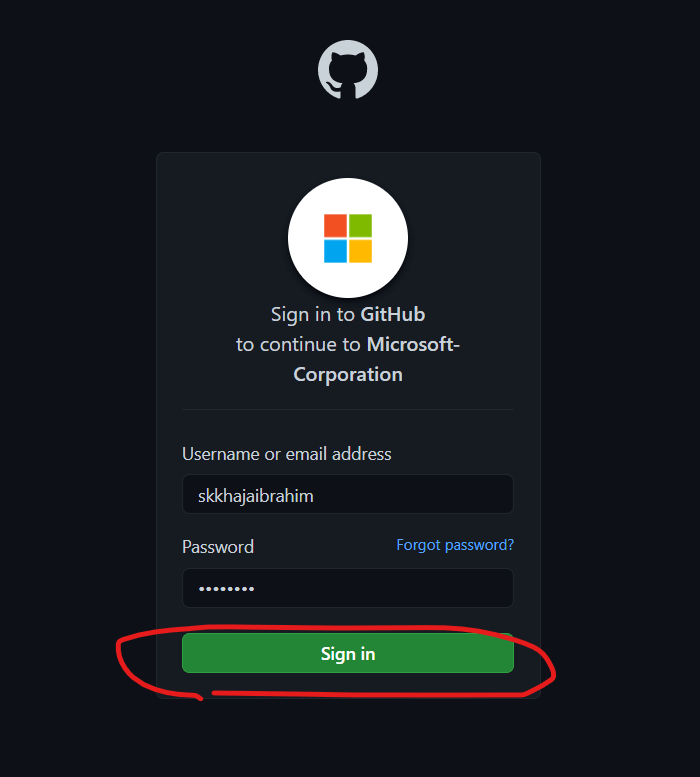
* For working with different technologies in Azure DevOps [Refer Here](https://docs.microsoft.com/en-us/azure/devops/pipelines/ecosystems/ecosystems?view=azure-devops)
* For building .net application on Windows
  + Install Visual Studio Build Tools
  + Restore the nuget package C:\hostedtoolcache\windows\NuGet\5.10.0\x64\nuget.exe restore D:\a\1\s\WebApplication1\WebApplication1.sln
  + Run the msbuild command C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise\MSBuild\Current\Bin\msbuild.exe" "D:\a\1\s\WebApplication1\WebApplication1.sln"
  + Run the testcases C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise\Common7\IDE\Extensions\TestPlatform\vstest.console.exe "@D:\a\\_temp\nml1iy2lu5z.tmp"
* Exercise:
  + Ansible:
    - Write an ansible playbook to deploy game-of-life on tomcat
  + Terraform:
    - Write a terraform template to create a linux instance (in any cloud) and deploy game of life application
  + Docker and K8s:
    - Write a Dockerfile for game-of-life
    - Create a K8s Manifest for game-of-life deployment

**Azure DevOps**

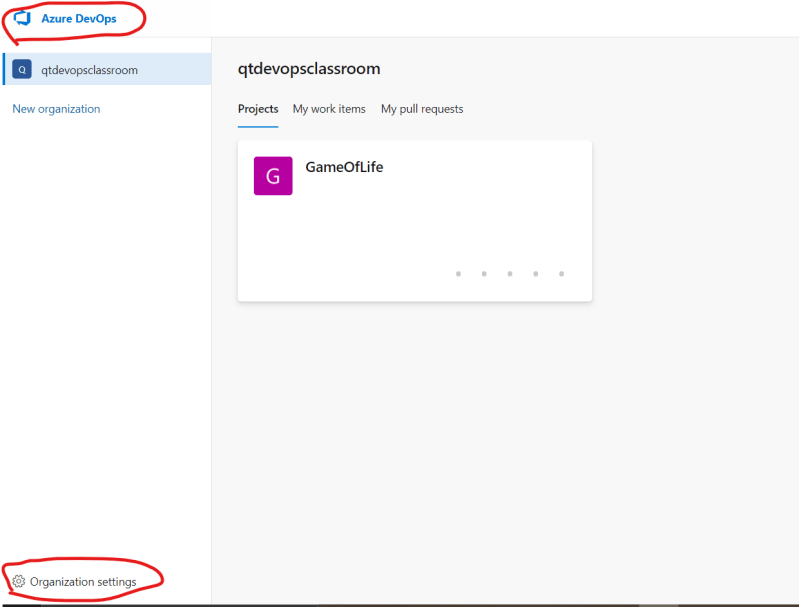
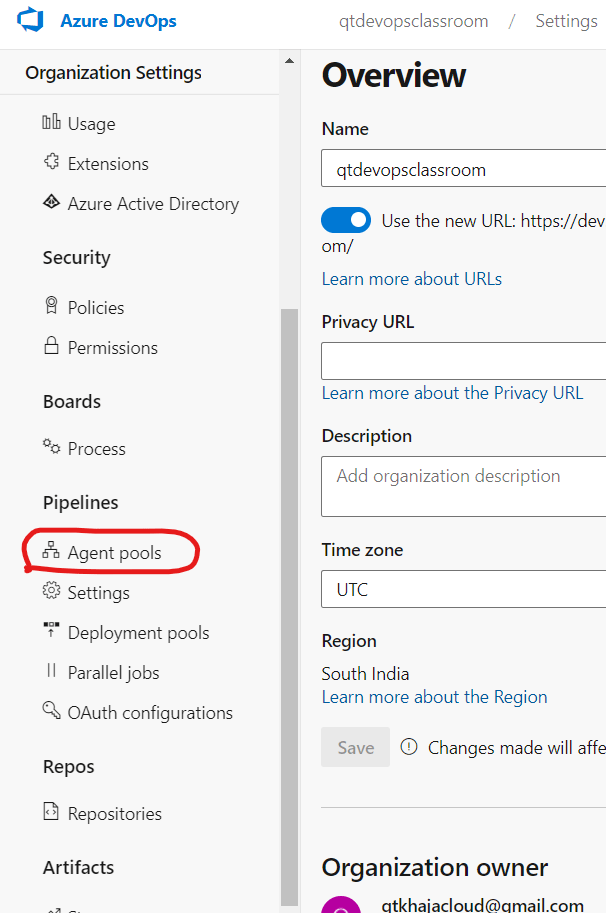
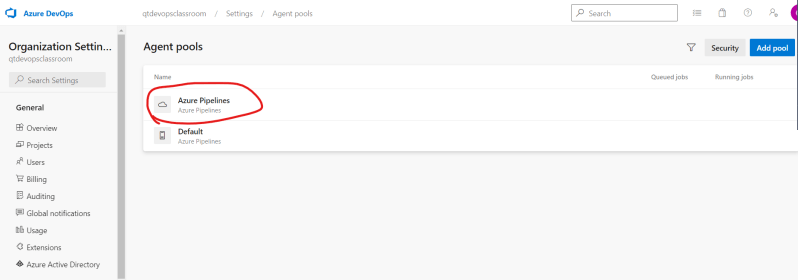
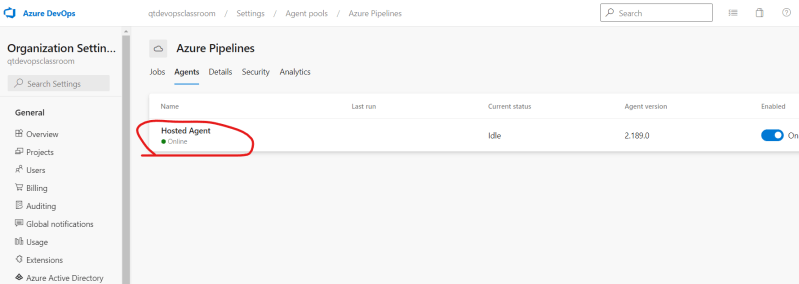
ULY 31, 2021

# DevOps Classroom Series – 31/Jul/2021

## Azure DevOps

* Accessing Azure DevOps
  + For existing Azure Users     
  + If you are not an Azure user we can use github account Naviagate to [Refer Here](https://azure.microsoft.com/en-in/services/devops/)  
* Azure DevOps offers two ways of using Azure Devops
  + Hosted Azure DevOps: Azure DevOps will be hosted and you need to create an account and manage users
    - pricing: [Refer Here](https://azure.microsoft.com/en-in/pricing/details/devops/azure-devops-services/)
  + Installing Azure DevOps on your on-premises server: [Refer Here](https://docs.microsoft.com/en-us/azure/devops/server/install/single-server?view=azure-devops-2020)
* Once the git repo is imported/created in Azure DevOps Source Repos, All we need to do is to create an azure-pipelines.yml file with the necessary build steps configured.
* Exercise: Create an Azure DevOps Organization and create a project called as gameoflife and push the master branch of gameoflife from github into AzureSourceRepo

## Writing Azure Pipelines in YAML

* YAML Tutorial [Refer Here](https://www.youtube.com/watch?v=ggOmHlnhPaM&list=PLuVH8Jaq3mLud3sVDvJ-gJ__0zd15wGDd&index=15)
* To write the Pipeline in YAML we need to understand the schema from Azure DevOps [Refer Here](https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=schema%2Cparameter-schema)
* Pipeline Schema [Refer Here](https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=schema%2Cparameter-schema#pipeline)
* Triggers [Refer Here](https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=schema%2Cparameter-schema#triggers)
  + For periodic builds equivalent => Scheduled Trigger [Refer Here](https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=example%2Cparameter-schema#scheduled-trigger)
  + For poll scm equivalent => Push Trigger [Refer Here](https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=example%2Cparameter-schema#push-trigger)
* Azure Pipleline job needs to run on some node. Azure DevOps uses the term agent to refer it. In Azure DevOps we have two types of agent
  + Microsoft Hosted Agents:
    - These are hosted by Azure/Microsoft
    - Free plan => 1800 minutes per month is free
  + Self Hosted Agents
    - We can add our agent to the Azure DevOps
    - Adding one agent is part of free plan
* For configuring agents    
* To configure a pool [Refer Here](https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema?view=azure-devops&tabs=example%2Cparameter-schema#pool) and use a microsoft hosted agent [Refer Here](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/hosted?view=azure-devops&tabs=yaml#use-a-microsoft-hosted-agent)
* In Azure Devops

stages

----> jobs

-----> steps

------> tasks

* For all the tasks [Refer Here](https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/?view=azure-devops)
* The Azure pipeline which is created in the class

---

trigger:

- master

variables:

MVN\_GOAL: 'package'

pool:

name: Azure Pipelines

vmImage: 'ubuntu-latest'

steps:

- task: Maven@3

inputs:

mavenPomFile: 'pom.xml'

goals: $(MVN\_GOAL)

jdkVersionOption: 1.8

testResultsFiles: '\*\*/surefire-reports/TEST-\*.xml'

* Equivalent in Jenkins

pipeline {

agent { label 'ubuntu' }

triggers {

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

}

stages {

stage('SCM') {

steps {

git 'htps:/lksdfskl'

}

}

stage('build') {

steps {

sh 'mvn package'

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

}

}

}

}

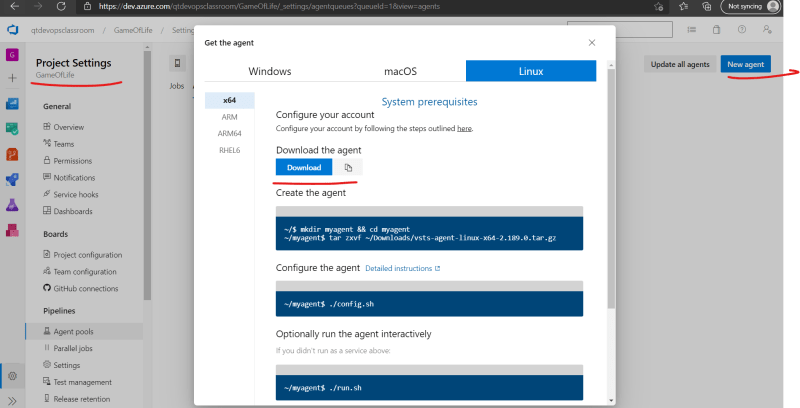
### Share this:

* [Click](https://directdevops.blog/2021/07/31/devops-classroom-series-31-jul-2021/?share=twitter&nb=1)

AUGUST 1, 2021

# DevOps Classroom Series – 31/Jul/2021

## Adding Agents to Azure DevOps

* For configuring self host agents go through the classroom video
* Create a linux agent (ubuntu 20)
  + Install java 8
  + install maven 
* With the azure-pipelines.yaml as shown below and with the self hosted agent

---

trigger:

- master

variables:

MVN\_GOAL: 'package'

pool:

name: Default

steps:

- task: Maven@3

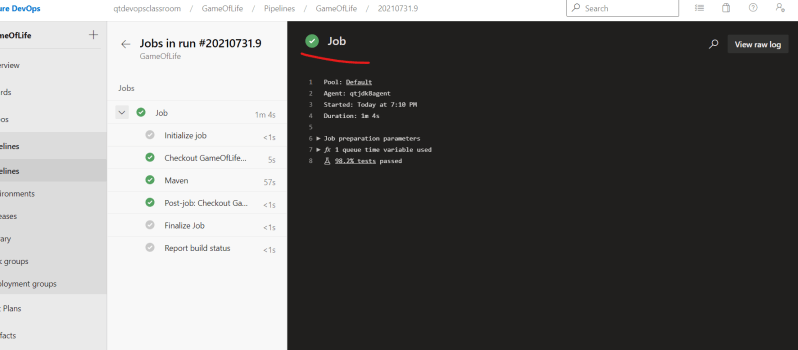
inputs:

mavenPomFile: 'pom.xml'

goals: $(MVN\_GOAL)

jdkVersionOption: 1.8

testResultsFiles: '\*\*/surefire-reports/TEST-\*.xml'



* For working with different technologies in Azure DevOps [Refer Here](https://docs.microsoft.com/en-us/azure/devops/pipelines/ecosystems/ecosystems?view=azure-devops)
* For building .net application on Windows
  + Install Visual Studio Build Tools
  + Restore the nuget package C:\hostedtoolcache\windows\NuGet\5.10.0\x64\nuget.exe restore D:\a\1\s\WebApplication1\WebApplication1.sln
  + Run the msbuild command C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise\MSBuild\Current\Bin\msbuild.exe" "D:\a\1\s\WebApplication1\WebApplication1.sln"
  + Run the testcases C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise\Common7\IDE\Extensions\TestPlatform\vstest.console.exe "@D:\a\\_temp\nml1iy2lu5z.tmp"
* Exercise:
  + Ansible:
    - Write an ansible playbook to deploy game-of-life on tomcat
  + Terraform:
    - Write a terraform template to create a linux instance (in any cloud) and deploy game of life application
  + Docker and K8s:
    - Write a Dockerfile for game-of-life
    - Create a K8s Manifest for game-of-life deployment