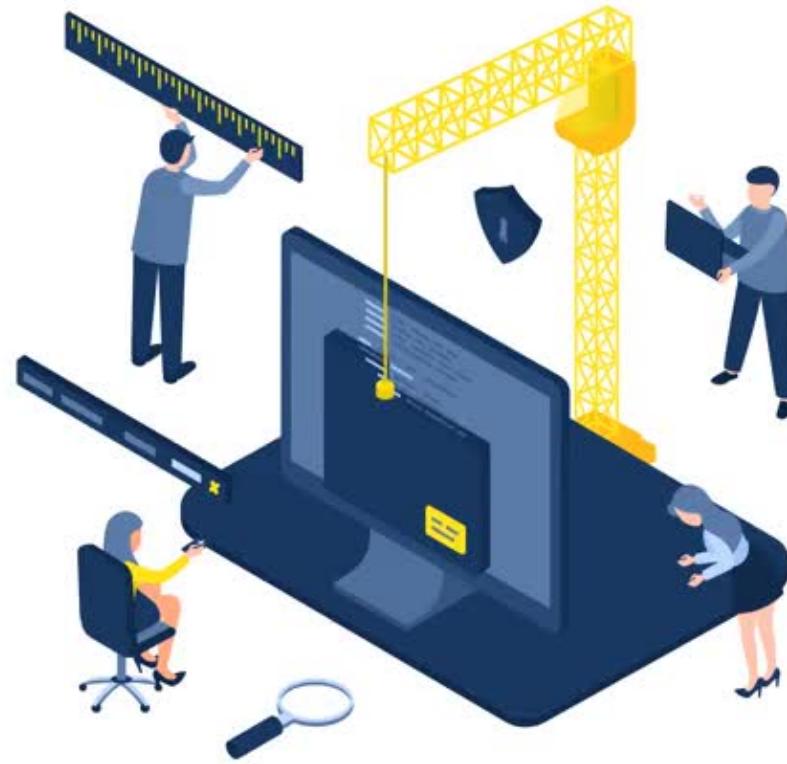


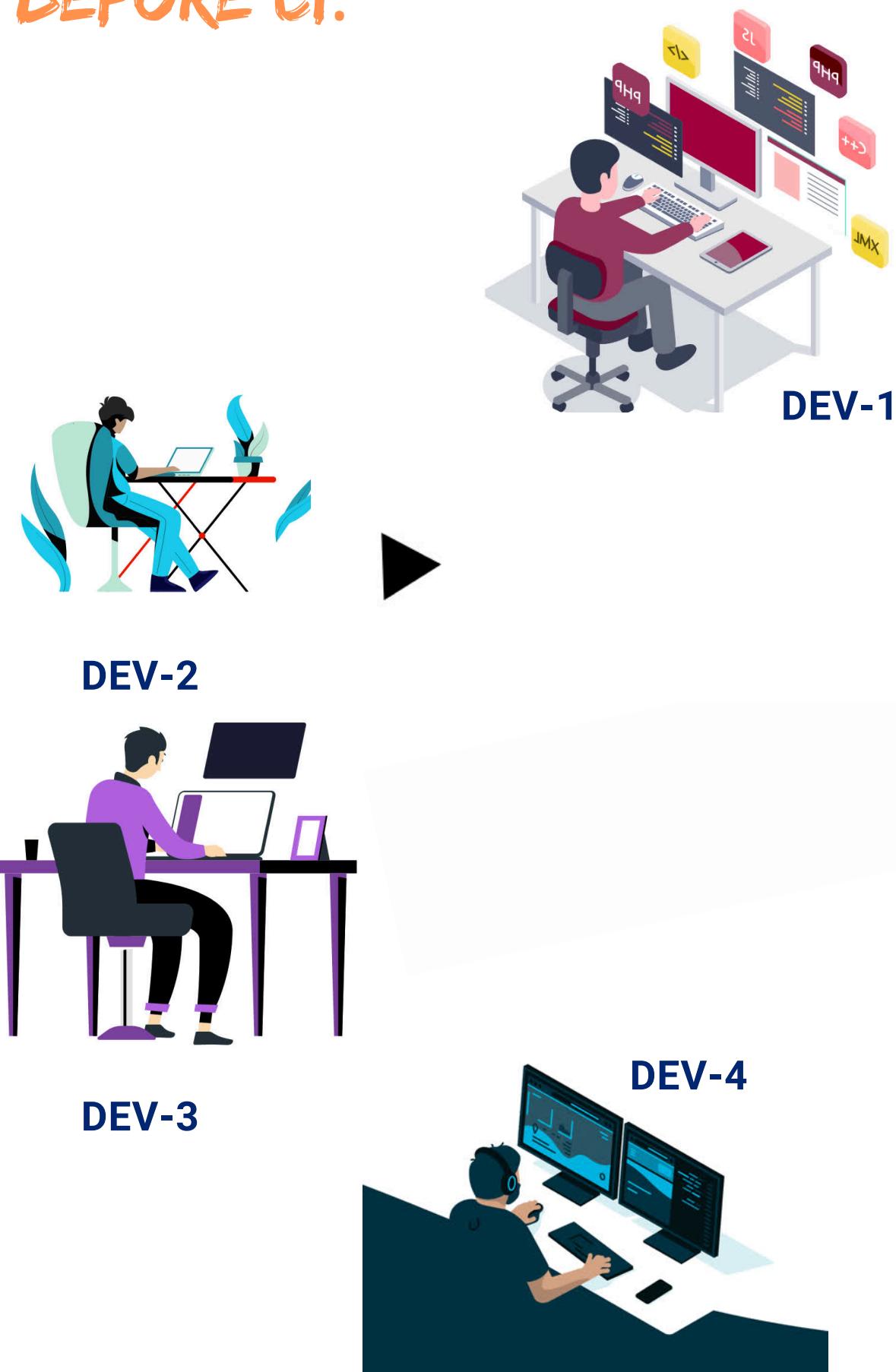
CONTINUOUS INTEGRATION:



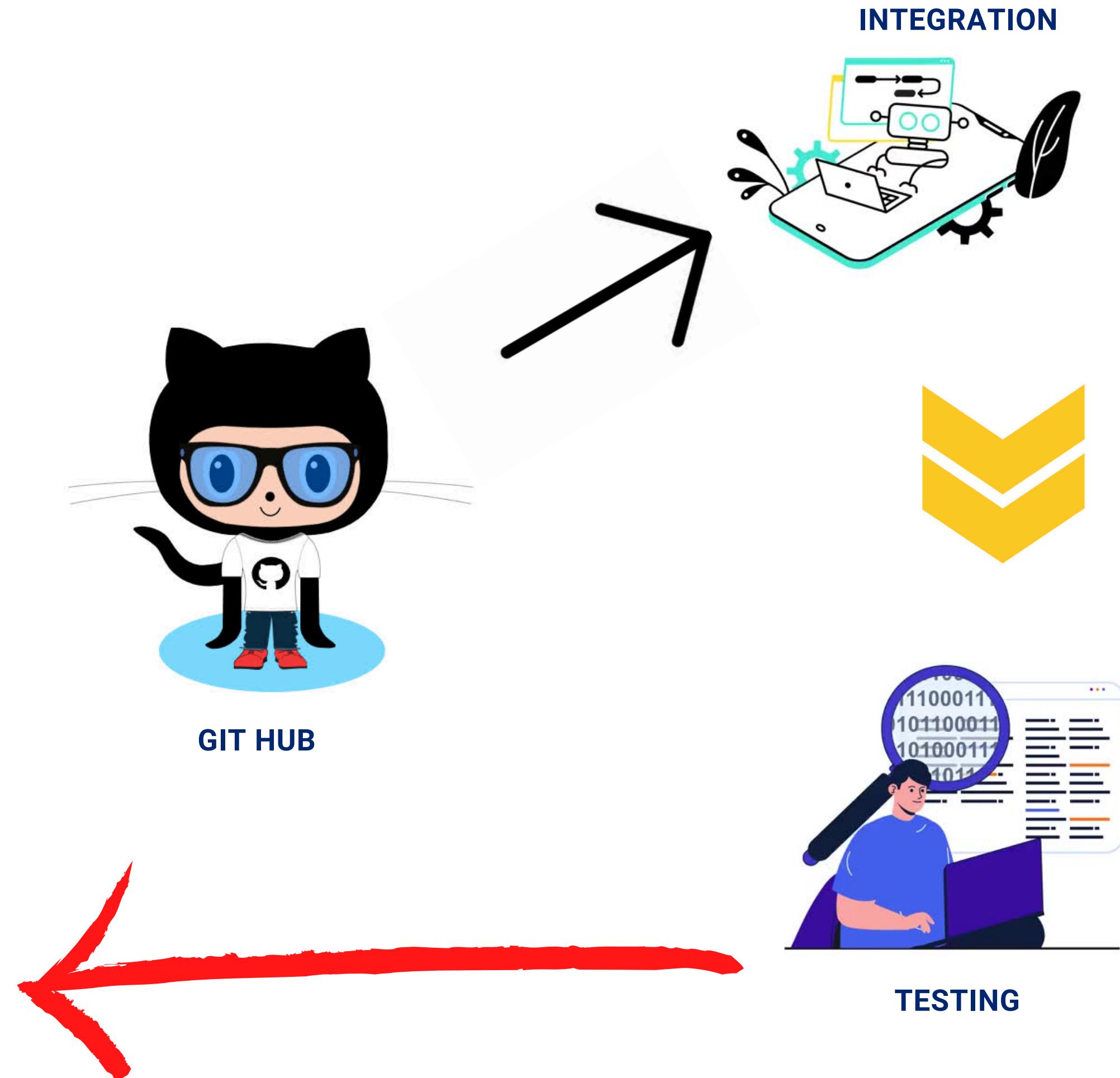
It is the combination of continuous build and continuous test

CONTINUOUS INTEGRATION = CONTINUOUS BUILD + CONTINUOUS TEST

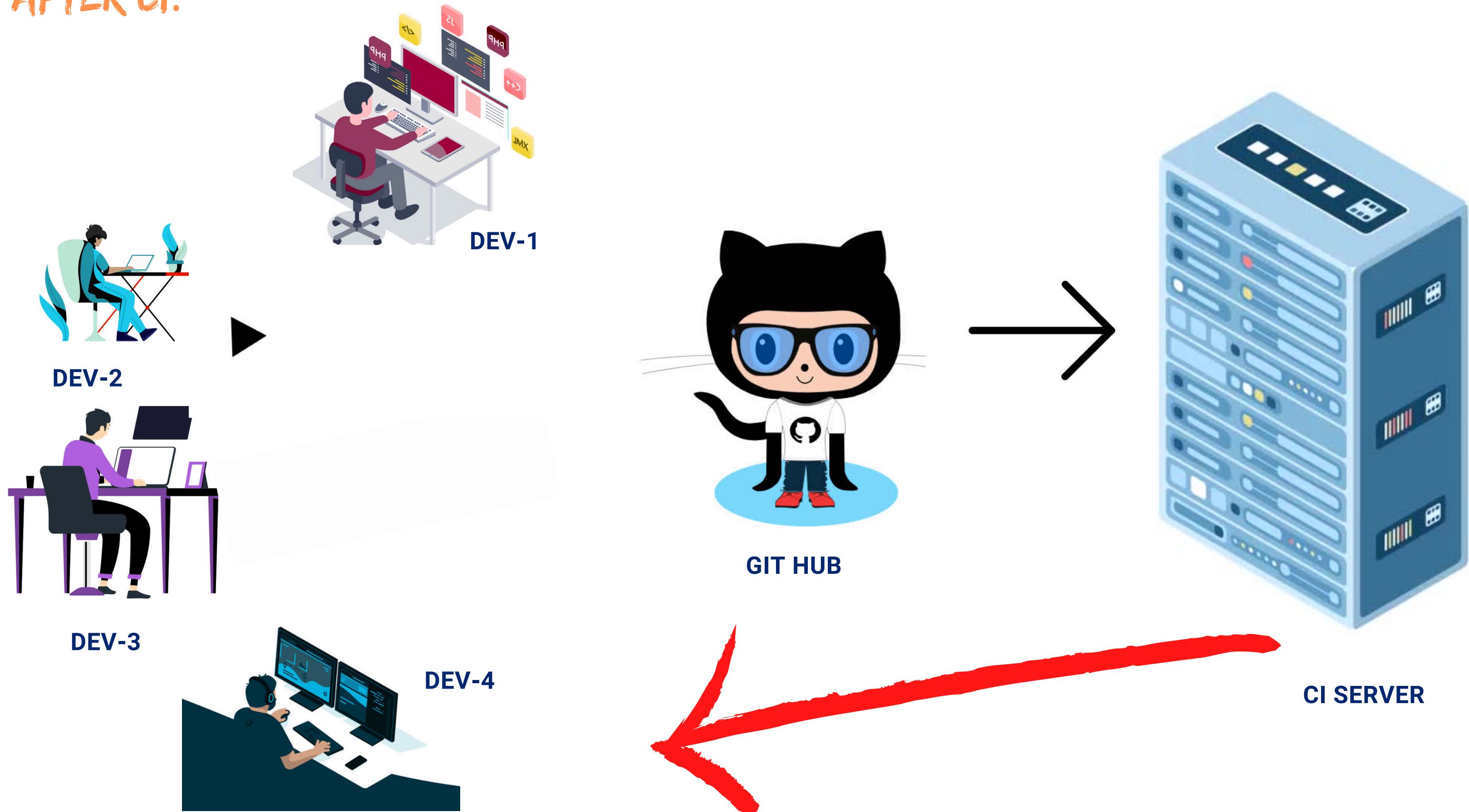
BEFORE CI:



INTEGRATION



AFTER CI:

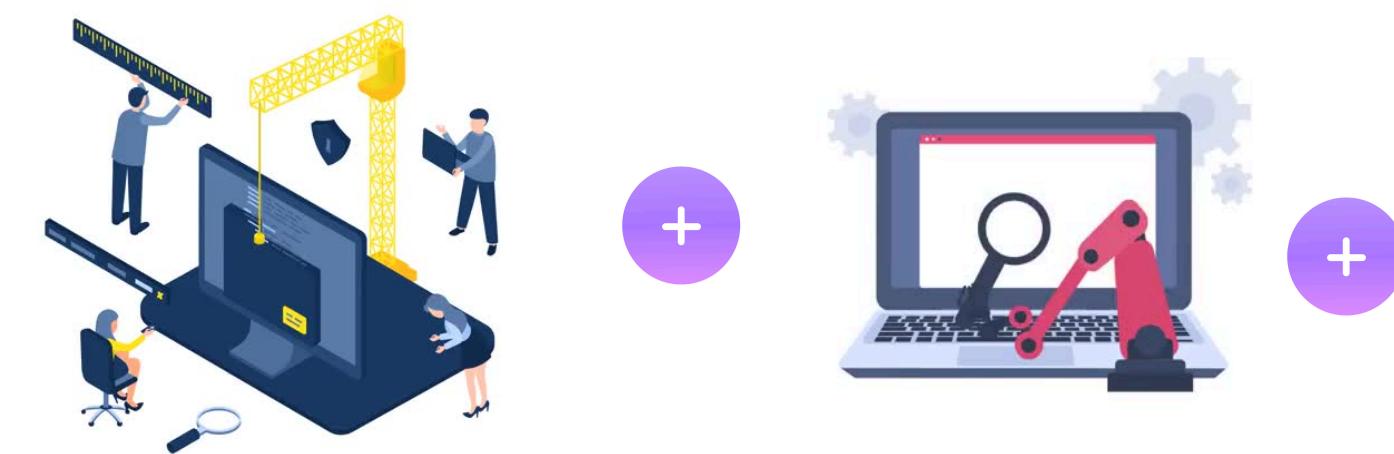


CI SERVER:

Here Build, Test & Deploy all these activities are performed in a single CI server



CI SERVER



BUILD

TEST

CONTINUOUS INTEGRATION:

Whenever a developer commits the code using source code management like GIT, then the CI pipeline gets the changes of the code runs automatically build and unit test.

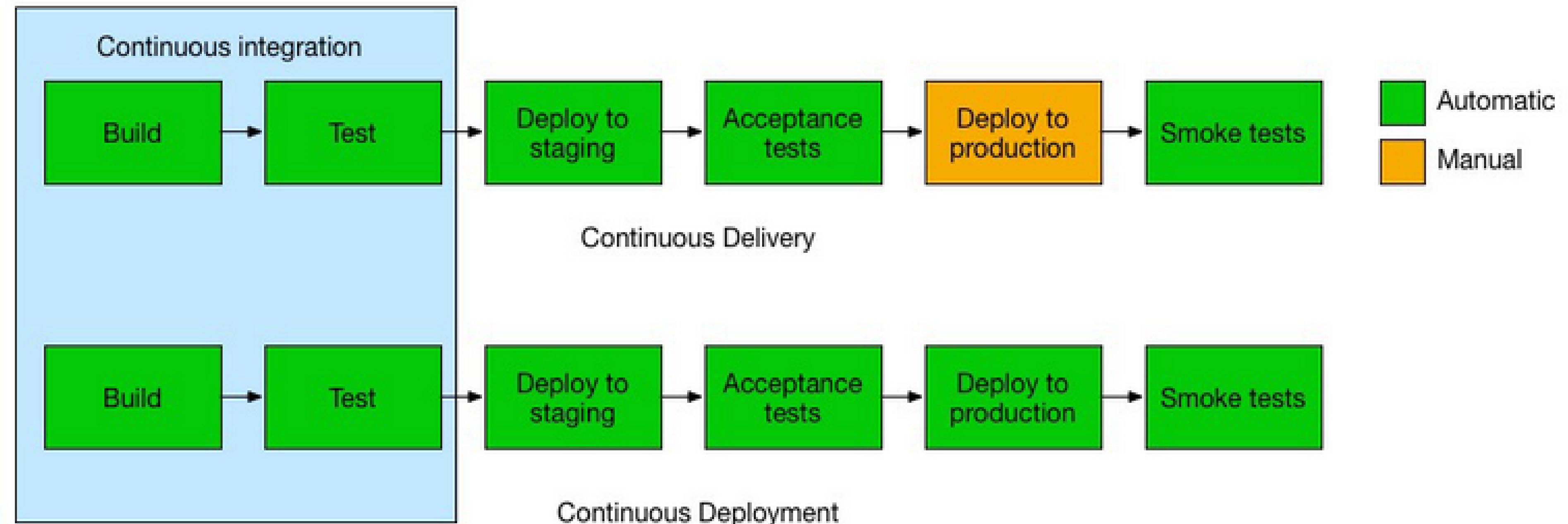
- Due to integrating the new code with old code, we can easily get to know the code is a success or failure.
- It finds the errors more quickly
- Delivery the products to client more frequently
- Developers don't need to manual tasks
- Reduces the developers time 20% to 30%

CD: CONTINUOUS DELIVERY/DEPLOYMENT

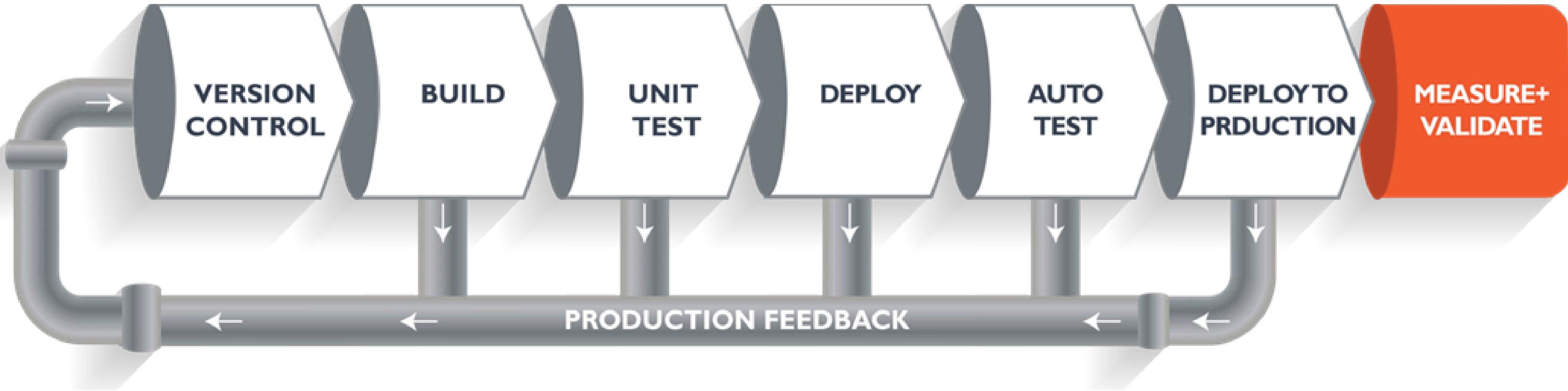
Continuous Delivery is making it available for deployment. Anytime a new build artifact is available, the artifact is automatically placed in the desired environment and deployed.

Continuous Deployment is when you commit your code then it gets automatically tested, build and deploy on the production server.

EX:



CI/CD PIPELINE:



It looks like Software Development Life Cycle, but let's see how it works.
Let's consider an example, if you are developing a web application

Version Control: Here developers need to write code for web applications. So it needs to be committed using version control system like GIT or SVN.

Build: Lets consider your code is written in java, it needs to be compiled before execution. In this build step code gets compiled.

Unit Test: If the build step is completed, then move to testing phase in this step unit step will be done.

Deploy: If the test step is completed, then move to Deploy phase in this step you can deploy your code in dev, testing environment. Here you can see your application output.

Auto Test: Once our code is working fine in testing servers, then we need to do Automation testing using Selenium or Junit.

Deploy to Production: If everything is fine then you can directly deploy your code in production server.

NOTE: If we have error in Code then it will give feedback and it will be corrected, if we have error in Build then it will give feedback and it will be corrected, Pipeline will work like this until it reaches Deploy.

Because of this pipeline, Bugs will be reported fast and get rectified so entire development is fast.

JENKINS:

Jenkins is an **open source** project written in **java** by Kohsuke Kawaguchi it runs on the **Window, Linux and Mac OS**

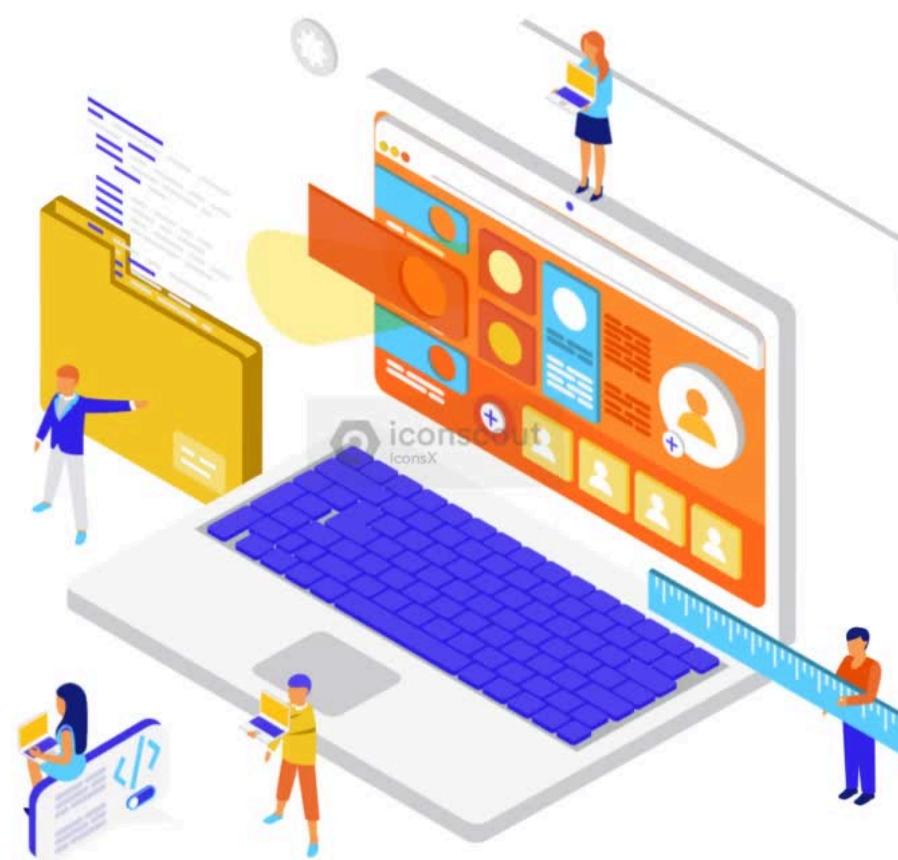


Jenkins



It is community-supported, Free to use, and the First choice for Continuous Integration.

- Consist of Plugins
- Automates the Entire Software Development Life Cycle (SDLC).

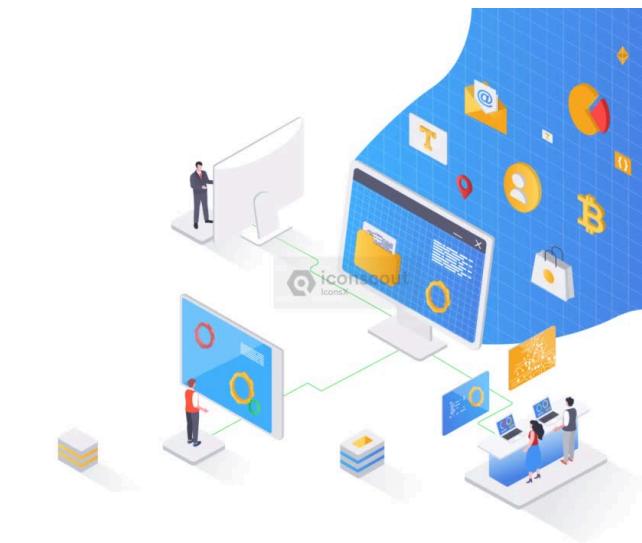
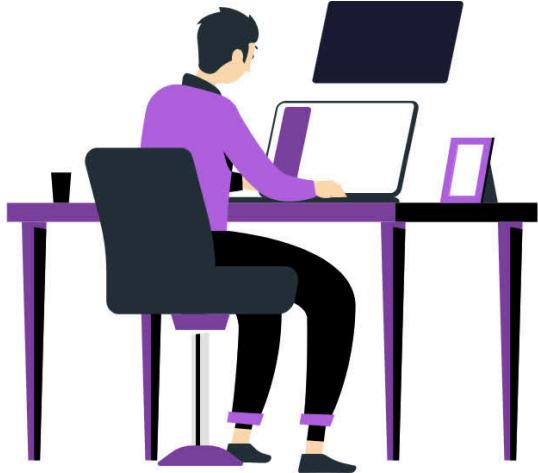


- It was originally developed by Sun Microsystem in 2004 as HUDSON.
- Hudson was an enterprise Edition we need to pay for it.
 1. The project was renamed Jenkins when Oracle brought the Microsystems.

- It can run on any major platform without Compatibility issues.

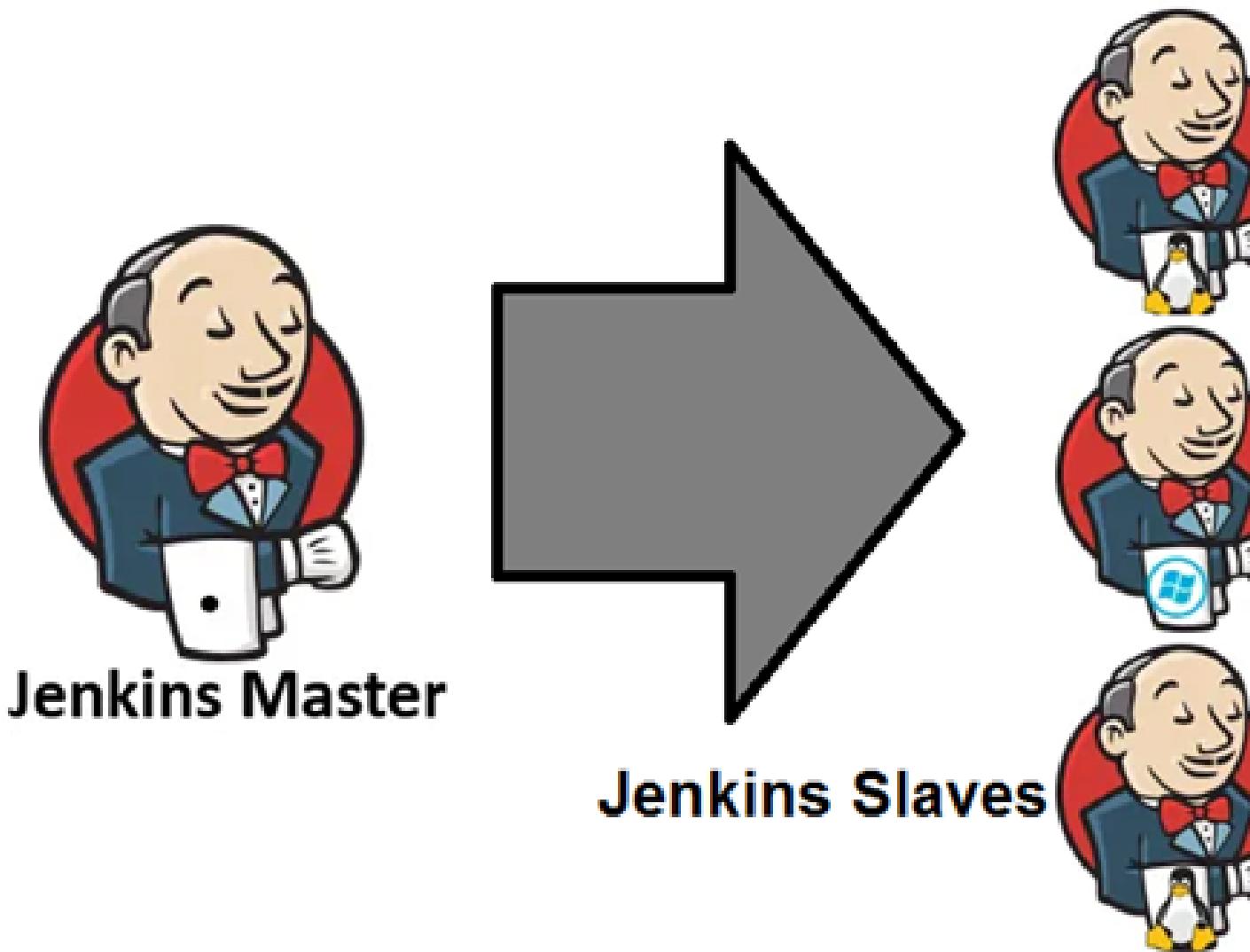


- Whenever developers write code, we integrate all the code of all developers at any point in time and we build, test, and deliver/deploy it to the client. This is called CI/CD.



ADVANTAGES:

- Jenkins follows Master-Slave Architecture.
- You can write your own plugin, can use the community plugin also.
- Can understand the process of what is going on.



- Jenkins master is going to assign a job to the slave..
- If Slaves are not available Jenkins itself does the job.
- By using the labels we can specify the jobs to the nodes.

JENKINS ALTERNATIVES:



JENKINS SETUP:

STEP-1: GETTING REPO (jenkins.io)

- sudo wget -O /etc/yum.repos.d/jenkins.repo <https://pkg.jenkins.io/redhat-stable/jenkins.repo>
- sudo rpm --import <https://pkg.jenkins.io/redhat-stable/jenkins.io.key>

STEP-2: INSTALL JAVA

- amazon-linux-extras install java-openjdk11 -y

STEP-3: INSTALL GIT MAVEN JENKINS

- yum install git maven jenkins -y

STEP-4: STARTING JENKINS SERVEICE

- systemctl start jenkins.service
- systemctl status jenkins.service

- Check Jenkins server is running or not:

1. [systemctl status jenkins](#)

- **Connect to dashboard:** copy the public IP address of the server and make a paste on the new tab with Jenkins port number (8080)

1. [public_ip:8080](#)

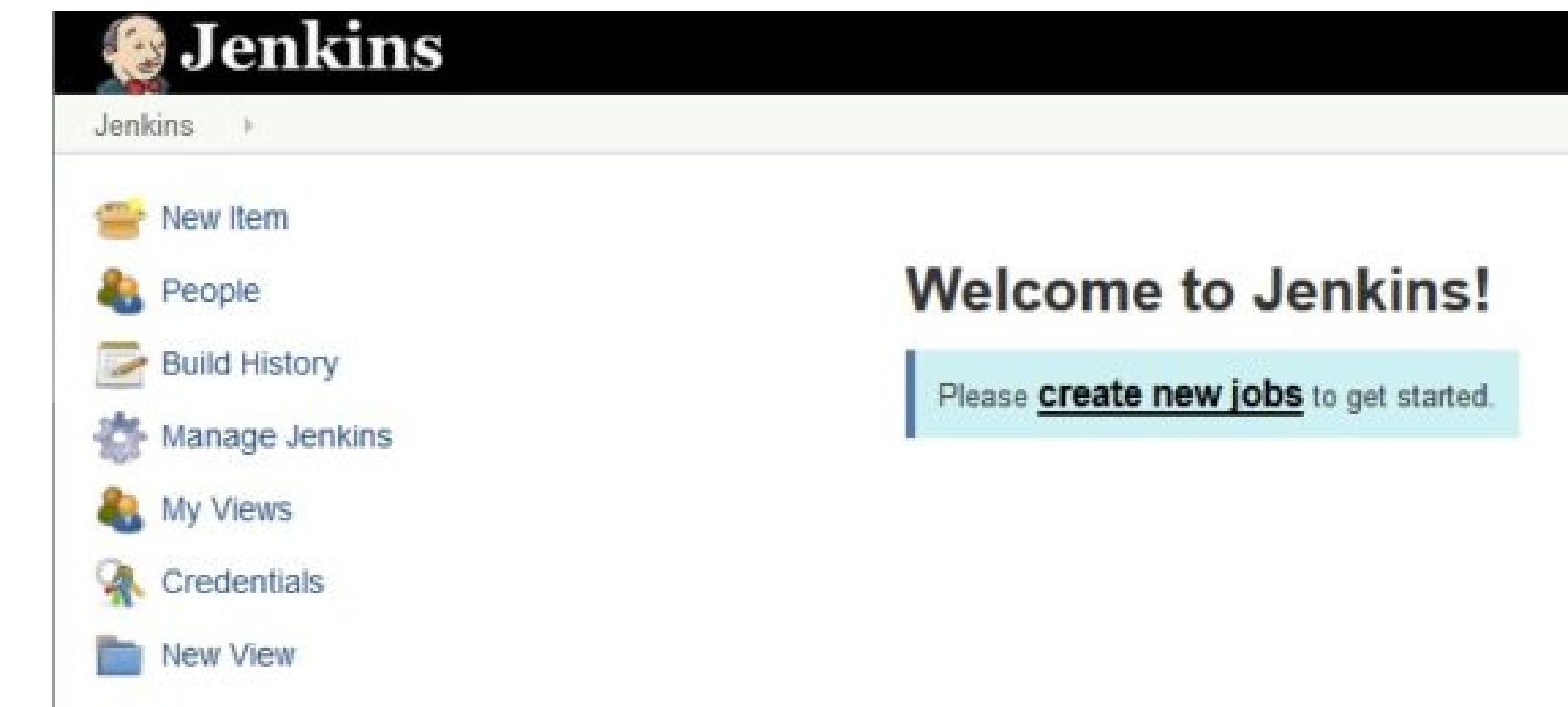
- **To unlock the jenkins:**

1. [cat /var/lib/jenkins/secrets/initialAdminPassword](#)

2. [Install suggested plugins](#)

3. [Add user name, Password and mail id](#)

4. [connect to dashboard](#)



JOB: To perform some set of tasks we use a job in jenkins.



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.



Maven project

Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.



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.



GitHub Organization

Scans a GitHub organization (or user account) for all repositories matching some defined markers.



Multibranch Pipeline

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

PARAMETER TYPES:

- String: any combination of characters and numbers
- Choice: a pre-defined set of strings from which a user can pick a value
- Credentials: a pre-defined Jenkins credential
- File: the full path to a file on the filesystem
- Multi-line String: same as String, but allows newline characters
- Password: similar to the Credentials type, but allows us to pass a plain text parameter specific to the job or pipeline
- Run: an absolute URL to a single run of another job

FILE PARAMETER:

This is used when we want to build our local files.

General — > This Project is Parameterized — > File Parameter

CHOICE PARAMETER:

This parameter is used when we have multiple options to generate a build but need to use only one specific one.

General — > This Project is Parameterized — > Choice Parameter

STRING PARAMETER:

This parameter is used when we need to pass a parameter as input by default.

It can be any combination of characters and numbers.

General — > This Project is Parameterized — > String Parameter

MULTI STRING PARAMETER:

This will work as same as String Parameter but the difference is instead of one single line string we can use multiple strings at a time as a Parameters.

General — > This Project is Parameterized — > Multi-String Parameter

LINKED JOBS : This is used when a job is linked with another job

TYPES: Up stream and Down stream

Here for job-1 both job-2 & job-3 are Downstream

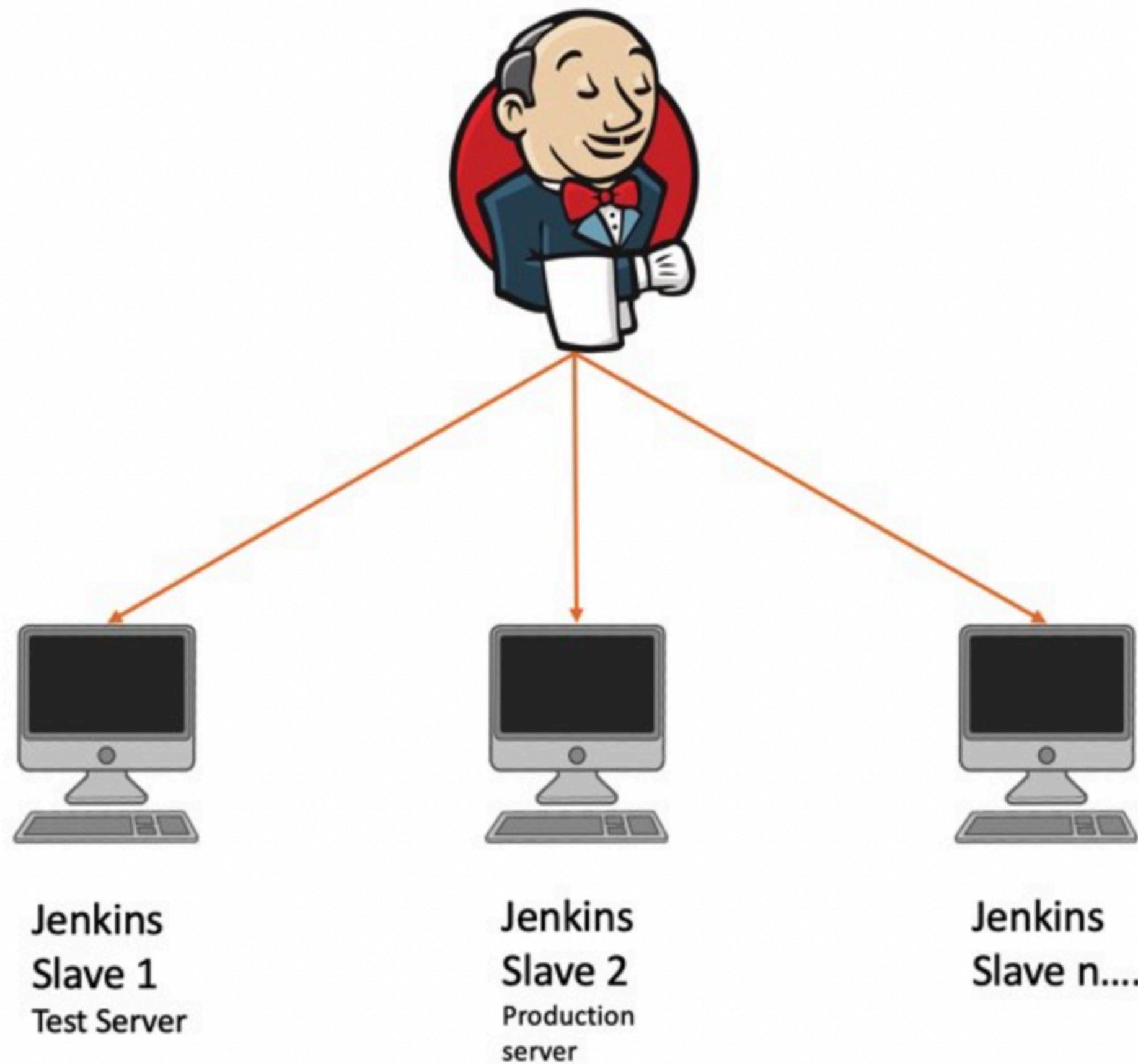
For job-2 upstream is job-1 and Downstream is job-3

for Job-3 Both Job-2 & job-1 are Upstream



MASTER & SLAVE:

Jenkins Server (master)



USES:

- Schedule the jobs
- store the builds
- sends the build to slave servers
-

SETUP:

```
sudo yum update -y
```

```
sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
```

```
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
```

```
sudo amazon-linux-extras install jenkins -y
```

```
sudo systemctl restart jenkins
```

```
sudo systemctl status jenkins
```

```
copy the IPV4 and paste it on browser like {ipv4:8080}
```

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

```
sudo passwd jenkins
```

```
sudo visudo
```

```
sudo vim /etc/ssh/sshd_config
```

```
sudo systemctl restart sshd
```

```
sudo systemctl status sshd
```

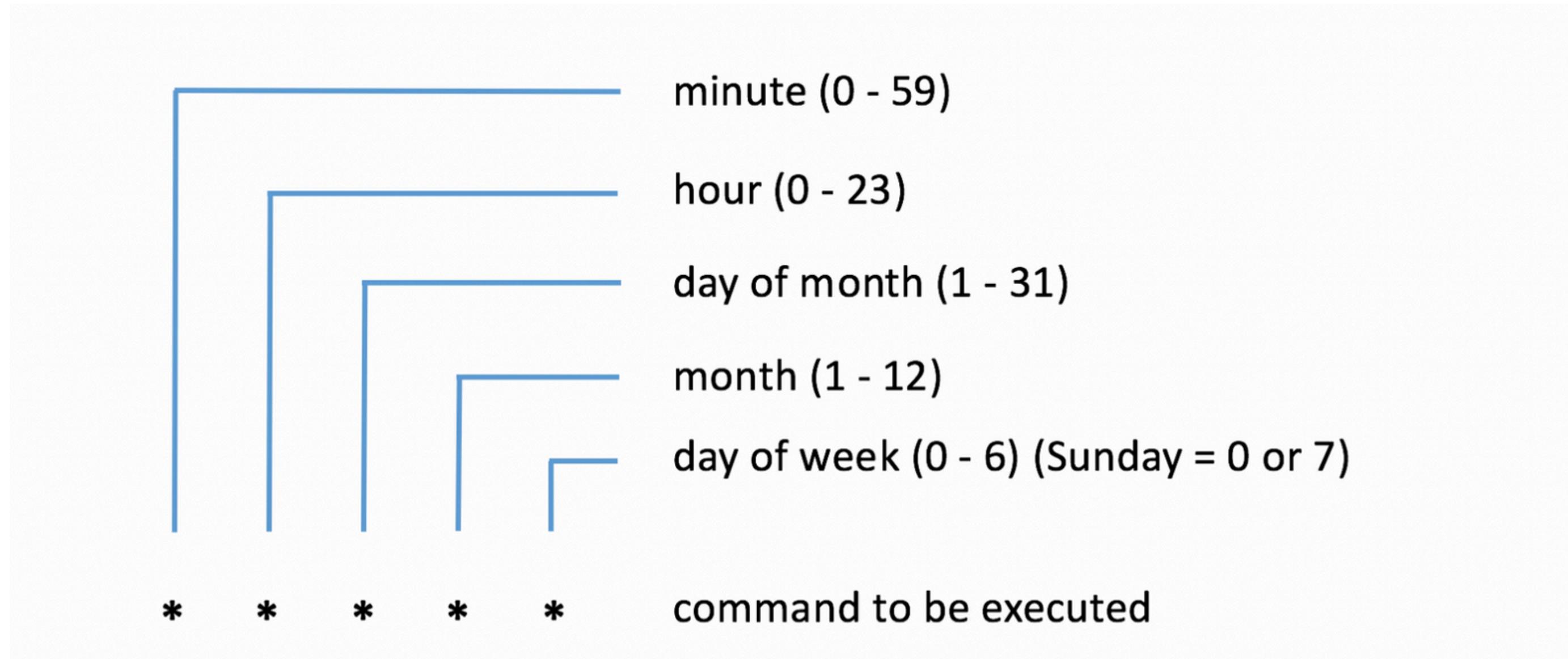

LOGIN TO SLAVE

```
sudo useradd jenkins  
sudo passwd jenkins  
sudo visudo  
sudo vim /etc/ssh/sshd_config  
sudo systemctl restart sshd  
sudo systemctl status sshd
```

LOGIN TO MASTER

```
sudo su jenkins  
ssh-keygen  
ssh-copy-id jenkins@localhost  
yes  
exit  
ssh-copy-id jenkins@public IPV4 of slave  
ssh jenkins@public IPV4 of Slave
```

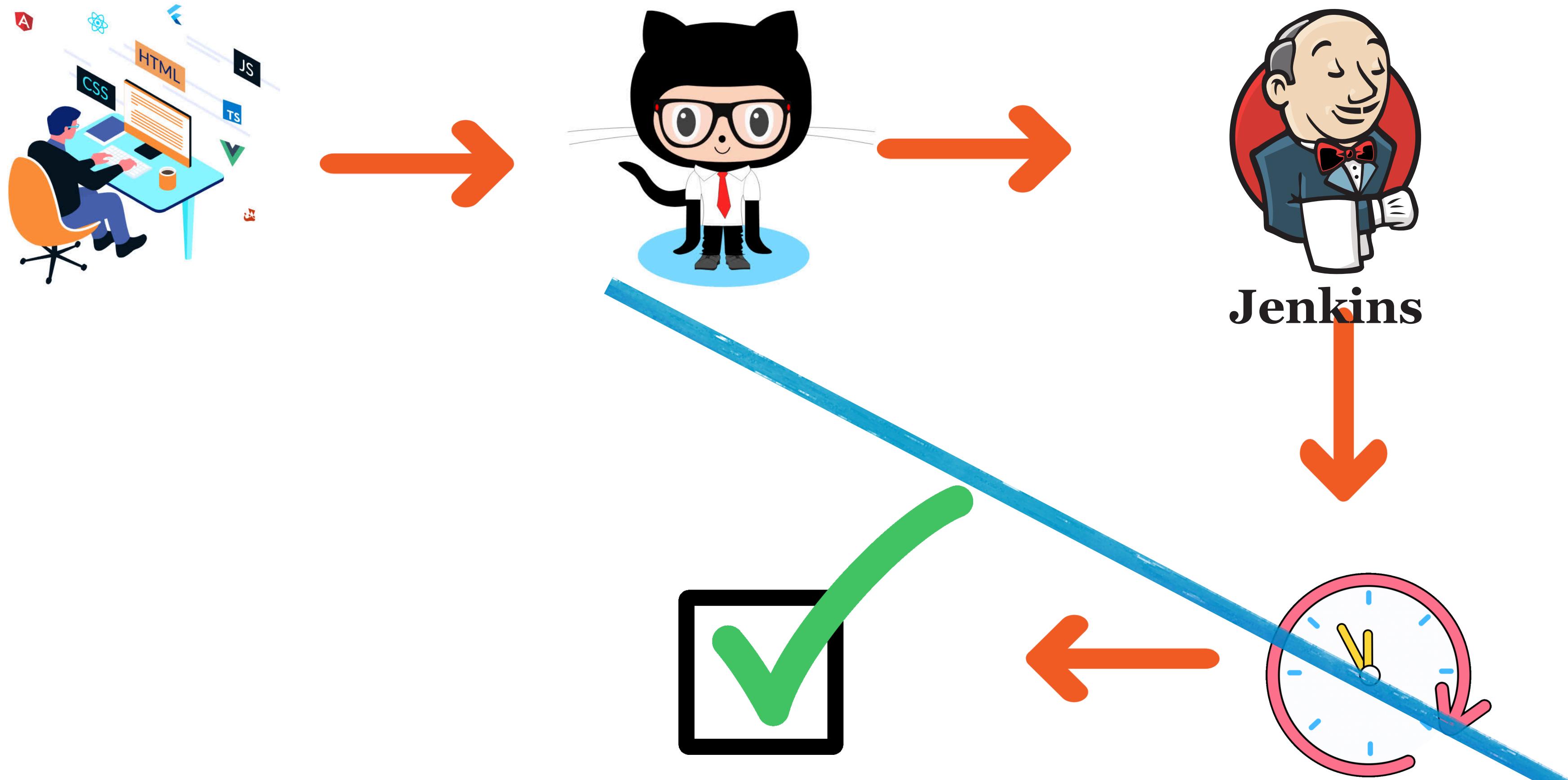
CRON JOB: We can schedule the jobs that need to be run at a particular intervals.



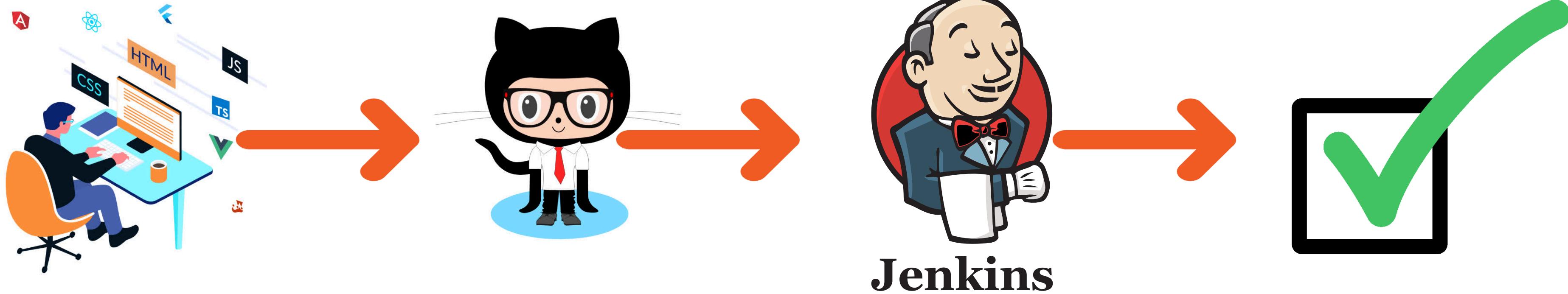
Build Triggers --> Build periodically --> * * * * * --> save

If you want to make it customize then we can use cron syntax generator.

POLL SCM: This is used when we want to build after developer pushes the code for regular interval



WEBHOOKS: If developer changes the code then it will build at that point of time.



Go to the github and select the repo
select settings --> webhooks --> create a webhook
Payload url : jenkinsurl/github-webhook/
Content type: Application.json
Now change the code in repo you will get a new build.

ENVIRONMENT VARIABLES:

BUILD — > SHELL SCRIPT

```
name=raham  
location=hyd  
echo "HAI, MY NAME IS ${name} IAM FROM ${location}"
```

SELECT TO SEE LIST OF ALL ENVIRONMENT VARIABLES

```
echo "BUILD ID IS ${BUILD_ID}"  
echo "JOB NAME IS ${JOB_NAME}"
```

echo "\${GLOBAL_VAR}" : THIS WONT WORK WE NEED TO SET IT MANUALLY

MANAGE JENKINS — >CONFIGURE SYSTEM — > GLOBAL PROPERTIES — > ENVIRONMENT NAME — >

NAME: GLOBAL_VAR & VALUE: ABCD — > SAVE

NOW IT WILL PRINT THE GLOBAL VAR OUTPUT

APPLY THESE TO PARAMETERISED BUILDS

```
echo "${GLOBAL_VAR}"  
echo "BUILD ID IS ${BUILD_ID}"  
echo "JOB NAME IS ${SERVER}"  
echo "DEPLOYED TO ${DEPLOY}"  
echo "YOUR PASSWORD IS ${yourPasswordPara}"  
echo "IM PRINTING ${multiLineStringParam}"  
echo "THIS IS FILE ${FILEPATH}"
```

TIMEOUT:

BUILD ENVIRONMENT —> TERMINATE THE BUILD IF IT STUCK —> ABSOLUTE: 3 —> EXECUTE SHELL SLEEP 240 (BUILD WILL TAKE 240 SECONDS) —> SAVE AFTER 4 MINUTES IT WILL ABORT AUTOMATICALLY IF YOU WANT TO MAKE IT AS FAIL CHECK TIME-OUT ACTIONS BELOW.

TIME STAMP:

```
echo "HAI ALL GOOD EVENING"  
sleep 10  
echo "WELCOME TO MY CLASS"  
TO CHECK IT ENABLE TIMESTAMP BUILD ENVIRONMENT  
CONCURRENT OR PARALLEL JOBS :
```

IF YOU WANT TO RUN PARALLEL BUILDS
GENERAL —> RUN PARALLEL BUILDS IF REQUIRED

THROTTLE BUILD:

IF YOU WANT TO RUN A SPECIFIC JOB FOR SPECIFIC TIMES IN A PERIOD OF TIME
NUMBER OF BUILDS: 4 TIME PERIOD: MINUTES
THAT MEANS ON A MINUTE MAXIMUM YOU CAN DO 4 BUILDS ONLY
YOU CAN SEE THE APPROXIMATE TIME BELOW THE NUMBER OF BUILDS

CUSTOM WORKSPACE:

THIS IS USED WHEN WE WANT TO CREATE OUR OWN CUSTOM WORK SPACE

GENERAL —> ADVANCED —> USE CUSTOM WORKSPACE —> PATH —> /tmp/test

NOW WE CREATED A TEST DIRECTORY UNDER /TMP

GIVE EXECUTE SHELL COMMAND (echo "HAI RAHAM" >> abc.txt)

NOW YOU WILL SEE abc.txt IN TEST DIRECTORY UNDER /tmp

RENAME A JOB:

IF YOU WANT TO RENAME A BUILD THEN

GENERAL —> ADVANCE —> DISPLAY NAME —> SAVE

CREATE A PIPELINE THROUGH BUILD PIPELINE:

PLUGINS —> BUILD PIPELINE —> INSTALL

CREATE TWO JOBS AND CONFIGURE EXECUTE SHELL FOR BOTH JOBS

JOB1-BUILD: { sleep 10 & echo "THIS IS BUILDING"}

JOB2-TEST: { sleep 10 & echo "THIS IS DEPLOYING TO TEST ENVIRONMENT"}

JOB1-BUILD —> CONFIGURE —> POST BUILD ACTION —> BUILD OTHER PROJECT —> JOB2-TEST

CLICK ON + SYMBOL ON DASH BOARD —> BUILD PIPELINE VIEW —> NAME —> CREATE

SELECT INITIAL JOB (JOB-1)—> OK & APPLY

JENKINS AGENT SETUP:

LAUNCH 2 INSTANCES (MASTER & SLAVE) WITH PEM FILE.

JENKINS SETUP IN MASTER

INSTALL JAVA-OPENJDK11 IN SLAVE

GO TO JENKINS DASHBOARD--> MANAGE JENKINS --> SETUP AGENT --> give node name and select permanent agent and create.

The screenshot shows the Jenkins 'Nodes' configuration screen. On the left, there's a sidebar with links like 'Back to Dashboard', 'Manage Jenkins', 'New Node', 'Configure Clouds', and 'Node Monitoring'. Below that are sections for 'Build Queue' (empty) and 'Build Executor Status' (1 Idle, 2 Idle). The main area is titled 'Name' with the value 'one', 'Description' with the value 'this is for testing purpose', 'Number of executors' with the value '2', 'Remote root directory' with the value '/home/ec2-user/jenkins/', 'Labels' with the value 'test', and 'Usage' with the dropdown value 'Only build jobs with label expressions matching this node'.

Dashboard > Nodes >

↑ Back to Dashboard

Manage Jenkins

+ New Node

Configure Clouds

Node Monitoring

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

Name ?

one

Description ?

this is for testing purpose

Number of executors ?

2

Remote root directory ?

/home/ec2-user/jenkins/

Labels ?

test

Usage ?

Only build jobs with label expressions matching this node

Dashboard > Nodes >

Only build jobs with label expressions matching this node

Launch method ?

Launch agents via SSH

Host ?

private_ip of the slave

Credentials ?

- none -

+ Add

! The selected credentials cannot be found

Host Key Verification Strategy ?

Non verifying Verification Strategy

Advanced...

Availability ?

This screenshot shows the 'Nodes' configuration page in Jenkins. It displays fields for setting up a new Jenkins node. The 'Host' field contains 'private_ip of the slave'. The 'Credentials' dropdown is set to '- none -' and shows an error message: '! The selected credentials cannot be found'. The 'Host Key Verification Strategy' dropdown is set to 'Non verifying Verification Strategy'. There is also an 'Advanced...' button and an 'Availability' section at the bottom.

Add jenkins credentials

Kind: SSH username with private key

Username: ec2-user

Private key: pem file copy paste

save & add node

PIPELINES:

Jenkins Pipeline is a combination of plugins that supports integration and implementation of continuous delivery pipelines.

A Pipeline is a group of events interlinked with each other in a sequence.

There are two types of Jenkins pipeline syntax used for defining your JenkinsFile

1. Declarative
2. Scripted

NEW ITEM —> NAME —> PIPELINE —> SAVE

DECLARATIVE

```
pipeline {  
    agent any  
  
    stages {  
        stage('Hello') {  
            steps {  
                echo 'Hello World'  
            }  
        }  
    }  
}
```

SCRIPTED

```
node {  
    stage ("stage1"){  
        echo "hai"  
    }  
}
```

Declarative pipeline :

They are a relatively new feature in Jenkins. They are quite easy to write and understand. The structure may seem to be a bit complex, but overall, it contains only a couple of basic sections. The “pipeline” block is the main block that contains the entire declaration of a pipeline.

Pipeline :

It is the outer main block that defines the entire build process.

Agent:

It defines the machine that will handle this pipeline. Jenkins provides the ability to perform distributed builds by delegating them to agent nodes. Doing this allows you to execute several projects with only one instance of the Jenkins server, while the workload is distributed to its agents.

Stages :

A sequence of one or more stage directives, the stages section is where the bulk of the "work" described by a Pipeline will be located. At a minimum it is recommended that stages contain at least one stage directive for each discrete part of the continuous delivery process, such as Build, Test, and Deploy.

Stage:

All of the real work done by a Pipeline are wrapped in one or more stage directives.

Steps :

Defines a series of steps to be executed in a given stage directive.

MULTISTAGE PIPELINES:

```
pipeline {  
    agent any  
  
    stages {  
        stage('Hello') {  
            steps {  
                echo 'Hello World'  
            }  
        }  
        stage('Test') {  
            steps {  
                echo 'Hello World test'  
            }  
        }  
        stage('Deploy') {  
            steps {  
                echo 'Hello World deploy'  
            }  
        }  
    }  
}
```

PIPELINE AS A CODE:

YOU CAN RUN COMMANDS HERE

```
pipeline {  
    agent any  
  
    stages {  
        stage('CMD') {  
            steps {  
                sh 'touch file1'  
                sh 'pwd'  
            }  
        }  
    }  
}
```

MULTIPLE COMMANDS OVER SINGLE LINE:

```
pipeline {  
    agent any  
  
    stages {  
        stage('CMD') {  
            steps {  
                sh ""  
                touch file2  
                pwd  
                date  
                whoami  
                ""  
            }  
        }  
    }  
}
```

ENVIRONMENT VARIABLES:

```
pipeline {  
    agent any  
  
    stages {  
        stage('ENV') {  
            steps {  
                sh 'echo "${BUILD_ID}"'  
            }  
        }  
    }  
}
```

```
pipeline {  
    agent any  
    environment {  
        name = 'raham'  
    }  
    stages {  
        stage('ENV') {  
            steps {  
                sh 'echo "${BUILD_ID}"'  
                sh 'echo "${name}"'  
            }  
        }  
    }  
}
```

```
pipeline {  
    agent any  
    environment {  
        name = 'raham'  
    }  
    stages {  
        stage('ENV1') {  
            steps {  
                sh 'echo "${BUILD_ID}"'  
                sh 'echo "${name}"'  
            }  
        }  
        stage('ENV2') {  
            environment {  
                name = 'shaik'  
            }  
            steps {  
                sh 'echo "${BUILD_ID}"'  
                sh 'echo "${name}"'  
            }  
        }  
    }  
}
```

PIPELINE AS A CODE PARAMETERS:

STRING

```
pipeline {  
    agent any  
    environment {  
        name = 'raham'  
    }  
    parameters {  
        string(name: 'person', defaultValue: 'raham shaik', description: "how are you")  
    }  
    stages {  
        stage('parameters') {  
            steps {  
                sh 'echo "${name}"'  
                sh 'echo "${person}"'  
            }  
        }  
    }  
}
```

BOOLEAN

```
pipeline {  
    agent any  
    parameters {  
        booleanParam(name: 'db', defaultValue: true, description: "")  
    }  
    stages {  
        stage('parameters') {  
            steps {  
                echo 'this isn my prod stage'  
            }  
        }  
    }  
}
```

CHOICE

```
pipeline {  
    agent any  
  
    parameters {  
        choice(name: 'server', choices: ['a','b','c'], description: "")  
    }  
    stages {  
        stage('parameters') {  
            steps {  
                echo 'this is my prod stage'  
            }  
        }  
    }  
}
```

INPUT

```
pipeline {  
    agent any  
  
    stages {  
        stage('git') {  
            steps {  
                git  
                'https://github.com/RAHAMSHAIK007/terrarepo.git'  
            }  
        }  
        stage('build') {  
            steps {  
                echo 'Hello World build'  
            }  
        }  
        stage('Deploy') {  
            input {  
                message "Can We Continue ?"  
                ok "Yes We Can Continue"  
            }  
            steps {  
                echo 'Hello World deploy'  
            }  
        }  
    }  
}
```

POST-BUILD:

THIS WILL BE PRINTED EVEN IF THE BUILD GETS FAILED

```
pipeline {  
    agent any  
  
    stages {  
        stage('Hello') {  
            steps {  
                echo 'Hello World'  
            }  
        }  
    }  
    post{  
        always {  
            echo 'THIS WILL BE PRINTED ANYWAY'  
        }  
    }  
}
```

```
pipeline {  
    agent any  
  
    stages {  
        stage('Hello') {  
            steps {  
                echo 'Hello World'  
            }  
        }  
        stage ('Deploy') {  
            steps {  
                echo12 'This is deployed'  
            }  
        }  
    }  
    post{  
        always {  
            echo 'THIS WILL BE PRINTED ANYWAY'  
        }  
    }  
}
```

```
pipeline {
    agent any

    stages {
        stage('Hello') {
            steps {
                echo 'Hello World'
            }
        }
        stage ('Deploy') {
            steps {
                echo 'This is deployed'
            }
        }
    }
    post{
        always {
            echo 'THIS WILL BE PRINTED ANYWAY'
        }
        failure {
            echo "THIS IS FAILED"
        }
        success {
            echo "THIS IS SUCCESS"
        }
    }
}
```

ALWAYS : WILL PRINT EVEN IF IT FAILS OR SUCCESS

FAILURE : WILL PRINT WHEN BUILD GOT FAILED

SUCCESS: WILL PRINT WHEN BUILD GOT SUCCESS

DO WITH THIS BY REMOVING 12 IN ECHO12

JENKINS CUSTOM WORK SPACE:

```
pipeline {  
    agent {  
        node {  
            label 'my-defined-label'  
            customWorkspace '/some/other/path'  
        }  
    }  
    stages {  
        stage ("one") {  
            steps {  
                echo "hai"  
            }  
        }  
    }  
}
```

PARALLEL STAGES ARE BUILD:

```
pipeline {  
    agent any  
    stages {  
        stage('stage-1') {  
            parallel {  
                stage('stage-2') {  
                    steps {  
                        echo "this is stage-1"  
                    }  
                }  
                stage('stage-3') {  
                    steps {  
                        echo "this is stage-2"  
                    }  
                }  
            }  
        }  
    }  
}
```

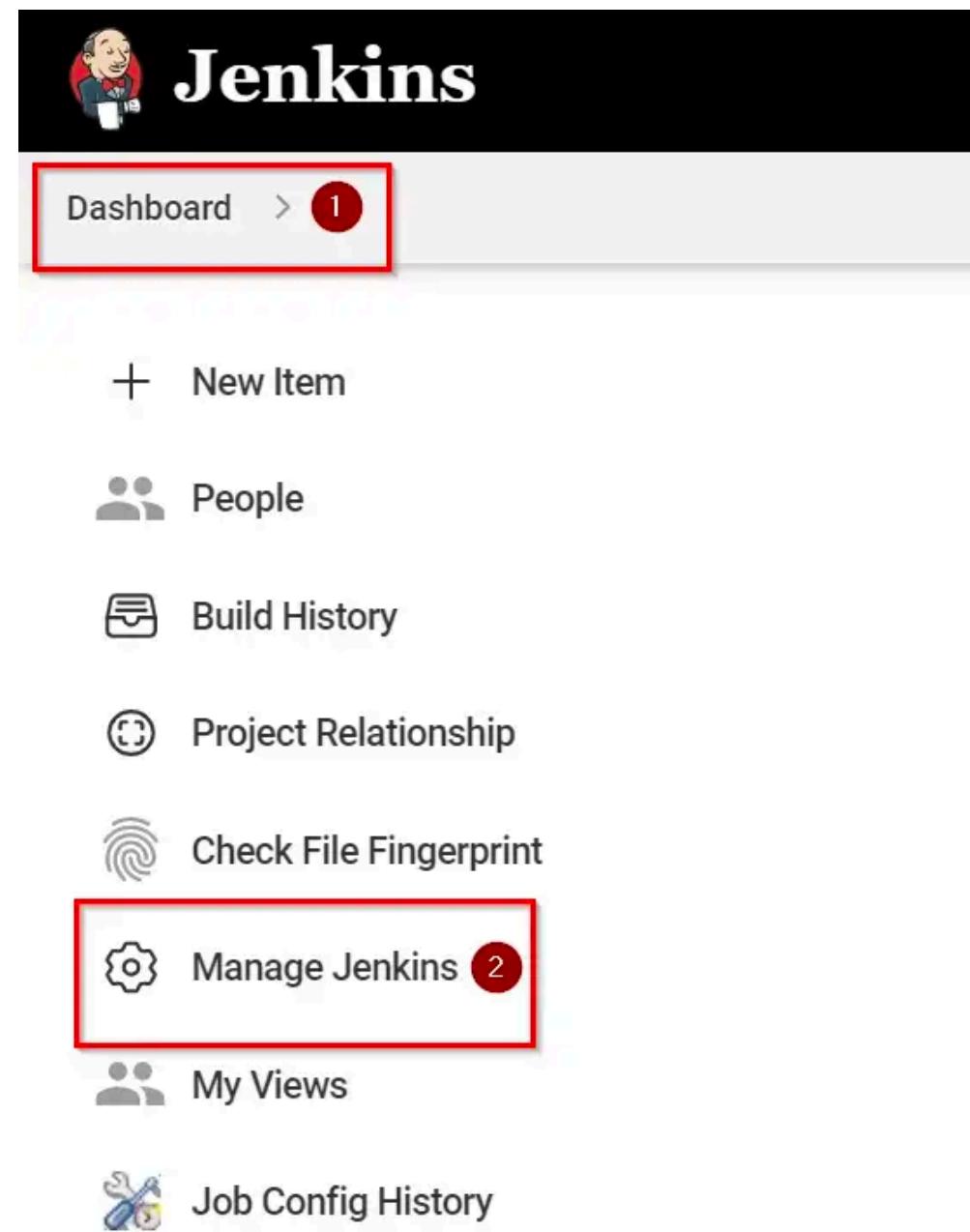
INPUT:

```
pipeline {  
    agent any  
    stages {  
        stage ('build') {  
            input{  
                message "Press Ok to continue"  
                submitter "user1,user2"  
                parameters {  
                    string(name:'username', defaultValue: 'user',  
                           description: 'Username of the user pressing  
                           Ok')  
                }  
            }  
            steps {  
                echo "User: ${username} said Ok."  
            }  
        }  
    }  
}
```

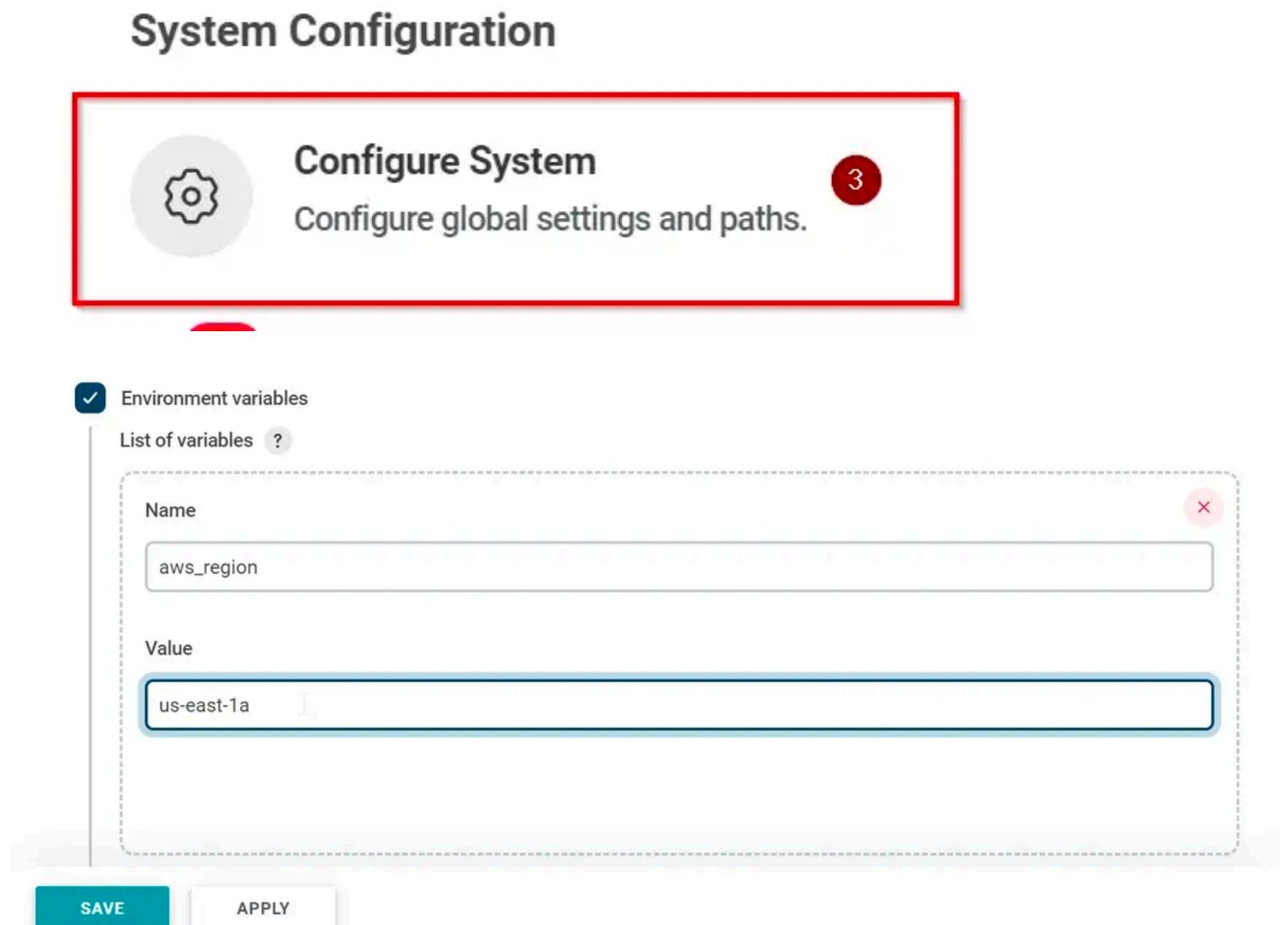
TASK USING PARAMETERS:

```
pipeline {  
    agent any  
    parameters {  
        string(name: 'NAME', description: 'Please  
        tell me your name')  
        choice(name: 'GENDER', choices: ['Male',  
                                         'Female'], description: 'Choose Gender')  
    }  
    stages {  
        stage('Printing name') {  
            steps {  
                script {  
                    def name = "${params.NAME}"  
                    def gender = "${params.GENDER}"  
                    if(gender == "Male") {  
                        echo "Mr. $name"  
                    } else {  
                        echo "Mrs. $name"  
                    }  
                }  
            }  
        }  
    }  
}
```

TO GET BUILD INFO:



The Jenkins dashboard interface. At the top left is the Jenkins logo. Below it is a navigation bar with 'Dashboard' and a red notification badge '1'. To the right of the navigation bar are links for 'New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins' (which has a red notification badge '2'), 'My Views', and 'Job Config History'.



The 'System Configuration' page with a red box highlighting the 'Configure System' section. This section contains a gear icon, the text 'Configure System', and the sub-instruction 'Configure global settings and paths.' A red notification badge '3' is located in the top right corner of this box.

Configure System
Configure global settings and paths.

Environment variables
List of variables ?

Name	Value
aws_region	us-east-1a

SAVE **APPLY**

note: you may get an error for running this pipeline, you need to approve the signature from admin. (click on the link and approve).

```
pipeline {  
    agent any  
    stages {  
        stage('Select docker registry') {  
            steps {  
                script {  
                    sh 'printenv'  
                }  
            }  
        }  
    }  
}
```

```
pipeline {  
    agent any  
    stages {  
        stage('list env') {  
            steps {  
                script {  
                    def environmentlist =  
                        env.getEnvironment()  
                    println environmentlist  
                }  
            }  
        }  
    }  
}
```

```
pipeline {
    agent any
    stages {
        stage('List env') {
            steps {
                script {
                    def environmentlist =
env.getEnvironment()
                    environmentlist.each {
                        key, value ->
println("${key} = ${value}");
                    }
                    println environmentlist
                }
            }
        }
    }
}
```

BUILD TRIGGERS:

Create a job --> job-1 --> build triggers --> Authentication Token : raham --> save
open new tab give like this [<http://54.184.85.228:8080/job/job-1/build?token=raham>]
Click enter your will see a new build
open incognito mode and execute then it will ask for login
Plugin --> Build Authorization Token Root --> install
<http://54.184.85.228:8080/buildByToken/build?job=job-1&token=raham>
(note if it doesnt work give like job=job-1\&

USER BASED ACCESS:

Manage jenkins --> create users 2 or 3
Plugins --> Role-Based Authorization Strategy --> install
Manage Jenkins --> Configure Global Security --> Authorization --> Role-Based Strategy
Manage and Assign Roles --> Manage role --> developer --> add --> permissions
Note: Give overall read access
item roles --> roles to add: developer --> Patteren: dev.*
item roles --> roles to add: tester --> Patteren: test.*
Assign roles --> enter user1 give developer & enter user2 give tester role --> save

PIPELINE CUSTOM WORKSPACE

```
pipeline {  
    agent {  
        node {  
            label 'prod'  
            customWorkspace '/tmp'  
        }  
    }  
}
```

EMAIL NOTIFICATION:

gmail --> manage your account --> security --> app passwords -->
other: jenkins --> generate -->
postbuild actions --> email notification --> save

PIPELINE UP STREAM & DOWN STREAM

```
pipeline {  
    agent any  
    stages {  
        stage('Start') {  
            steps {  
                echo 'Hello'  
            }  
        }  
        stage ('Invoke_pipelineA') {  
            steps {  
                build job: 'pipeline2'  
            }  
        }  
        stage('End') {  
            steps {  
                echo 'Bye'  
            }  
        }  
    }  
}
```

JENKIINS PORT NUMBER CHNAGE:

1. cd /usr/lib/systemd/system
2. vim jenkins.service
3. chnage port number in line 67
4. systemctl daemon-reload
5. systemctl restart jenkins

JENKIINS UI CHNAGE:

1. Manage Jenkins -> Plugins -- > simple theme plugin.
2. Manage Jenkins -> system -- > theme -- > CSS url and copy the below
<https://cdn.rawgit.com/afonsof/jenkins-material-theme/gh-pages/dist/material-cyan.css>
3. save and check the color.

CUSTOM CSS FOR JENKINS:

1. mkdir layout
2. **cd layout**
3. **vi style.css**
- 4.

