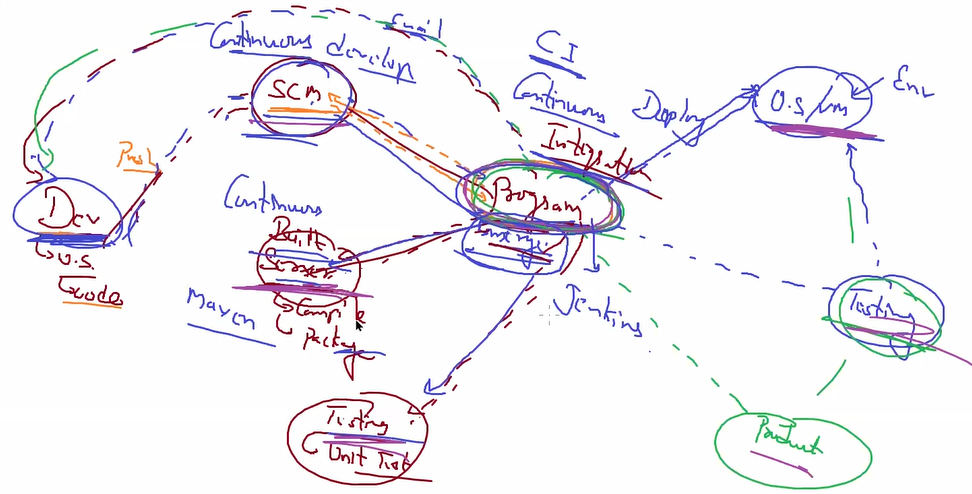
**Session 01 – Intro**

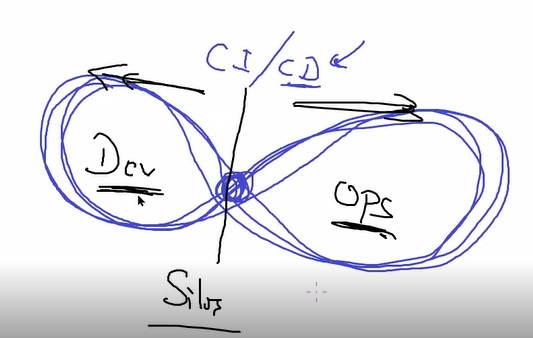
* In our manual process dev will push the code in different branch.
* If code is working fine then we will merge it, solve merge conflicts and then code go to compile/ Build and we make a package of it and after this we share to production world.
  + While deploying we may face many challenges, and may found error, we have to give this code back to the developer and he again work on it.
  + Here we find 2 issues.

1. Developer may forget what he wants to do in that part because of 4-5 days are gone.
2. Our final product comes very late in the production environment.

* Here we don’t want to remove any important step.
  + We want to remove that part which makes this process slow, which can be got by automation.
  + We will make this entire N-to-N pipeline automated.
* This program will fetch code from GitHub, if some error give feedback to dev and again fetch.
  + Then it will build and create package, after this it will test.
  + If all things are fine deploying to the OS, again test & give feedback to OS (team), then release to the production environment.
  + This program is integrating everything.
  + It is known as Jenkins.
    - Jenkins is a **Continuous Integration** tool.



* + In manual world dev team don’t have idea about operation team and vice versa.



* + - **Infinity cycle**.
    - But Jenkins will do this entire connectivity, which make tasks easy.

**Session 2 – Installation//plugin//demo**

* Jenkins is build on top of **JAVA**.
  + JAVA is supported by almost all OS.
  + We only require jdk in a system.
* In our setup we are going to use redhat VM built on top of oracle virtualbox.
  + Download software from drive & copy in the rhel.
    - Rpm -ivh jdk-….rpm
    - Rpm -ivh jen…
  + It will generate random password for Jenkins dashboard.
    - Default username is admin.
    - Copy it from /var/lib/Jenkins/secrets/initialAdminPassword
  + You can install plugin at starting (not recommanded).
  + Change admin password.
    - Click on admin 🡪 manage users 🡪 admin.
* Jenkins is a simple tool.
  + - It can’t do anything much but you can use plugins.
    - Suppose Jenkins want to integrate to GitHub, so for this we can download GitHub **plugin**.
    - Almost for everything we have plugin available in the Jenkins.
      * Plugin mean we are giving some extra knowledge to Jenkins with that specific tools & os command.
  + For doing anything in Jenkins we have to create a job.
    - * Eg, Go to GitHub 🡪 Download the code 🡪 show me output.
    - Job is also known as **item**.
    - If you want to execute job in Jenkins it is known as **Build**.
    - Jenkins will download data in their program, in their own OS.
    - For every job, Jenkins provides separate workspace.
      * Your data will be stored per job secured workspace.
* Jenkins have WebUI, CLI.
  + Nowadays also supporting As code (Pipeline as code).
    - It is something like cloud formation in AWS.
    - After knowing all things in AWS, we can use cloud formation to automate these things.
* Systemctl restart Jenkins
  + Netstat -tnlp | grep java
    - Jenkins is working on top of java.
  + Here you can see Jenkins is working on port 8080.
* To connect with Jenkins, you have to type this thing in chrome tab.
  + Localhost:8080
  + Final moto of Jenkins to create a Job for a Project.
* In our first task we are going to use date command using job.
  + For this you have to use **build shell**.
  + After creating a job you have to build it for running.
  + At left side down you can see history of jobs.
    - You can check console output from it.
* For changing something in the job you have to click on configure.
* You can also run your job from dashboard home.
  + At dashboard we have special symbol to check job status.
    - **Shiny** – working very good.
    - **Shiny + cloud** – working good, but job has some failure.
    - **Cloudy** – job has lots of failures.
  + It is for **visualize** because in real world we have multiple jobs.
* Jenkins with GitHub
  + Go to dashboard 🡪 manage Jenkins 🡪 available 🡪 search for github 🡪 download (GitHub plugin)
  + Create a repository & initialize it in GitHub.
    - Write basic rhel command in one file (script).
  + Create a Job.
    - In source code management you can option for Git.
      * This option comes from plugin.
    - Add your repository URL & branch name.
      * If URL does not exist it will give you error.
    - Click on execute shell.
      * Bash ./mytest.sh
    - Here you can see if you change your code & run job it will take some mini seconds time, but if you run job without changing anything then it will run faster.
  + Whenever you run any job with GitHub plugin, they will check for any change or not.
    - If there is no change, Jenkins will not download same code again.

**Session 03 – triggers//build periodically**

* In Jenkins we are creating jobs.
* Here we can do one thing.
  + We can tell Jenkins keep on checking GitHub account, whenever new code come up automatically **trigger** this job.
  + And that job will create a copy of GitHub code.
  + You can also connect to Jenkins dashboard from windows, but for this you have to disable firewall.
* Create a new GitHub repository and initialize it.
  + Create some html code in your base OS and push to github.
  + Install httpd in rhel and make it permanent enable.
* Create a new Job.
  + Add GitHub URL.
  + Build execute shell.
    - Copy code in your document root of rhel.
    - Sudo Cp -f \* /var/www/html
* But here we also have to build it, manually.
* Here we can use triggers.
  + Suppose some interval of time you create a backup.
  + For this we can use trigger **build periodically**.
  + Format for schedule.
    - Min hr date month day
    - Suppose you want every night 11:50
    - 50 23 \* \* \*
  + After setting trigger job will automatically run after particular interval of time.
* But it will impact a performance.
  + Every minute it is going to run.
  + Here we want some intelligence.
  + Jenkins will only go there only when something has been changed.
  + For this we can use **GitHub trigger**.

**Session 04 – Poll SCM // Github hook trigger // trigger builds remotely**

* Connect from public world to local vm using ngrok.
* We want as soon as something changed in GitHub, Jenkins will download it automatically.
* We have one build trigger – **Poll SCM**.
  + They will go to SCM for particular interval of time if something changed then only it will download code and run something for you.
  + But here we also we are wasting our resources.
  + We are not downloading and running job but every second we are going to GitHub which will waste resources.
* Instead of going manually we can ask GitHub to come to Jenkins when something is changed.
  + For this we can use GitHub events.
  + GitHub has concept called hooks.
  + For this we have to use **GitHub hook trigger with GitSCM pooling**.
  + You have to give IP & API of Jenkins which can listen GitHub.
  + For public IP we have to create tunnel.
    - ./ngrok http 8080
  + Set this inside GitHub webhooks.
    - <https://ip/github-webhook>/
  + Now when you change something in the code it will automatically trigger Jenkins.
* We can use one more trigger, **trigger builds remotely**.
  + It will give you one **API** when someone hit on that URL then it will run a job for you.
  + For this you have to set a token (password).
  + Now you will get one URL (API), you can do now whatever you want to do.
    - [https://admin:<os\_pass>@ip/job/<job\_no>/build?token=<token](https://admin:%3cos_pass%3e@ip/job/%3cjob_no%3e/build?token=%3ctoken)>

Session 05 – cluster//distributed Jenkins//executor//concurrent job//ssh build agents

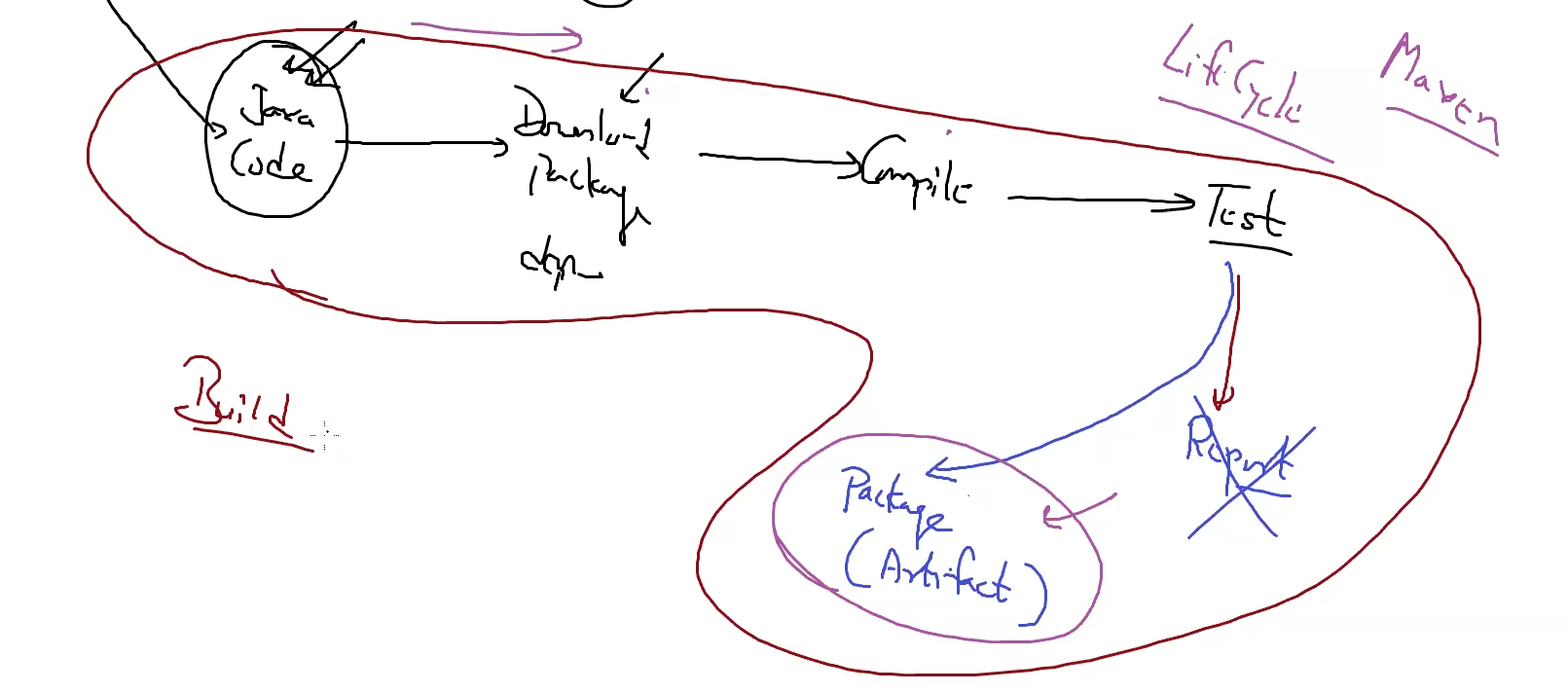
* Till now we are using Jenkins jobs and resources from same OS.
* We have multiple team running so may be security issue comes up, we require CPU & ram.
* Here role of Jenkins cluster or distributed Jenkins comes.
  + We can launch one independent virtual machine // OS.
  + Suppose team 1 wants python 3.6 and IN connectivity.
  + Team 2 wants Java & no IN.
  + Team 3 wants python 3.6 & IN connectivity.
* Here team 1 & team 3 wants same things so we can give one OS and both teams are working on that.
* In all of this perspective VM//OS is known as Node.
  + But for managing this 3 we launch one more Node.
  + Master will manage all nodes.
  + This cluster is managing a Job so we can say it is job cluster.
    - Based on requirement we distribute job, that’s y it is known as distributed Jenkins cluster.
    - In slave you **don’t need Jenkins** installed.
  + Till now Jenkins is working with a single node cluster.
* By default it is set in master that you can run 2 job parallelly.
  + If you write one more job3, then it will wait in a queue.
  + This is known as **Executor**.
* For checking this thing you can use sleep command for 30 seconds and try to run 3 job.
  + 2 job will work, one is in queue.
  + manage Jenkins 🡪Manage nodes and clouds 🡪 master 🡪 configure
    - here you can change executor to the 3 or 4 or any.
* But if you try to run same job multiple times parallelly, it will not work.
  + Once the first job is finished then other job will work.
  + If you want to run same job multiple times parallelly, then you have to click on Execute concurrent builds if necessary in general of a job.
* Running same job parallely is known as concurrent job.
* Go to manage Jenkins 🡪Manage nodes and clouds
  + Here you can see we have one slave.
  + New node 🡪 node name 🡪 (You have to install SSH plugin so master can use this to go there and launch a job).
    - We require plugin for agent.
    - You have to install **SSH build agents**.
  + Now inside launch method you can ssh.
    - Launch method 🡪 ssh.
    - Give remote directory name.
      * It will work as a root directory.
      * All data of workspace job & all things will be stored here.
    - IP & Add credentials.
    - Host key verification strategy 🡪 Non verifying verification strategy.
      * Connecting at first time we have to pass yes/no for transferring key.
  + Save this.
  + It will take some time for launching agent.
* Now you can see you can launch 6 jobs parallelly.
  + 4 from master.
  + 2 from slave.
* If you want to launch some job in particular node.
  + While configuring node set **usage 🡪 only build with label expression**.
  + Then you can give node name in general 🡪 restrict where this project can be run

Session 06 – windows//cloud Jenkins cluster

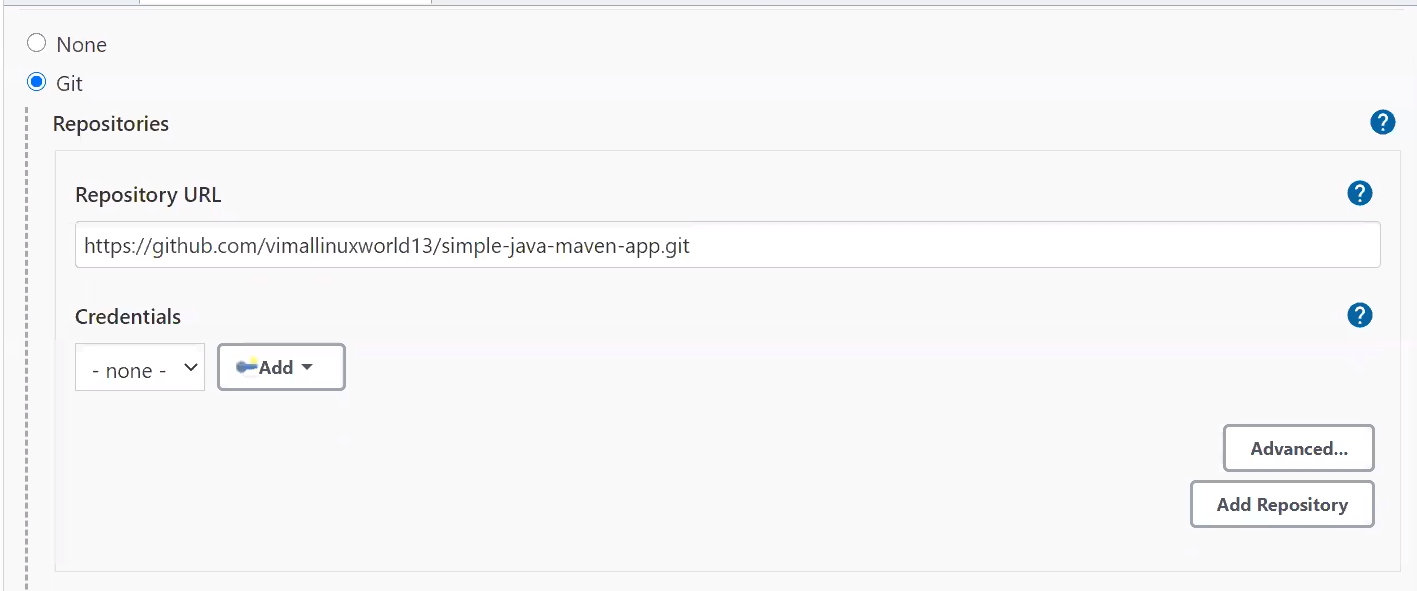
* We want to make our windows also as a Jenkins slave.
* But here we cannot use ssh agent.
  + We can use somehow but mostly we don’t do this.
* So from windows you have to manually go to Jenkins master and ask them to give java agent and run this program in your windows.
  + Here windows use **JNLP protocol** for connecting to Jenkins master.
  + Here our agent is java agent so for running this we require java runtime in our system.
  + So, we require java **jdk** in windows.
* Create one more node as worker node.
  + **Launch method 🡪 launch agent by connecting it to master**
  + Here agent is connecting to master so we don’t require to give IP while configuring.
  + But if you see it is saying you something like JNLP port disabled.
    - Go to security and disable it and make anyone can do anything.
  + Now you can see it will give you command for connecting to the master.
* You have to download **agent.jar** file in windows.
  + It is available on that screen.
  + Now run second command.
    - Agent successfully connected to Jenkins master.
* Now we are going to add a cloud instance to our cluster.
  + Give credentials.
  + No ssh verification.
  + But you have to install java there.
    - Yum install java-1.8.0-openjdk
    - Mkdir /ws1
    - Chown ec2-user /ws1
      * We are changing file ownership to ec2 user.
  + Now run any job and check it is working or not.

Session 07 – maven//pom.xml//mvn integration plugin

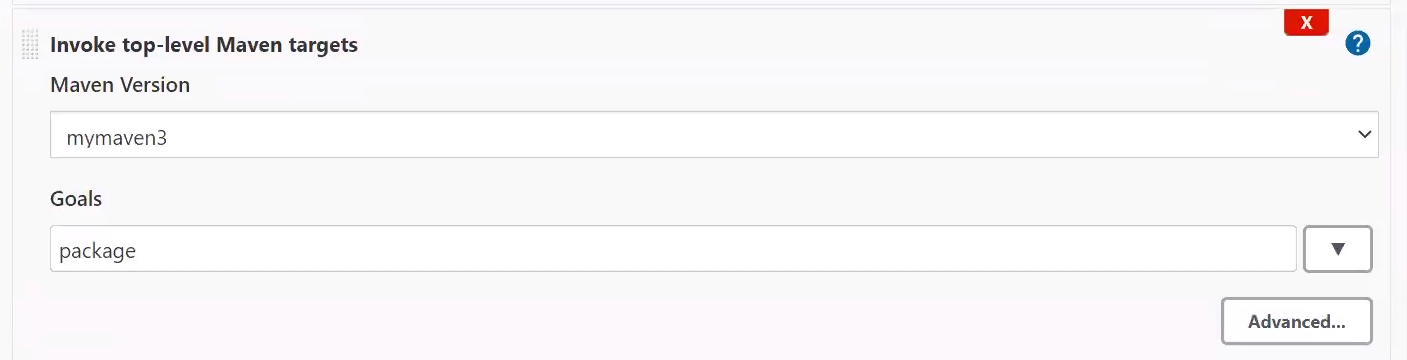
* Maven is one kind of automation tool meant for **automate build process**.
  + **Automate lifecycle** for software development.
* In java code you must first **compile** a code and **run** compiled file.
  + In python we directly run our code.
  + For compilation also we require lots of things.
    - Maybe we have one centralized storage where we store all things we are using on regular basis.
  + So, we can go there manually and download all codes easily.
    - But problem is may be one software is dependent on another software and may be on some version of that software.
  + So, **maven** is a software which can do this automatically.
    - They go to repo and download all dependencies and all other things with their required version.
    - Maven can also do many other things.
  + **Pom.xml** is one kind of configuration file for maven.
  + When we want to compile our code, we go to pom.xml file.
  + Maven will download all packages.
    - Maven will also compile our code.
    - It will also do unit testing; code is working properly or not.
    - It will create a report of unit testing.
      * One of the ways to create a report is surefire.
  + Suppose we have created a complete package (archive).
    - If app is website, then archive package is known as **war** (web archive).
    - archive for enterprise application is **ear**.
    - Archive for normal java basic code, normal app then **jar**.
  + Maven can also do this thing for you.
  + Here this final package is known as **artifact**.

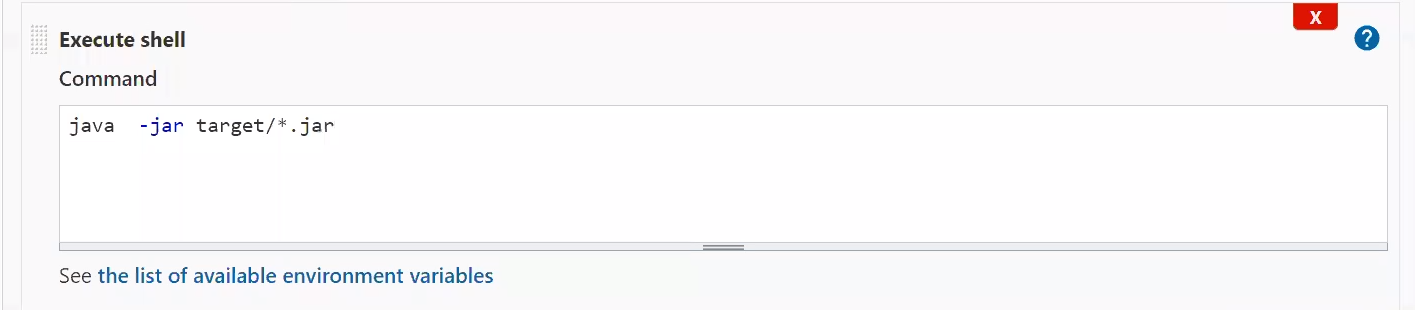


* + This complete diagram is known **build**.
    - It is also known as build process or **build pipeline**.
      * Here one of the part fail then other part will not work.
      * Everything is dependent on each other.
      * This is known as **pipeline**.
    - We must just provide a source code to maven.
  + Maven also works for some other languages.
    - But 99% we use maven for java.
  + After maven build this now Jenkins part comes.
    - We can also use maven in our pipeline.
* Download maven from official site.
  + For linux you have to use tar file.
  + Tar -xvzf apache-maven….
    - Cd apache-maven…
    - Cd bin
    - Ls
  + Here is mvn code which we have to use.
* We can check our environment with some simple maven code available online.
  + Now instead of javac command we can use maven.
  + Which software maven must install for this maven take help from pom.xml file.
  + Mv /apache-maven-3.6.3 /maven3
    - Here we have to copy file in / folder so any user can run this.
  + Echo “Export MAVEN\_HOME=/maven3” >> /root/.bash.rc
    - This is default name for maven.
    - We have to set this so Jenkins can use maven.
  + Echo “Export PATH=/maven3/bin/:$PATH” >> /root/.bash.rc
* Download some demo maven file and run it using **mvn compile**.
  + It will go to pom.xml file & check which software to install.
  + It will install this software from maven central.
    - They will install this in your local cache.
  + **Mvn packages**
    - Here you don’t have to provide format like jar, war, ear.
* Here we have set 2 goals compile & packages.
* You can run final package with **java -jar <package\_name>**
* Lets create one more goal,
  + **Mvn test**
* Jenkins
* Install **maven Integration plugin**.
  + Go to manage Jenkins 🡪 global tool configuration 🡪 maven 🡪 add maven
    - Give maven\_home & name.





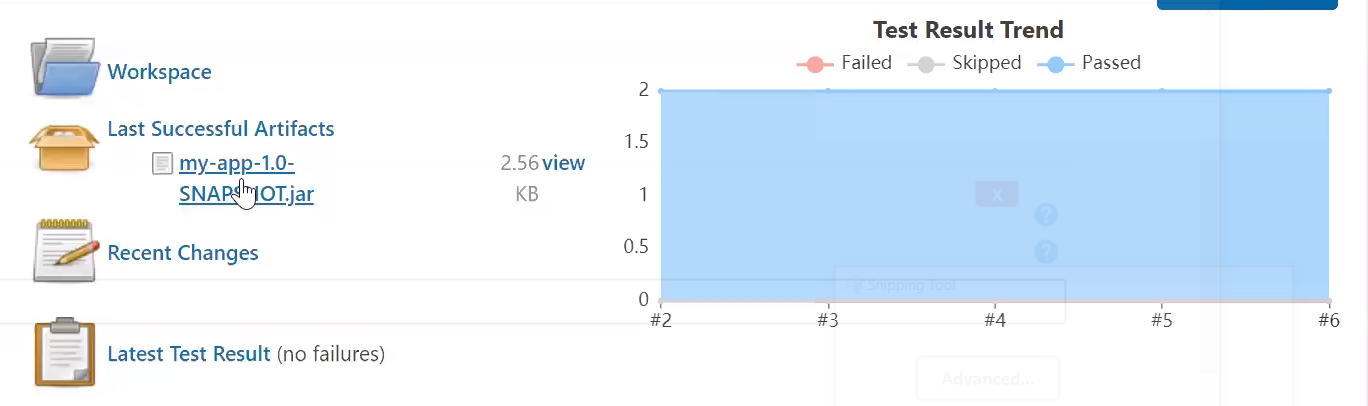


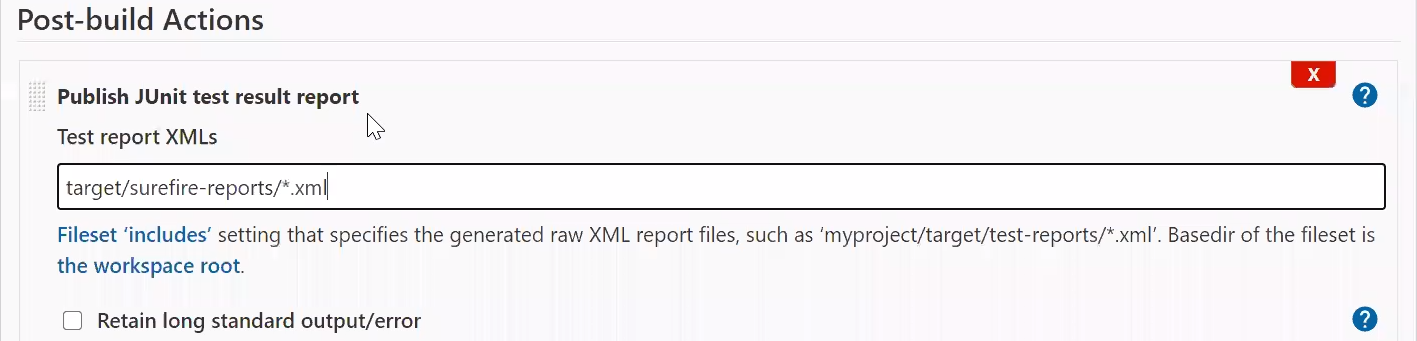


* + Here maven will give you report in xml file which is harder to read so here we can use post build actions.



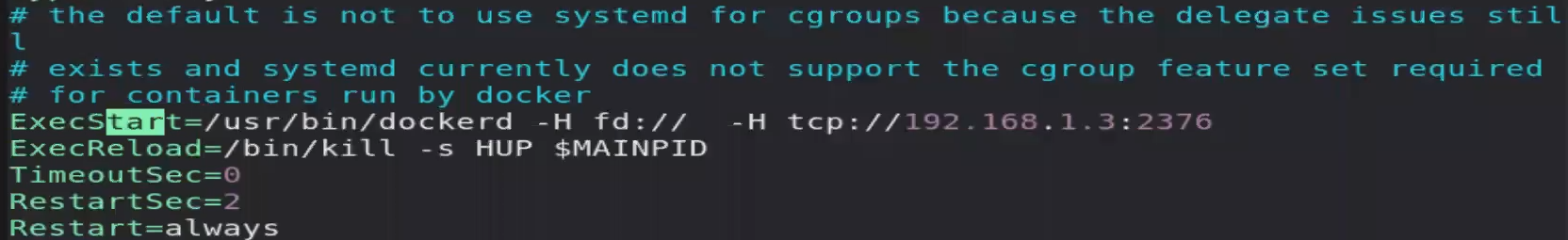
* + If you **archive the artifacts**, then instead of going to target folder we can directly see our package in job dashboard.
  + We can directly download from here.



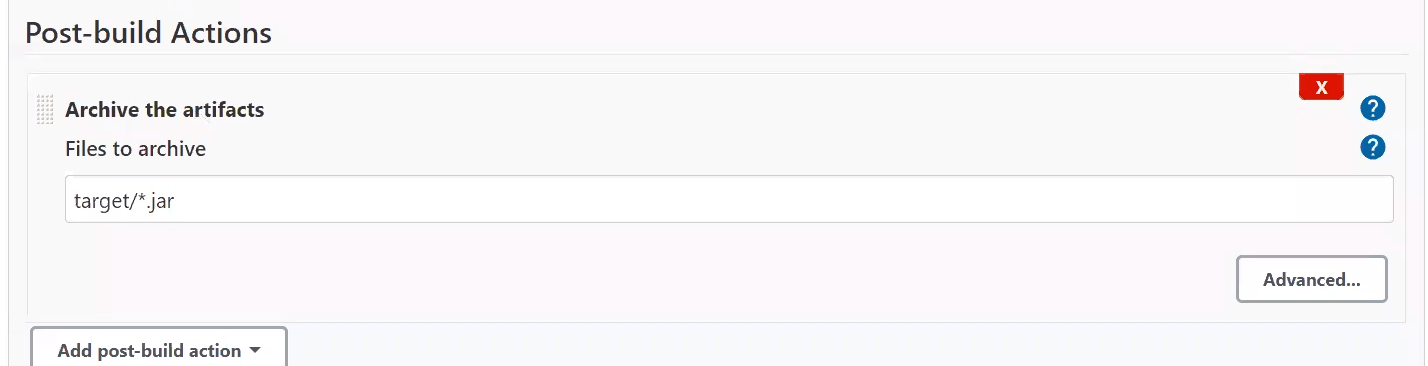


Session 08 – dynamic provisioning

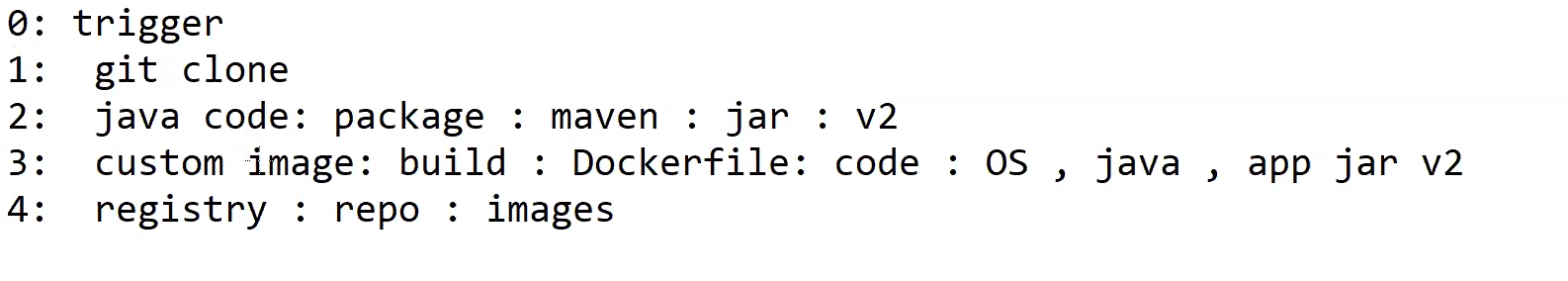
* Suppose we have master-slave architecture.
* Here we have one specific pod for maven job.
  + But we have to keep on running that node so Jenkins job will work, but it will waste our money & resources.
  + And if we stop the instance then how will our job work?
* So instead of this we want to launch instance when the demand comes.
  + And when our job is finished it will terminate automatically.
* Here in one node we have Jenkins master and in second node we are launching docker env and we have also created one image of Jenkins.
  + Install **docker plugin**.
  + But docker engine don’t have capability to connect from outside.
  + So you have to add support from outside world to docker tcp port.
  + Go to /usr/lib/system/system/docker.service



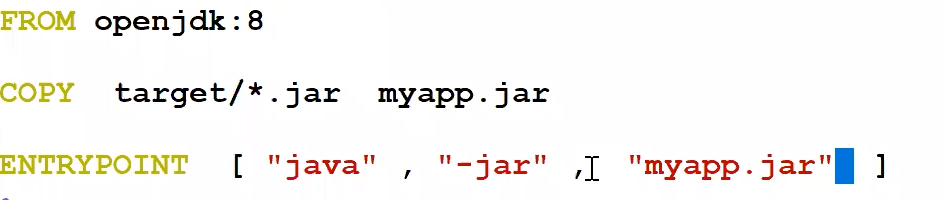
* + - Add TCP support for local IP and give port number.
      * We normally use 2376 port for docker.
    - This concept is called socket binding.
  + Here we are changing in daemon file (systemd of base OS)
  + For any file you change in system you have to reload it.
    - Systemctl daemon-restart
  + We have to also restart docker.
    - Systemctl restart docker
* Go to manage nodes and clouds 🡪 configure clouds
  + Here we don’t need fix node.
  + Select docker
    - Provide IP.
      * Tcp://192.168.1.3:2376
    - Here we don’t have to tell any credentials because we haven’t configure authentication for this.
      * May be firewall issue comes up.
    - Click on enable.
  + Now we have to give con name, image name.
  + Docker agent templates,
    - Name
      * They will add this as a prefix and some random values behind it.
    - Image name – vimal13/Jenkins-slave-maven
    - Add labels.
* Now when you create a job with this label, then it will launch a new container and run his job there.
  + Add label here so it will directly launch to container.
  + Add some demo command to run.
* Now download some demo code of maven and run maven job.
  + But here you face one issue, when the job is completed container automatically terminate.
  + But here our final need is jar file which is build by maven.
  + For this we can do one thing.
    - You can use archive the artifacts in post-build action.



Session 09 –



* We are going to build pipeline like this.
* Developer will push the code.
  + We clone this code and create a package from it.
  + We create a custom image from this package and upload into docker hub.



* + - Docker build -t japp:v1 .
    - Docker tag japp:v1 vimal13/japp:v1
  + Now push this into docker hub.
* So lets do this setup using Jenkins.
* Here we need 2 plugins.
  + **Cloudbees docker build and publish plugin**
  + **Docker build step**
* Create new job.
  + In general 🡪 GitHub project
    - It will add one icon in dashboard from there you can directly go to docker hub.
  + In SCM 🡪 Git 🡪 repo url
  + In build 🡪 invoke maven 🡪 maven version & goals
  + In build 🡪 docker build & publish 🡪



* + - Also add you registry credentials.
    - But here you have to add Jenkins into docker Group or use sudo.
    - You might also face issue of selinux so we can stop selinux.
      * Setenforce 0

Session 10 – Pipeline//delivery pipeline plugin//build pipeline plugin

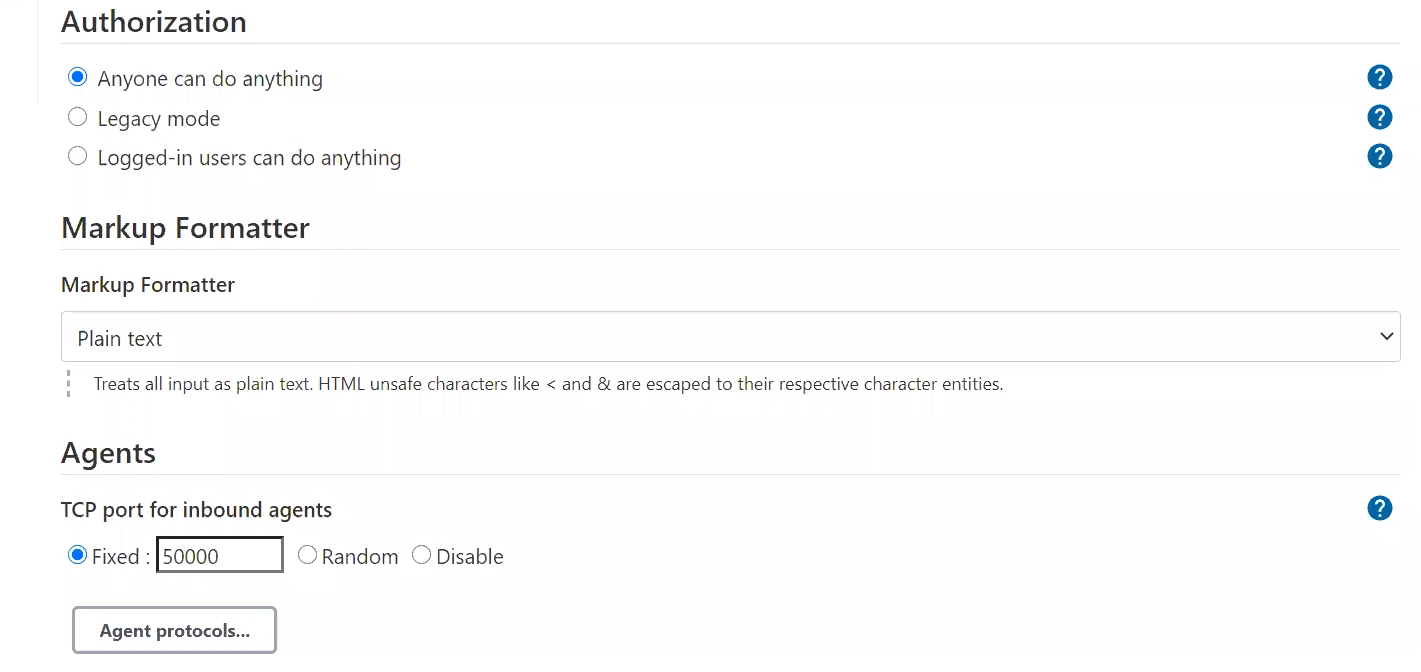
* Suppose you want to clone GitHub url, create a mvn package and deploy it to app server to real production environment.
  + Here in one worker node we have git, in another we have mvn and in another system we have app server deployment setup, we have created separate jobs for this 3 things.
  + Now there is no meaning of app server if package is not build properly.
  + Here we want to run this job in **same sequence**.
  + Here we are linking one job with each other.
    - This is known as **pipeline**.
  + Pipeline is complete end-to-end process.
  + When complete 3 job finished, we can say pipeline completed.
    - Here by just clicking on job1 our complete job finish.
* So here we can use **triggers**.
  + In second job we can set that this job only run when job1 runs successfully.
  + If first job fails then we can run some another job.
  + Now when you first job it will automatically trigger other jobs.
  + Here we cant able to visualize it properly.
* So we can use **delivery pipeline plugin**.
  + So you can create a new **view** with delivery pipeline view.
  + In your delivery pipeline **view 🡪 edit view**, click on **enable start of new pipeline build**.
  + Now you can directly run your job from there.
* Suppose you have some error in second job so third job will not work.
  + Somehow you fix the error, now there is no meaning to run first job again.
  + We directly want to run from second job.
  + So for this you have to enable option called enable **rebuild** in edit view.
    - Now you can directly run from second job.
  + We have one more pipeline called build pipeline which is more advance than delivery pipeline.
  + For this we have to install **build pipeline plugin**.
    - This plugin will give you some better view and some extra control option.
    - Here you have to just tell your initial view.

Session 11 –

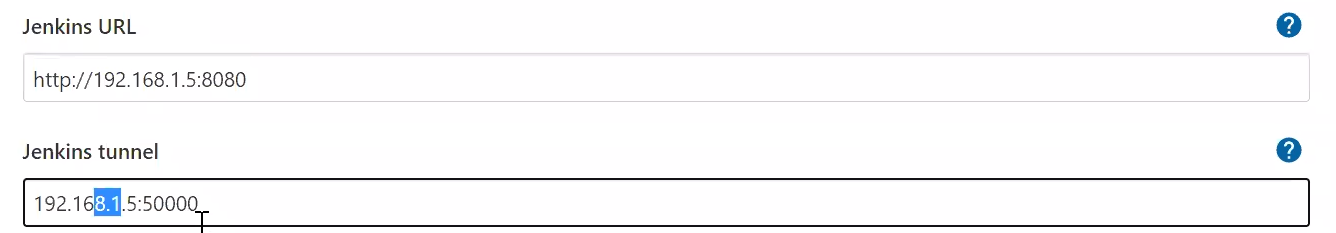
* Kubectl cluster-info
* We have to connect K8s from Jenkins.
  + We have to download **Kubernetes plugin** for this.
  + For this we have to provide basic details to Jenkins, like IP, Port, User, Password.
* Suppose your developer has created a JAVA file and uploaded in the GitHUb.
  + We use maven to compile/build a package.
  + We create a image from it and push to Docker Hub.
* We have to provide config file to Jenkins plugin.
  + **Cd /etc/Kubernetes**
  + We want to transfer **admin.conf** file.
  + But we have to give permission so we can transfer it to windows using winSCP.
    - You have to change IP of cluster in this file.
* If you check we have **different** network card in Minikube VM.
  + We cant change minikube VM n/w card which is by default host-only.
  + So we can **add** a new network card in Jenkins VM.
* Go to Jenkins 🡪 manage Jenkins 🡪 manage nodes and clouds 🡪 Configure clouds.
  + Select Kubernetes.
  + Kubernetes cloud details
    - Add credentials.
      * Here through credentials we are giving admin.conf file here.
        + This file contains all details like IP, user password (secret-key).
      * For this select secret file option.
    - Add this credential and click on test.



* + - In Jenkins URL they are asking you for Jenkins IP.
      * Here k8s VM will work as a Jenkins agent and current VM is Jenkins master.
      * Jenkins agent has to know where his master is.
      * So here we have to give current VM IP where Jenkins is running.
    - In pod label give some labels.
  + When Jenkins contact to Kubernetes they download default image from hub, launch a pod.
    - This image contains **Jenkins agent**.
    - Now Kubernetes becomes a worker node.
      * You can provide some extra things or change default image using **pod template**.
    - You have to give pod name and labels in pod template.
* Create a new Job 🡪 select on **restrict where this project can be run.** 
  + Give your label name here.
  + For simplicity we have used **date** command in build shell.
* Here we are using a port so Kubernetes pod can connect to this and using this pod it can connect to Jenkins.
  + For this you have to go inside Jenkins 🡪manage Jenkins 🡪 global security
  + Here you have to add a port number.
  + This port will work as a proxy and create a tunnel for you.
  + Many times we use port number 50000.



* You have to write this port number with IP address in Jenkins tunnel configure clouds.



* Plugin Kubernetes use in Jenkins for image which have Jenkins agents inbuild, it is by default use JNLP protocol and Jenkins tunnel port.

Session 12 – automating job using groovy code// DSL job// seed// child

* Till now we are running Jenkins manually.
* But we can also run it automatically using code.
  + This is known as Job As Code.
  + Infrastructure as a code.
* Suppose developer wants to create a job by himself.
  + But we cannot give user password of Jenkins portal to developer.
  + So we can use this automated way.
  + We can ask developer to create a template of Jobs which they want to do and provide to the Jenkins admin.
  + Now Jenkins admin will run this code and job automatically created.
* For this we can also create a code in java language but we have to create a class and lots of things we have to take care.
  + JVM supports many languages so for this kind of requirement we use Groovy language.
  + Jenkins supports **groovy** syntax.
  + For this you have to install **job DSL** plugin.
* Here we have to create one job in Jenkins, duty of this job is to run the code given by developer and create job automatically.
  + This job is known as **Seed job**.
  + And automatically created jobs are known as **child job**.
* For creating seed job in build step choose process job DSLs.
  + Here you have to add path of groovy code.
    - You can also check document from here.



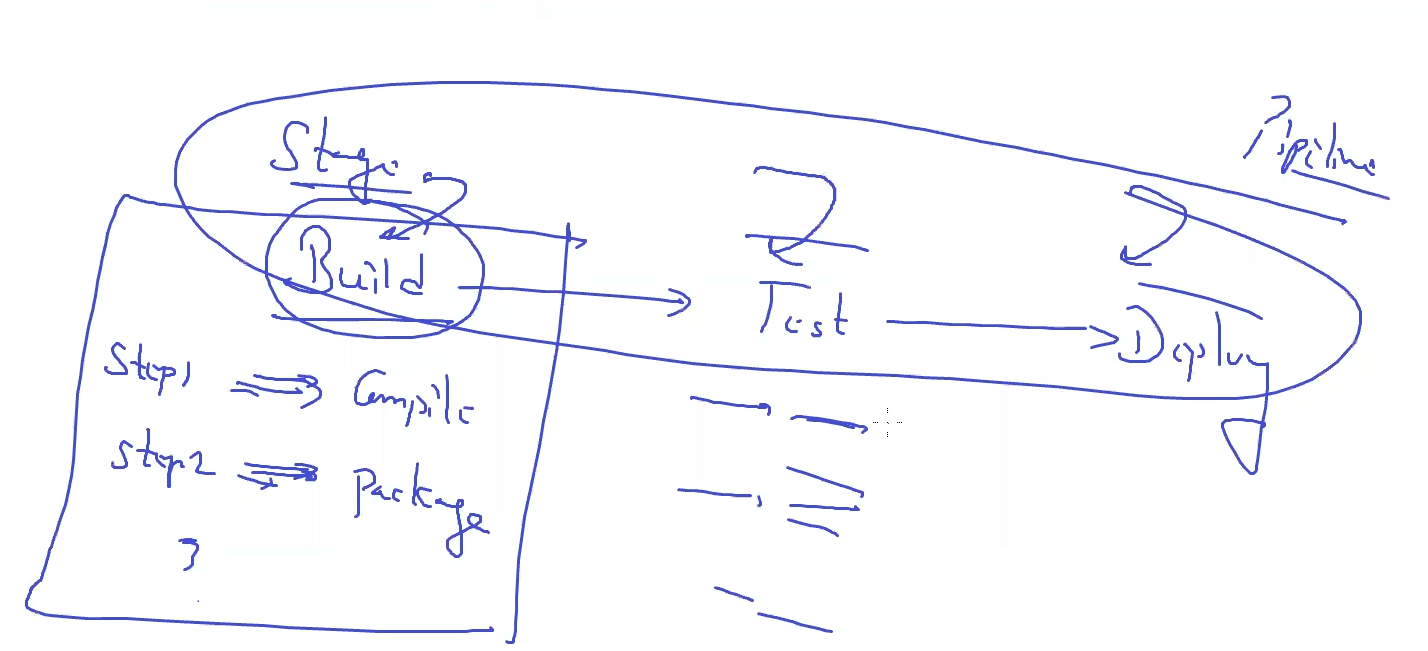
* + - Now when you run this job then it will create a new job with name example.
    - In document you can search for step (for running shell).



* + Here maveninstallation should be part of argument so you have to use inside maven.
    - Here jar file is created so instead of running data command we can run java -jar target/\*.jar.

Session 13 – pipeline as code// pipeline (plugin) // parameter// stage – step

* In last lecture we have discussed how to write job as code.
  + Developer create a DSL code and from this we create a job.
  + But we have to manually set some triggers do this job after this job, launch child job and some other things are also manually.
  + Here topic comes **pipeline as code**.
  + For this you have to install **pipeline** plugin
* Job DSL plugin is created by Netflix while this plugin is native plugin.
* For creating pipeline as code we have to write the code in **Jenkinsfile**.
  + This is specific name.
  + It supports both **declarative** and **imperative** language.



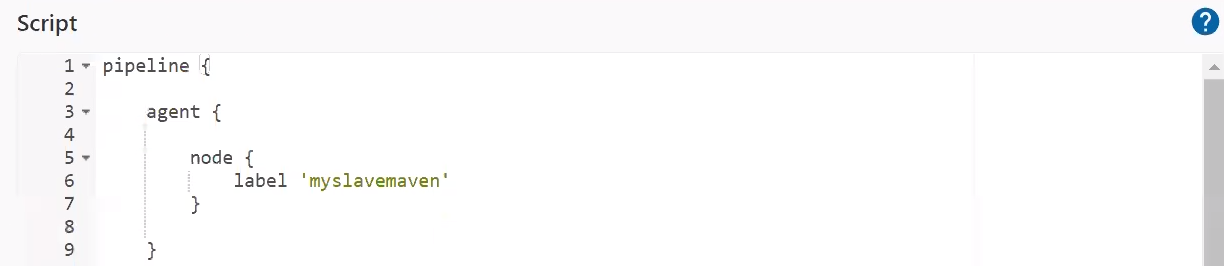
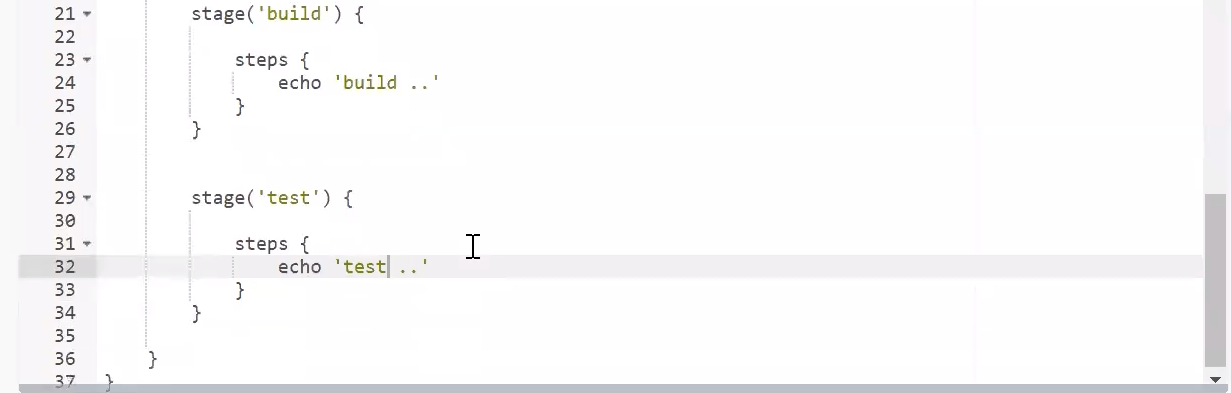
* + In above pipeline we have 3 stages.
  + First stage has 2 steps.
* Here while creating job instead of **freestyle** project we will select **pipeline**.
* We are creating pipeline directly here.
  + We have to write all stages inside stages.
  + This plugin has own personal UI/view, so we don’t need to use delivery or build pipeline for visuals.
  + Now this is **not known** as **job** this is known as **pipeline**.



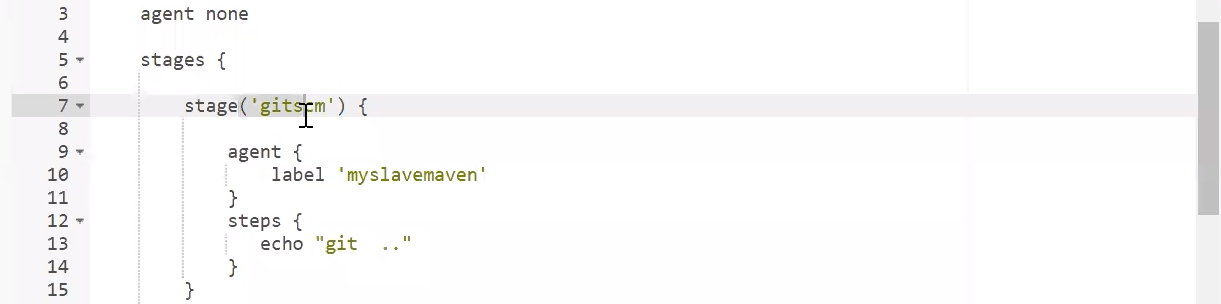
* Suppose in normal job you want to get some input from user while running code.
  + For this click on **this project is parameterized** in general part.
  + And you can use in your job.
  + Suppose we want to echo this using build shell.
    - Echo $x
    - Use this in build shell.

Session 14 – snippet generator // global tool use

* We are going to create a pipeline for maven package with four stages.
  + But here we want to create separate container dynamically for separate job.
* Create a dynamic docker con with label & maven image.
* Using labels in pipeline with some example.

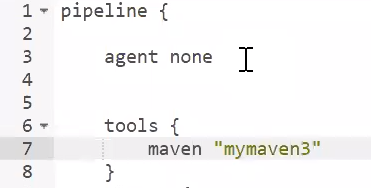
* + But here all job will run in same node but we want to create a separate container for per **stage**.



* + Here we can use this.
    - Make agent none, and give agent inside a stage, it will create a separate agent for separate stage.
  + Below this you can see option for pipeline syntax
    - When you click on this it will launch some **snippet generator** webUI.
    - Here you can generate a snippet (code) for your requirement.
  + You can check this using GitHub example and as you can see it will dynamically generate a pipeline snippet (code) for you.
    - Paste this code inside your script.
* Lets check maven and git example.



* + Here path is already set.
* Suppose you want to run maven where path is not already set.
  + We have set maven path in global tool configuration.
    - This page is known as **global tool**.
  + So we have to follow something like this.
    - You have write tools outside everything (stages).

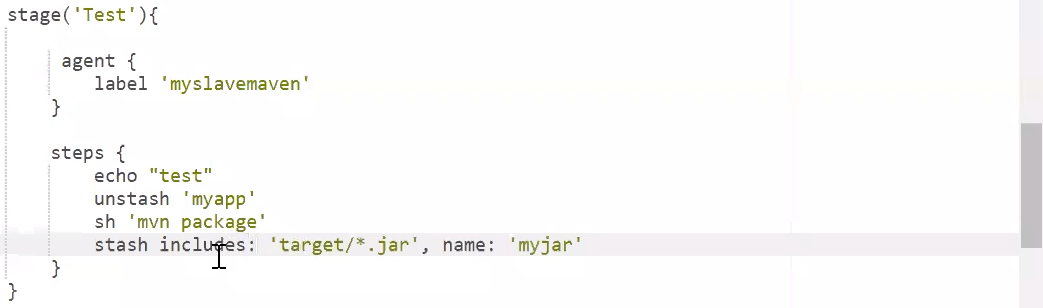


Session 15 –

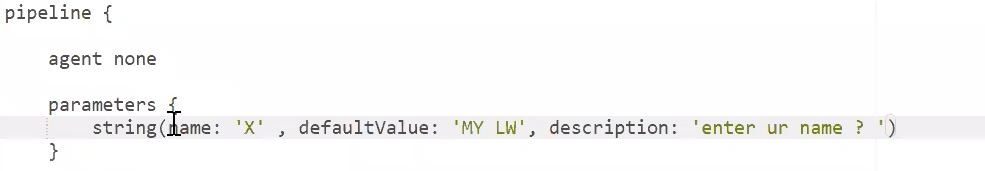
* Here we have one issue.
* We have dynamically provision OS so if we use separate job for stage then as soon as it download source code from git, then it will terminate and we cannot use this in another container.
  + For solving this we use all code in same stage.
  + Here we have to pass output of first stage into second stage or you can say pass to another container.
  + Here we can get output file/folder in master as a temporary storage.
    - Here the life of this storage is life of pipeline.
  + We can send this storage folders to another agent which wants to run this code.
  + This concept is known as **Stash**.
    - We can give name to stash, (give name to file) so we can easily recognise.



* + - If you want to include all files then use \*.
  + **Unstash** these files where you want to use this.


* Now lets use parameter in our Pipeline as code.



* Jenkins manage some extra variable per job wise.
  + Like we have job name stored in some variable.
  + Build number, id, executor number and lots of things they store.
* In Jenkins we have 2 types of variables.
  + **Job variable** 
    - We can directly use it.
  + **User defined variable**
    - This is stored inside different **namespace** called **params**.
    - So for print above variable you have to use params namespace.



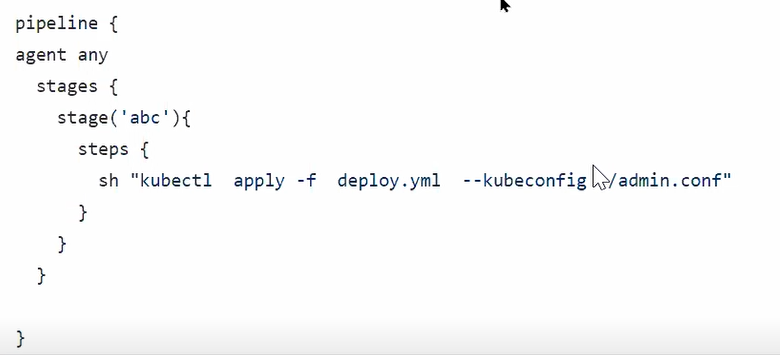
Session 16 –

Session 17 – IAM

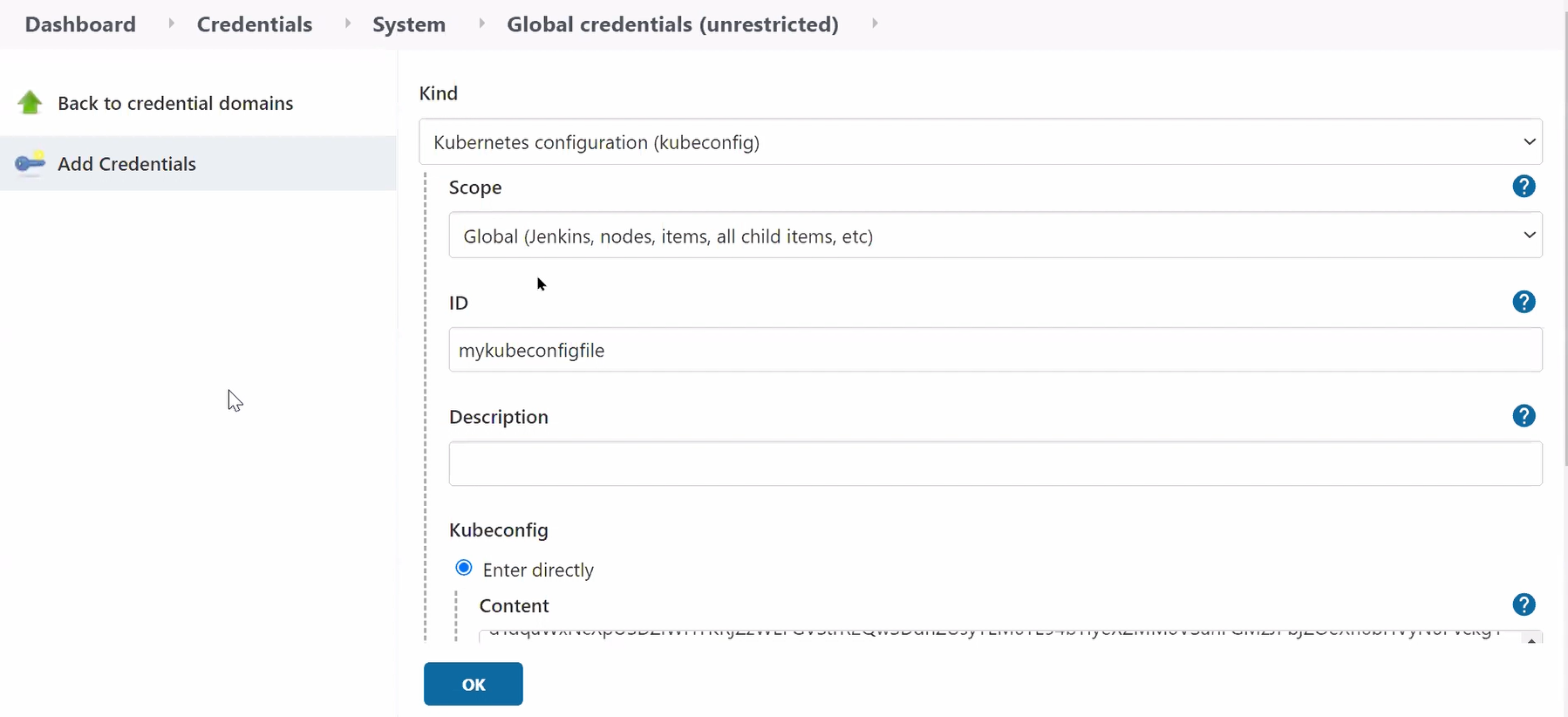
* Till now anyone can access portal using login name & password.
* But in Jenkins we can set particular identity has particular access.
  + Manage Jenkins 🡪 Configure global security.
  + **Jenkin’s own user database.**
    - Here you have to give database from where Jenkins can check password.
    - Jenkins supports almost all type of security check like kerbos and so on.
* You can see on top right corner we have option for login.
  + Here we have tell Jenkins to give access to user, but anonymous still has access to portal.
  + So you are logged in or not you can access the portal.
* Role based access control (RBAC)
  + Instead of giving access to particular person, here you can give power to role, like manager of company has this role and so on.
  + And manager has assigned with manager role.
  + For this you have to install **role-based Authorization strategy** plugin.
* Go to manage Jenkins 🡪 manage and assign roles.
  + In assign roles you can see that we have assigned **anonymous** with role called **admin**.
  + You can add users here and assign it to any role.
  + Here this user has full power, but we want to provide limited power.
* Create 2 new user with name eric & tom.
  + We want to assign developer job to associative role & operation job to executive role.
  + Inside manage roles you can add role called associative in item roles.
  + Here you can add job wise user also.

Session 18 – Kubernetes

* We have already talk about continuos building with k8s, here we use k8s as a slave.
* But here we want continuos deployment.
* For this you have to download a plugin called **Kubernetes continuos deploy plugin**.
* Here we have uploaded yml file for deployment and Jenkinsfile both on the github.
  + Here we need kubectl command and kubeconfig file in our system.
  + Kubeconfig file know where is our cluster is running.



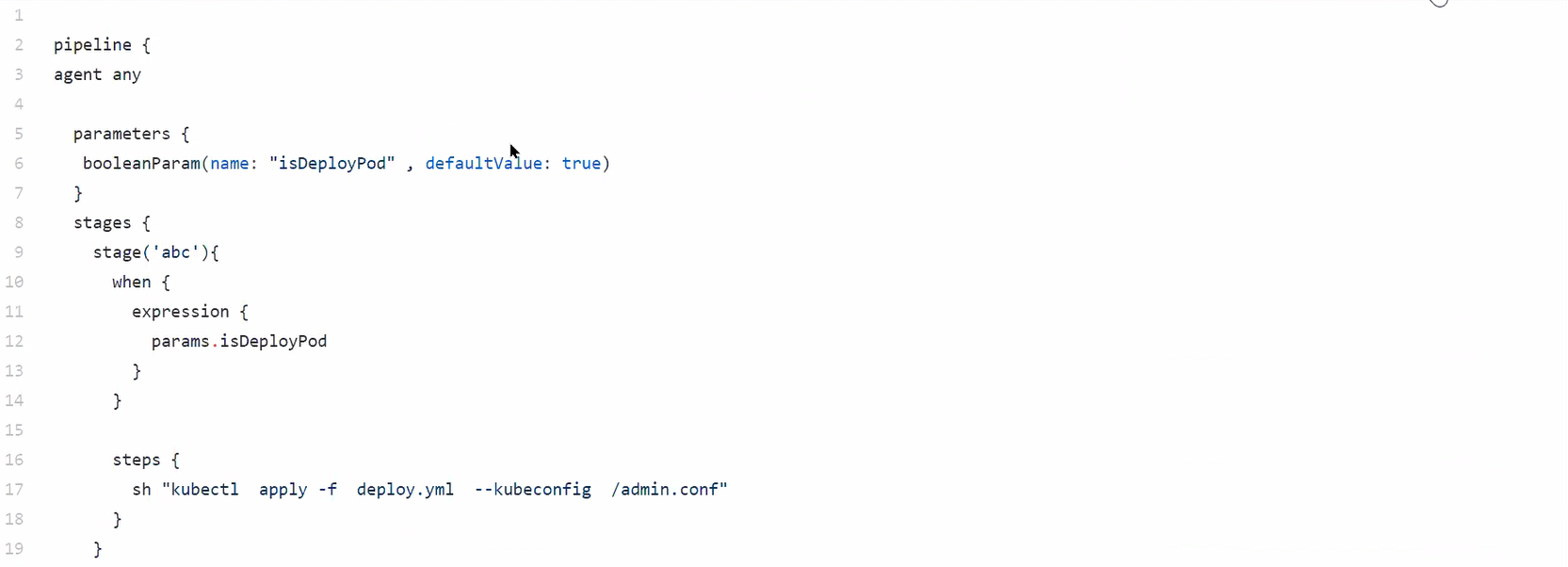
* + Here we have manually write a command inside Jenkinsfile.
* Now lets use plugin for same.
  + For this inside steps we can write **kubernetesDeploy(configs: “deploy.yml”)**
  + It will give you error because it don’t know where is our cluster.
  + Here we 2 can use 2 keywords.
    - We can use **kubeConfig**, but for this we need kubeconfig file in our workspace, so we have to upload it in the github.
    - Instead of this we can use **kubeconfigId** keyword.
  + For this go to your dashboard 🡪 security 🡪 manage credentials.
    - Add a new credentials here.

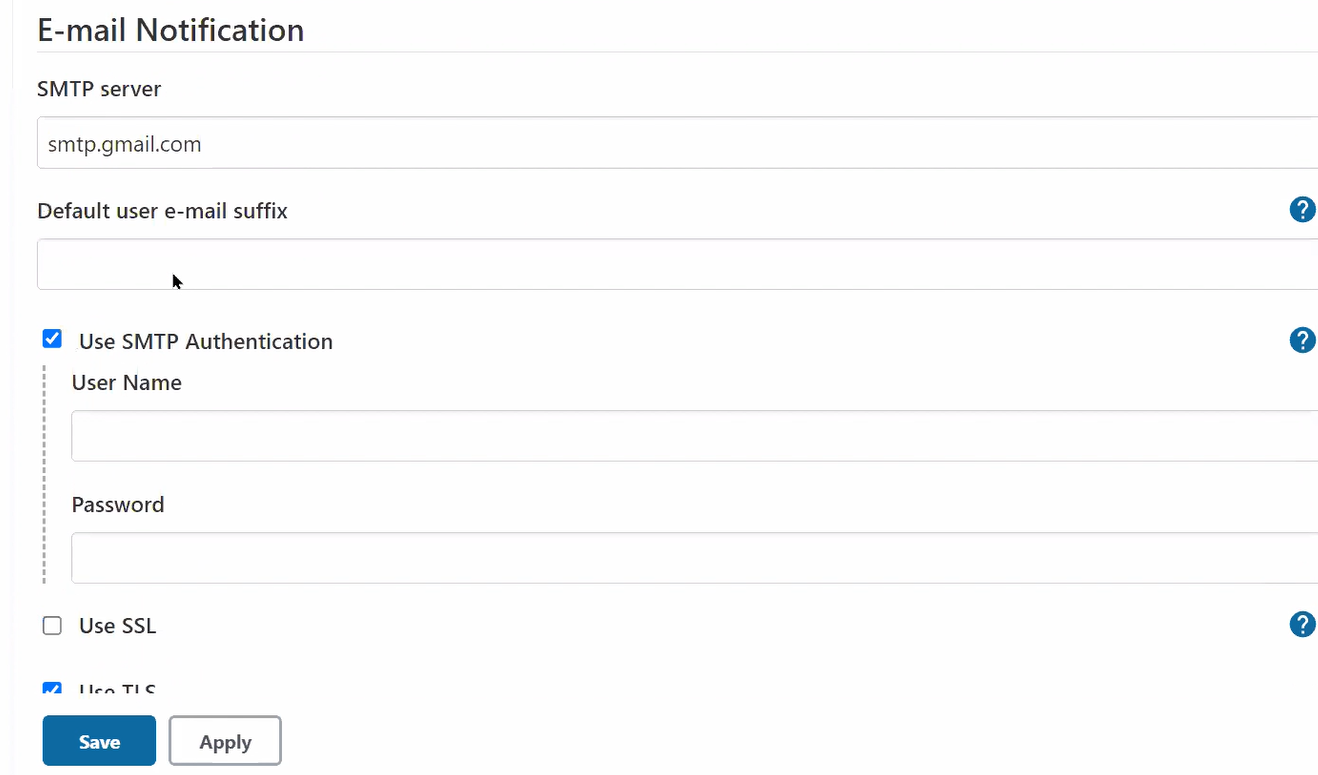


* Final Jenkinsfile,



* It will also not run, but here reason is this version is not supported by this plugin.
* You can use this type of syntax for deciding on the fly to run the file or not.



* You can also use Jenkinsfile for mailing.
* Inside configure system, give your mail server name.
  + Here you can set your own mail server or **gmail server**.
  + 
  + Here you can add your gmail username-password.
* **Session 01 – Intro** 
  + Jenkins intro
  + CI tool
  + Infinity cylcle
* **Session 02 – installation//plugin//demo** 
  + Java – JDK
  + Rhel installation
  + Plugins
    - GitHub
  + Job
  + Build
  + Shiny-cloudy ball for visualization
  + 2 demos
  + Build shell
  + Command
    - Rpm -ivh jdk-….rpm
    - Rpm -ivh jen…
    - Systemctl restart Jenkins
* **Session 03 – triggers//** 
  + Trigger
    - Build periodically
    - Min hr date month day
* **Session 04 –** **Poll SCM // Github hook trigger // trigger builds remotely**
  + Poll SCM
    - interval of time
  + GitHub hook trigger with GitSCM pooling
    - GitHub webhooks
    - <https://ip/github-webhook>/
  + Trigger builds remotely
    - API
    - [https://admin:<os\_pass>@ip/job/<job\_no>/build?token=<token](https://admin:%3cos_pass%3e@ip/job/%3cjob_no%3e/build?token=%3ctoken)>
* Session 05 –

**Installed plugin**

* GitHub
* Ssh build agent
* Maven integration plugin
* Docker plugin
* Cloudbees docker build & publish plugin
* Docker build step
* Delivery pipeline plugin
* Build pipeline plugin
* Kubernetes plugin
* Job DSL
* Pipeline