



# Jenkins

# Table of Contents

1. Introduction to Jenkins
  - 1.1. What is Jenkins?
  - 1.2. Continuous Integration and Continuous Delivery (CI/CD) Explained
  - 1.3. Benefits of Using Jenkins for CI/CD
2. Jenkins Features for Better Management
  - 2.1. Timeout
  - 2.2. Timestamp
  - 2.3. Disable/Enable Job
  - 2.4. Build a Job Concurrently
  - 2.5. Retry Count
  - 2.6. Throttle Builds
3. Creating Your First Jenkins Job
  - 3.1. Click on New Item
  - 3.2. Enter the Job Name and Choose Freestyle Project
  - 3.3. Follow the Steps
  - 3.4. Click on Build Now
  - 3.5. Viewing the Console Output
4. Changing Jenkins Theme using Plugin
  - 4.1. Installing and Configuring the Simple Theme Plugin
  - 4.2. Changing the Jenkins URL
5. User Management in Jenkins
  - 5.1. Creating a New User in Jenkins
  - 5.2. Jenkins Role-Based Access Control (RBAC)
  - 5.3. Managing Roles and Assigning Permissions
6. Jenkins Integration with GitHub
  - 6.1. Building a Job without GitHub Plugin

- 6.2. Building a Job with GitHub Plugin
  - 6.3. Building a Job Using Trigger Builds Remotely (Authentication Token)
- 7. Jenkins Build Triggers
  - 7.1. Build After Other Projects are Built
  - 7.2. Build Job Periodically
  - 7.3. Poll SCM (Source Code Management)
- 8. Jenkins Job Configuration and Customization
  - 8.1. Defining Variables Globally
  - 8.2. Parameterized Jobs
  - 8.3. Custom Workspace
  - 8.4. Changing Display Name and Project Name
- 9. Building Upstream and Downstream Projects
  - 9.1. Blocking Build When Upstream Project is Building
  - 9.2. Blocking Build When Downstream Project is Building
- 10. Jenkins Pipelines
  - 10.1. Creating Jenkins Pipeline Using Build Pipeline
  - 10.2. Understanding Continuous Deployment vs. Continuous Delivery
  - 10.3. Running Two Jobs in Parallel in Jenkins Pipeline
  - 10.4. Deploying WAR to Tomcat Server Through Jenkins (Automation)
- 11. Creating Jenkins Slaves
  - 11.1. Configuring Jenkins Slaves
  - 11.2. Running Commands in Pipeline as Code
  - 11.3. Setting Environment Variables in Pipeline as Code
  - 11.4. Taking User Input in Pipeline

## Conclusion

# Jenkins

Jenkins is a powerful application that allows continuous integration and continuous delivery of projects, regardless of the platform you are working on. It is a free source that can handle any kind of build or continuous integration. You can integrate Jenkins with a number of testing and deployment technologies. In this notes, we will explain how you can use Jenkins to build and test your software projects continuously.

**Continuous Integration and Continuous Delivery (CI/CD)** are essential practices in modern software development that aim to streamline and automate the process of building, testing, and deploying code changes. These practices help development teams deliver software more efficiently, with higher quality, and at a faster pace.

## **Continuous Integration (CI):**

CI is the practice of automatically integrating code changes from multiple developers into a shared repository on a frequent basis, typically multiple times a day. The main goal of CI is to detect integration issues early in the development cycle, thereby reducing the chances of conflicts and bugs that arise when multiple developers work on the same codebase.

Key features of CI:

**Automatic code integration:** Developers frequently merge their code changes into a central repository, where automated build and testing processes are triggered.

**Automated build and tests:** CI systems automatically compile the code, run unit tests, and perform other validation checks to ensure the codebase remains stable.

**Early feedback:** By catching issues early, CI allows developers to fix problems quickly, preventing the accumulation of bugs.

### **Continuous Delivery (CD):**

CD is an extension of CI that focuses on automating the software release process. It ensures that code changes are always in a deployable state and ready for production release. The ultimate goal of CD is to enable frequent and reliable software releases to end-users with minimal manual intervention.

### **Key features of CD**

- **Automated deployment:** The CD pipeline automates the deployment of code changes to various environments, including staging and production, with consistent configurations.
- **Release orchestration:** CD pipelines manage the entire release process, coordinating tasks such as database updates and infrastructure provisioning.
- **Rollbacks and monitoring:** CD ensures that automated rollbacks are possible if any issues arise during deployment. Monitoring is also integrated to track the application's health post-deployment.

### **Benefits of CI/CD**

- **Faster time-to-market:** CI/CD streamlines the development and deployment process, reducing the time it takes to deliver new features and updates to end-users.
- **Higher software quality:** Automated testing and validation catch bugs early, leading to a more stable and reliable codebase.
- **Risk reduction:** Frequent integration and automated deployment enable faster bug identification and recovery, minimizing risks associated with manual processes.
- **Increased collaboration:** CI/CD encourages collaboration among developers, testers, and operations teams, fostering a culture of continuous improvement.

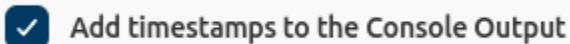
Jenkins features of every tool:

1. **Timeout**- It will be found under the Build Environment section-> Terminate a build if it's stuck.

When the system sleeps for some time e.g. 240 seconds we define the Time-out strategy time e.g. 3 minutes. It means that a job will not build or fail.

The screenshot shows the Jenkins 'Build Environment' configuration page. Under the 'Timeout' section, there is a checked checkbox labeled 'Terminate a build if it's stuck'. Below it is a dropdown menu labeled 'Time-out strategy' set to 'Absolute'. Underneath that is a field labeled 'Timeout minutes' containing the value '3'.

2. **Timestamp**- It will be found under the Build Environment section->Add timestamps to the Console Output which means it will notify the time whatever command is running at what time.



3. **Disable/Enable Job**- It will be found under the status of a specific project. You can use this feature to disable or enable the job build. If you want that nobody can build the job except you then, you can disable the job. And whenever you want to enable it you can do that too.



4. **Build a Job concurrently-** As you are aware you couldn't build a job concurrently. But if you want to do this then, you have to enable the feature that will be found under the section of Description named -> Execute concurrent builds if necessary.

Execute concurrent builds if necessary ?

5. **Retry count-** When your job is not built successfully. So you can retry as many as you want but for that, you have to enable it to define time and number of counts.

Retry Count ?

6. **Throttle Builds-** When you want that there should be a limited number of builds asked then, you will use throttle builds. For eg, the Number of builds is 3 and the Time period is minute. Then there are only 3 builds that will be successful and possible, not more than that.

Throttle builds ?

Number of builds ?

3

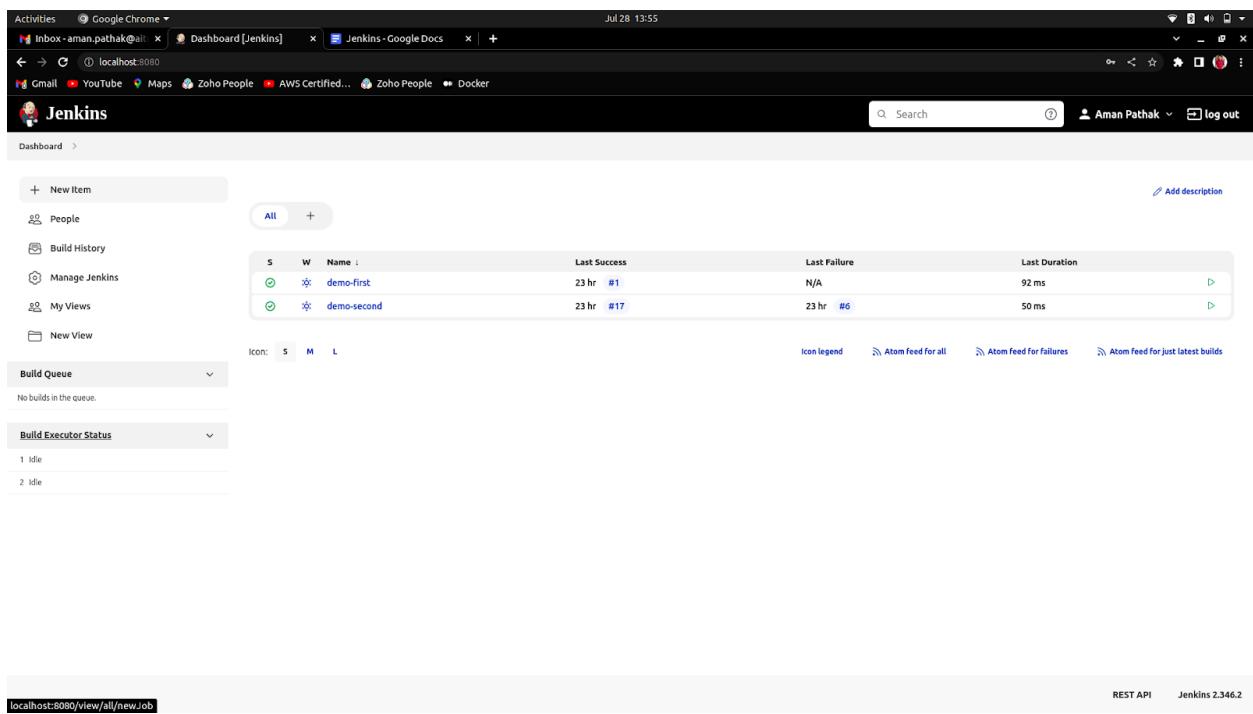
Approximately 20 seconds between builds

Time period ?

Minute

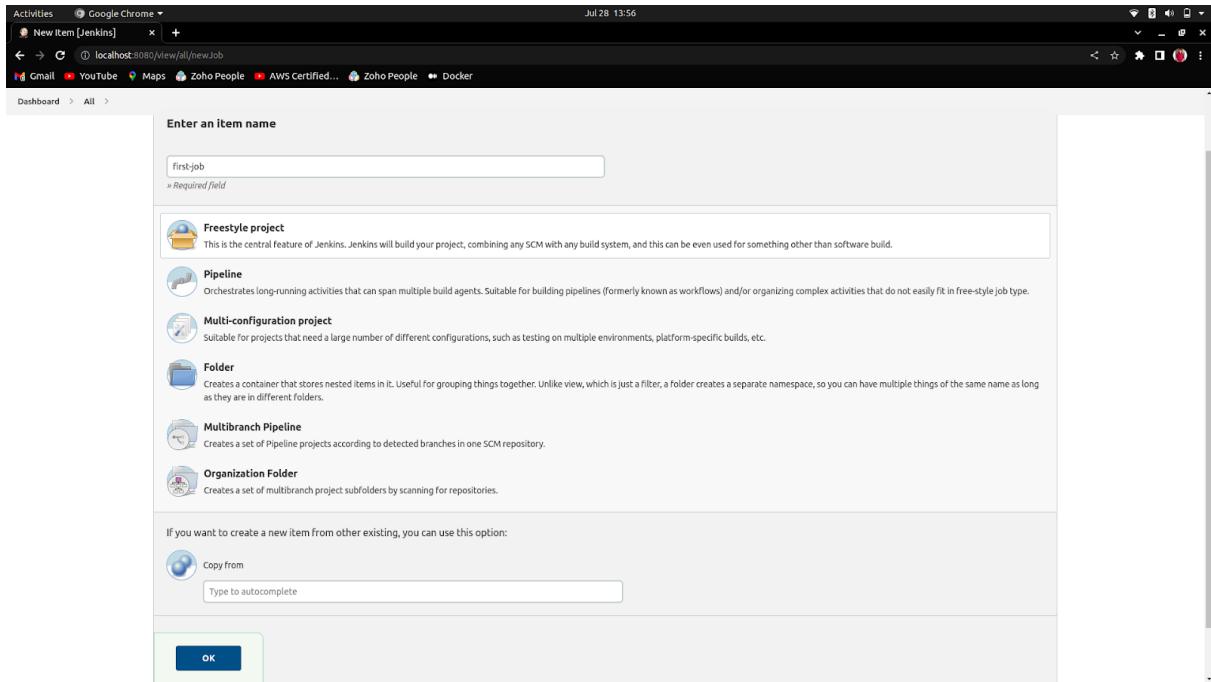
# Creating First Job

## 1. Click on New Item.

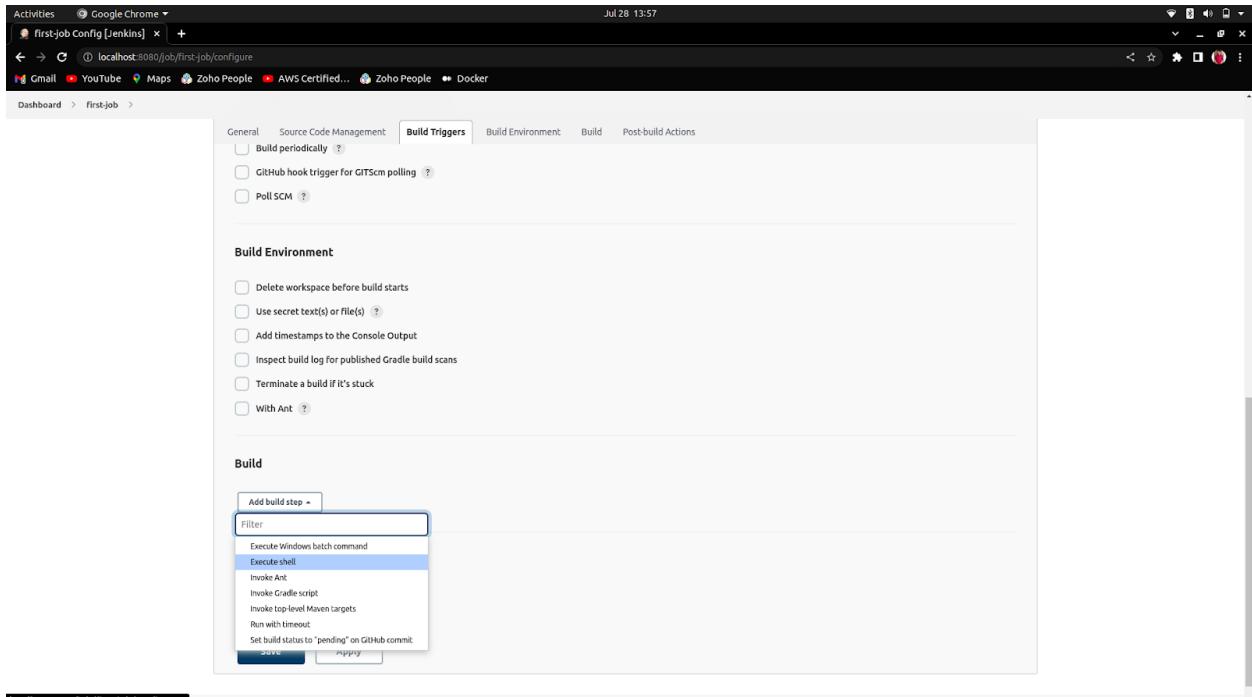


The screenshot shows the Jenkins dashboard interface. At the top, there are browser tabs for 'Inbox - aman.pathak@all...', 'Dashboard [Jenkins]', and 'Jenkins - Google Docs'. The main content area is titled 'Dashboard' and features a 'New Item' button. Below it, there are sections for 'People', 'Build History', 'Manage Jenkins', 'My Views', and 'New View'. A 'Build Queue' section indicates 'No builds in the queue.' A 'Build Executor Status' section shows '1 Idle' and '2 Idle'. On the right side, there's a table of build jobs with columns for status (S), warning level (W), name, last success, last failure, and last duration. The table contains two entries: 'demo-first' (last success: 23 hr #1, last failure: N/A, duration: 92 ms) and 'demo-second' (last success: 23 hr #17, last failure: #6, duration: 50 ms). At the bottom of the page, the URL 'localhost:8080/view/all/newJob' is visible, along with links for 'REST API' and 'Jenkins 2.346.2'.

## 2. Enter the job name and choose freestyle project.



## 3. Follow the below steps.



Activities Google Chrome ▾

first-job Config [Jenkins] + ↻ ➔ ⓘ localhost:8080/job/first-job/configure

Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > first-job >

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

Delete workspace before build starts  
 Use secret text(s) or file(s) ?  
 Add timestamps to the Console Output  
 Inspect build log for published Gradle build scans  
 Terminate a build if it's stuck  
 With Ant ?

**Build**

Execute shell ?

Command

See the list of available environment variables

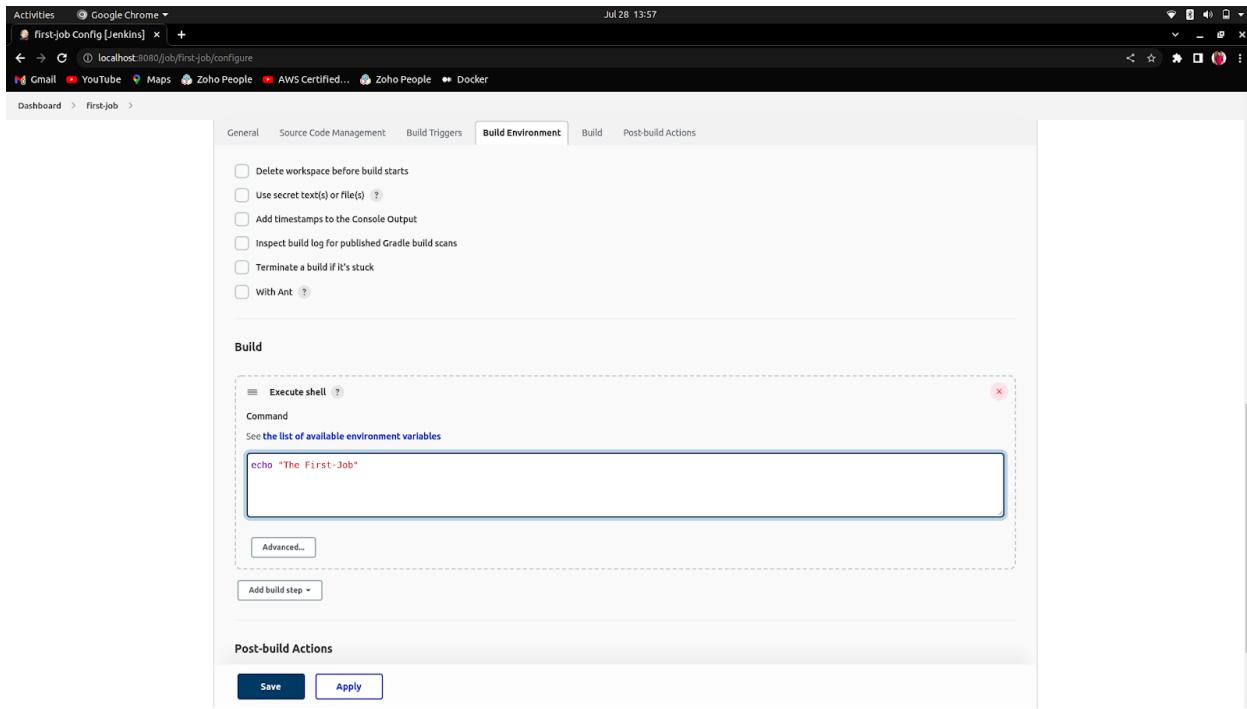
```
echo "The First-Job"
```

Advanced... Add build step ▾

**Post-build Actions**

Save Apply

Jul 28 13:57



This screenshot shows the Jenkins job configuration interface for a job named 'first-job'. The 'Build Environment' tab is selected. Under the 'Build' section, there is a single build step: 'Execute shell' with the command 'echo "The First-Job"'. The 'Post-build Actions' section is empty. At the bottom, there are 'Save' and 'Apply' buttons.

Activities Google Chrome ▾

first-job Config [Jenkins] + ↻ ➔ ⓘ localhost:8080/job/first-job/configure

Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > first-job >

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

✓ Saved

Delete workspace before build starts  
 Use secret text(s) or file(s) ?  
 Add timestamps to the Console Output  
 Inspect build log for published Gradle build scans  
 Terminate a build if it's stuck  
 With Ant ?

**Build**

Execute shell ?

Command

See the list of available environment variables

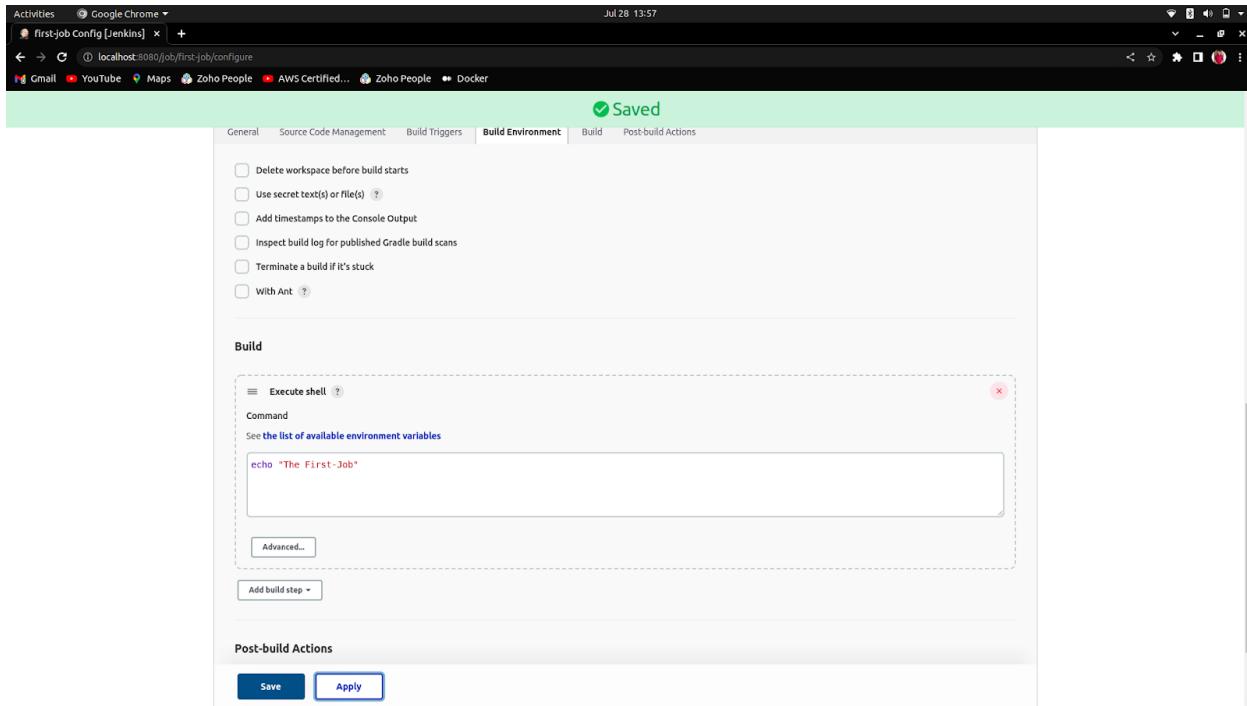
```
echo "The First-Job"
```

Advanced... Add build step ▾

**Post-build Actions**

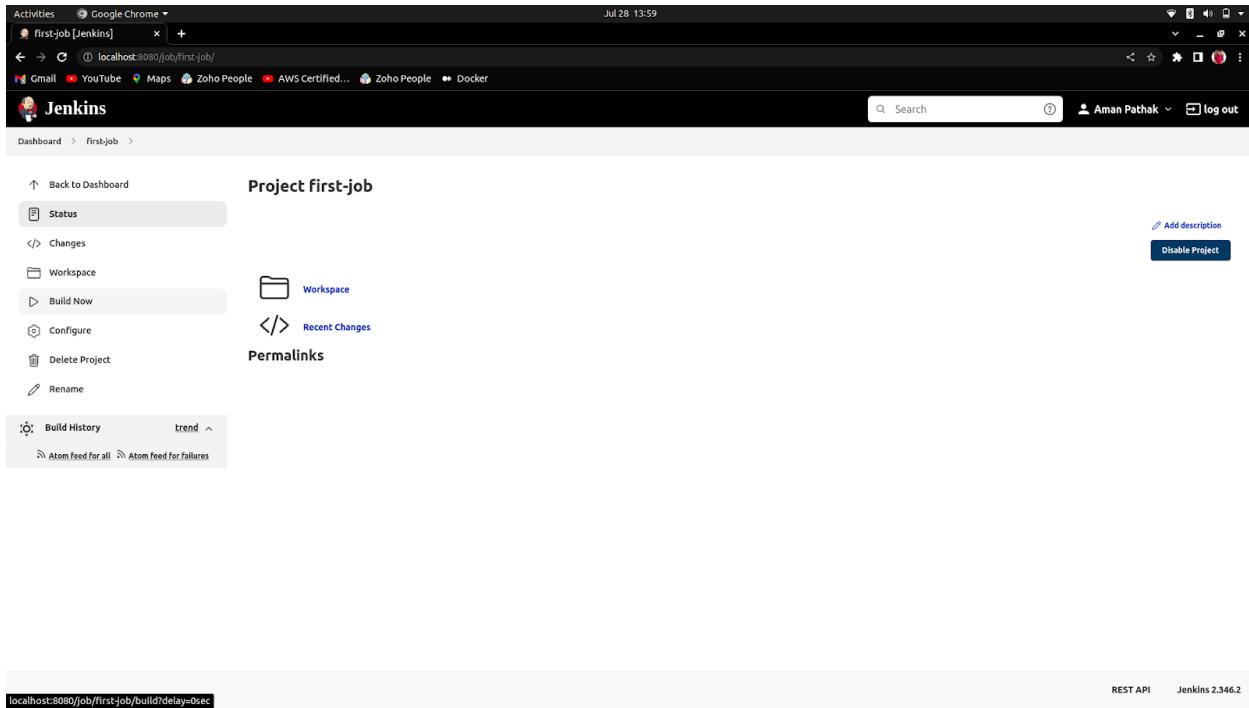
Save Apply

Jul 28 13:57

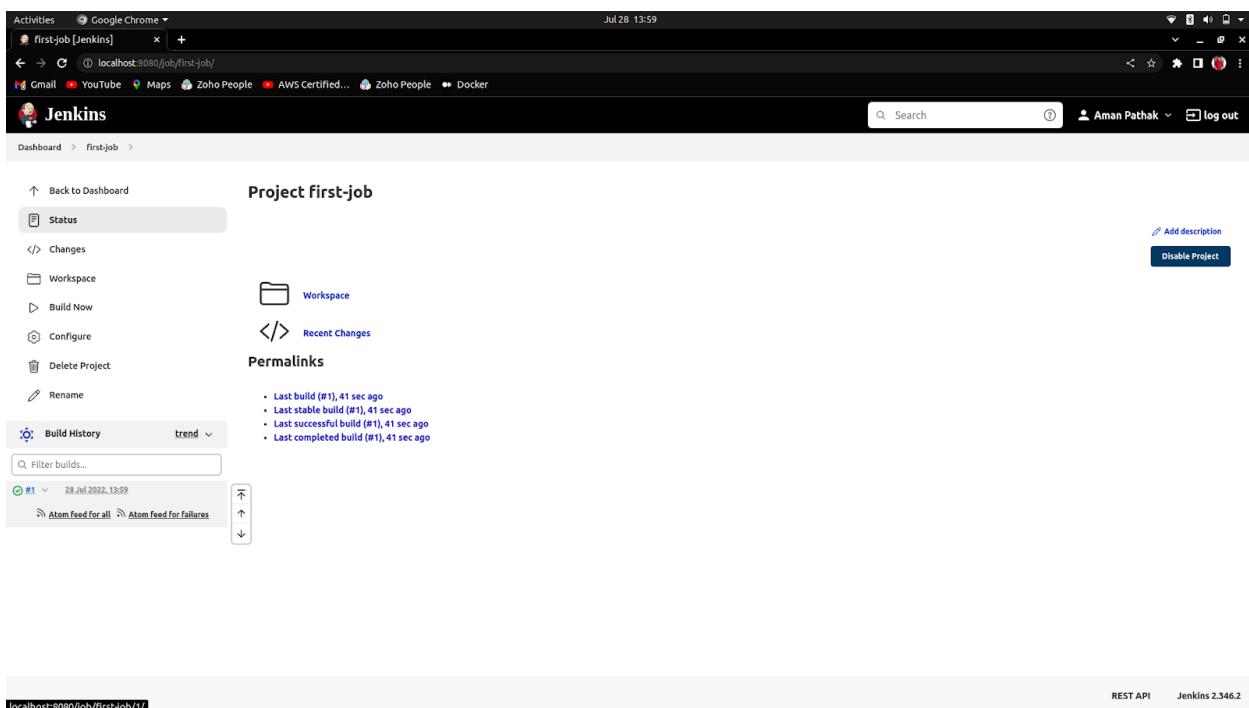


This screenshot shows the Jenkins job configuration interface for a job named 'first-job'. The 'Build Environment' tab is selected. Under the 'Build' section, there is a single build step: 'Execute shell' with the command 'echo "The First-Job"'. The 'Post-build Actions' section is empty. A green success message '✓ Saved' is displayed above the tabs. At the bottom, there are 'Save' and 'Apply' buttons.

## 4. Click on Build Now



## 5. As the build is successful. Now click on the green color text (#1).



## 6. Now, To see the output of your created job. Click on the Console Output.

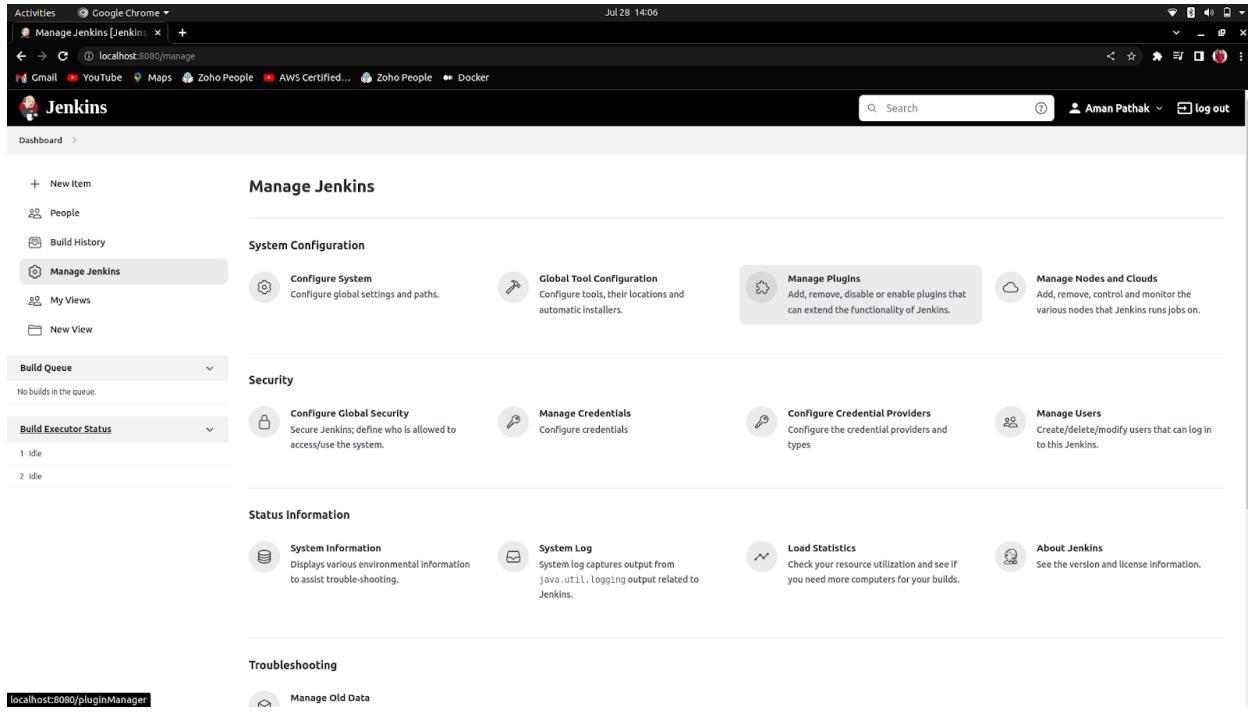
The screenshot shows a Jenkins job named 'first-job #1' with a green checkmark icon indicating it's successful. The build was started by user 'Aman Pathak' and took 0.1 seconds. The 'Console Output' tab is selected, showing the command 'echo The First-Job' and the output 'The First-Job'. The Jenkins version is 2.346.2.

## 7. The Output:

The screenshot shows the same Jenkins job and build details as the previous screenshot. The 'Console Output' tab is selected, displaying the command 'echo The First-Job' and the output 'The First-Job'. The Jenkins version is 2.346.2.

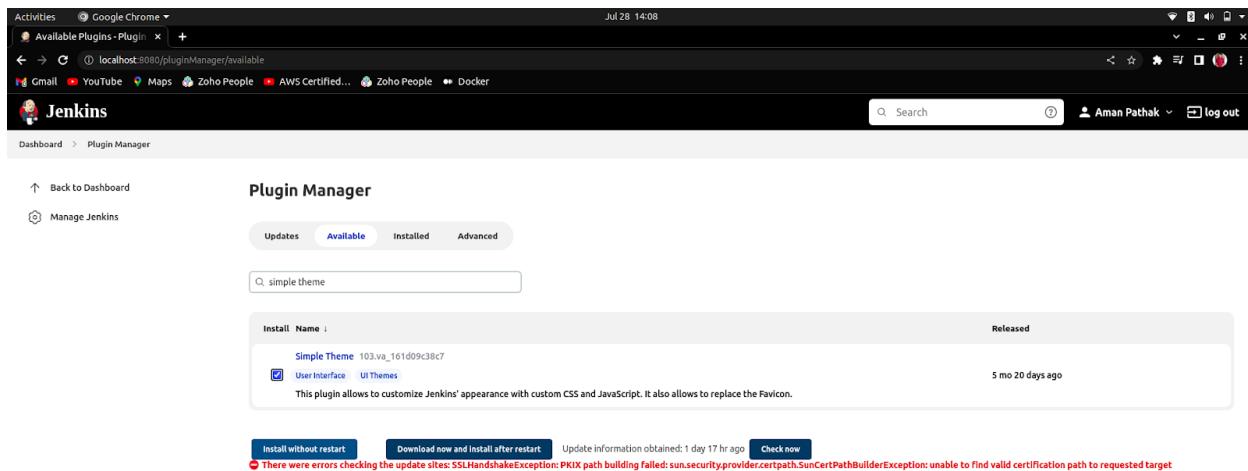
# Change Jenkins Theme using Plugin

## 1. Click on Manage Jenkins -> Manage Plugins.



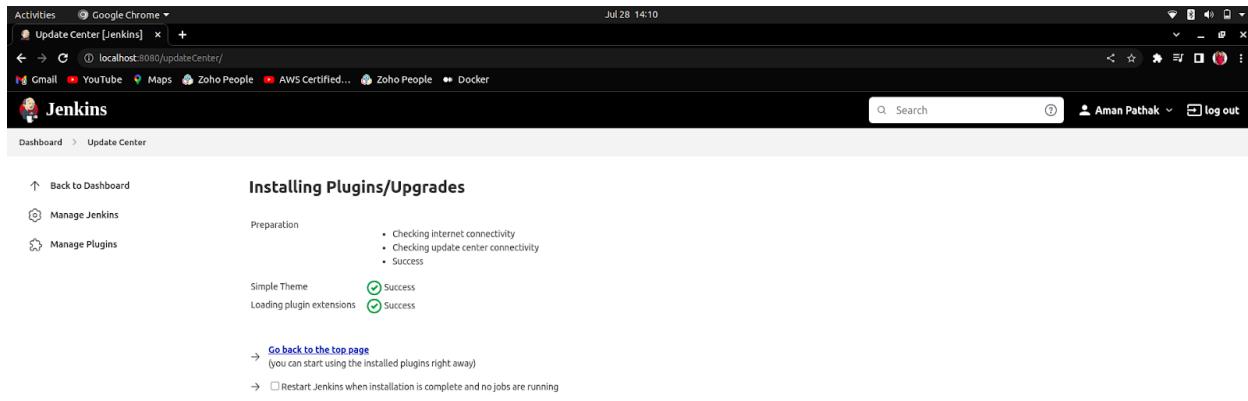
The screenshot shows the Jenkins 'Manage Jenkins' interface. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Manage Jenkins' (which is currently selected), 'My Views', and 'New View'. The main content area is titled 'Manage Jenkins' and contains several sections: 'System Configuration' (Configure System, Global Tool Configuration, Manage Plugins, Manage Nodes and Clouds), 'Security' (Configure Global Security, Manage Credentials, Configure Credential Providers, Manage Users), 'Status Information' (System Information, System Log, Load Statistics, About Jenkins), and 'Troubleshooting' (Manage Old Data). The 'Manage Plugins' link is highlighted with a red box.

## 2. Now search the Simple theme plugin, checked it and install without restart.



The screenshot shows the Jenkins 'Plugin Manager' interface. It has tabs for 'Updates', 'Available' (which is selected), 'Installed', and 'Advanced'. A search bar at the top has 'simple theme' typed into it. Below the search bar, a table lists the 'Simple Theme' plugin. The table includes columns for 'Name', 'Version', 'Type', 'Last Published', and 'Description'. The 'Simple Theme' entry shows version 103.v4\_161d09c38c7, type 'User Interface / UI Themes', was last published 5 months and 20 days ago, and the description states it allows customization of Jenkins' appearance. At the bottom of the table, there are buttons for 'Install without restart' (highlighted with a red box), 'Download now and install after restart', and 'Check now'. A note below the table says 'There were errors checking the update sites: SSLHandshakeException: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target'.

### 3. Now, that the plugin has been installed check the restart Jenkins server and log in again.

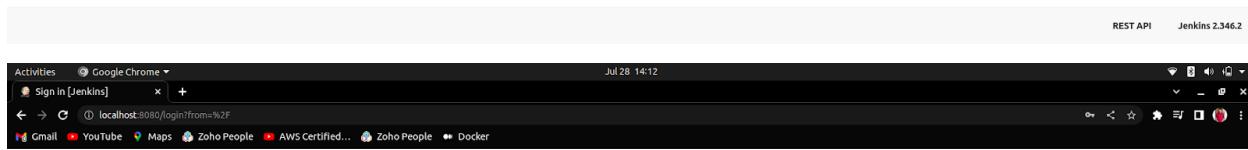


The screenshot shows the Jenkins Update Center page. At the top, there's a navigation bar with links for 'Activities', 'Google Chrome', 'Dashboard', and 'Update Center'. Below the navigation is a toolbar with links for 'Gmail', 'YouTube', 'Maps', 'Zoho People', 'AWS Certified...', 'Docker', and user information for 'Aman Pathak'.

The main content area is titled 'Installing Plugins/Upgrades'. It shows two sections: 'Preparation' and 'Simple Theme'. Under 'Preparation', there are three bullet points: 'Checking internet connectivity', 'Checking update center connectivity', and 'Success'. Under 'Simple Theme', there are two green circular icons with the word 'Success' next to them. Below these sections, there are three arrows pointing right:

- [Go back to the top page](#)  
(you can start using the installed plugins right away)
- Restart Jenkins when installation is complete and no jobs are running

At the bottom right of the page, it says 'REST API' and 'Jenkins 2.346.2'.



The screenshot shows the Jenkins login page. The URL in the address bar is 'localhost:8080/login?from=%2F'. The page features a cartoon Jenkins character holding a coffee cup. The text 'Welcome to Jenkins!' is displayed above a form. The form contains two input fields: one for 'username' with 'aman19' typed in, and another for 'password' with several dots. There is a checked checkbox labeled 'Keep me signed in' and a blue 'Sign in' button at the bottom.

## 4. Now, go to the Manage Jenkins again and configure the Plugin by clicking on Configure System.

The screenshot shows the Jenkins Manage Jenkins interface. In the top left, there's a sidebar with links like 'New Item', 'People', 'Build History', and 'Manage Jenkins'. The main area is titled 'Manage Jenkins' with a sub-section 'System Configuration'. It displays a message: 'It appears that your reverse proxy set up is broken.' Below this, there are several configuration items: 'Configure System' (Configure global settings and paths), 'Global Tool Configuration' (Configure tools, their locations and automatic installers), 'Manage Plugins' (Add, remove, disable or enable plugins that can extend the functionality of Jenkins), and 'Manage Nodes and Clouds' (Add, remove, control and monitor the various nodes that Jenkins runs jobs on). At the bottom, there's a 'Status Information' section with links for 'System Information', 'System Log', 'Load Statistics', and 'About Jenkins'.

## 5. Now, enter the URL of CSS to change the theme and apply then, then save it. URL Link-> [Themes Link](#)

The screenshot shows the Jenkins Configure System interface. In the top left, there's a sidebar with links like 'Dashboard' and 'Configure System'. The main area is titled 'Configure System'. Under the 'Theme' section, there's a 'Theme elements' section with a 'CSS URL' input field containing the URL 'https://cdn.rawgit.com/afonsof/jenkins-material-theme/gh-pages/dist/material-blue.css'. Below this, there's a 'Folder' section for 'Health Metrics' and a 'Global Build Time Out' section. At the bottom, there are 'Save' and 'Apply' buttons.

# To change the Jenkins URL

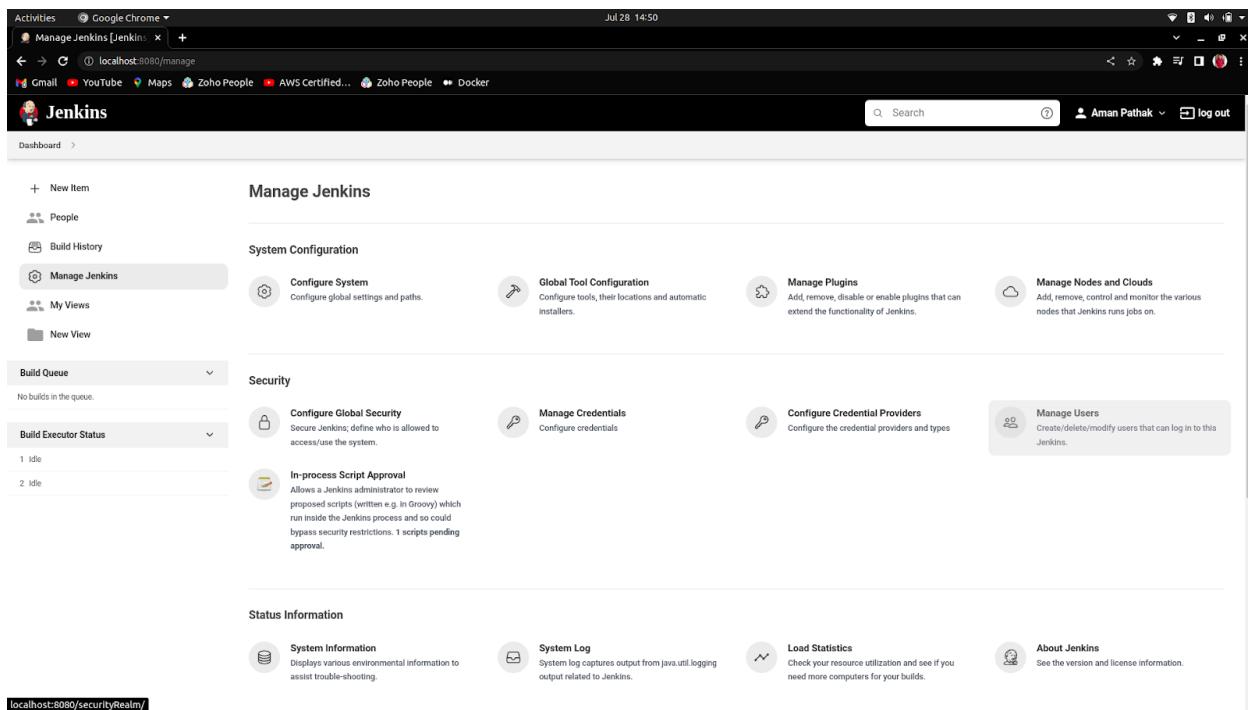
Manage Jenkins -> Configure System

Enter the desired location or URL.

By Default URL-> <http://localhost:8080/>

## +Create a new user in Jenkins

### 1. Go to Manage Plugins -> Manage Users.



The screenshot shows the Jenkins Manage Jenkins interface. On the left, there's a sidebar with options like 'New Item', 'People', 'Build History', 'Manage Jenkins' (which is highlighted), 'My Views', and 'New View'. Below that are sections for 'Build Queue' (no builds in the queue) and 'Build Executor Status' (1 idle, 2 idle). The main content area has several tabs: 'System Configuration' (Configure System, Global Tool Configuration, Manage Plugins, Manage Nodes and Clouds), 'Security' (Configure Global Security, Manage Credentials, Configure Credential Providers, Manage Users), and 'Status Information' (System Information, System Log, Load Statistics, About Jenkins). At the bottom of the page, the URL 'localhost:8080/securityRealm/' is visible.

## 2. Click on Create user. And enter the further details.

The screenshot shows the Jenkins 'Create User' form. The 'Create User' button is highlighted with a grey background. The form fields are as follows:

- Username: aman20
- Password: ..... (redacted)
- Confirm password: ..... (redacted)
- Full name: Jon snow
- E-mail address: jon@snow.com

At the bottom right of the form is a blue 'CREATE USER' button.

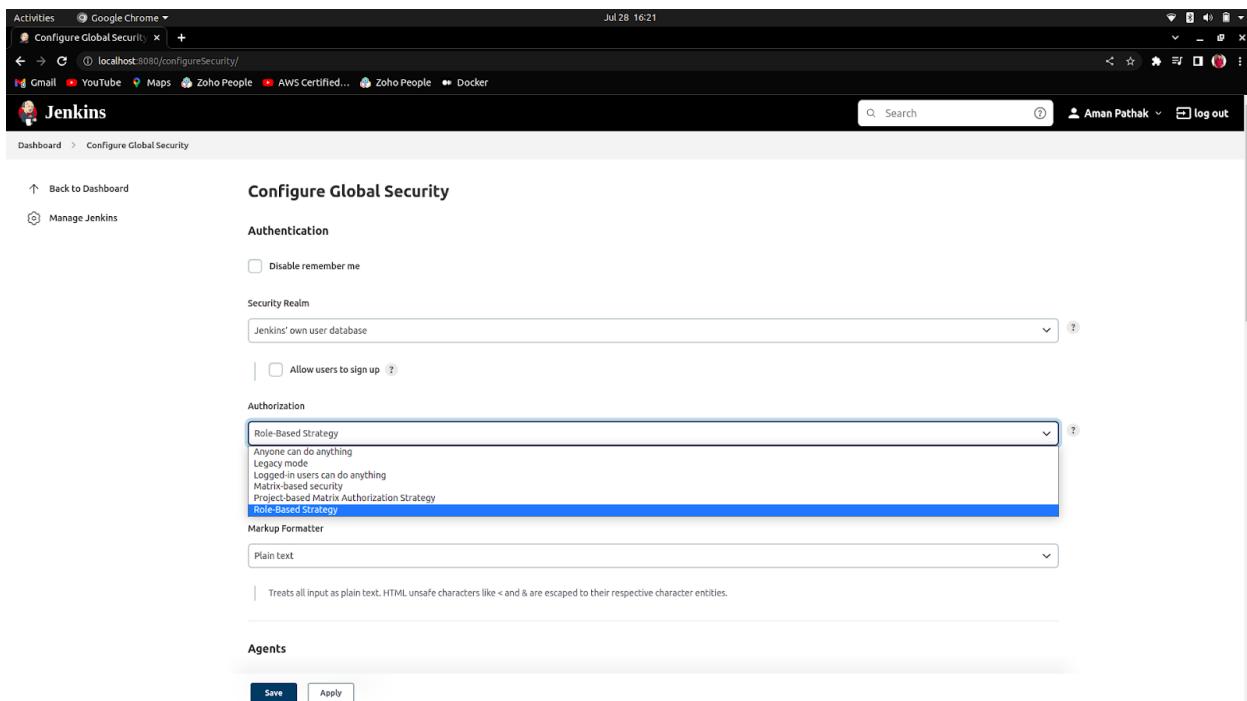
## 3. New User

The screenshot shows the Jenkins 'Users' page. The table displays the following user information:

User ID	Name	Action
aman19	Aman Pathak	
aman20	Jon snow	

# Jenkins Role-Based Access Control (RBAC)

1. Install the plugin named Role-based Authorization Strategy by going into the Manage-Jenkins -> Manage Plugins - Available and then search the plugin name which is written above.
2. Now, you have to configure this plugin by going into Manage Plugin -> Configure Global Security selecting the Role-based Strategy, and applying and saving the configuration.



### 3. Now, the Manage and Assign roles option will appear under the security section in the Manage Jenkins.

The screenshot shows the Jenkins Manage Jenkins dashboard. On the left sidebar, under 'Manage Jenkins', there is a 'Manage and Assign Roles' link. The main content area is titled 'Manage Jenkins' and includes sections for 'System Configuration' (Configure System, Global Tool Configuration, Manage Plugins, Manage Nodes and Clouds), 'Security' (Configure Global Security, Manage Credentials, Configure Credential Providers, In-process Script Approval, Manage Users), and 'Status Information' (System Information, System Log, Load Statistics, About Jenkins). The 'Manage and Assign Roles' link is highlighted.

### 4. Now, go into the section and click on Manage Roles. Then, create the role named <developer> which has no access then apply and save.

The screenshot shows the Jenkins Manage Roles page. Under 'Global roles', there is a table where a new role named 'developer' is being created. The 'developer' role has no checked permissions across all categories (Overall, Credentials, Agent, Job, Run, View, SCM). Below the table, there is a 'Role to add' input field containing 'developer' and an 'Add' button. At the bottom, there is a 'Save' button and an 'Apply' button.

## 5. Now, when you log in with another user e.g. Jon Snow. You'll get this.

A screenshot of a Google Chrome browser window displaying a Jenkins dashboard. The title bar shows 'Jenkins' and the address bar shows 'localhost:8080'. The main content area displays an 'Access Denied' message with the text 'aman20 is missing the Overall/Read permission' in red. At the bottom right of the page, there are links for 'REST API' and 'Jenkins 2.346.2'.

## 6. Now, go to the Assign Roles and add the user then, give the developer role to that user. Click on Apply and save.

A screenshot of a Google Chrome browser window displaying the 'Manage Roles' page in Jenkins. The title bar shows 'Manage Roles [Jenkins]' and the address bar shows 'localhost:8080/role-strategy/manage-roles'. The main content area shows a 'Manage Roles' grid where the 'developer' role is being assigned to a user. The 'Role to add' field contains 'aman20'. Below the grid, there is an 'Item roles' section and buttons for 'Save' and 'Apply'.

The screenshot shows the Jenkins 'Assign Roles' configuration page. The left sidebar includes links for New Item, People, Build History, Manage Jenkins, My Views, and New View. Under 'Build Queue' and 'Build Executor Status', both show 1 Idle. The main content area is titled 'Assign Roles' and contains two sections: 'Global roles' and 'Item roles'. In 'Global roles', under 'User/group', 'Aman Pathak' has checkboxes for 'Read' (checked), 'Write' (unchecked), and 'Administer' (unchecked). 'Jon snow' has checkboxes for 'Read' (unchecked), 'Write' (checked), and 'Administer' (unchecked). 'Anonymous' has all three checkboxes unchecked. Below this is a search bar for 'User/group to add' and an 'Add' button. In 'Item roles', there is a similar structure for 'User/group' and 'Anonymous', with a search bar and 'Save' and 'Apply' buttons.

**7. If you give access by changing the roles and assigning the roles to read and build jobs to the other user, It will look like this.**

The screenshot shows the Jenkins dashboard after role assignment. The top navigation bar shows 'Dashboard [Jenkins]' and the URL 'localhost:8080'. The right side shows the user 'Jon snow' is logged in. The left sidebar includes links for People, Build History, My Views, Build Queue (No builds in the queue), and Build Executor Status (1 Idle). The bottom right corner displays 'REST API' and 'Jenkins 2.346.2'.

## 8. If you add one more permission which is job read then, the other can see the jobs.

**Manage Roles [Jenkins]**

localhost:8080/role-strategy/manage-roles

Jul 28 16:42

Search Aman Pathak log out

Dashboard > Manage and Assign Roles

**Manage Roles**

**Global roles**

Role	Overall	Credentials	Agent	Job	Run	View	SCM
Administrator	<input checked="" type="checkbox"/>						
Developer	<input type="checkbox"/>						
admin	<input checked="" type="checkbox"/>						
developer	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Role to add: **Permission: Job/Read Role: developer**

Add

---

**Item roles**

Role Pattern	Credentials	Job	Run	View	SCM
Administrator	<input type="checkbox"/>				
Developer	<input type="checkbox"/>				
user@domain	<input type="checkbox"/>				
Create	<input type="checkbox"/>				

Save Apply

**Dashboard [Jenkins]**

localhost:8080

Jul 28 16:43

Search Jon snow log out

Dashboard >

All

Build Queue

No builds in the queue.

Build Executor Status

1 Idle 2 Idle

S	W	Name	Last Success	Last Failure	Last Duration
		demo-first	1 day 2 hr #1	N/A	92 ms
		demo-second	1 day 2 hr #17	1 day 2 hr #6	50 ms
		First-job	2 hr 43 min #1	N/A	0.1 sec

Icon: S M L

Icon legend Atom feed for all Atom feed for failures Atom feed for just latest builds

REST API Jenkins 2.346.2

# GitHub Job without GitHub Plugin

## 1. Create a Job with freestyle projects.

The screenshot shows the Jenkins 'New Item' creation interface in Google Chrome. The title bar says 'Activities Google Chrome'. The address bar shows 'localhost:4080/view/all/newJob'. The main content area has a search bar with 'demo-none-github' and a note '» Required field'. Below it, under 'Freestyle project', there is a description: '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.' There are five options listed:

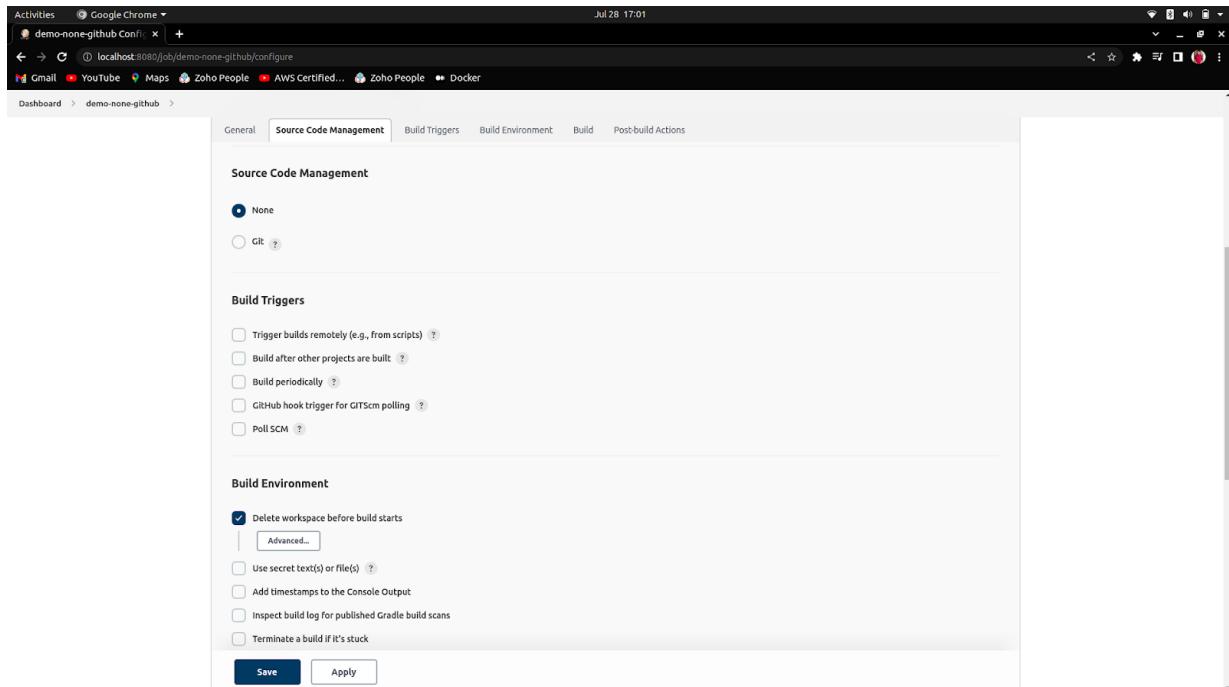
- Pipeline**: Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Multi-configuration project**: Suitable for projects that need a 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.
- Organization Folder**: Creates a set of multibranch project subfolders by scanning for repositories.

If you want to create a new item from other existing, you can use this option:

**Copy from**: A dropdown menu with the placeholder 'Type to autocomplete'.

A green button labeled 'OK' is at the bottom of the dialog.

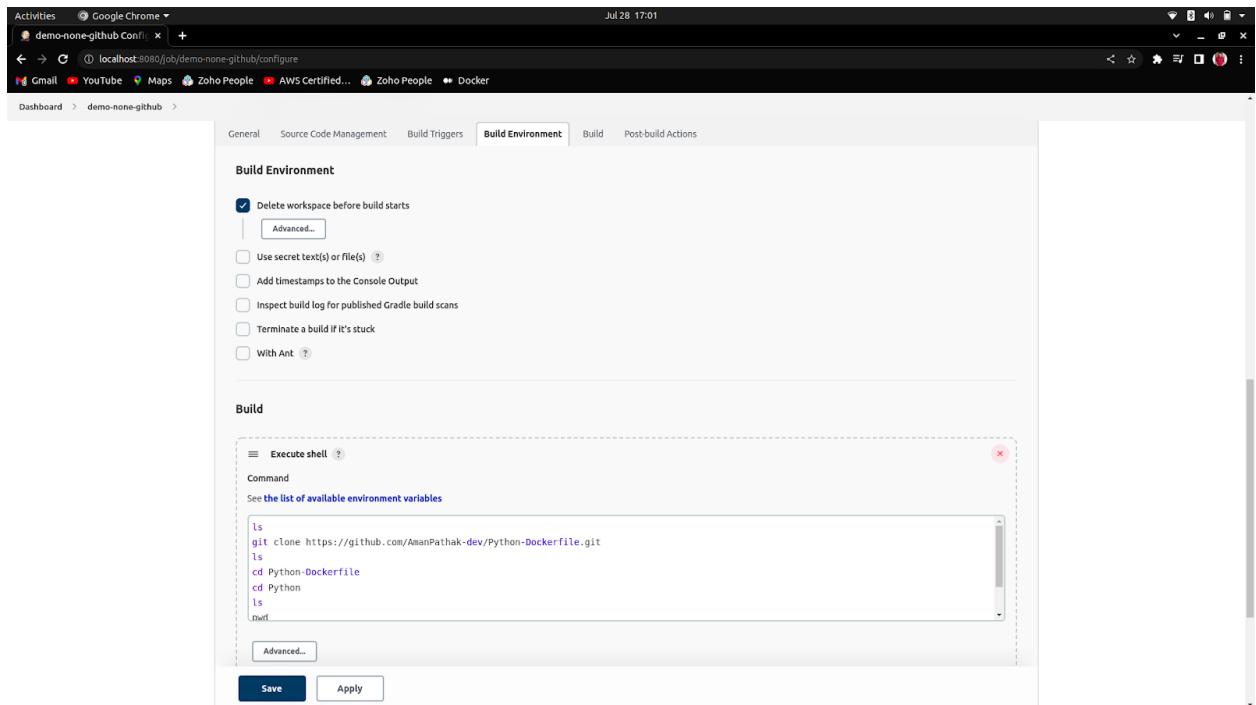
## 2. Take Source code Management as None(for not github).



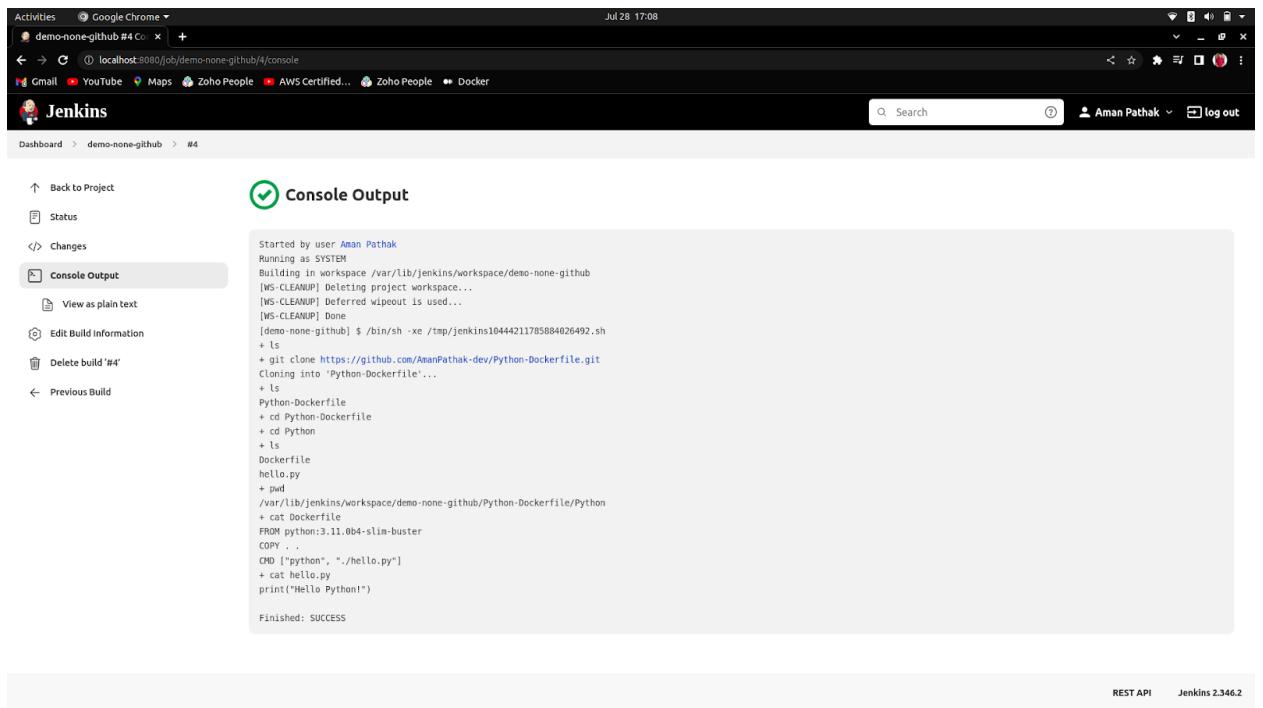
The screenshot shows a web-based build configuration interface. At the top, there's a navigation bar with tabs: General, Source Code Management (which is selected), Build Triggers, Build Environment, Build, and Post-build Actions. Below the tabs, under 'Source Code Management', the 'None' option is selected (indicated by a blue circle). There's also a 'Git' option with a grey circle. The 'Build Triggers' section contains several checkboxes: 'Trigger builds remotely (e.g., from scripts)', 'Build after other projects are built', 'Build periodically', 'GitHub hook trigger for GITScm polling', and 'Poll SCM'. Under 'Build Environment', there are several checkboxes: 'Delete workspace before build starts' (which is checked), 'Advanced...', 'Use secret text(s) or file(s)', 'Add timestamps to the Console Output', 'Inspect build log for published Gradle build scans', and 'Terminate a build if it's stuck'. At the bottom of the configuration panel are two buttons: 'Save' and 'Apply'.

**3. Build Environment -> Check the  Delete workspace before build starts.** It is checked because when you are going to build the job again it couldn't succeed with the <git clone> because the file will already exist. That's why we checked that.

**Then, write your bash/shell script.**



## 4. The Output



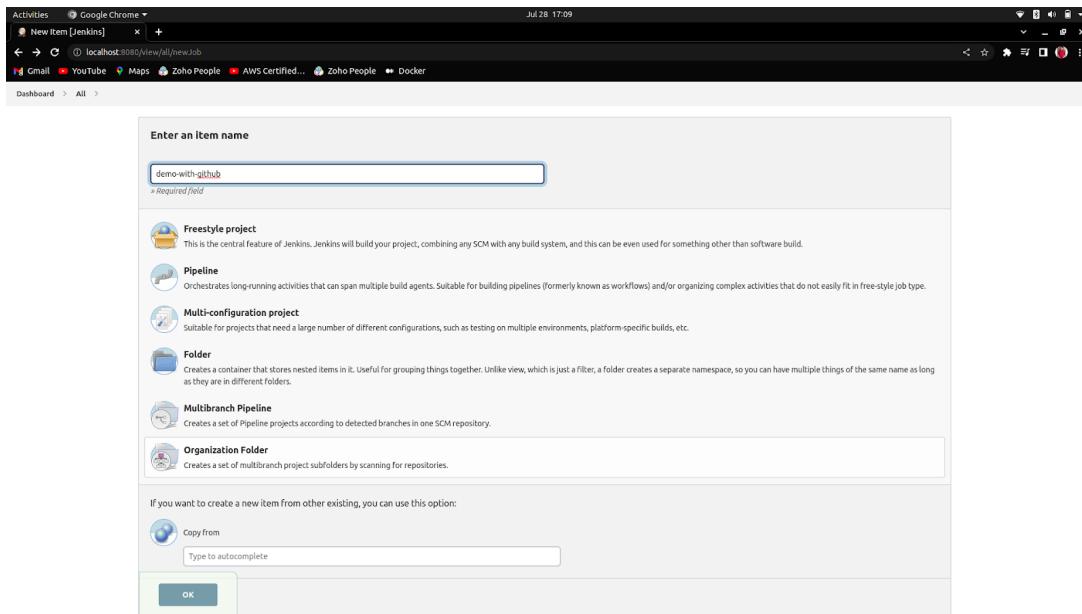
A screenshot of a Google Chrome browser window showing the Jenkins console output for build #4 of the 'demo-none-github' project. The title bar shows 'Activities Google Chrome' and the address bar shows 'localhost:8080/job/demo-none-github/4/console'. The Jenkins logo is at the top left, and the user 'Aman Pathak' is logged in. The main content area is titled 'Console Output' with a checkmark icon. It displays the command-line logs of the build process, which includes cloning a GitHub repository, creating a Dockerfile, building a Docker image, and running a Python script. The build status is 'SUCCESS'.

```
Started by user Aman Pathak
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/demo-none-github
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
[WS-CLEANUP] Done
[demo-none-github] $ /bin/sh -xe /tmp/jenkins10444211785884026492.sh
+ ls
+ git clone https://github.com/AmanPathak-dev/Python-Dockerfile.git
Cloning into 'Python-Dockerfile'...
+ ls
Python-Dockerfile
+ cd Python-Dockerfile
+ cd Python
+ ls
Dockerfile
hello.py
+ pwd
/var/lib/jenkins/workspace/demo-none-github/Python-Dockerfile/Python
+ cat Dockerfile
FROM python:3.11.0b4-slim-buster
COPY .
CMD ["python", "./hello.py"]
+ cat hello.py
print("Hello Python!")

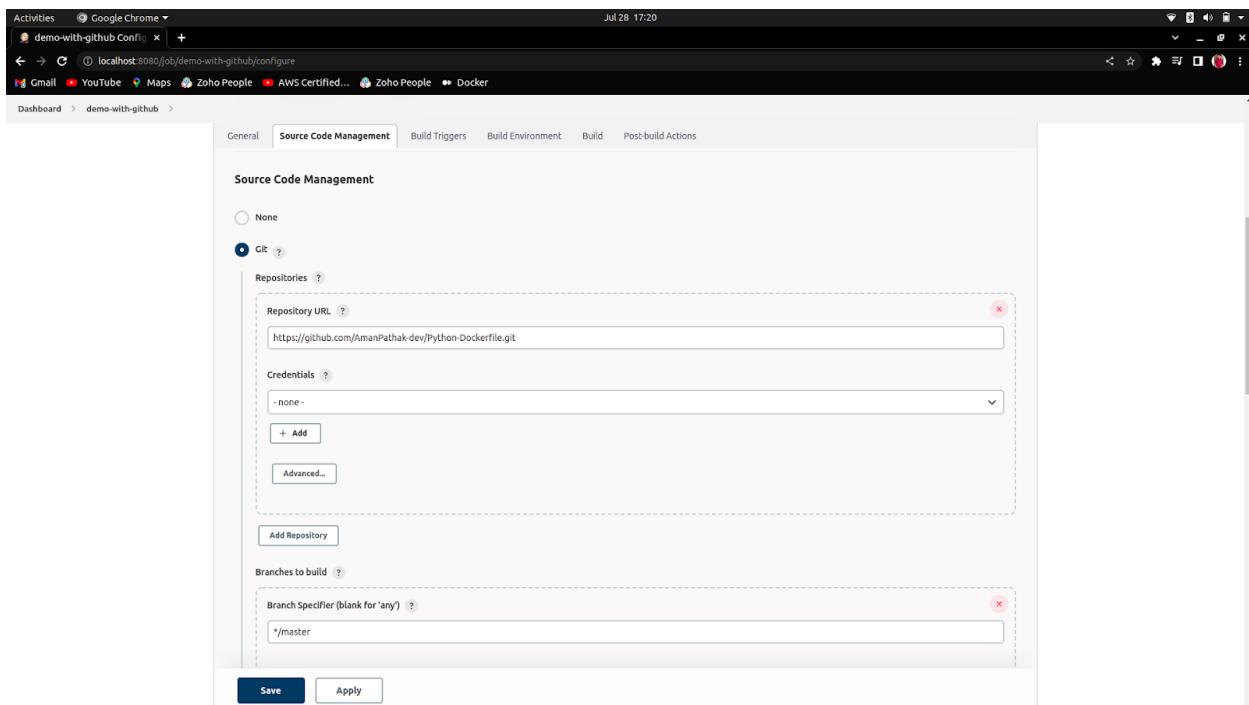
Finished: SUCCESS
```

# GitHub Job with GitHub Plugin

## 1. Create a freestyle project.



## 2. Source code Management must be GitHub. And add the further details as written below.



The screenshot shows the Jenkins 'Build Environment' configuration page for a job named 'demo-with-github'. The 'Build' section contains a single step: 'Execute shell' with the command:

```
ls
cd Python
ls
cat Dockerfile
```

At the bottom are 'Save' and 'Apply' buttons.

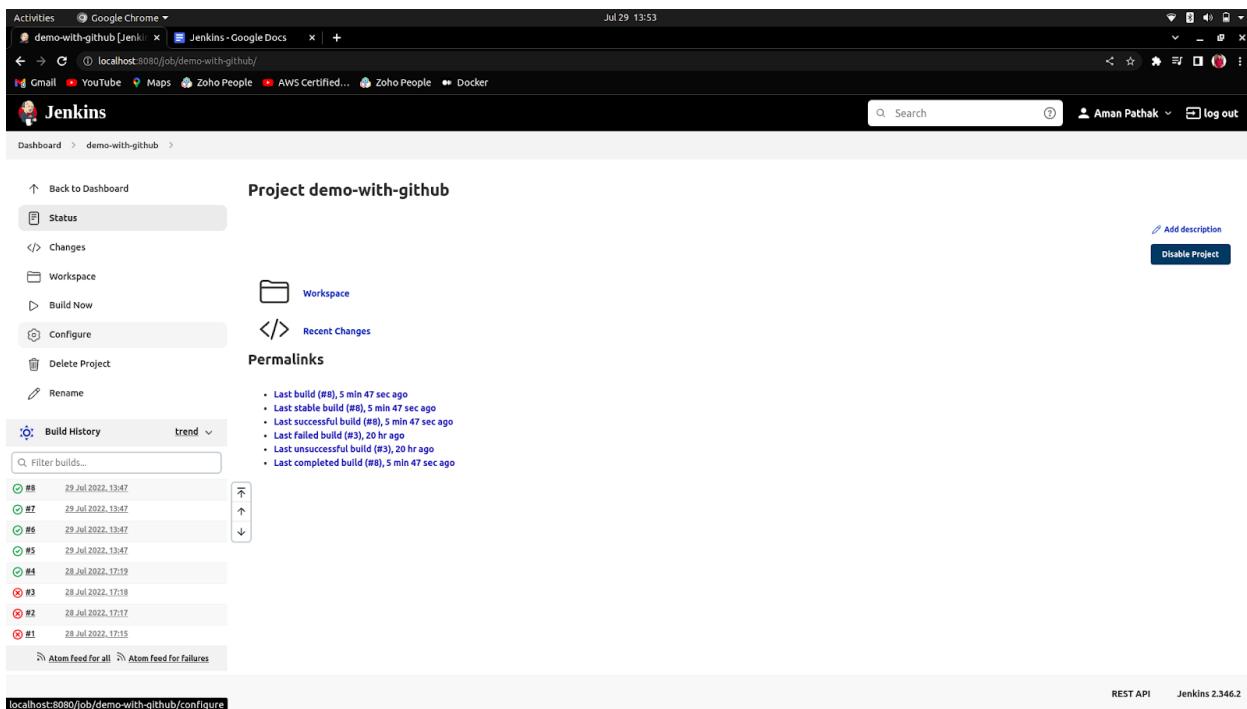
## The Output

The screenshot shows the Jenkins 'Console Output' page for a build labeled '#4'. The log output is as follows:

```
Started by user Aman Pathak
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/demo-with-github
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
[WS-CLEANUP] Done
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
  Cloning repository https://github.com/AmanPathak-dev/Python-Dockerfile.git
    > git init /var/lib/jenkins/workspace/demo-with-github # timeout=10
   Fetching upstream changes from https://github.com/AmanPathak-dev/Python-Dockerfile.git
    > git --version # timeout=10
    > git -version # 'git' version 2.25.1'
    > git fetch --tags --progress -- https://github.com/AmanPathak-dev/Python-Dockerfile.git +refs/heads/*:refs/remotes/origin/* # timeout=10
    > git config remote.origin.url https://github.com/AmanPathak-dev/Python-Dockerfile.git # timeout=10
    > git config -add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
  Avoid second fetch
    > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
  Checking out Revision 1a8cc24455ea454b1c6b94b564eecc528f29fbda8 (refs/remotes/origin/master)
    > git config core.sparsecheckout # timeout=10
    > git checkout -f 1a8cc24455ea454b1c6b94b564eecc528f29fbda8 # timeout=10
Commit message: "Delete DockerJava directory"
    > git rev-list --no-walk 1a8cc24455ea454b1c6b94b564eecc528f29fbda8 # timeout=10
[demo-with-github] $ /bin/sh -xe /tmp/jenkins3686238978885875408.sh
+ ls
Python
+ cd Python
+ ls
Dockerfile
hello.py
+ cat Dockerfile
FROM python:3.11.0b4-slim-buster
COPY . .
CMD ["python", "./hello.py"]
+ pwd
/var/lib/jenkins/workspace/demo-with-github/Python
Finished: SUCCESS
```

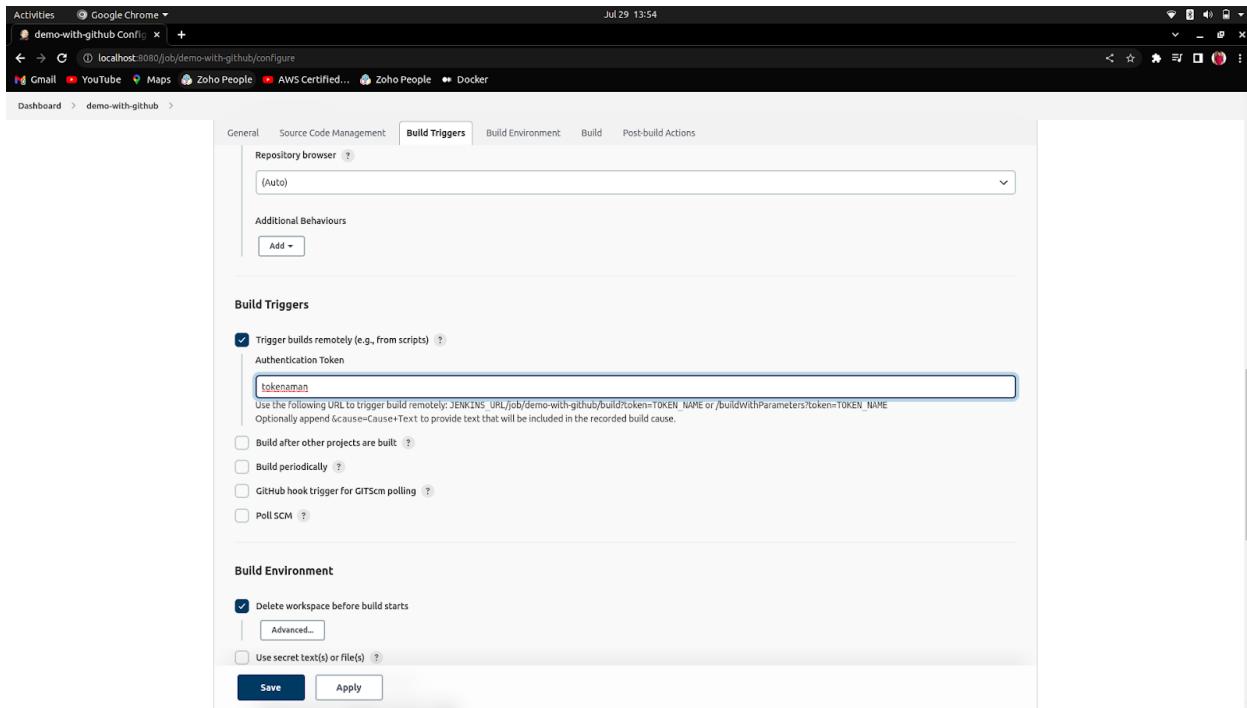
# How to build a job using Trigger Builds remotely (Authentication token)

## 1. Go to any Job.



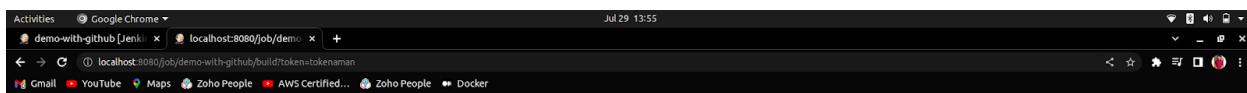
The screenshot shows a Jenkins project page for 'demo-with-github'. The top navigation bar includes links for 'Activities', 'Google Chrome', 'localhost:8080/job/demo-with-github/' (the current tab), 'Jenkins - Google Docs', and '+'. The user 'Aman Pathak' is logged in. The main content area displays the 'Project demo-with-github' page with tabs for 'Status', 'Changes', 'Workspace', 'Build Now', 'Configure', 'Delete Project', 'Rename', and 'Build History'. The 'Build History' tab is selected, showing a list of builds from July 29, 2022, to July 28, 2022. The builds are color-coded: green for successful (#8, #7, #6, #5, #4, #3, #2) and red for failed (#1). Each build entry includes a timestamp. To the right of the build history, there are sections for 'Recent Changes' and 'Permalinks'. At the bottom, there are links for 'Atom Feed for all' and 'Atom Feed for failures', and the URL 'localhost:8080/job/demo-with-github/configure'.

## 2. Enter the token name inside the Trigger build remotely section.



The screenshot shows the Jenkins job configuration interface for a job named "demo-with-github". The "Build Triggers" tab is selected. Under "Build Triggers", the "Trigger builds remotely (e.g., from scripts)" checkbox is checked, and the "Authentication Token" field contains the value "tokenname". Other trigger options like "Build after other projects are built", "Build periodically", "GitHub hook trigger for GITScm polling", and "Poll SCM" are unchecked. Below the triggers, the "Build Environment" section has the "Delete workspace before build starts" checkbox checked and the "Advanced..." button visible. The "Save" and "Apply" buttons are at the bottom right. The browser title bar shows "localhost:8080/job/demo-with-github/configure".

## 3. Now, cop the URL that was under the token name field and paste in the new tab



The screenshot shows a browser tab titled "localhost:8080/job/demo-with-github/build?token=tokenname". The URL is displayed in the address bar. The browser title bar shows "localhost:8080/job/demo-with-github/configure". The page content is mostly blank, indicating the build has not yet started or completed.

## 4. Now, come to the Jenkins dashboard. You will see the build has been automatically started.

The screenshot shows the Jenkins dashboard for the 'demo-with-github' project. The build history table lists the following builds:

Build	Timestamp
#10	29 Jul 2022, 13:55
#9	29 Jul 2022, 13:47
#8	29 Jul 2022, 13:47
#7	29 Jul 2022, 13:47
#6	29 Jul 2022, 13:47
#5	29 Jul 2022, 13:47
#4	28 Jul 2022, 17:19
#3	28 Jul 2022, 17:18
#2	28 Jul 2022, 17:17
#1	28 Jul 2022, 17:15

## Build Successful

The screenshot shows the Jenkins dashboard for the 'demo-with-github' project. The build history table now shows all builds as successful:

Build	Timestamp
#10	29 Jul 2022, 13:55
#9	29 Jul 2022, 13:47
#8	29 Jul 2022, 13:47
#7	29 Jul 2022, 13:47
#6	29 Jul 2022, 13:47
#5	29 Jul 2022, 13:47
#4	28 Jul 2022, 17:19
#3	28 Jul 2022, 17:18
#2	28 Jul 2022, 17:17
#1	28 Jul 2022, 17:15

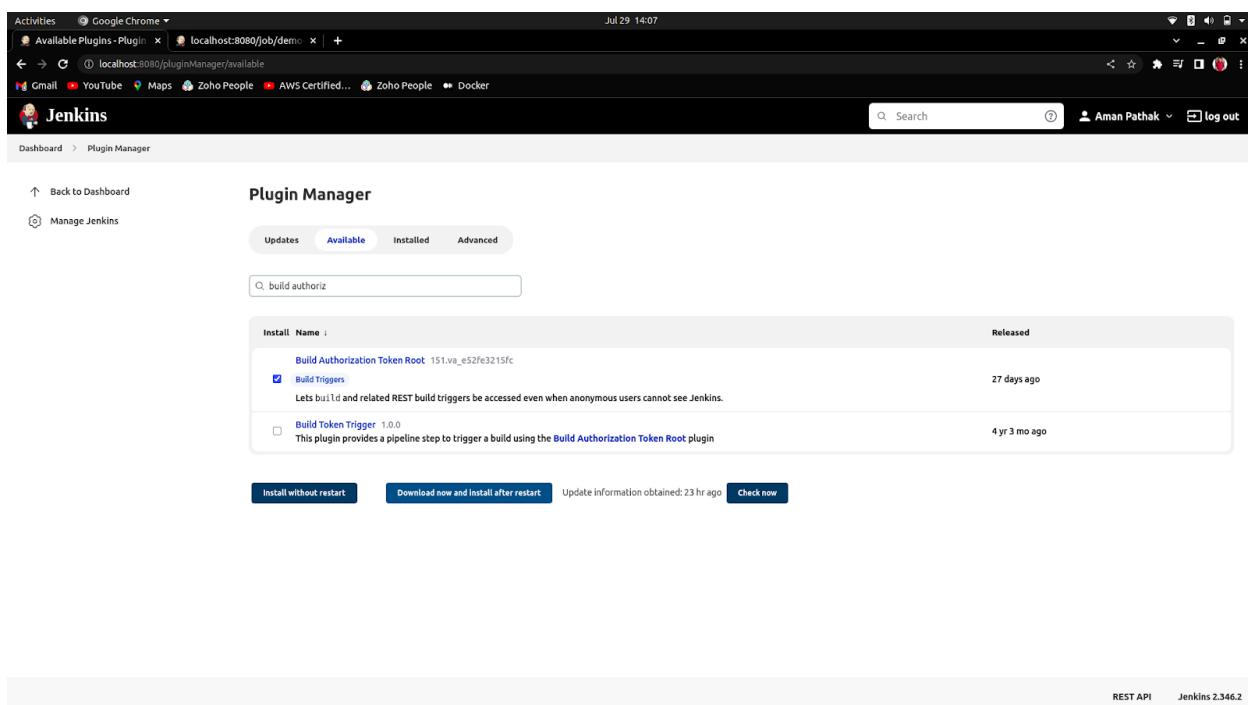
# Build Triggers

## 1. Build Job through Incognito Mode and Terminal

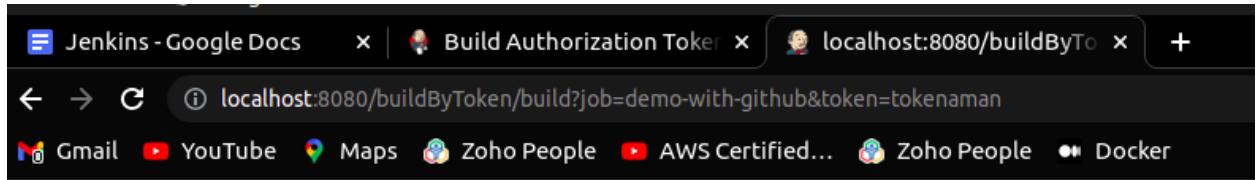
### Why?

As I am building a job through just a URL. But when I put the same URL in incognito mode of any browser it asked me to log in again, to resolve that issue. Follow the steps below:

1. First of all, I need to install a plugin named *Build Authorization Token Root* as [Build Authorization Token Link Official Plugin](#)



## 2. Enter the URL which is written in the tab.



## 3. The result you can see.

A screenshot of the Jenkins dashboard. The left sidebar includes links for Activities, Google Chrome, Dashboard [Jenkins], localhost:8080/job/demo, People, Build History, Manage Jenkins, My Views, New View, and Build Queue (which is empty). The main content area shows a table of build history with columns for Status (S), Workstation (W), Name, Last Success, Last Failure, and Last Duration. The table contains six rows for jobs: demo-with-github, demo-none-github, first-job, demo-first, demo-for-other-user, and demo-second. Below the build history is a section titled 'Build Executor Status' showing one active executor named 'demo-with-github' with build #11, and another entry for 'Idle'. At the bottom right, there are links for REST API and Jenkins 2.346.2.

# The build has been successful.

A screenshot of a Jenkins project page titled "Project demo-with-github". The page shows a successful build (#11) from 29.Jul.2022 at 14:11. The build history table includes the following entries:

Build	Date
#11	29.Jul.2022, 14:11
#10	29.Jul.2022, 13:55
#9	29.Jul.2022, 13:55
#8	29.Jul.2022, 13:47
#7	29.Jul.2022, 13:47
#6	29.Jul.2022, 13:47
#5	29.Jul.2022, 13:47
#4	28.Jul.2022, 17:19
#3	28.Jul.2022, 17:18
#2	28.Jul.2022, 17:17

# Through Terminal

1. Enter the same URL and make one change before the & symbol. Use backslash (\).

```
amanpathak@aman-pathak:~$ curl http://localhost:8080/buildByToken/build?job=demo-with-github\&token=tokenaman  
amanpathak@aman-pathak:~$
```

2. The build has been successful.

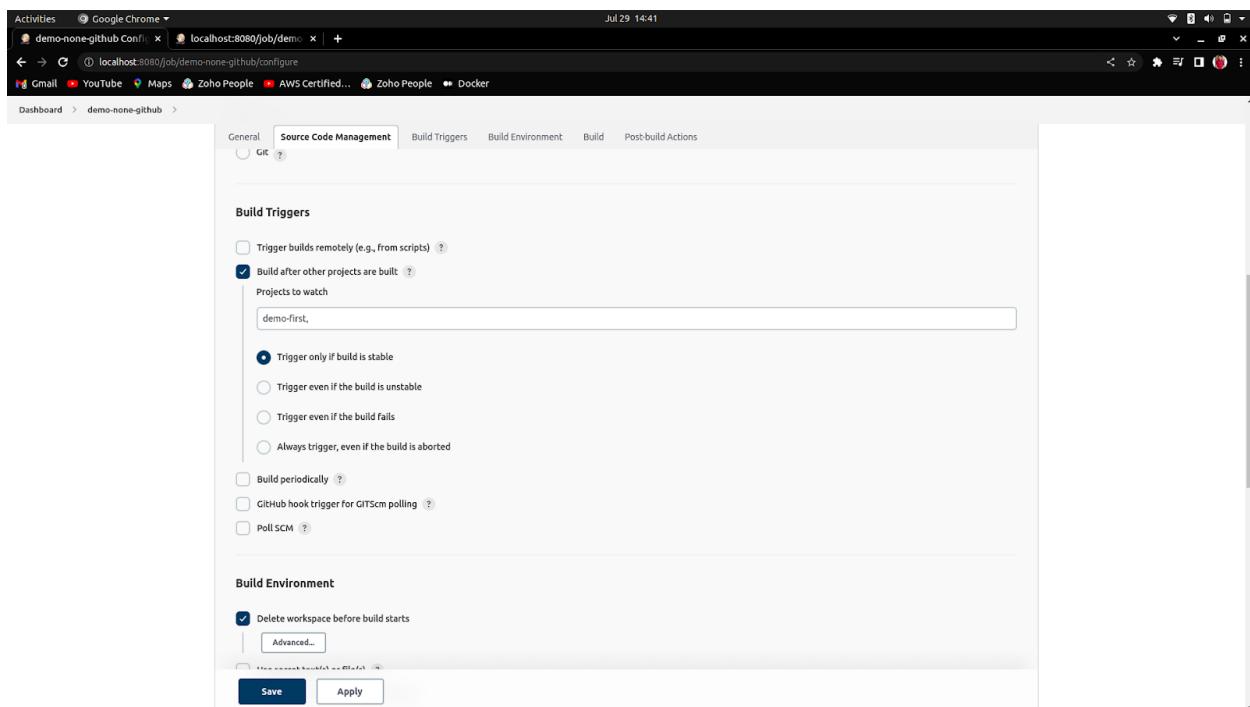
The screenshot shows a Jenkins project page for 'demo-with-github'. The top navigation bar includes links for 'Activities', 'Google Chrome', and the Jenkins logo. The main content area displays the project status as 'Status' (green), with options to 'Changes', 'Workspace', 'Build Now', 'Configure', 'Delete Project', and 'Rename'. A 'Recent Changes' section lists the last five builds. The 'Build History' section shows a list of builds from #12 down to #4, all of which are green (successful). The 'Permalinks' section provides direct links to each build's details. On the right side, there are buttons for 'Add description' and 'Disable Project'. The user 'Aman Pathak' is logged in, as shown in the top right corner.

## 2. Build after other projects are built

### Why?

When you have to build your specific project after some desired projects. You can use this Build Trigger.

1. Enter the other project that you want to build first.



2. Now, run the project that you entered under the Projects to Watch section.

S	W	Name	Last Success	Last Failure	Last Duration :
✓	*	demo-none-github	4 min 18 sec #11	21 hr #3	1 sec
✓	*	demo-with-github	10 min #15	21 hr #3	0.95 sec
✓	*	demo-for-other-user	21 hr #2	N/A	62 ms
✓	*	demo-second	2 days 0 hr #17	2 days 0 hr #6	50 ms
✓	*	first-job	11 min #2	N/A	50 ms
✓	*	demo-first	4 min 26 sec #2	N/A	37 ms

**Build Executor Status:**

- Icon: S M L
- 1 Idle
- 2 Idle

**Build History:**

- </> Changes
- Workspace
- Build Now
- Configure
- Delete Project
- Rename

REST API Jenkins 2.346.2

3. You will see on the left, the project is built.

**Build Queue (1)**

demo-with-github

S	W	Name	Last Success	Last Failure	Last Duration :
✓	*	demo-none-github	4 min 18 sec #11	21 hr #3	1 sec
✓	*	demo-with-github	10 min #15	21 hr #3	0.95 sec
✓	*	demo-for-other-user	21 hr #2	N/A	62 ms
✓	*	demo-second	2 days 0 hr #17	2 days 0 hr #6	50 ms
✓	*	first-job	11 min #2	N/A	50 ms
✓	*	demo-first	4 min 26 sec #2	N/A	37 ms

**Build Executor Status:**

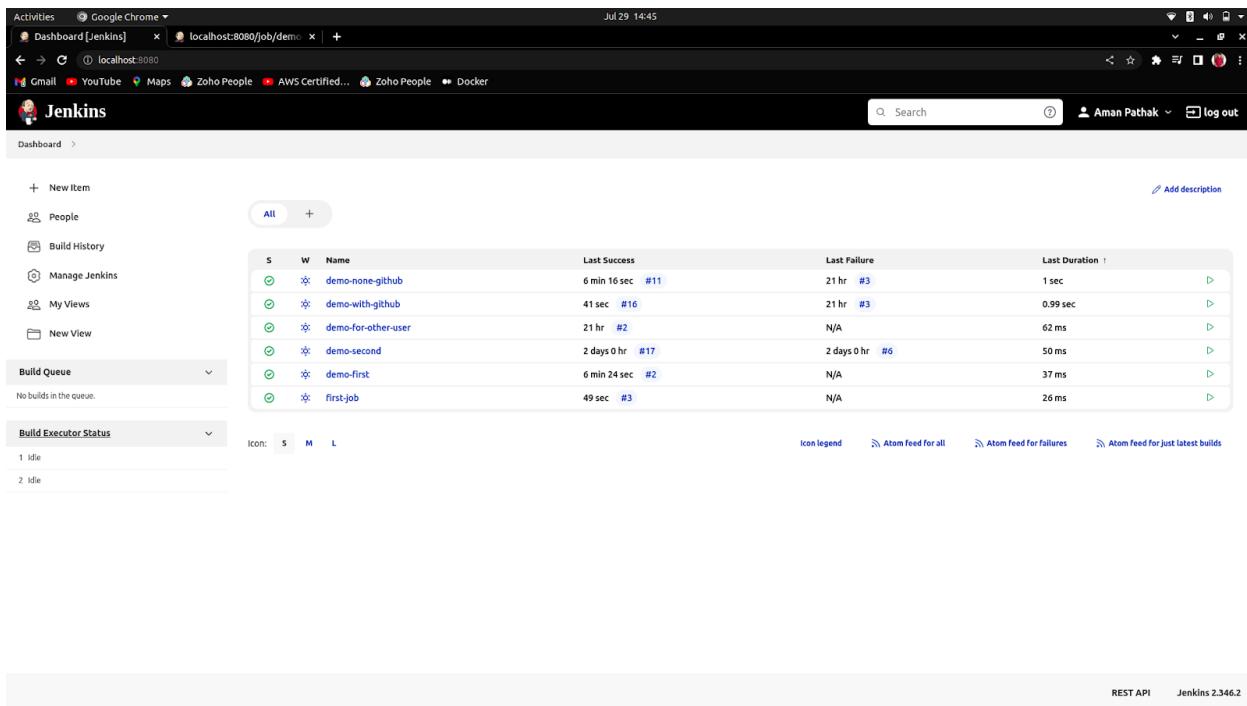
- Icon: S M L
- 1 Idle
- 2 Idle

**Build History:**

- </> Changes
- Workspace
- Build Now
- Configure
- Delete Project
- Rename

REST API Jenkins 2.346.2

#### 4. If you observe, Both build numbers are incremented by 1.



The screenshot shows the Jenkins dashboard with a table of build history. The table has columns for S (Status), W (Workstation), Name, Last Success, Last Failure, and Last Duration. The builds listed are:

S	W	Name	Last Success	Last Failure	Last Duration
Green	None	demo-none-github	6 min 16 sec #11	21 hr #3	1 sec
Green	None	demo-with-github	41 sec #16	21 hr #3	0.99 sec
Green	None	demo-for-other-user	21 hr #2	N/A	61 ms
Green	None	demo-second	2 days 0 hr #17	2 days 0 hr #6	50 ms
Green	None	demo-first	6 min 24 sec #2	N/A	37 ms
Green	None	first-job	49 sec #3	N/A	26 ms

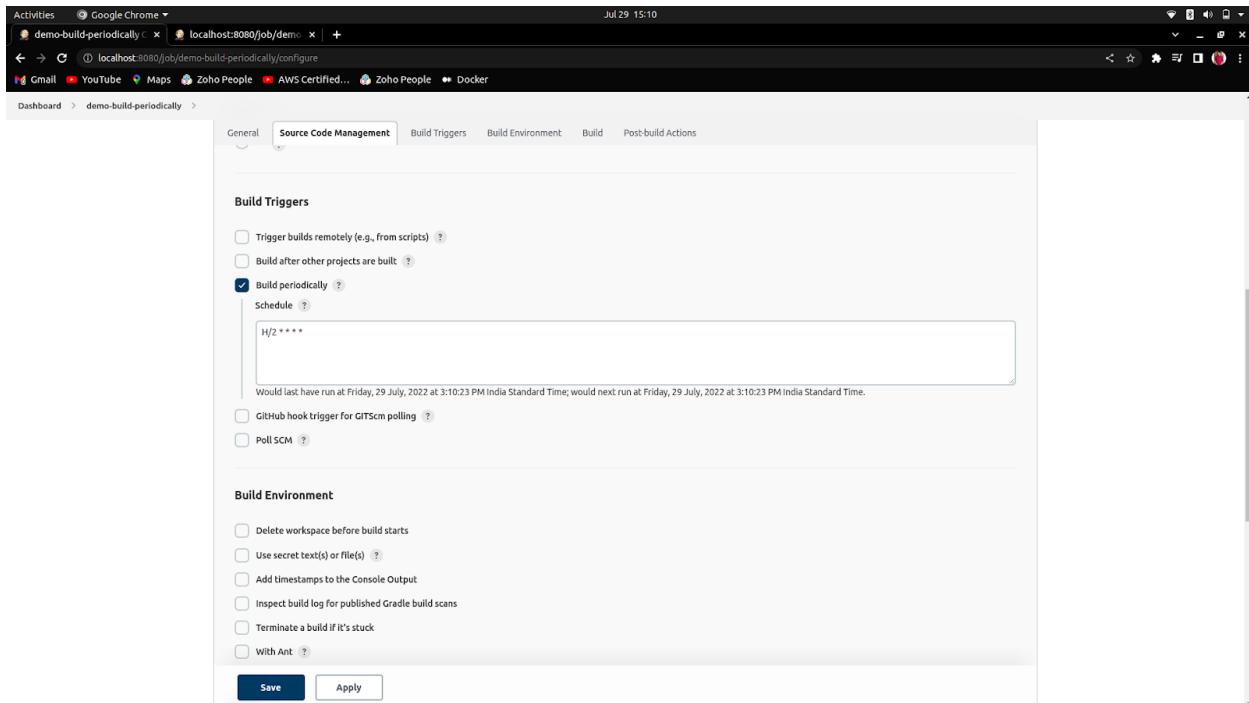
Build Executor Status: 1 Idle, 2 Idle. REST API Jenkins 2.346.2

# 3. Build Job Periodically

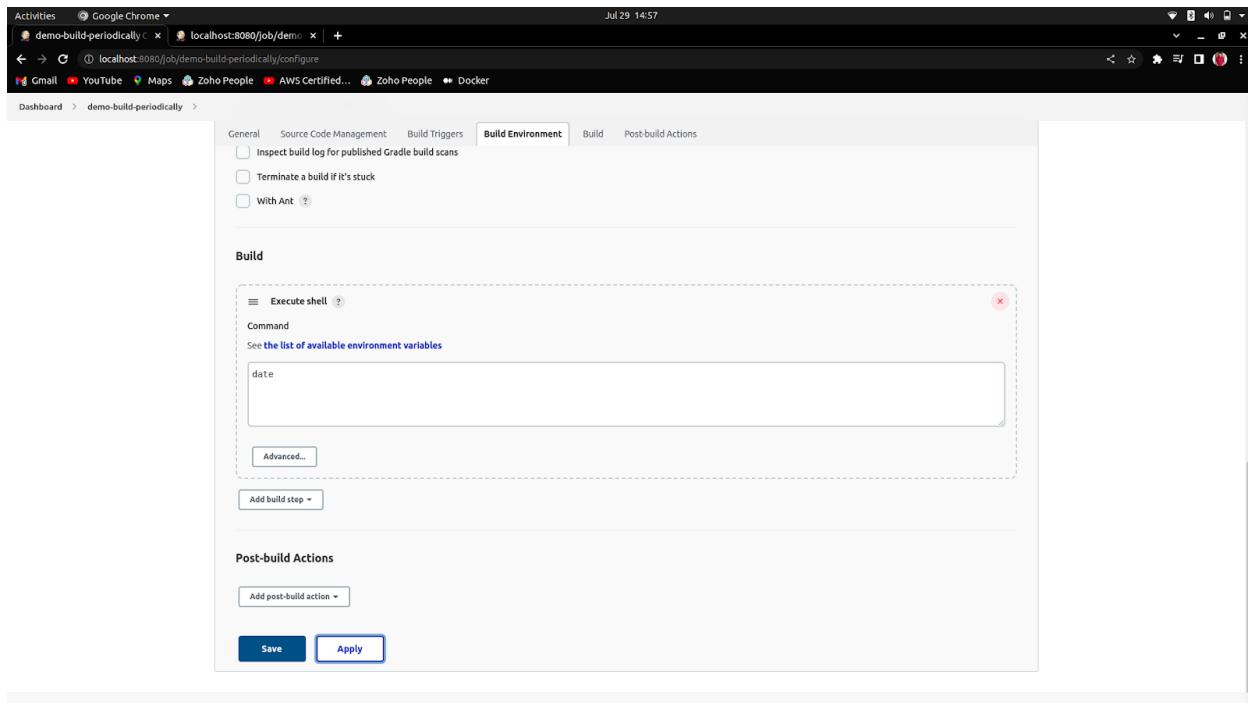
## Why?

When you have to build your job at a specific time or same interval e.g. every two minutes. You can use Build periodically under the Build Triggers section.

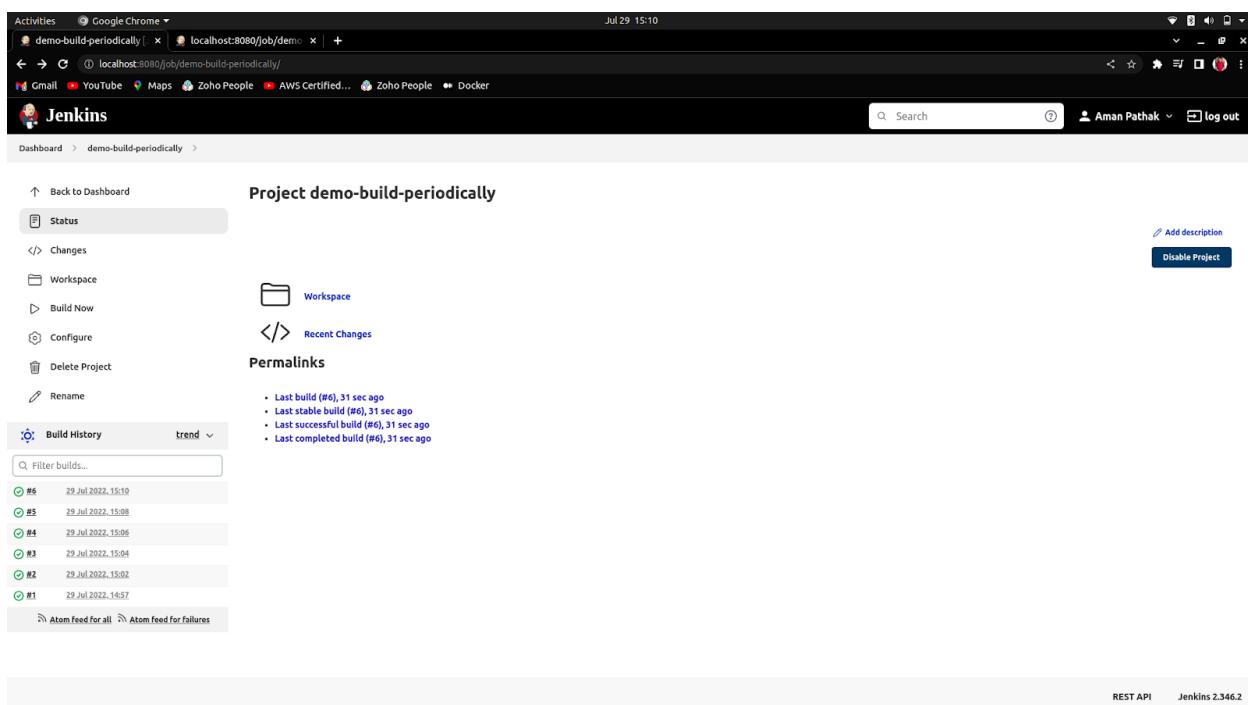
1. Write the time interval the same as written in the cron job.



## 2. Write the command and apply then save it.



## 3. Here, you can check it out. The build is automatically done every two minutes.

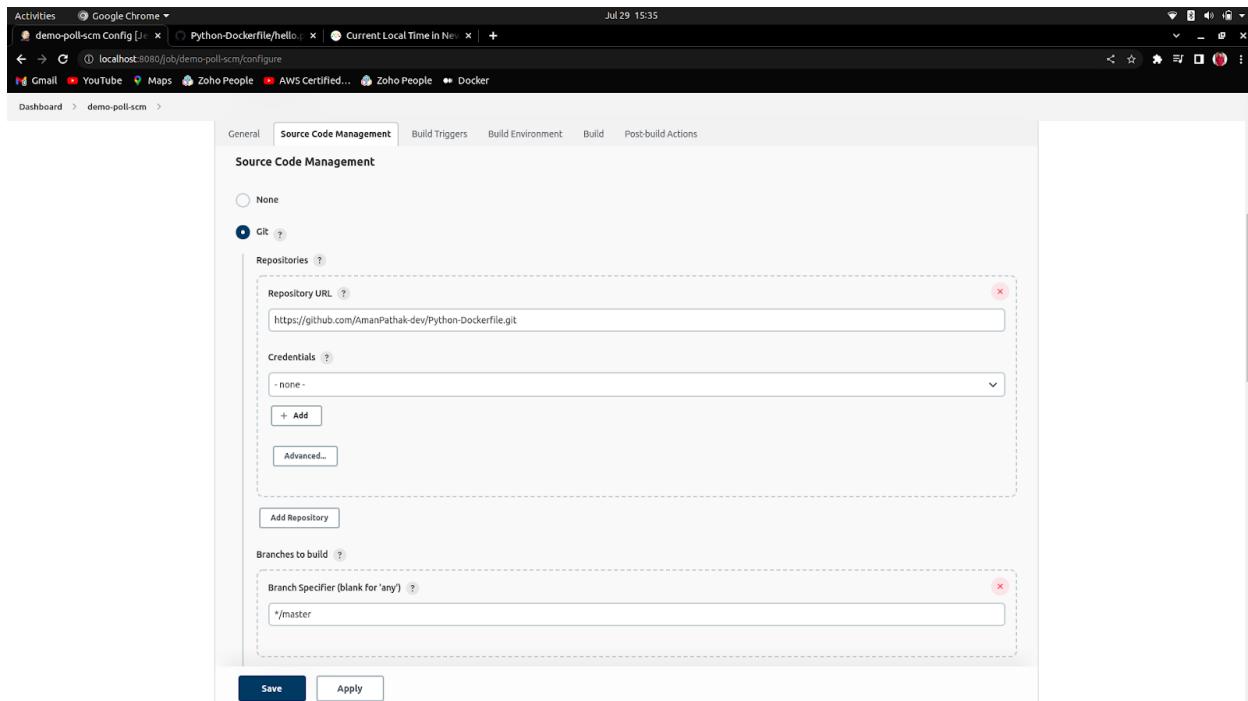


# 4. Poll SCM(Source Code Management)

## Why?

This Build trigger is the same as Build Periodically. But there is only one difference which is, that the project will fetch from the GitHub repository(mandatory) and if there is no commit in the repository then, the job will not build. But, if there is any commit change in the GitHub repository, then the job will build as a per-time scheduler which is also known as Cronjob/Crontab.

## 1. Configuration for the Project.



Activities Google Chrome ▾ Jul 29 15:35

demo-poll-scm Config [x] Python-Dockerfile/hello... [x] Current Local Time in Ne... [x] +  
localhost:8080/job/demo-poll-scm/configure

Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > demo-poll-scm >

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

Repository browser ? (Auto)

Additional Behaviours Add ▾

**Build Triggers**

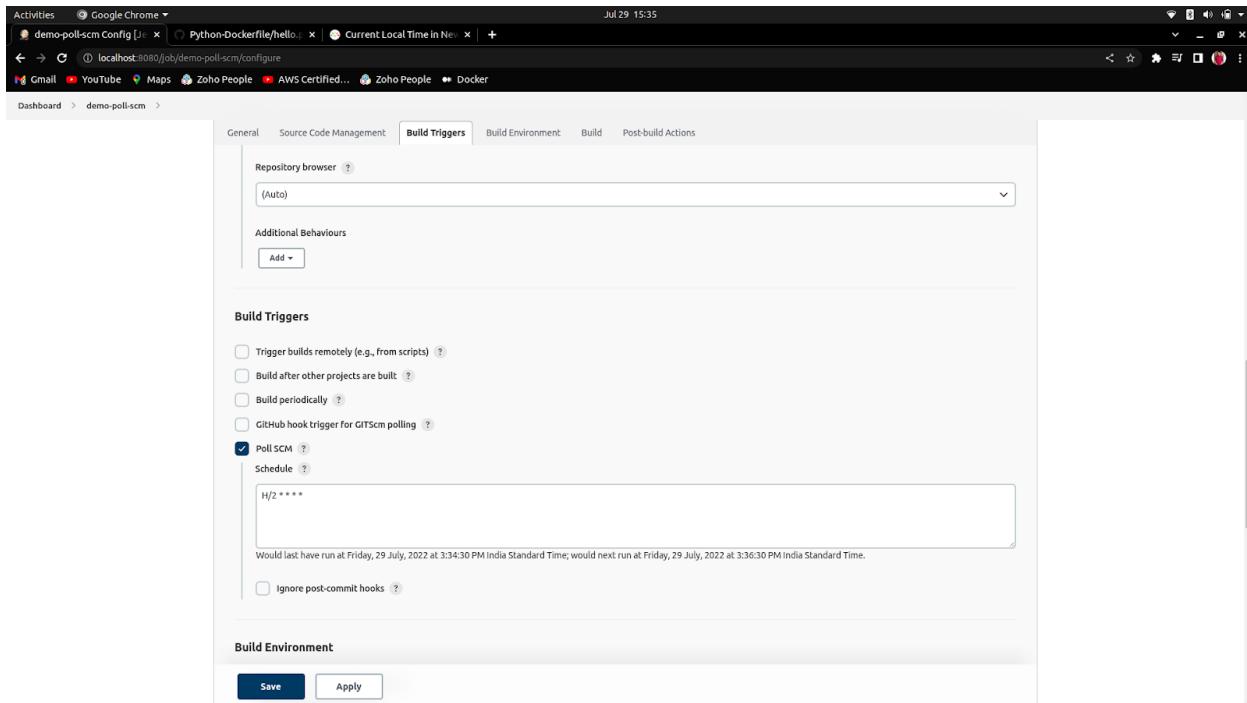
Trigger builds remotely (e.g., from scripts) ?  
 Build after other projects are built ?  
 Build periodically ?  
 GitHub hook trigger for GITScm polling ?  
 Poll SCM ? Schedule ? H/2 \* \* \* \*

Would last have run at Friday, 29 July, 2022 at 3:34:30 PM India Standard Time; would next run at Friday, 29 July, 2022 at 3:36:30 PM India Standard Time.

Ignore post-commit hooks ?

**Build Environment**

Save Apply



Activities Google Chrome ▾ Jul 29 15:35

demo-poll-scm Config [x] Python-Dockerfile/hello... [x] Current Local Time in Ne... [x] +  
localhost:8080/job/demo-poll-scm/configure

Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > demo-poll-scm >

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

Ignore post-commit hooks ?

**Build Environment**

Delete workspace before build starts  
 Use secret text(s) or file(s) ?  
 Add timestamps to the Console Output  
 Inspect build log for published Gradle build scans  
 Terminate a build if it's stuck  
 With Ant ?

**Build**

Execute shell ?

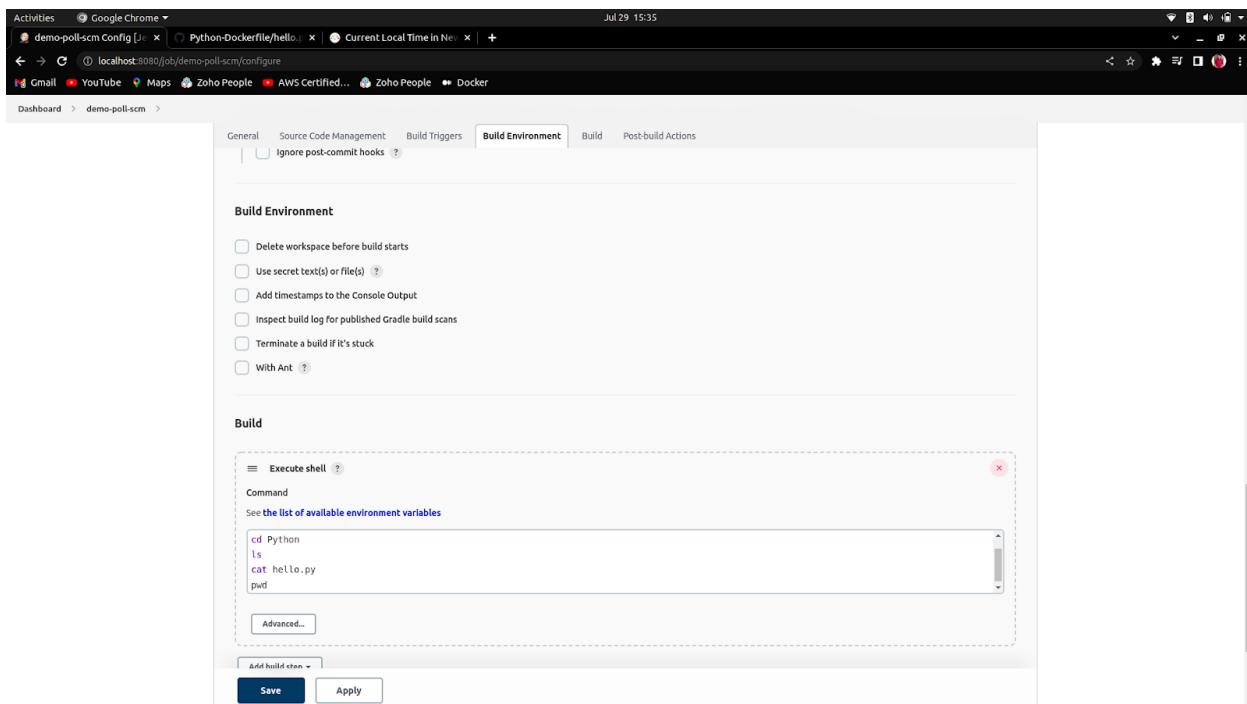
Command

See the list of available environment variables

```
cd Python
ls
cat hello.py
pwd
```

Advanced... And build step ▾

Save Apply



## 2. Job is built automatically. But then no job build

The screenshot shows the Jenkins interface for the project 'demo-poll-scm'. The top navigation bar includes tabs for 'Activities', 'Google Chrome', and 'Current Local Time in New York'. The main content area displays the 'Project demo-poll-scm' page. On the left, there's a sidebar with links like 'Back to Dashboard', 'Status' (which is currently selected), 'Changes', 'Workspace', 'Build Now', 'Configure', 'Delete Project', 'Git Polling Log', and 'Rename'. The 'Status' section shows a green icon for 'Recent Changes' and a list of four recent builds. Below this is a 'Build History' section with a dropdown menu set to 'trend'. A search bar for 'Filter builds...' is present. At the bottom of the sidebar, there are links for 'Atom feed for all' and 'Atom feed for failures'. The right side of the page has a 'Permalinks' section with a link to the project's permanent URL. Top right buttons include 'Add description' and 'Disable Project'. The footer of the page shows 'REST API' and 'Jenkins 2.346.2'.

## 3. Changing some files data of the GitHub repository to build a job.

The screenshot shows a GitHub commit dialog for the file 'hello.py' in the 'master' branch. The title of the dialog is 'Commit changes'. It contains a text input field with the placeholder 'Update hello.py' and a larger text area for an optional extended description. Below these fields are two radio button options: one selected for 'Commit directly to the master branch.' and another for 'Create a new branch for this commit and start a pull request.' A 'Commit changes' button is at the bottom left, and a 'Cancel' button is at the bottom right. The GitHub footer at the bottom of the screen includes links for Terms, Privacy, Security, Status, Docs, Contact GitHub, Pricing, API, Training, Blog, and About.

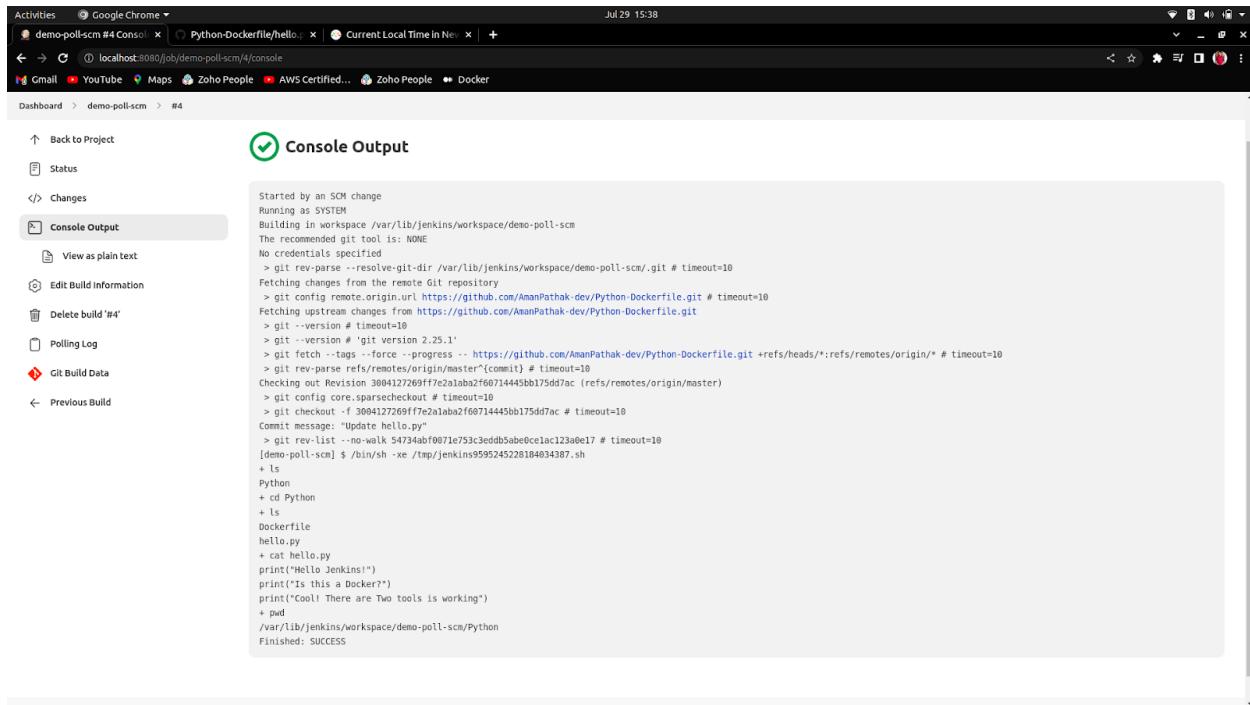
## 4. As GitHub repository has a new commit. Now, it is building a job.

The screenshot shows the Jenkins interface for the project 'demo-poll-scm'. The 'Status' tab is selected. In the 'Recent Changes' section, there is one entry: '#3 (pending—in the quiet period. Expires in 0.77 sec)'. The 'Build History' section shows three builds: #3 (pending), #2 (29 Jul 2022, 15:38), and #1 (29 Jul 2022, 15:26). The Jenkins version is 2.346.2.

## 5. The Build is successful.

The screenshot shows the Jenkins interface for the project 'demo-poll-scm'. The 'Status' tab is selected. In the 'Recent Changes' section, there are three entries: #4 (29 Jul 2022, 15:38), #3 (29 Jul 2022, 15:36), and #1 (29 Jul 2022, 15:26). The 'Build History' section shows four builds: #4 (29 Jul 2022, 15:38), #3 (29 Jul 2022, 15:36), #2 (29 Jul 2022, 15:38), and #1 (29 Jul 2022, 15:26). The Jenkins version is 2.346.2.

# The Output



A screenshot of a Google Chrome browser window displaying the Jenkins console output for a build named "demo-poll-scm #4". The title bar shows "Activities Google Chrome" and the URL "localhost:8080/job/demo-poll-scm/4/console". The page header includes links for "Back to Project", "Status", "Changes", "Console Output" (which is selected), "View as plain text", "Edit Build Information", "Delete build '#4'", "Polling Log", "Git Build Data", and "Previous Build". The main content area is titled "Console Output" with a green checkmark icon. It shows the command-line output of the build process:

```
Started by an SCM change
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/demo-poll-scm
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/demo-poll-scm/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/AmanPathak-dev/Python-Dockerfile.git # timeout=10
Fetching upstream changes from https://github.com/AmanPathak-dev/Python-Dockerfile.git
> git --version # timeout=10
> git -version # 'git' version 2.25.1'
> git fetch -tags --force --progress -- https://github.com/AmanPathak-dev/Python-Dockerfile.git +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 3004127269ff7e2a1aba2f60714445bb175dd7ac (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 3004127269ff7e2a1aba2f60714445bb175dd7ac # timeout=10
Commit message: "Update hello.py"
> git rev-list --no-walk 54734abf0071e752c3eddb5abe0ce1ac123a0e17 # timeout=10
[demo-poll-scm] $ /bin/sh -xe /tmp/jenkins959524520184034307.sh
+ ls
Python
+ cd Python
+ ls
Dockerfile
hello.py
+ cat hello.py
print("Hello Jenkins!")
print("Is this a Docker?")
print("Cool! There are two tools is working")
+ pwd
/var/lib/jenkins/workspace/demo-poll-scm/Python
Finished: SUCCESS
```

# How to define variables globally

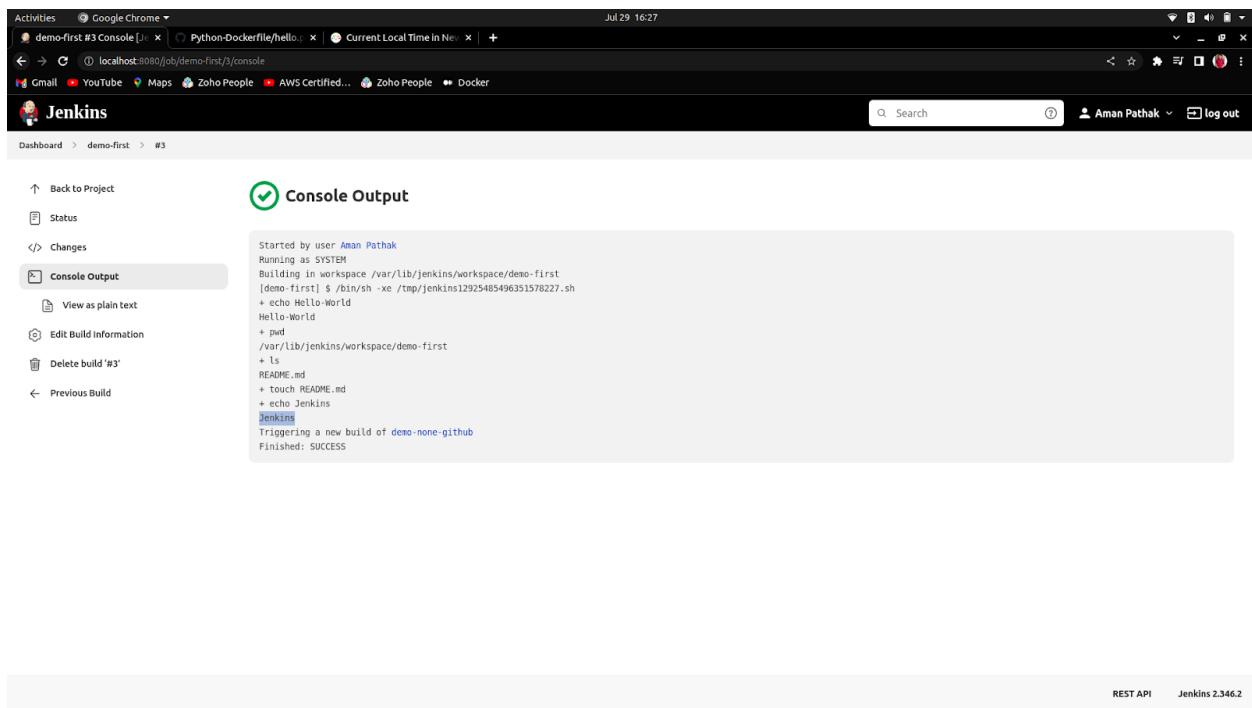
1. Go to the Manage Jenkins -> Configure System and scroll down then, look for Global properties.

The screenshot shows the Jenkins 'Configure System' page. In the 'Global properties' section, there is a checked checkbox for 'Environment variables'. A sub-section titled 'List of variables' contains one entry: 'Name: G\_VAR' and 'Value: Jenkins'. Below this, there is an 'Add' button and a 'Tool Locations' section which is currently empty. At the bottom of the page are 'Save' and 'Apply' buttons.

2. Now, Create a job and print that value of variable by passing the environment variable.

The screenshot shows the Jenkins 'demo-first' job configuration page. The 'Build Environment' tab is selected. Under the 'Build' section, there is a 'Execute shell' step with the command: 'pwd', 'ls', 'touch README.md', and 'echo \$G\_VAR'. Below this is an 'Advanced...' button. Under the 'Post-build Actions' section, there is a 'Save' and 'Apply' button at the bottom.

### 3. Here, you can see the value of variable is appeared.



The screenshot shows a Jenkins console output page. The left sidebar has links for Back to Project, Status, Changes, Console Output (which is selected), View as plain text, Edit Build Information, Delete build '#3', and Previous Build. The main area is titled 'Console Output' and shows the following log:

```
Started by user Aman Pathak
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/demo-first
[demo-first] $ /bin/sh -xe /tmp/jenkins12925465496351578227.sh
+ echo Hello-World
Hello-World
+ pwd
/var/lib/jenkins/workspace/demo-first
+ ls
README.md
+ touch README.md
+ echo Jenkins
Jenkins
Triggering a new build of demo-none-github
Finished: SUCCESS
```

At the bottom right of the page, it says 'REST API' and 'Jenkins 2.346.2'.

# Parameterised in Job

1. Define the variable and pass the default value.

The screenshot shows two consecutive steps of a configuration interface for a job named "demo-parameterized-job".

**Step 1 (Top Screenshot):** The "General" tab is selected. A checkbox labeled "This project is parameterised" is checked. Below it, there are two sections: "String Parameter" and "Choice Parameter".

- String Parameter:** Name is set to "NAME", Default Value is set to "DEFAULT". There is a "Description" field with a preview area showing "[Plain text] Preview" and a "Trim the string" checkbox.
- Choice Parameter:** Name is set to "CHOICES". The Choices field contains the values: Cricket, Football, Baseball, Volleyball.

**Step 2 (Bottom Screenshot):** The "General" tab is still selected. The "String Parameter" section has been removed. The "Choice Parameter" section remains, showing the same configuration. Below it, a new section has been added:

- Boolean Parameter:** Name is set to "GRADUATED". There is a "Set by Default" checkbox which is unchecked.

Both screenshots show a "Save" and an "Apply" button at the bottom.

Activities Google Chrome ▾

demo-parameterized-job x Python-Dockerfile/hello... x +

localhost:8080/job/demo-parameterized-job/configure

Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > demo-parameterized-job >

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

Add Parameter ▾

Throttle builds ?  
 Disable this project ?  
 Execute concurrent builds if necessary ?

Advanced...

**Source Code Management**

None  
 Git ?

**Build Triggers**

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

**Build Environment**

Save Apply

Activities Google Chrome ▾

demo-parameterized-job x Python-Dockerfile/hello... x +

localhost:8080/job/demo-parameterized-job/configure

Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > demo-parameterized-job >

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

**Password Parameter** ?

Name ?  
PASSWD

Default Value ?  
Concealed

Description ?  
passwd

[Plain text] Preview

**Multi-line String Parameter** ?

Name ?  
MULTILINE

Default Value ?  
This  
is  
Jenkins

Description ?

Save Apply

The screenshot shows the Jenkins configuration interface for a job named "demo-parameterized-job". The "Source Code Management" tab is selected. A "File Parameter" section is open, showing a "FILE" parameter with a description field. Below it are checkboxes for "Throttle builds", "Disable this project", and "Execute concurrent builds if necessary". At the bottom are "Save" and "Apply" buttons.

The screenshot shows the Jenkins configuration interface for the same job. The "Build Environment" tab is selected. It contains sections for workspace cleanup, timestamps, build logs, and Ant termination. The "Build" section is expanded, showing an "Execute shell" step with a command field containing a multi-line echo script. At the bottom are "Save" and "Apply" buttons.

```
echo "Are you Graduated? ${GRADUATE}"
echo "Enter the Password!! ${PASSWD}"
echo "Some lines!! ${MULTILINE}"
echo "Content of File"
cat FILE
```

# Console Output

The screenshot shows a Jenkins job named "demo-parameterized-job" running on port 8080. The job has a build number of #5. The "Console Output" tab is selected, showing the following log:

```
Started by user Aman Pathak
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/demo-parameterized-job
Copying file to FILE
[demo-parameterized-job] $ /bin/sh -xe /tmp/jenkins345212945448071791.sh
+ echo Hello Imran Khan!!!
Hello Imran Khan!!!
+ echo Which sports do you play Cricket
Which sports do you play Cricket
+ echo Are you Graduated?
Are you Graduated?
+ echo Enter the Password!! aman@123
Enter the Password!! aman@123
+ echo Some lines!! This
is
Jenkins
print("Hello Jenkins!")
print("Is this a Docker?")
print("Cool! There are Two tools is working")
Some lines!! This
is
Jenkins
print("Hello Jenkins!")
print("Is this a Docker?")
print("Cool! There are Two tools is working")
+ echo Content of File
Content of File
+ cat FILE
FROM httpd:2.4.54-alpine
COPY . /usr/local/apache2/htdocs/
Finished: SUCCESS
```

# Custom Workspace

1. Create a new job and do the configuration as given below:

The screenshot shows the Jenkins job configuration interface for a job named "demo-custom-workspace".

**General Settings:**

- Project Description: [Empty]
- Checkboxes (unchecked): Discard old builds, GitHub project, This project is parameterised, Throttle builds, Disable this project, Execute concurrent builds if necessary, Quiet period, Retry Count, Block build when upstream project is building, Block build when downstream project is building.
- Checkboxes (checked): Use custom workspace.
- Directory: /var/lib/jenkins/myws/demospace

**Build Section:**

**Execute shell:**

```
pwd
touch init.txt
```

**Post-build Actions:**

Add post-build action ▾

**Buttons:**

Save      Apply

2. Now, the directory has been created and file too.

To check it, go to your terminal.

```
amanpathak@aman-pathak:/var/lib/jenkins$ cd myws/
amanpathak@aman-pathak:/var/lib/jenkins/myws$ ls
demospace
amanpathak@aman-pathak:/var/lib/jenkins/myws$ cd demospace/
amanpathak@aman-pathak:/var/lib/jenkins/myws/demospace$ ls
init.txt
```

# Change display name and project name

1. Go to in the Configure of the Specific Project-> Enter the desired name that you want in the text field.



## The Output

But the Project hasn't changed. For that you have to click on the Rename button which is left on the screen and enter the desired name.

## Now, The project name has changed too.

The screenshot shows a Jenkins project page titled "Project Change Workspace Project". The URL in the browser is `localhost:8080/job/demo-custom-my-workspace/`. The page header includes the Jenkins logo, user "Aman Pathak", and a "log out" link. On the left, there's a sidebar with links like "Back to Dashboard", "Status" (which is selected), "Changes", "Workspace", "Build Now", "Configure", "Delete Project", and "Rename". The main content area shows a "Recent Changes" section with a list of build logs:

- Last build (#2), 8 min 36 sec ago
- Last stable build (#2), 8 min 36 sec ago
- Last successful build (#2), 8 min 36 sec ago
- Last failed build (#1), 10 min ago
- Last unsuccessful build (#1), 10 min ago
- Last completed build (#2), 8 min 36 sec ago

Below this is a "Build History" section with two entries:

- #2 29 Jul 2022, 19:17 (green circle)
- #1 29 Jul 2022, 19:17 (red circle)

At the bottom, there are links for "Atom feed for all" and "Atom feed for failures". The footer of the page includes links for "REST API" and "Jenkins 2.346.2".

# Building Upstream and Downstream Project

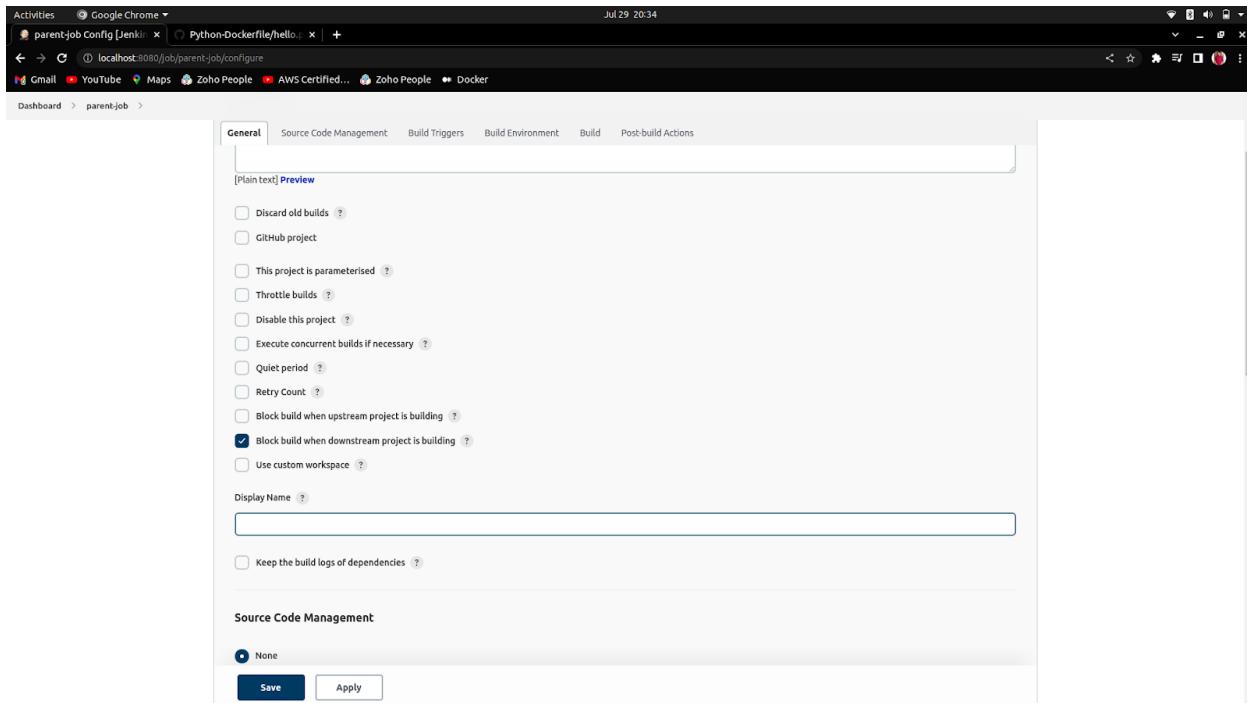
Upstream- Parent job

Downstream- Child job

That job that is triggering another job is known as the Parent job and that job that is triggered by another job/parent job is known as the Child job.

## Block build when the upstream project is building

Here, the scenario is if you are building a parent job then, you couldn't build a child job at the same time. To build the child job, you have to complete the build of the parent job first.



Activities Google Chrome ▾ Jul 29 20:40

parent-job Config [Jenkins] | Python-Dockerfile/hello... | +  
localhost:8080/job/parent-job/configure

Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > parent-job >

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

**Build Triggers**

Trigger builds remotely (e.g., from scripts) ?  
 Build after other projects are built ?  
Projects to watch  
child-job,

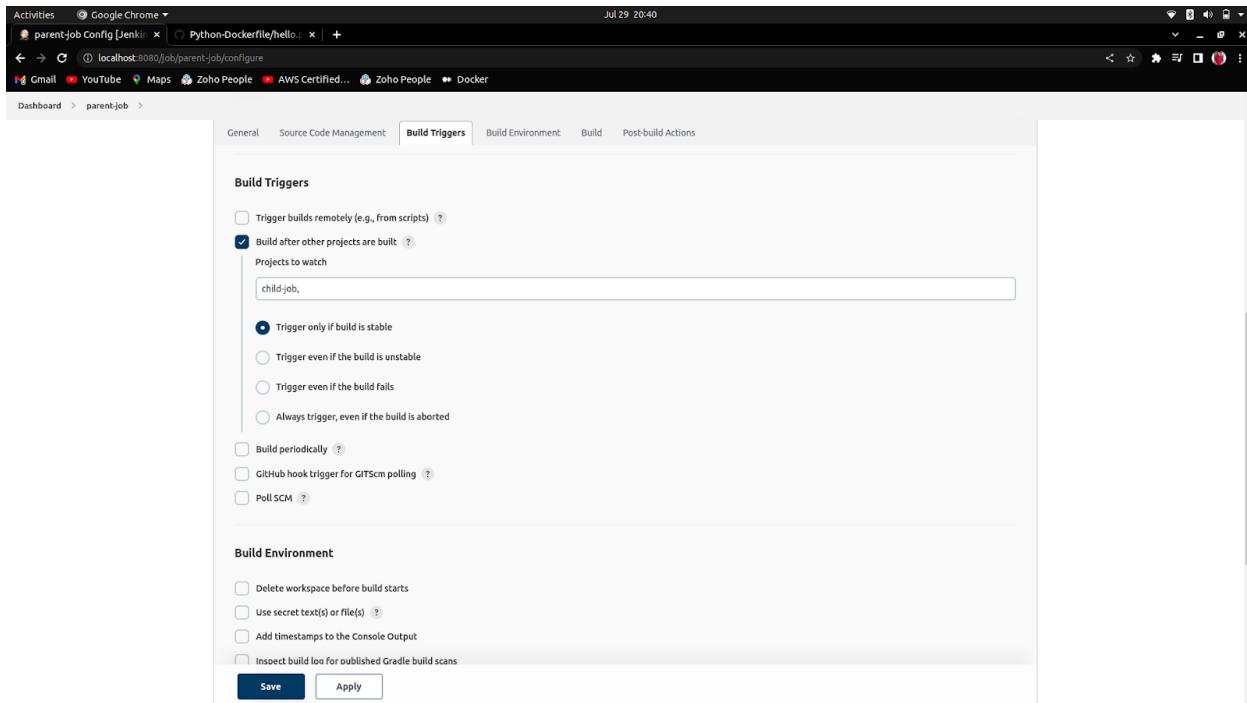
Trigger only if build is stable  
 Trigger even if the build is unstable  
 Trigger even if the build fails  
 Always trigger, even if the build is aborted

Build periodically ?  
 GitHub hook trigger for GITScm polling ?  
 Poll SCM ?

**Build Environment**

Delete workspace before build starts  
 Use secret text(s) or file(s) ?  
 Add timestamps to the Console Output  
 Inspect build log for published Gradle build scans

**Buttons:** Save Apply



# Block build when downstream project is building

Here, the scenario is if you are building a child job then, you couldn't build a parent job at the same time. To build the parent job, you have to complete the build of the child job first.

Complete Reverse of the previous one.

The screenshot shows a Jenkins configuration page for a job named 'child-job'. The 'General' tab is selected. In the 'Build Triggers' section, the checkbox 'Block build when downstream project is building' is checked. Other options like 'Block build when upstream project is building' and 'Use custom workspace' are also present but unchecked. The 'Source Code Management' section shows 'None' selected. At the bottom, there are 'Save' and 'Apply' buttons.

The screenshot shows the Jenkins configuration interface for a job named "child-job". The "Source Code Management" tab is selected. The "Build Triggers" section contains the following configuration:

- Trigger builds remotely (e.g., from scripts) ?
- Build after other projects are built. Projects to watch: parent-job.
- Trigger only if build is stable
- Trigger even if the build is unstable
- Trigger even if the build fails
- Always trigger, even if the build is aborted

The "Build Environment" section contains the following configuration:

- Delete workspace before build starts
- Use secret text(s) or file(s) ?
- Add timestamps to the Console Output

At the bottom are "Save" and "Apply" buttons.

The screenshot shows a Google Chrome browser window with two tabs open: 'Available Plugins - Plugin Manager' and 'Python-Dockerfile/hello...'. The main content area is the Jenkins 'Plugin Manager' interface, specifically the 'Available' tab. A search bar at the top contains the query 'C. build pipe'. Below it, a table lists available plugins:

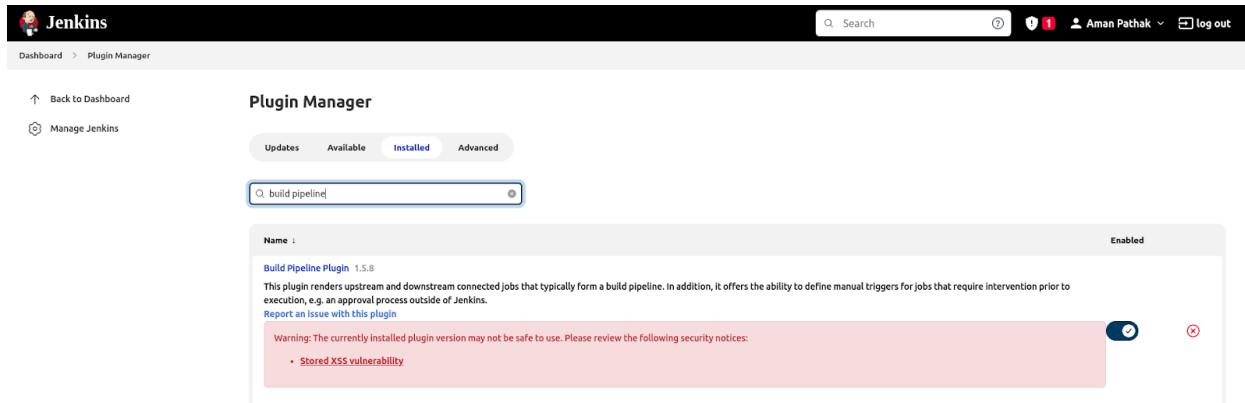
Install	Name	Released
<input checked="" type="checkbox"/>	<a href="#">Build Pipeline 1.5.8</a>	4 yr 7 mo ago
<input type="checkbox"/>	<a href="#">Webhook Step 173.vfa_b_93560b_977</a>	2 mo 12 days ago
<input type="checkbox"/>	<a href="#">Pipeline timeline 1.0.3</a>	3 yr 5 mo ago

A red box highlights a warning message for the 'Build Pipeline' plugin: "Warning: This plugin version may not be safe to use. Please review the following security notices:" followed by a bullet point: "• Stored XSS vulnerability".

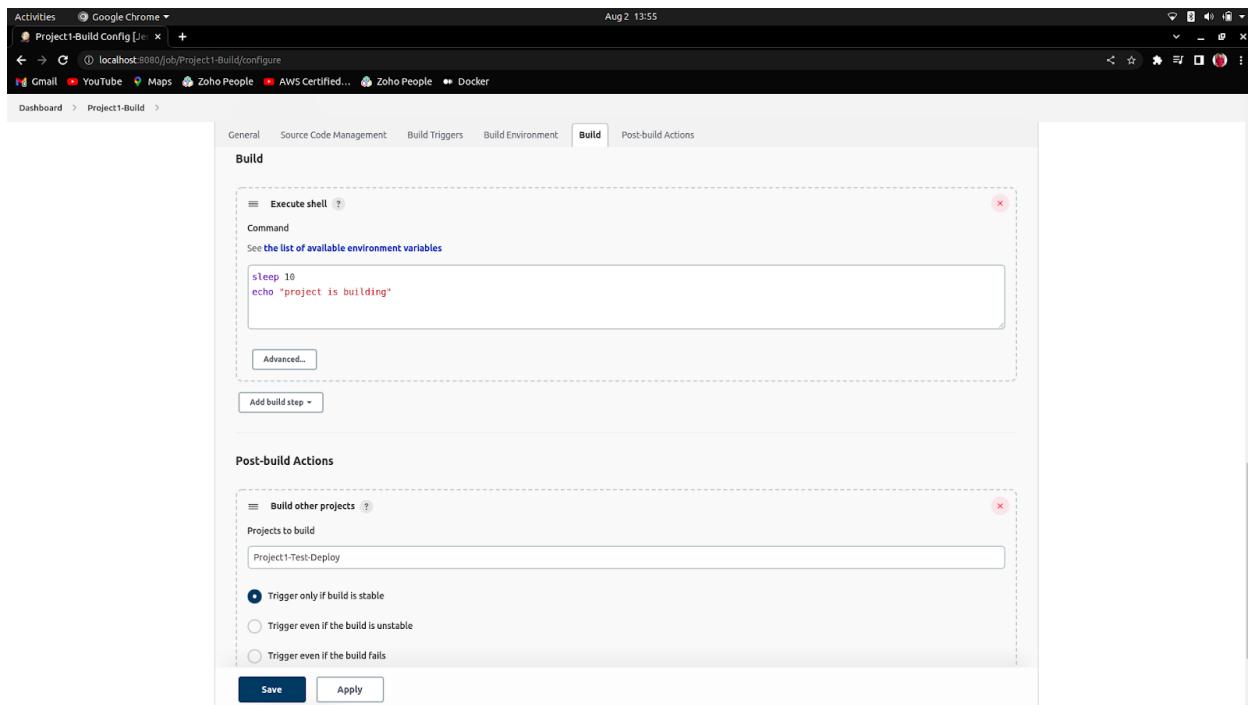
At the bottom of the page, there are three buttons: 'Install without restart' (disabled), 'Download now and install after restart' (highlighted in blue), and 'Check now'.

# Jenkins Pipeline using Build Pipeline

1. First of all, Install the plugin named Build Pipeline.



2. Create two Jobs, first will be Parent and the second will be Child job and assign the Post-build actions in Parent Job for Child job as per below:



The screenshot shows the Jenkins configuration interface for the 'Project1-Test-Deploy' job. The 'Build Environment' tab is selected. Under the 'Build' section, there is a step titled 'Execute shell' containing the command:

```
sleep 10  
echo "deploying to testing environment"
```

Below the build steps, the 'Post-build Actions' section is empty. At the bottom, there are 'Save' and 'Apply' buttons.

### 3. Click on “+” to create the Build Pipeline.

The screenshot shows the Jenkins Pipeline library interface. It lists several jobs under the 'All' category:

S	W	Name	Last Success	Last Failure	Last Duration
✓	Cloud	Change Workspace Project	3 days 18 hr #2	3 days 18 hr #1	26 ms
✓	DSL	child-job	3 days 17 hr #13	N/A	23 ms
✓	DSL	demo-build-periodically	3 days 21 hr #49	N/A	25 ms
✓	DSL	demo-first	3 days 21 hr #3	N/A	33 ms

## 4. Choose Type as “Build Pipeline view” and assign the name of your Pipeline.

### New view

Name

Type

Build Pipeline View  
Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.

List View  
Shows items in a simple list format. You can choose which jobs are to be displayed in which view.

My View  
This view automatically displays all the jobs that the current user has an access to.

**Create**

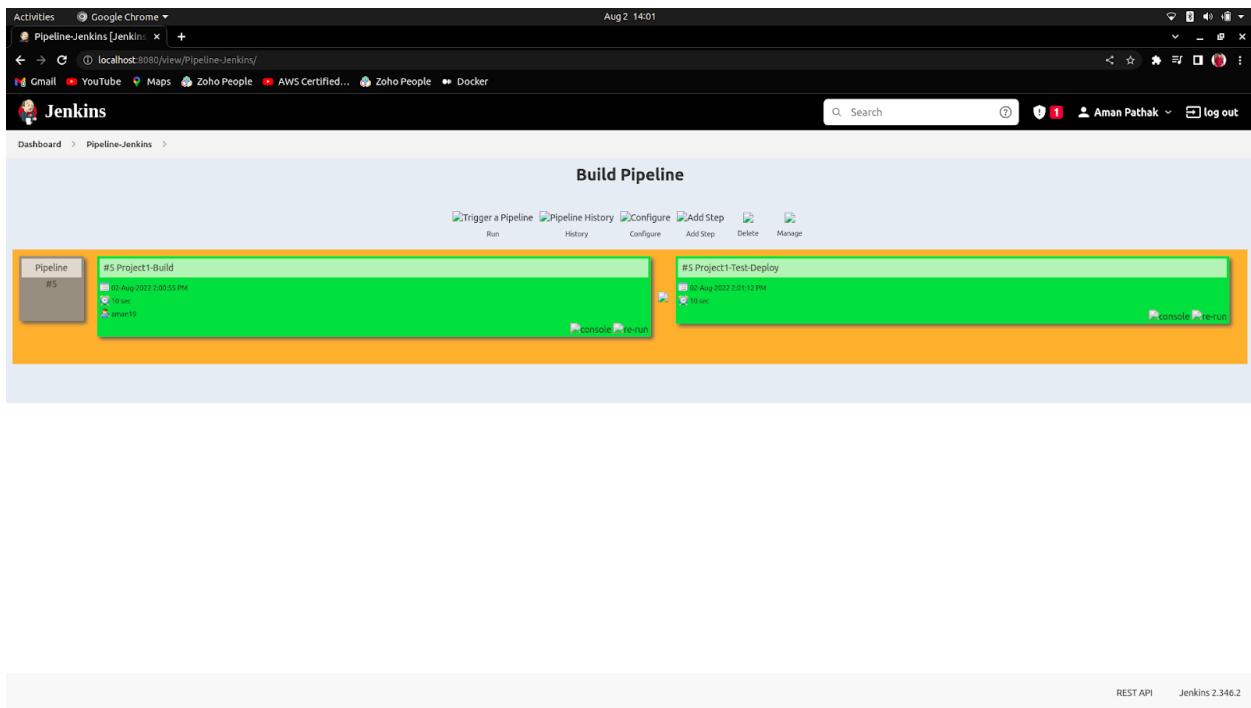
## 5. Now, all you have to do is “Select the Initial Job” which will be your Parent's job.

The screenshot shows the Jenkins Pipeline View configuration page. At the top, there is a header with the title "Edit View [Jenkins]". Below the header, there are several sections:

- Build Queue**: Shows "No builds in the queue."
- Build Executor Status**: Shows "1 Idle" and "2 Idle".
- Pipeline Flow**: A section titled "Build Pipeline View Title" contains a text input field. Below it is a dropdown menu set to "Based on upstream/downstream relationship". A note states: "This layout mode derives the pipeline structure based on the upstream/downstream trigger relationship between jobs. This is the only out-of-the-box supported layout mode, but is open for extension."
- Upstream / downstream config**: A dropdown menu titled "Select Initial Job" is set to "Project1-Build".
- Trigger Options**:
  - Build Cards**: A dropdown menu set to "Standard build card". A note says: "Use the default build cards".
  - Restrict triggers to most recent successful builds**: A radio button labeled "Yes" is selected.

At the bottom of the page, there are "OK" and "Apply" buttons.

6. In the last, Click on the run button and the result will be in front of you like this:

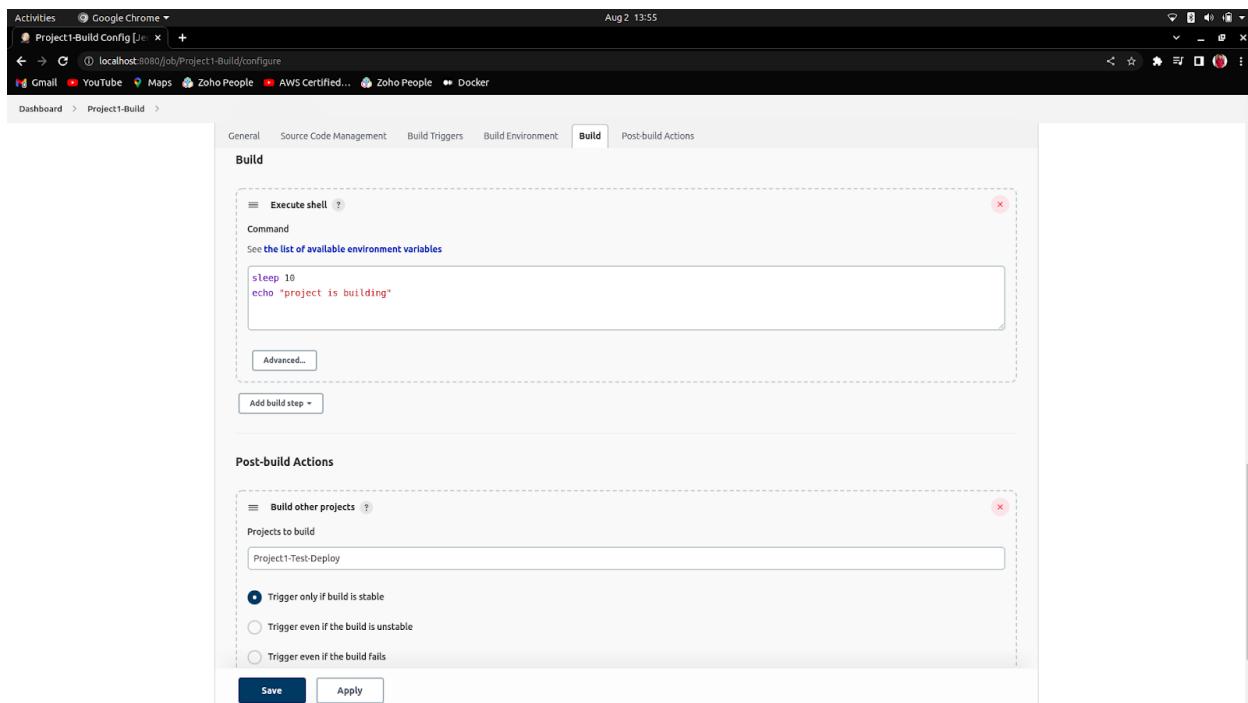


# Continuous Deployment vs Continuous Delivery

In Continuous Deployment, whenever the code changes to an application is released automatically into the production environment. There is no inclusion of human intervention or manual click.

E.g

1. Create three Jobs, first will be GrandParent, the second will be Parent job and the third will be Child job and assign the Post-build actions in Grand Parent Job for Parent job and in Parent Job for Child Job as per below:



Activities Google Chrome Project1-Test-Deploy Configuration

localhost:8080/job/Project1-Test-Deploy/configure

Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > Project1-Test-Deploy >

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

Command

See the list of available environment variables

```
sleep 10  
echo "deploying to testing environment"
```

Advanced...

Add build step ▾

Post-build Actions

Build other projects

Projects to build

Project1-Prod-Deploy

Trigger only if build is stable

Trigger even if the build is unstable

Trigger even if the build fails

Add post-build action ▾

Save Apply

Aug 2 14:18

Activities Google Chrome Project1-Prod-Deploy Configuration

localhost:8080/job/Project1-Prod-Deploy/configure

Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > Project1-Prod-Deploy >

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

Add timestamps to the Console Output

Inspect build log for published Gradle build scans

Terminate a build if it's stuck

With Ant

Build

Execute shell

Command

See the list of available environment variables

```
sleep 10  
echo "Production Deployment"
```

Advanced...

Add build step ▾

Post-build Actions

Add post-build action ▾

Save Apply

Aug 2 14:17

2. Click on “+” to create the Build Pipeline.

The screenshot shows the Jenkins dashboard with a list of jobs. At the top, there are buttons for 'All' and '+'. On the right, there is a link 'Add description'. The table has columns for S, W, Name, Last Success, Last Failure, and Last Duration. The jobs listed are:

S	W	Name	Last Success	Last Failure	Last Duration
🕒	🕒	Change Workspace Project	3 days 18 hr #2	3 days 18 hr #1	26 ms
🕒	⌚	child-job	3 days 17 hr #13	N/A	23 ms
🕒	⌚	demo-build-periodically	3 days 21 hr #49	N/A	25 ms
🕒	⌚	demo-first	3 days 21 hr #3	N/A	33 ms

3. Choose “Build Pipeline view” and assign the name of your Pipeline.

#### New view

Name

Type

**Build Pipeline View**

Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.

**List View**

Shows items in a simple list format. You can choose which jobs are to be displayed in which view.

**My View**

This view automatically displays all the jobs that the current user has an access to.

**Create**

4. Now, all you have to do is “Select the Initial Job” which will be your Parent's job.

The screenshot shows the Jenkins Pipeline configuration interface. At the top, there's a navigation bar with 'Activities', 'Edit View [Jenkins]', and a search bar. Below it, the URL is 'localhost:8080/view/Pipeline-Jenkins/configure'. The main area has tabs for 'Build Queue' (No builds in the queue), 'Build Executor Status' (1 Idle, 2 Idle), and 'Pipeline Flow'. Under 'Pipeline Flow', there's a 'Layout' section set to 'Based on upstream/downstream relationship'. A note says: 'This layout mode derives the pipeline structure based on the upstream/downstream trigger relationship between jobs. This is the only out-of-the-box supported layout mode, but is open for extension.' Below this is the 'Upstream / downstream config' section with a dropdown for 'Select Initial Job' containing 'Project1-Build'. The 'Trigger Options' section includes a 'Build Cards' dropdown set to 'Standard build card' and a radio button for 'Restrict triggers to most recent successful builds' (set to 'Yes'). Buttons for 'OK' and 'Apply' are at the bottom.

## 5. The Final Output:

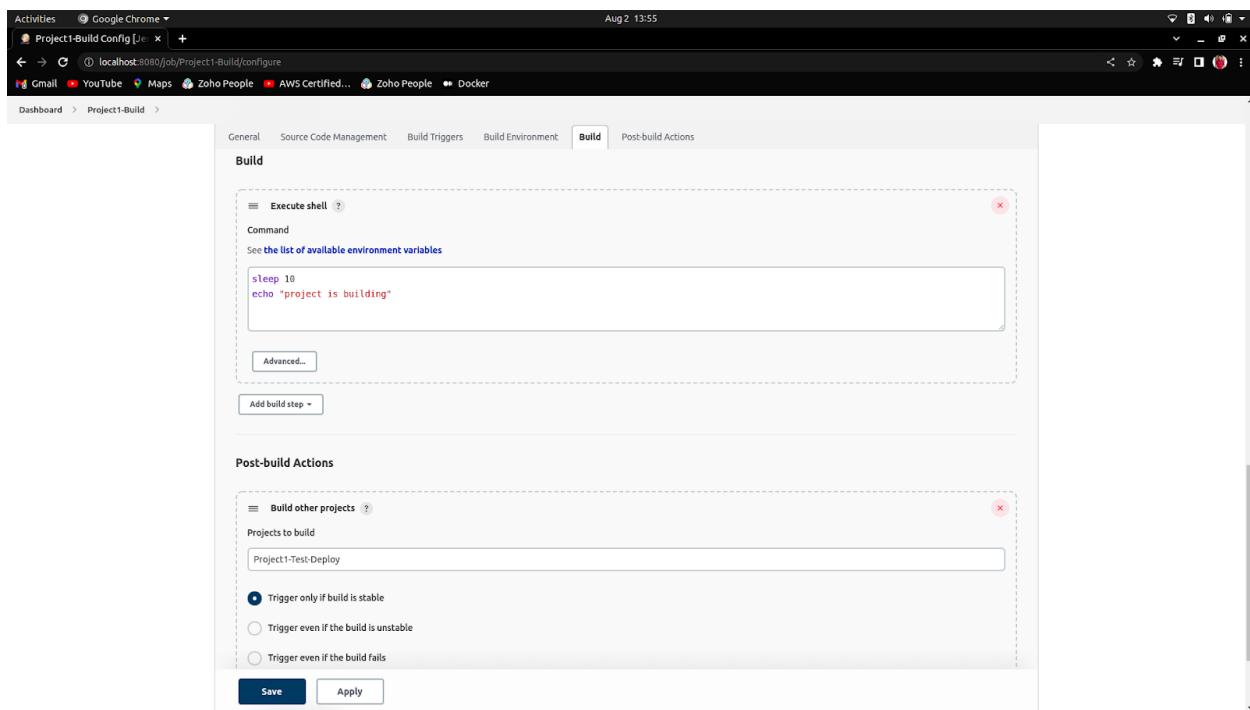
The screenshot shows the Jenkins Pipeline execution results. The top navigation bar includes 'Jenkins', 'Search', and user 'Aman Pathak'. The main area is titled 'Build Pipeline' with tabs for 'Run', 'History', 'Configure', 'Add Step', 'Delete', and 'Manage'. It displays three pipeline items: '#6 Project1-Build' (status green, last run 10 Aug 2022 2:59:12 PM, duration 79 sec, console link), '#6 Project1-Test-Deploy' (status green, last run 10 Aug 2022 2:59:27 PM, duration 17 sec, console link), and '#1 Project1-Prod-Deploy' (status green, last run 10 Aug 2022 2:59:47 PM, duration 26 sec, console link). Each item has a 're-run' button at the end.

# Continuous Delivery

In Continuous Delivery, whenever the code changes to an application are released only by human intervention or through manual clicks into the production environment. There is no any Automation in the Continuous Delivery.

E.g,

1. Create three Job, first will be GrandParent, the second will be Parent job and third will be Child job and assign the Post-build actions in Grand Parent Job for Parent job and rest the last will be **Manually** as per below:



Activities Google Chrome Project1-Test-Deploy Configuration Aug 2 14:23

localhost:8080/job/Project1-Test-Deploy/configure

Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > Project1-Test-Deploy >

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

Command  
See the list of available environment variables

```
sleep 10
echo "deploying to testing environment"
```

Advanced... Add build step ▾

Post-build Actions

Build other projects (manual step) ?

Downstream Project Names

Project1-Prod-Deploy, Add Parameters ▾

Add post-build action ▾

Save Apply

REST API Jenkins 2.346.2

This screenshot shows the Jenkins configuration interface for a job named 'Project1-Test-Deploy'. The 'Build' tab is active. Under the 'Command' section, there is a text input containing the command 'sleep 10' followed by the echo command 'echo "deploying to testing environment"'. Below this is an 'Advanced...' button. There is also a 'Add build step ▾' button. The 'Post-build Actions' section contains a single entry for 'Build other projects (manual step)' with the downstream project name 'Project1-Prod-Deploy' listed. There is an 'Add Parameters ▾' button next to it. At the bottom of the configuration area are 'Save' and 'Apply' buttons.

Activities Google Chrome Project1-Prod-Deploy Configuration Aug 2 14:17

localhost:8080/job/Project1-Prod-Deploy/configure

Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > Project1-Prod-Deploy >

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

Add timestamps to the Console Output

Inspect build log for published Gradle build scans

Terminate a build if it's stuck

With Ant ?

Build

Execute shell ?

Command  
See the list of available environment variables

```
sleep 10
echo "Production Deployment"
```

Advanced... Add build step ▾

Post-build Actions

Add post-build action ▾

Save Apply

This screenshot shows the Jenkins configuration interface for a job named 'Project1-Prod-Deploy'. The 'Build Environment' tab is active. Under the 'Build' section, there is a command step with the command 'sleep 10' followed by 'echo "Production Deployment"'. Below this is an 'Advanced...' button and an 'Add build step ▾' button. The 'Post-build Actions' section contains a single entry for 'Build other projects (manual step)' with the downstream project name 'Project1-Prod-Deploy' listed. There is an 'Add Parameters ▾' button next to it. At the bottom of the configuration area are 'Save' and 'Apply' buttons.

2. Click on “+” to create the Build Pipeline.



A screenshot of the Jenkins dashboard showing a list of jobs. The table has columns for S (Status), W (Workspace), Name, Last Success, Last Failure, and Last Duration. There are four jobs listed:

S	W	Name	Last Success	Last Failure	Last Duration
Green	Cloud	Change Workspace Project	3 days 18 hr #2	3 days 18 hr #1	26 ms
Green	Job	child-job	3 days 17 hr #13	N/A	23 ms
Green	Job	demo-build-periodically	3 days 21 hr #49	N/A	25 ms
Green	Job	demo-first	3 days 21 hr #3	N/A	33 ms

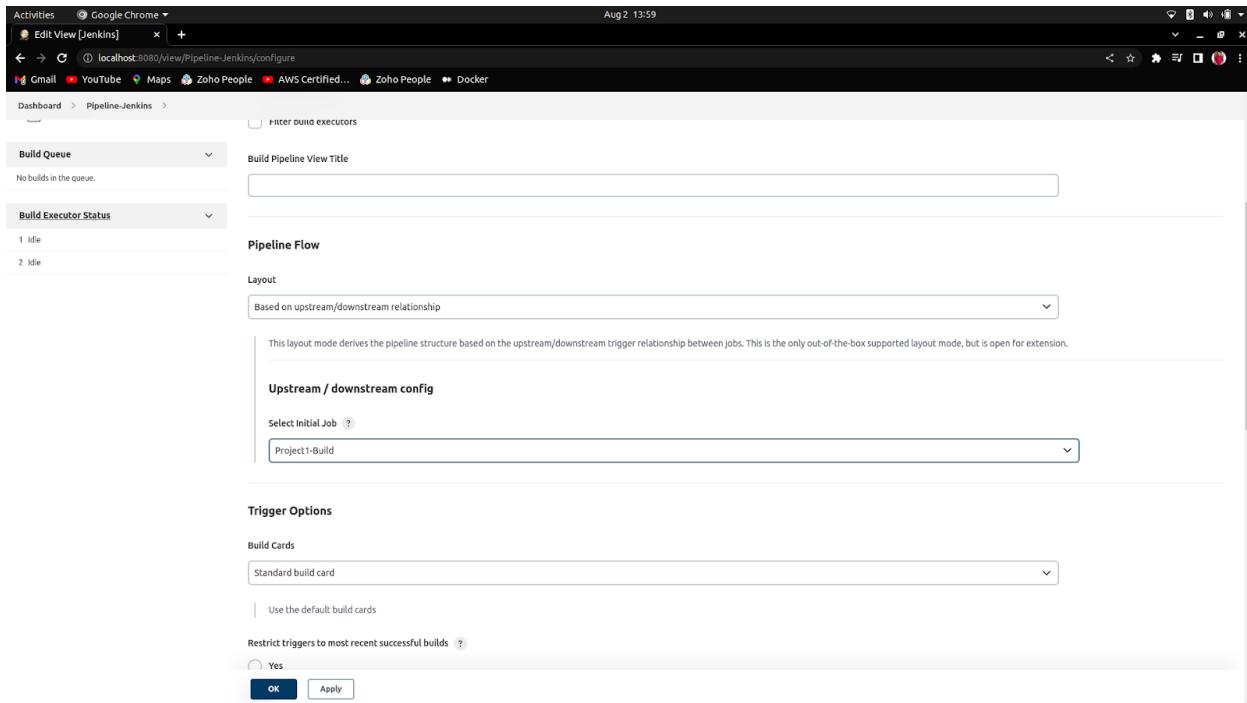
3. Choose Type “Build Pipeline view” and assign the name of your Pipeline.

#### New view

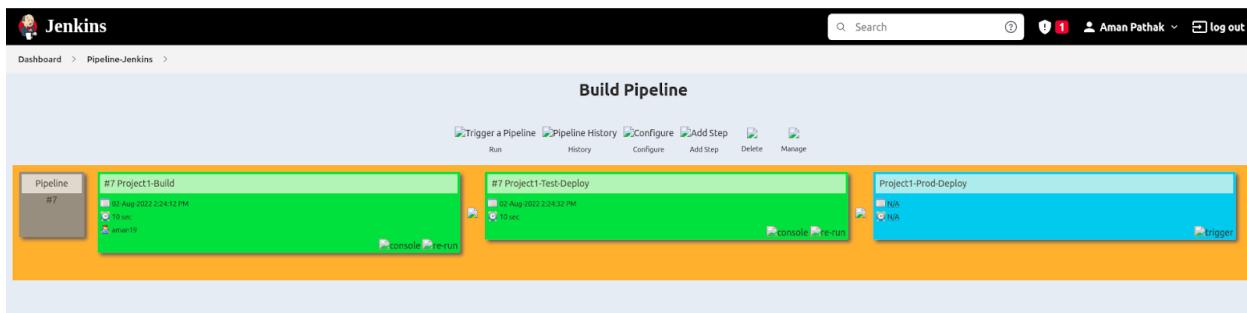
Name

A text input field for naming the new view, containing the text "Pipeline-Jenkins".

4. Now, all you have to do is “Select the Initial Job” and which will be your Parent job.

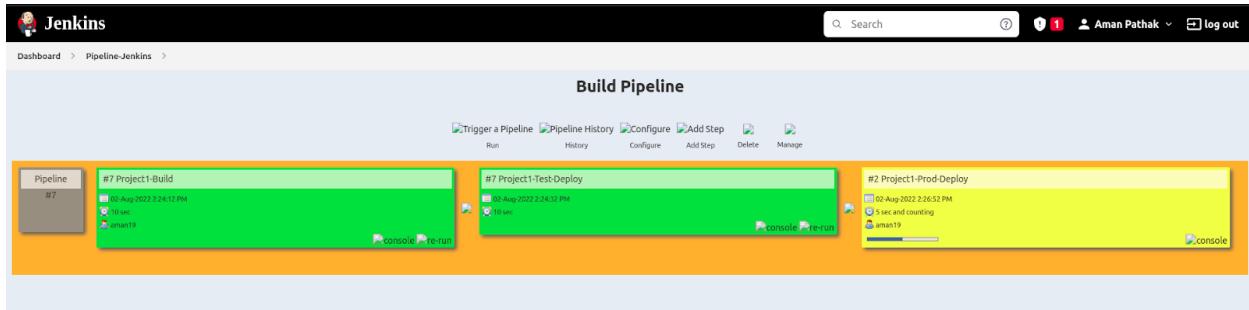


5. Now, when you run the Pipeline then you will face like this.

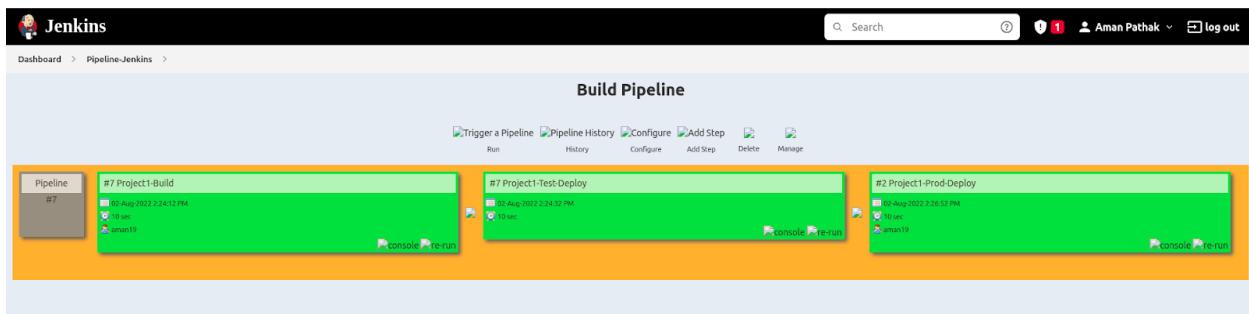


The last job will not run automatically as it is built with manual intervention.

6. So, to run that build too. You have to click on the trigger or run that build(on right bottom).



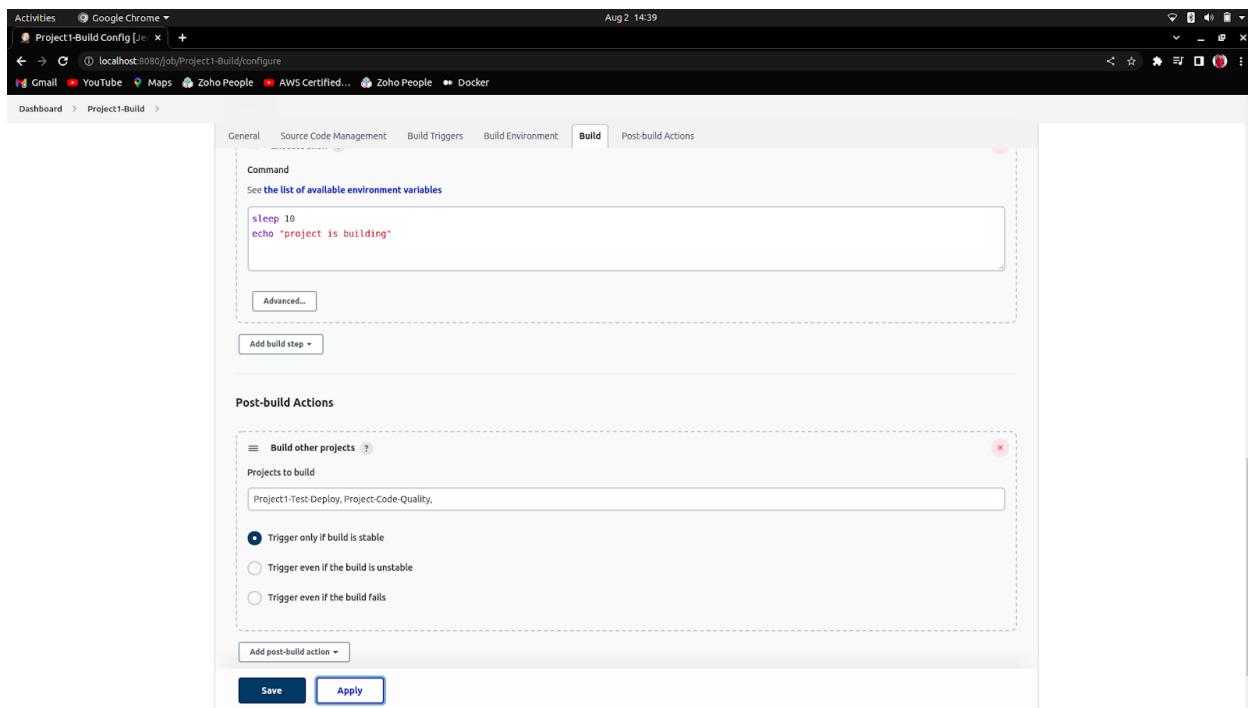
## 7. The Final Output:





# Run Two Job Parallel in Jenkins Pipeline

1. Create three Job, first will be GrandParent, the second will be Parent job and third will be Child job and assign the Post-build actions in Grand Parent Job for Parent job and in Parent Job for Child Job as per below:



The screenshot shows the Jenkins configuration interface for the 'Project-Code-Quality' job. The 'Build Environment' tab is active. Under the 'Build' section, there is a button labeled 'Add build step'. In the 'Post-build Actions' section, a 'Build other projects' dialog is open, showing 'Project1-Prod-Deploy' listed under 'Projects to build'. There are three radio button options: 'Trigger only if build is stable' (selected), 'Trigger even if the build is unstable', and 'Trigger even if the build fails'. A 'Save' and 'Apply' button are at the bottom.

The screenshot shows the Jenkins configuration interface for the 'Project1-Test-Deploy' job. The 'Build' tab is active. Under the 'Command' section, the command 'sleep 10' followed by 'echo "deploying to testing environment"' is entered. Below this is an 'Advanced...' button. A 'Post-build Actions' section is present, identical to the one in the first screenshot, with a 'Build other projects' dialog showing 'Project1-Prod-Deploy' and the 'Trigger only if build is stable' option selected. A 'Save' and 'Apply' button are at the bottom.

The screenshot shows a build configuration interface. The 'Build Environment' tab is active. Under the 'Build' section, there is an 'Execute shell' step with the following command:

```
sleep 10
echo "Production Deployment"
```

6. Click on “+” to create the Build Pipeline.

S	W	Name	Last Success	Last Failure	Last Duration
✓	Cloud	Change Workspace Project	3 days 18 hr #2	3 days 18 hr #1	26 ms
✓	Job	child-job	3 days 17 hr #13	N/A	23 ms
✓	Job	demo-build-periodically	3 days 21 hr #49	N/A	25 ms
✓	Job	demo-first	3 days 21 hr #3	N/A	33 ms

7. Choose Type “Build Pipeline view” and assign the name of your Pipeline.

## New view

Name

Type

Build Pipeline View  
Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.

List View  
Shows items in a simple list format. You can choose which jobs are to be displayed in which view.

My View  
This view automatically displays all the jobs that the current user has an access to.

**Create**

8. Now, all you have to do is “Select the Initial Job” and which will be your Parent job.

The screenshot shows the Jenkins Pipeline Flow configuration page. At the top, there's a header bar with tabs like 'Activities', 'Edit View [Jenkins]', and a search bar. Below the header, the URL is 'localhost:8080/view/Pipeline-Jenkins/configure'. The main content area is titled 'Pipeline Flow' and contains several sections:

- Build Pipeline View Title:** A text input field containing 'Pipeline-Jenkins'.
- Layout:** A dropdown menu set to 'Based on upstream/downstream relationship'.
- Upstream / downstream config:** A section with a 'Select Initial Job' dropdown containing 'Project1-Build'.
- Trigger Options:** A section with a 'Build Cards' dropdown set to 'Standard build card'.
- Restrict triggers to most recent successful builds:** A radio button option with 'Yes' selected.

At the bottom right of the configuration form, there are 'OK' and 'Apply' buttons.

## 9. The Final Output:

The screenshot shows the Jenkins Build Pipeline interface. At the top, there's a navigation bar with links for Dashboard, Pipeline-Jenkins, Trigger a Pipeline, Pipeline History, Configure, Add Step, Delete, and Manage. The main area is titled "Build Pipeline" and displays five build jobs arranged horizontally:

- #10 Project1-Build: Status is green, last run at 09-Aug-2022 2:41:12 PM, duration 16 ms.
- #1 Project-Code-Quality: Status is green, last run at 09-Aug-2022 2:41:12 PM, duration 16 ms.
- #5 Project1-Prod-Deploy: Status is green, last run at 09-Aug-2022 2:41:12 PM, duration 16 ms.
- #10 Project1-Test-Deploy: Status is green, last run at 09-Aug-2022 2:41:32 PM, duration 29 ms.
- #6 Project1-Prod-Deploy: Status is green, last run at 09-Aug-2022 2:41:52 PM, duration 21 ms.

Each job card includes "console" and "re-run" buttons.

# Deploy WAR to Tomcat Server through Jenkins (Automation)

Reference:

- [Web Link](#)
- [Video Link](#)

1. First of all, Configure the Tomcat Manager by following the below link:

[Tomcat Configuration with Manager](#)

[Apache Tomcat 9.0.65 Download Link](#)

```
<role rolename="admin-gui"/>
<role rolename="manager-script"/>
<role rolename="manager-gui"/>
<user username="aman" password="mypassword" roles="admin-gui,manager-gui,manager-script"/>
```

The Output should be like this:

Home Documentation Configuration Examples Wiki Mailing Lists Find Help

**Apache Tomcat/9.0.11**

If you're seeing this, you've successfully installed Tomcat. Congratulations!

TM

Recommended Reading:

[Security Considerations HOW-TO](#)

[Manager Application HOW-TO](#)

[Clustering/Session Replication HOW-TO](#)

Server Status

Manager App

Host Manager

**Developer Quick Start**

[Tomcat Setup](#) [Realms & AAA](#) [Examples](#) [Servlet Specifications](#)

[First Web Application](#) [JDBC DataSources](#) [Tomcat Versions](#)

2. Now, You have to Create Four Jobs:
  - a. **HelloWorld-Test:** This Job will perform the testing of the Project which is a Spring Boot-based Project.
  - b. **Hello-World-Build:** This Job will perform the building of the Spring Boot Project which will be .war file as in the result.
  - c. **HelloWorld-Deploy-Test:** In this Job, the deployment will be performed as a testing deployment to check whether the Job is successful or not.
  - d. **HelloWorld-Deploy-Prod:** This job will perform the same task that was performed by the last job (HelloWorld-Deploy-Test) but it will be on the Production level where the inclusion is present of human intervention or manual clicks.

Now, All the jobs have their configuration, which is given below sequentially:

- a. **HelloWorld-Test**

The screenshot shows the Jenkins configuration interface for the 'HelloWorld-Test' job. The 'Source Code Management' tab is selected. Under the 'Source Code Management' section, 'Git' is chosen as the provider. A repository URL is set to `https://github.com/AmanPathak-dev/SpringBoot.Jenkins.git`. The 'Branches to build' field contains the pattern `+/master`. At the bottom, there are 'Save' and 'Apply' buttons.

The screenshot shows the Jenkins configuration interface for the 'HelloWorld-Test' job. The 'Build Triggers' tab is selected. Under 'Build Triggers', the checkbox for 'Build whenever a SNAPSHOT dependency is built' is checked. Other options like 'Schedule build when some upstream has no successful builds', 'Trigger builds remotely', 'Build after other projects are built', 'Build periodically', 'GitHub hook trigger for GITScm polling', and 'Poll SCM' are unchecked. Below the triggers, the 'Build Environment' section includes options like 'Delete workspace before build starts', 'Use secret text(s) or file(s)', 'Add timestamps to the Console Output', 'Inspect build log for published Gradle build scans', 'Terminate a build if it's stuck', and 'With Ant'. The 'Pre Steps' section contains an 'Add pre-build step' button. At the bottom, there are 'Save' and 'Apply' buttons.

Activities Google Chrome ▾ Aug 3 18:13

Document shared with you Jenkins Tomcat Deploy... SpringBoot-Jenkins/Hell... Connect to instance | EC... How to Install Tomcat 9... HelloWorld-Test Config +

localhost:8080/job/HelloWorld-Test/configure Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > HelloWorld-Test >

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

**Build**

Root POM ? pom.xml

Goals and options ? test

Advanced...

**Post Steps**

Run only if build succeeds  
 Run only if build succeeds or is unstable  
 Run regardless of build result

Should the post-build steps run only for successful builds, etc.

Add post-build step ▾

**Build Settings**

**Post Steps**

**Build Settings**

E-mail Notification

**Post-build Actions**

Build other projects ? Projects to build Hello-World-Build

Trigger only if build is stable  
 Trigger even if the build is unstable  
 Trigger even if the build fails

Add post-build action ▾

Save Apply

Activities Google Chrome ▾ Aug 3 18:13

Document shared with you Jenkins Tomcat Deploy... SpringBoot-Jenkins/Hell... Connect to instance | EC... How to Install Tomcat 9... HelloWorld-Test Config +

localhost:8080/job/HelloWorld-Test/configure Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > HelloWorld-Test >

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

Add post-build step ▾

**Build Settings**

E-mail Notification

**Post-build Actions**

Build other projects ? Projects to build Hello-World-Build

Trigger only if build is stable  
 Trigger even if the build is unstable  
 Trigger even if the build fails

Add post-build action ▾

Save Apply

REST API Jenkins 2.346.2

## b. Hello-World-Build:

The screenshot shows the Jenkins General configuration page for the 'Hello-World-Build' job. The 'General' tab is selected. Under 'Permission to Copy Artifact', the checkbox 'Permission to Copy Artifact' is checked, and 'HelloWorld-\*' is listed under 'Projects to allow copy artifacts'. Other checkboxes for 'This project is parameterised', 'Throttle builds', 'Disable this project', and 'Execute concurrent builds if necessary' are unchecked. A 'Save' and 'Apply' button are at the bottom.

The screenshot shows the Jenkins Source Code Management configuration page for the 'Hello-World-Build' job. The 'Source Code Management' tab is selected. Under 'Repositories', the 'Git' option is selected, and the 'Repository URL' is set to 'https://github.com/AmanPathak-dev/SpringBoot-Jenkins.git'. The 'Credentials' dropdown is set to '- none -'. A 'Save' and 'Apply' button are at the bottom.

Activities Google Chrome ▾ Aug 3 18:15

Document shared with you Jenkins Tomcat Deploy... SpringBoot-Jenkins/Hell... Connect to instance | EC... How to Install Tomcat 9... Hello-World-Build Config

localhost:8080/job>Hello-World-Build/configure Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > Hello-World-Build >

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

**Build**

Root POM ? pom.xml

Goals and options ? install

Advanced...

**Post Steps**

Run only if build succeeds  
 Run only if build succeeds or is unstable  
 Run regardless of build result

Should the post-build steps run only for successful builds, etc.

Add post-build step ▾

**Build Settings**

E-mail Notification

Save Apply

Activities Google Chrome ▾ Aug 3 18:15

Document shared with you Jenkins Tomcat Deploy... SpringBoot-Jenkins/Hell... Connect to instance | EC... How to Install Tomcat 9... Hello-World-Build Config

localhost:8080/job>Hello-World-Build/configure Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > Hello-World-Build >

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

**Post-build Actions**

Archive the artifacts ?  
Files to archive ? \*\*/\*.war  
Advanced...

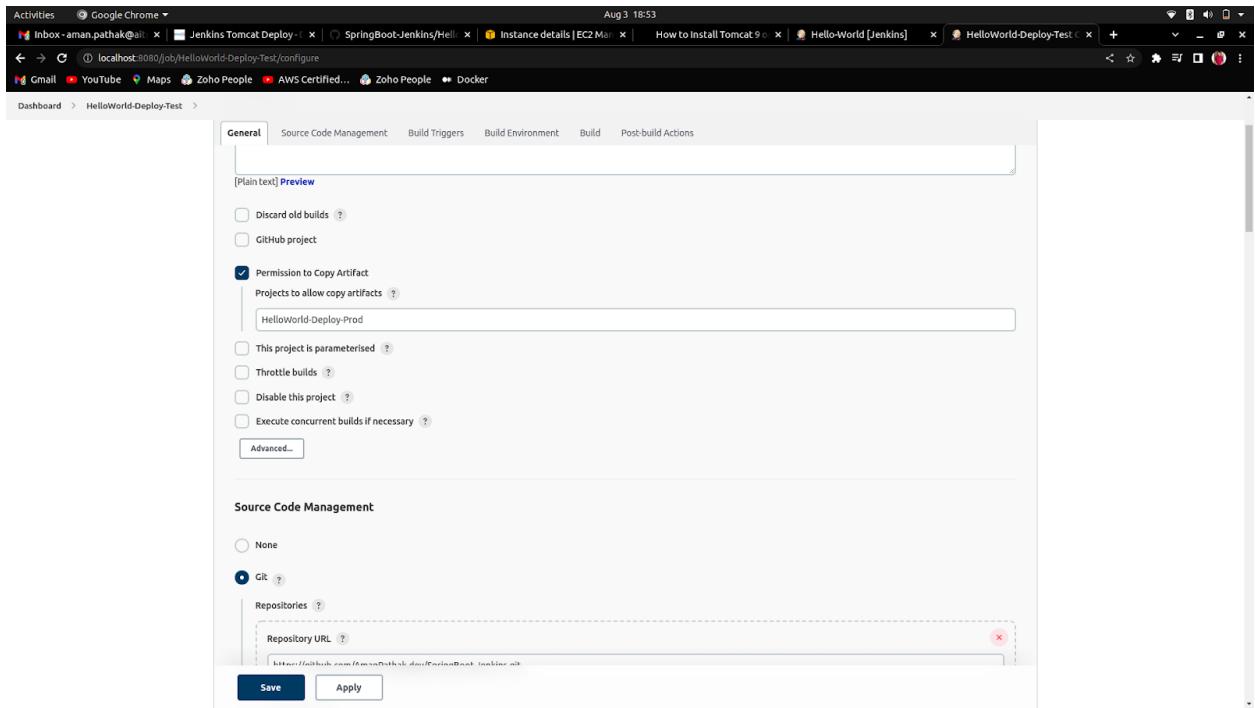
Build other projects ?  
Projects to build HelloWorld-Deploy-Test  
Trigger only if build is stable (checked)  
Trigger even if the build is unstable  
Trigger even if the build fails

Add post-build action ▾

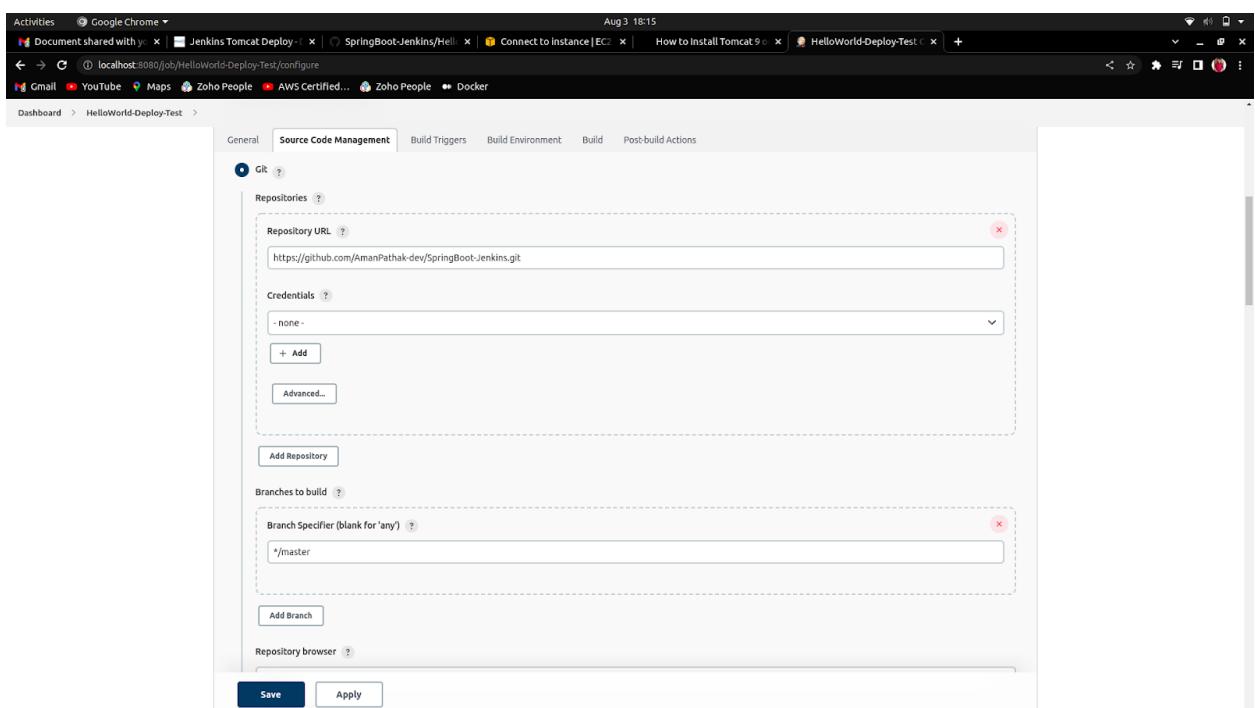
Save Apply

REST API Jenkins 2.346.2

## c. HelloWorld-Deploy-Test



The screenshot shows the Jenkins General configuration page for the 'HelloWorld-Deploy-Test' job. The 'General' tab is selected. Under 'Source Code Management', the 'Git' option is chosen, and the 'Repository URL' field contains the value `https://github.com/AmanPathak-dev/SpringBoot-Jenkins.git`. The 'Credentials' dropdown is set to '-none-'.



The screenshot shows the Jenkins Source Code Management configuration page for the 'HelloWorld-Deploy-Test' job. The 'Source Code Management' tab is selected. Under 'Repositories', the 'Repository URL' field contains the value `https://github.com/AmanPathak-dev/SpringBoot-Jenkins.git`. The 'Branch Specifier (blank for 'any')' field contains the value `*/*master`.

The screenshot shows the Jenkins configuration interface for a job named "HelloWorld-Deploy-Test".

**Build Triggers:**

- GitHub hook trigger for GITScm polling
- Poll SCM
  - Schedule: H/2 \* \* \* \*
  - Note: Would last have run at Wednesday, 3 August, 2022 at 6:14:49 PM India Standard Time; would next run at Wednesday, 3 August, 2022 at 6:16:49 PM India Standard Time.
  - Ignore post-commit hooks

**Build Environment:**

- Delete workspace before build starts
  -
- Use secret text(s) or file(s)
- Add timestamps to the Console Output
- Inspect build log for published Gradle build scans
- Terminate a build if it's stuck
- With Ant

**Build:**

Copy artifacts from another project

**Build**

≡ Copy artifacts from another project

Project name ?  
Hello-World-Build  
⚠ Artifacts will be copied from all modules of this Maven project; click the help icon to learn about selecting a particular module.

Which build ?  
Latest successful build

Stable build only

Activities Google Chrome ▾ Aug 3 18:16

Document shared with you | Jenkins Tomcat Deploy-Test | SpringBoot-Jenkins/Hell | Connect to instance | EC2 | How to install Tomcat 9 | HelloWorld-Deploy-Test | +

localhost:8080/job/HelloWorld-Deploy-Test/configure

Gmail YouTube Maps Zoho People AWS Certified... Zoho People Docker

Dashboard > HelloWorld-Deploy-Test >

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

**Post-build Actions**

≡ Archive the artifacts ?  
Files to archive ?  
\*\*/\*.war  
 Advanced...

≡ Deploy war/ear to a container  
WAR/EAR files ?  
\*\*/\*.war  
Context path ?  
/app

Containers

≡ Tomcat 9.x Remote  
Credentials  
aman\*\*\*\*\*  
 + Add

**Save** **Apply**

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

aman/\*\*\*\*\*

+ Add

Tomcat URL ?  
http://54.89.215.232:8080/

Advanced...

Add Container ▾

Deploy on failure

≡ Build other projects (manual step) ? X

Downstream Project Names  
HelloWorld-Deploy-Prod

Add Parameters ▾

Add post-build action ▾

**Save**   **Apply**

This screenshot shows the 'Post-build Actions' tab of a Jenkins job configuration. The 'Tomcat URL' field contains 'http://54.89.215.232:8080/'. The 'Downstream Project Names' field contains 'HelloWorld-Deploy-Prod'. There is a checked checkbox for 'Deploy on failure'. A manual step named 'Build other projects (manual step)' is listed, with a red 'X' icon next to it. At the bottom are 'Save' and 'Apply' buttons.

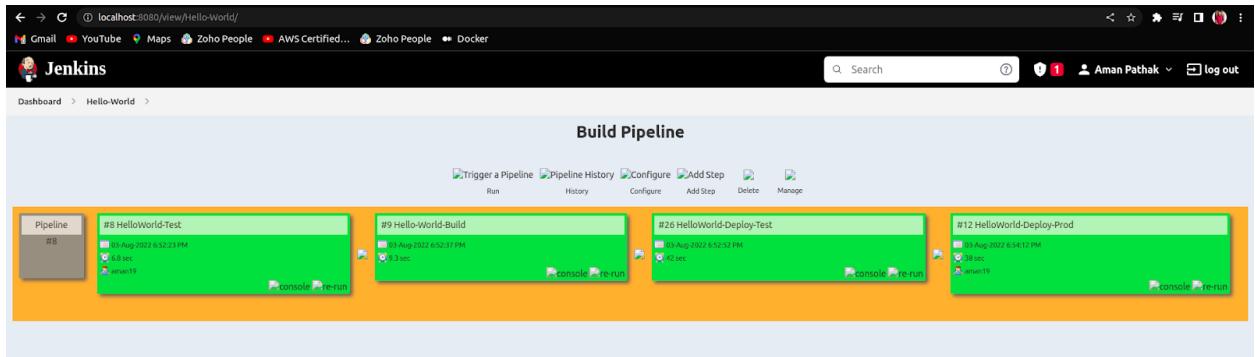
# d. HelloWorld-Deploy-Prod

The screenshot shows the Jenkins job configuration page for 'HelloWorld-Deploy-Prod'. The 'Source Code Management' tab is selected, showing options for 'None' or 'Git'. The 'Build Triggers' section contains several checkboxes for remote triggers, periodic builds, GitHub hooks, and SCM polling. The 'Build Environment' section includes checkboxes for workspace cleanup, secret text usage, timestamps, log inspection, and Ant termination. At the bottom are 'Save' and 'Apply' buttons.

The screenshot shows the Jenkins job configuration page for 'HelloWorld-Deploy-Prod'. The 'Build' tab is selected, displaying a configuration for copying artifacts from another project named 'HelloWorld-Deploy-Test'. It specifies the latest successful build and includes options for filtering artifacts by type and target directory. At the bottom are 'Save' and 'Apply' buttons.

The screenshot shows a Jenkins job configuration page for 'HelloWorld-Deploy-Prod'. The 'Post-build Actions' tab is selected. Under the 'Deploy war/ear to a container' section, the 'WAR/EAR files' field contains '\*\*/\*.war'. The 'Context path' field is set to '/app'. In the 'Containers' section, there is one entry for 'Tomcat 9.x Remote'. It includes a dropdown for 'Credentials' containing 'aman\*\*\*\*\*', a '+ Add' button, and a 'Tomcat URL' field with 'http://3.83.235.174:8080/'. There is also an 'Advanced...' button and an 'Add Container' button. A checkbox for 'Deploy on failure' is present but unchecked. At the bottom are 'Save' and 'Apply' buttons.

# The Output



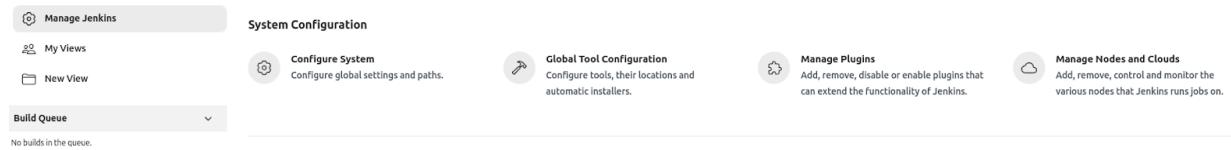
This is All About Spring Boot! This is The Second Version of The Spring Boot



This is All About Spring Boot! This is The Second Version of The Spring Boot

# Creating Jenkins Slave

1. First of all, go into Manage Jenkins -> Manage Nodes and Clouds.



The screenshot shows the Jenkins Manage Jenkins dashboard. At the top, there is a navigation bar with links for 'Manage Jenkins', 'My Views', 'New View', 'Build Queue' (which shows 'No builds in the queue.'), 'System Configuration' (with 'Configure System' and 'Global Tool Configuration' options), 'Manage Plugins' (with 'Manage Plugins' and 'Cloud' icons), and 'Manage Nodes and Clouds' (with a detailed description). The 'Manage Nodes and Clouds' link is highlighted.

2. To create the new Slave, Click on the New node.

↑ Back to Dashboard

⚙ Manage Jenkins

+ New Node

☁ Configure Clouds

⚙ Node Monitoring

Build Queue

No builds in the queue.

Build Executor Status

💻 Built-In Node

1 Idle

2 Idle

3. Give the name to your slave and check the radio button of Permanent Agent then, click on the Create button.

#### New node

Node name  
Linux1

Type

Permanent Agent

Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.

4. Before going to the subsequent Configurations of your Jenkins slave,

Follow some steps:

- a. Create an EC2 Instance with Inbound rule 22(SSH) at least.

- b. SSH into your instance.

- c. Install OpenJDK by the command:

*sudo apt install openjdk-8-jre-headless -y*

- d. Create a directory named jenkins inside the /var directory.

- e. Change the owner of that directory by the given command:

*sudo chown ubuntu:ubuntu jenkins/*

- f. Now, change the password of your EC2 Instance by the following commands:

*sudo passwd ubuntu*

*sudo vim sshd\_config*

*sudo systemctl restart jenkins*

***For more reference, follow the link below:***

**[Change EC2 Instance Password](#)**

5. Enter the number of executors as per your choice, then enter the Remote root directory which will be /var/jenkins then, the launch type must be “Launch agents via SSH” Enter the Host Name which will be the Public IP of your instance, then add the credentials of your EC2 Instance (username=ubuntu and password= aman@123) and In the last Host, the Key verification Strategy must be “Non-verifying Verification Strategy”.

The screenshot shows the Jenkins Node configuration interface. The 'Configure' tab is active. The configuration details are as follows:

- Name:** Linux
- Description:** (empty)
- Number of executors:** 2
- Remote root directory:** /var/jenkins
- Labels:** linux2
- Usage:** Use this node as much as possible
- Launch method:** Launch agents via SSH
- Host:** 54.82.113.239

Host [?](#)  
54.82.113.239

Credentials [?](#)  
ubuntu/\*\*\*\*\*  
[+ Add](#)

Host Key Verification Strategy [?](#)  
Non verifying Verification Strategy

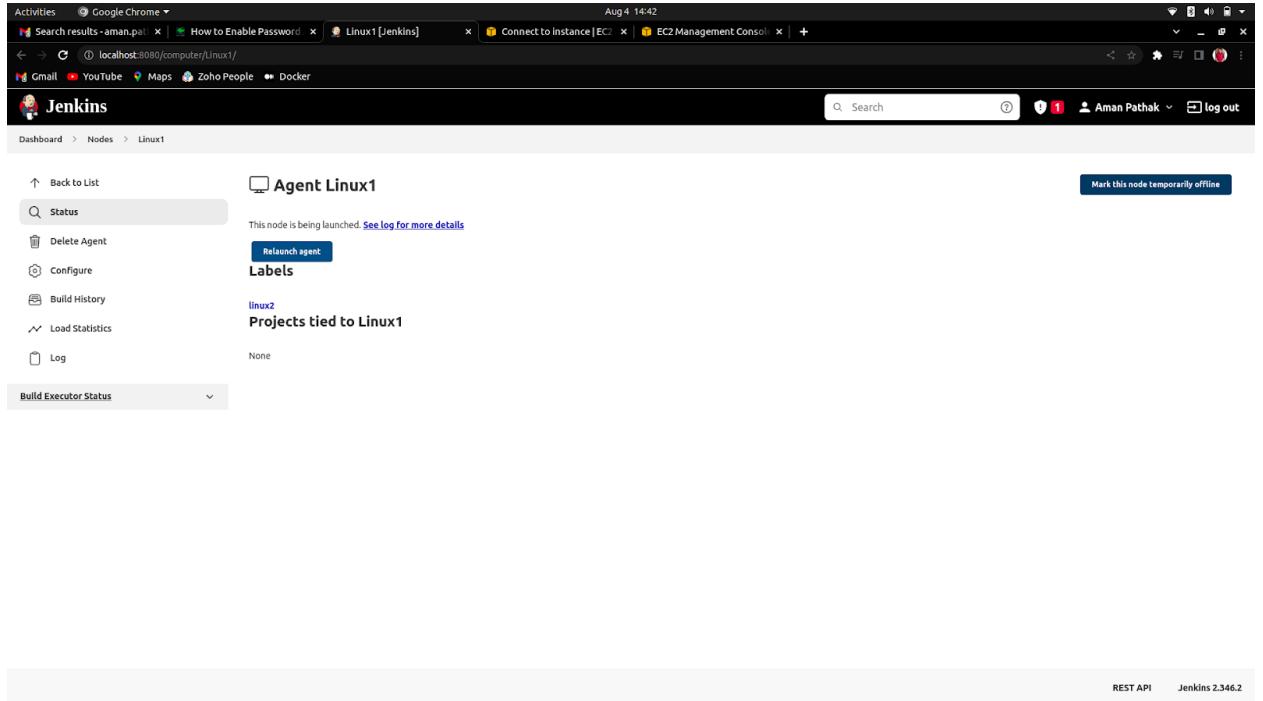
Availability [?](#)  
Keep this agent online as much as possible

**Node Properties**

Disable deferred wipeout on this node [?](#)  
 Environment variables  
 Tool Locations

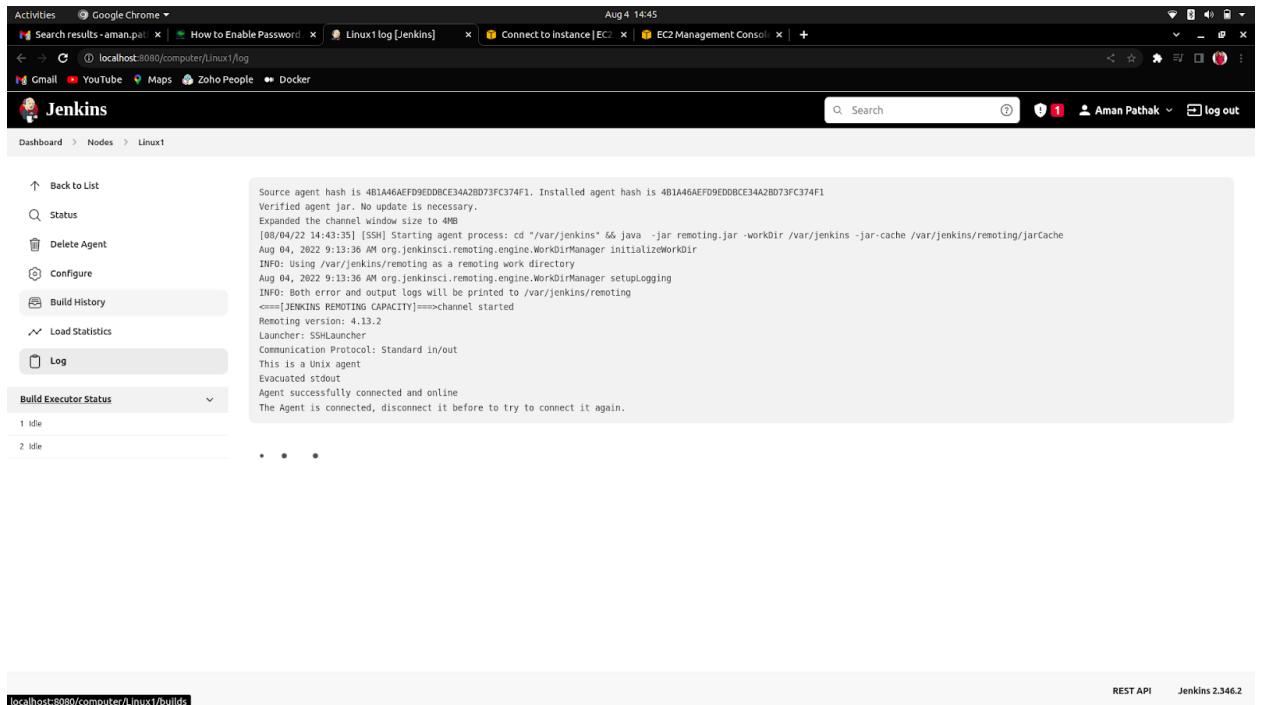
[Save](#)

## 6. Now go to that Slave and click on Relaunch agent.



The screenshot shows the Jenkins interface for managing a slave node named 'Agent Linux1'. On the left, there's a sidebar with options like 'Status', 'Delete Agent', 'Configure', 'Build History', 'Load Statistics', and 'Log'. The main area displays the node's status as 'This node is being launched.' Below this, there's a section titled 'Projects tied to Linux1' with a 'Labels' field set to 'linux2'. At the bottom of the main content area, there's a 'Build Executor Status' dropdown. In the top right corner, there's a button labeled 'Relaunch agent' which is currently highlighted with a blue border. The top navigation bar shows several tabs and the current URL as 'localhost:8080/computer/Linux1'.

## 7. As the Slave is connected.



This screenshot shows the same Jenkins node configuration page for 'Agent Linux1', but now the 'Log' tab is selected. The log output window is filled with a detailed log of the agent's startup process. It includes messages like 'Source agent hash is 4B1A46AEFD9EDDBCE34A2B073FC374F1. Installed agent hash is 4B1A46AEFD9EDDBCE34A2B073FC374F1', 'Verified agent jar. No update is necessary.', and 'Starting agent process: cd "/var/jenkins" && java -jar remoting.jar -workDir /var/jenkins -jar-cache /var/jenkins/remoting/jarCache'. The log also shows the agent connecting to the master: 'Both error and output logs will be printed to /var/jenkins/remoting', 'The Agent is connected, disconnect it before to try to connect it again.', and 'Agent successfully connected and online'. The rest of the interface remains consistent with the previous screenshot, including the sidebar and the top navigation bar.

## 8. You can check on the left side named Linux1.

↑ Back to Dashboard

Manage Jenkins

+ New Node

Configure Clouds

Node Monitoring

Build Queue

No builds in the queue.

Build Executor Status

Built-In Node

1 Idle

2 Idle

Linux

1 Idle

2 Idle

Linux1

1 Idle

2 Idle

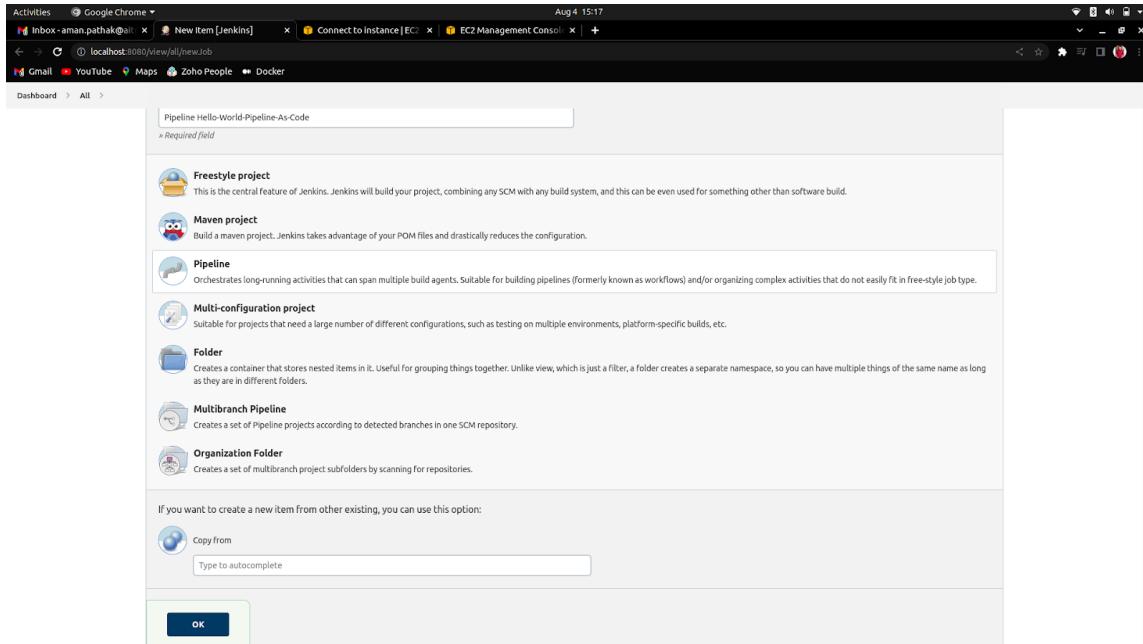
Manage nodes and clouds

Refresh status

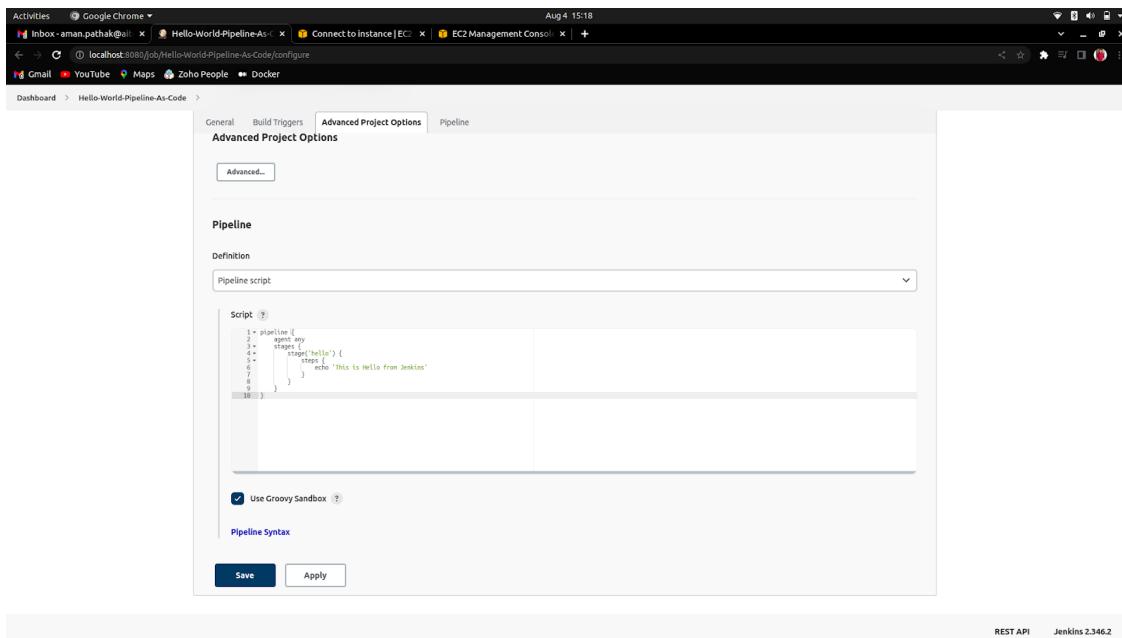
S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	413.20 GB	1.31 GB	413.20 GB	0ms ⓘ
	Linux	Linux (amd64)	⌚ 6.2 sec behind	5.81 GB	⌚ 0 B	5.81 GB	5190ms ⓘ
	Linux1	Linux (amd64)	⌚ 6.2 sec ahead	5.81 GB	⌚ 0 B	5.81 GB	7238ms ⓘ
		last checked	3 min 33 sec	3 min 23 sec	3 min 23 sec	3 min 21 sec	3 min 29 sec

# Create Jenkins Pipeline as a Code

## 1. Create a Job that will be a Pipeline.



## 2. Write the Declarative Pipeline to run the job of Hello World.



3. Now, when you build the job you'll see look like this:

The screenshot shows the Jenkins dashboard for the 'Hello-World-Pipeline-As-Code' project. The top navigation bar includes links for 'Inbox', 'Hello-World-Pipeline-As-Code', 'Connect to instance | EC2', and 'EC2 Management Console'. The main content area displays the 'Stage View' for the 'hello' stage. The stage has an average stage time of 190ms. A tooltip indicates 'Aug 04 15:16 No Changes'. Below the stage view, there's a 'Permalinks' section with a link to the last build. At the bottom right, it says 'REST API Jenkins 2.346.2'.

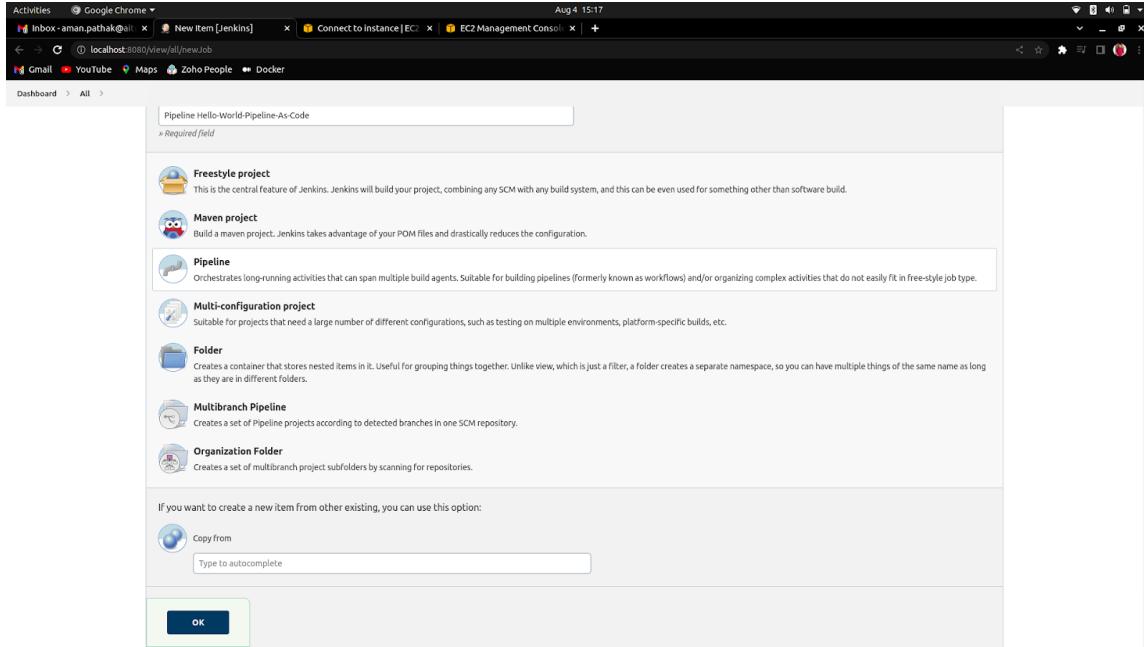
4. To see the output hover on the green box and click on the logs.

## Stage View

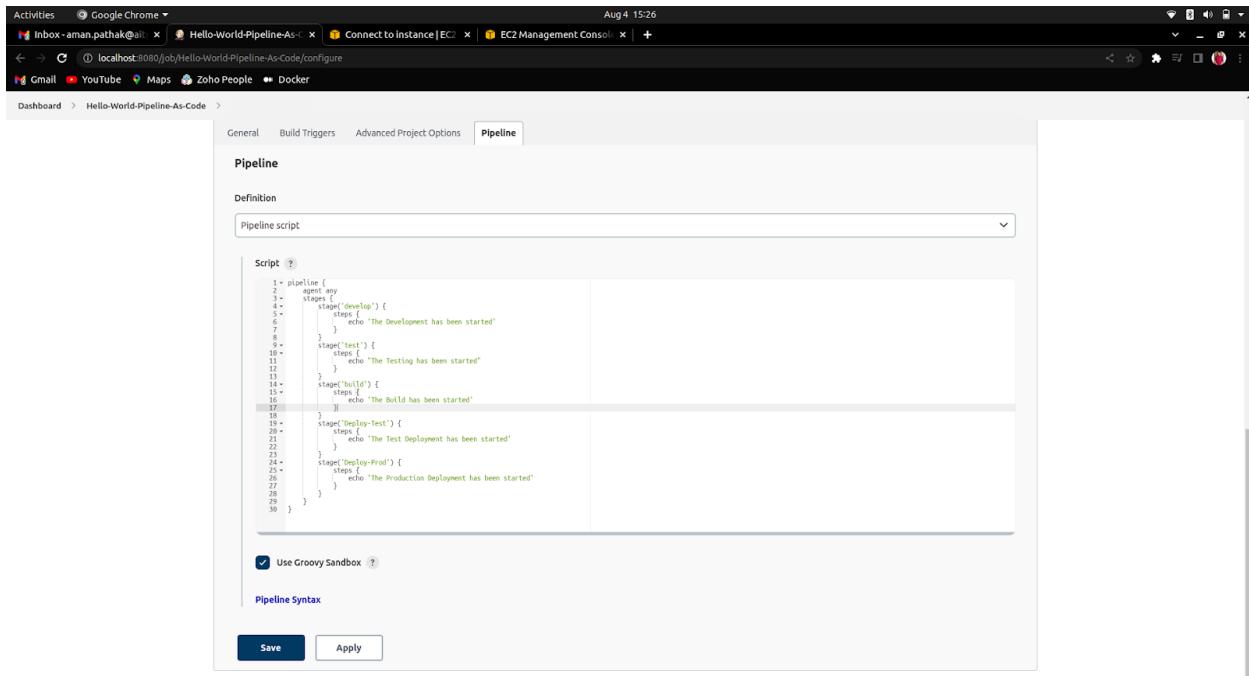


# Multi-Staging Pipeline as a Code

## 1. Create a Job that will be a Pipeline.



## 2. Write the Declarative Pipeline.



### 3. The Output:

The screenshot shows the Jenkins dashboard for the pipeline "Hello-World-Pipeline-As-Code". The pipeline consists of five stages: develop, test, build, Deploy-Test, and Deploy-Prod. The "develop" stage has an average stage time of 96ms. The "test" stage has an average stage time of 80ms. The "build" stage has an average stage time of 80ms. The "Deploy-Test" stage has an average stage time of 81ms. The "Deploy-Prod" stage has an average stage time of 79ms. There are two builds listed: #3 (Aug 04, 15:24) and #2 (Aug 04, 15:23). Both builds show "No Changes". The "Permalinks" section lists the following links:

- Last build (#3), 2 min 3 sec ago
- Last stable build (#3), 2 min 3 sec ago
- Last successful build (#3), 2 min 3 sec ago
- Last failed build (#2), 3 min 2 sec ago
- Last unsuccessful build (#2), 3 min 2 sec ago
- Last completed build (#3), 2 min 3 sec ago

# How to run Commands in Pipeline as a Code

The screenshot shows the Jenkins interface for creating a new item. The title bar indicates it's on 'localhost:8080/view/all/newJob'. The main area has a heading 'Enter an item name' with a text input field containing 'Command-As-Pipeline'. Below this, there's a list of project types with descriptions:

- 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.
- Multibranch Pipeline**: Creates a set of Pipeline projects according to detected branches in one SCM repository.
- Organization Folder**: Creates a set of multibranch project subfolders by scanning for repositories.

At the bottom of the dialog, there's a note: "If you want to create a new item from other existing, you can use this option:" followed by an 'OK' button.

The screenshot shows the Jenkins configuration page for the 'Command-As-Pipeline' job. The title bar shows the URL 'localhost:8080/job/Command-As-Pipeline/configure'. The top navigation tabs are General, Build Triggers, Advanced Project Options (which is selected), and Pipeline. The Pipeline tab is currently active.

The 'Pipeline' section contains the 'Definition' dropdown set to 'Pipeline script'. Below it is a code editor window containing Groovy pipeline script:

```
1 pipeline {
2     agent any
3     stages {
4         stage('Some Commands') {
5             steps {
6                 echo 'date'
7             }
8         }
9     }
10 }
```

Below the code editor, there's a checkbox 'Use Groovy Sandbox' and a link 'Pipeline Syntax'. At the bottom are 'Save' and 'Apply' buttons.

The screenshot shows the Jenkins stage log for the 'Some Commands' stage. The title bar says 'Stage Logs (Some Commands)'. The log output is as follows:

```
+ date
Thu Aug 4 10:07:29 UTC 2022
```

# How to run multiple commands in Pipeline as a code

The screenshot shows two screenshots of the Jenkins web interface. The top screenshot is a 'New Item' creation page where 'Command-As-Pipeline' is selected as a Freestyle project. The bottom screenshot shows the 'Advanced Project Options' tab for the 'Command-As-Pipeline' job, specifically the 'Pipeline' section. It displays a Groovy script for defining a pipeline with stages and steps, and includes a 'Use Groovy Sandbox' checkbox.

```
1 < pipeline {  
2     agent any  
3     stages {  
4         stage('Some Commands') {  
5             step('sh',...)  
6             step('pwd',...)  
7             step('date',...)  
8             step('cal 2022',...)  
9         }  
10    }  
11 }  
12  
13  
14  
15  
16  
17  
18 }
```

## The Output:

```

Activities Google Chrome ▾ Aug 4 15:51
Inbox -aman.pathak@ali | Command-As-Pipeline #9 | Connect to Instance | EC2 Management Console | +
localhost:8080/job/Command-As-Pipeline/b/console
Gmail YouTube Maps Zoho People Docker

Dashboard > Command-As-Pipeline > #9
/var/jenkins/workspace/Command-As-Pipeline
+ ls
+ date
Thu Aug 4 10:21:09 UTC 2022
+ cal 2022
2022
January February March
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
2 3 4 5 6 7 8 6 7 8 9 10 11 12 6 7 8 9 10 11 12
9 10 11 12 13 14 15 13 14 15 16 17 18 19 13 14 15 16 17 18 19
16 17 18 19 20 21 22 20 21 22 23 24 25 26 20 21 22 23 24 25 26
23 24 25 26 27 28 29 27 28 29 30 31

April May June
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
1 2 1 2 3 4 5 6 7 1 2 3 4
3 4 5 6 7 8 9 8 9 10 11 12 13 4 5 6 7 8 9 10
10 11 12 13 14 15 16 15 16 17 18 19 20 21 12 13 14 15 16 17 18
17 18 19 20 21 22 23 22 23 24 25 26 27 28 19 20 21 22 23 24 25
24 25 26 27 28 29 30 29 30 31 26 27 28 29 30

July August September
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
1 2 1 2 3 4 5 6 1 2 3
3 4 5 6 7 8 9 7 8 9 10 11 12 13 4 5 6 7 8 9 10
10 11 12 13 14 15 16 14 15 16 17 18 19 20 11 12 13 14 15 16 17
17 18 19 20 21 22 23 21 22 23 24 25 26 27 18 19 20 21 22 23 24
24 25 26 27 28 29 30 28 29 30 31 25 26 27 28 29 30 31

October November December
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
1 2 1 2 3 4 5 6 1 2 3
2 3 4 5 6 7 8 6 7 8 9 10 11 12 13 4 5 6 7 8 9 10
9 10 11 12 13 14 15 13 14 15 16 17 18 19 11 12 13 14 15 16 17
16 17 18 19 20 21 22 20 21 22 23 24 25 26 18 19 20 21 22 23 24
23 24 25 26 27 28 29 27 28 29 30 31 25 26 27 28 29 30 31

23 24 25 26 27 28 29 27 28 27 28 29 30 31
30 31

April May June
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
1 2 1 2 3 4 5 6 7 1 2 3 4
3 4 5 6 7 8 9 8 9 10 11 12 13 4 5 6 7 8 9 10
10 11 12 13 14 15 16 15 16 17 18 19 20 21 12 13 14 15 16 17 18
17 18 19 20 21 22 23 22 23 24 25 26 27 28 19 20 21 22 23 24 25
24 25 26 27 28 29 30 28 29 30 31 26 27 28 29 30

July August September
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
1 2 1 2 3 4 5 6 1 2 3
3 4 5 6 7 8 9 7 8 9 10 11 12 13 4 5 6 7 8 9 10
10 11 12 13 14 15 16 14 15 16 17 18 19 20 11 12 13 14 15 16 17
17 18 19 20 21 22 23 21 22 23 24 25 26 27 18 19 20 21 22 23 24
24 25 26 27 28 29 30 28 29 30 31 25 26 27 28 29 30 31

October November December
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
1 2 1 2 3 4 5 6 1 2 3
2 3 4 5 6 7 8 6 7 8 9 10 11 12 13 4 5 6 7 8 9 10
9 10 11 12 13 14 15 13 14 15 16 17 18 19 11 12 13 14 15 16 17
16 17 18 19 20 21 22 20 21 22 23 24 25 26 18 19 20 21 22 23 24
23 24 25 26 27 28 29 27 28 29 30 31 25 26 27 28 29 30 31

[Pipeline] ]
[Pipeline] // stage
[Pipeline] ]
[Pipeline] End of Pipeline
Finished: SUCCESS

```

REST API Jenkins 2.346.2

# How to set the Environment variable Globally in the Code Pipeline

Pipeline

Definition

Pipeline script

Script ?

```
1 pipeline {  
2     agent any  
3     environment {  
4         name = "jenkins_user"  
5     }  
6     stages {  
7         stage('Global Variable') {  
8             steps {  
9                 sh 'echo $name'  
10            }  
11        }  
12    }  
13}
```

Use Groovy Sandbox ?



Stage Logs (Global Variable)

Shell Script – echo \$name (self time 3s)

```
+ echo jenkins_user  
jenkins_user
```



# How to set the Environment variable Locally in Code Pipeline

Definition

Pipeline script

```
1 - pipeline {
2   agent any
3   environment {
4     name = "jenkins_user"
5   }
6   stages {
7     stage('Global Variable') {
8       steps {
9         sh 'echo $name'
10        sh 'echo $BUILD_ID'
11      }
12    }
13    stage('Local Variable') {
14      environment {
15        user_id = '34'
16      }
17      steps {
18        sh 'echo $user_id'
19      }
20    }
21    stage('Accessing Local Variable') {
22      steps {
23        sh 'echo $user_id'
24      }
25    }
26  }
27 }
```

Use Groovy Sandbox ?

## The Output

Activities Google... Aug 4 16:07

Inbox - aman.pathak@altis | Environment\_Variable-C... | Connect to instance | EC2 Management Console | +

localhost:8080/job/Environment\_Variable-Code-Pipeline/5/console

Gmail YouTube Maps Zoho People Docker

Dashboard > Environment\_Variable-Code-Pipeline > #5

</> Changes Started by user Aman Pathak

Console Output View as plain text

```
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Linux in /var/jenkins/workspace/Environment_Variable-Code-Pipeline
[Pipeline] {
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] {
[Pipeline] { (Global Variable)
[Pipeline] sh
+ echo jenkins_user
jenkins_user
[Pipeline] sh
+ echo $user_id
34
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] {
[Pipeline] { (Local Variable)
[Pipeline] withEnv
[Pipeline] {
[Pipeline] {
[Pipeline] sh
+ echo 34
34
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] {
[Pipeline] { (Accessing Local Variable)
[Pipeline] sh
+ echo
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

# How to take input from the users in the Code Pipeline

Pipeline

Definition

Pipeline script

Script ?

```
1 pipeline {
2     agent any
3     parameters {
4         string(name: 'username', defaultValue: 'Jenkins', description: 'Who is the Current User')
5         booleanParam(name: 'Graduated?', defaultValue: 'true', description: 'Are you Graduated')
6         choice(name: 'hobby', choices: ['Cricket','Chess','Football','Hockey'], description: 'Favourite Sport')
7     }
8     stages {
9         stage('String Value') {
10            steps {
11                sh 'echo $username'
12            }
13        }
14        stage('Boolean Value') {
15            steps {
16                sh 'echo $Graduated?'
17            }
18        }
19        stage('Choices Values') {
20            steps {
21                sh 'echo $hobby'
22            }
23        }
24    }
25 }
```

## The Output:

Activities Google Chrome ▾

Inbox - aman.pathak@alt... Parameterized-Value-Code-Pipeline Connect to Instance | EC2 Management Console

localhost:8080/job/Parameterized-Value-Code-Pipeline/6/console

Gmail YouTube Maps Zoho People Docker

Dashboard > Parameterized-Value-Code-Pipeline > #6

Back to Project Status Changes

Console Output View as plain text Edit Build Information Delete build #6 Parameters Restart from Stage Replay Pipeline Steps Workspaces Previous Build

Console Output

```
Started by user Aman Pathak
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/Parameterized-Value-Code-Pipeline
[Pipeline]
[Pipeline] stage
[Pipeline] {
[Pipeline] {
[Pipeline] {
[Pipeline] {
[Pipeline] sh
+ echo Groovy Script
Groovy Script
[Pipeline]
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] {
[Pipeline] {
[Pipeline] {
[Pipeline] {
[Pipeline] sh
+ echo Hockey
Hockey
[Pipeline]
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] {
[Pipeline] {
[Pipeline] sh
+ echo Hockey
Hockey
[Pipeline]
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] End of Pipeline
Finished: SUCCESS
```

REST API Jenkins 2.346.2

# Pipeline Script of the Last Job

```
pipeline {
    agent any
    parameters {
        string(name: 'username', defaultValue: 'Jenkins', description: 'Who is the Current User')
        booleanParam(name: 'Graduated?', defaultValue: 'true', description: 'Are you Graduated')
        choice(name: 'hobby', choices: ['Cricket','Chess','Football','Hockey'], description: 'Favourite Sport')
    }
    stages {
        stage('String Value') {
            input {
                message "Do we start?"
                ok "Yes, Let's do this"
            }
            steps {
                sh 'echo $username'
            }
        }
        stage('Boolean Value') {
            steps {
                sh 'echo $Graduated?'
            }
        }
        stage('Choices Values') {
            input {
                message "Want to Continue?"
                ok "Yeah! Definitely"
            }
            steps {
                sh 'echo $hobby'
            }
        }
    }
    post {
        always {
            echo 'The Jenkins!!!!'
        }
        success {
            echo 'The Final Acheivement is Here!!!'
        }
        failure {
            echo 'Seriously! We got the Failure from the other End!!!!'
        }
    }
}
```

**In conclusion,** the above notes provide a beginner-level introduction to Jenkins and its role in enabling Continuous Integration and Continuous Delivery (CI/CD) practices. By understanding the fundamental concepts and features of Jenkins, beginners can kickstart their journey towards automating their software development processes and achieving faster, more reliable project deliveries.

For those seeking to advance their knowledge and skills in Jenkins, there is a world of possibilities to explore. Jenkins offers a vast array of plugins and integrations with various tools, allowing advanced users to tailor their CI/CD pipelines to meet specific project requirements. By delving deeper into Jenkins' capabilities, experienced users can optimize their workflows, integrate advanced testing and deployment strategies, and harness the full potential of CI/CD for continuous improvement.

Moreover, Jenkins Pipelines offers an exciting avenue for advanced users to adopt a "pipeline-as-code" approach, enabling better version control, collaboration, and scalability. With declarative pipeline scripts, developers can encapsulate complex build and deployment logic, making it easier to manage and maintain their CI/CD workflows effectively.

While the notes provide a solid foundation for beginners, it is essential to continue exploring Jenkins and other related technologies to stay up-to-date with the latest advancements in CI/CD practices. Online tutorials, documentation, forums, and community resources are valuable sources to deepen one's expertise and stay ahead in the rapidly evolving landscape of DevOps and CI/CD.

Whether at the beginner or advanced level, embracing Jenkins as a pivotal tool in the software development lifecycle empowers teams to automate, collaborate, and deliver high-quality software efficiently. By leveraging Jenkins' capabilities and staying curious, developers can continually refine their CI/CD processes and drive innovation in the dynamic world of modern software development.

# Did you find it Useful?

Leave a **comment!**



ABD UR REHMAN NAWAZ

@ABD-UR-REHMAN-  
NAWAZ

REHMANWARRAICH8282@GMAIL.COM

FOLLOW FOR MORE

Like

Comment

Repost

