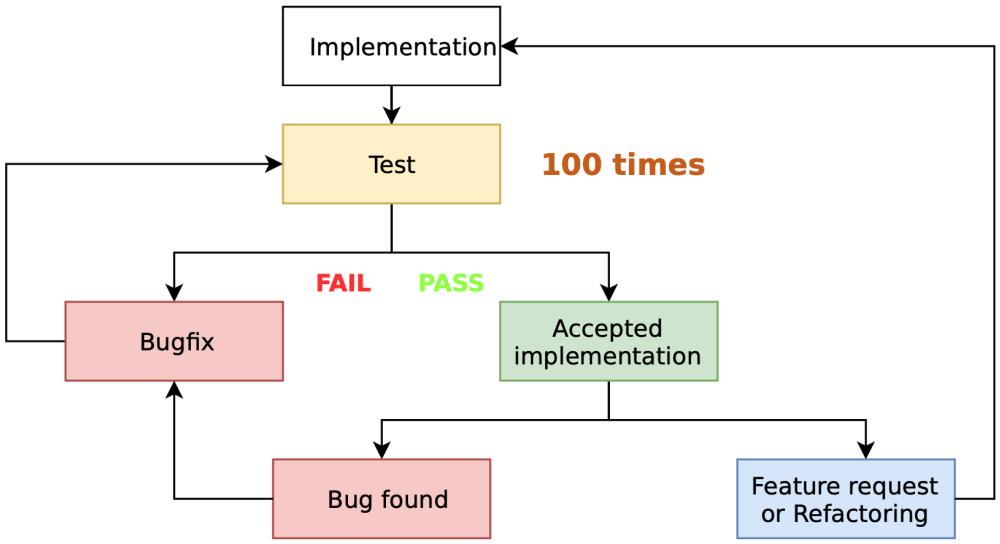
Life cycle of a function



Checking for None values

Do this for checking if `var` is `None`.

```
assert var is None
```

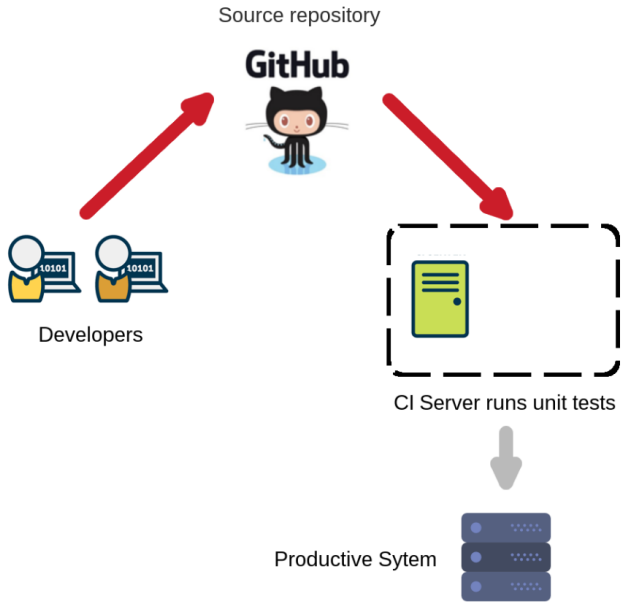
Do *not* do this.

```
assert var == None
```

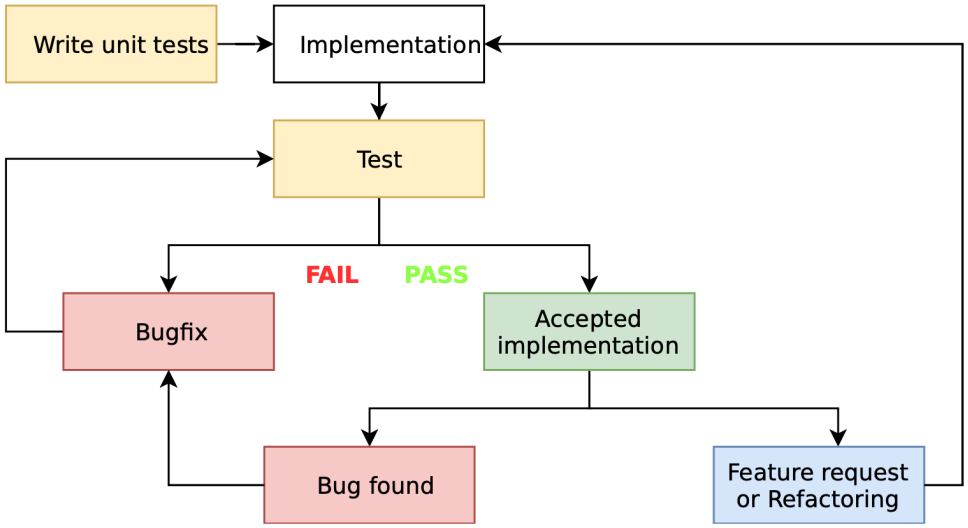
Section 2: Test result

```
collecting ...
collected 3 items

test_row_to_list.py .F.                                  [100%]
```

Character	Meaning	When	Action
F	Failure	An exception is raised when running unit test.	Fix the function or unit test.
.	Passed	No exception raised when running unit test	Everything is fine. Be happy!



Test Driven Development (TDD)

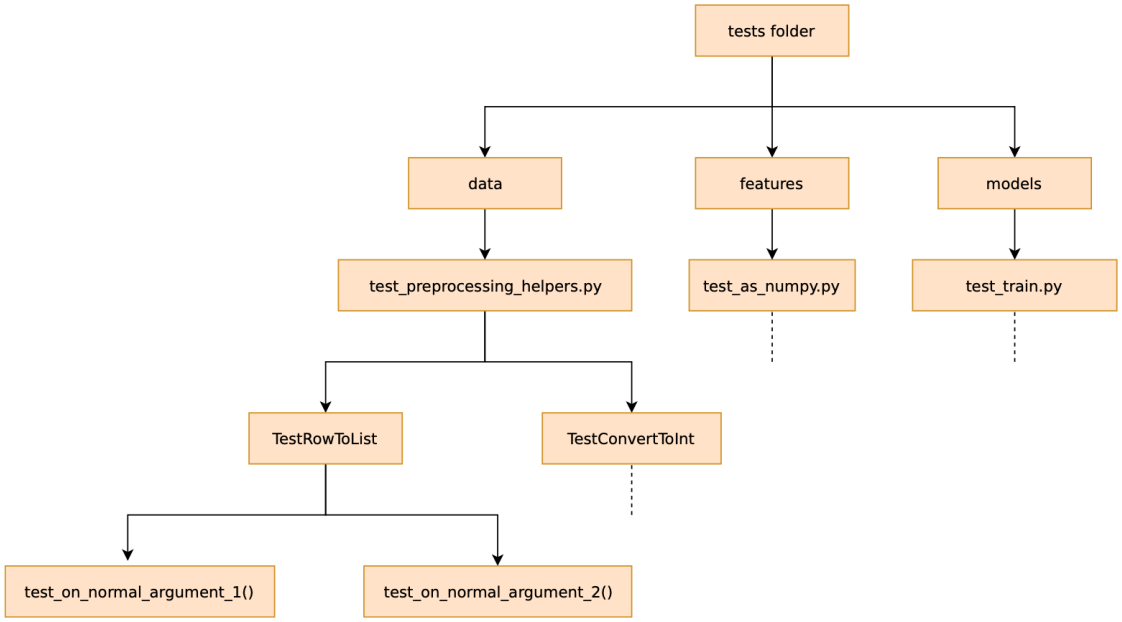


TDD
1 => create the test file
2 => run the test and fail
3 => write the file and then test to make sure it passes
4 => repeat process!

Python module and test module correspondence

- `my_module.py` ⟺ `test_my_module.py`.

```
src/                                # All application code lives here
|-- data/                           # Package for data preprocessing
|   |-- __init__.py
|   |-- preprocessing_helpers.py    # Contains row_to_list(), convert_to_int()
|-- features/                       # Package for feature generation from preprocessed data
|   |-- __init__.py
|   |-- as_numpy.py                 # Contains get_data_as_numpy_array()
|-- models/                         # Package for training/testing linear regression model
|   |-- __init__.py
|   |-- train.py                    # Contains split_into_training_and_testing_sets()
tests/                              # Test suite: all tests live here
|-- data/
|   |-- __init__.py
|   |-- test_preprocessing_helpers.py   # Corresponds to module src/data/preprocessing_helpers.py
|-- features/
|   |-- __init__.py
|-- models/
    |-- __init__.py
```

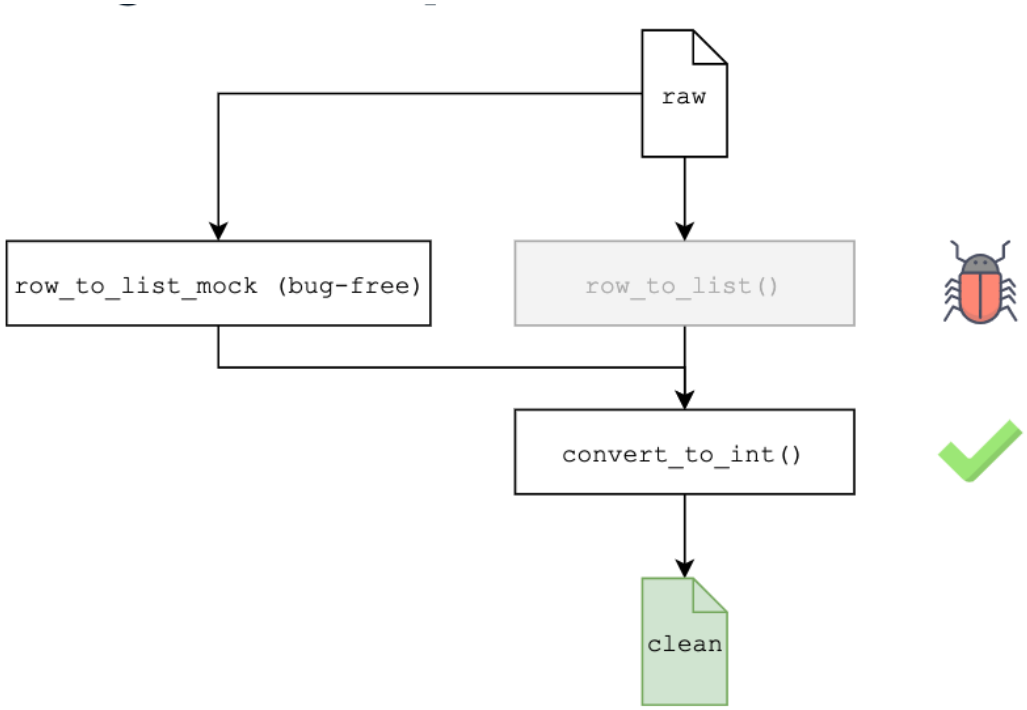


Step 1: Modify the Travis CI configuration file

- File: `.travis.yml`

```
language: python
python:
  - "3.6"
install:
  - pip install -e .
  - pip install pytest-cov codecov    # Install packages for code coverage report
script:
  - pytest --cov=src tests            # Point to the source directory
after_success:
  - codecov                           # uploads report to codecov.io
```

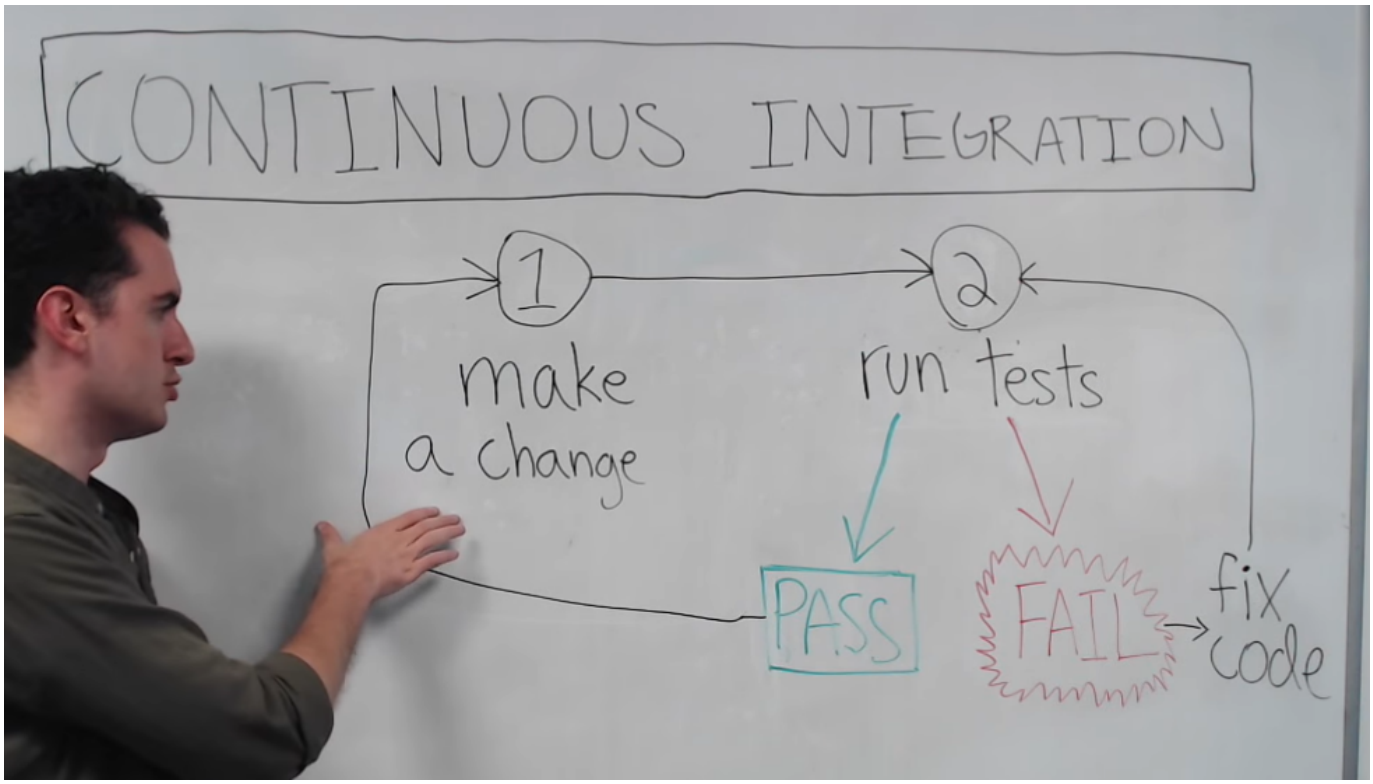
How mocking tests work:

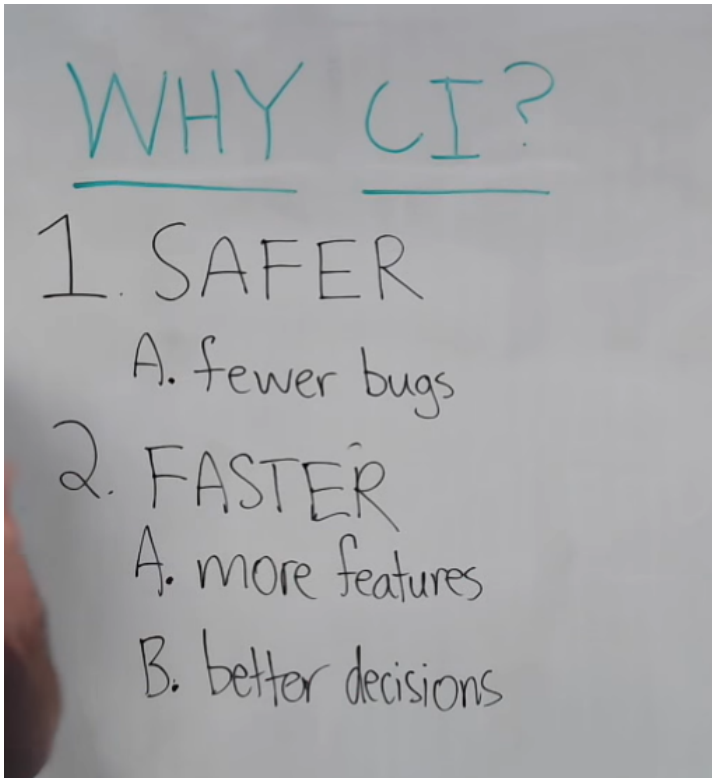


Nowadays we are using Agile Methodology, you have to showcase product to the client every two to three weeks, usually one feature of the program to be showcased

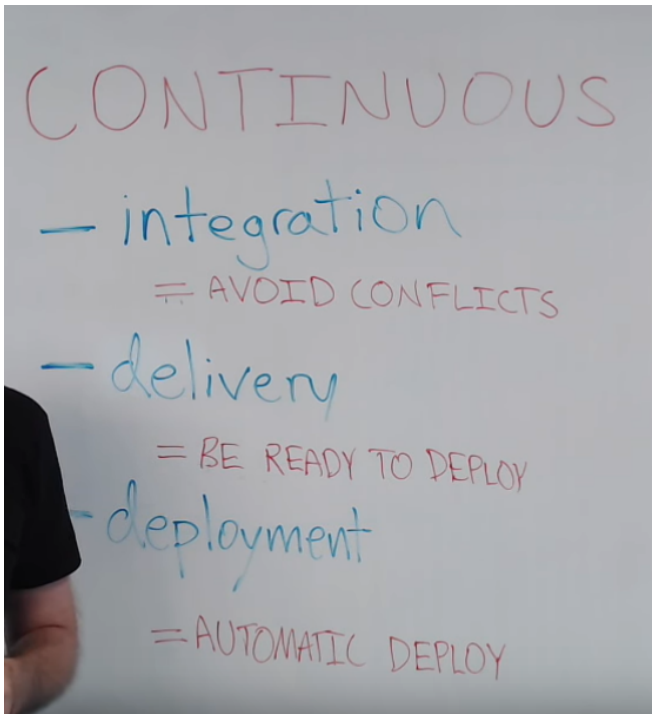
Continuous Integration : After every commit of the project there will be an automated test to make sure it works

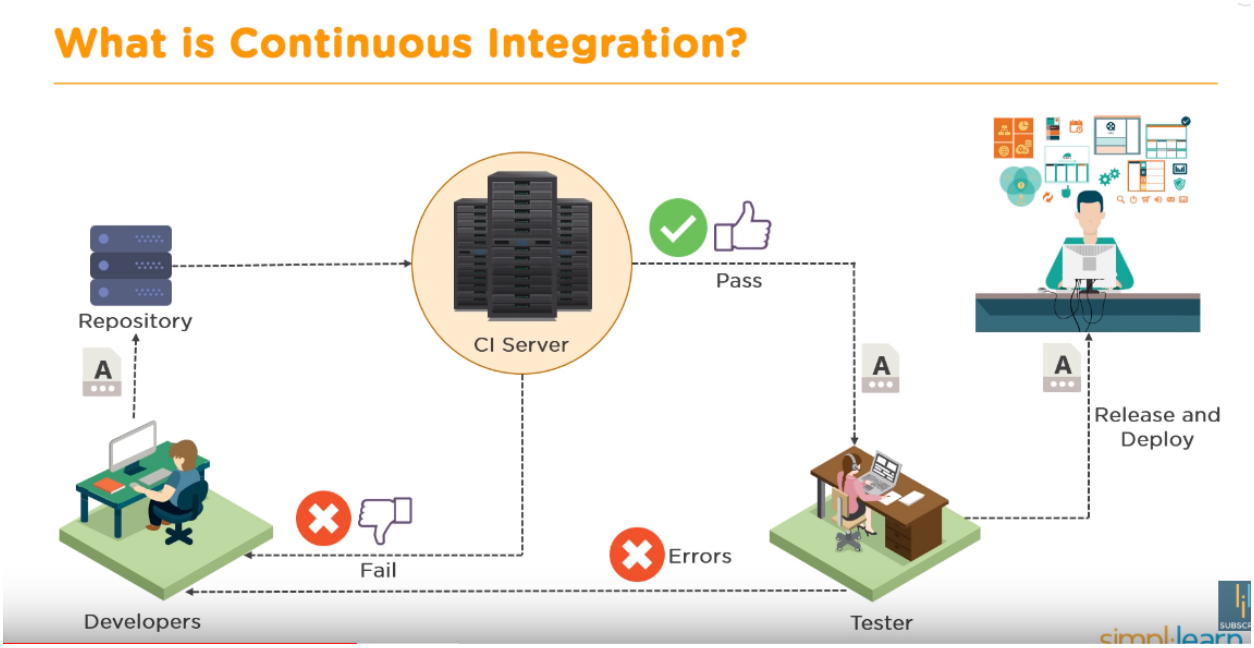
Tests should run automatically without the developer applying them manually.
-After tests are run if there is no error you can try to add new code and again commit them and run this same testing process
-However if the code fails the test then we have to fix it and run it again until it passes our test.
** this also makes it clear why we use TDD or Test Driven Development

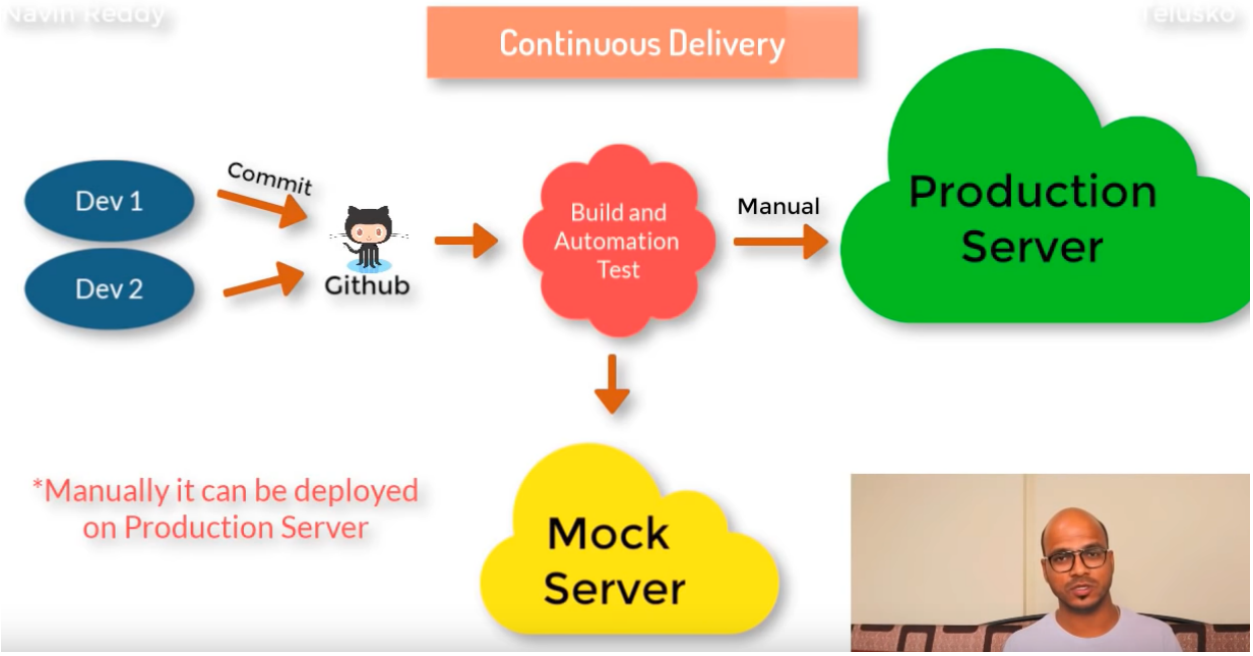




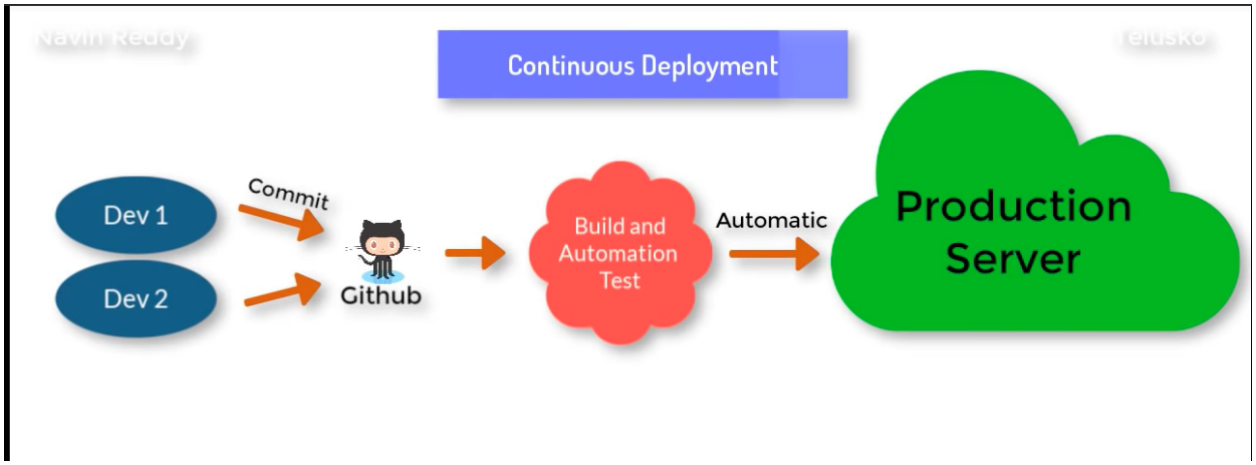
Continuous Delivery: load the project on a mock server to make sure that it works properly
Basically we want to make sure that the code is always ready to be deployed on a production server.



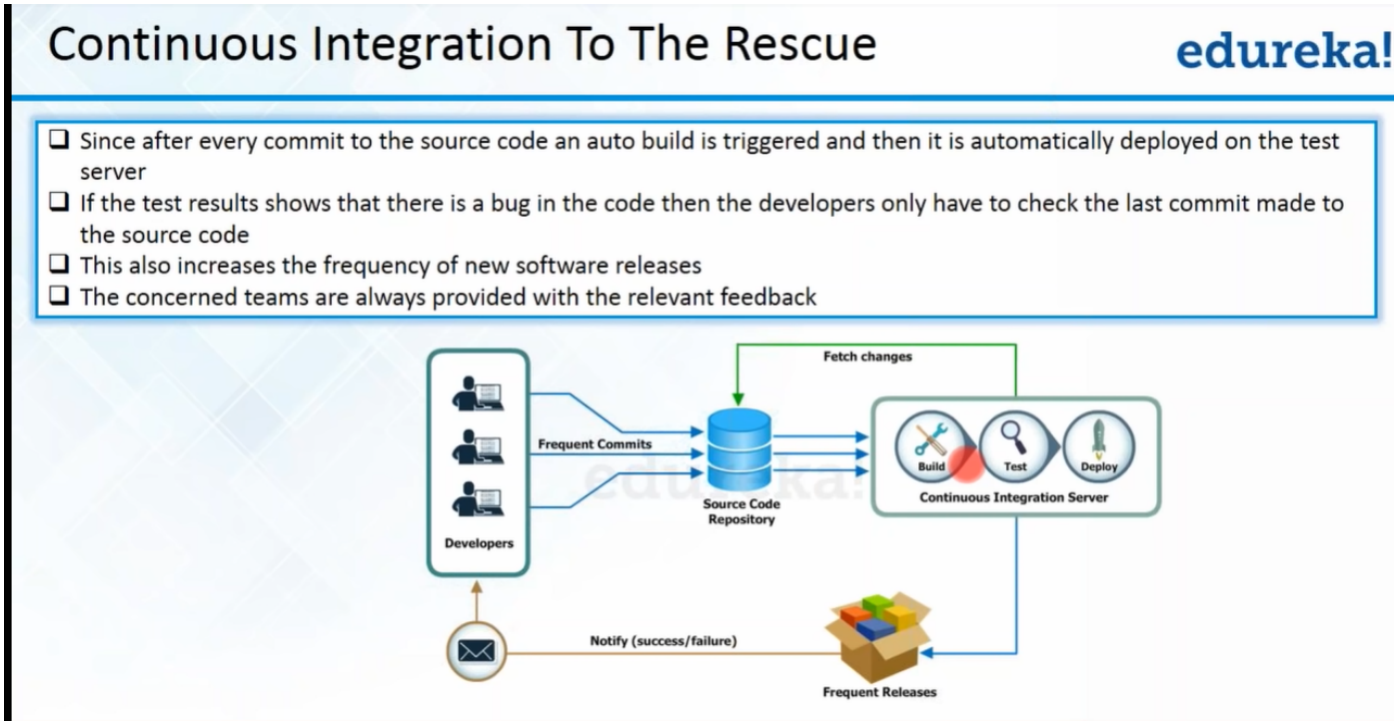




Continuous Deployment: after automated testing the code will be automatically loaded on the production server (**not very common**)



** Jenkins is the most common tool for CI/CD



Continuous Integration To The Rescue

edurel

Before Continuous Integration

- The entire source code was built and then tested.
- Developers have to wait for test results
- No Feedback

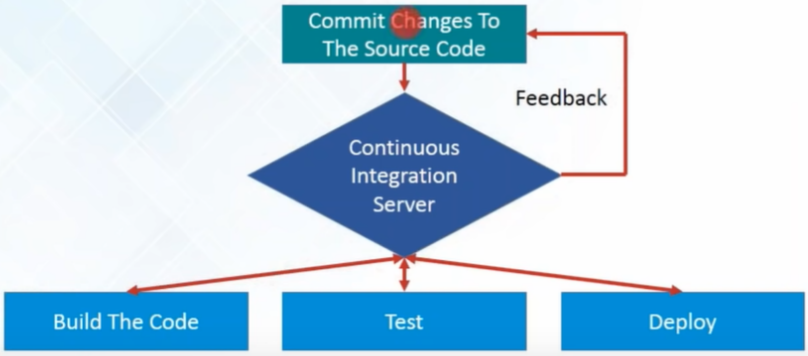
After Continuous Integration

- Every commit made in the source code is built and tested.
- Developers know the test result of every commit made in the source code on the run
- Feedback is present

What Is Continuous Integration

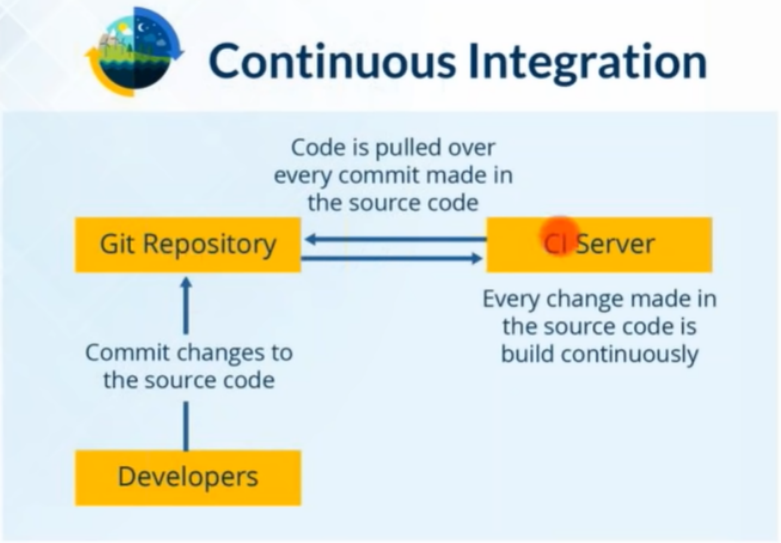
edureka!

- Continuous Integration is a development practice in which the developers are required to commit changes to the source code in a shared repository several times a day or more frequently.
- Every commit made in the repository is then built. This allows the teams to detect the problems early.



Continuous Integration Case-Study: Nokia

Solution:



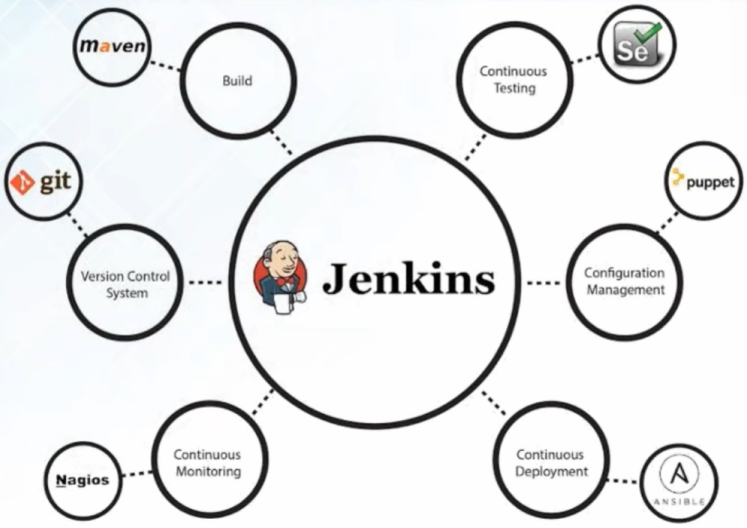
Continuous Integration Tools

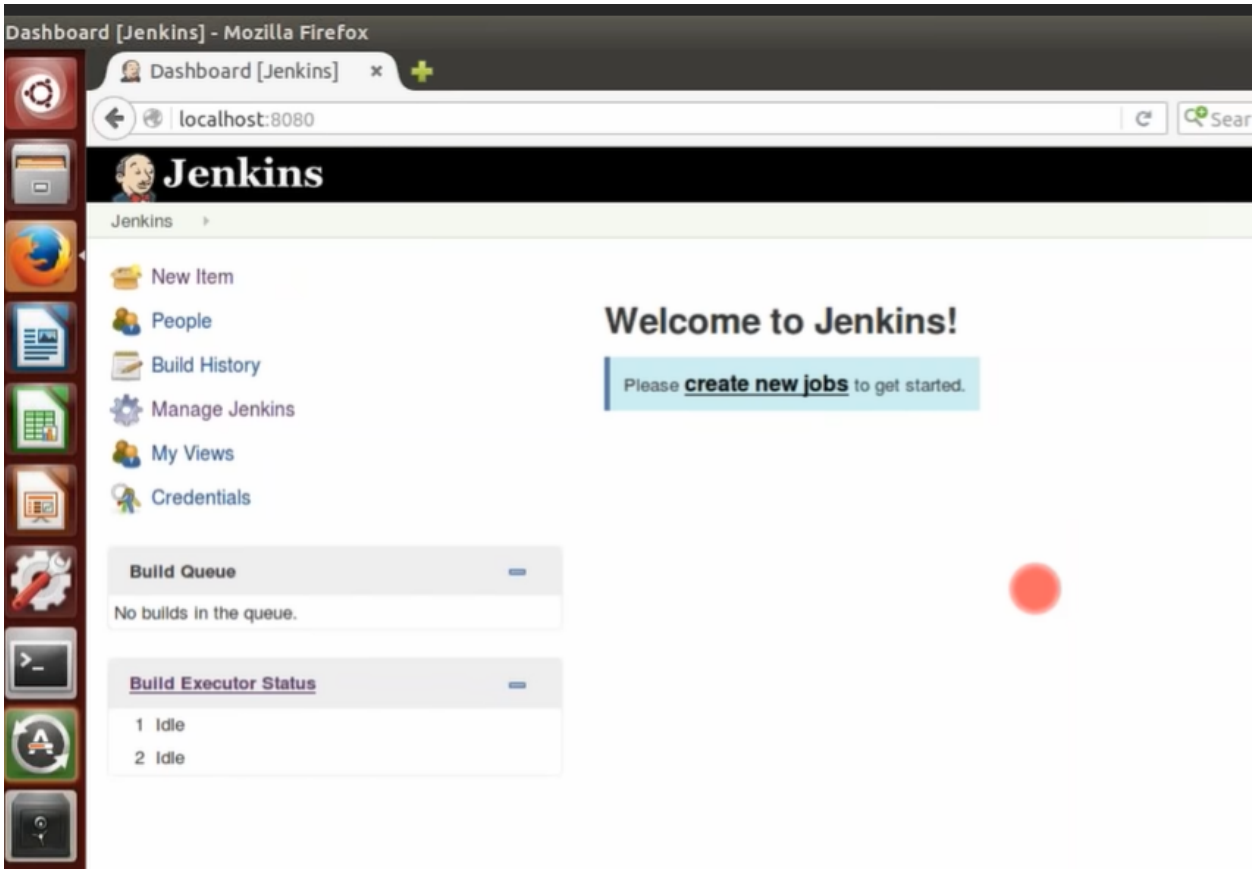
edureka!

What Is Jenkins?

edureka!

Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purpose. Plugins allows integration of various DevOps stages.

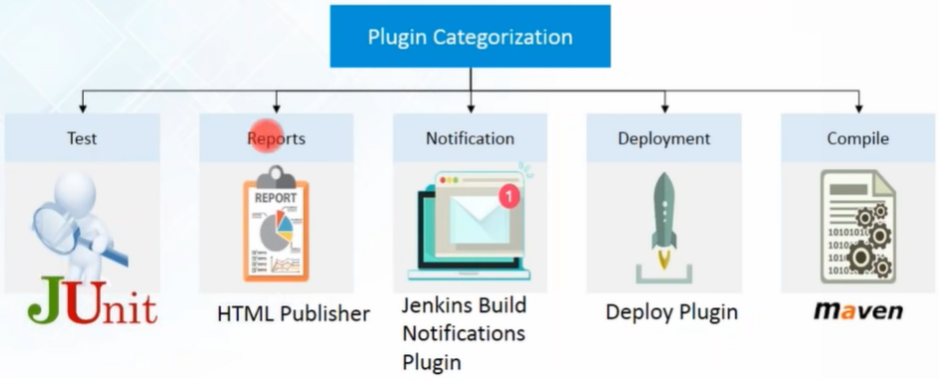




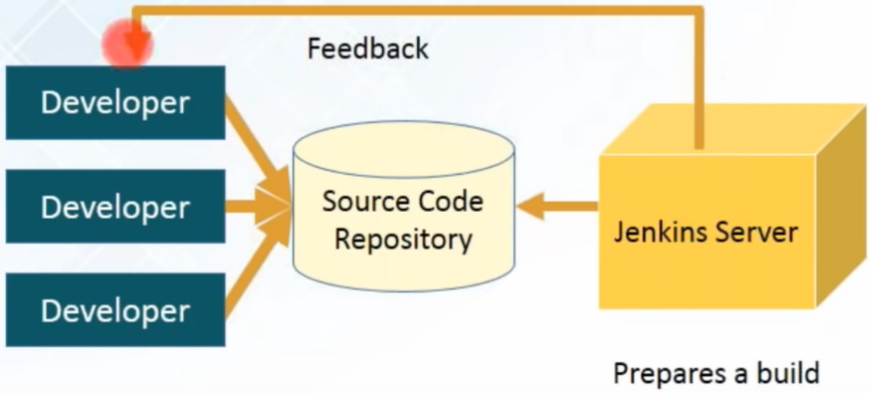
Jenkins Plugins

edureka!

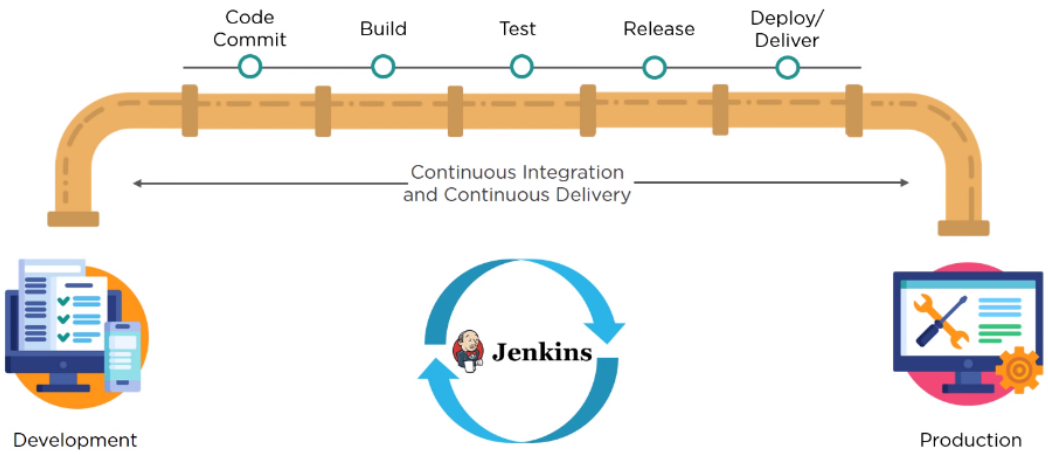
Jenkins supports plugins, which allow Jenkins to be extended to meet specific needs of individual projects



- Developers commit changes to the source code
- Continuous Integration server pulls that code and triggers a build

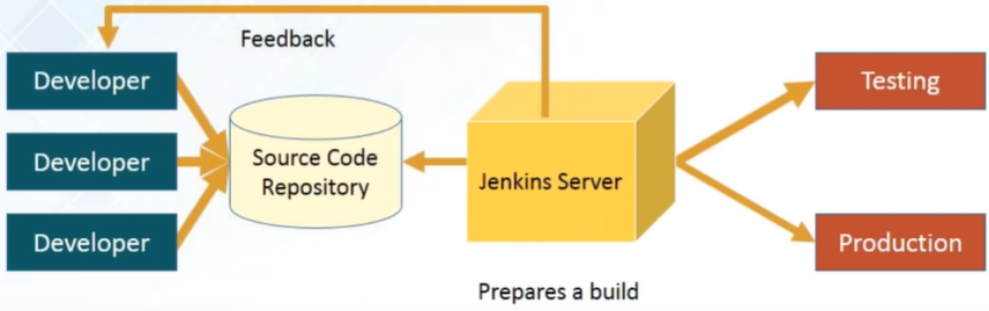


Jenkins Pipeline

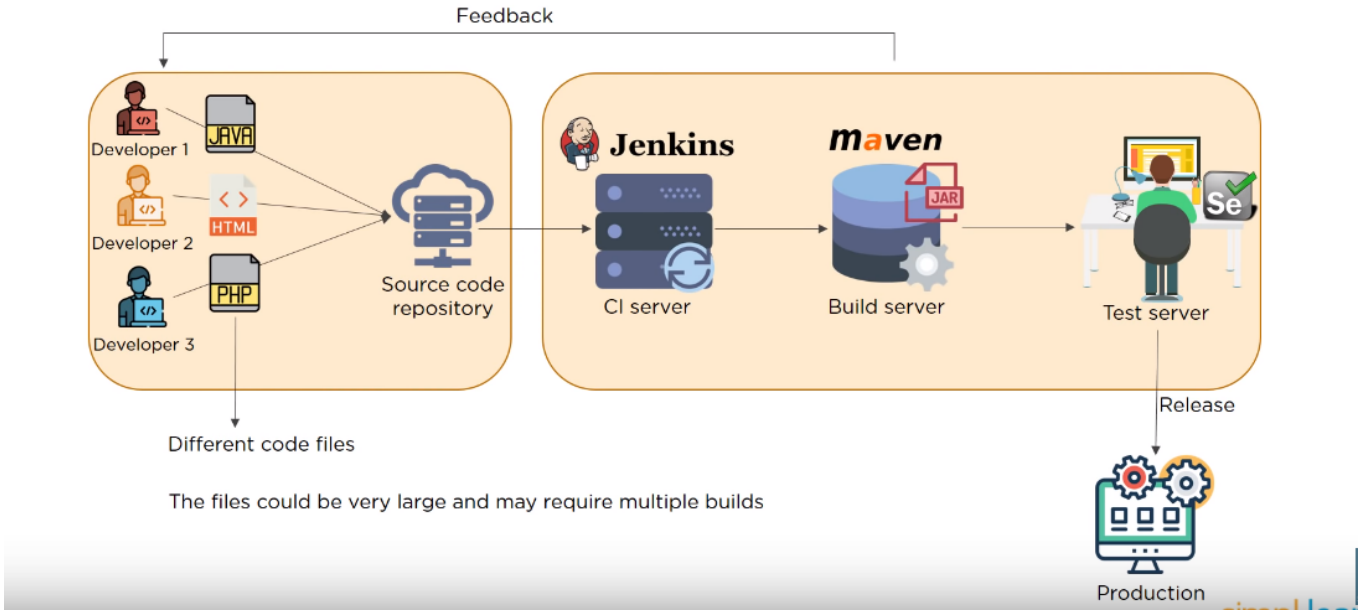


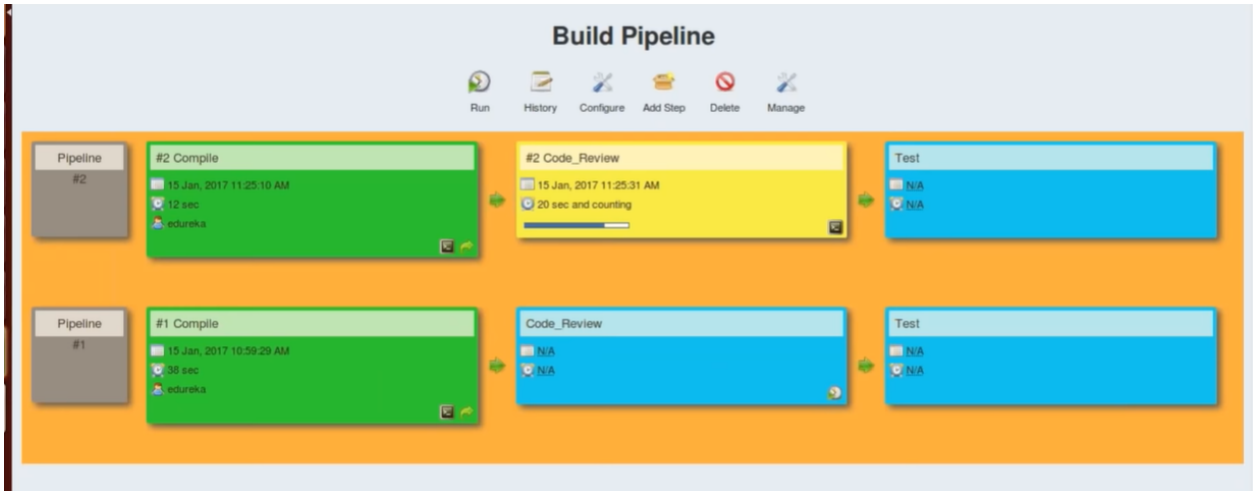
Jenkins Example

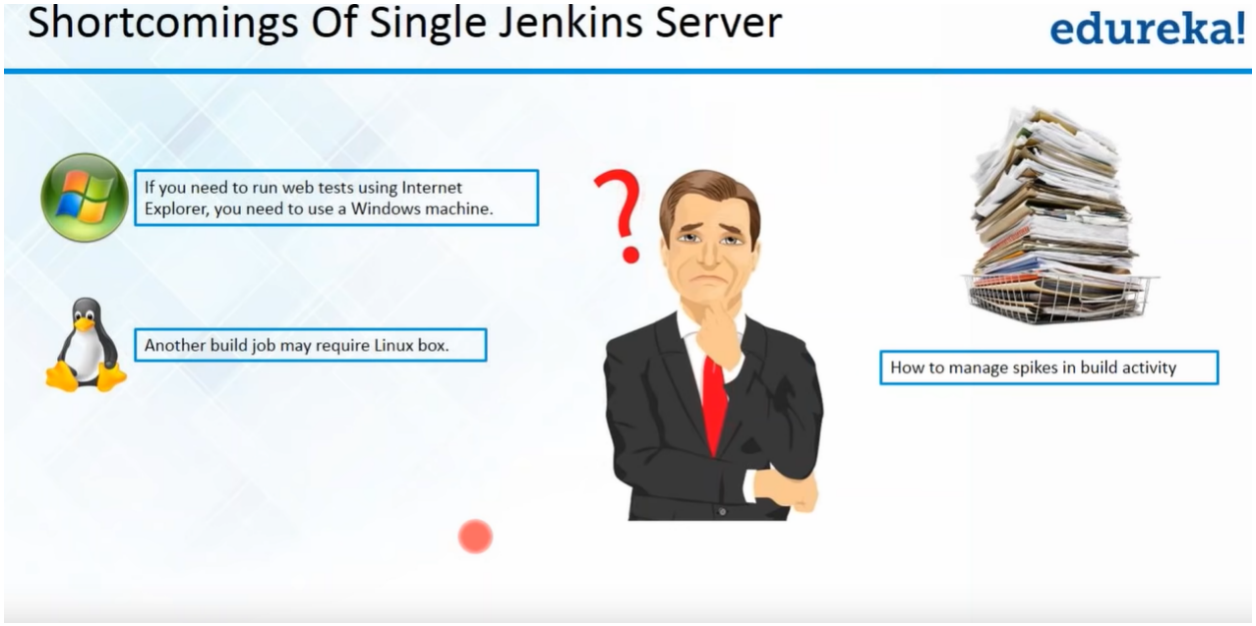
- Developers commit changes to the source code
- Continuous Integration server pulls that code and triggers a build
- The build application is then deployed on the testing server for testing
- After testing the application, it is then deployed on the production server
- The concerned teams are constantly notified about the build and test results



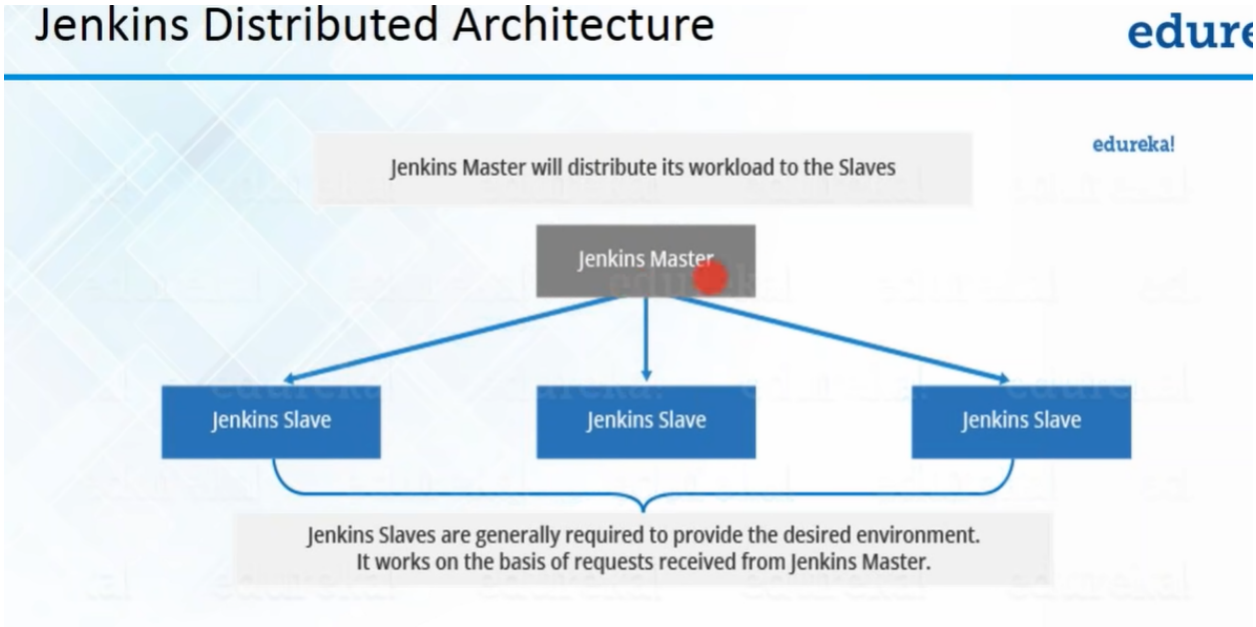
Jenkins Architecture





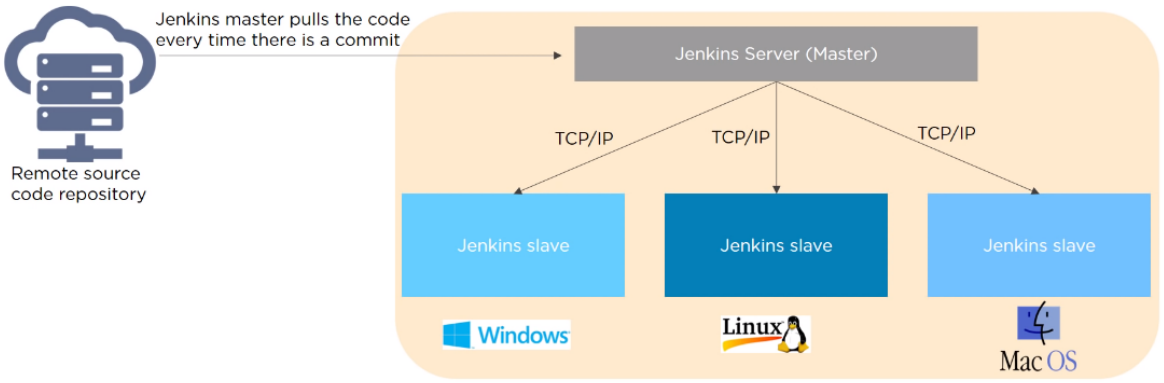


To Address The Shortcomings Of Single Jenkins Server, Jenkins Distributed Architecture Was Introduced

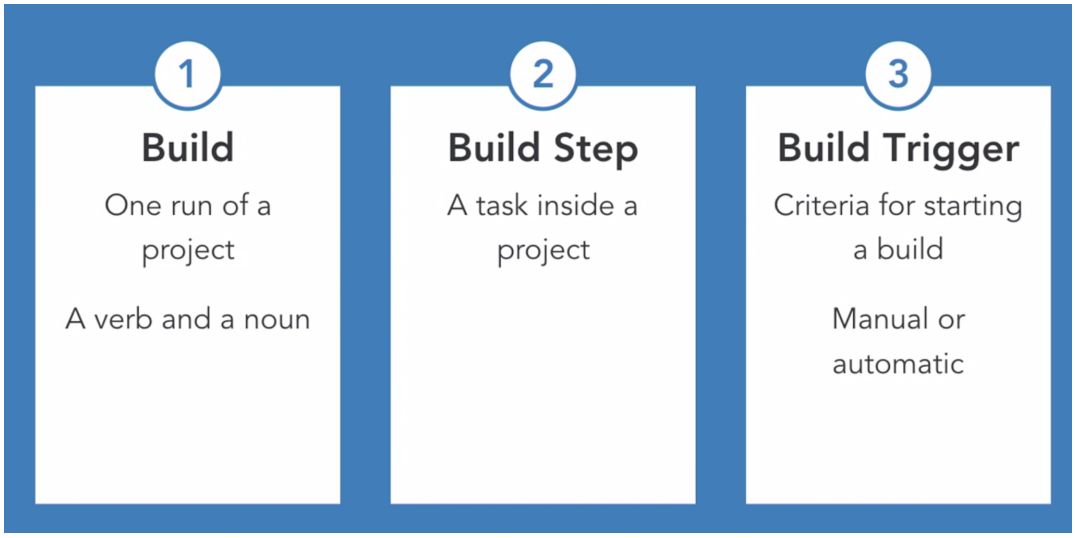


Jenkin slaves are java executables running on machines

Jenkins Master-Slave Architecture

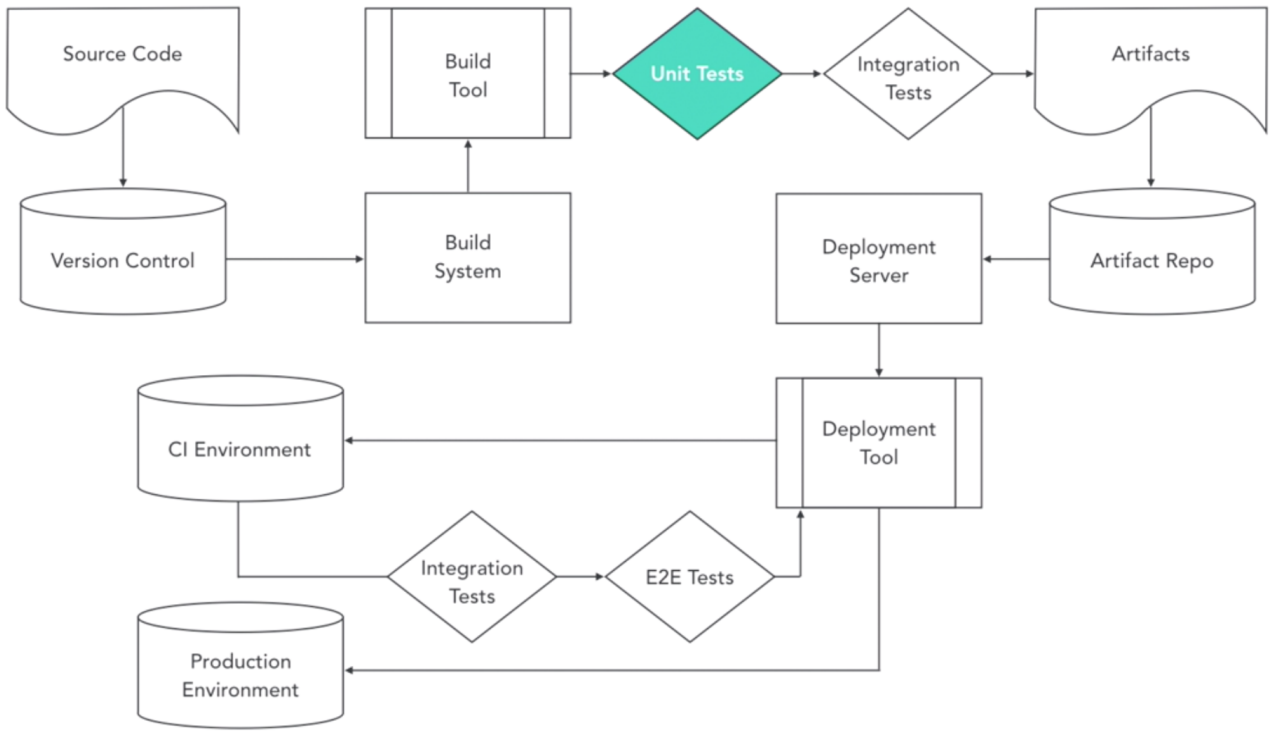


- Jenkins master distributes its workload to all the slaves
- On request from Jenkins master, the slaves carry out builds and tests and produce test reports

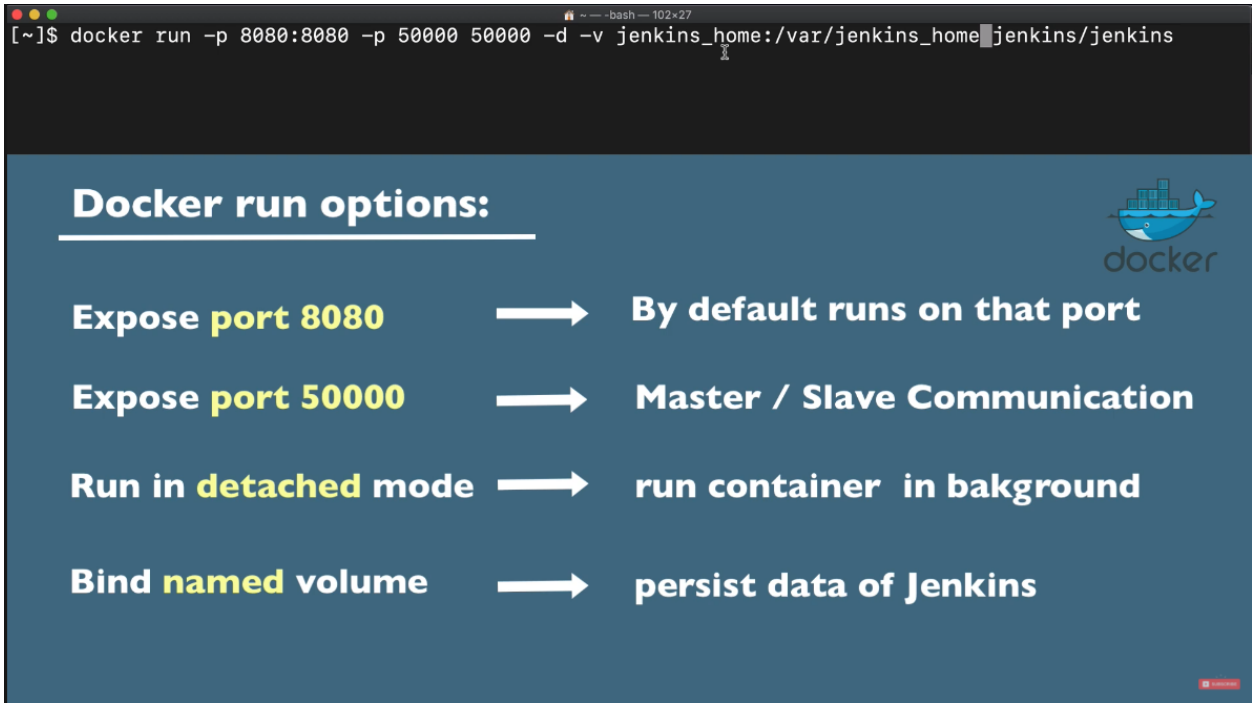


Jenkins Scheduler Format

```
*    *    *    *    *
-    -    -    -    -
|    |    |    |    |
|    |    |    |    +---------- day of week (0 - 7) (Sunday=0 or 7)
|    |    |    +--------------- month (1 - 12)
|    |    +-------------------- day of month (1 - 31)
|    +------------------------- hour (0 - 23)
|
+------------------------------ min (0 - 59)
```



Create and run jenkins from docker

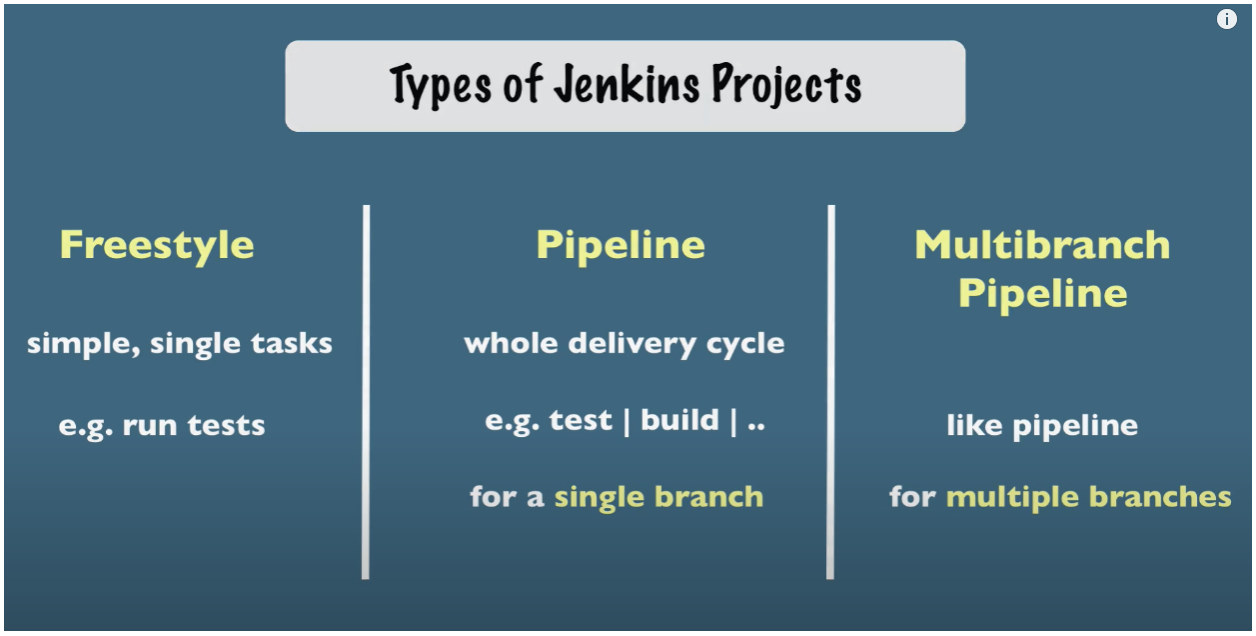


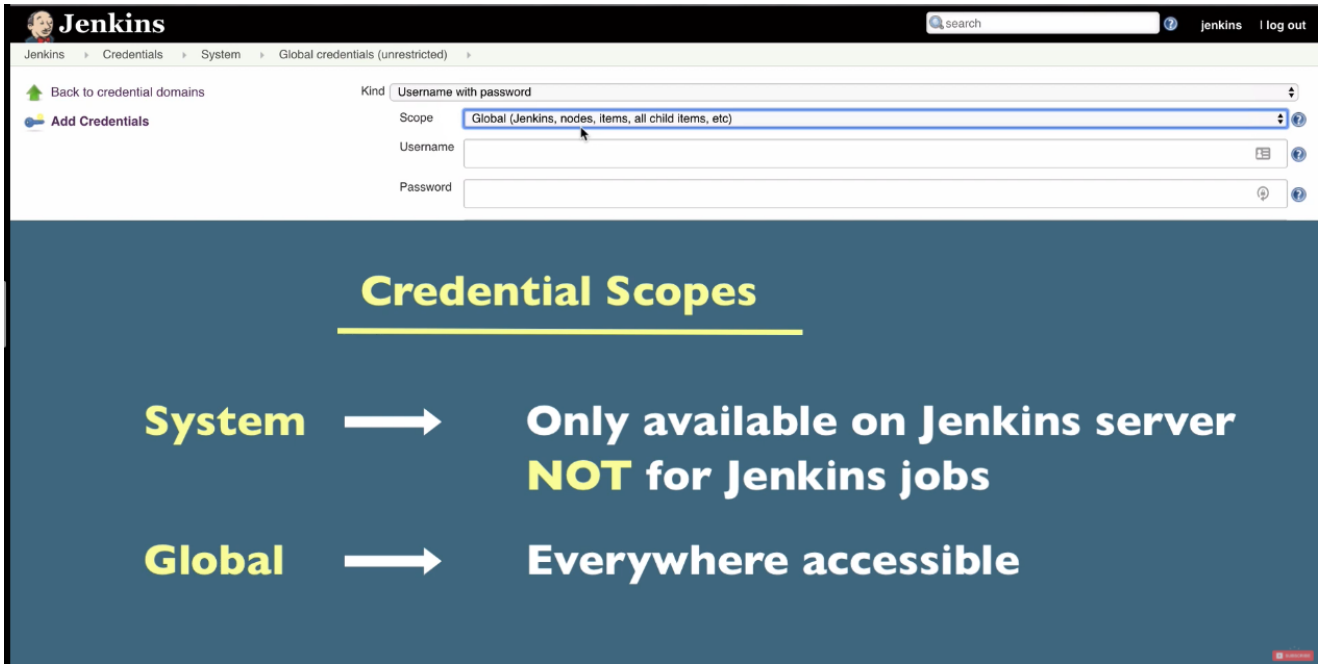

```
root@ubuntu-s-2vcpu-2gb-fra1-01:~# docker run -p 8080:8080 -p 50000:50000 -d \
> -v jenkins_home:/var/jenkins_home jenkins/jenkins:lts
```

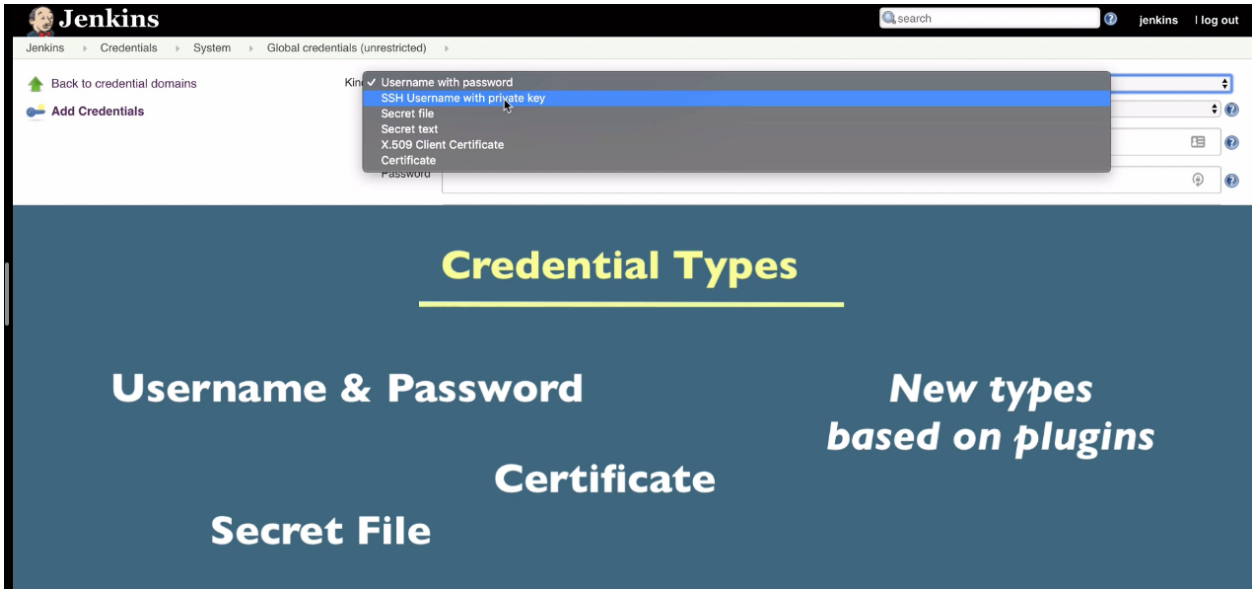
```
root@ubuntu-s-2vcpu-4gb-fra1-01:~# docker volume inspect jenkins_home
[
    {
        "CreatedAt": "2020-11-01T19:54:47Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/jenkins_home/_data",
        "Name": "jenkins_home",
        "Options": null,
        "Scope": "local"
    }
]
```

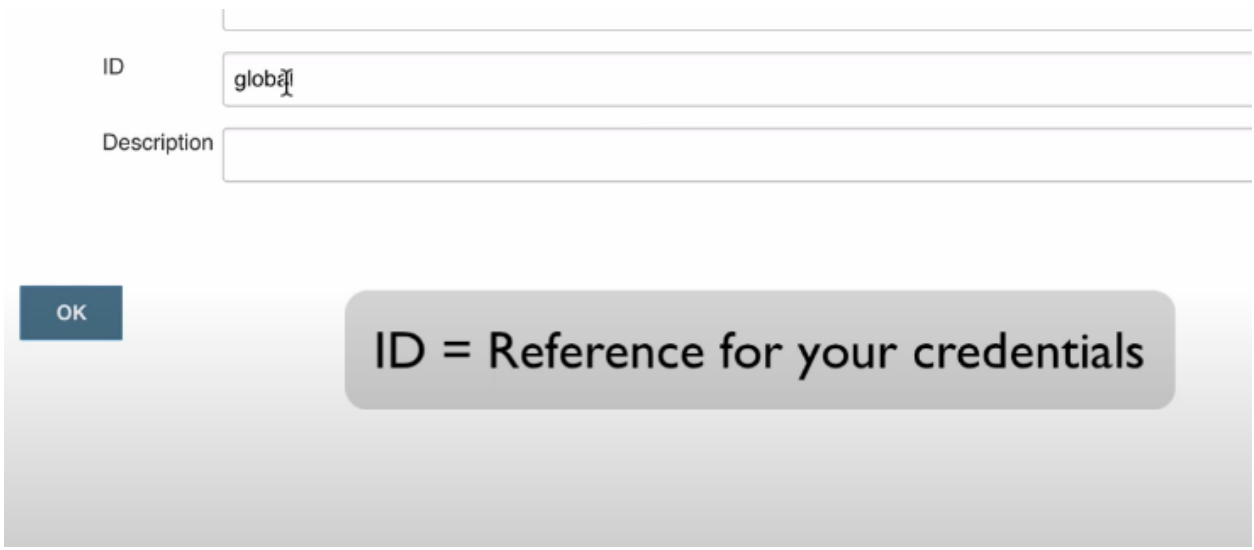
Use the given password to login into jenkins GUI on browser

```
*************************************************************
*************************************************************
*************************************************************

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

29858dcfec3245e7ac3586963c02ac39

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*************************************************************
*************************************************************
*************************************************************
```



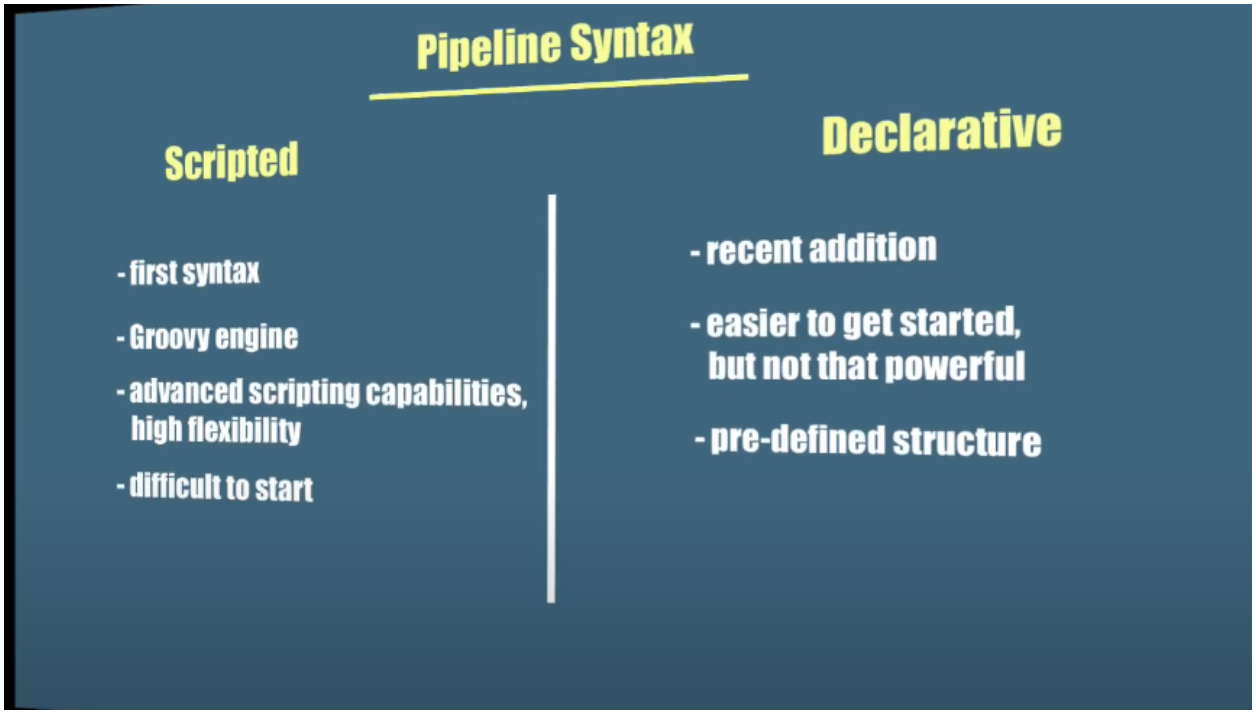

Jenkins
search
jenkins
log out
Jenkins
Credentials
System
Global credentials (unrestricted)
Back to credential domains
Add Credentials
Kind
Username with password
Scope
Global (Jenkins, nodes, items, all child items, etc)
Username
Password
Credential Scopes
System
Only available on Jenkins server
NOT for Jenkins jobs
Global
Everywhere accessible

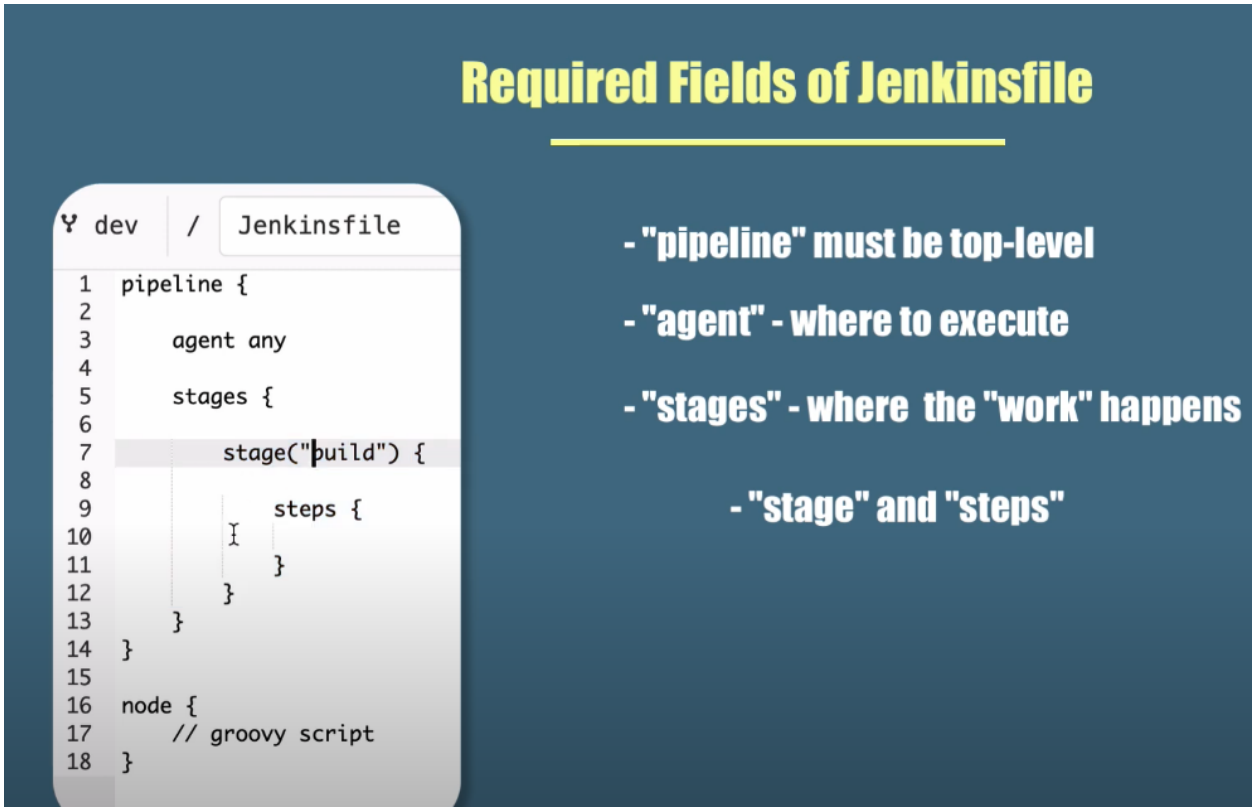

Jenkins
search
jenkins
log out
Jenkins
Credentials
System
Global credentials (unrestricted)
Back to credential domains
Add Credentials
Username with password
SSH Username with private key
Secret file
Secret text
X.509 Client Certificate
Certificate
Credential Types
Username & Password
New types
based on plugins
Certificate
Secret File



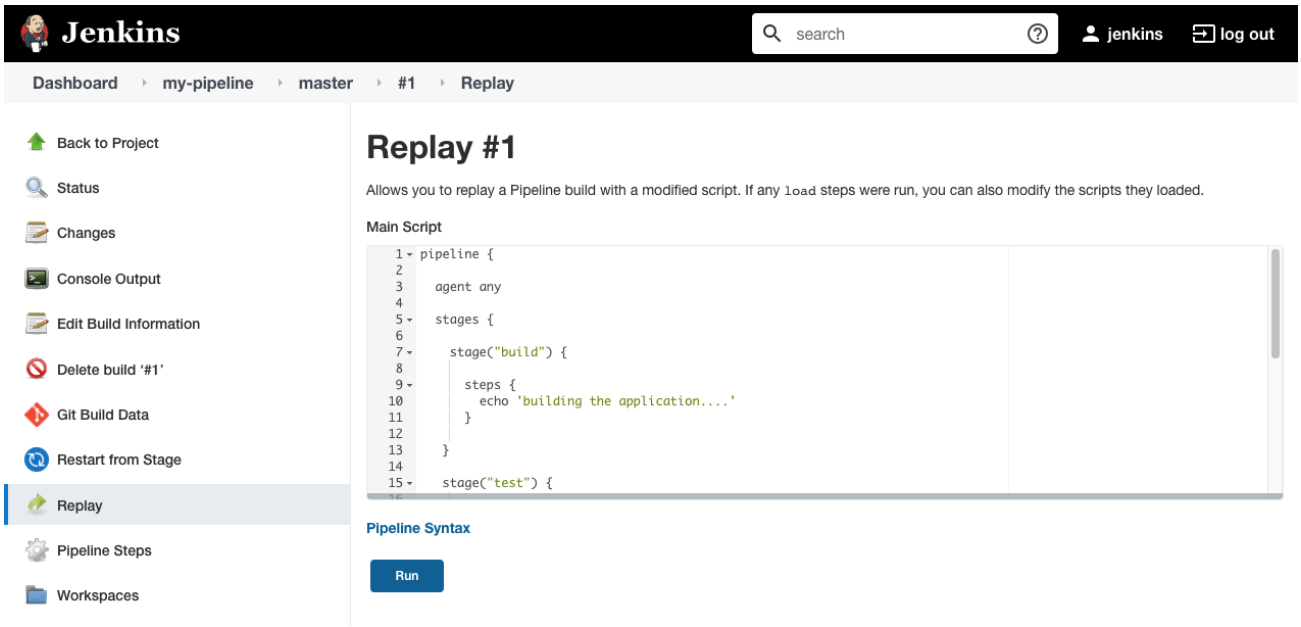
Credential Scopes

Project → **Limited to project, ONLY with multibranch pipeline**

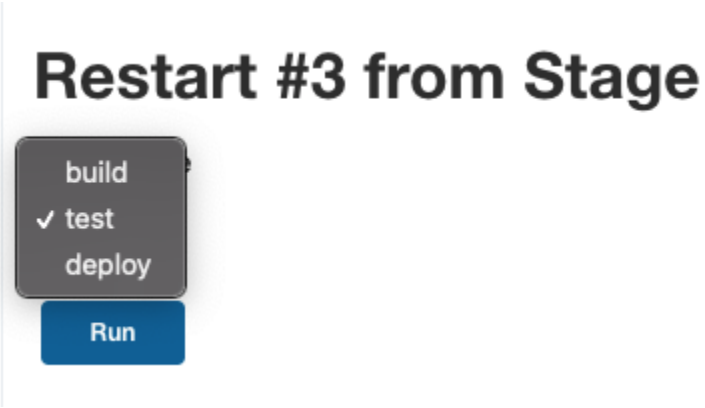




Here is how you can edit the JenkinsFile in the browser, to test it (won't affect the git source file)



Instead of running the whole app, you can restart it from an specific stage:



2 ways to trigger build

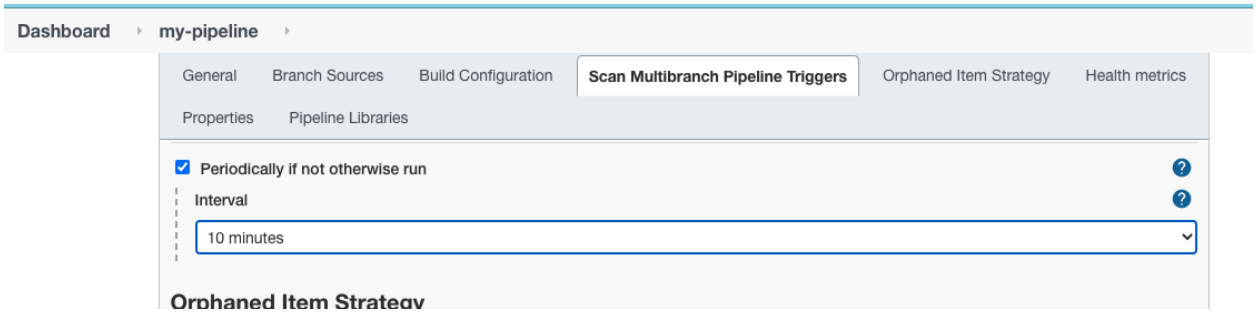
Push Notification	Version Control notifies Jenkins on new commit
Polling	Jenkins polls in regular intervalls

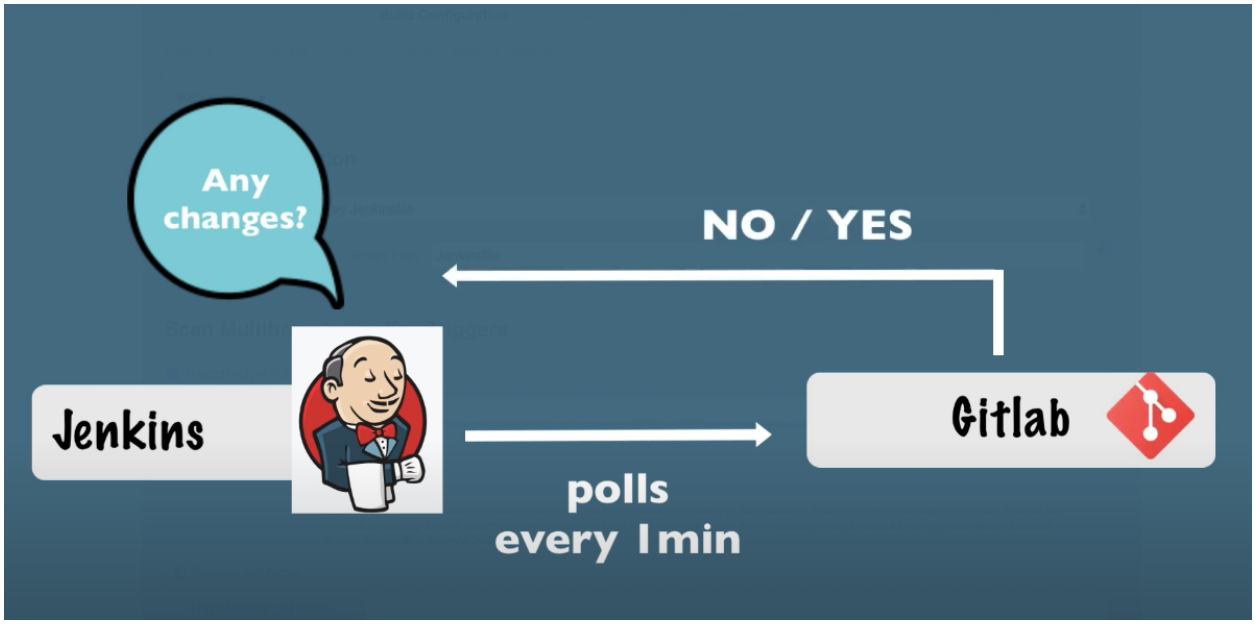
Trigger build

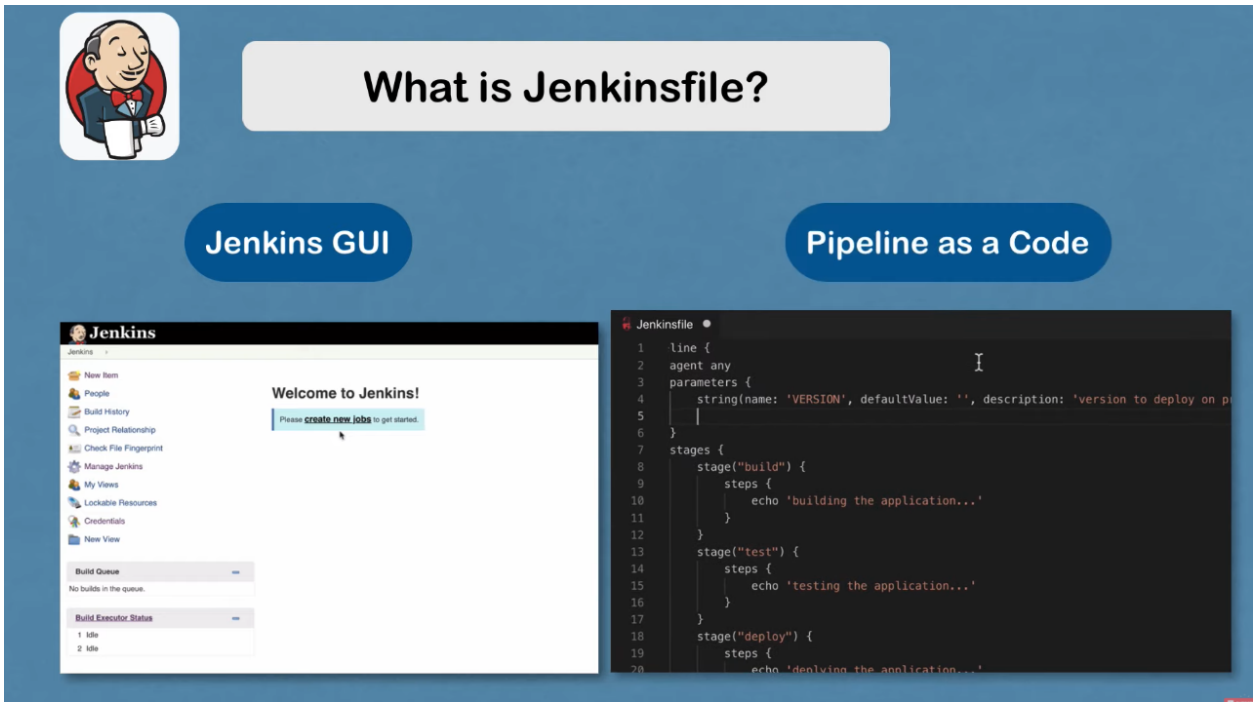
1. Install Jenkins plugin based on your Version Control System
2. Configure Repository Server Hostname
3. And Access Token or Credential

Webhooks of github can auto-trigger a new jenkins build when source code changes

Or you can manually set a trigger for each X minutes or hours to check the git source code:



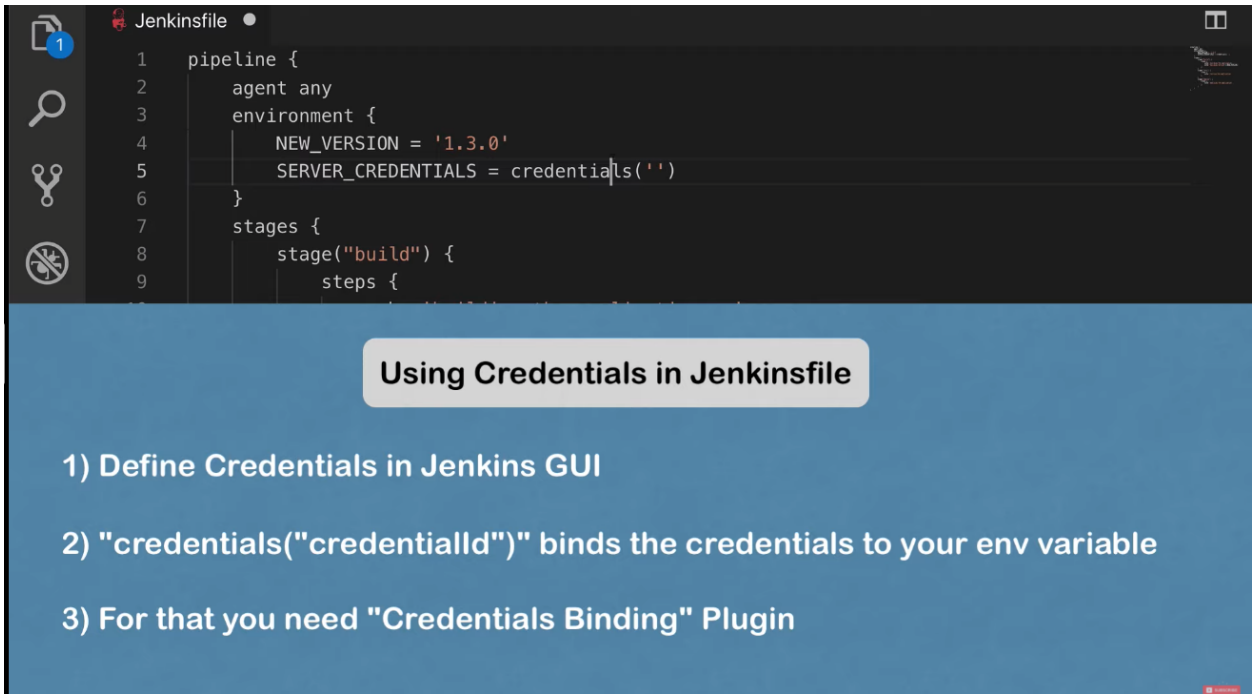


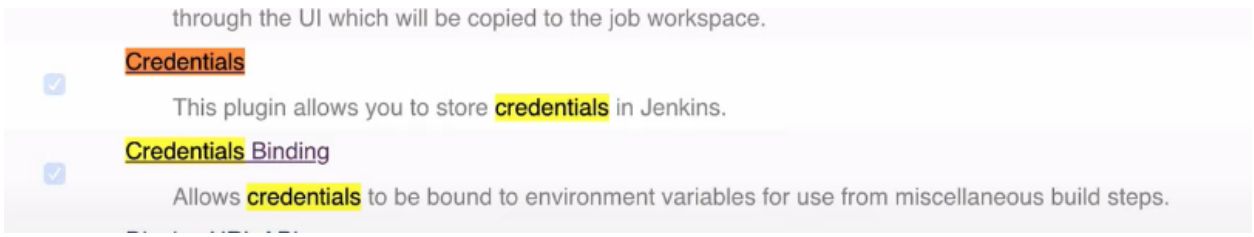


Different variables available to jenkinsFile :

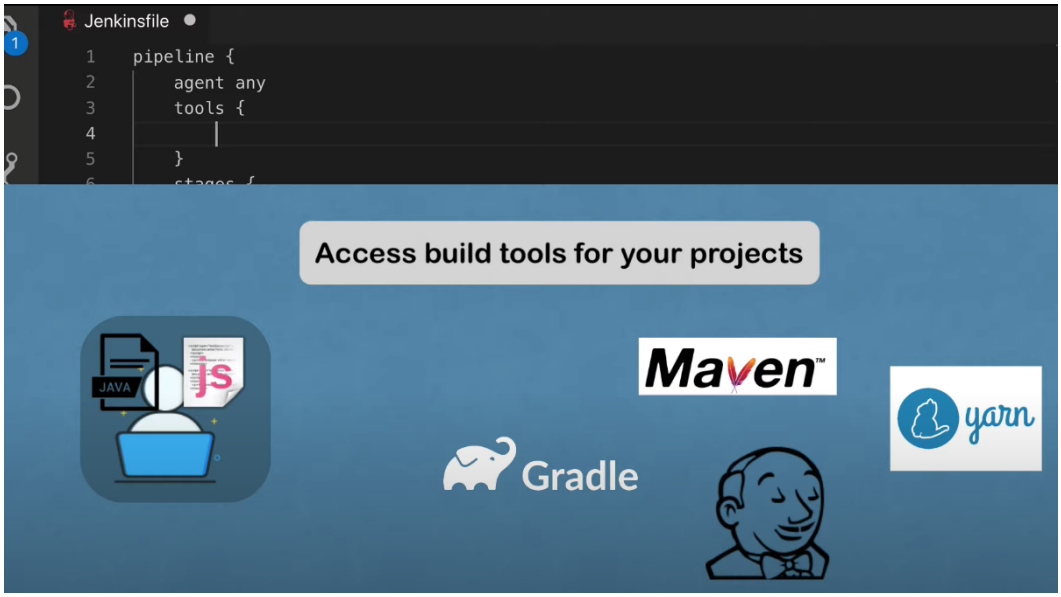
The following variables are available to shell scripts

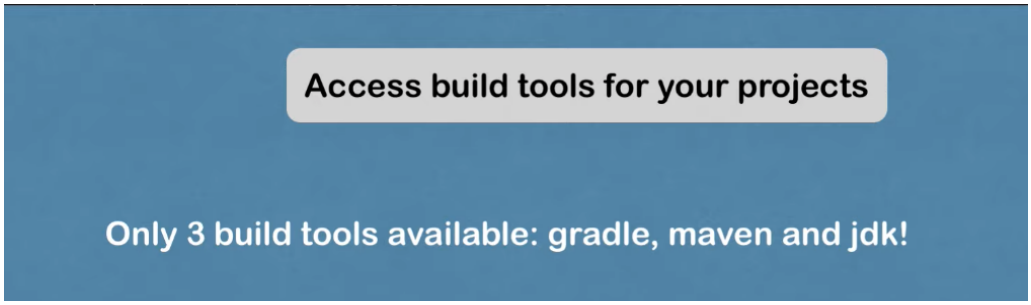
BRANCH_NAME
For a multibranch project, this will be set to the name of the branch being built, for example in case you wish to deploy to production from `master` but not from feature branches; if corresponding to some kind o change request, the name is generally arbitrary (refer to `CHANGE_ID` and `CHANGE_TARGET`).
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.
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

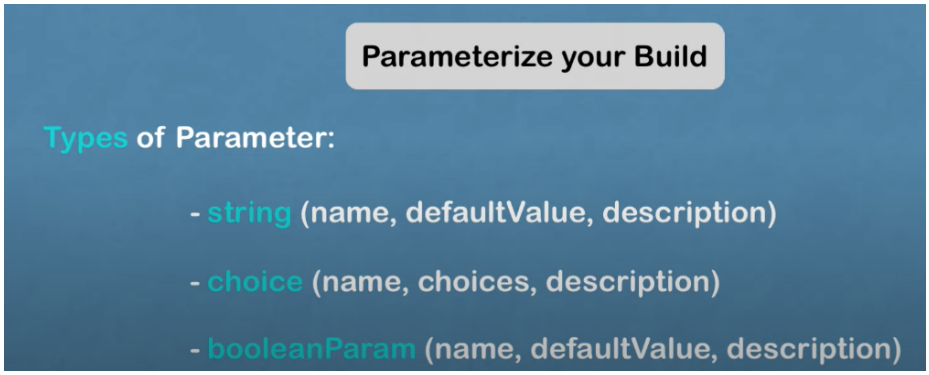


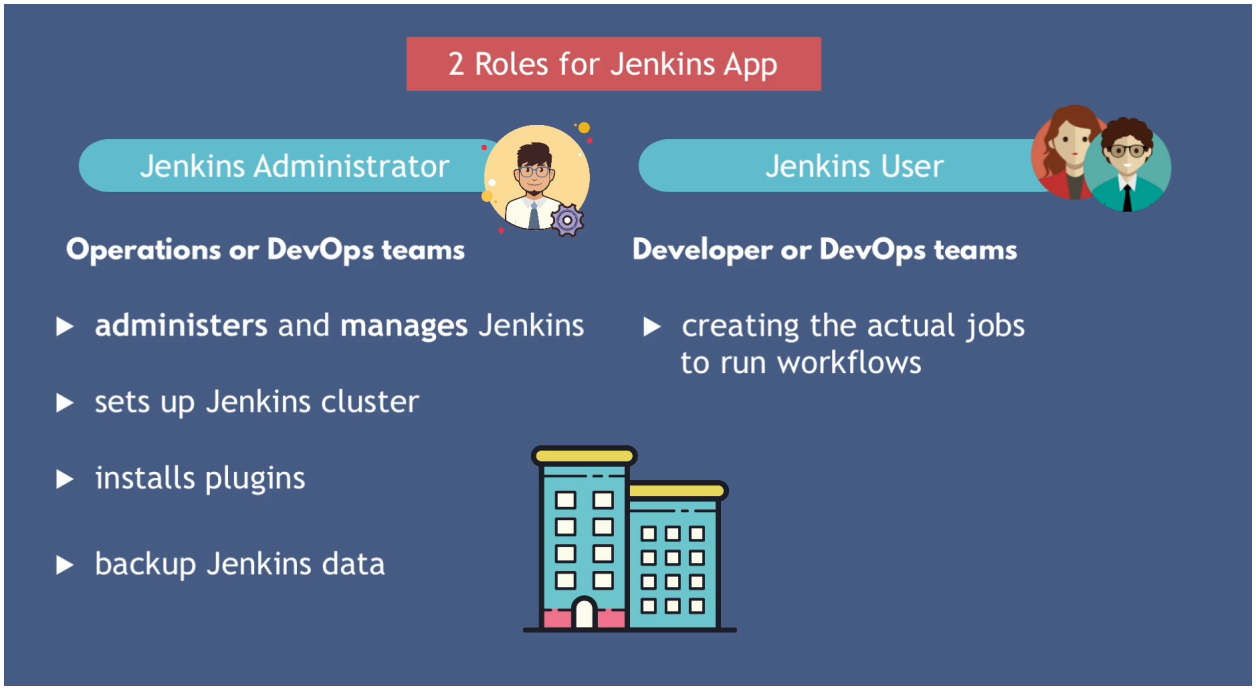


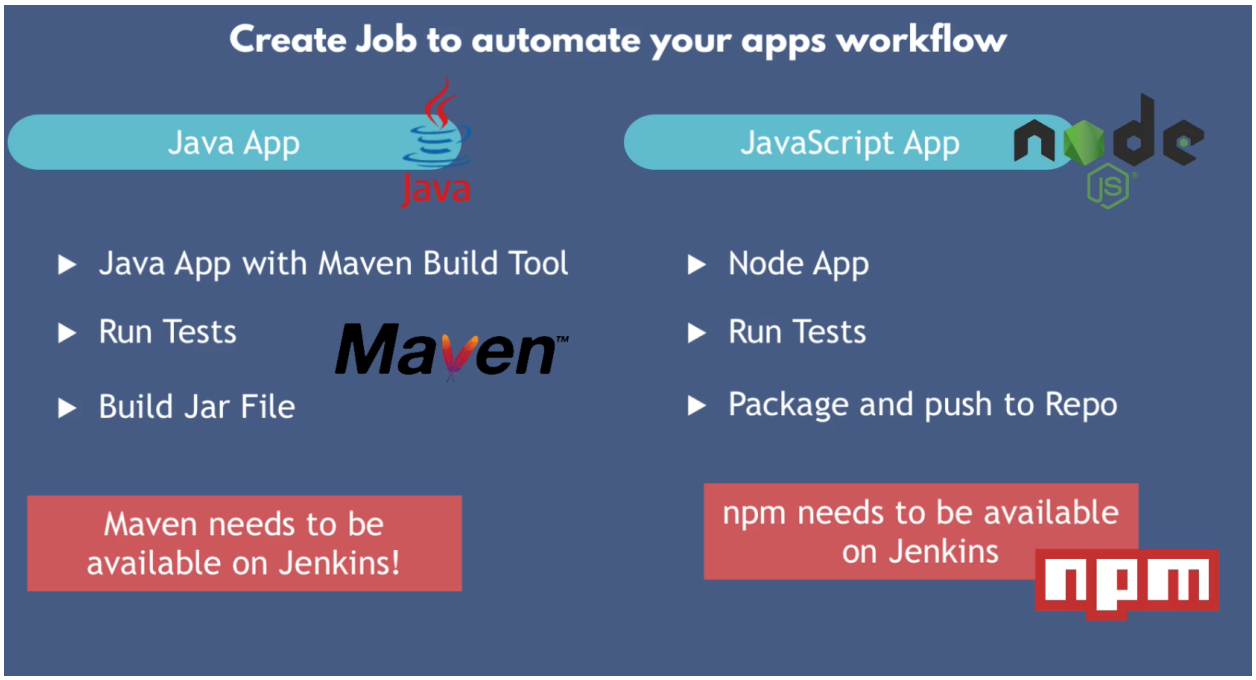
Build tools, you can access them in JenkinsFile, but first install them on Jenkins UI











To automate the test and building of an application, you need to have your package managers and libraries installed on your Jenkins environment.

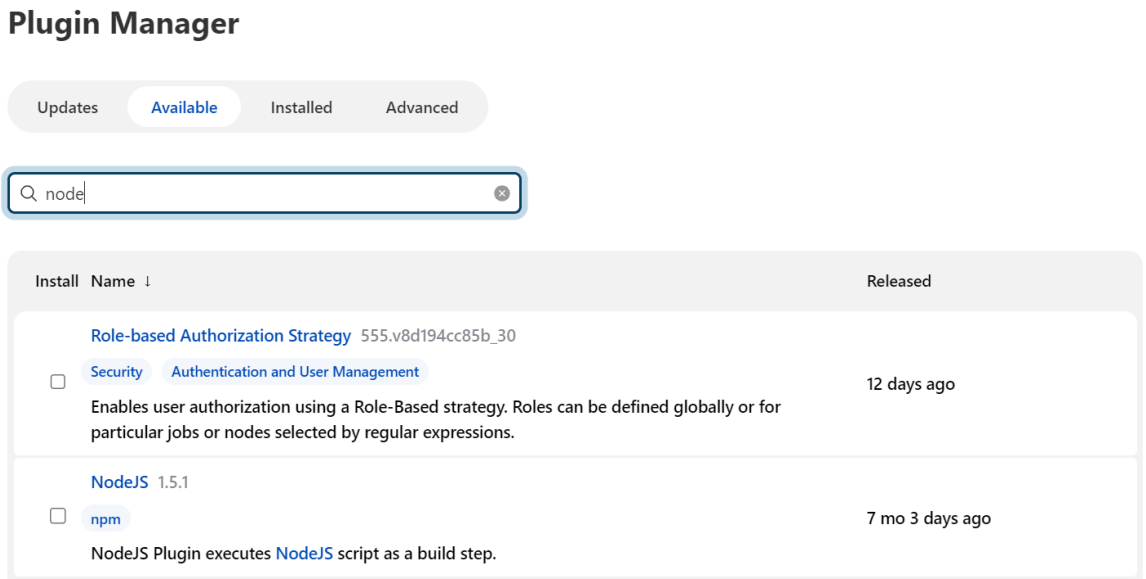
All config files related to jenkins are saved in its directory

```
root@e4c1d9906c6d:/var/jenkins_home# ls
config.xml                                      jobs
copy_reference_file.log                         logs
credentials.xml                                 nodeMonitors.xml
fingerprints                                    nodes
hudson.model.UpdateCenter.xml                   org.jenkinsci.plugins.gitclient.JGitApacheTool.xml
hudson.plugins.emailext.ExtendedEmailPublisher.xml  org.jenkinsci.plugins.gitclient.JGitTool.xml
hudson.plugins.git.GitTool.xml                  plugins
hudson.plugins.gradle.Gradle.xml                queue.xml
hudson.tasks.Ant.xml                            secret.key
hudson.tasks.Maven.xml                          secret.key.not-so-secret
```

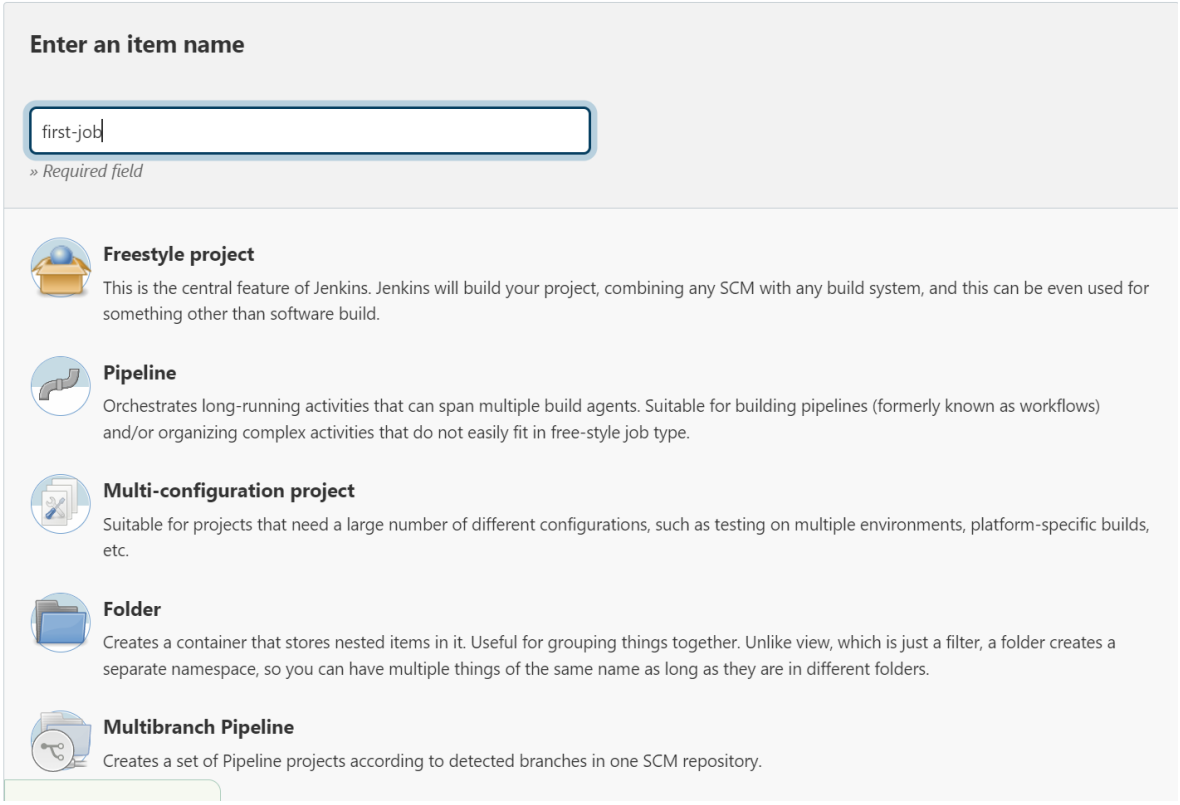
```
root@e4c1d9906c6d:/var/jenkins_home/jobs/first-job# ls
builds  config.xml  nextBuildNumber
```

Here you can optimize and configure all the downloaded plugins of jenkins

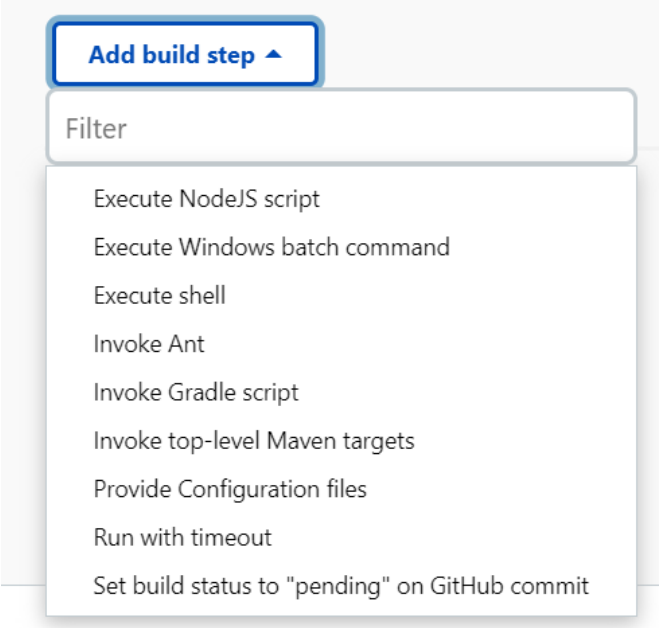
Installing needed plugins



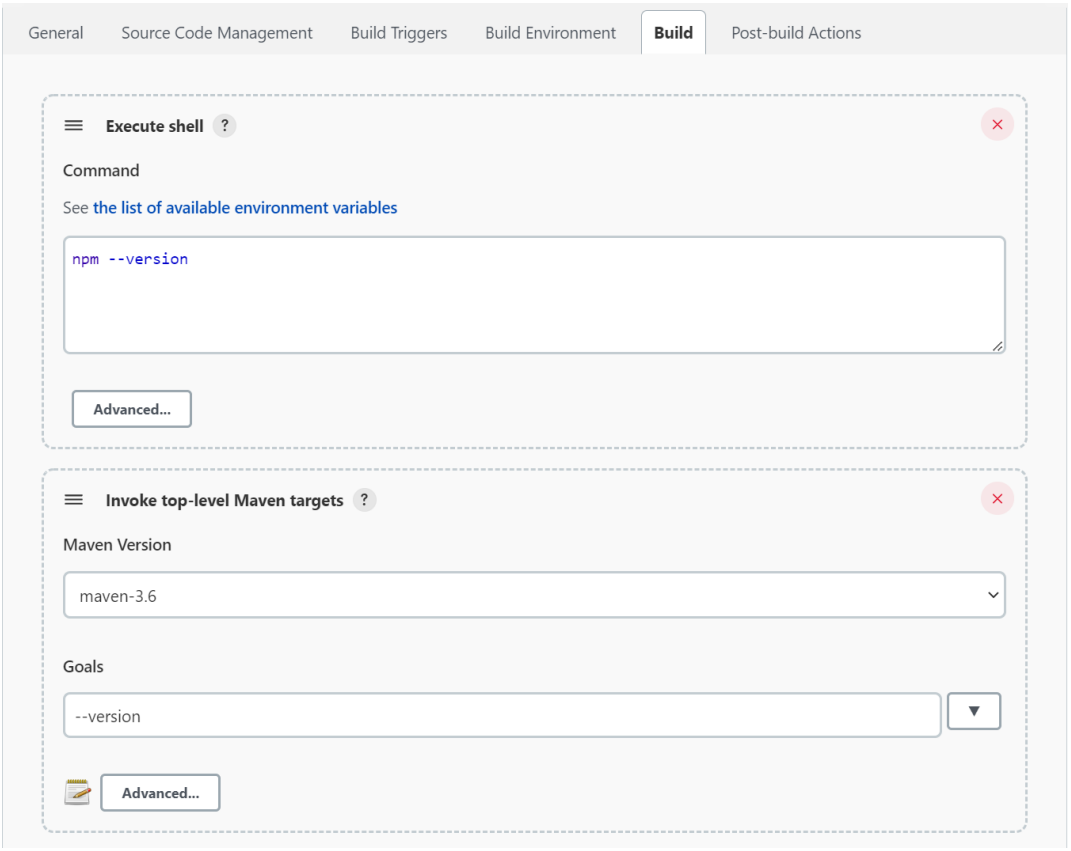
Another way is to directly install software inside the server that jenkins is running on, for example from the linux command line of the server.

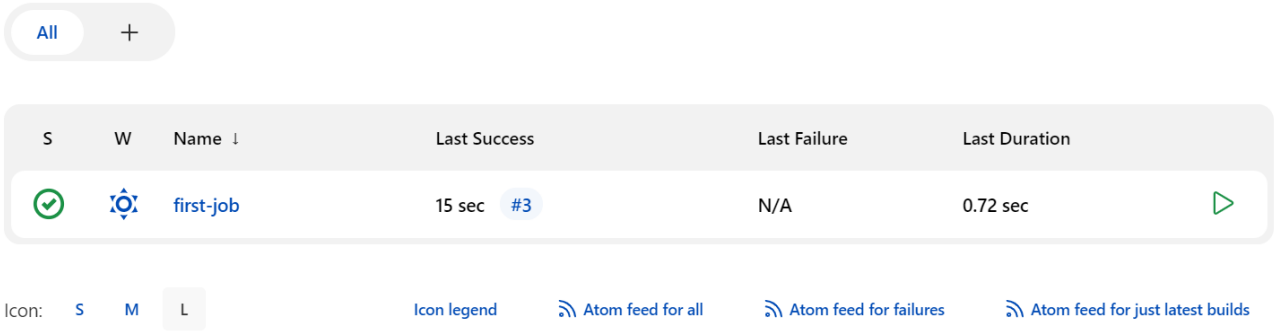


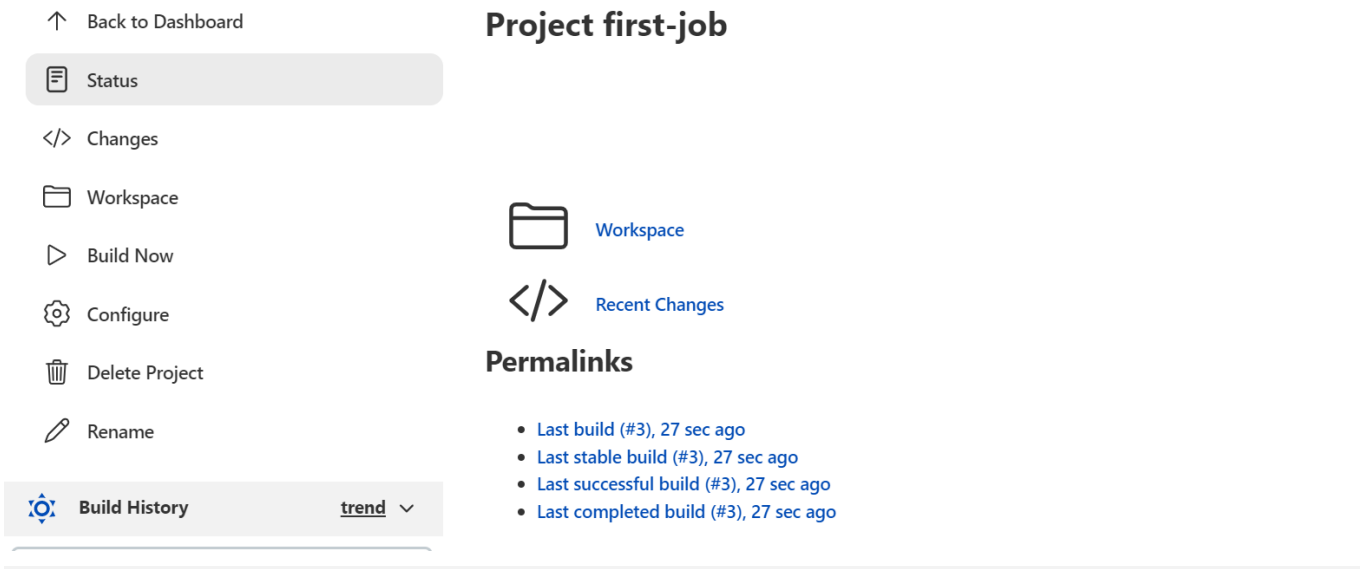
If a plugin is installed on the jenkins server, we have that tool as a “Build Step”

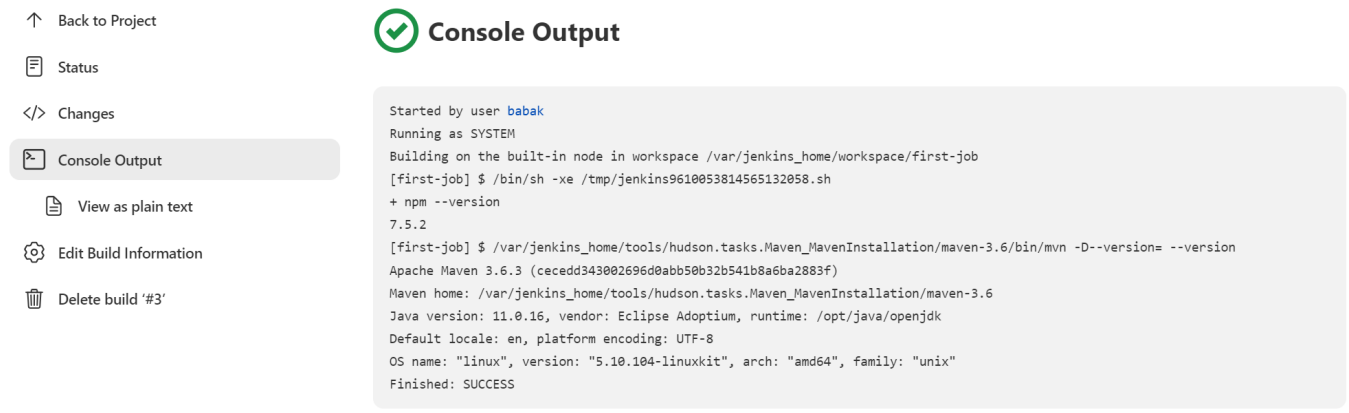


Build section is where you can execute the script or commands









Connecting jenkins build to the repository

General | **Source Code Management** | Build Triggers | Build Environment | Build | Post-build Actions

Git ?

Repositories ?

Repository URL ?

https://github.com/Babak-H/gitplay.git

Credentials ?

Babak-H/******

+ Add

Advanced...

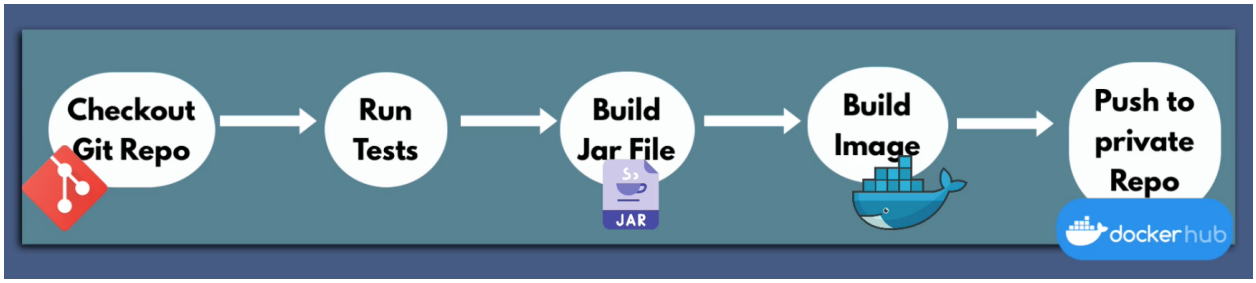
Add Repository

Branches to build ?

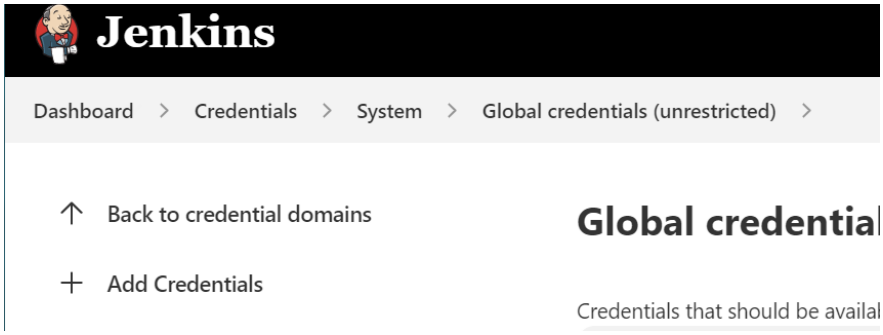
Branch Specifier (blank for 'any') ?

*/master

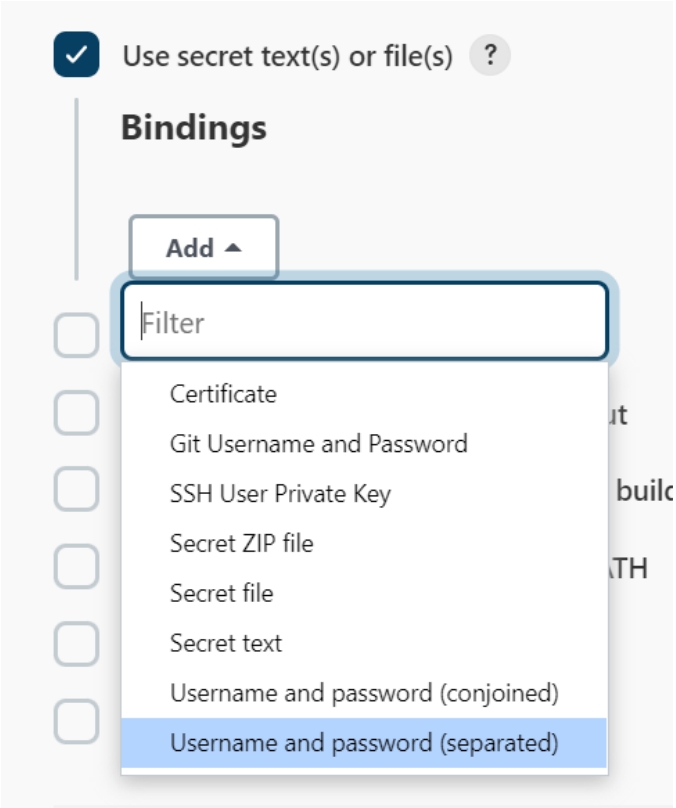
Add Branch

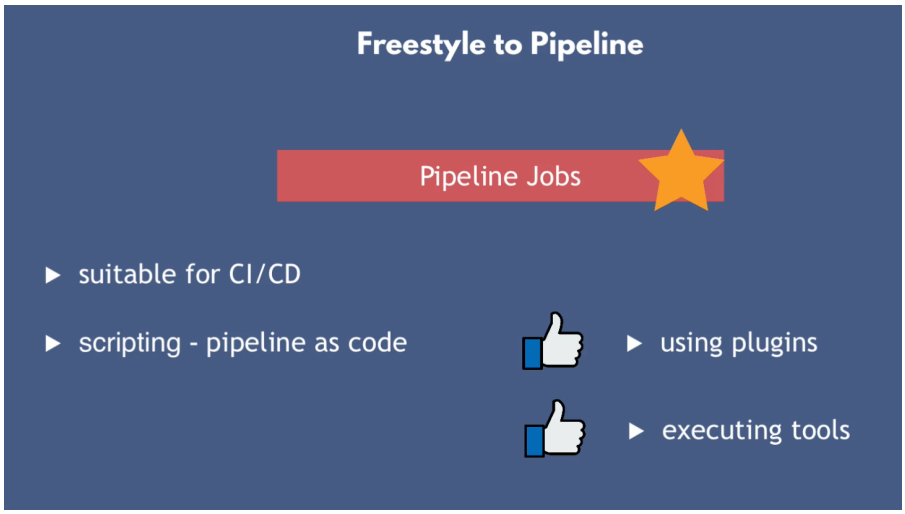


In here we can add the needed dockerHub credentials :



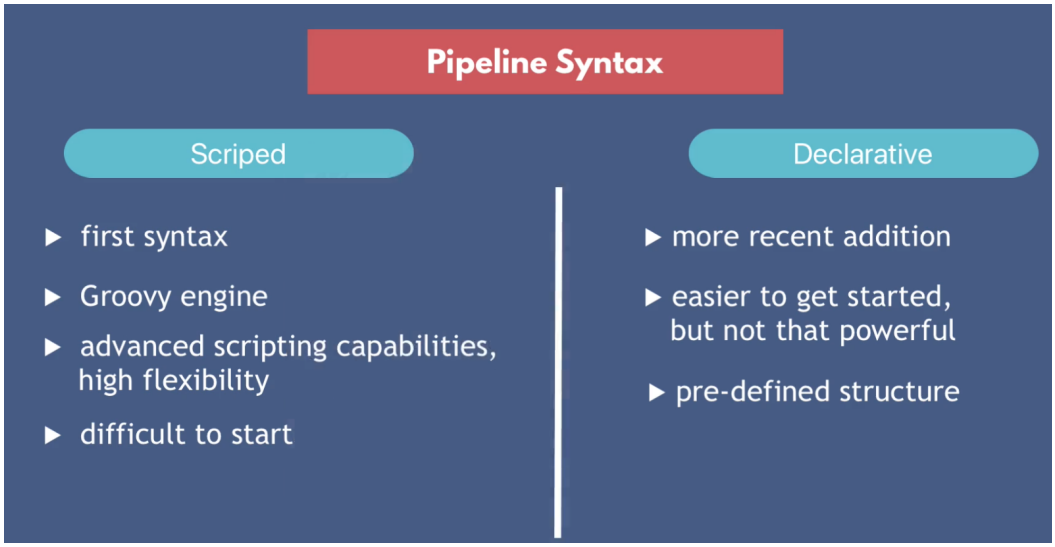
How to use those creds in the pipeline





Freestyle pipeline is mostly based on UI, but pipeline projects are scripted, script can come from the source code or be written on the Jenkins page itself.

JenkinsFile (similar to .gitlabCI file) is the file you can put in your repo that Jenkins will read the pipeline from it



Using Credentials in Jenkinsfile

1) Define Credentials in Jenkins GUI (explained in next)

2) "credentials("credentialId")" binds the credentials to your env variable

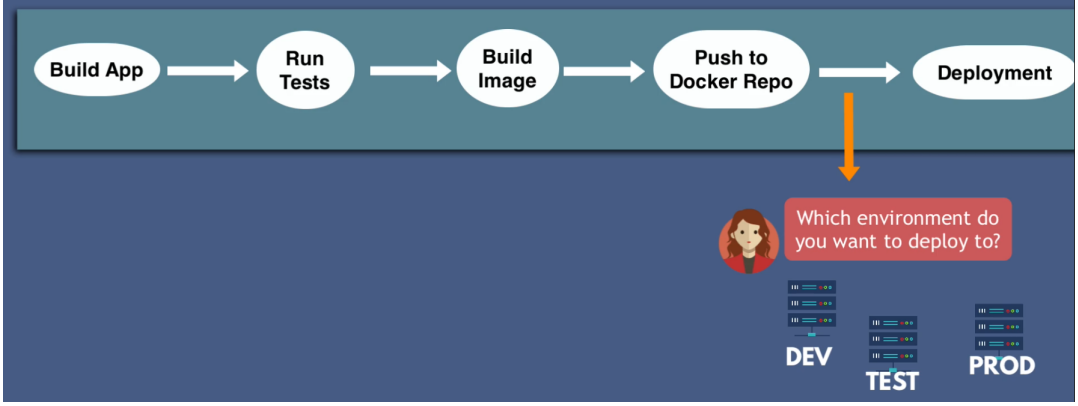
3) For that you need "Credentials Binding" Plugin

Paramerize your Build

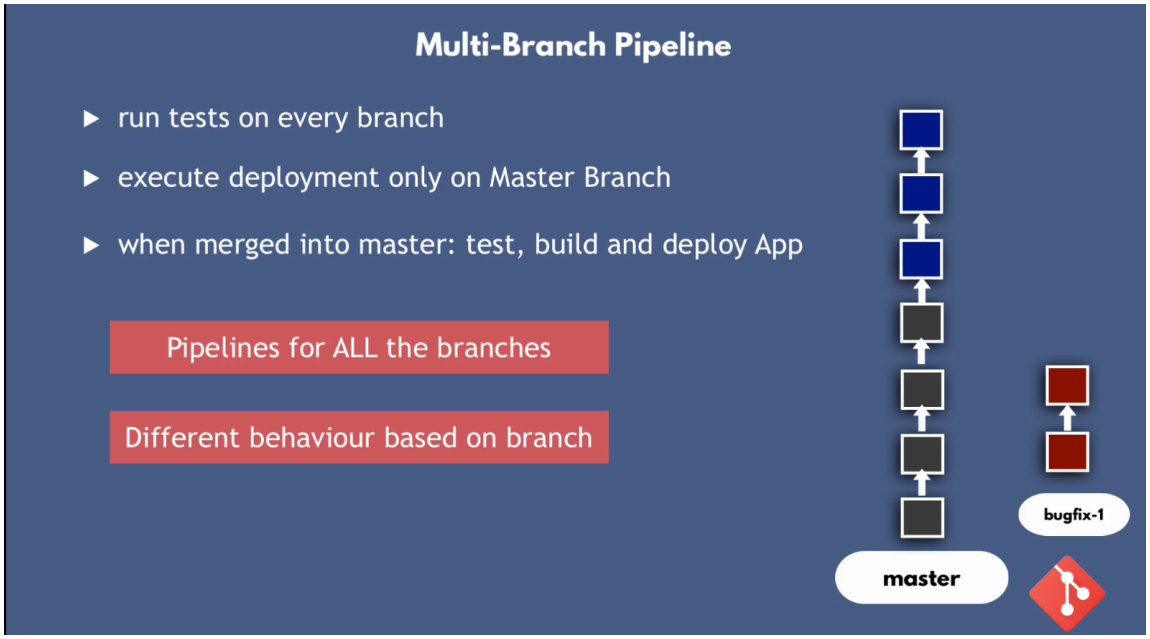
Types of Parameter:

- **string** (name, defaultValue, description)
- **choice** (name, choices, description)
- booleanParam (name, defaultValue, description)

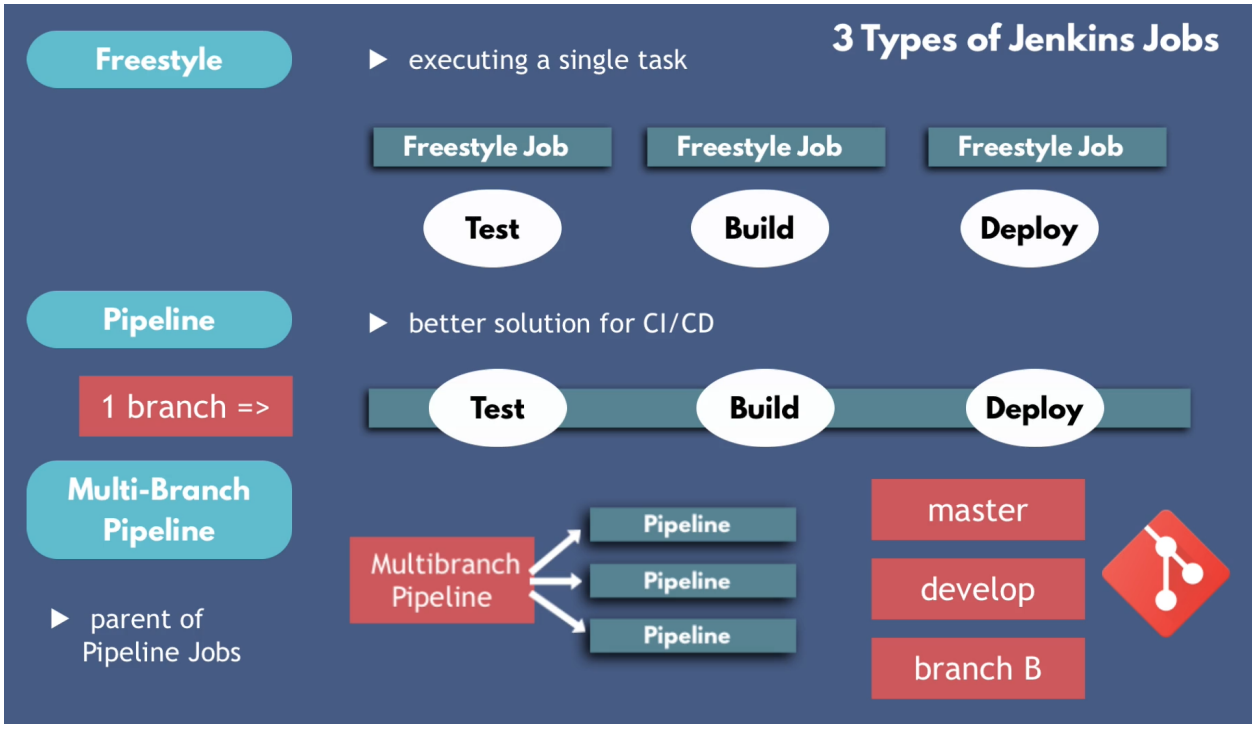
Input Parameter for User Input

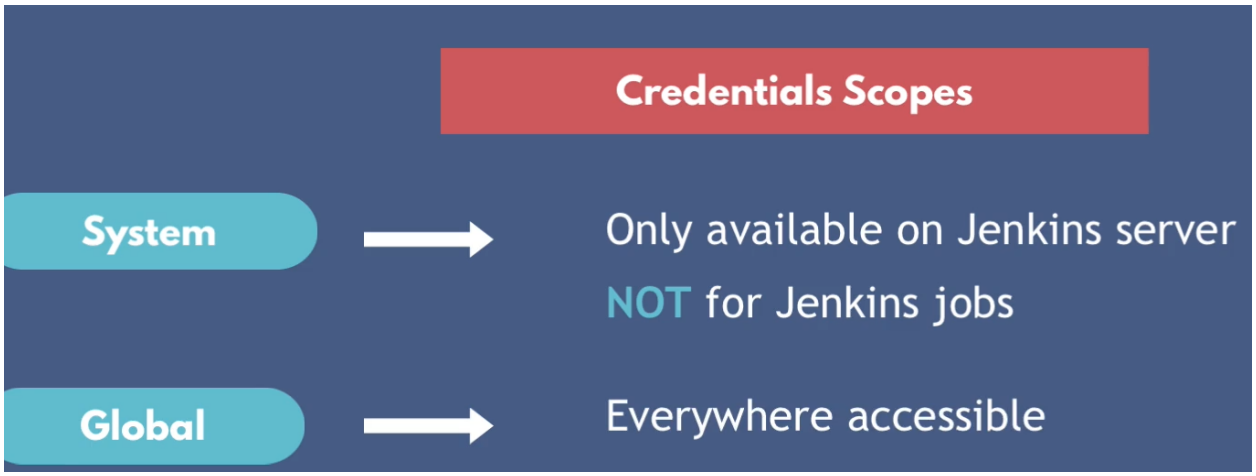


Multi-branch pipeline



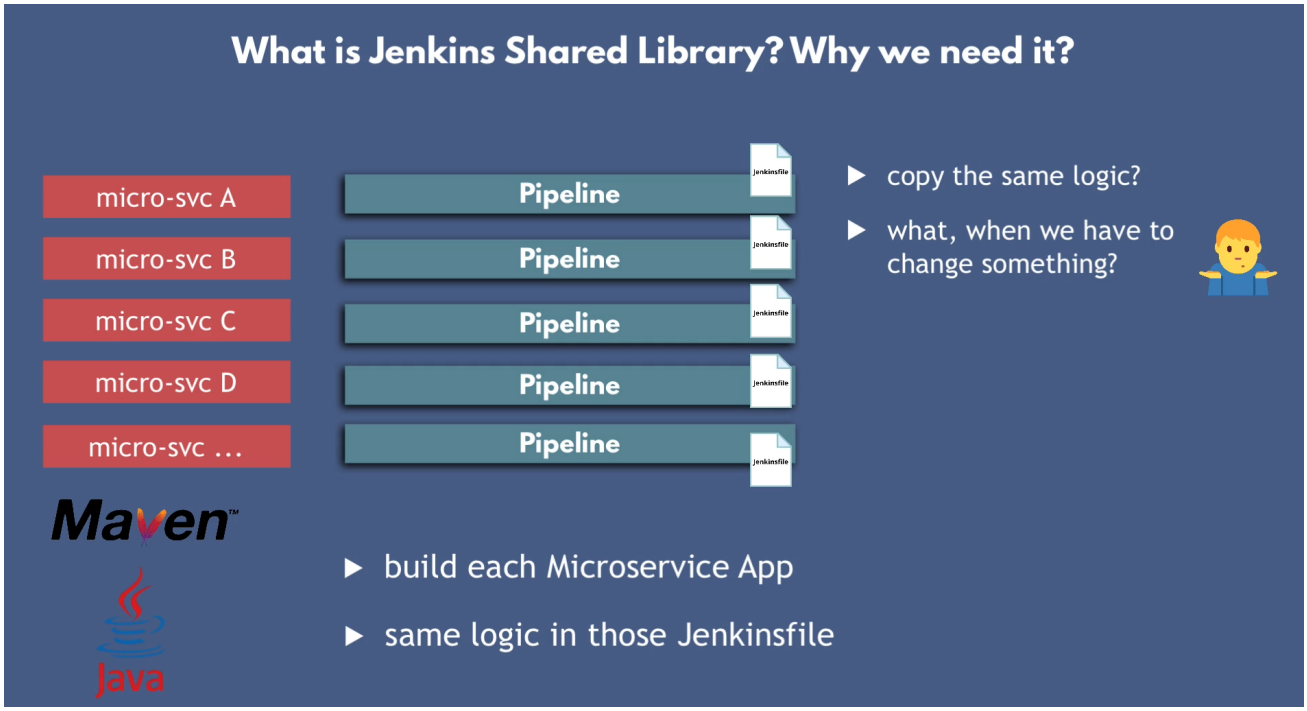
Jenkins will search all mentioned branches for the JenkinsFile and will try to automatically run all those pipelines

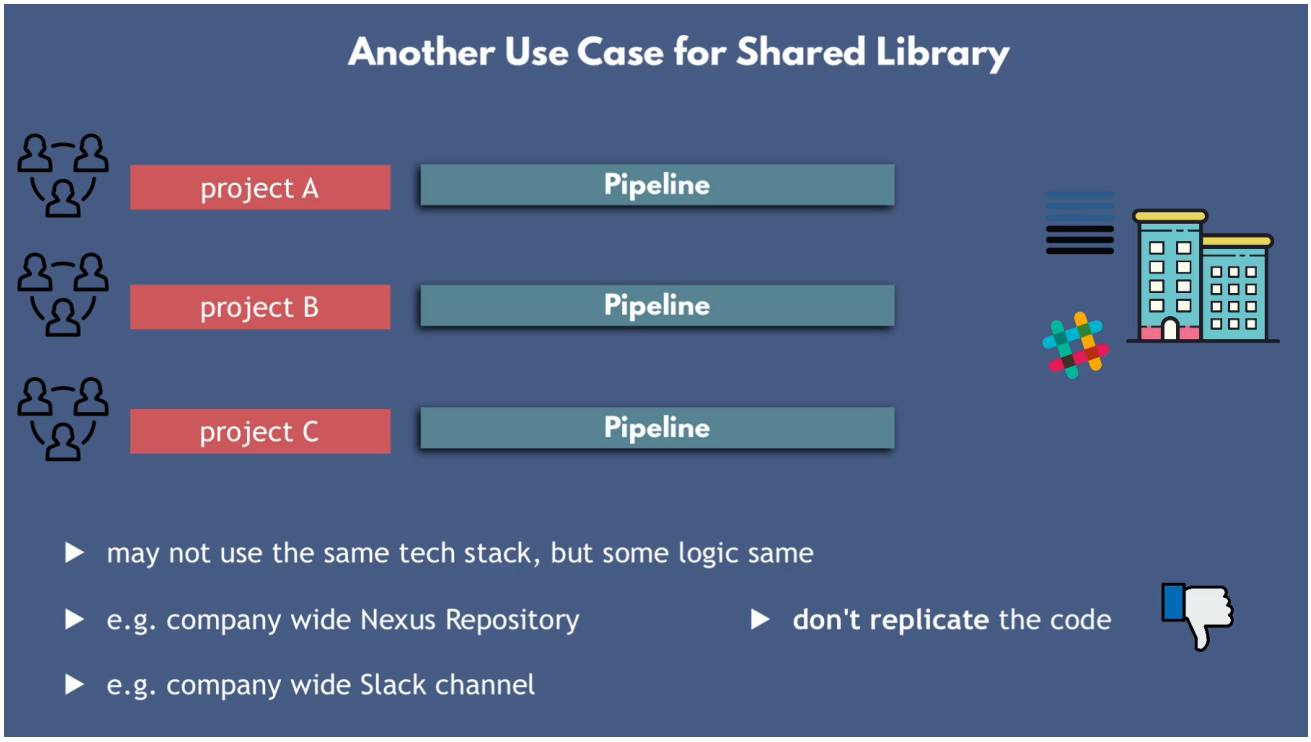




** multi-pipeline projects can also have their own credentials inside their scope that can't be visible from other pipelines out of it.

Jenkins shared library





Create the Shared Library

- Create Repository
- Write the Groovy Code
- Make the Shared Library available globally or for project
- Use the Shared Library in Jenkinsfile to extend the Pipeline

Structure of Shared Library

vars folder

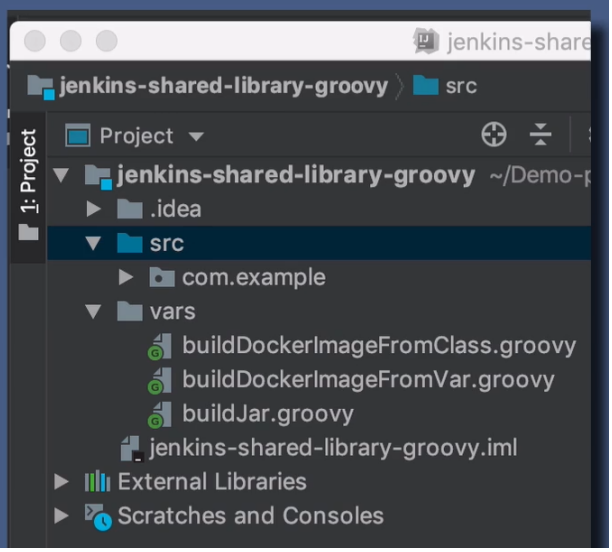
- functions that we call from Jenkinsfile
- Each function/execution step is its own Groovy file

src folder

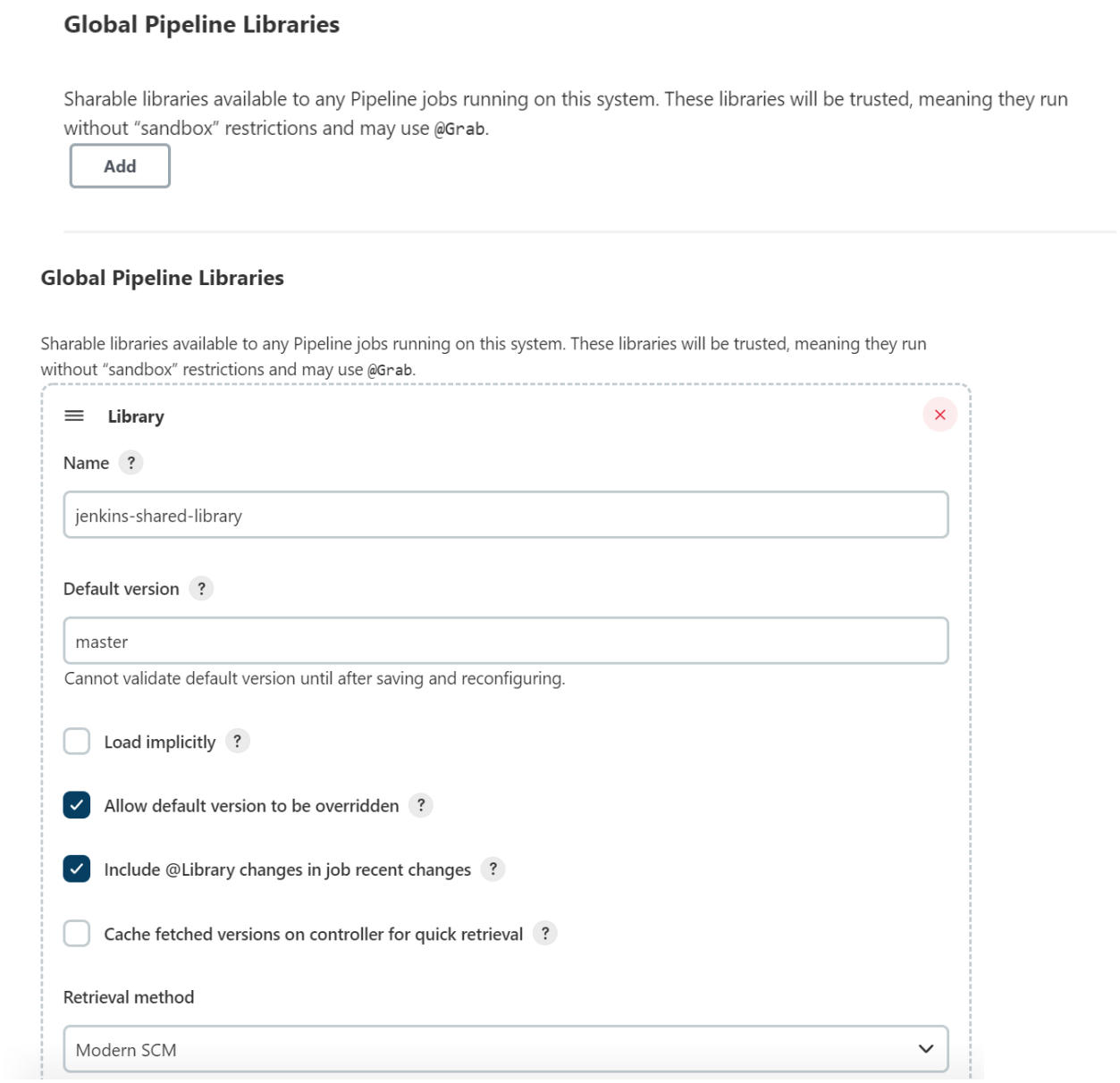
- helper code

resources folder

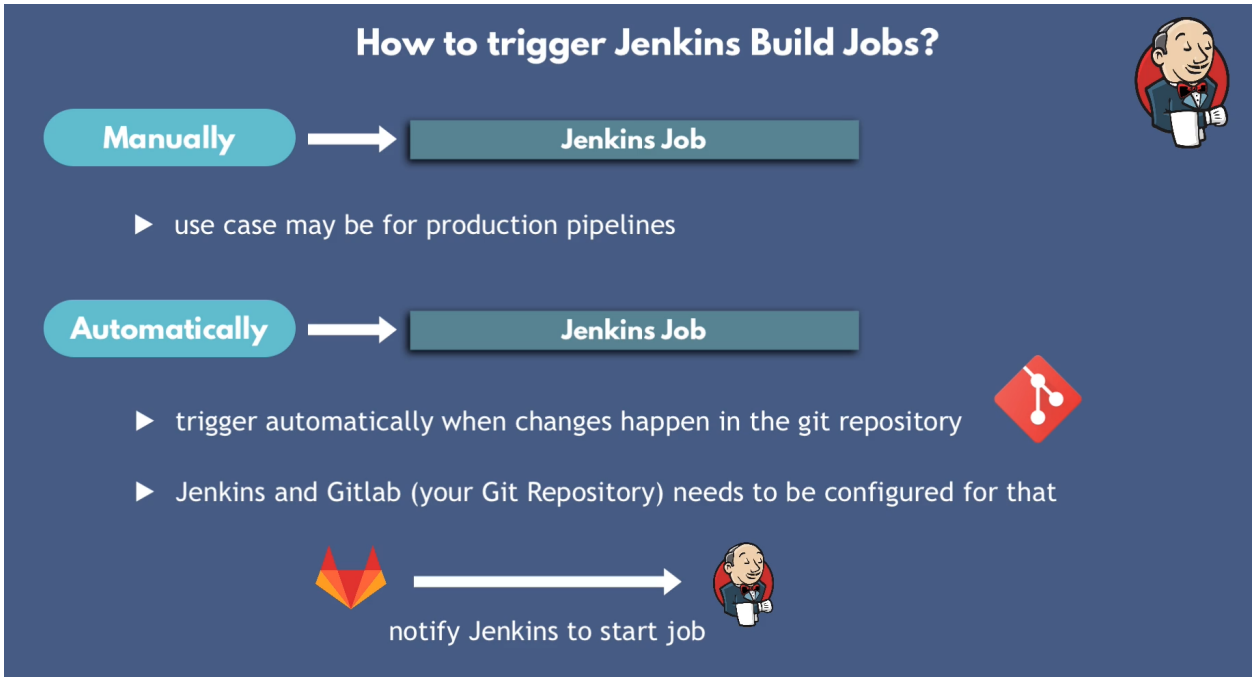
- use external libraries
- non groovy files

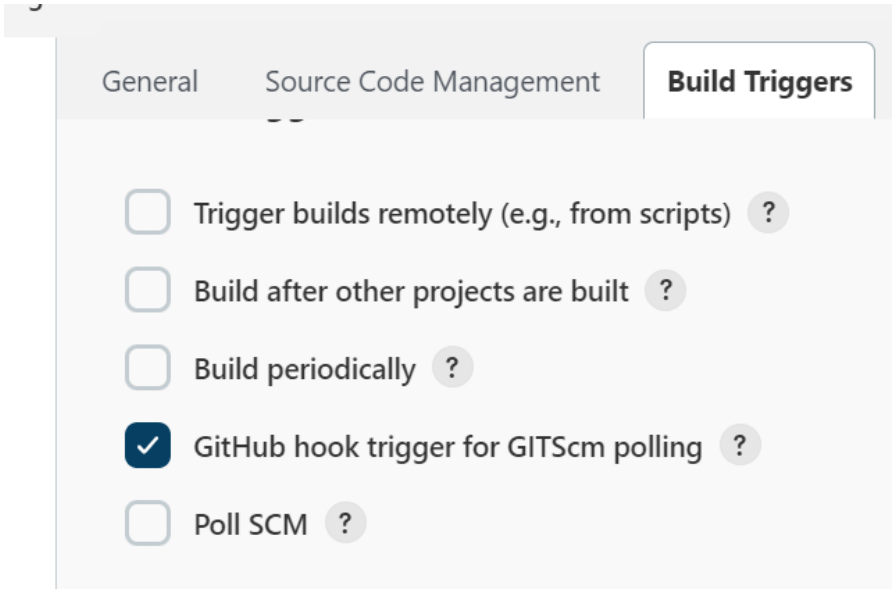


To add a shared library go to configure system -> global pipeline libraries -> add

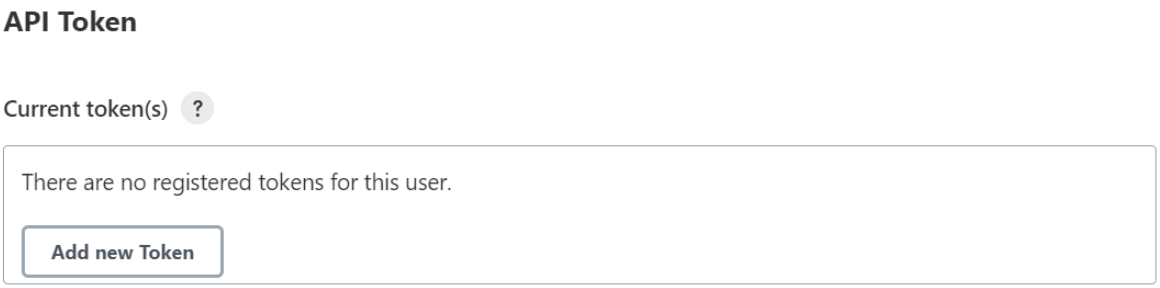


Trigger

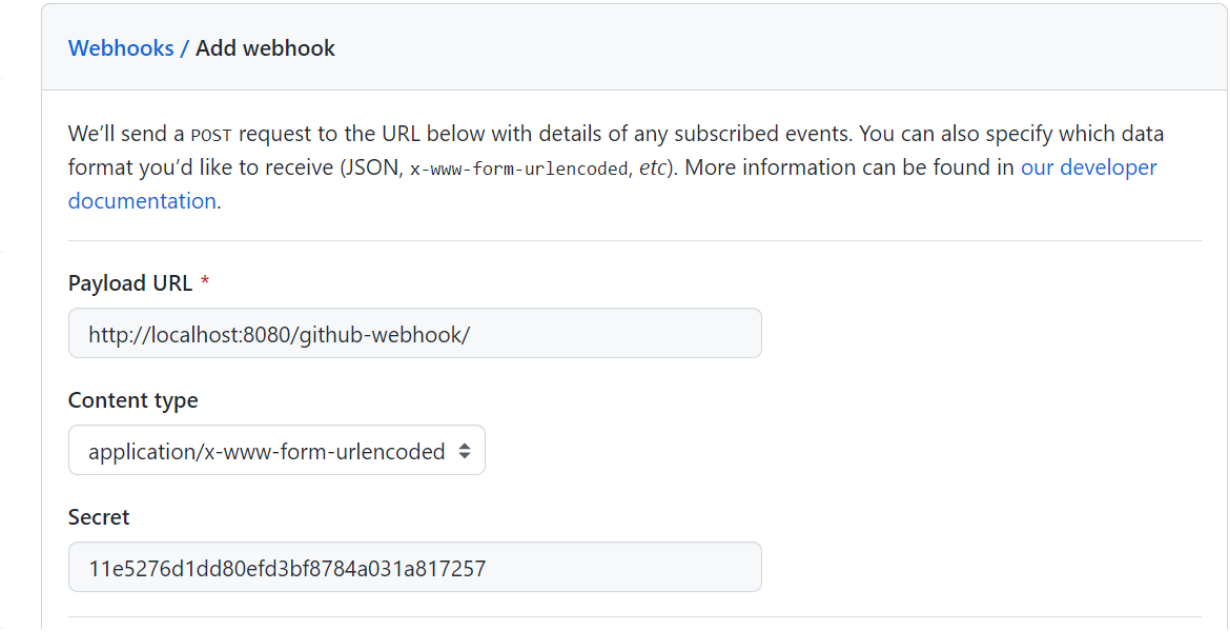




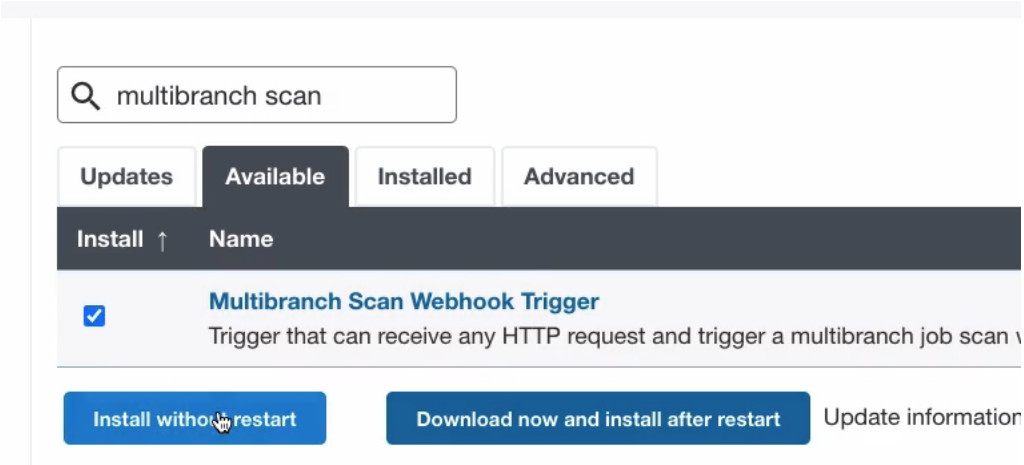
We need to have a token for connection to github webhooks

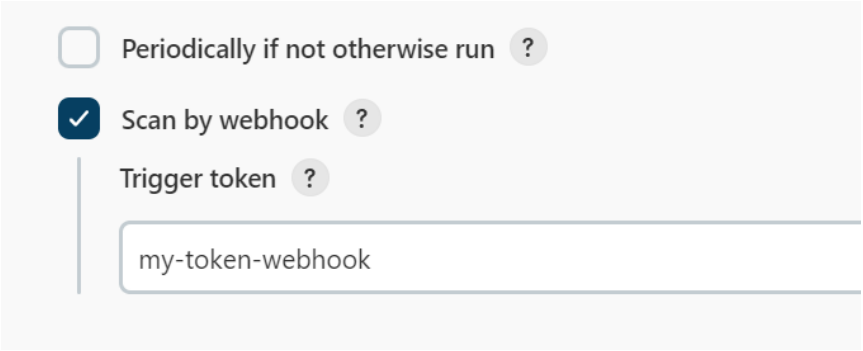


Put server address of jenkins and the token jenkins generated in webhook section of github project, https://github.com/Babak-H/gitplay/settings/hooks



We need this plugin for webhooks of multi-branch pipelines





The token to match with webhook token. Receive any HTTP request, JENKINS_URL/multibranch-webhook-trigger/invoke?token=[Trigger token] If a token match, than a multibranch scan will bi triggered.

This is the url that we should use with our github webhook
http://localhost:8080/multibranch-webhook-trigger/invoke?token=my-token-webhook

Application Versioning

- For Java files, package to update/read pom.xml file for version
- For JS applications, package to read from package.json and increment it
- Same for gradle…

Version incrementation is part of cicd pipeline