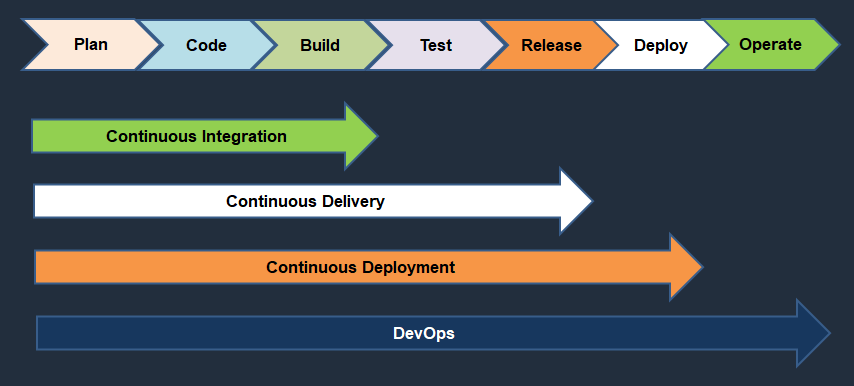
1. **What is Continuous Integration with reference to Jenkins?**

**Continuous Integration:**

* All Developers commit their source code changes to the shared Git repository
* Jenkins server checks the shared Git repository at specified intervals and detected changes are taken into the build.
* The build results and test results are shared to the developers
* The build application is displayed on test servers like Junit/Selenium and automated tests are run.
* The clean and tested build is deployed to the production server.
* The commit, build,test, and deploy is a continuous process.

1. **What are the difference between Continuous Integration, continuous Delivery and Continuous Deployment.**



**Continuous Integration:**

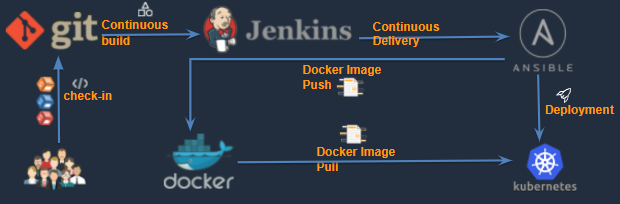
* Developers will push their code several times in a day to a central repository, every time there is code change it should be pulled, built, tested and notified.

**Continuous Delivery**

* The process of CI, the artifact has been generated.
* The artifact should be delivering to all servers in different environments like Dev, QA, Staging.
* It should be automatically delivered to QA servers where testers will do functional tests, load tests etc.
* After it passes the QA tests it should automatically deliver the code to staging area where customer or client or users check the changes and approval to deploy it to production.

**Continuous Deployment**

* If the approval is manual then code delivery is continuous Delivery but if the approval process becomes automated then after staging, the code change is done directly to Production systems.
* This is called as Continuous Deployment.



1. **What is workspace location in Jenkins**

* The workspace directory is where Jenkins builds your project.
* It contains the source code Jenkins checks out, plus any files generated by the build itself.
* This workspace is reused for each successive build.
* Default Workspace is JENKINS\_HOME =/var/lib/jenkins

1. **What are the few major plugins using in your project and explain each every plug-in use case**

Some of plugins are used are here

* + Green Balls
  + Git
  + Maven
  + pipeline
  + publish over ssh
  + ssh plugin
  + copy artifacts
  + Blue Ocean
  + Sonarqube scanner
  + artifactory
  + Monitoring
  + Docker
  + Kubernetes
  + ThinBackup

1. **Name the two components that Jenkins is mostly integrated with**

* Version Control System
* Build Tool (Maven)

1. **What is a multi branch pipeline?**

* The **Multibranch Pipeline** project type enables you to implement different Jenkinsfiles for different branches of the same project.
* In a Multibranch Pipeline project, Jenkins automatically discovers, manages and executes Pipelines for branches which contain a Jenkinsfile in source control.

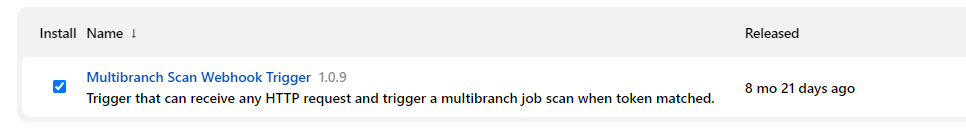
**Jenkinsfile:** <https://gitlab.com/arsravis/taxigrabber/-/blob/main/Jenkinsfile>

Refer link: https://www.youtube.com/watch?v=fo36b23cpIU

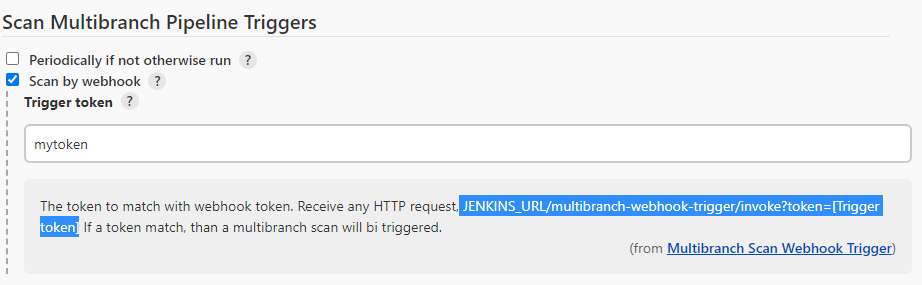
1. **How to enable webhook for Jenkins in Git.**

**Configure Web hook:**

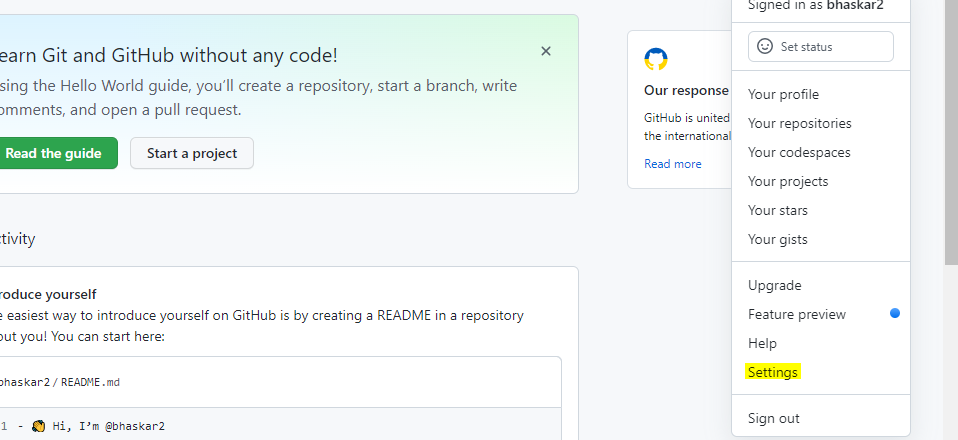
* We have to configure our Jenkins machine to communicate with our GitHub repository for that we need Hook URL of the Machine.
* Go to Mange Jenkins and Install Multibranch Scan Webhook Trigger plugin



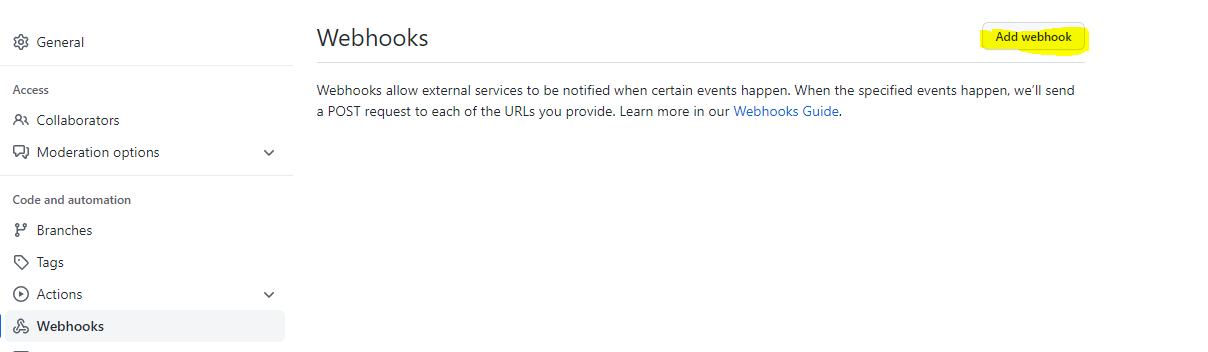
* Take the token from scan Multibranch Pipeline Triggers



* Go to GitHub and select setting



* Select the projects 🡪 settings-- > then Webhooks –Add Webhooks



1. **What is Jenkins' shared library ?**

* As an organization starts using more and more pipeline jobs, there is a chance for more and more code being duplicated in every pipeline job, since a part of the build/automation processes will be the same for most of the jobs. In such a situation, every other new upcoming job should also duplicate the same piece of code.
* To avoid duplications, the Jenkins project brought in the concept of Shared Libraries, to code - DRY - Don't Repeat Yourself.
* Shared libraries are a set of code that can be common for more than one pipeline job and can be maintained separately. Such libraries improve the maintenance, modularity & readability of the pipeline code. And it also speeds up the automation for new jobs.

1. **How to create & use a Shared Library in Jenkins?**

Basic requirements for a Jenkins shared library to be used in a Pipeline Code are -

A Repository with pipeline shared library code in SCM.

An appropriate SCM Plugin configuration for the Jenkins instance.

Global Shared Library should be configured in Jenkins Global configuration.

Include the Shared Library in the Pipeline Code and use the methods defined in the Jenkins Shared Library.

E.g.

#!/urs/bin/env groovy

@Library('fs\_jenkins\_shared\_library@v2.0.7')\_

1. **How to setup Jenkins slave system.**

**What this in YouTube: https://www.youtube.com/watch?v=9RsmPNs7gT0**

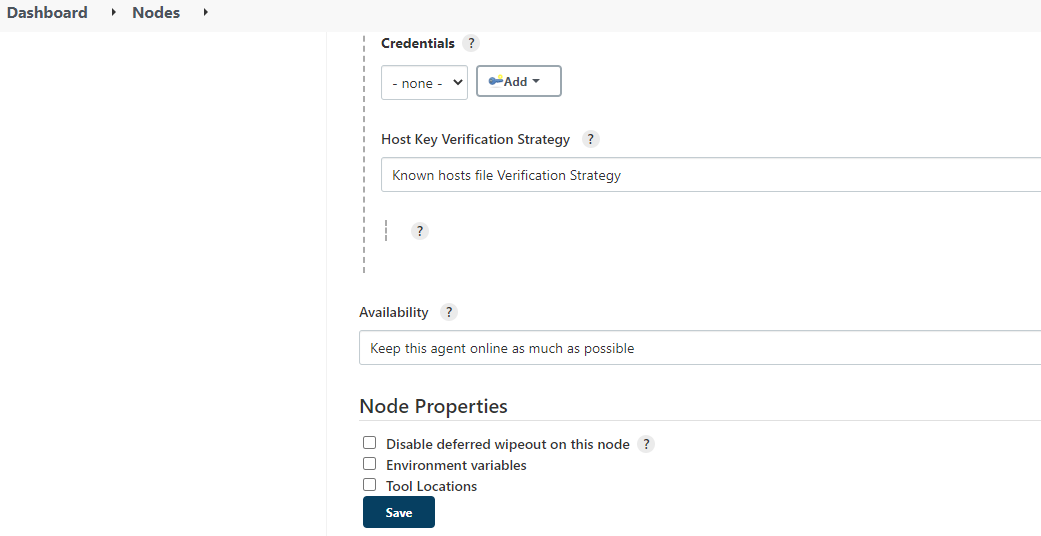
The Jenkins master acts to schedule the jobs and assign slaves and send builds to slaves to execute the jobs**.**

**Configure Jenkins master and slave nodes**

1. Click on **Manage Jenkins** in the left corner on the Jenkins dashboard
2. Click on Manage Nodes
3. Select New Node and enter the name of the node in the Node Name field
4. Select Permanent Agent and click on the OK button.

Initially you will get only one option, “**Permanent Agent**”. Once you have one or more slaves you will get the copy existing node Option.

1. Enter required information
2. Enter the host name in the Host field.
3. Select the Add button to add credentials. And click Jenkins
4. Enter username, password, ID, and Description
5. Select the dropdown menu to add credentials in the Credential field.
6. Select the next dropdown to add the Host key verification Strategy under None verifying verification Strategy.
7. Keep this agent online as much as possible
8. Click the save button



1. **What is Jenkins Pipeline?**

* Instead of creating Jenkins jobs from the user interface you would write a file or script to create a job.
* So it’s a “scripted file”
* Jenkins Pipeline is a suite of plugins that supports implementing and integrating continuous delivery pipelines into Jenkins.
* Document : https://www.jenkins.io/doc/book/pipeline/
* Creating a Jenkinsfile and committing it to source control provides a number of immediate benefits:
* Automatically creates a Pipeline build process for all branches and pull requests.
* Code review/iteration on the Pipeline (along with the remaining source code).
* Audit trail for the Pipeline.

**Pipeline Types:**

* Scripted Pipeline
* Declarative Pipeline (new feature)
* **Declarative Pipeline** is a more recent feature of Jenkins Pipeline which: provides richer syntactical features over Scripted Pipeline syntax, and is designed to make writing and reading Pipeline code easier.

1. **How to identify a pipeline is declarative or scripted?**
2. **git Pull vs Git fetch**

|  |  |
| --- | --- |
| **Git pull** | **Git Fetch** |
| This command pulls new changes from the currently working branch located in the remote central repository. | This command is also used for a similar purpose but it follows a two step process:  1. Pulls all commits and changes from desired branch and stores them in a new branch of the local repository.  current  2. For changes to be reflected in the current / target branch, git fetch should be followed by git merge command. |

1. **How to enable versioning for the artifacts in Jenkins. (BUILD\_ID)**

**Why Versioning:**

* Every application needs to be documented well including what version you deploy.
* Versions of the application will help which version (code) deploy to the development environment, which version (code) deploy to the production environment.
* Jenkins Pipeline exposes environment variables via the global variable env, which is available from anywhere within a Jenkinsfile.
* Here the example list of environment variables accessible from within Jenkins Pipeline is documented at ${YOUR\_JENKINS\_URL}/pipeline-syntax/globals#env

**Here the below few example :**

* **BUILD\_ID**

The current build ID, identical to BUILD\_NUMBER for builds created in Jenkins versions 1.597+

* **BUILD\_NUMBER**

The current build number, such as "153"

* **BUILD\_TAG**

String of jenkins-${JOB\_NAME}-${BUILD\_NUMBER}. Convenient to put into a resource file, a jar file, etc for easier identification

* **BUILD\_URL**

The URL where the results of this build can be found (for example http://buildserver/jenkins/job/MyJobName/17/ )

* **EXECUTOR\_NUMBER**

The unique number that identifies the current executor (among executors of the same machine) performing this build. This is the number you see in the "build executor status", except that the number starts from 0, not 1

* **JAVA\_HOME**

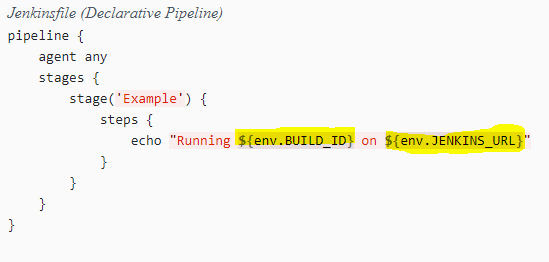
If your job is configured to use a specific JDK, this variable is set to the JAVA\_HOME of the specified JDK. When this variable is set, PATH is also updated to include the bin subdirectory of JAVA\_HOME

* **JENKINS\_URL**

Full URL of Jenkins, such as https://example.com:port/jenkins/ (NOTE: only available if Jenkins URL set in "System Configuration")

* **JOB\_NAME**

Name of the project of this build, such as "foo" or "foo/bar".



1. **How can you update artifacts on to Artifactory?**
2. **What is your approach when Jenkins job fails?**

**Here are few ways to handle the Jenkins job fails:**

* Enable auto trigger emails for failed job to the normal user(Developer)
* When Jenkins job failed, the email notification to send to Developer Group
* The DevOps team will check and validate the tool’s related issue or not. Such as when an agent is running , credential added is right etc.
* The DevOps team found the issue and resolved it ,if it is from the DevOps team.
* Sometimes Internet is also reason for failed Jenkins job

**Note:** ***Check the video to understand the reason better***

1. **How do you store credentials in Jenkins securely?**

A systems administrator of such an application can configure credentials in the application for dedicated use by jenkins.

Jenkins can store the following types of credentials:

* **Secret text**: A token such as an API Token (e.g. GitHub personal access token)
* **Username and Password**- which could be handled as separate components or as a colon separated string in the format username:password
* **Secret file** - which is essentially secret content in a file
* **SSH Username with private key** - an [SSH public/private key pair](http://www.snailbook.com/protocols.html),
* **Certificate** - a [PKCS#12 certificate file](https://tools.ietf.org/html/rfc7292) and optional password, or
* **Docker Host Certificate Authentication** credentials.

1. **What is protected branch**

**Protected branches:**

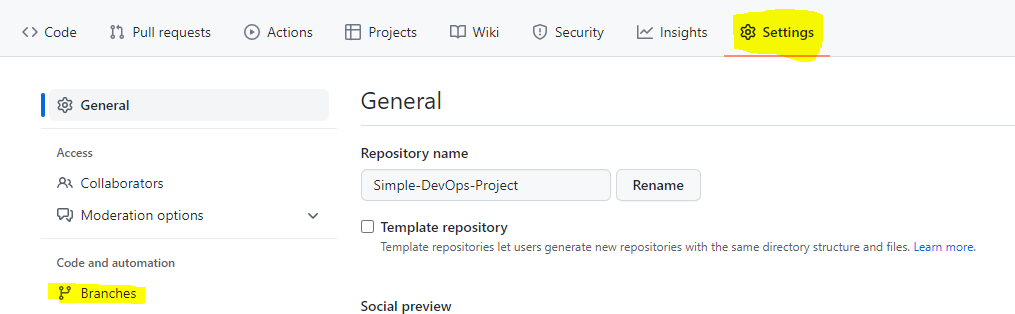
* In GitHub, permissions are fundamentally defined around the idea of having read or write permissions to repositories and branches.
* To impose further restrictions on certain branches, they can be protected.
* Here we can write a rule to protect the branch based on the rule.

**Configure a protected branch:**

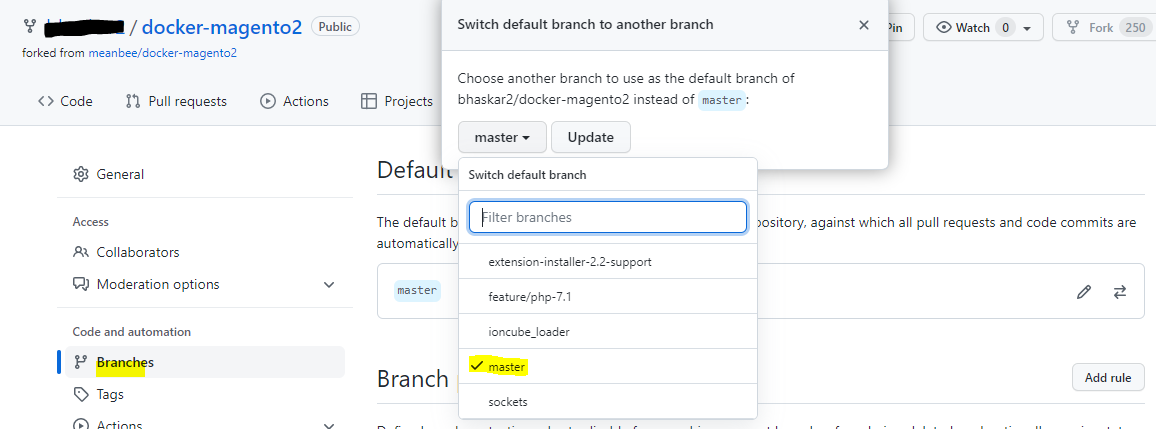
You must have an admin role.

To protect a branch:

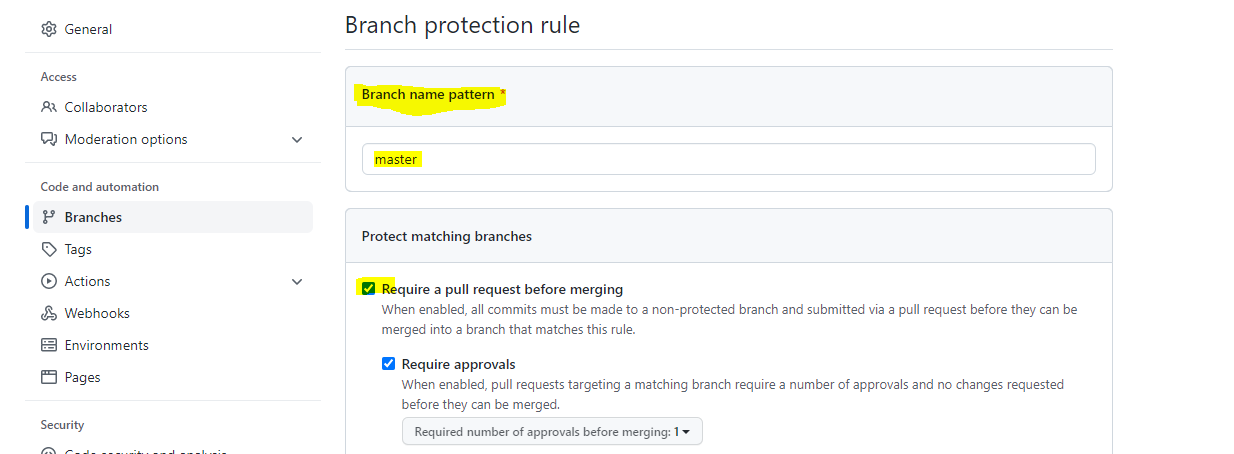
1. Go to your repository and select Settings → Branches

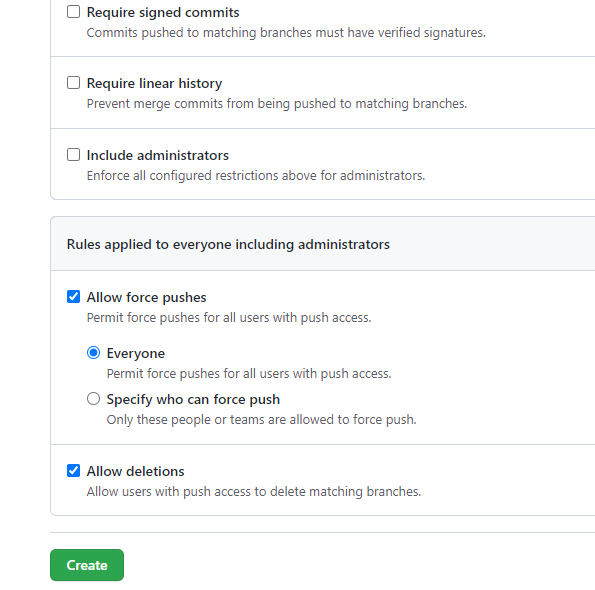


1. Expand Protected branches



1. From the Branch dropdown list, select the branch you want to protect.
2. Select “**Add rule**” and Branch protection rule is open
3. Mention branch name which you want to change





1. After set the protection rule, click the create bottom
2. Now your branch got protected
3. **What is a pull request? How the approval process goes.**

**Pull Request:**

* Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub.
* Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.
* Pull requests are proposed changes to a repository submitted by a user and accepted or rejected by a repository's collaborators. Like issues, pull requests each have their own discussion forum.

1. **Explain types of testing done in your pipeline**

* After code build, Run the tests(continuous integration server to generate builds and arrange releases).
* Test individual models, run integration test, and run user acceptance tests
* Basically unit test cases happened i.n. Junit or Selenium test.

1. **What are local, virtual and remote artifact repositories ?**

**Local repositories:**

* Local repositories are physical, locally-managed repositories into which you can deploy artifacts.
* Artifact in a local repository can be accessed directly using the following URL:

http://<host>:<port>/artifactory/<local-repository-name>/<artifact-path>

**Remote repositories:**

* A remote repository serves as a caching proxy for a repository managed at a remote URL (which may itself be another Artifactory remote repository).
* Artifacts are stored and updated in remote repositories according to various configuration parameters that control the caching and proxing behavior.
* You can remove artifacts from a remote repository cache but you can’t actually deploy a new artifact into a remote repository.

**Virtual repositories:**

* A virtual repository (or “repository group”) aggregates several repositories with the same package type under a common URL.

1. **Can you walk through the Jenkins pipeline which you have written in your environment?**
2. **How frequently do you run your Jenkins jobs**
3. **What are maven goals are you using in your project**
4. **What is the source code you are using in your project**
5. **What is the code coverage what is the code coverage percentage do you use**
6. **What are the quality gateways you use?**
7. **I want to create a dependent Jenkins job. It should run when the previous job is successful**
8. **Cron jobs importance?**
9. **Have you worked on Jenknsfile? can we use docker container as a node in Jenkinsfile? Who will handle docker container creation and deletion? If i am building a maven project always docker container is fresh instance it will try to download dependency from repository, what measures you will take to reduce build time?**

**->pipeline{**

**agent {**

**docker {**

**image 'maven'**

**args '-v /root/.m2:/root/.m2'**

**}**

**}**

**parameters {**

**choice(name:'mvnaction',**

**choices: 'Clean\nCompile\nTest\nInstall',**

**description: 'based on selection jenkins will run resepective maven command')**

**}**

**stages{**

**stage('build')**

**{**

**steps{**

**script{**

**sh "mvn --version"**

**sh "echo $mvnaction"**

**if ("${mvnaction}" == "Clean")**

**{**

**sh "mvn clean"**

**}**

**else if ("${mvnaction}" == "Compile")**

**{**

**sh "mvn clean compile"**

**}**

**else if ("${mvnaction}" == "Test")**

**{**

**sh "mvn clean test"**

**}**

**else if ("${mvnaction}" == "Install")**

**{**

**sh "mvn clean install"**

**}**

**}**

**}**

**}**

**}**

**}**

**Q.If you forgot Jenkins pwd ,how would you be login back?**

**->go to .jenkins folder->config.xml-><usesecurity>false</usesecurity>**

**Q.you want to create 50 freestyle jobs with same configurations, but only change is job name. how would you achieve the same?**

**Ans:import hudson.plugins.git.\*;**

**def scm = new GitSCM("git@github.com:dermeister0/Tests.git")**

**scm.branches = [new BranchSpec("\*/develop")];**

**def flowDefinition = new org.jenkinsci.plugins.workflow.cps.CpsScmFlowDefinition(scm, "Jenkinsfile")**

**def parent = Jenkins.instance**

**def job = new org.jenkinsci.plugins.workflow.job.WorkflowJob(parent, "New Job")**

**job.definition = flowDefinition**

**parent.reload()**

**Q.How can you copy job from your local jenkins instance to other local jenkins instance?**

**->user->.jenkins->.jobs->config.xml ->copy to machine->login on jenkins UI->manager jenkins ->tools and actions ->reload configuration from disk->new job will be added ->go to job configuration->**

**set github scm option**

**Q. How do you take Jenkins backup?**

**->install thinbackup plugin and do the settings. or don't want to use plugin then use shell script**

**Q.what is seed jobs in Jenkins?**

**ANS:-** **The seed job is a normal Jenkins job that runs the Job DSL script; in turn, the script contains instructions that create additional jobs. In short, the seed job is a job that creates more jobs. In this step,**

**you will construct a Job DSL script and incorporate it into a seed job.**

**Q.What is upstream and downstream jobs in Jenkins ?**

**ANS:** **An upstream project is one in which a job is triggered before the actual project is triggered. Whereas, Downstream project is one in which a job is triggered after the project has been triggered.**