

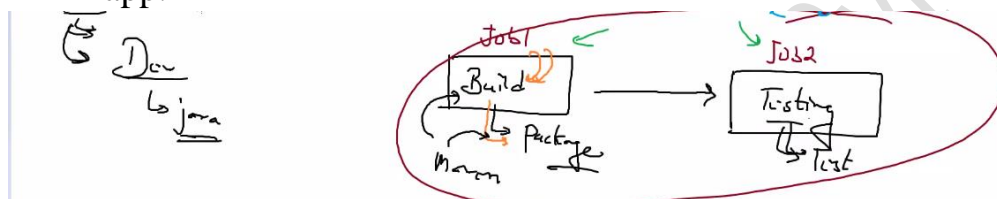


Jenkins Session 07

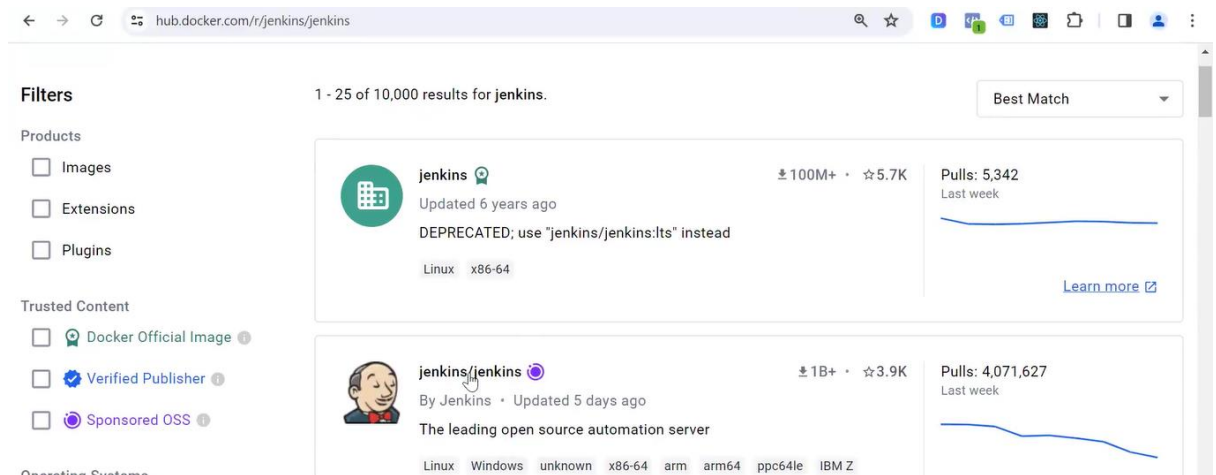
Summary 17-03-2024

- If we want to automate anything in this world Jenkins can help us in the automation of that but first, we should have the proper knowledge of the technology that we want to automate.
 - Technology can be machine learning, cloud computing, generative AI etc.
- Having the knowledge of the technology is must because without that we will not be able to automate it.
- Jenkins perform end to end automation once the jobs are created, but the thing is we must create the job first to do any automation task.
- There is one challenge in the Jenkins that is not being automated till date and it is the **job creation** part. We have to create the job manually.
- Other challenge is the dependency issue, it basically means depending on the other team for performing any task.
- There are three ways for creating the jobs in the Jenkins.
 - Using the Web UI
 - Using CLI
 - API

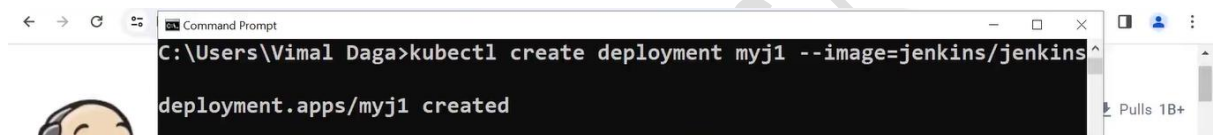
- In real world we have to create a lot of jobs to solve any use case multiple jobs work together, creates a pipeline and creating a pipeline is a manual task.
- Now we want to deal with both the job creation issue and the dependency issue to remove the manual part.
 - Example for dependency issue is, suppose there is a developer, and he writes a code for any app in java, now he wants to build the code and test the code. For that he has to depend on the other operations team who will create the job for building and testing the app.



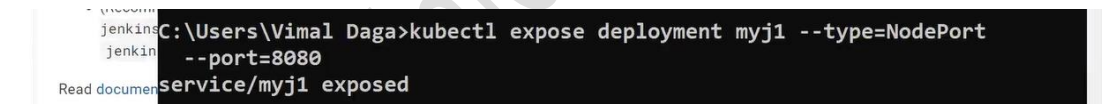
- Now for creating these two jobs, Jenkins guys will go the Jenkins UI and login to it and then will create the job.
 - This dependency issue will slow down all the processes.
- Rather than going to the Jenkins UI and creating the pipeline manually, we can use another method known as the **pipeline as a code**.
 - Pipeline as Code describes a set of features that allow Jenkins users to define pipelined job processes with code, stored and versioned in a source repository.
 - Launching the Jenkins on the Kubernetes.
 - For this you must have Kubernetes available in your system.
 - For launching the Jenkins pod we need a Jenkins image, and we can find it on the dockerhub.



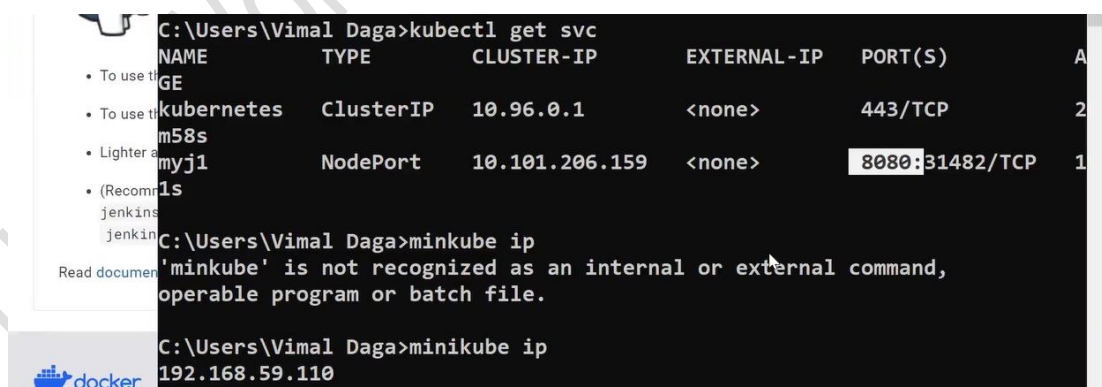
- Use the command ***kubectl create deployment myj1 --image=Jenkins/jenkins***. This will launch a pod with the Jenkins image.

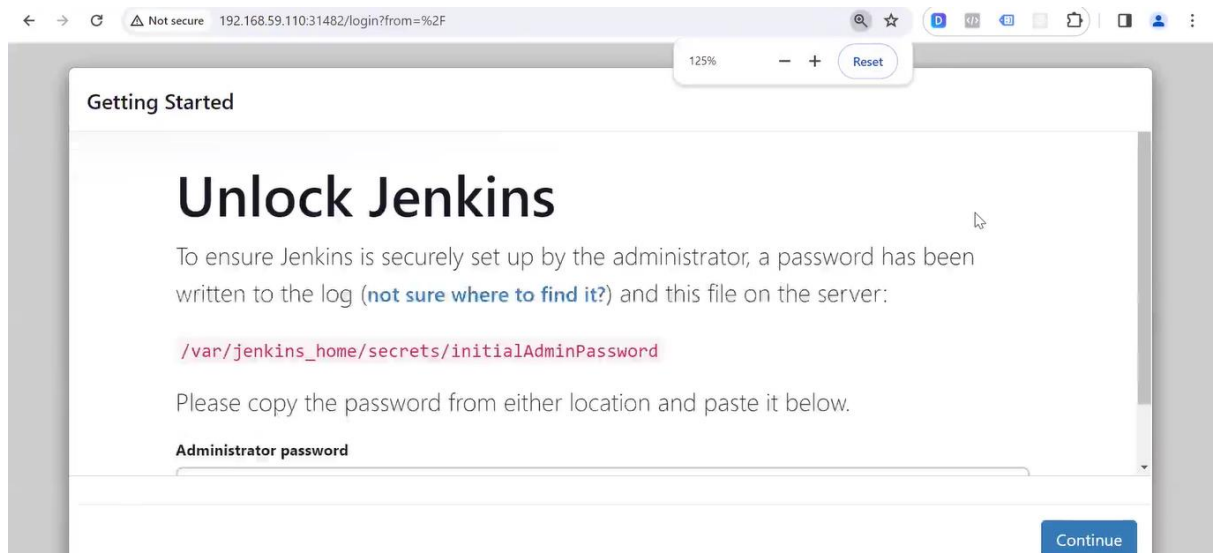


- Now we have to login to the jenkins but before that we have to expose the Jenkins pod using the command ***kubectl expose deployment myj1 --type=Nodeport --port=8080***.

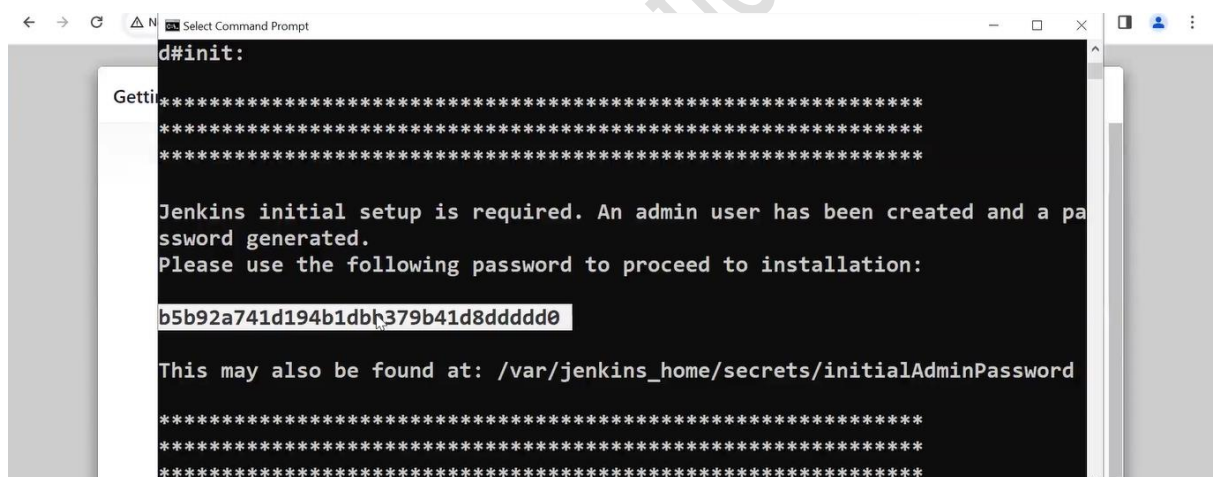


- Go to the minikube ip with the port number to login to the Jenkins.

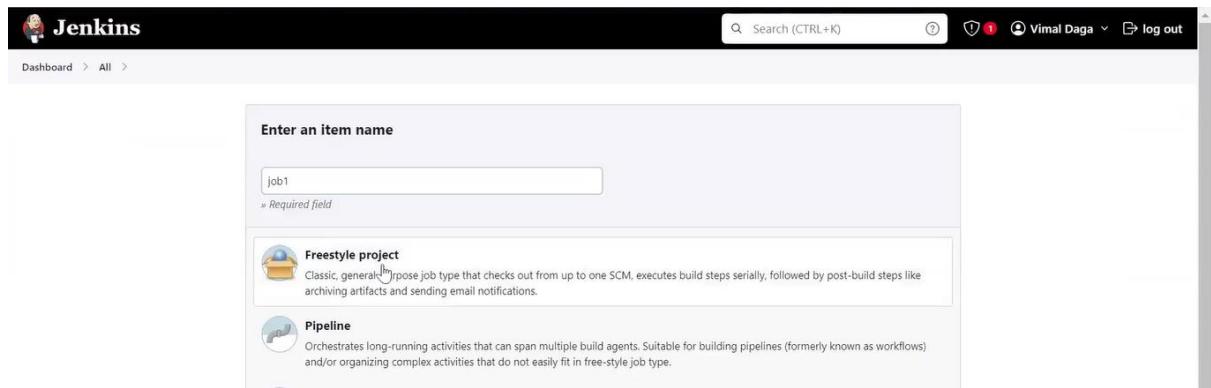




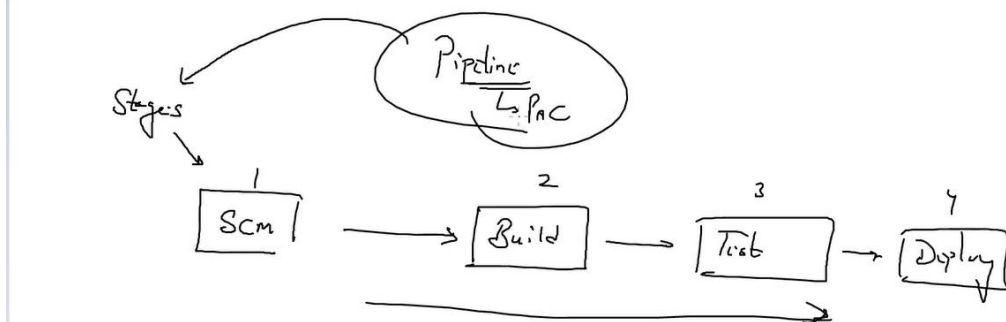
- We need the password to unlock the Jenkins for the first time, this password can be found in the logs using *kubectl logs <pod name>* or the file mentioned in the Jenkins.



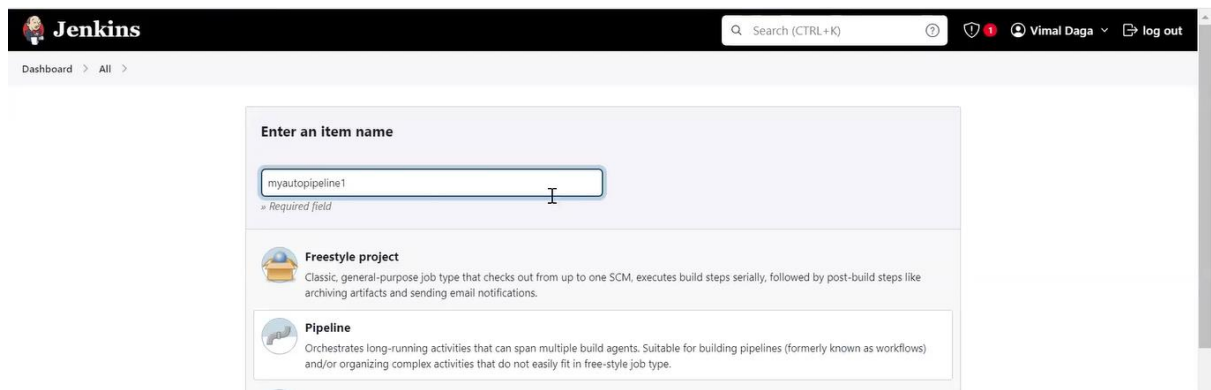
- Use this password and login to the Jenkins, also install the required plugins.
- Till now we are creating the freestyle jobs manually in the WebUI of the Jenkins.



- Suppose we want to create multiple jobs, one for pulling the code from the SCM, one for building the code, one for testing the code and one is for deploying the code.



- We will create all these jobs using the pipeline as code and in the PAC the jobs are known as the **stages**.
- There are two ways to create pipeline as a code.
 - Scripted way in which we use some scripting language like groovy.
 - Declarative way.
- For the declarative way, we must have the pipeline plugin installed in the Jenkins.
- Creating pipeline using the code.
 - Click on the pipeline option for creating the pipeline.



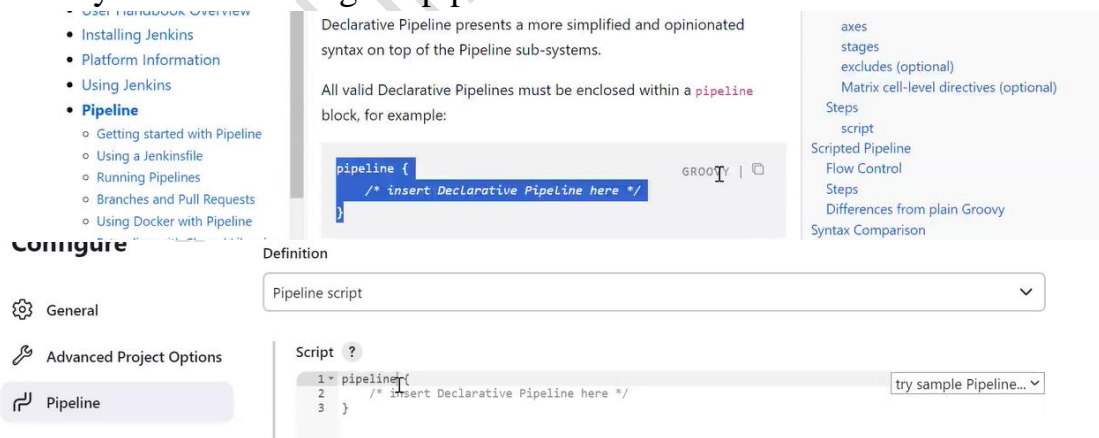
The screenshot shows the Jenkins 'Enter an item name' form. At the top, there's a search bar and user information. Below, a text input field contains 'myautopipeline1' with a 'Required field' label. Two options are listed: 'Freestyle project' (described as a classic job type) and 'Pipeline' (described as a tool for orchestrating long-running activities).

- Whatever we want to do here, we have to write the code for that or we have to declare it.



The screenshot shows the Jenkins Pipeline configuration page. On the left, a sidebar has 'General', 'Advanced Project Options', and 'Pipeline' tabs. The 'Pipeline' tab is active. The main area shows a 'Definition' dropdown set to 'Pipeline script'. Below it is a large 'Script' text area with a 'try sample Pipeline...' button. A 'Use Groovy Sandbox' checkbox is checked at the bottom.

- Typically, declarative pipelines contain one or more declarative steps or directives.
- First thing we want to do is to create a pipeline and here is the syntax for creating the pipeline.



This block contains two screenshots. The top one is a Jenkins documentation page for 'Declarative Pipeline' with a table of contents and an example code block:

```
pipeline {  
    /* insert Declarative Pipeline here */  
}
```

. The bottom screenshot shows the Jenkins configuration page with the 'Script' field containing the same example code.

- Now we have to write everything inside this pipeline.
- Inside the pipeline we have multiple **stages**.
- **Stages** block constitutes different executable stage blocks. At least one stage block is mandatory inside stages block.

Configure

General

Advanced Project Options

Pipeline

Definition

Pipeline script

Script ?

```
1 pipeline {
2
3   stages {
4
5     stage('SCM') {}
6     stage('Build') {}
7     stage('Test') {}
8     stage('Deploy') {}
9
10  }
11
12 }
13
14 }
```

try sample Pipeline...

- **Stage** block contains the actual execution steps. “Stage” block has to be defined within “Stages” block. It’s mandatory to have at least one stage block inside the stage block. Also its mandatory to name each stage block & this name will be shown in the Stage View after we run the job.

- If we want to create a job, we need an agent for that.
- **Agent** specifies where the Jenkins build job should run. Agent can be at pipeline level or stage level. It’s mandatory to define an agent.

Advanced Project Options

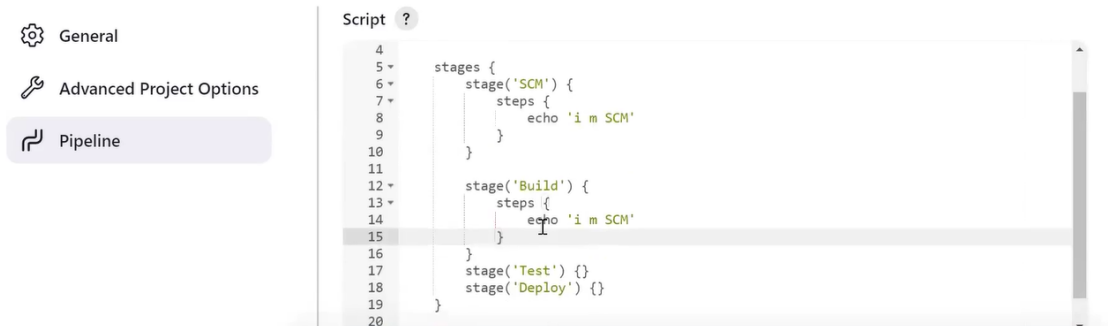
Pipeline

Script ?

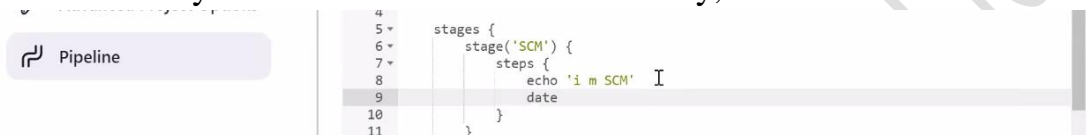
```
1 pipeline {
2
3   agent any
4
5   stages {
6     stage('SCM') {}
7     stage('Build') {}
8     stage('Test') {}
9     stage('Deploy') {}
10  }
11
12 }
13 }
```

- Possible values for agents are:-
 - **any** — Run Job or Stage on any available agent.
 - **none** — Don’t allocate any agent globally for the pipeline. Every stage should specify their own agent to run.
 - **label** — Run the job in agent which matches the label given here.
 - **none** — Don’t allocate any agent globally for the pipeline. Every stage should specify their own agent to run.

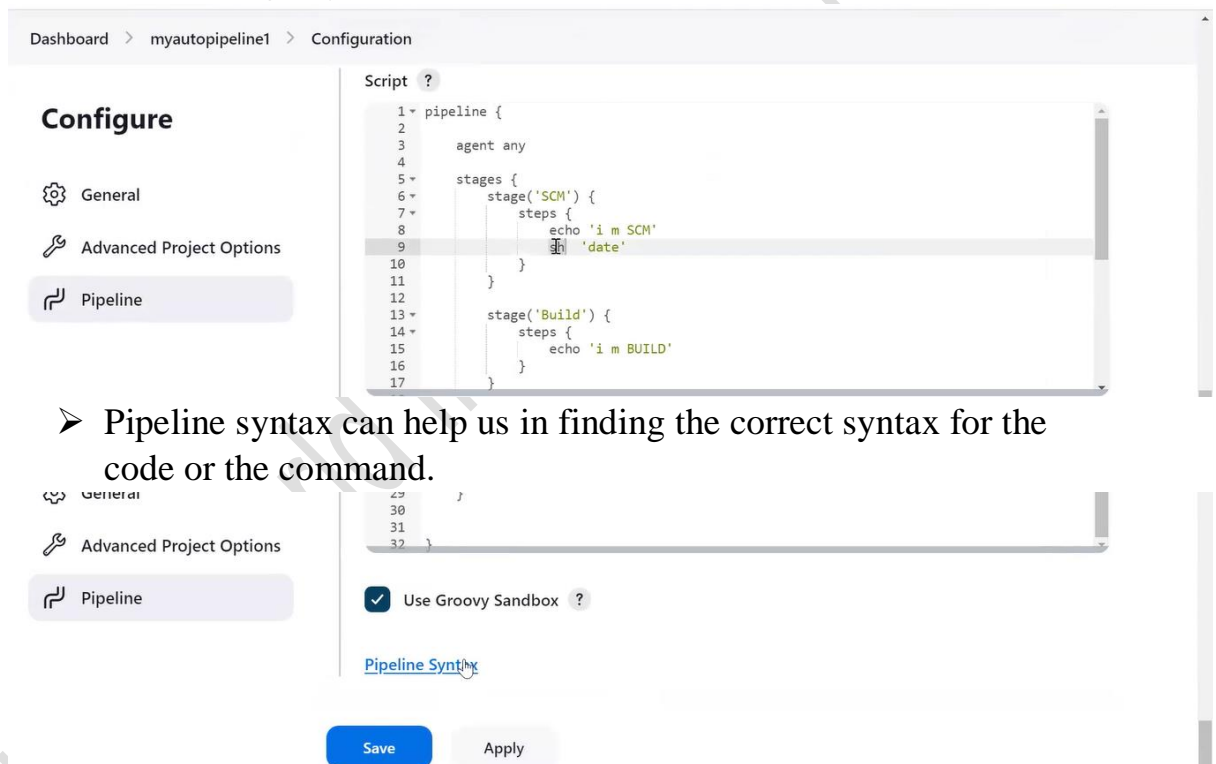
- **Steps** block contains the actual build step. It’s mandatory to have at least one step block inside a stage block.



- Here 'echo' is not acting as a command, it is a keyword in the PAC.
- If we try to run the date command directly, it will fail.



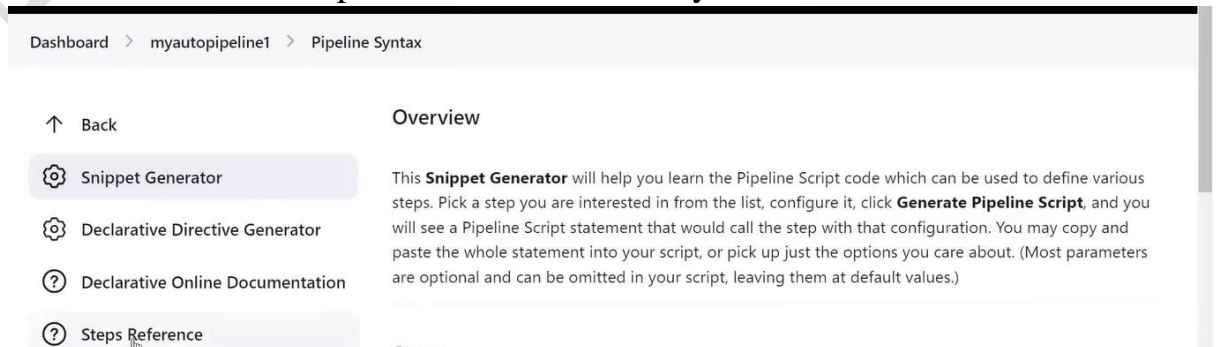
- For running any command in the PAC we have to use the **sh**.



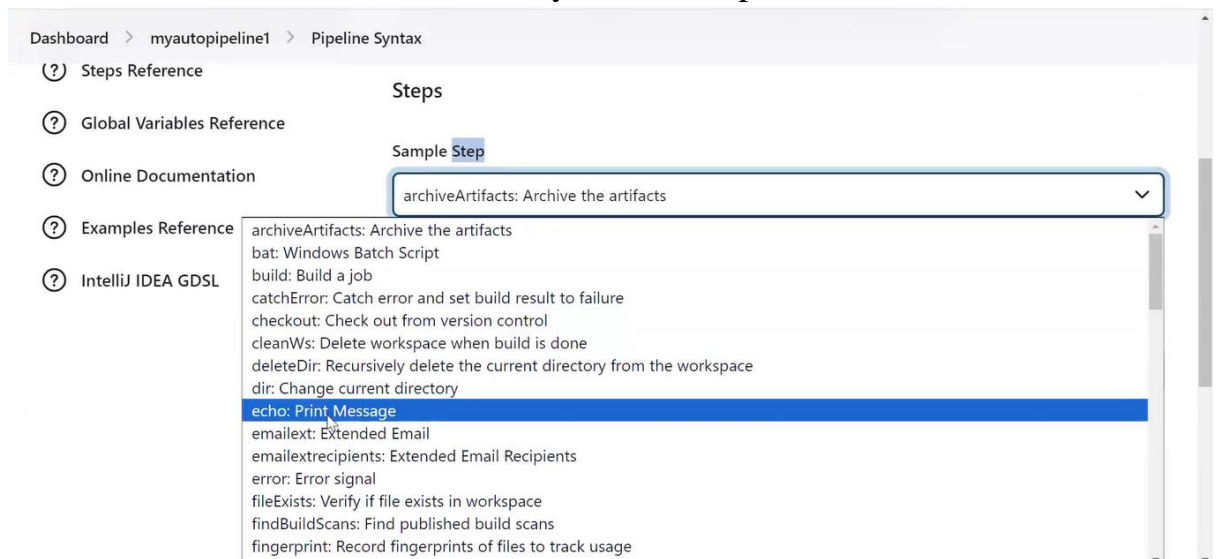
- Pipeline syntax can help us in finding the correct syntax for the code or the command.



- Click on the steps reference to see the syntax.

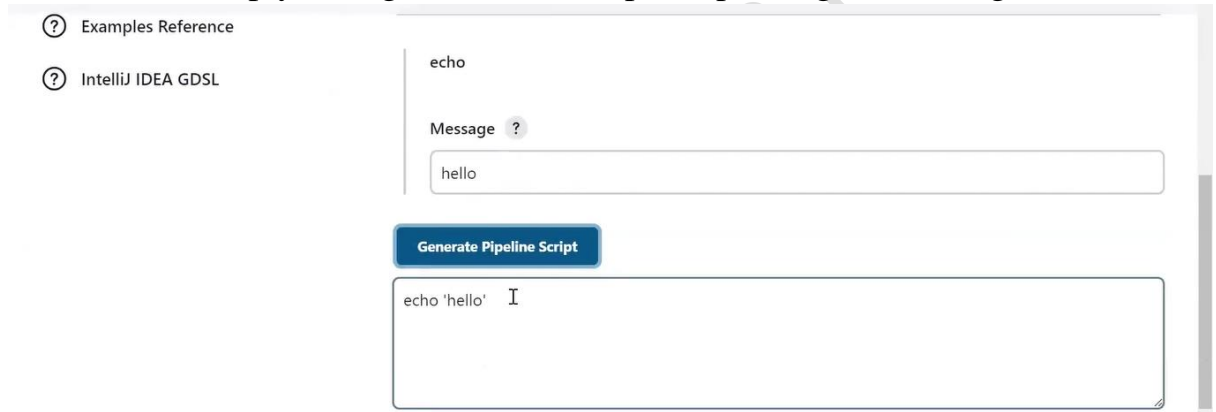


➤ Here we can see the commonly used examples.



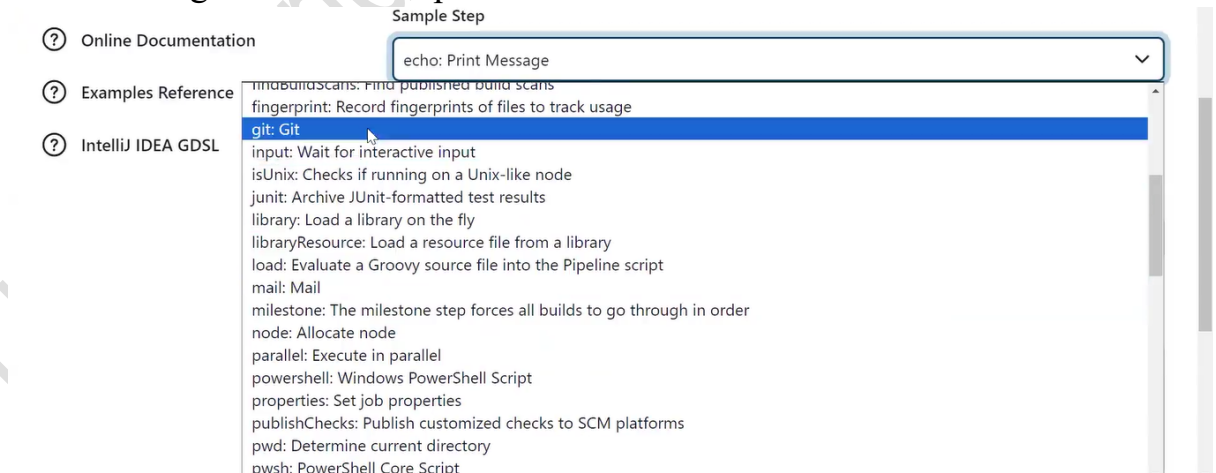
The screenshot shows the Jenkins Pipeline Syntax page. On the left, there is a sidebar with links: Steps Reference, Global Variables Reference, Online Documentation, Examples Reference, and IntelliJ IDEA GDSL. The main content area is titled 'Steps' and features a 'Sample Step' dropdown menu. The dropdown is open, showing a list of steps: archiveArtifacts: Archive the artifacts, bat: Windows Batch Script, build: Build a job, catchError: Catch error and set build result to failure, checkout: Check out from version control, cleanWs: Delete workspace when build is done, deleteDir: Recursively delete the current directory from the workspace, dir: Change current directory, echo: Print Message (highlighted), emailExt: Extended Email, emailExtrecipients: Extended Email Recipients, error: Error signal, fileExists: Verify if file exists in workspace, findBuildScans: Find published build scans, and fingerprint: Record fingerprints of files to track usage.

➤ It will help you to generate the script for printing the message.



The screenshot shows the Jenkins Pipeline Syntax page with the 'echo' step selected. The 'Message' field is set to 'hello'. Below the field is a 'Generate Pipeline Script' button. The generated script is displayed in a text area: `echo 'hello' I`.

➤ Let's generate the script for the Git.



The screenshot shows the Jenkins Pipeline Syntax page with the 'git: Git' step selected. The 'Sample Step' dropdown menu is open, showing a list of steps: archiveArtifacts: Archive the artifacts, bat: Windows Batch Script, build: Build a job, catchError: Catch error and set build result to failure, checkout: Check out from version control, cleanWs: Delete workspace when build is done, deleteDir: Recursively delete the current directory from the workspace, dir: Change current directory, echo: Print Message, emailExt: Extended Email, emailExtrecipients: Extended Email Recipients, error: Error signal, fileExists: Verify if file exists in workspace, findBuildScans: Find published build scans, fingerprint: Record fingerprints of files to track usage, git: Git (highlighted), input: Wait for interactive input, isUnix: Checks if running on a Unix-like node, junit: Archive JUnit-formatted test results, library: Load a library on the fly, libraryResource: Load a resource file from a library, load: Evaluate a Groovy source file into the Pipeline script, mail: Mail, milestone: The milestone step forces all builds to go through in order, node: Allocate node, parallel: Execute in parallel, powershell: Windows PowerShell Script, properties: Set job properties, publishChecks: Publish customized checks to SCM platforms, pwd: Determine current directory, and pwsh: PowerShell Core Script.

➤ Give the repository URL.

? Online Documentation
? Examples Reference
? IntelliJ IDEA GDSDL

git: Git

git ?

Repository URL ?

`https://github.com/vimallinuxworld13/jenkins_training_2024_docker.git`

! Please enter Git repository.

➤ Here is the script for using the git.

Dashboard > myautopipeline1 > Pipeline Syntax

Generate Pipeline Script

git 'https://github.com/vimallinuxworld13/jenkins_training_2024_docker.git'

Dashboard > myautopipeline1 > Configuration

Configure

General
Advanced Project Options
Pipeline

Script ?

```
2  
3 agent any  
4  
5 stages {  
6   stage('SCM') {  
7     steps {  
8       echo 'i m SCM'  
9       sh 'date'  
10      git 'https://github.com/vimallinuxworld13/jenkins_training_2024_docker.git'  
11     }  
12   }  
13 }
```

➤ Either we can write the script directly in the console or we can also create a Jenkinsfile and pull it from the SCM.

➤ We will put both the app code and the Jenkinsfile in the scm.

```
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/jenkins_training_2024/jenkins_PaC_training_2024 (master)  
$ ls  
README.md index.html  
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/jenkins_training_2024/jenkins_PaC_training_2024 (master)  
$ vim Jenkinsfile
```

```
Jenkinsfile  
stage('Build') {  
  steps {  
    echo 'i m BUILD'  
    sh 'date'  
  }  
}  
  
stage('Test') {  
  steps {  
    echo 'i m Test'  
  }  
}  
  
stage('Deploy') {  
  steps {  
    echo 'i m Deploy'  
  }  
}  
}
```

Jenkinsfile[+] [unix] (05:29 01/01/1970) 30,2 Bot
-- INSERT --

➤ Push this to the SCM.

```
JenkinVimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/jenkins_training_2024/jenkins_PaC
training_2024 (master)
$ git add .
warning: LF will be replaced by CRLF in Jenkinsfile.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/jenkins_training_2024/jenkins_PaC
training_2024 (master)
$ git commit -m "auto1"
warning: LF will be replaced by CRLF in Jenkinsfile.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory
[master 0beb209] auto1
2 files changed, 35 insertions(+)
create mode 100644 Jenkinsfile
create mode 100644 index.html
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/jenkins_training_2024/jenkins_PaC
training_2024 (master)
$ git push
```

- As the code is pushed to the github, copy the repository URL and paste it inside the Jenkins.

Dashboard > mypipe2 > Configuration

Configure

Definition: Pipeline script from SCM

SCM: Git

Repositories:

Repository URL:

Please enter Git repository.

Save Apply

- In the script path give the name of the Jenkinsfile which contain the pipeline code.

Pipeline

Script Path:

☒ Lightweight checkout

Pipeline Syntax

Save Apply

- We can also use the triggers here, so that as soon as any changes are made in the code, it will be pulled again automatically.
- Just like this we have multiple blocks to be used in the pipeline, we can check them from the documentation and can use them as per the requirements.

- We can understand PAC in more detail using this interesting project <https://github.com/vimallinuxworld13/jenkins-docker-maven-java-webapp>

➤ In this project we are building a pipeline and also using the agent in it.

```
pipeline {
  agent {
    label "linuxbuildnode"
  }

  stages {
    stage('SCM') {
      steps {
        git 'https://github.com/vimallinuxworld13/jenkins-docker-maven-java-webapp.git'
      }
    }

    stage('Build by Maven Package') {
      steps {
        sh 'mvn clean package'
      }
    }
  }
}
```

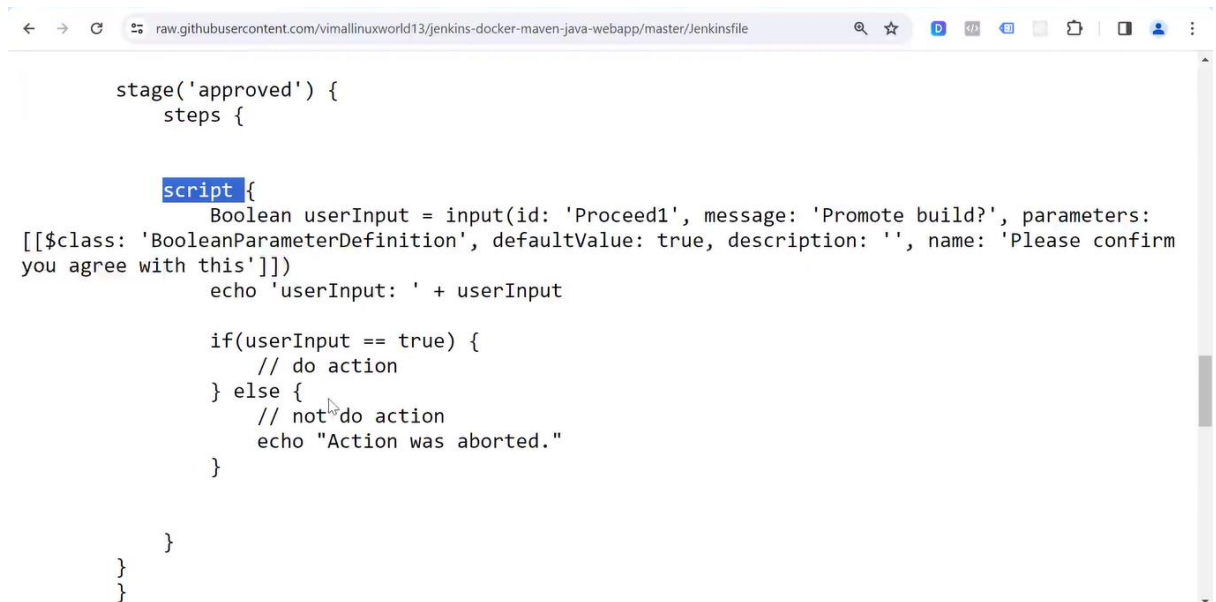
➤ We have multiple stages for pulling the code from the SCM, building the code, creating the docker image, pushing the image to the docker hub, deploying the app in the testing environment etc.

```

  stage('Build Docker OWN image') {
    steps {
      sh "sudo docker build -t vimal13/javaweb:${BUILD_TAG} ."
      //sh 'whoami'
    }
  }

  stage('Push Image to Docker HUB') {
    steps {
      withCredentials([stringCredentialsId: 'DOCKER_HUB_PWD', variable:
'DOCKER_HUB_PASS_CODE']) {
        // some block
        sh "sudo docker login -u vimal13 -p $DOCKER_HUB_PASS_CODE"

        sh "sudo docker push vimal13/javaweb:${BUILD_TAG}"
      }
    }
  }
}
```

A screenshot of a web browser displaying a Jenkinsfile code snippet. The browser's address bar shows the URL: raw.githubusercontent.com/vimallinuxworld13/jenkins-docker-maven-java-webapp/master/Jenkinsfile. The code is a Groovy script for a Jenkins pipeline. It defines a stage named 'approved' with a single step named 'script'. Inside the 'script' step, it prompts the user for confirmation to promote a build. If the user agrees (userInput == true), it proceeds with the action; otherwise, it echoes a message that the action was aborted. The code is as follows:

```
stage('approved') {
    steps {
        script {
            Boolean userInput = input(id: 'Proceed1', message: 'Promote build?', parameters:
[[${class: 'BooleanParameterDefinition', defaultValue: true, description: '', name: 'Please confirm
you agree with this'}]])
            echo 'userInput: ' + userInput

            if(userInput == true) {
                // do action
            } else {
                // not do action
                echo "Action was aborted."
            }
        }
    }
}
```

- This is the perfect real world example of Jenkins Pipeline as a code.