JENKINS

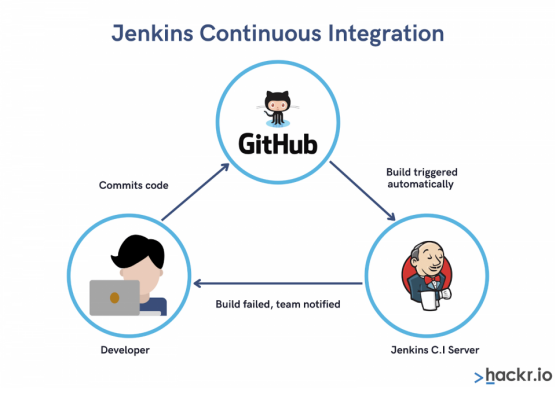
For continuous integration and for continuous deployment we are using jenkins

CI: Continuous Integration

It is the combination of continuous build + continuous test

Whenever Developer commits the code using source code management like GIT, then the CI Pipeline gets the change 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 do manual tasks
- It reduces the developer time 20% to 30%

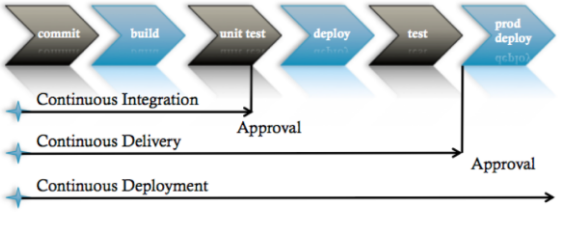


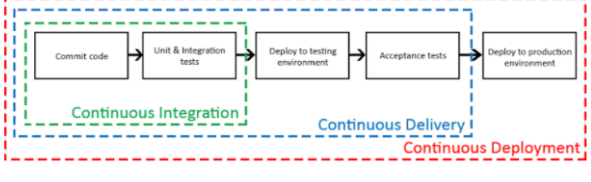
CI Server

Here only Build, test & Deploy all these activities are performed in a single CI Server

Overall, CI Server = Build + Test + Deploy

CD: Continuous Delivery/Deployment





Continuous Delivery

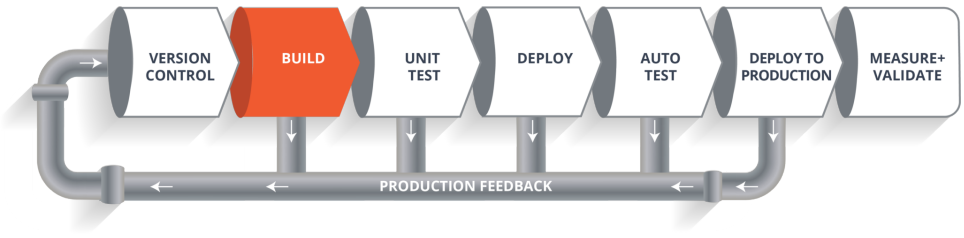
CD is making it available for deployment. Anytime a new build artifact is available, the artifact is automatically placed in the desired environment and deployed

- Here, Deploy to production is manual here

Continuous Deployment

- CD is when you commit your code then its gets automatically tested, build and deploy on the production server.
- It does not require approval
- 99% of customers don't follow this
- Here, Deploy to production is automatic

CI/CD Pipeline



It looks like a Software Development Life Cycle (SDLC). Here we are having 6 phases

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

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

- Let's consider your code is written in java, it needs to be compiled before execution. In this build step code gets compiled
- For build purpose we're using maven

Unit Test

- If the build step is completed, then move to testing phase in this step unit step will be done.
- Here we can use SonarQube/mvn test
- Here, application/program components are perfectly worked/not we will check in this testing
- Overall, It is code level testing

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
- Overall, we are deploying our application in Pre-Prod server. So, Internally we can access

Auto Test

- Once your code is working fine in testing servers, then we need to do Automation testing
- So, overall it is Application level testing
- Using Selenium (or) Junit testing

Deploy to Production

If everything is fine then you can directly deploy your code in production server

Because of this pipeline, bugs will be reported fast and get rectified so entire development is fast. Here, Overall SDLC will be automatic using Jenkins

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

WHAT IS JENKINS

- It is an open source project written in Java by kohsuke kawaguchi
- The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.
- It is platform independent
- It is community-supported, free to use
- It is used for CI/CD
- If we want to use continuous integration first choice is jenkins
- It consists of plugins. Through plugins we can do whatever we want. Overall without plugins we can't run anything in jenkins
- It is used to detect the faults in the software development
- It automates the code whenever developer commits
- It was originally developed by SUN Microsystem in 2004 as HUDSON
- HUDSON was an enterprise addition we need to pay for it
- The project was renamed jenkins when oracle brought the microsystems
- Main thing is It supports master & slave concepts
- It can run on any major platform without complexity issues
- We can create the pipelines by our own
- We have speed release cycles
- Jenkins default port number is 8080

Jenkins Installation

1. Launch an linux server in AWS and add security group rule [Custom TCP and 8080]
2. Install java → amazon-linux-extras install java-openjdk11 -y
3. Getting keys and repo i.e.. copy those commands from "jenkins.io" in browser and paste in terminal
    - open browser → jenkins.io → download → Download Jenkins 2.401.3 LTS for under → Redhat
    - 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-2023.key
    - Copy above 2 links and enter in terminal
4. Install Jenkins - yum install jenkins -y
5. systemctl status jenkins - It is in inactive/dead state
6. systemctl start/restart jenkins - Start the jenkins

Now, open the jenkins in browser - publicIP:8080

JENKINS Default Path : /var/lib/jenkins

- Enter the password go to the particular path i.e. cd path
- Click on install suggested plugins

Now, Start using jenkins

Alternative way to install jenkins:

- Everytime we have to setup jenkins manually means it will takes time instead of that we can use shell scripting i.e.
- vim jenkins.sh > add all the manual commands here > :wq
- Now, we execute the file
- First we need to check whether the file has executable permissions/not, if it's not
    - #chmod +x jenkins.sh
- Run the file
    - ./ jenkins.sh (or) sh jenkins.sh

Create a new Job/task

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

In Jenkins jobs are of two types

- [ ] Freestyle (old)

☐ Pipeline (new)

Now, we are creating the jobs in freestyle

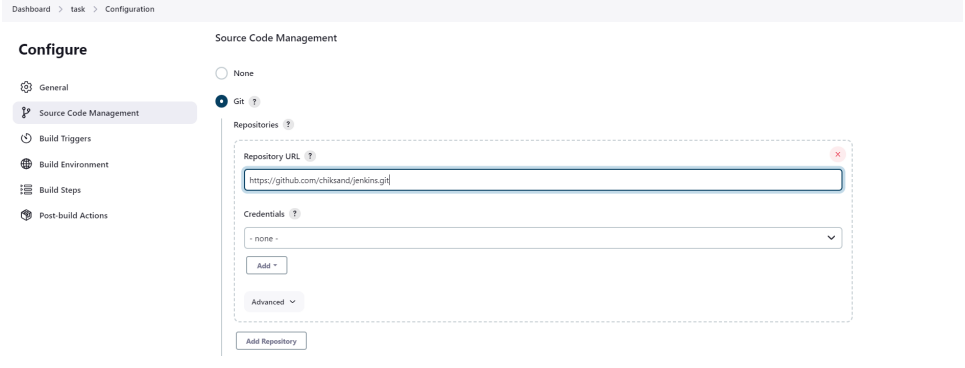
1. Click on create a job (or) new item
2. Enter task name
3. click on freestyle project (or) pipeline [Depends on your requirement]

These are the basic steps to Create a Job

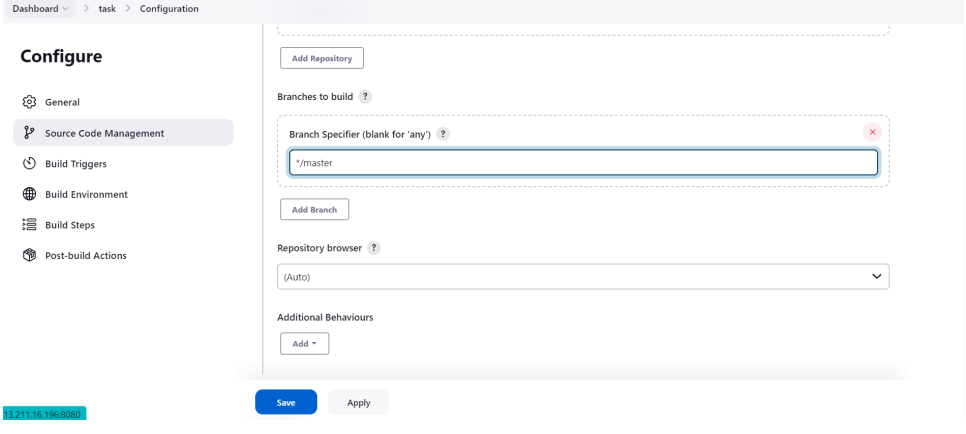
Get the Git Repo

Follow above 3 steps then after

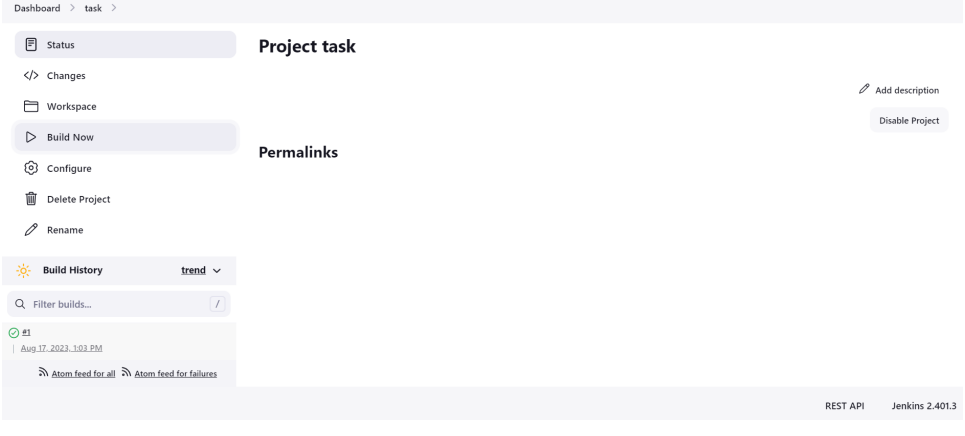
1. Copy the github repo url and paste in under SCM. It is showing error
2. So, now in your AWS terminal → Install GIT → yum install git -y
3. Whenever we are using private repo, then we have to create credentials. But right now, we are using public repo. So, none credentials



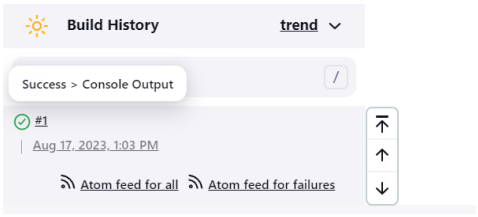
4. If we want to get the data from particular branch means you can mention the branch name in branch section. But default it takes master



5. Click on save and Build now and build success



If you want to see output in jenkins. Click on console output i.e., (click green tick mark)



If you want to see the repo in our linux terminal

Go to this path → cd /var/lib/jenkins/workspace/task_name → now you can see the files from git repo

- [ ] If we edit the data in github, then again we have to do build, otherwise that change didn't reflect in linux server
- [ ] Once run the build, open the file in server whether the data is present/not
- [ ] So, if we're doing like this means this is completely under manual work. But, we are DevOps engineers we need automatically

How are you triggering your jenkins Jobs ?

Jenkins job can be triggered either manually (or) automatically

1. Github Webhook
2. Build Periodically
3. Poll SCM

WebHooks

Whenever developer commits the code that change will be automatically applied in server. For this, we use WebHooks

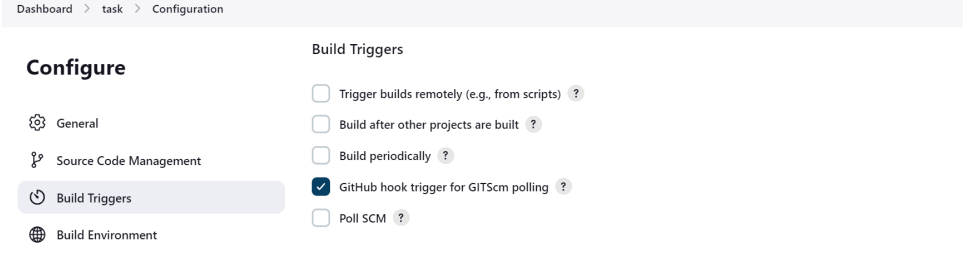
How to add webhooks from GitHub

Open repository → settings → webhooks → Add webhook →

- payload URL : jenkinsURL:8080/github-webhook/
- Content-type : Application/json
- Click on Add webhook

So, we are created webhooks from github

Now, we have to activate in jenkins dashboard, here Go to Job → Configure → select below option → save



Schedule the Jobs

Build Periodically

Select one job → configure → build periodically

Here, it is working on "CRON SYNTAX"

- Here we have 5 starts i.e. * * * * *

        - 1st star represents minutes
        - 2nd star represents hours [24 hours format]
        - 3rd star represents date
        - 4th star represents month
        - 5th star represents day of the week
            - i.e.., Sunday - 0
                - Monday - 1
                - Tuesday - 2
                - Wednesday - 3
                - Thursday - 4
                - Friday - 5
                - Saturday - 6
    - Eg: Aug 28, 11:30 am, sunday Build has to be done → 30 11 28 08 0 → copy this in build periodically
- If we give " * * * * * " 5 stars means → At every minute build will happen
- If i want every 5 minutes build means → */5 * * * *

Click on Save and Build

Note: Here changes 'happen/not' automatically build will happen in "schedule the jobs"

For Reference, Go to browser → Crontab-guru

Poll SCM

Select one job → configure → select Poll SCM

- It only works whenever the changes happened in "GIT" tool (or) github
- We have to mention between the time like 9am-6pm in a day
- same it's also working on cron syntax
    - Eg: * 9-18 * * *

Difference between WebHooks, Build periodically, Poll SCM (FAQ)

Webhooks:

- Whenever developer commits the code, on that time only build will happen.
- It is 24x7 no time limit
- It is also working based on GIT Tool (or) github

Poll SCM:

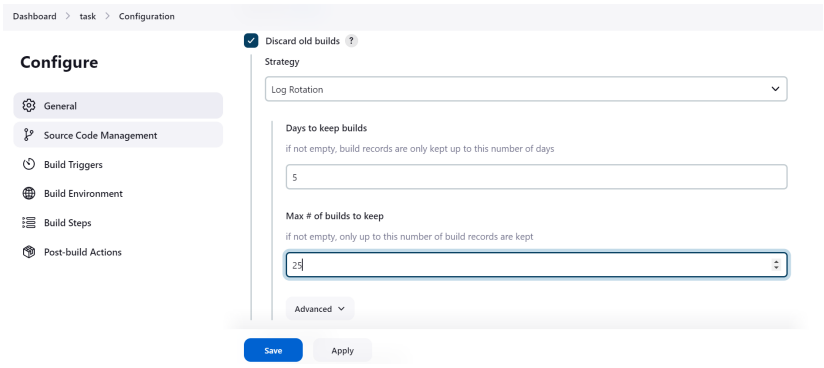
- Same as webhooks, But here we have time limit
- Only for GIT

Build Periodically:

- Automatically build, whether the changes happen/not (24x7)
- It is used for all devops tools not only for git
    - It will support on every work
- Every Time as per our schedule

▢ Discard old builds

- Here, we remove the builds, i.e., Here we can see how many builds we have to see (or) max of builds to keep (or) how many days to keep builds. we can do this thing here



- But, when we are in jenkins, it is little bit confusion to see all the builds
- So, here max. 3 days we can store the builds.
- In our dashboard we can see latest 25 builds
- More than 25 means automatically old builds get deleted
- So, overall here we can store, delete builds.
- These type of activities are done here

In server, If you want to see build history ?

Go to jenkins path (cd /var/lib/jenkins) → jobs → select the job → builds

If we want to see log info i.e., we can see console o/p info

Go inside builds → 1 → log Here, In server we don't have any problem

Parameter Types:

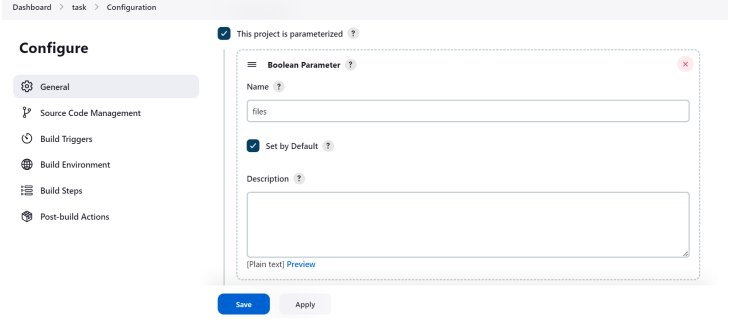
1. String → Any combination of characters & numbers
2. Choice → A pre-defined set of strings from which a user can pick a value
3. Credentials → A pre-defined jenkins credentials
4. File → The full path to a file on the file system
5. Multi-line string → Same as string, but allows newline characters
6. password → Similar to the credentials type, but allows us to pass a plain text parameter specific to the job (or) pipeline
7. Run → An absolute URL to a single run of another job

☐ This project is parameterized

Here, we are having so many parameters, In real life we will use this

1.Boolean parameter

Boolean means used in true (or) false conditions



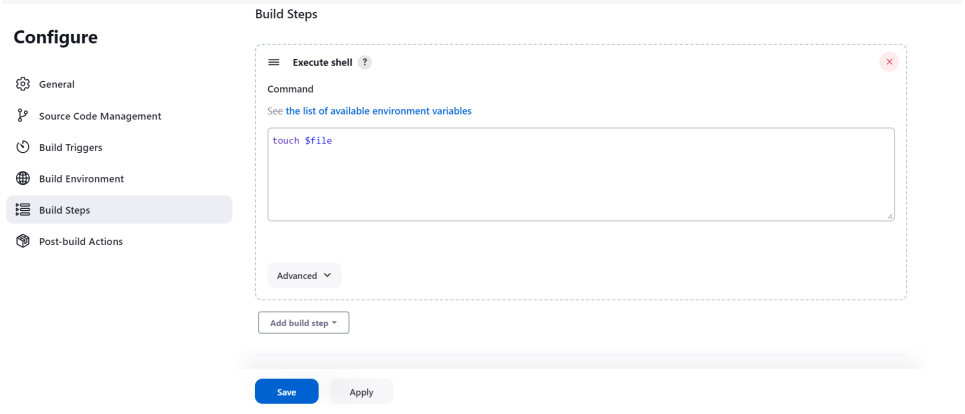
Here, Set by Default enable means true, Otherwise false

☐ Choice parameter

- This parameter is used when we have multiple options to generate a build but need to use only on specific one
- If we have multiple options i.e., either branches (or) files (or) folders etc.., anything we have multiple means we use this parameter

Suppose, If you want to execute a linux command through jenkins means

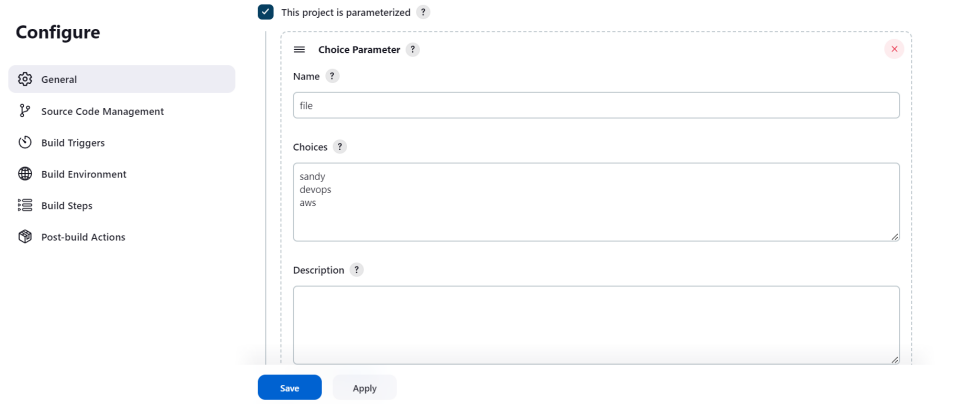
Job → Configure → build steps → Execute shell → Enter the command → save & build



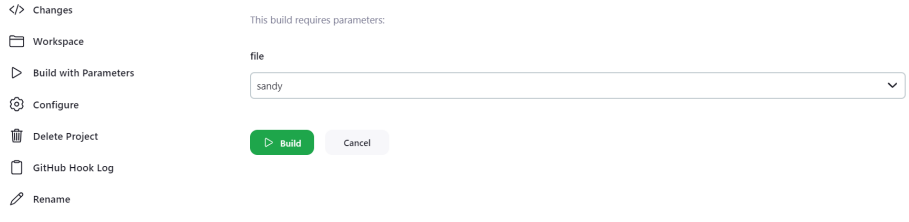
After build, Check in server → go to workspace → job → we got file data

So, above step is we are creating a file through jenkins

Now, $filename it is variable name, we have to mentioned in choice parameterized inside the name



Save and build we got options select the file and build, we will see that output in server

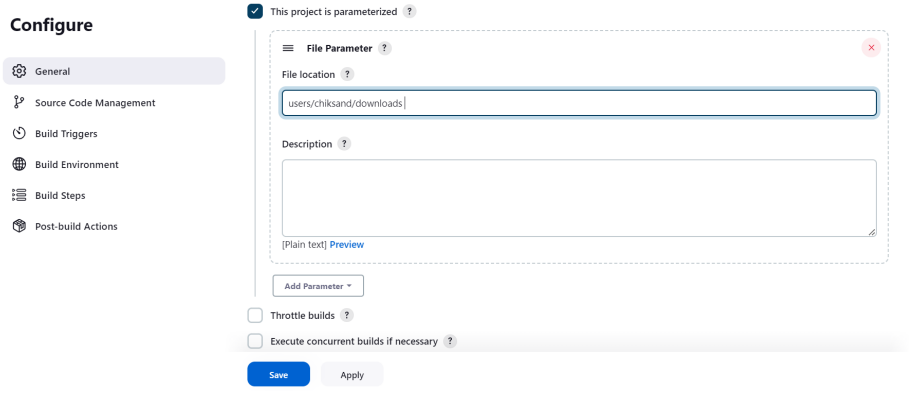


So, overall it provides the choices. Based on requirements we will build

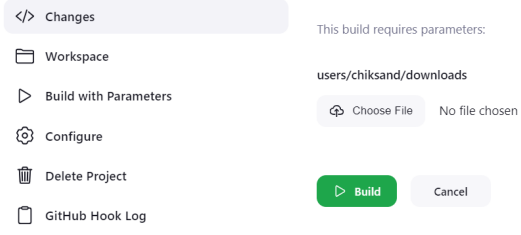
So, every time we no need to do configure and change the settings

File parameter

- This parameter is used when we want to build our local files
- Local/computer files we can build here
- file location → starts with user
- select a file and see the info and copy the path eg: users/sandeep/downloads



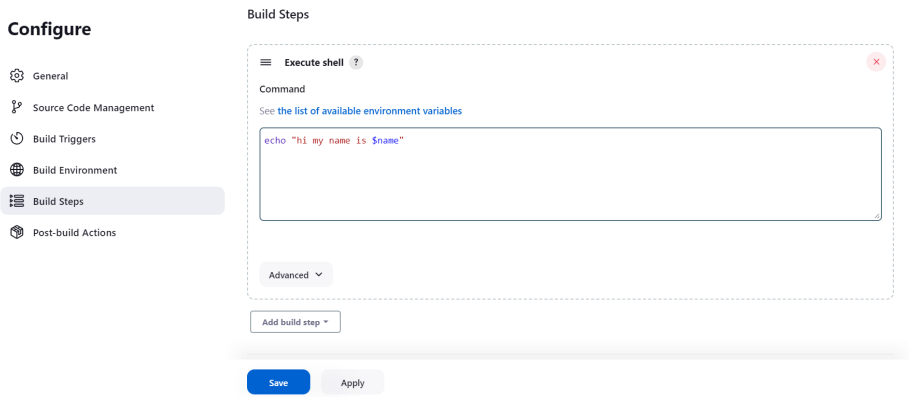
- **Build with → browse a file → open and build**



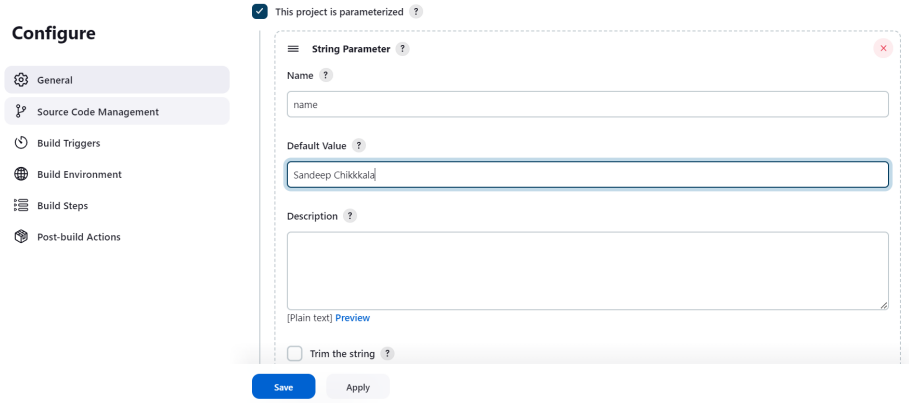
- **So, here at a time we can build a single file**

☐ String parameters (for single line)

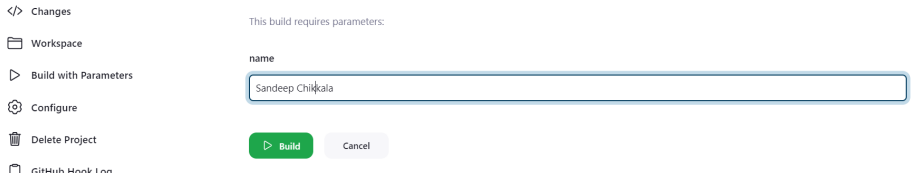
- **This parameter is used when we need to pass an parameter as input by default**
- **String it is a group/sequence of characters**
- **If we want to give input in the middle of the build we will use this**
- **first, write the command in execute shell**



Then write the data in string parameter



Save & build

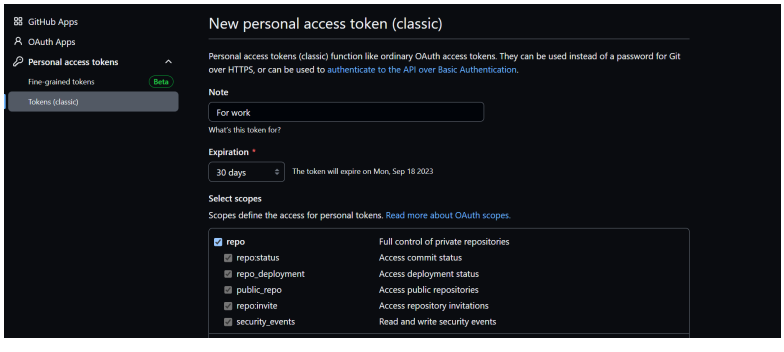


Multi-line string parameters(for multiple lines)

- Multi-line string parameters are text parameters in pipeline syntax. They are described on the jenkins pipeline syntax page
- 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

How to access the private repo in git

1. Copy the github repo url and paste in under SCM. It is showing error
2. So, now in your AWS terminal → Install GIT → yum install git -y
3. Now, we are using private repo then we have to create credentials.
4. So, for credentials Go to github, open profile settings → developer settings → personal access tokens → Tokens(classic) → Generate new token (general use) → give any name



Same do like above image and create token. So this is your password

- Now, In jenkins go to credentials → add credentials → select username and password → username (github username) → password (paste token) →ID (ec2-user) → Description(github-credentials) → save

So, whenever if you want to get private repo from github in jenkins follow above steps

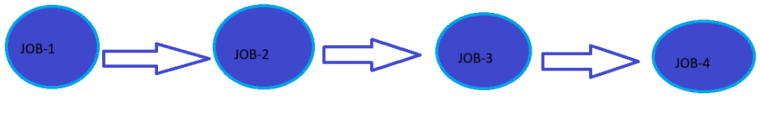
Linked Jobs

This is used when a job is linked with another job

Upstream & Downstream

An upstream job is a configured project that triggers a project as part of it execution.

A downstream job is a configured project that is triggered as part of a execution of pipeline

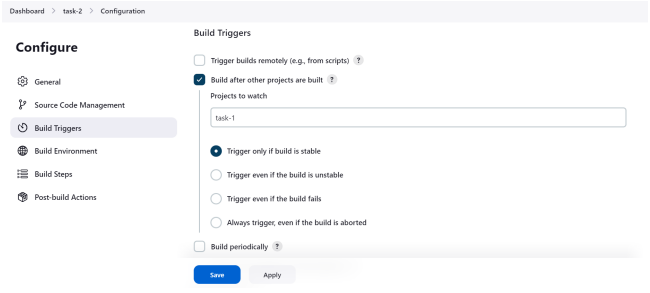


So, here I want to run the jobs automatically i.e., here we need to run the 1st job, So automatically job-2, job-3 has also build. Once the 1st build is done

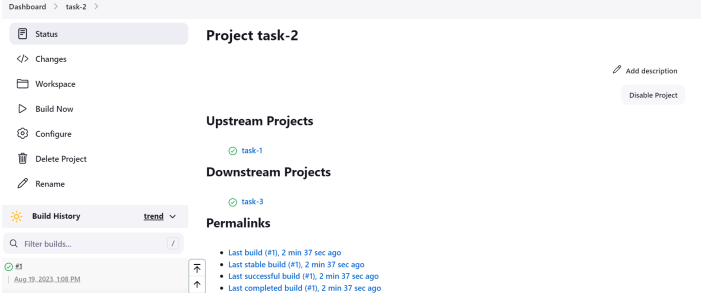
- Here for Job-1, Job-2 is downstream
- For Job-2 upstream is Job-1 and downstream is job-3 & Job-4
- For Job-3 upstream is Job-1 & Job-2 and downstream is Job-4
- For Job-4 both Job-1 & Job-2 & Job-3 are upstream

So, here upstream and downstream jobs help you to configure the sequence of execution for different operations. Hence, you can arrange/orchestrate the flow of execution

- First, create a job-1 and save
- Create another job-2 and here perform below image steps like this and save. So do same for remaining job-3 and job-4



- So, we can select based on our requirements
- Then build Job-1, So automatically other jobs builds also started after successfully job-1 builded. because we linked the jobs using upstream and downstream concept
- If you open any task/job, It will show like below

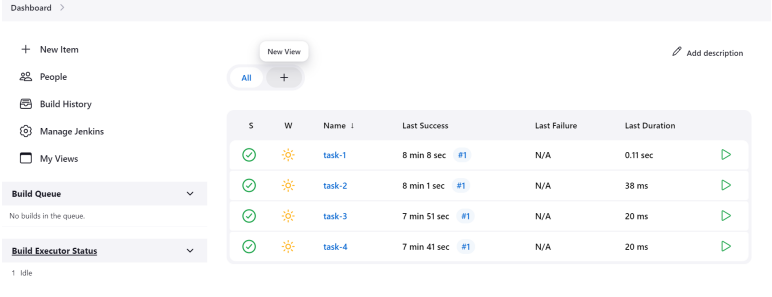


In this dashboard we can see the changes, this is step by step pipeline process

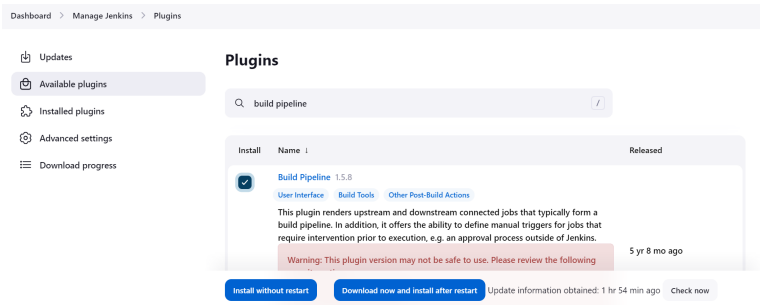
Create the pipeline in freestyle

If I want to see my pipeline in step by step process like above. So, we have to create a pipeline for these

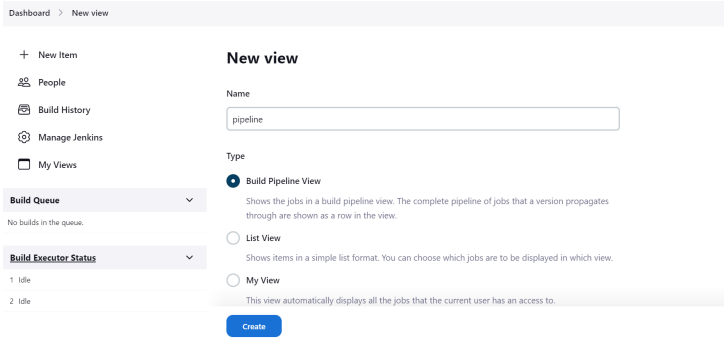
- In dashboard we have builds and click on (+) symbol



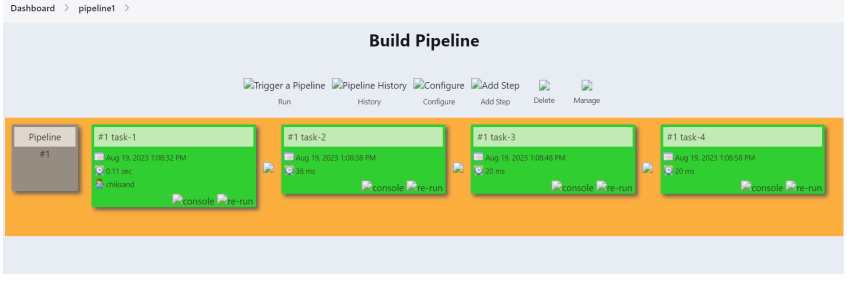
- But we need plugin for that pipeline view
- So, Go to manage jenkins → plugins → available plugins we need to add plugin - (build pipeline) and click install without restart



- once you got success, go to dashboard and click the (+ New view) symbol



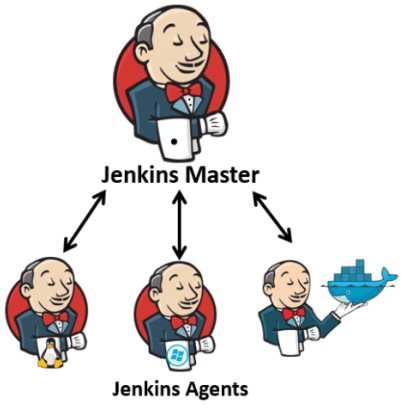
- Perform above steps and click on create and select initial job - Job1, So here once job-1 is build successfully so remaining jobs will be automatically builded
- Don't touch/do anything and click OK
- So, we can see the visualized pipeline below like this



- Here, when you click on 'RUN' Trigger a Pipeline you got the view,
- Here, trigger means it is in queue/progress for build
- whenever, you refresh the page, trigger will change from old job to new job
  - history : If you want to see history, select above pipeline history option
  - Configure : This is option in pipeline, If you want to configure the job instead of Job-1, click this
  - Add Step : Suppose, you want to add a new job after Job-4
    - So, first create a job, with option build after other projects, and give Job-4
    - So, we have that in pipeline and when you click on run
- But, If your new job wants to come in first (or) middle of the pipeline you have to do it in manually

Note : If parameters is on inside a job means we can't see the pipeline view

Master & Slave Architecture



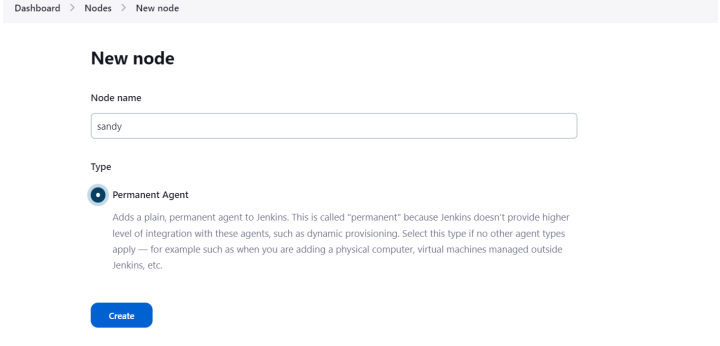
- Here, the communication between these servers, we will use master & slave communication
- Here, Master is Jenkins server and Slave is other servers
- Jenkins uses a Master-Slave architecture to manage distributed builds.
- In this architecture, master & slave nodes communicate through TCP/IP protocol
- Using Slaves, the jobs can be distributed and load on master reduces and jenkins can run more concurrent jobs and can perform more
- It allows set up various different environments such as java, .Net, terraform, etc..,
- It supports various types of slaves
  - Linux slaves
  - Windows slaves
  - Docker slaves
  - Kubernetes slaves
  - ECS (AWS) slaves
- If Slaves are not there means by default master only do the work

Setup for Master & Slave

1. Launch 3 instances at a time with key-pair, because for server to server communication we are using key-pair
   a. Here name the 3 instances like master, slave-1, slave-2 for better understanding
   b. In master server do jenkins setup
   c. In slave servers you have to install one dependency i.e. java.
   d. Here, in master server whatever the java version you installed right, same you have to install the same version in slave server.
2. Open Jenkins-master server and do setup
   a. Here Go to manage jenkins → click on set up agent

(or)

Go to manage jenkins → nodes & clouds → click on new node → Give node name any → click on permanent agent and create



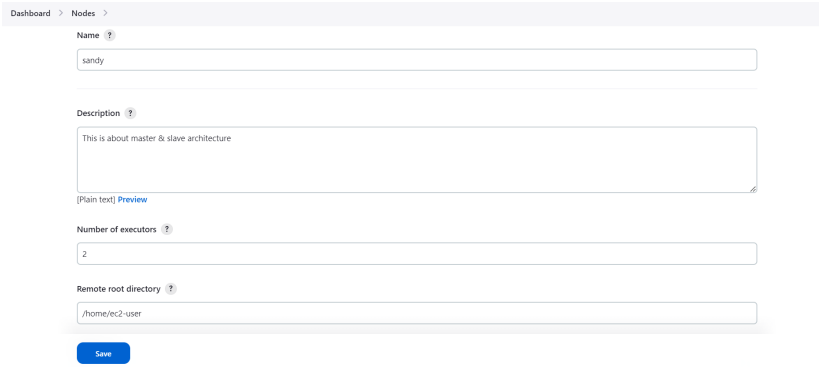
b. Number of executors -

- Default we have 2 executors.
- Maximum we can take 5 executors

If we take more executors then build will perform speed and parallely we can do some other builds. For that purpose we are taking this nodes

c. Remote root directory -

- we have to give slave server path. Here, jenkins related information stored here



So, on that remote path jenkins folder created. we can see build details, workspace, etc..,

d. Labels -

- When creating a slave node, Jenkins allows us to tag a slave node with a label
- Labels represent a way of naming one or more slaves
- Here, we can give environment (or) slave names
- i.e. dev server - take dev
- production server means take prod (or) take linux, docker

e. Usage -

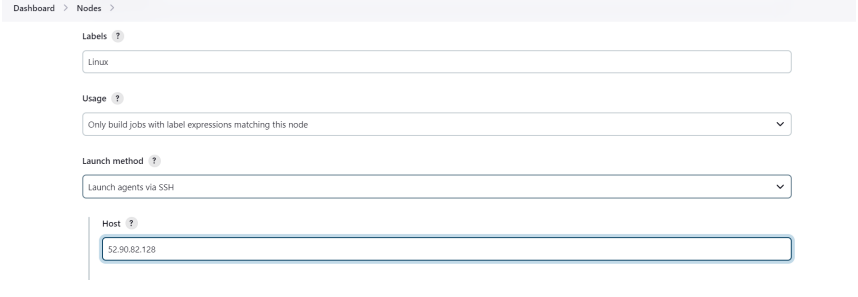
- Usage describing, how we are using that labels .!
- Whenever label is matches to the server then only build will perform
- i.e. select “only build jobs with label expressions matching this node”

f. Launch method -

- It describes how we are launching master & slave server
- Here, we are launching this agents via SSH connection

g. Host -

- Here, we have to give slave server public IP address



h. Credentials -

- Here, we are using our key-pair pem file in SSH connection

- Here, In the key you have to add the slave key-pair pem data

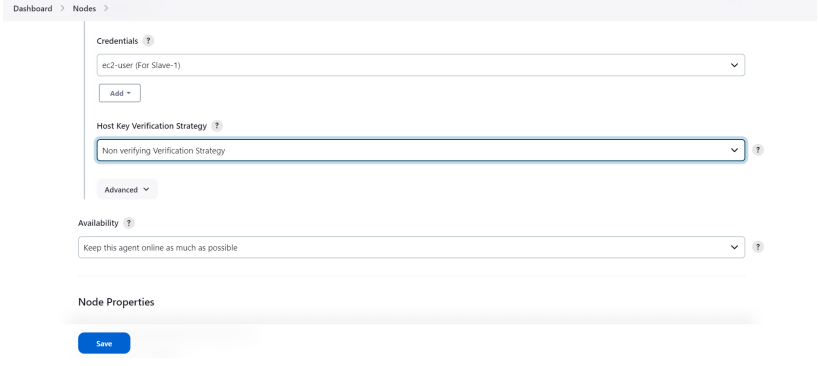
- click on add and select this credentials

g. Host Key Verification Strategy -

- Here, when you are communicating from one server to another server, on that time if you don't want verification means
- we are selecting "Non verifying verification strategy" option

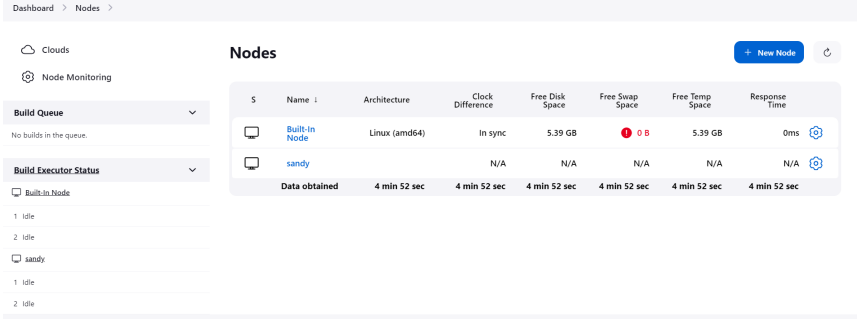
h. Availability -

- We need our Agent always must be running i.e., keep this agent online as much as possible



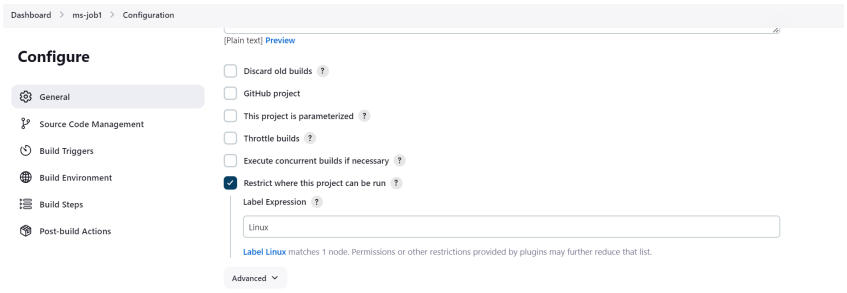
Perform above steps and Click on save

Here, If everything is success means we will get like below image



Note: Sometimes in Build Executor status under, It will shows one error. That is dependency issue. For that one you have to install the same java version in slave server, which we installed in master server

- Now, Go to Jenkins dashboard, create a job



- select above option and give label name
- create one file in execute shell under build steps.
- perform save & build

So, whatever the jobs data we're having, we can see in slave server. not in master.

Because you're using master & slave concept that means slave is working behalf of master.

Note :

- If you don't give the above Label option inside a job means, it will runs inside a master
- If you want to work with another slave server, create another node and use that node as a label

  This is all about Master & Slave Architecture in Jenkins

User Management in Jenkins

For security configuration purpose we're using user management

1. Security is all about authentication and authorization.
2. By default, jenkins requires username and password to access
3. By default, all new users will have full access
4. Jenkins stored details about users in local file system
   - In the real word, we use third party identity management systems such as active directory, LDAP etc..,
5. Here, we are having 2 types
   a. Role-based strategy
   b. Project based Matrix Authorization strategy (Advanced concept)
   c. Matrix-based security

a. Role-based strategy

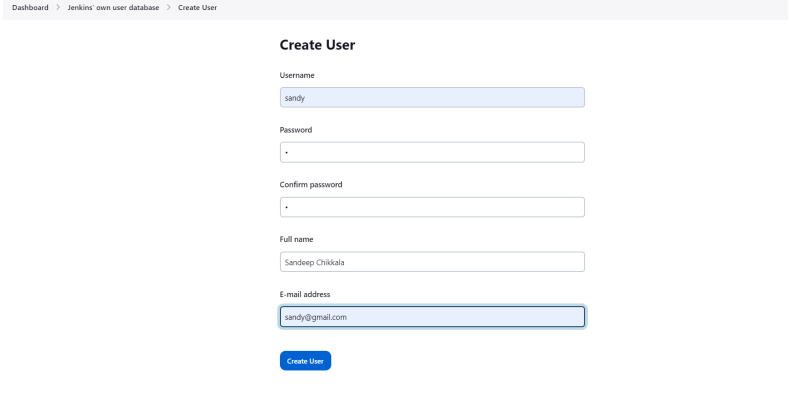
In our dashboard, we have 3 main roles

a. Developer - Here we can give read permissions i.e., he can see the build
b. Tester - Read, cancel, testing permissions we can give
c. DevOps - Here we can give full permissions

Steps :

Default we're having one user. Go to dashboard → people → we can see users

1. Add Users : Go to manage jenkins → users → create user

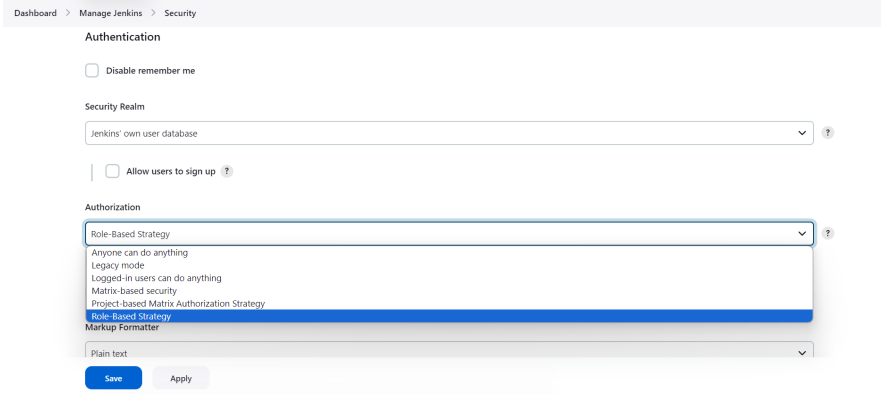


Here, we can't directly mention the roles. For that we need plugin

Go to manage plugins → Add plugins → Role-based Authorization Strategy → Install

2. Configure the plugin

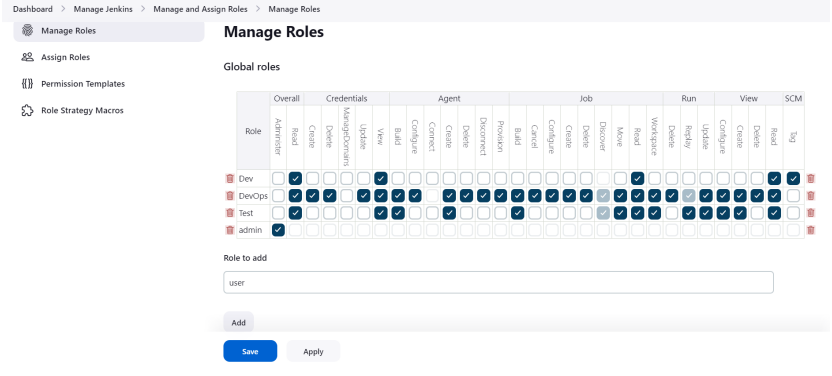
- Go to manage jenkins → Security → Authentication → select role-based strategy → save



- Once you configured the plugin, automatically you will get a new feature in manage jenkins i.e.., manage & assign roles

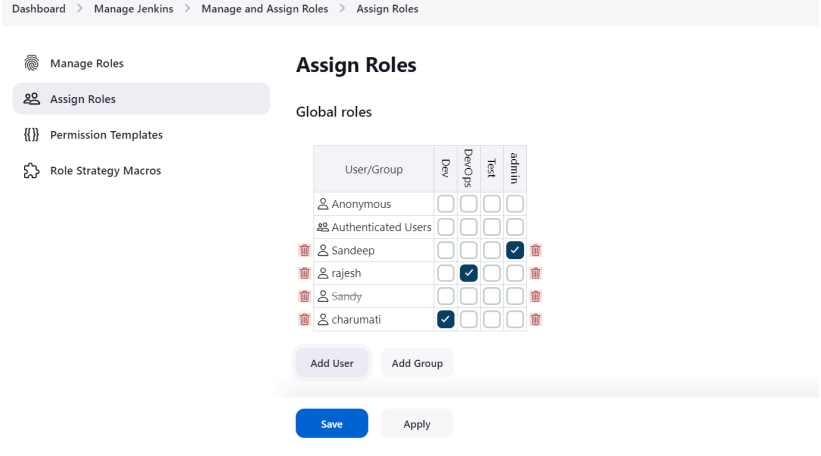
3. Adding roles

- Now, go inside Manage & Assign roles → Manage roles → Add roles
- Give the permissions for developer, tester and check the boxes based on their roles and save
- eg: Developer can only see the view, DevOps engineer can do anything like that



4. Assign the roles

- In the above path we're having assign roles
- Go inside → Add User → give user name → save



- If you give wrong user name, it will take but we can see the user name is striked
- Do Above process, save

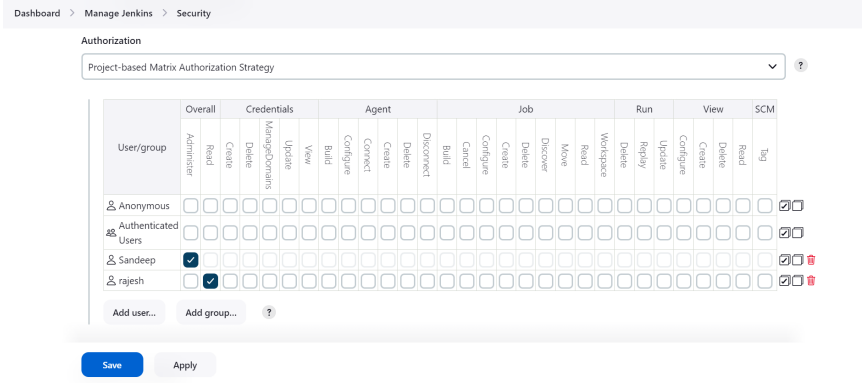
5. Login

- After done above 4 steps, click on log out and login as another user
- Go to dashboard, Here we can see the changes
- Like this you can login as multiple user and do perform the operations

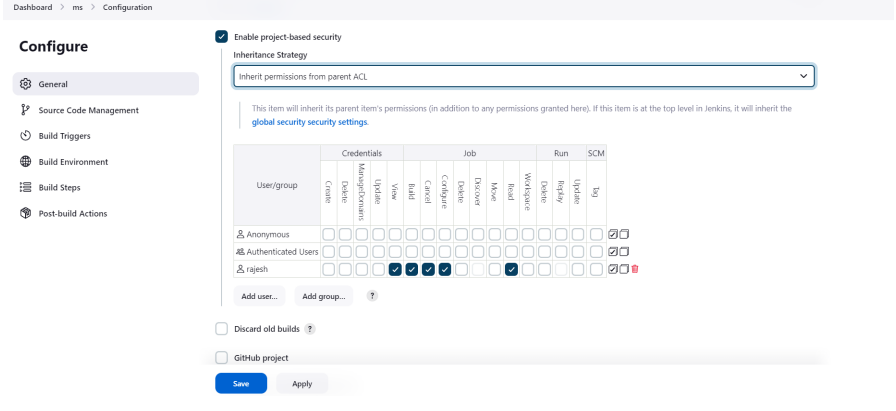
b. Project-based matrix authorization strategy

Here, we can give job-level permissions. that means specific users can access only specific jobs

1. First install the plugin - Role-based authorization
2. Go to manage jenkins → add user → save
3. Go to security → Authorization → project-based matrix authorization strategy → add user → give either read/view any permissions → save

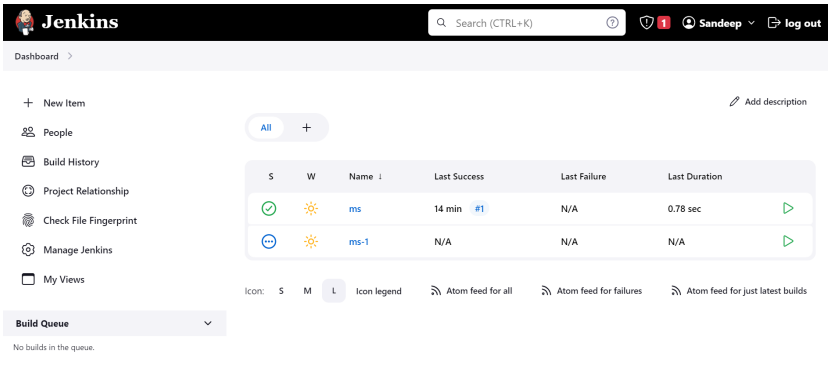


4. Go to dashboard → select a job → configure → click enable project based security → add user → give permissions → save

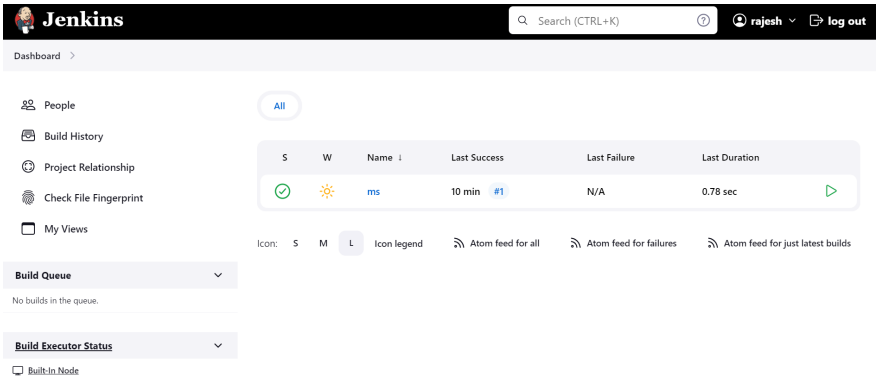


Now, that job is only access for that particular user

FYI, open dashboard and see the jobs



5. Logout and login as another user



Now that user can see only that particular job in his dashboard. User can't see any jobs

This is the way you can restrict the users inside a job

JENKINS-PIPELINE

- 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
- Here, using Groovy syntax we're writing a pipeline

We have 3 types of pipelines

1. Freestyle pipeline
2. scripted pipeline
3. Declarative pipeline

Difference between freestyle and pipeline

- In pipeline, we are writing the script for deployment. It is updated
- In freestyle we are having manual options we can go through that. It is little bit old
- In real time we use 2 pipelines based on our requirement

Jenkins file - it is nothing but a file, it contains the scripted (or) declarative code

Scripted pipeline syntax:

```
Eg: node {
    stage ("stage 1") {
    echo "hi"
                }
        }
```

Declarative pipeline syntax:

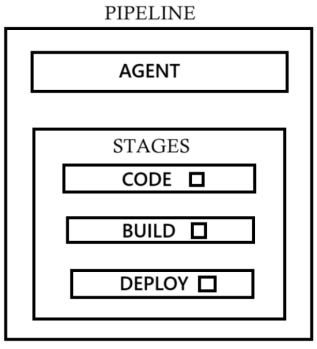
```
        pipeline {
        agent any
        stages {
            stage("code") {
                steps {
                    echo "hi"
                    }
                }
```

```
    }

}
```

Here, In our pipeline we're using declarative syntax

Declarative pipeline :



- Here, pipeline is a block
- In this block we have agents
- Through agent we will decide in which server we have to run our tasks/job
  - So, here we created a label, through label we will define
- Inside the stages we have multiples stages
  - Eg: Code, build, test, deploy
- Inside every stages we have one step
- Inside the steps we can write our code/commands

Launch jenkins server and open dashboard

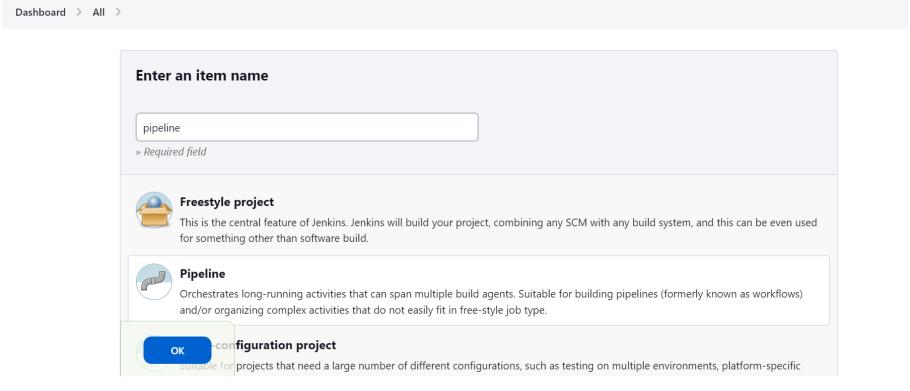
1. Create a job → select pipeline → OK

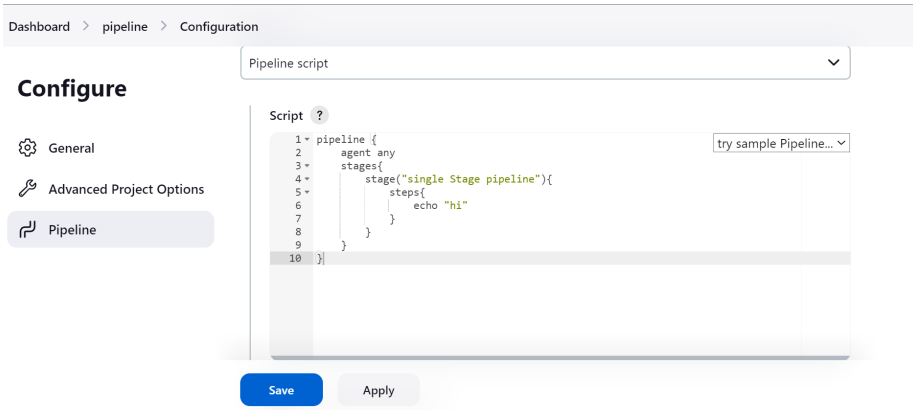
2. Select pipeline → here we have to write the groovy syntax

3. Write the script

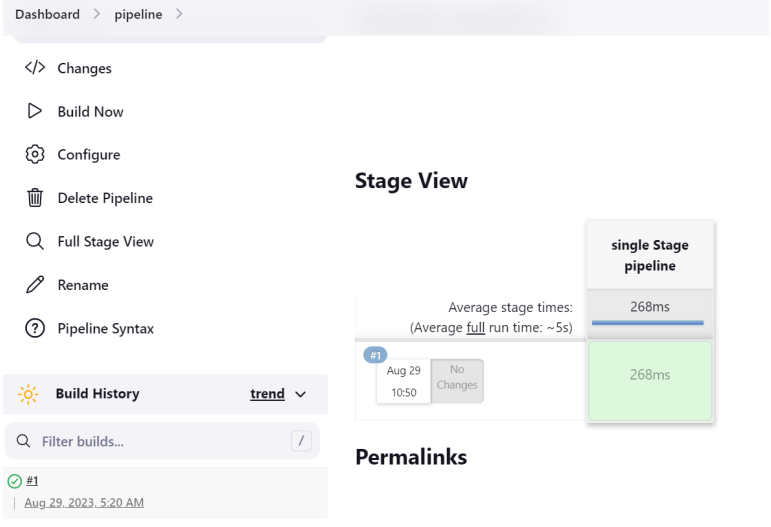
☐ Single stage pipeline

Here, automatically indentations will take i.e. a tab space (or) 4 spaces

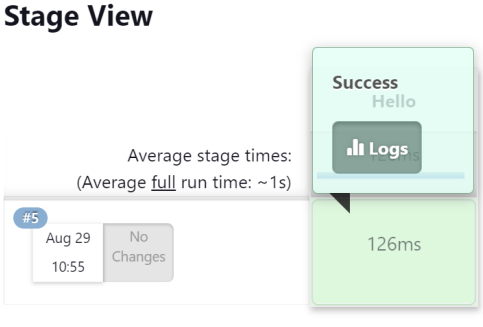




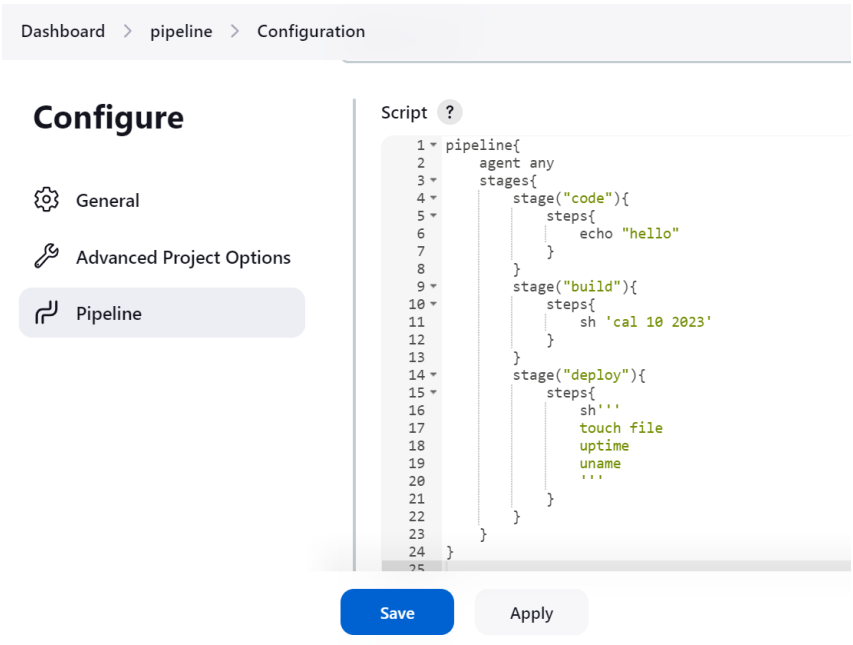
- Once you write your script → build
- GUI will be different. Here we can see step by step process



- If you want to see the output, click on the build view click on logs



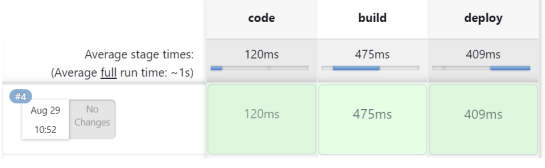
- [ ] Multi stage pipeline



Click on build, you will see the o/p like given below

Pipeline pipeline

Stage View



Variables :

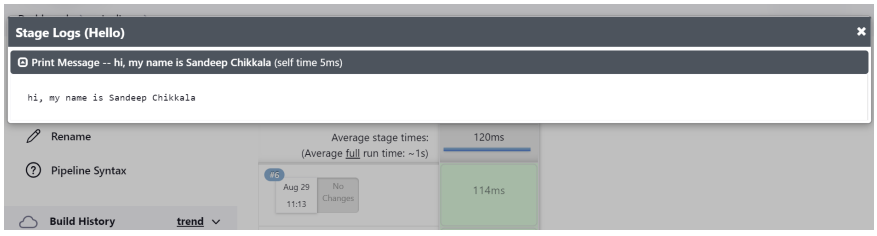
variables are used to store the values (or) data. Here, we are having 2 types of variables

1. Global variable
2. Local variable

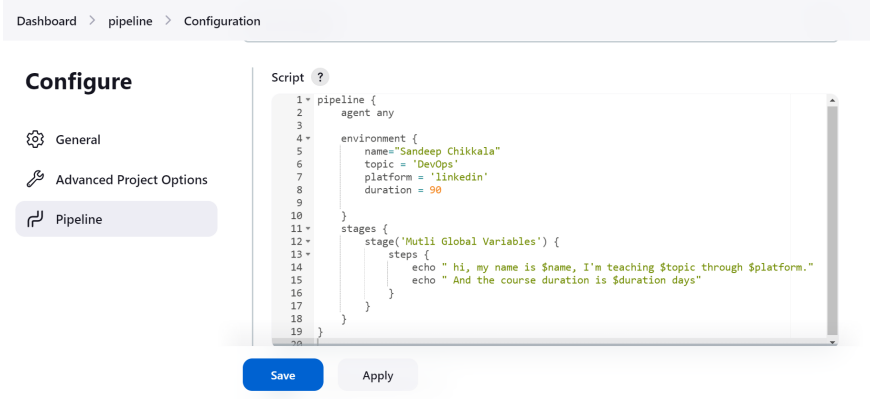
Global variable

- Here, we're declaring the environment variable after the agent.
- And we have to use $variable in stages to call the variables

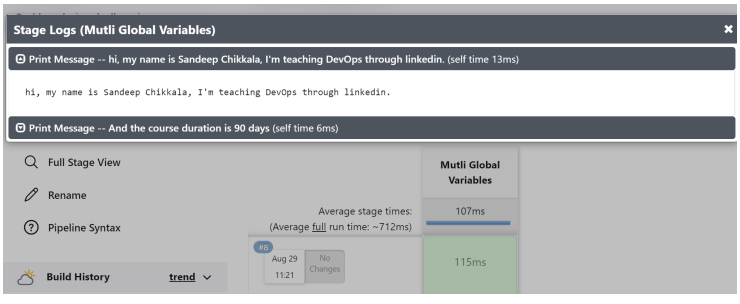
Click on build and click on logs to see the output



Multiple Global variables

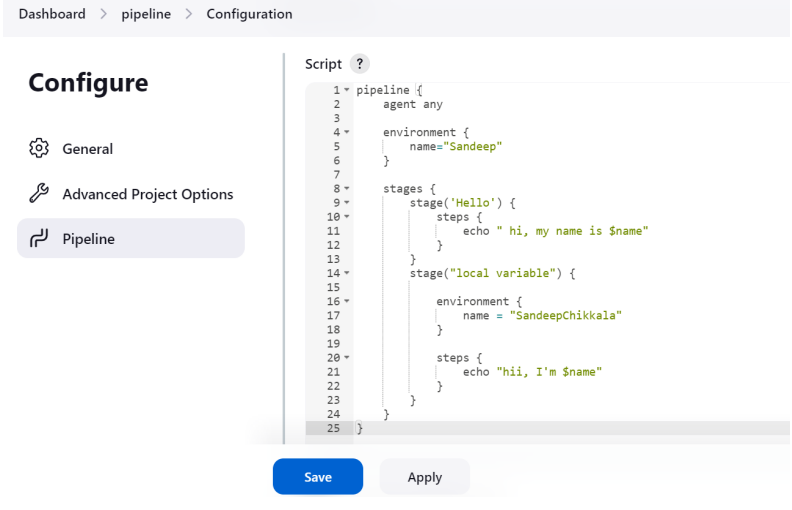


Click on build and click on logs to see the output

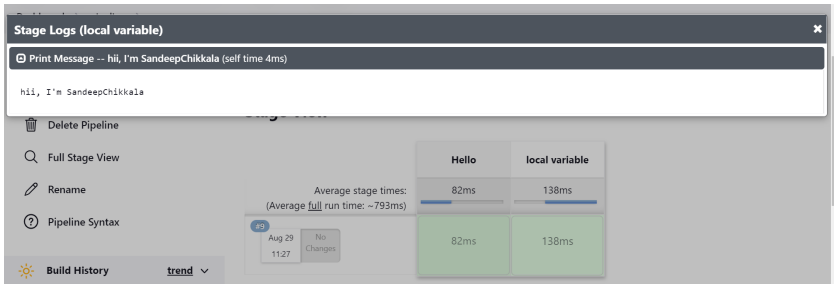


Local variable

- Local variable override the Global variable
- We're declaring local variable inside the stages



Click on build and here we have 2 stages. First is global and second is local variable. Now, you can easily find out the difference between local and global



So, when we're using local variable means, some specific/particular stage we need another value. On that case we're using local

This is all about local and global variables

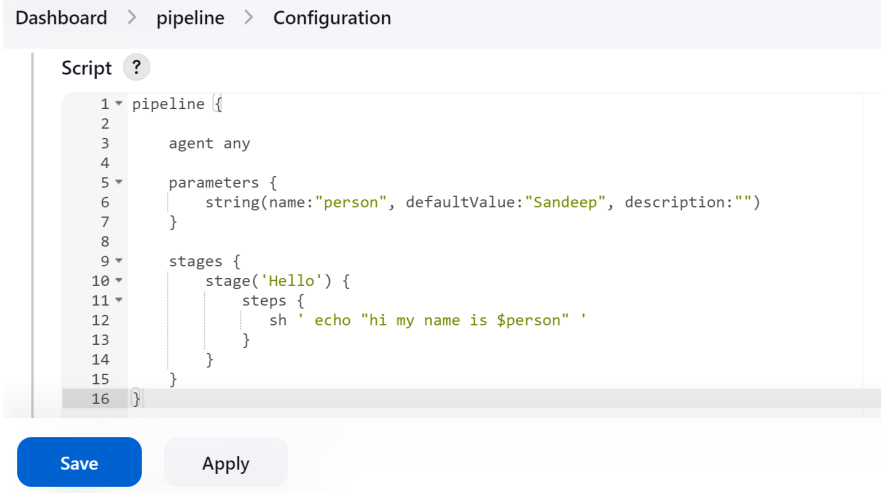
▢ Parameters pipeline

Instead of manually selecting parameters, we can write the code in pipeline

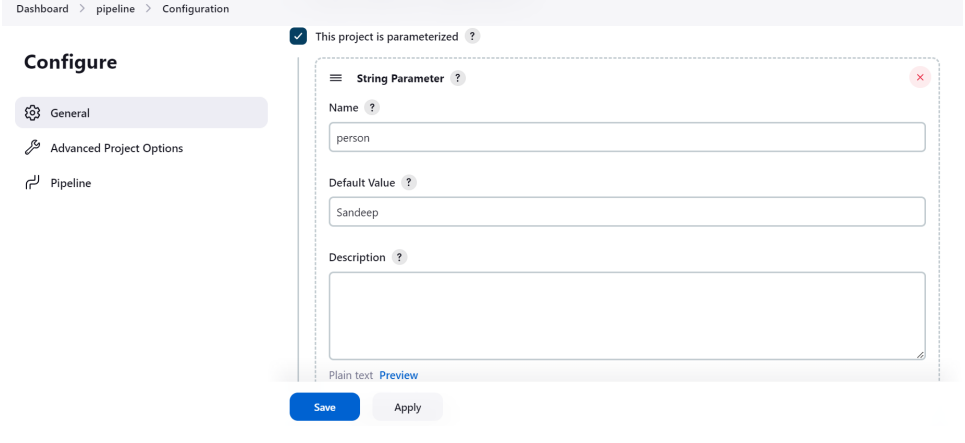
- For the first time build, Automatically selecting the parameters based on our code.
  - for the 1st build → code will executed
- After 1st build, Go to configure and check the parameters selected (or) not and do save
- For the second time build, click on build with parameters, we can see the output
  - for the 2nd build → parameters executed
- Here, Overall we have to build 2 times to see our output
- We have to take parameters block after the agent

Whenever we're using parameters we don't need to use Environment block

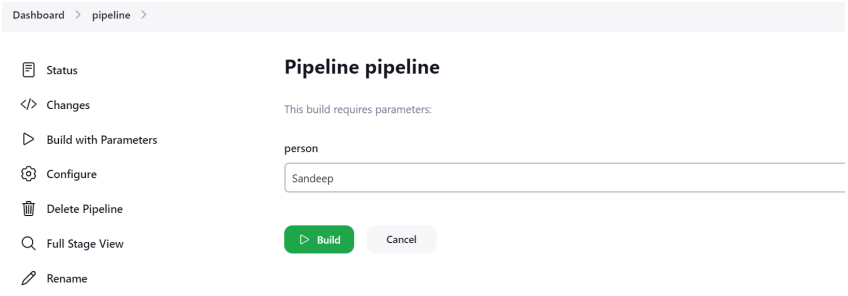
String parameter pipeline



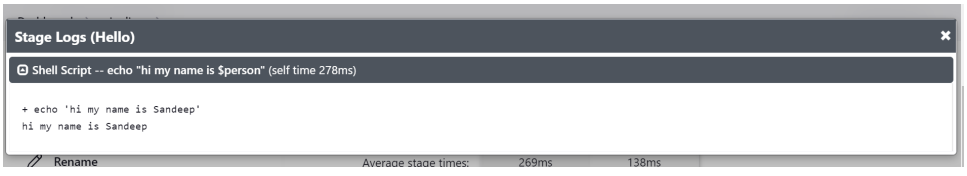
This is our code, click on save and build. Here, our code will get executed



- After the first time build, Automatically selecting the parameters based on our code.

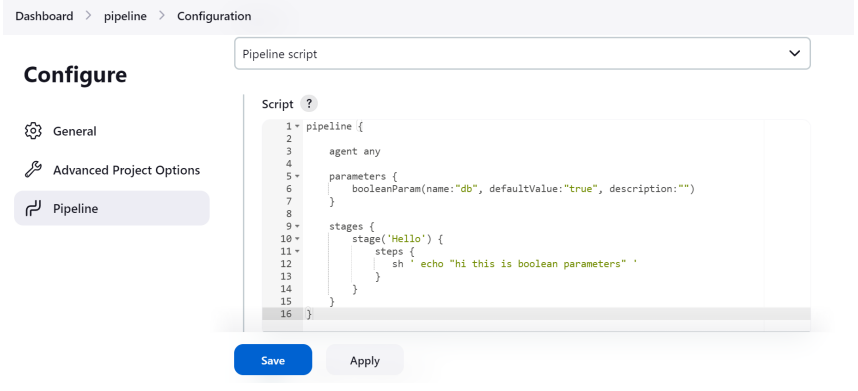


- Click on build with parameters, you will get above image and now click on build

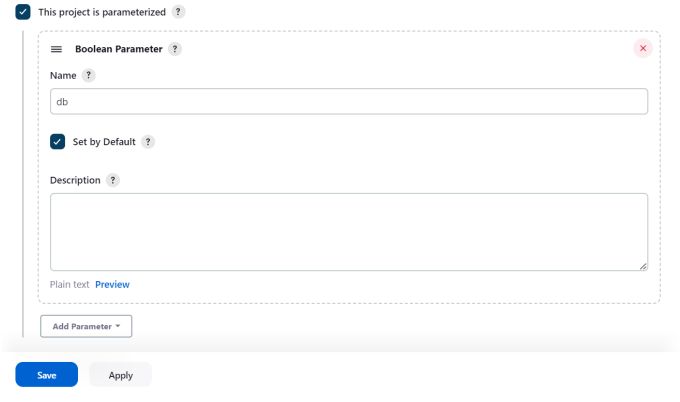


This is all about string parameters

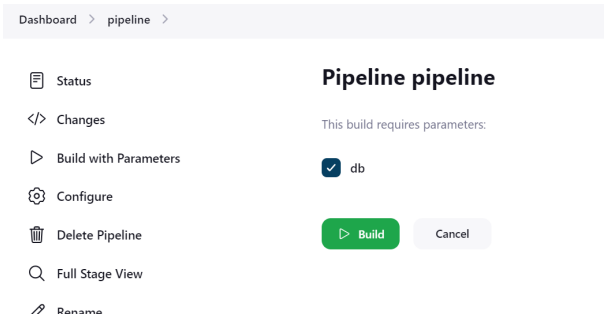
Boolean parameter pipeline



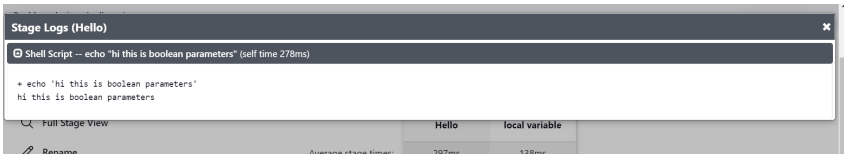
This is our code, click on save and build. Here, our code will get executed



- After the first time build, Automatically selecting the parameters based on our code.

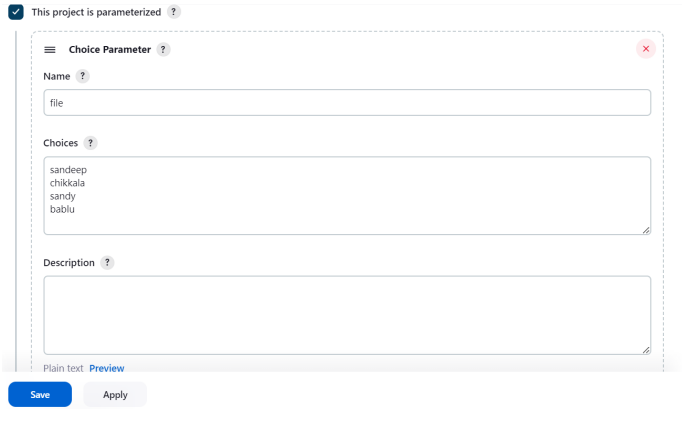


- Click on build with parameters, you will get above image and now click on build
- In the above code We written defaultValue is true. So, db checkbox is enabled. if we write false it is disabled

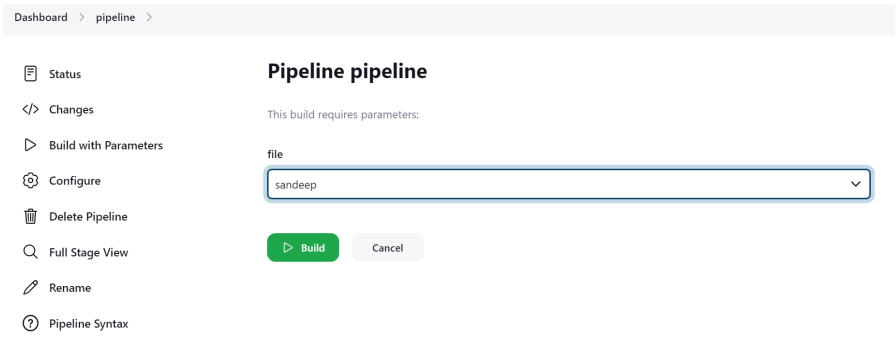


Choice parameter pipeline

This is our code, click on save and build. Here, our code will get executed



- After the first time build, Automatically selecting the parameters based on our code.



- Here, we select the file names based on our requirements
- Click on build with parameters, you will get above image and now click on build

- After build click on logs we can see the output

▢ Input function pipeline

It takes the input from the user, based on the input it will performs the operations

- Here, we are taking the input from the user
- If User said OK means build will happen
- If User said NO means build will fail

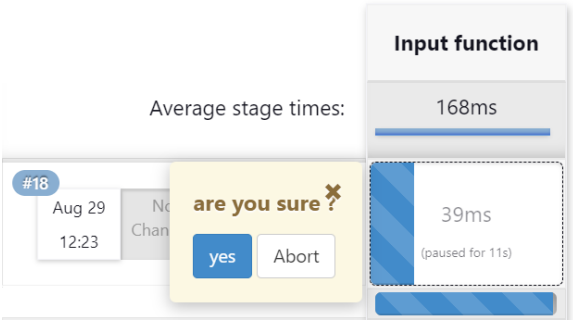
So, here we are having one condition. That condition we can called input function

- Here continuous integration performed. i.e. build +test
- But when it comes to deploy stage. It has to be asked the input from the user

Real-time Scenario :

- Whenever you're doing deployment, this input function we have to give to approval manager. So, manager check everything. If everything is correct he will click OK i.e. he will approve the deployment
- Here, how we're giving the permissions means we're using role based strategy and for all managers we have to give full build perform permissions.

Click on save and build you will get below image



Here, if we click on yes means build will success, Click on abort means build aborted/stopped

Once you click on yes, you will get below image

This is all about input function

▢ Post Build Actions/Functions pipeline

A Jenkins Post-build action is a task executed after the build has been completed

- When you perform build, you won't care about the build whether it is success(or) fail. Automatically, you want to build the particular stage
- on that case we're using post build actions

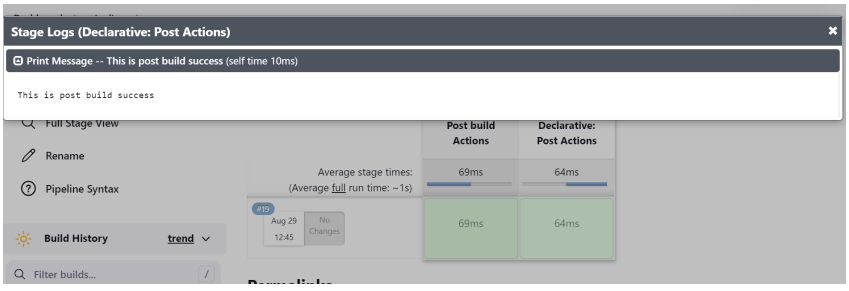
Here, we are having post conditions in jenkins

1. Always
2. Success
3. Failure

Success:

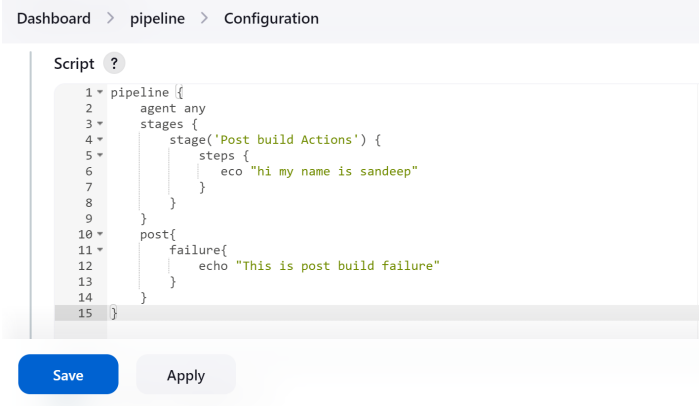
When the above stage build gets success means, then the post block will executed

click on save and build

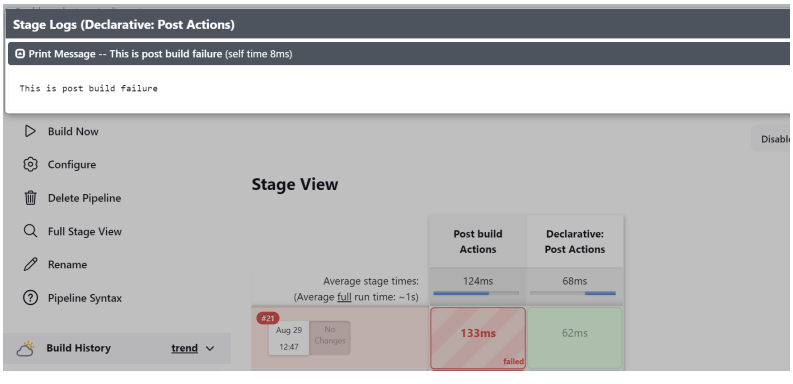


Failure:

When the above stage build gets failed means, then the post block will executed

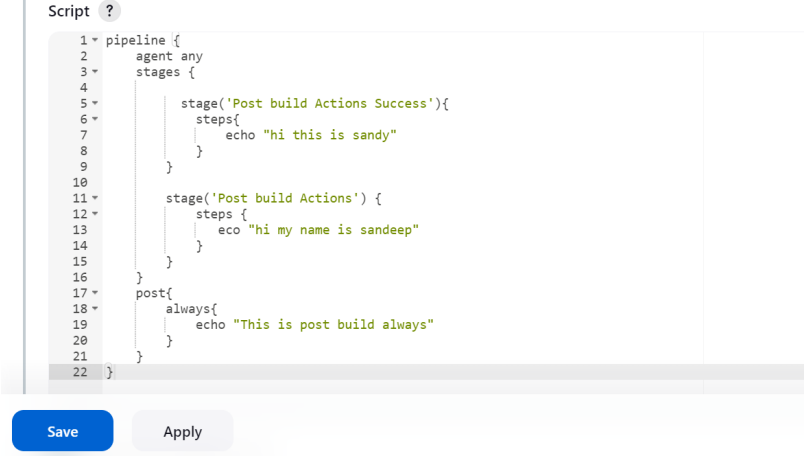


click on save and build

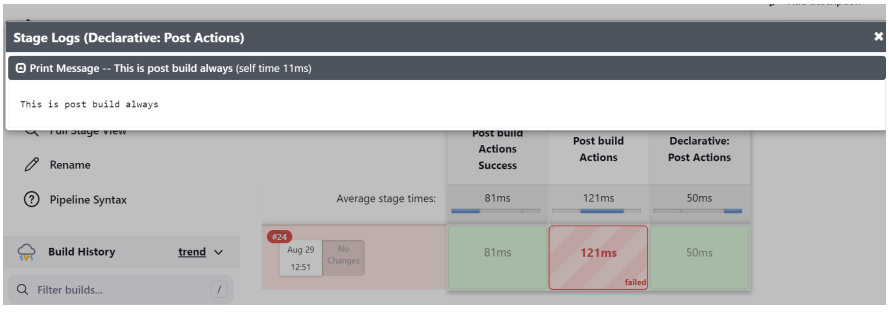


Always:

When the above stage build either success (or) Failure. This post block don't care it will always executed



Click on save and build



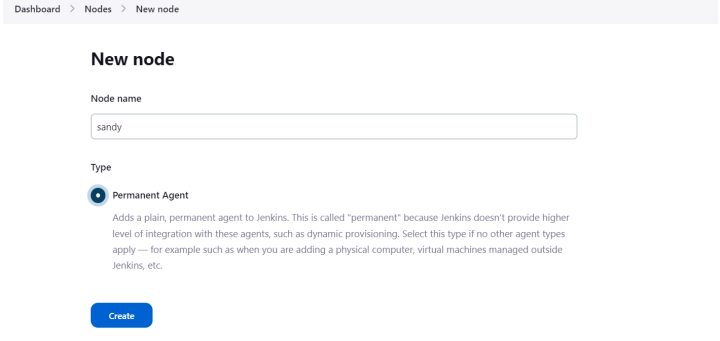
This is all about Post-Build Actions

Setup for Master & Slave

1. Launch 3 instances at a time with key-pair, because for server to server communication we are using key-pair
   a. Here name the 3 instances like master, slave-1, slave-2 for better understanding
   b. In master server do jenkins setup
   c. In slave servers you have to install one dependency i.e., java.
   d. Here, in master server whatever the java version you installed right, same you have to install the same version in slave server.
2. Open Jenkins-master server and do setup
   a. Here Go to manage jenkins → click on set up agent

(or)

Go to manage jenkins → nodes & clouds → click on new node → Give node name any → click on permanent agent and create



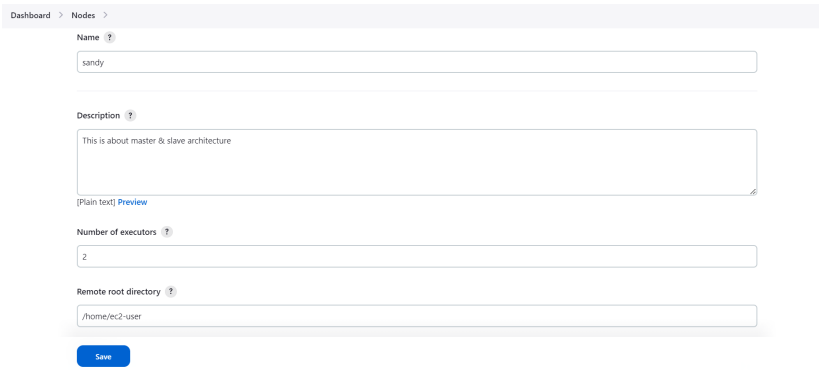
b. Number of executors -

- Default we have 2 executors.
- Maximum we can take 5 executors

If we take more executors then build will perform speed and parallely we can do some other builds. For that purpose we are taking this nodes

c. Remote root directory -

- we have to give slave server path. Here, jenkins related information stored here



So, on that remote path jenkins folder created. we can see build details, workspace, etc..,

d. Labels -

- When creating a slave node, Jenkins allows us to tag a slave node with a label
- Labels represent a way of naming one or more slaves
- Here we can give environment (or) slave names
- i.e., dev server - take dev
- production server means take prod (or) take linux, docker

e. Usage -

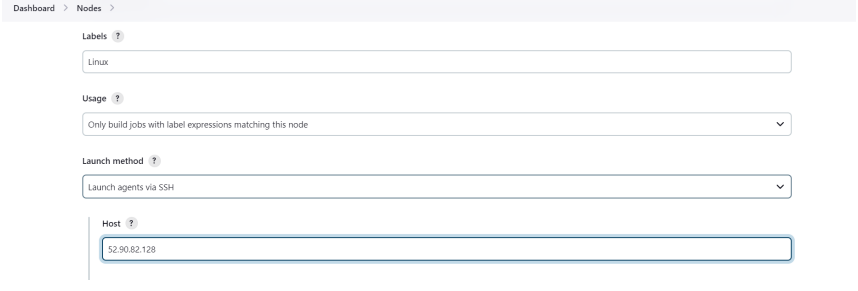
- Usage describing, how we are using that labels .!
- Whenever label is matches to the server then only build will perform
- i.e., select "only build jobs with label expressions matching this node"

f. Launch method -

- It describes how we are launching master & slave server
- Here, we are launching this agents via SSH connection

g. Host -

- Here, we have to give slave server public IP address



h. Credentials -

- Here, we are using our key-pair pem file in SSH connection

- Here, In the key you have to add the slave key-pair pem data

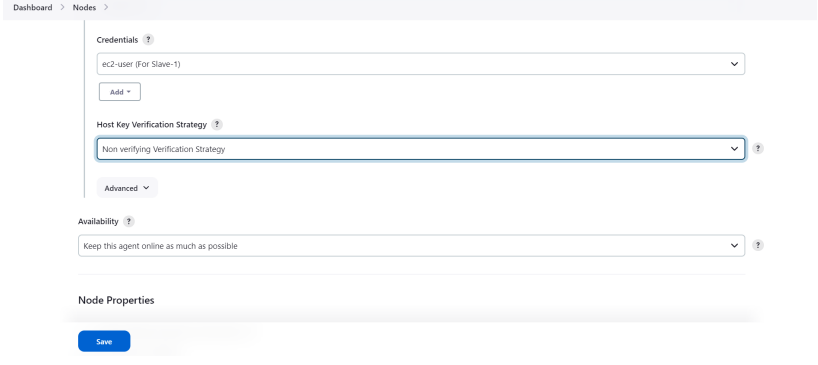
- click on add and select this credentials

g. Host Key Verification Strategy -

- Here, when you are communicating from one server to another server, on that time if you don't want verification means
- we can select "Non verifying verification strategy" option

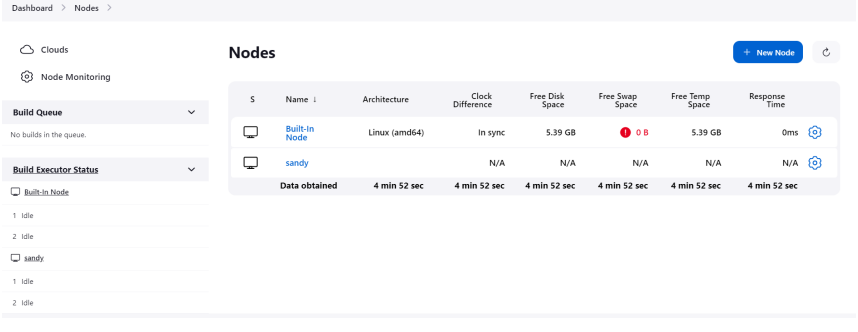
h. Availability -

- We need our Agent always must be running i.e., keep this agent online as much as possible



Perform above steps and Click on save

Here, If everything is success means we will get like below image



Note: Sometimes in Build Executor status under, It will shows one error. That is dependency issue. For that one you have to install the same java version in slave server, which we installed in master server

Now, create the pipeline for master-slave

Dashboard > pipeline > Configuration

Pipeline script

Script ?

```
pipeline{
    agent {
        label "Linux"
    }
    stages{
        stage("master-slave"){
            steps{
                sh 'touch sandy'
            }
        }
    }
}
```

Save Apply

Save and Build and see the output. Now, go to the jenkins path & check

This is all about Jenkins