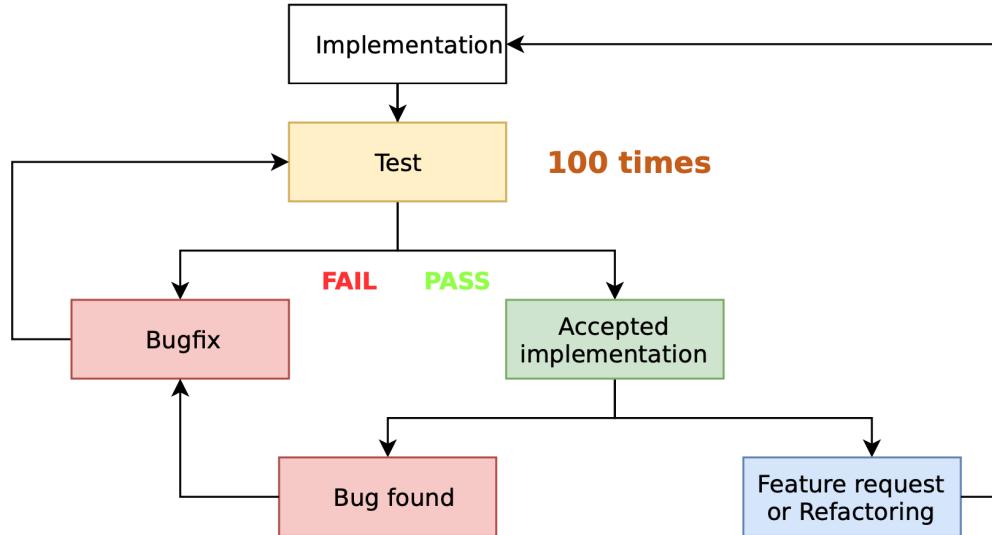


## Life cycle of a function



## Checking for None values

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

```
assert var is None
```

Do *not* do this.

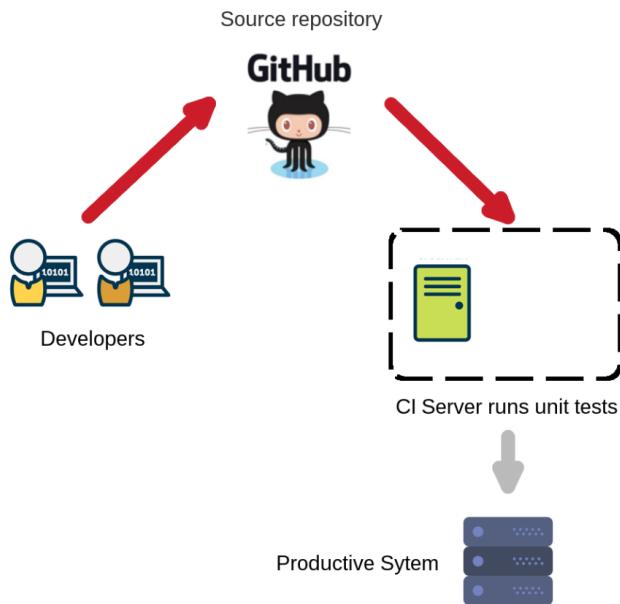
```
assert var == None
```

## Section 2: Test result

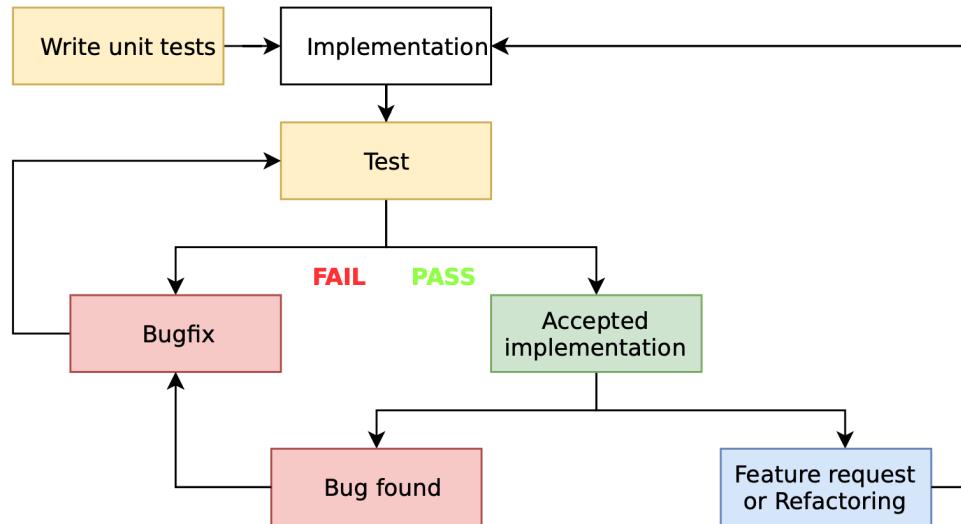
```
collecting ...
collected 3 items

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)



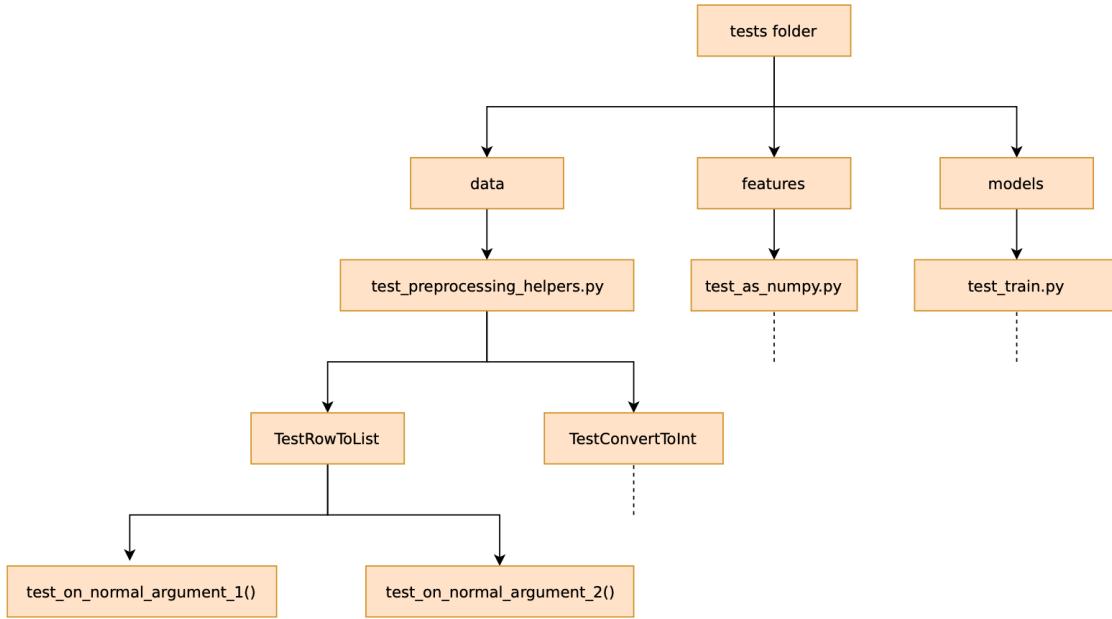
TDD

- 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`  $\iff$  `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/
|-- data/                                 # Test suite: all tests live here
|   |-- __init__.py
|   |-- test.preprocessing_helpers.py    # Corresponds to module src/data/preprocessing_helpers.py
|-- features/
|   |-- __init__.py
|-- models/
|   |-- __init__.py
```



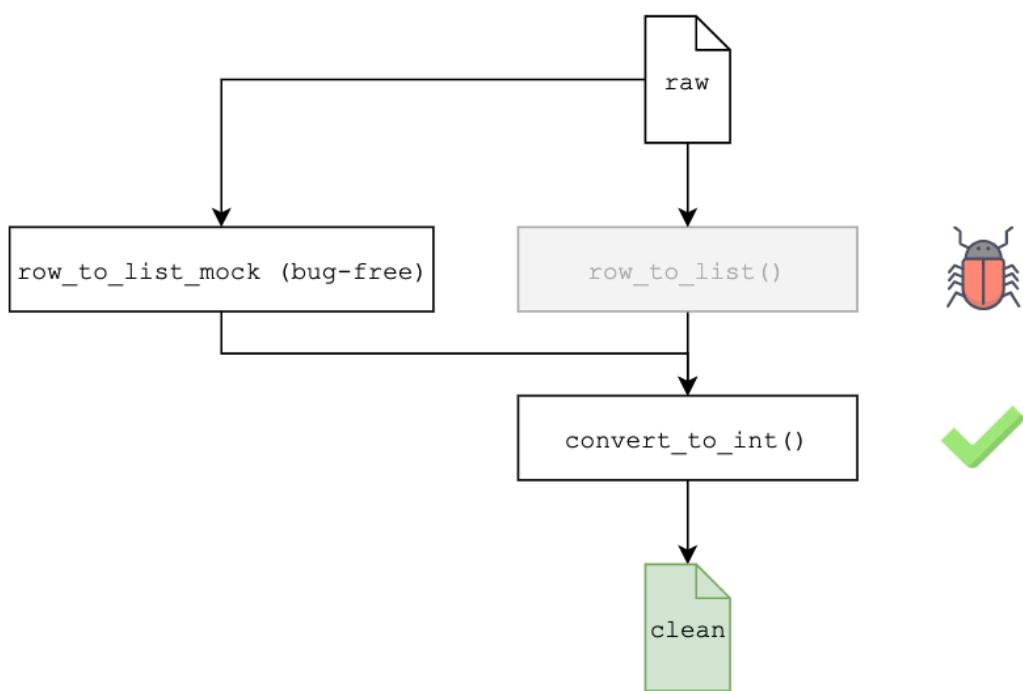
## Step 1: Modify the Travis CI configuration file

- 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

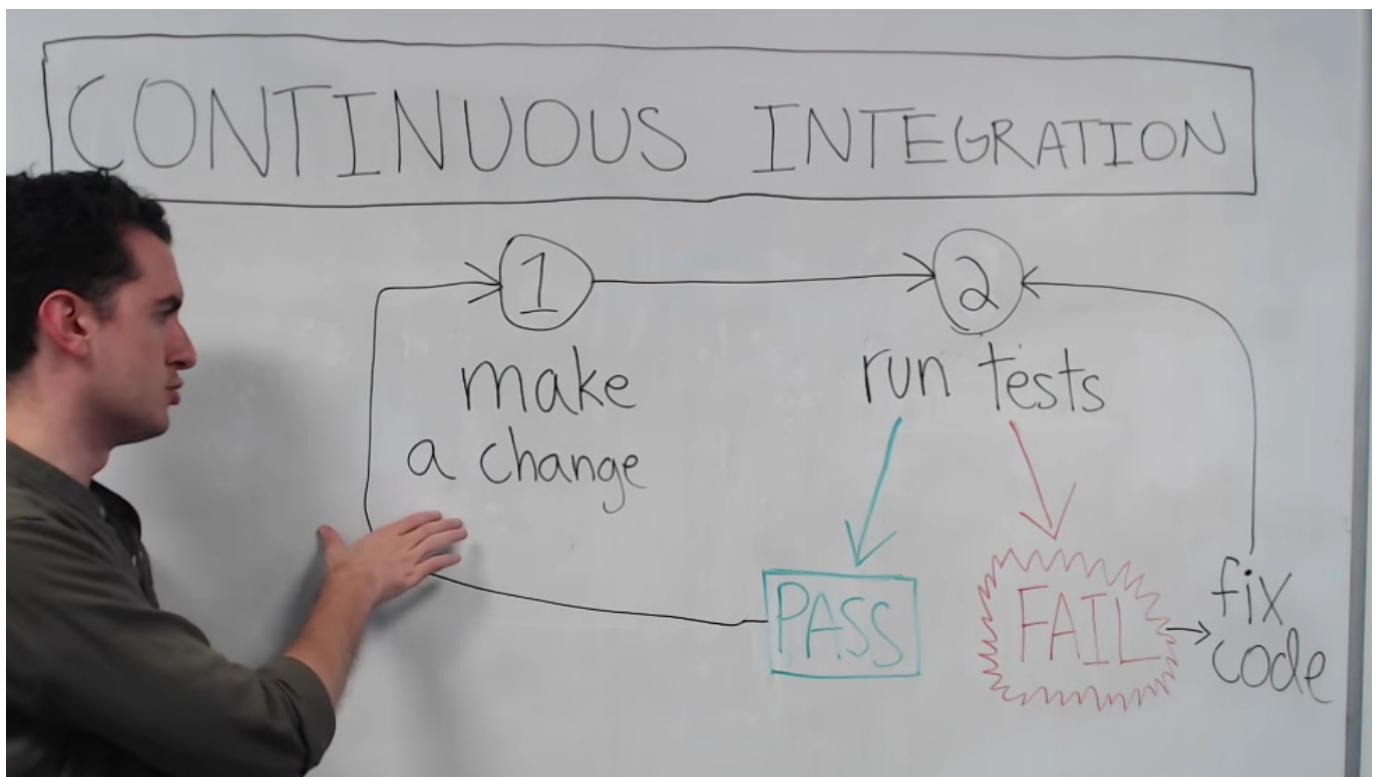
Continuous Integration : After every commit of the project there will be an automated test to make sure it works

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



## WHY CI?

1. SAFER

A. fewer bugs

2. FASTER

A. more features

B. better decisions

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

- integration

= AVOID CONFLICTS

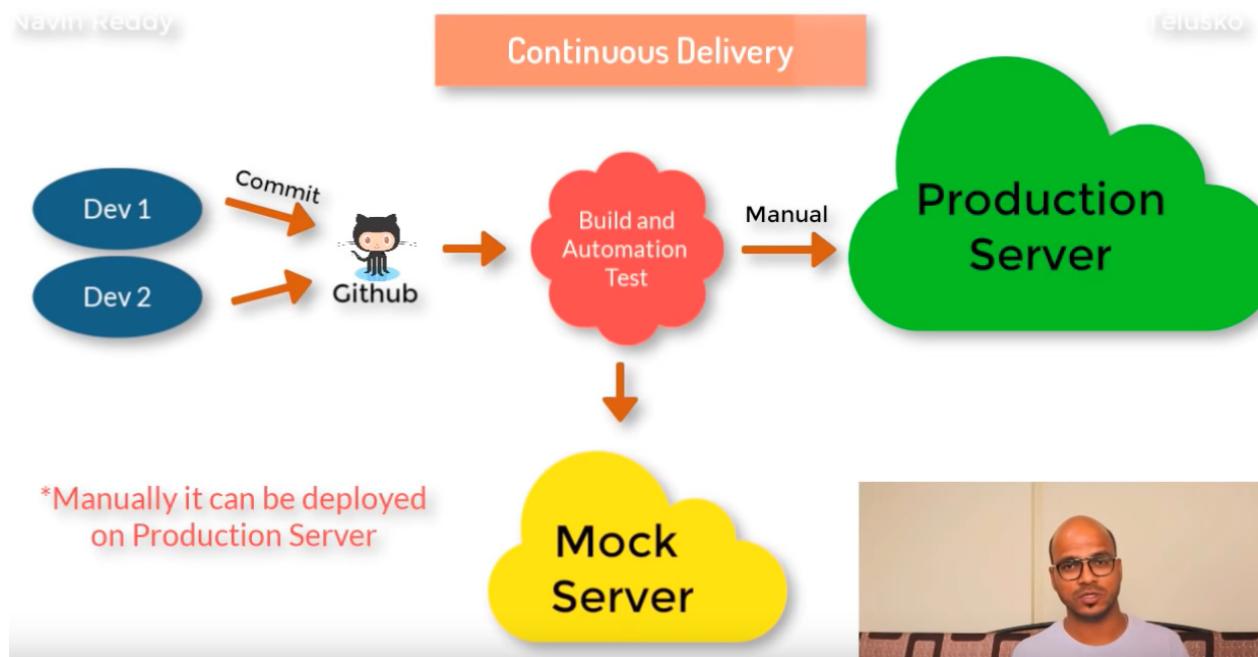
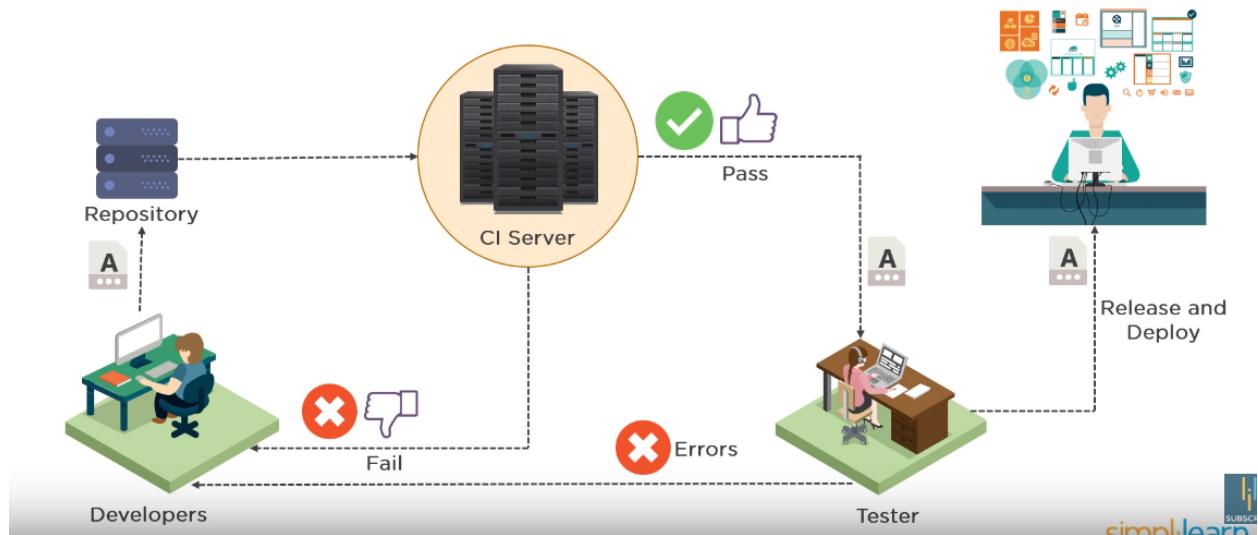
- delivery

= BE READY TO DEPLOY

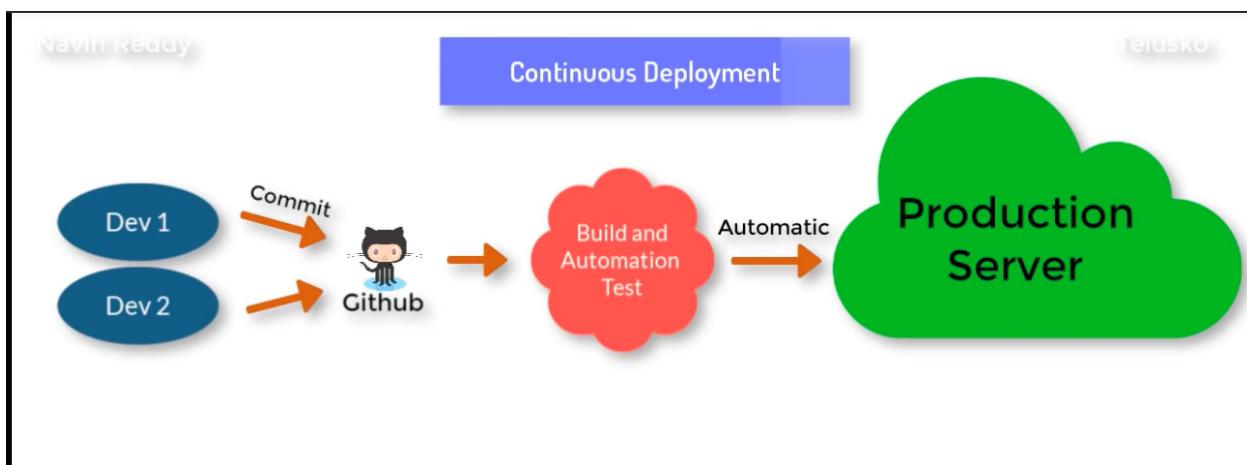
- deployment

= AUTOMATIC DEPLOY

## What is Continuous Integration?



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

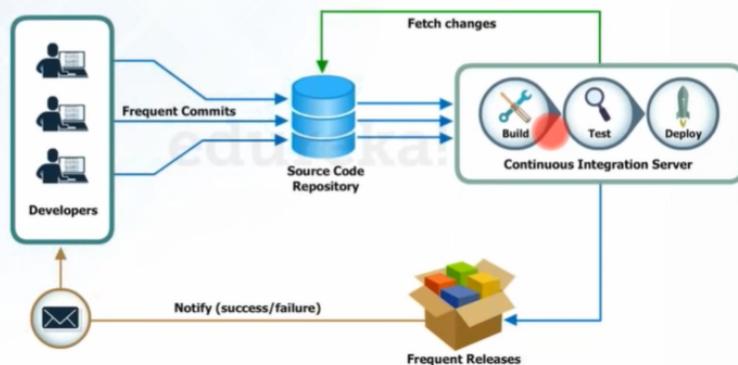


\*\* Jenkins is the most common tool for CI/CD

## Continuous Integration To The Rescue

**edureka!**

- ❑ Since after every commit to the source code an auto build is triggered and then it is automatically deployed on the test server
- ❑ If the test results shows that there is a bug in the code then the developers only have to check the last commit made to the source code
- ❑ This also increases the frequency of new software releases
- ❑ The concerned teams are always provided with the relevant feedback



## Continuous Integration To The Rescue

edureka!

### 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.

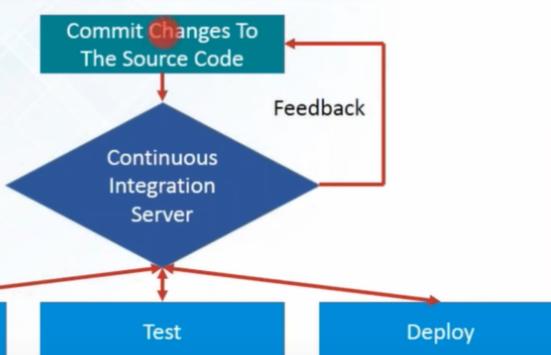
Developers know the test result of every commit made in the source code on the run

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.

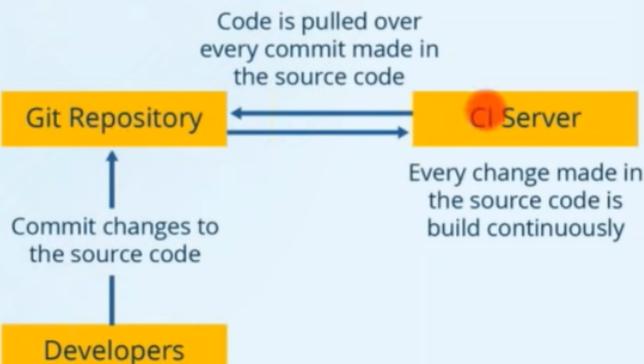


## Continuous Integration Case-Study: Nokia

Solution:



### Continuous Integration



## Continuous Integration Tools

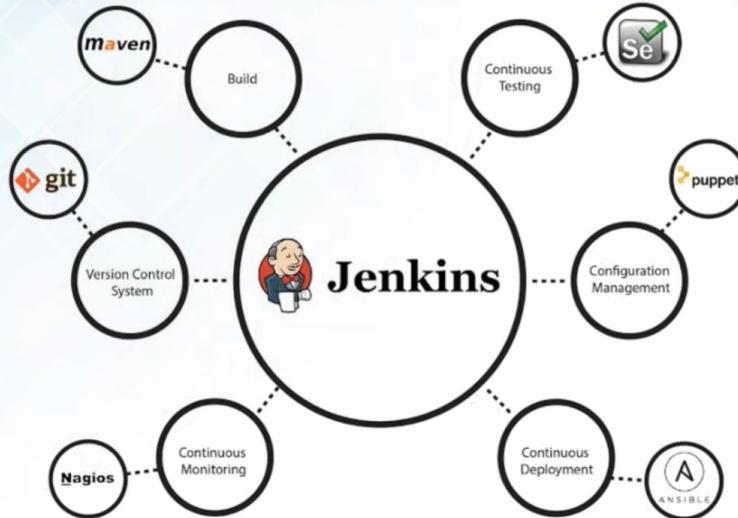
edureka!



# 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.



Dashboard [Jenkins] - Mozilla Firefox

localhost:8080

**Jenkins**

New Item

People

Build History

Manage Jenkins

My Views

Credentials

Welcome to Jenkins!

Please [create new jobs](#) to get started.

Build Queue

No builds in the queue.

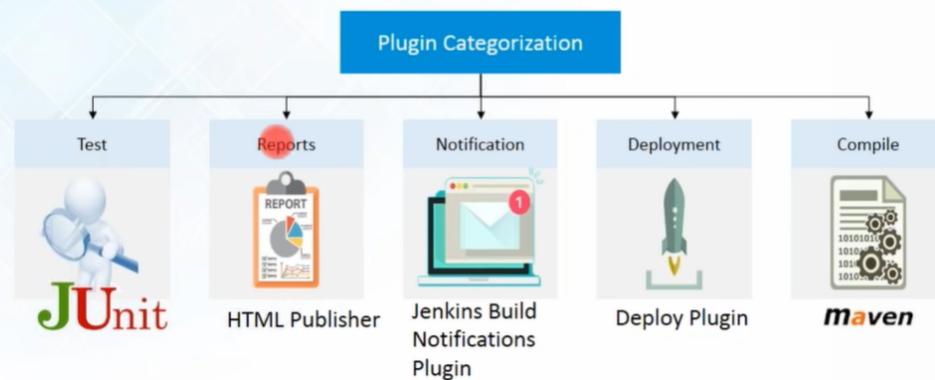
Build Executor Status

1 Idle

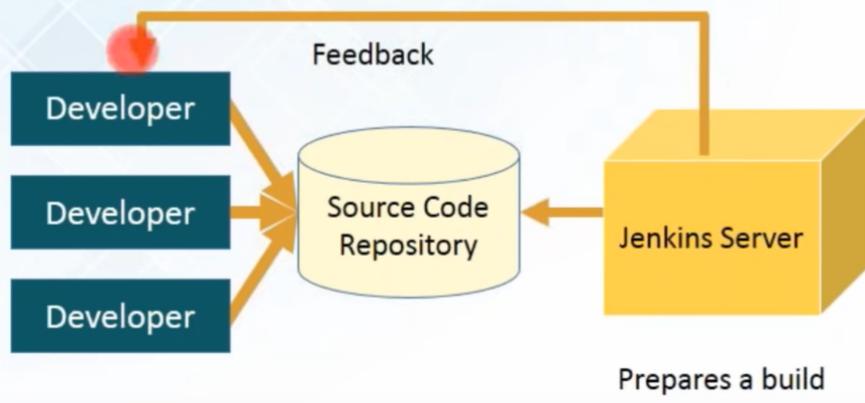
2 Idle

This screenshot shows the Jenkins dashboard interface. On the left is a sidebar with icons for New Item, People, Build History, Manage Jenkins, My Views, and Credentials. The main area features a "Welcome to Jenkins!" message with a call to action to "create new jobs". Below this are sections for the Build Queue (empty) and Build Executor Status (showing 1 and 2 idle executors). The top navigation bar shows the URL as localhost:8080.

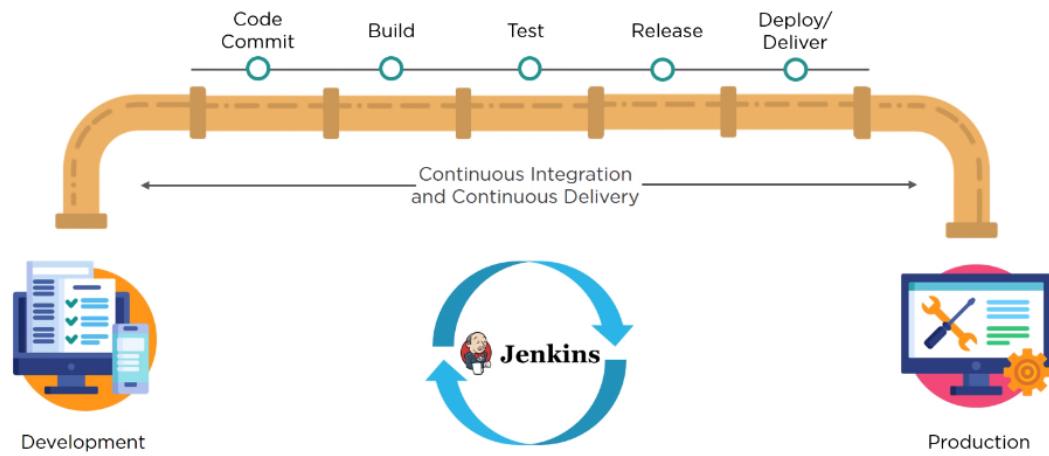
Jenkins supports plugins, which allow Jenkins to be extended to meet specific needs of individual projects



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



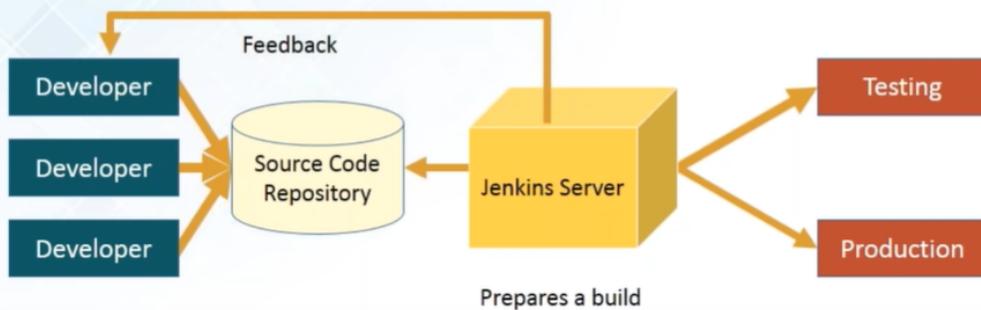
## Jenkins Pipeline



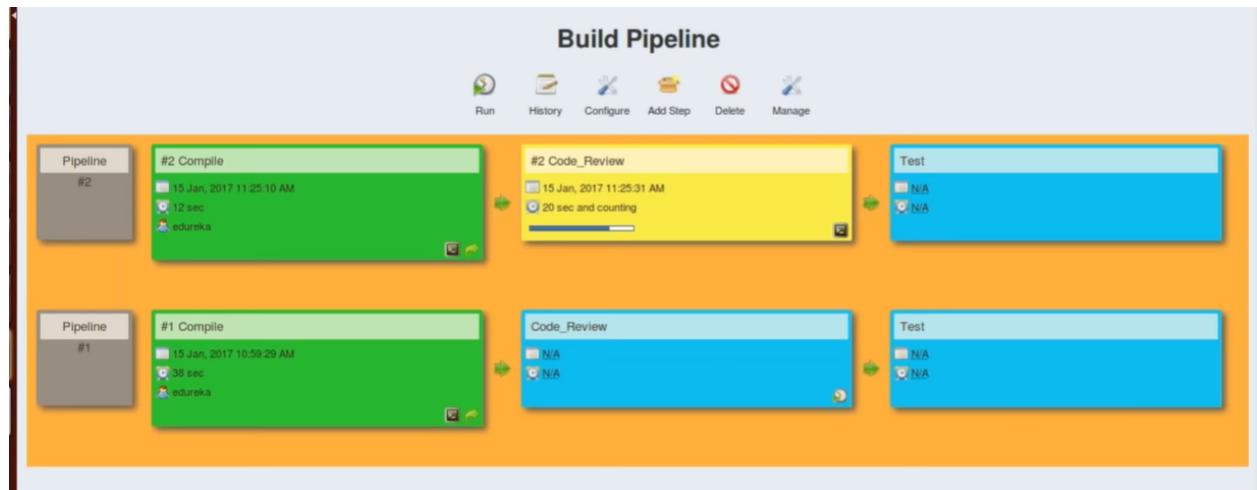
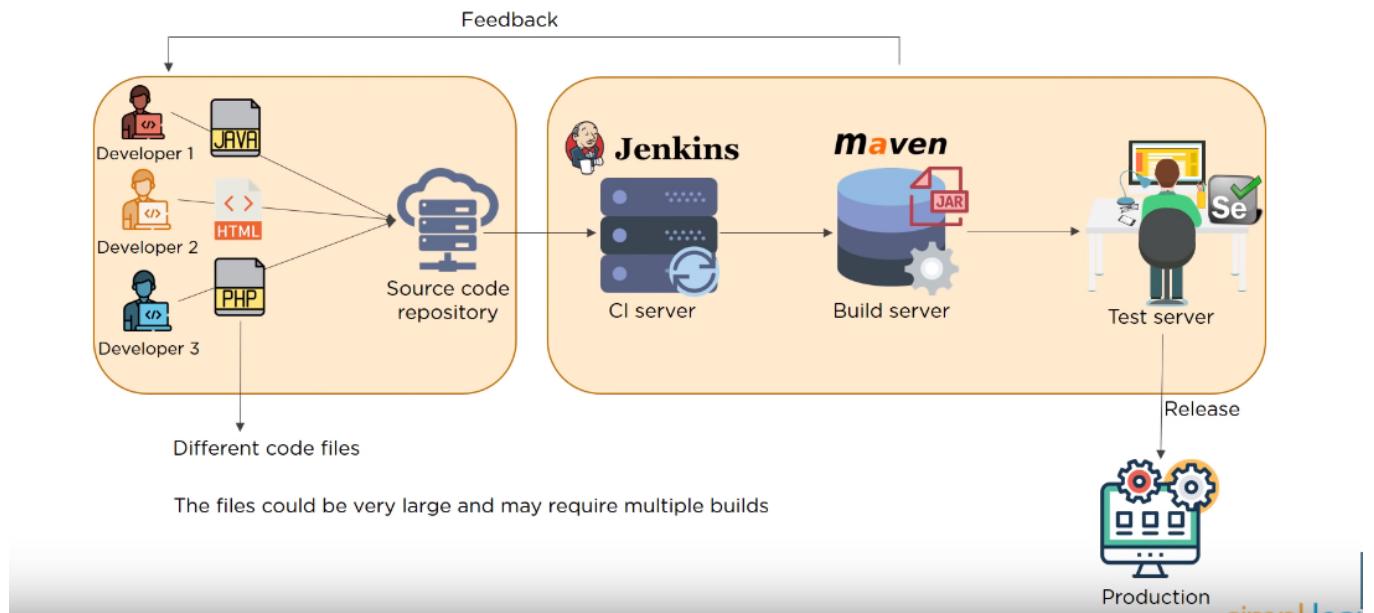
## Jenkins Example

ec

- 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

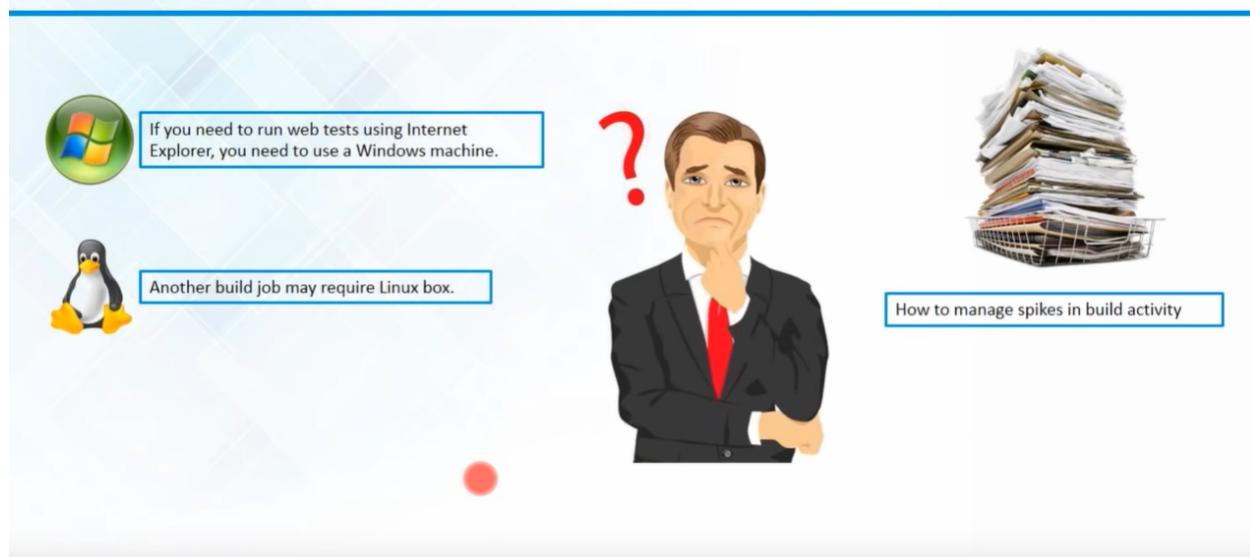


# Jenkins Architecture



## Shortcomings Of Single Jenkins Server

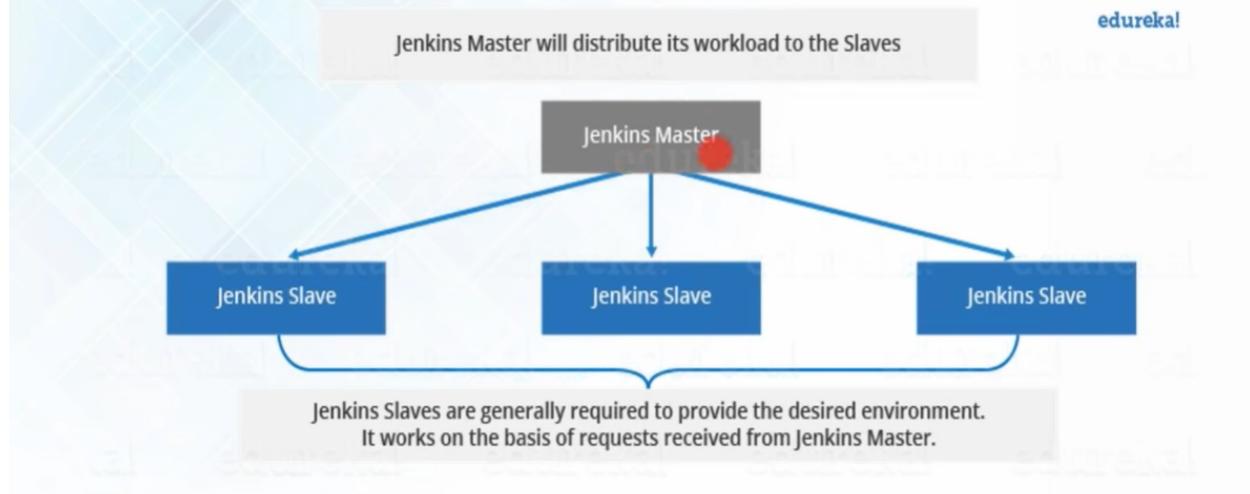
edureka!



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

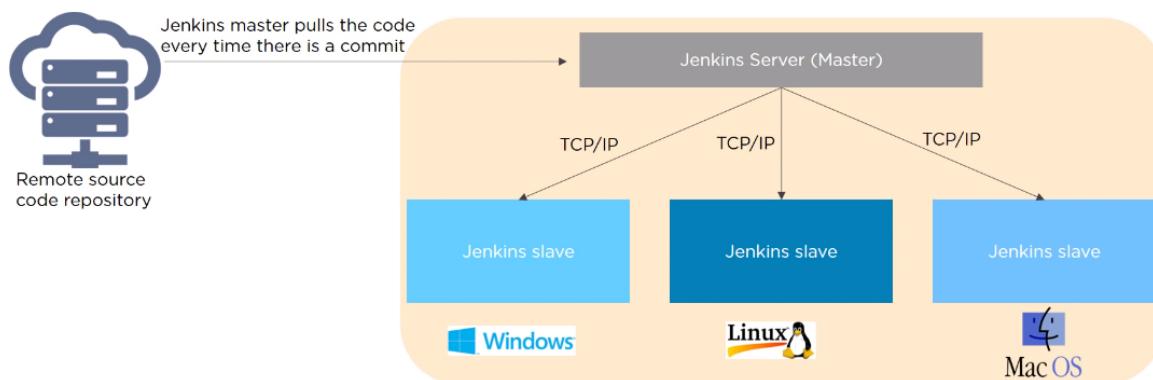
## Jenkins Distributed Architecture

edureka!

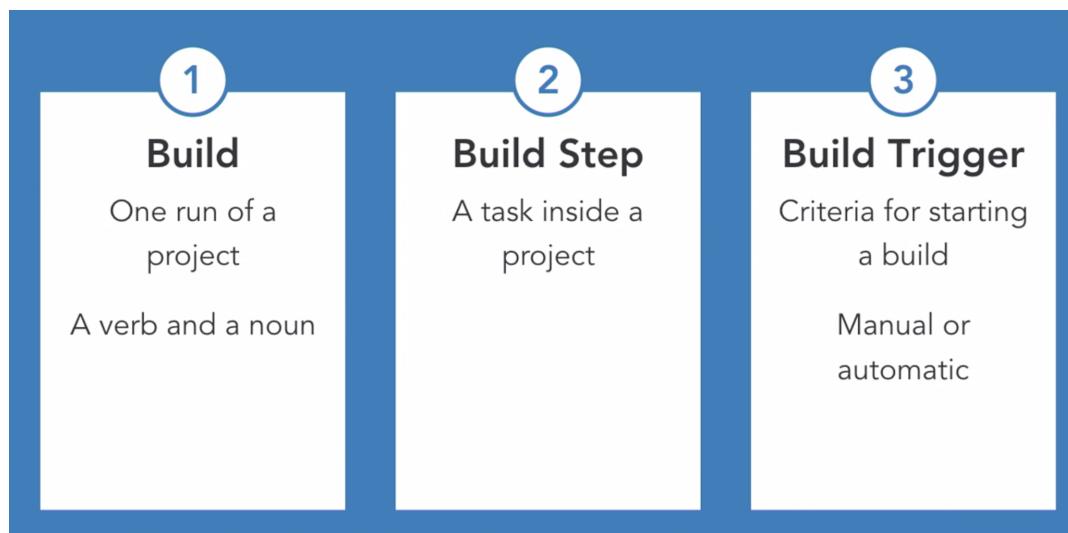


Jenkin slaves are java executables running on machines

## JENKINS Master-Slave Architecture

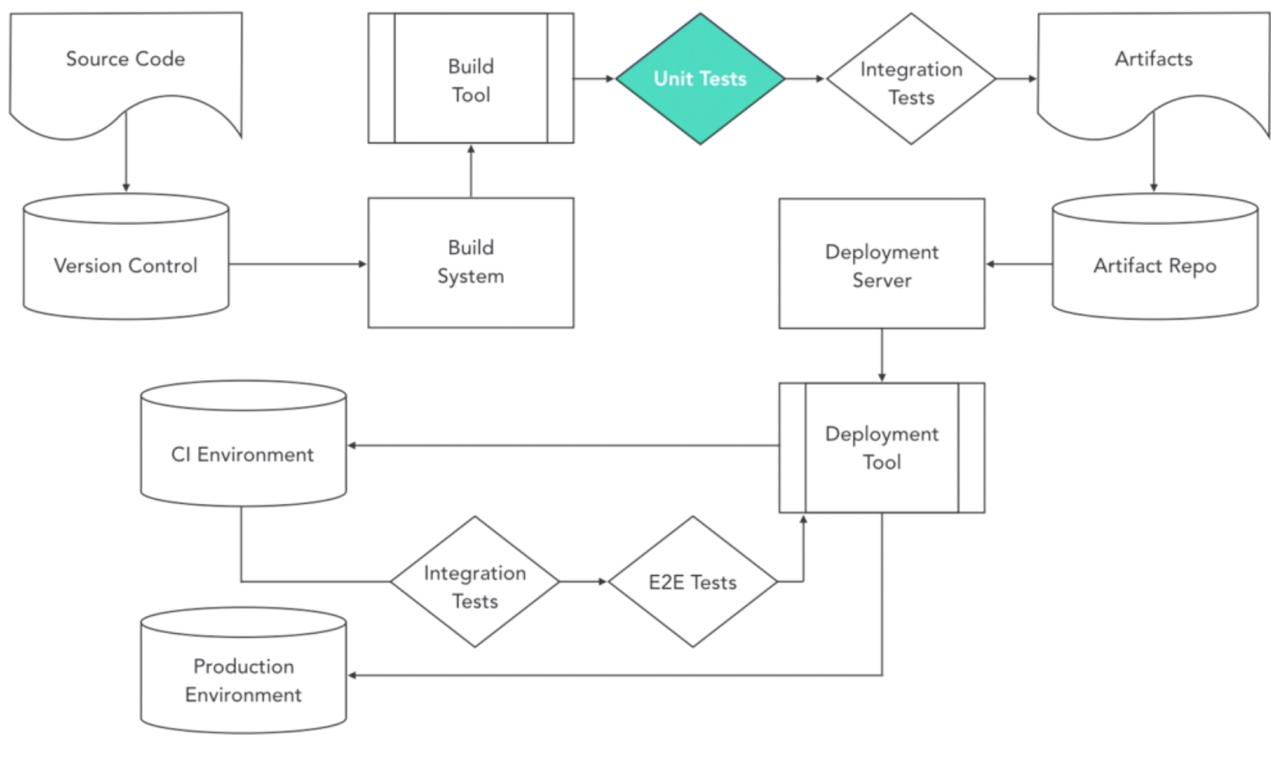


- Jenkins master distributes its workload to all the slaves
- On request from Jenkins master, the slaves carry out builds and tests and produce test reports



# 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

The screenshot shows a terminal window with a blue header bar containing the text "Docker run options:" and the Docker logo. Below the header, there are four entries, each with a yellow arrow pointing to its description:

- Expose port 8080** → By default runs on that port
- Expose port 50000** → Master / Slave Communication
- Run in detached mode** → run container in background
- Bind named volume** → persist data of Jenkins

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

## Types of Jenkins Projects

### Freestyle

simple, single tasks

e.g. run tests

### Pipeline

whole delivery cycle

e.g. test | build | ..

for a single branch

### Multibranch Pipeline

like pipeline

for multiple branches

The screenshot shows the Jenkins Global credentials (unrestricted) page. At the top, there is a navigation bar with links for Jenkins, Credentials, System, and Global credentials (unrestricted). Below the navigation is a search bar and user information. On the left, there are links for Back to credential domains and Add Credentials. The main content area has a heading "Credential Scopes" with two entries: "System" pointing to "Only available on Jenkins server NOT for Jenkins jobs" and "Global" pointing to "Everywhere accessible".

Scope	Description
System	Only available on Jenkins server NOT for Jenkins jobs
Global	Everywhere accessible

The screenshot shows the Jenkins Global credentials (unrestricted) page. The "Kind" dropdown menu is open, displaying various credential types: Username with password (selected), SSH Username with private key, Secret file, Secret text, X.509 Client Certificate, Certificate, and Fingerprint. The main content area has a heading "Credential Types" with three entries: "Username & Password", "Certificate", and "Secret File". To the right, there is a note: "New types based on plugins".

Kind
Username with password
SSH Username with private key
Secret file
Secret text
X.509 Client Certificate
Certificate
Fingerprint

ID

Description

OK

**ID = Reference for your credentials**

## Credential Scopes

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

## Pipeline Syntax

### Scripted

- first syntax
- Groovy engine
- advanced scripting capabilities, high flexibility
- difficult to start

### Declarative

- recent addition
- easier to get started, but not that powerful
- pre-defined structure

## Required Fields of Jenkinsfile

```
Ψ dev | / Jenkinsfile
1 pipeline {
2     agent any
3     stages {
4         stage("build") {
5             steps {
6                 }
7             }
8         }
9     }
10    }
11    }
12    }
13    }
14    }
15    }
16    node {
17        // groovy script
18    }
```

- "pipeline" must be top-level
- "agent" - where to execute
- "stages" - where the "work" happens
- "stage" and "steps"

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

The screenshot shows the Jenkins interface for a pipeline named 'my-pipeline' under the 'master' branch, specifically build '#1'. The left sidebar has a 'Replay' tab selected. The main content area is titled 'Replay #1' and contains a section titled 'Main Script' with the following Groovy code:

```
1 pipeline {  
2     agent any  
3     stages {  
4         stage("build") {  
5             steps {  
6                 echo 'building the application....'  
7             }  
8         }  
9         stage("test") {  
10            steps {  
11                echo 'running tests'  
12            }  
13        }  
14    }  
15}
```

Below the script, there's a 'Pipeline Syntax' section with a 'Run' button.

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

## Restart #3 from Stage



## 2 ways to trigger build

**Push Notification**    **Version Control notifies Jenkins on new commit**

**Polling**

**Jenkins polls in regular intervals**

## 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



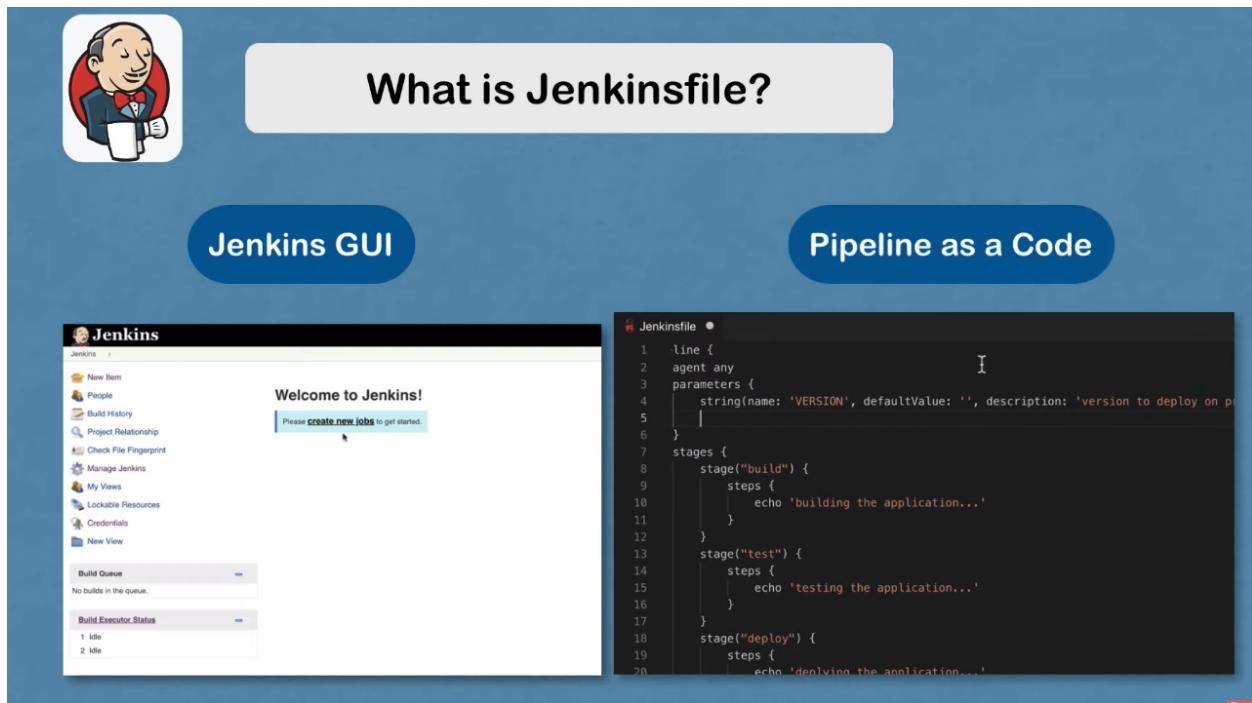
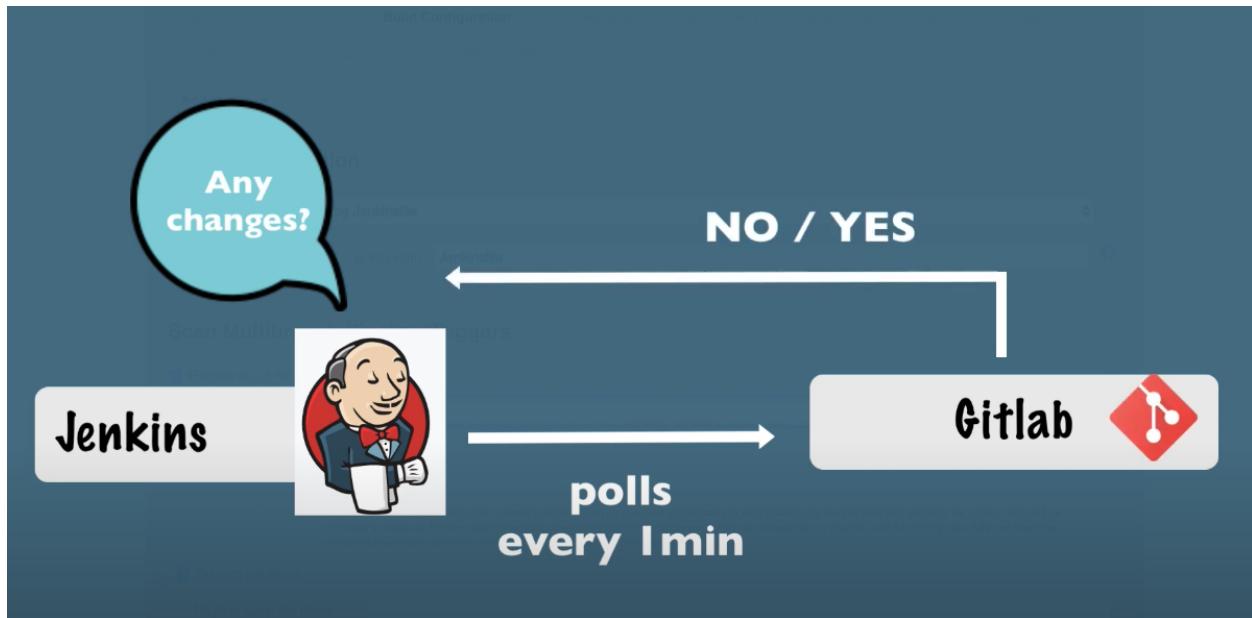
Or you can manually set a trigger for each X minutes or hours to check the git source code:

The screenshot shows the Jenkins Pipeline configuration page for a pipeline named "my-pipeline". The top navigation bar includes links for "Dashboard", "my-pipeline", "General", "Branch Sources", "Build Configuration", "Scan Multibranch Pipeline Triggers" (which is currently selected), "Orphaned Item Strategy", and "Health metrics".

The "Scan Multibranch Pipeline Triggers" section contains the following configuration:

- A checked checkbox labeled "Periodically if not otherwise run".
- An "Interval" dropdown menu set to "10 minutes".

Below this section is a header for "Orphaned Item Strategy".



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 of 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**

```

Jenkinsfile •
1 pipeline {
2     agent any
3     environment {
4         NEW_VERSION = '1.3.0'
5         SERVER_CREDENTIALS = credentials('')
6     }
7     stages {
8         stage("build") {
9             steps {

```

**Using Credentials in Jenkinsfile**

- 1) Define Credentials in Jenkins GUI
- 2) "credentials("credentialId")" binds the credentials to your env variable
- 3) For that you need "Credentials Binding" Plugin

through the UI which will be copied to the job workspace.

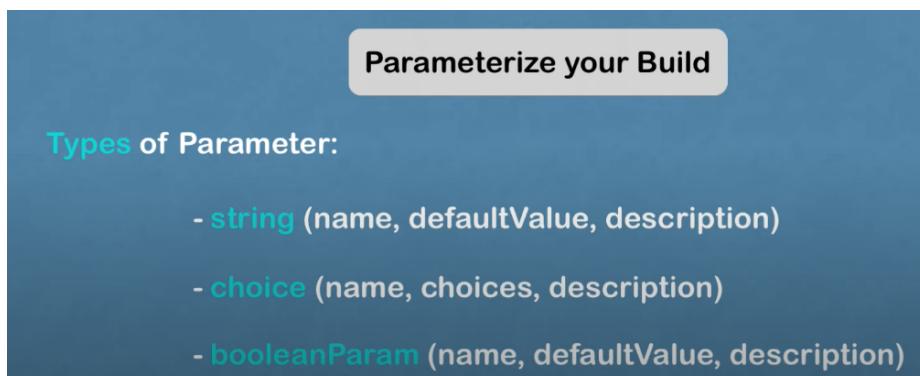
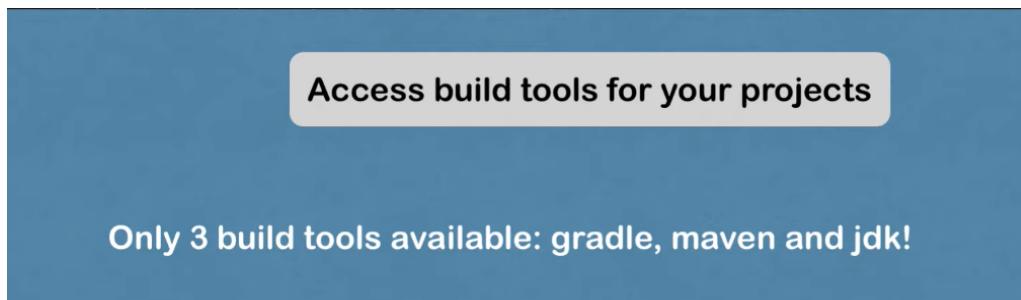
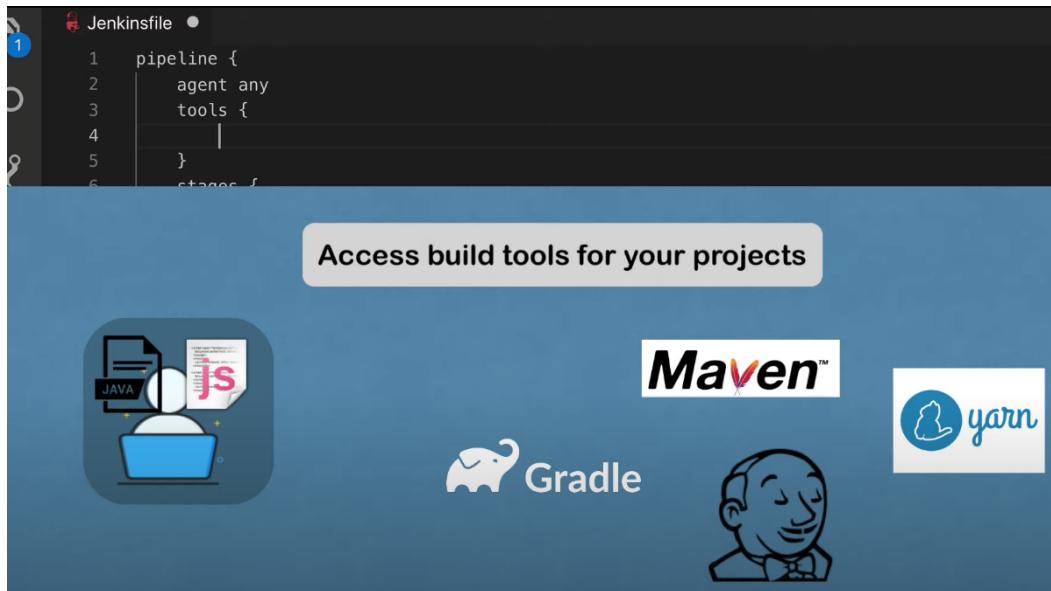
#### Credentials

This plugin allows you to store **credentials** in Jenkins.

#### Credentials Binding

Allows **credentials** to be bound to environment variables for use from miscellaneous build steps.

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



## 2 Roles for Jenkins App

Jenkins Administrator



Jenkins User



### Operations or DevOps teams

- ▶ administers and manages Jenkins
- ▶ sets up Jenkins cluster
- ▶ installs plugins
- ▶ backup Jenkins data

### Developer or DevOps teams

- ▶ creating the actual jobs to run workflows



## Create Job to automate your apps workflow

Java App



JavaScript App



- ▶ Java App with Maven Build Tool
- ▶ Run Tests
- ▶ Build Jar File



- ▶ Node App
- ▶ Run Tests
- ▶ Package and push to Repo

Maven needs to be available on Jenkins!

npm needs to be available on Jenkins



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
copy_reference_file.log
credentials.xml
fingerprints
hudson.model.UpdateCenter.xml
hudson.plugins.emaiext.ExtendedEmailPublisher.xml
hudson.plugins.git.GitTool.xml
hudson.plugins.gradle.Gradle.xml
hudson.tasks.Ant.xml
hudson.tasks.Maven.xml
jobs
logs
nodeMonitors.xml
nodes
org.jenkinsci.plugins.gitclient.JGitApacheTool.xml
org.jenkinsci.plugins.gitclient.JGitTool.xml
plugins
queue.xml
secret.key
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

## Global Tool Configuration

### Maven Configuration

Default settings provider

Use default maven settings



Default global settings provider

Use default maven global settings



Installing needed plugins

## Plugin Manager

Updates

Available

Installed

Advanced

Q node



Install Name ↴

Released

Role-based Authorization Strategy 555.v8d194cc85b\_30

Security Authentication and User Management

12 days ago

Enables user authorization using a Role-Based strategy. Roles can be defined globally or for particular jobs or nodes selected by regular expressions.

NodeJS 1.5.1

npm

7 mo 3 days ago

NodeJS Plugin executes NodeJS script as a build step.

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.

Enter an item name

» Required field

 **Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

 **Pipeline**  
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

 **Multi-configuration project**  
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

 **Folder**  
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

 **Multibranch Pipeline**  
Creates a set of Pipeline projects according to detected branches in one SCM repository.

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

Add build step ▾

Filter

- Execute NodeJS script
- Execute Windows batch command
- Execute shell
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets
- Provide Configuration files
- Run with timeout
- Set build status to "pending" on GitHub commit

## Build section is where you can execute the script or commands

The screenshot shows the 'Build' tab selected in a configuration interface. There are two main sections:

- Execute shell**: A step with a command field containing "npm --version".
- Invoke top-level Maven targets**: A step with a Maven Version dropdown set to "maven-3.6" and a Goals input field containing "--version".

All +

S	W	Name ↓	Last Success	Last Failure	Last Duration
		first-job	15 sec #3	N/A	0.72 sec

Icon: S M L

Icon legend

Atom feed for all

Atom feed for failures

Atom feed for just latest builds

[↑ Back to Dashboard](#)

## Project first-job

[!\[\]\(3a826c315649e5ff8d9ba7aee7a8e49e\_img.jpg\) Status](#)

[!\[\]\(6c52b702f5bb101efc4b3234d01ee644\_img.jpg\) Changes](#)

[!\[\]\(b7315602bb91b3d742e7604f9ff807ab\_img.jpg\) Workspace](#)

[!\[\]\(24ce36ad8a1745263e2734b9313a9dc2\_img.jpg\) Build Now](#)

[!\[\]\(0d417cfc0d70f74bf5febdb4ffef61a7\_img.jpg\) Configure](#)

[!\[\]\(0f56135fbafc50fedfcfff94393fe0f3\_img.jpg\) Delete Project](#)

[!\[\]\(240c12821e227464ff6b7614924c0018\_img.jpg\) Rename](#)

[!\[\]\(5c15ba9377947e38b2e2544b77c612d5\_img.jpg\) Build History](#)

[trend ▾](#)



Workspace



Recent Changes

## Permalinks

- [Last build \(#3\), 27 sec ago](#)
- [Last stable build \(#3\), 27 sec ago](#)
- [Last successful build \(#3\), 27 sec ago](#)
- [Last completed build \(#3\), 27 sec ago](#)

[↑ Back to Project](#)



## Console Output

[!\[\]\(7295dc0df8330a929114fec392966ea6\_img.jpg\) Status](#)

[!\[\]\(68460ad283ba1377980bd2dc095e3b44\_img.jpg\) Changes](#)

[!\[\]\(f38df260f53225785e48a20a998ff142\_img.jpg\) Console Output](#)

[!\[\]\(b3cbf752c5bff88adb60e9b04f1d8ac4\_img.jpg\) View as plain text](#)

[!\[\]\(12faef1b512a1da2be8f7866ae86112f\_img.jpg\) Edit Build Information](#)

[!\[\]\(0cd4818170771b234ac4f297e7522283\_img.jpg\) Delete build '#3'](#)

```
Started by user babak
Running as SYSTEM
Building on the built-in node in workspace /var/jenkins_home/workspace/first-job
[first-job] $ /bin/sh -xe /tmp/jenkins9610053814565132058.sh
+ npm --version
7.5.2
[first-job] $ /var/jenkins_home/tools/hudson.tasks.Maven_MavenInstallation/maven-3.6/bin/mvn -D--version= --version
Apache Maven 3.6.3 (ceceddd343002696d0abb50b32b541b8a6ba2883f)
Maven home: /var/jenkins_home/tools/hudson.tasks.Maven_MavenInstallation/maven-3.6
Java version: 11.0.16, vendor: Eclipse Adoptium, runtime: /opt/java/openjdk
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "5.10.104-linuxkit", arch: "amd64", family: "unix"
Finished: SUCCESS
```

Connecting jenkins build to the repository

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

Git ?

Repositories ?

Repository URL ?  X

Credentials ?  V

+ Add

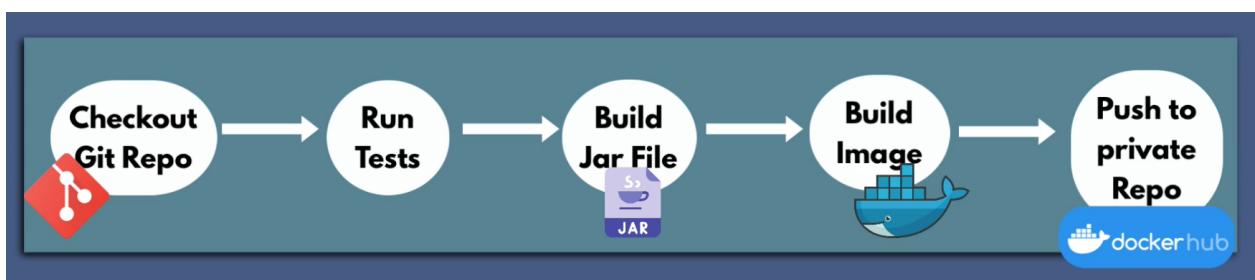
Advanced...

Add Repository

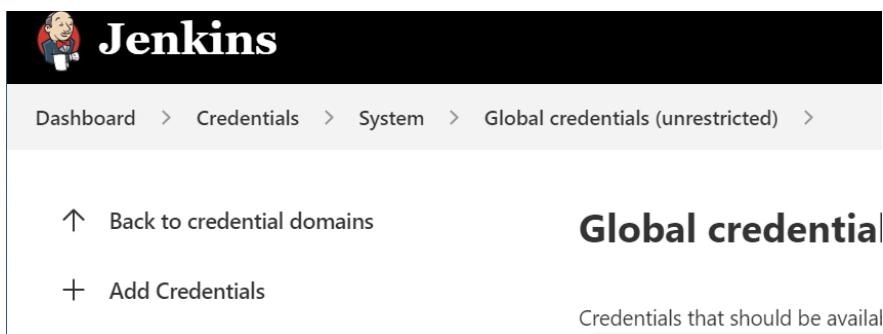
Branches to build ?

Branch Specifier (blank for 'any') ?  X

Add Branch

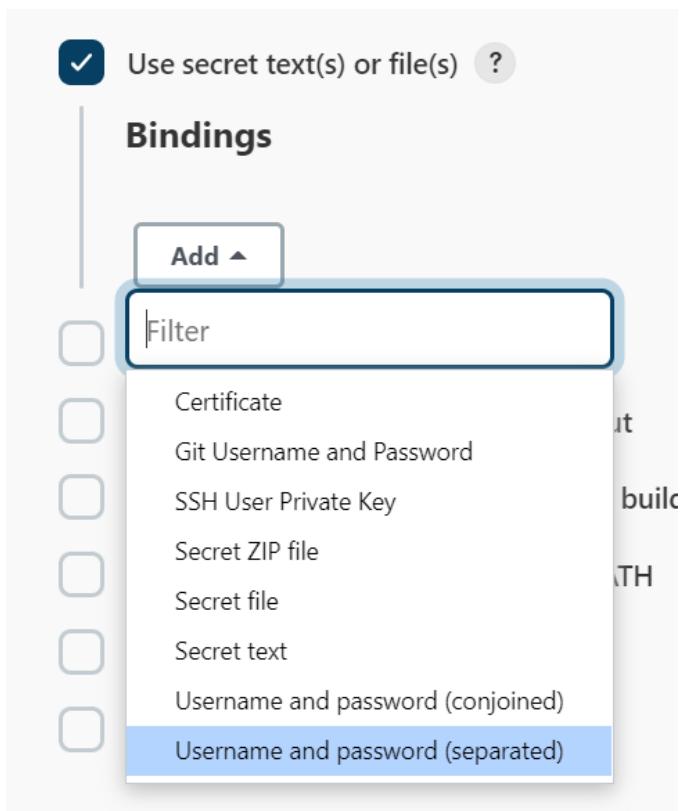


In here we can add the needed dockerHub credentials :



The screenshot shows the Jenkins interface for managing global credentials. The top navigation bar includes the Jenkins logo, 'Dashboard', 'Credentials', 'System', 'Global credentials (unrestricted)', and a back arrow. Below the navigation is a breadcrumb trail: 'Back to credential domains' and 'Add Credentials'. The main title is 'Global credentials'. A sub-header says 'Credentials that should be available'. On the left, there's a sidebar with a 'Bindings' section containing an 'Add' button and a 'Filter' input field. A dropdown menu lists various credential types: Certificate, Git Username and Password, SSH User Private Key, Secret ZIP file, Secret file, Secret text, Username and password (conjoined), and Username and password (separated). The 'Username and password (separated)' option is highlighted with a blue selection bar.

How to use those creds in the pipeline



The screenshot shows a Jenkins pipeline configuration step. It features a 'Bindings' section with an 'Add' button and a 'Filter' input field. A dropdown menu lists several credential types: Certificate, Git Username and Password, SSH User Private Key, Secret ZIP file, Secret file, Secret text, Username and password (conjoined), and Username and password (separated). The 'Username and password (separated)' option is selected, indicated by a blue selection bar.

## Freestyle to Pipeline

### Pipeline Jobs



- ▶ suitable for CI/CD
- ▶ scripting - pipeline as code
- ▶ using plugins
- ▶ executing tools



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

## Pipeline Syntax

### Scriped

- ▶ first syntax
- ▶ Groovy engine
- ▶ advanced scripting capabilities, high flexibility
- ▶ difficult to start

### Declarative

- ▶ more recent addition
- ▶ easier to get started, but not that powerful
- ▶ pre-defined structure

## What variables are available from Jenkins?

See at: </env-vars.html>

## Using Credentials in Jenkinsfile

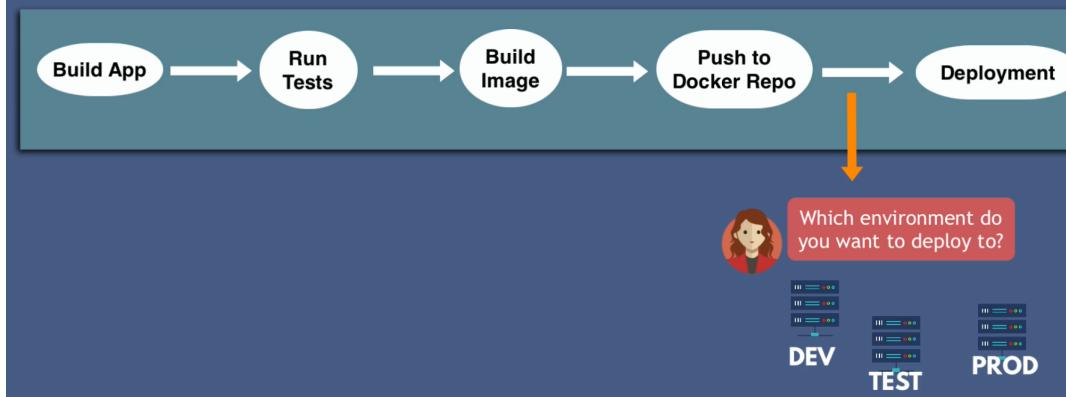
- 1) Define Credentials in Jenkins GUI (explained in next)
- 2) "credentials("credentialId")" binds the credentials to your env variable
- 3) For that you need "Credentials Binding" Plugin

## Parametrize your Build

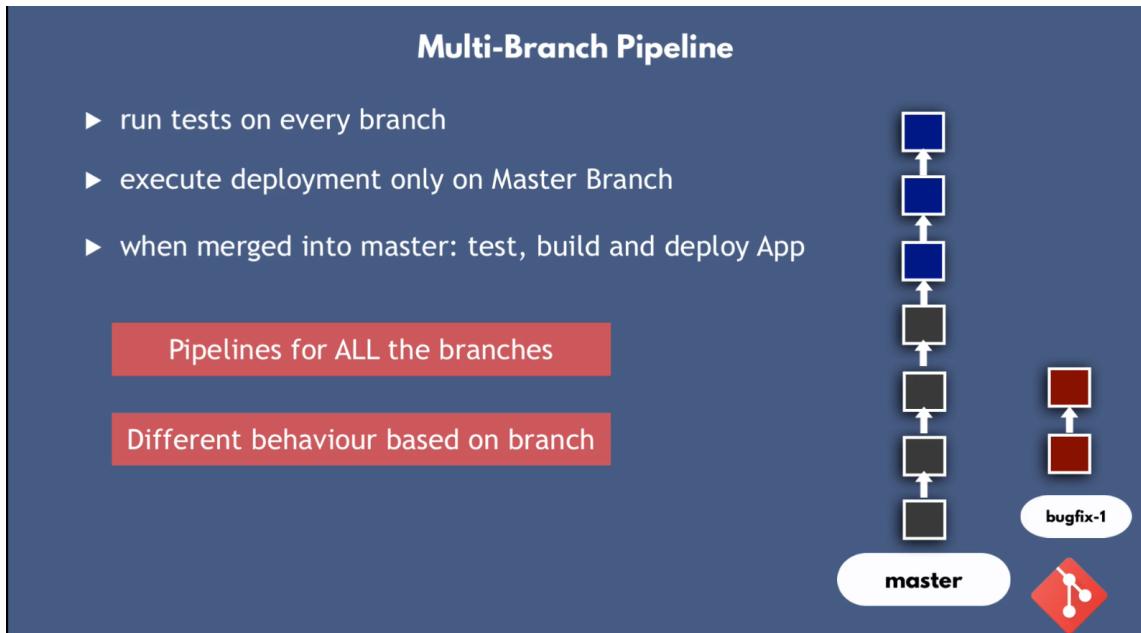
### Types of Parameter:

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

## Input Parameter for User Input



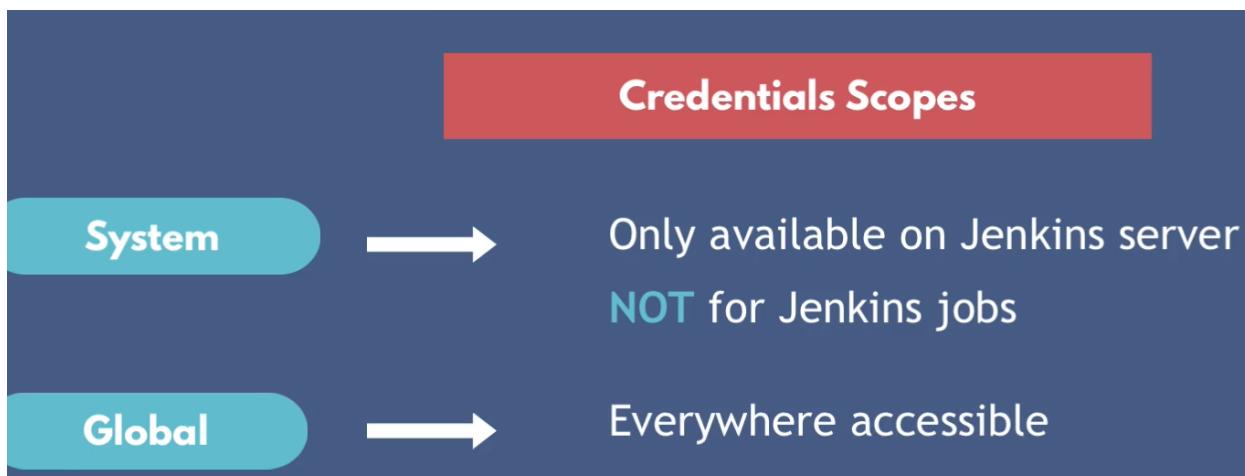
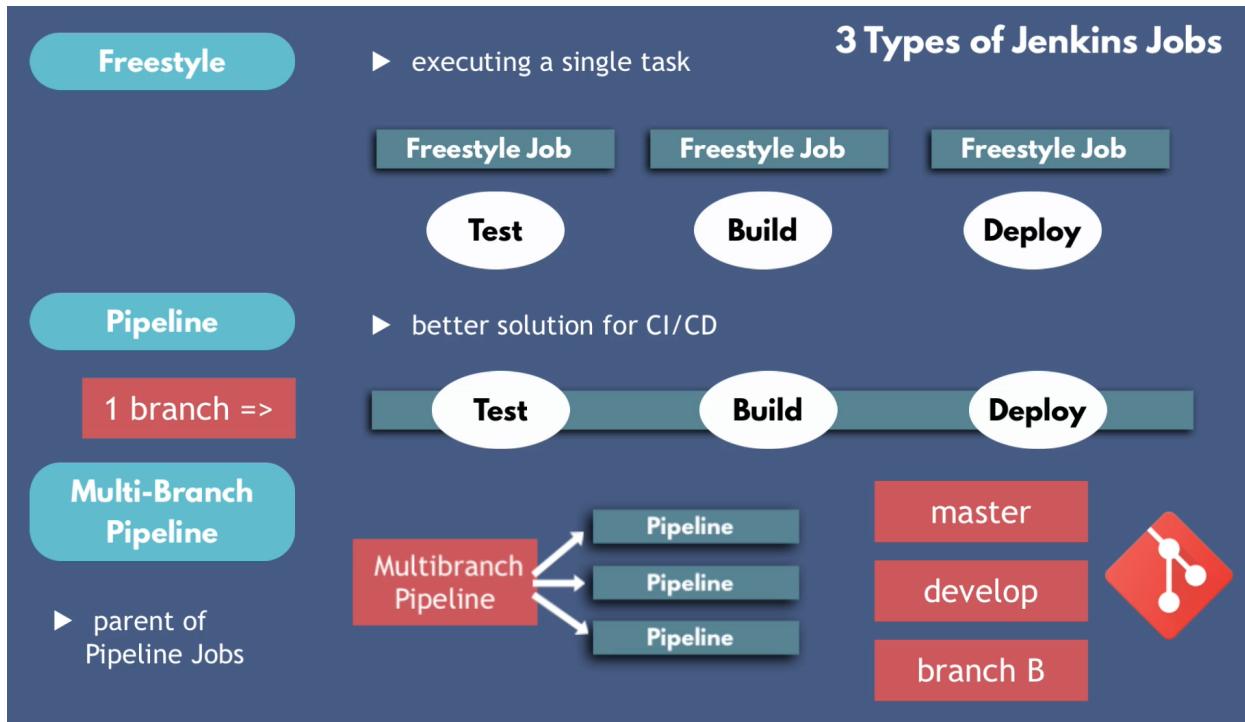
## Multi-branch pipeline



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

Scan Multibranch Pipeline Log Progress: ██████████ ✖

```
Started
[Tue Aug 16 20:12:39 UTC 2022] Starting branch indexing...
> git --version # timeout=10
> git --version # 'git version 2.30.2'
using GIT_ASKPASS to set credentials
> git ls-remote --symref -- https://gitlab.com/Babak-Gohardani/java-maven-app.git # timeout=10
> git rev-parse --resolve-git-dir /var/jenkins_home/caches/git-f4642414e6596c9c3740659daf219073/.git # timeout=10
Setting origin to https://gitlab.com/Babak-Gohardani/java-maven-app.git
> git config remote.origin.url https://gitlab.com/Babak-Gohardani/java-maven-app.git # timeout=10
```



\*\* 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

### What is Jenkins Shared Library? Why we need it?

The diagram illustrates a common challenge in Jenkins pipelines across multiple projects. On the left, five boxes represent different microservices: 'micro-svc A', 'micro-svc B', 'micro-svc C', 'micro-svc D', and 'micro-svc ...'. To the right of each service is a teal 'Pipeline' box containing a 'Jenkinsfile' icon. This visualizes the repetition of Jenkinsfile logic for each project. To the right of the pipelines, two bullet points highlight the inefficiencies:

- ▶ copy the same logic?
- ▶ what, when we have to change something?

**Maven™**  
  
**Java**

- ▶ build each Microservice App
- ▶ same logic in those Jenkinsfile

### Another Use Case for Shared Library

This diagram shows another scenario where multiple projects share a common Jenkins pipeline. Three boxes on the left are labeled 'project A', 'project B', and 'project C'. Each project has its own teal 'Pipeline' box with a 'Jenkinsfile' icon. To the right of the pipelines, there is a Jenkins logo icon and a building icon, suggesting a company-wide shared infrastructure. Below the pipelines, a list of reasons for using a shared library is provided.

- ▶ may not use the same tech stack, but some logic same
- ▶ e.g. company wide Nexus Repository
- ▶ e.g. company wide Slack channel
- ▶ don't replicate the code

## 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

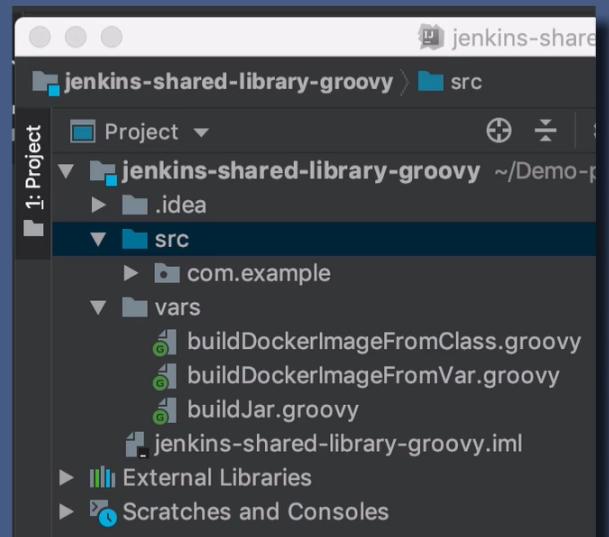
- ▶ functions that we call from Jenkinsfile
- ▶ Each function/execution step is its own Groovy file

### 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

## Global Pipeline Libraries

Sharable libraries available to any Pipeline jobs running on this system. These libraries will be trusted, meaning they run without "sandbox" restrictions and may use @Grab.

Add

## Global Pipeline Libraries

Sharable libraries available to any Pipeline jobs running on this system. These libraries will be trusted, meaning they run without "sandbox" restrictions and may use @Grab.

≡ Library ✖

Name ?  
jenkins-shared-library

Default version ?  
master  
Cannot validate default version until after saving and reconfiguring.

Load implicitly ?

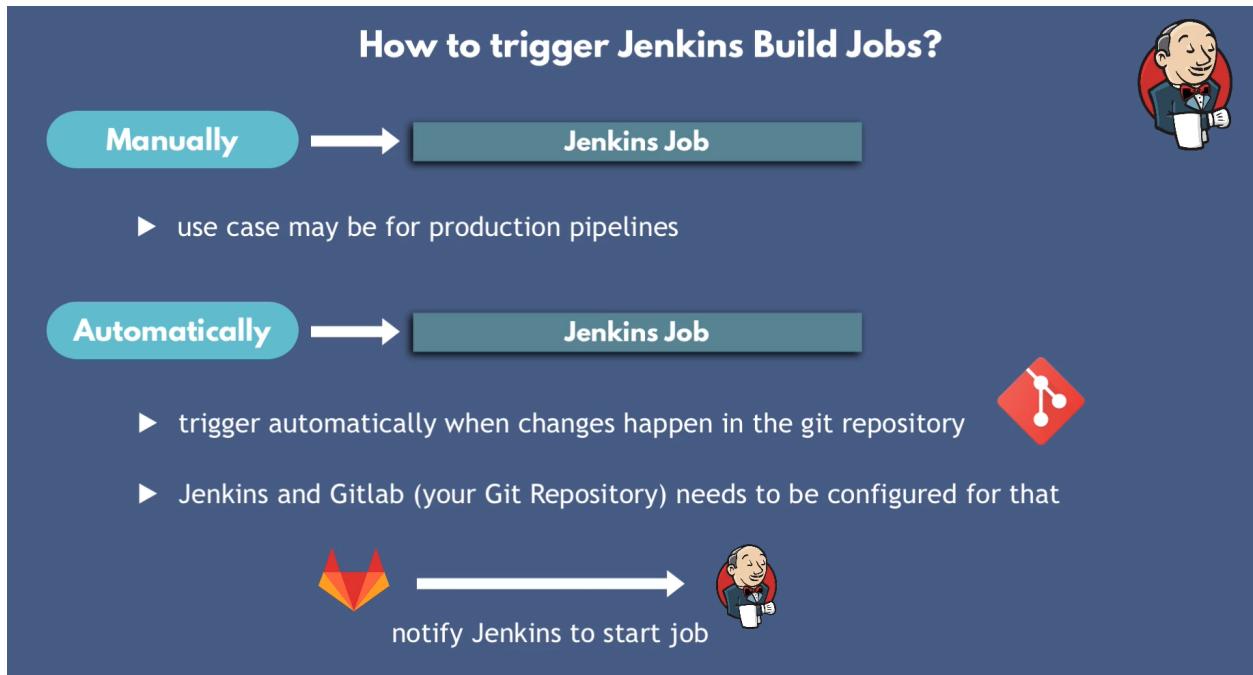
Allow default version to be overridden ?

Include @Library changes in job recent changes ?

Cache fetched versions on controller for quick retrieval ?

Retrieval method  
Modern SCM

## Trigger



A screenshot of the Jenkins job configuration interface. The top navigation bar shows tabs: General, Source Code Management, and Build Triggers. The Build Triggers tab is selected, indicated by a thicker border and bold text. The configuration area lists several trigger options with checkboxes:

- Trigger builds remotely (e.g., from scripts) ?
- Build after other projects are built ?
- Build periodically ?
- GitHub hook trigger for GITScm polling ?
- Poll SCM ?

We need to have a token for connection to github webhooks

### API Token

Current token(s) ?

There are no registered tokens for this user.

[Add new Token](#)

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

[Webhooks](#) / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

**Payload URL \***

**Content type**

**Secret**

We need this plugin for webhooks of multi-branch pipelines

multibranch scan

Updates Available Installed Advanced

Install ↑ Name

**Multibranch Scan Webhook Trigger**  
Trigger that can receive any HTTP request and trigger a multibranch job scan

Install without restart Download now and install after restart Update information

Periodically if not otherwise run ?

Scan by webhook ?  
Trigger token ?

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 be 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

