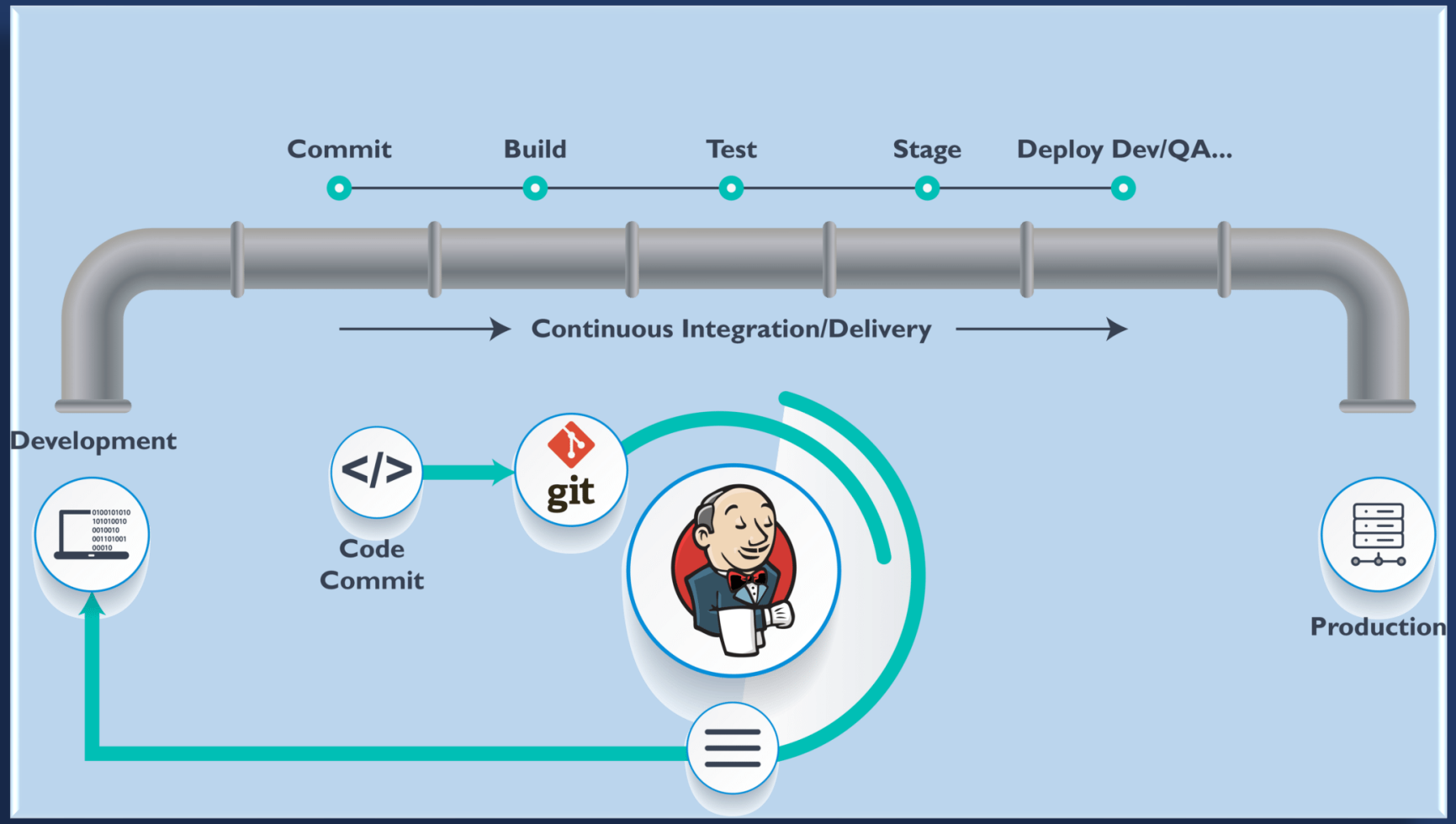
# Software Production Engineering

## SDLC, Jenkins and CI/CD Pipeline

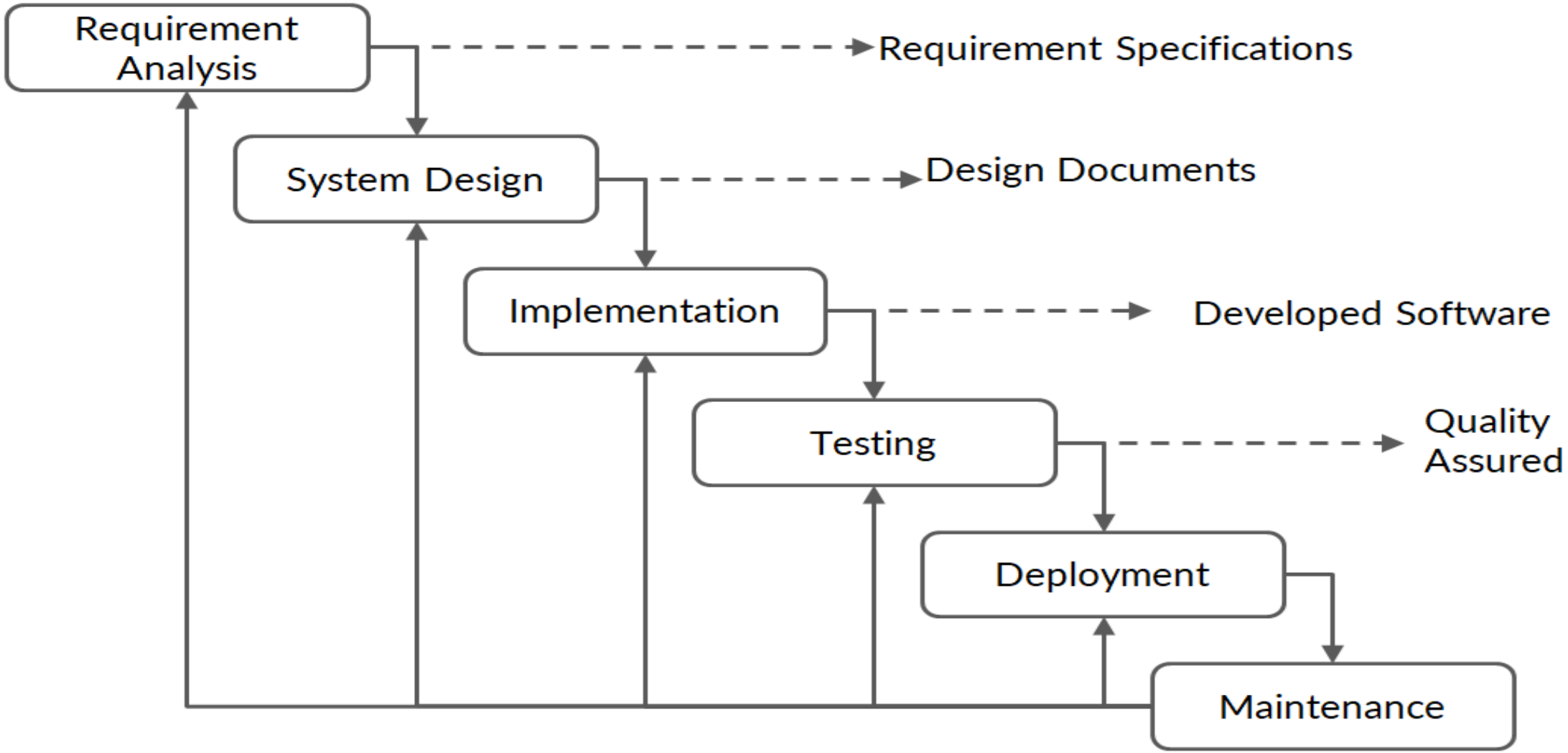


# Agenda

- Understanding the working of traditional SDLC approach
- Introduction to Jenkins as a CI tool
- Creating Jobs in Jenkins and execute internally as well as remotely
- Sending email notifications to update status
- Parameterised jobs and their need
- User Management - Configuring role-based strategy plugin
- Integrating Jenkins with Git via Poll SCM and web hooks
- CI/CD pipeline creation using Jenkins file
- Setting up master slave architecture in Jenkins

# Introduction to CI/CD

# Software Development Life Cycle



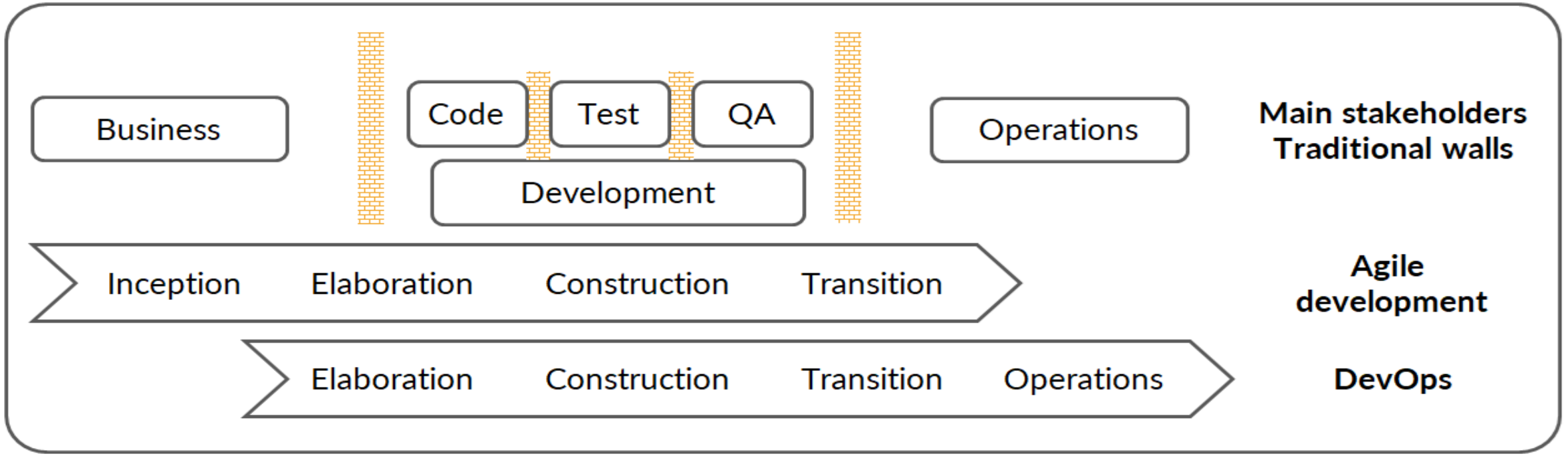
# Introduction to CI/CD

- In software engineering, **continuous integration** (**CI**) is defined as a process of integrating all developers' code to a shared repository in a version-control system frequently.
- The term CI was first proposed by Grady Booch in 1991.
- CI is supposed to be used in conjunction with automated build, test, and QA.
- A build server compiles the code and reports the results to the developers on a regular basis, or even after each commit.
- Test: Unit test in the developer's local environment before committing to the mainline helps avoid one developer's work-in-progress while breaking another developer's copy.

# Introduction to CI/CD

- Along with running the unit and integration checks, you can check the quality of your code and profile performance and format the documentation from the source code, thereby facilitating QA processes.
- This quality check application mainly aims to enhance software quality and reduce delivery time continuously.
- CI is connected closely with continuous delivery or continuous deployment, which is also called the CI/CD pipeline.
- ‘Continuous delivery’ ensures that the incremental feature of a software product checked in on the mainline is ready to deliver to the end user.
- ‘Continuous deployment’ makes the deployment process fully automated.

# Drawbacks of Traditional SDLC

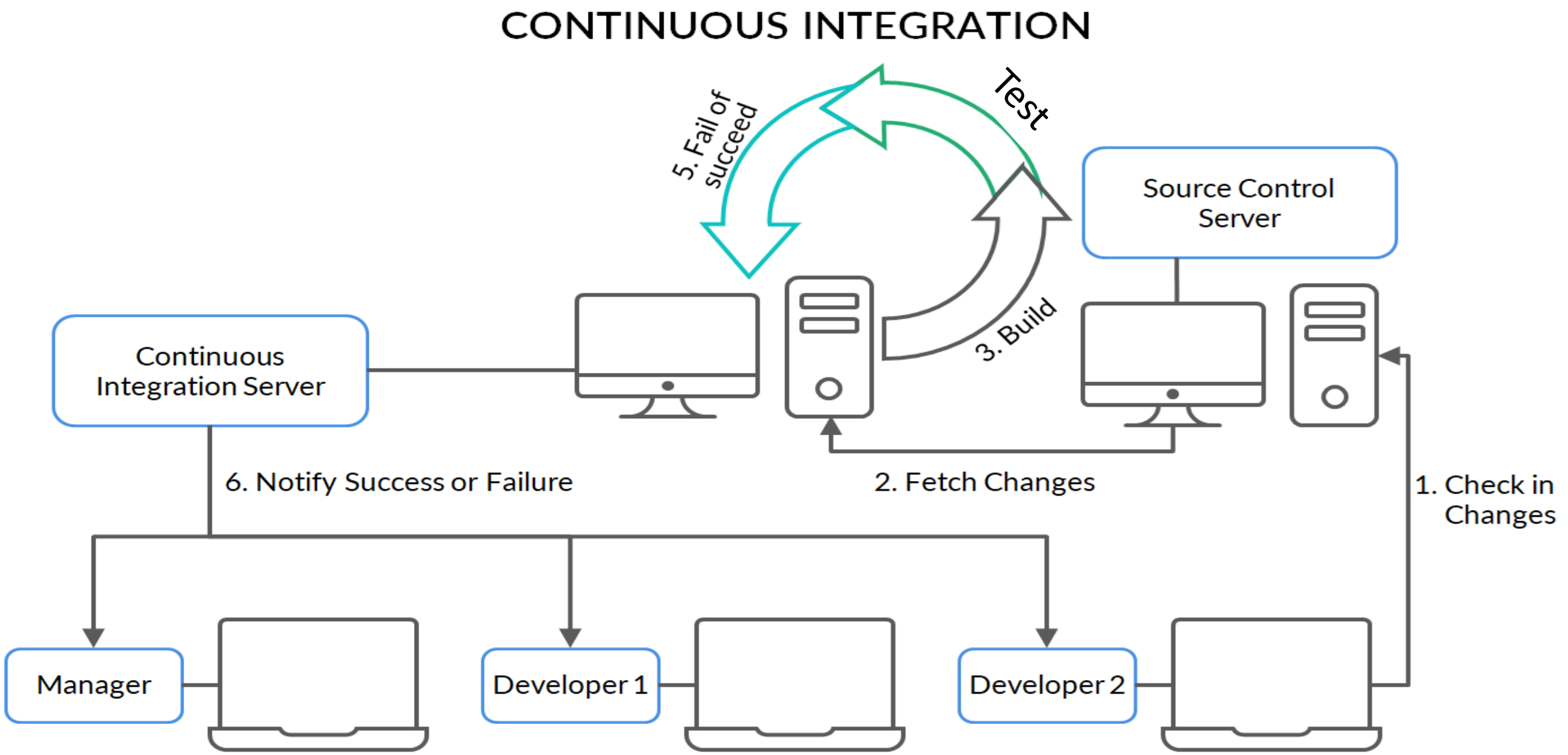



- Agile breaks the wall between Business and Development teams.
- DevOps breaks the wall between Development and Operations teams.
- DevOps centres on the concept of sharing (ideas, issues, processes, tools and goals).

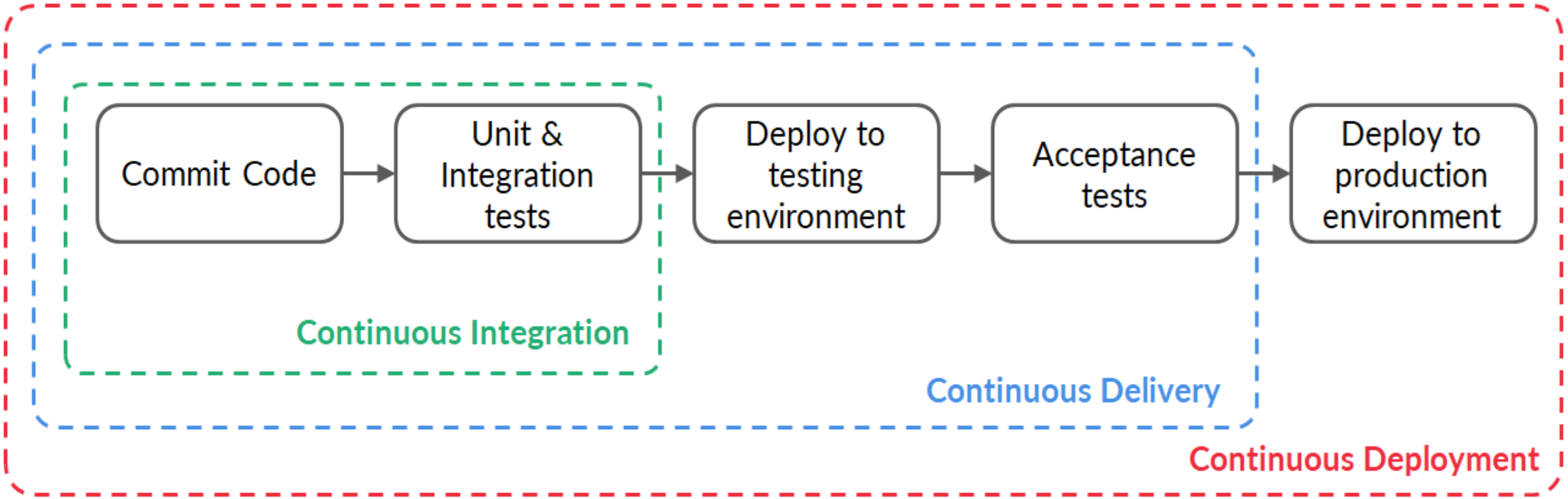
# Continuous Integration

- CI, in its simplest form, involves a tool that continuously monitors version-control systems for any new changes.
- Whenever any new changes are pushed, this tool starts compiling and testing your application.
- If a bug is found or the code fails, the developers are immediately notified to fix the issue.
- If you integrate it with automated end-to-end acceptance checks, CI can serve as a feedback method, offering a straightforward image of the current state of development efforts.
- It will allow you to deploy the latest version of your application either automatically or as a one-click process.

# Continuous Integration



# Continuous – Integration, Delivery and Deployment



# Benefits of CI/CD

- Automation – ensures build, test, QA, delivery and deployment
- Improves consistencies and quality of code
- Delivers new features quickly to the end user
- Increases product visibility
- Helps avoid manual errors
- Helps fix any issues that may arise
- Reduces costs and labour

# Introduction to Jenkins

# Jenkins – An Introduction


- Jenkins is basically a free and open-source automation server.
- Jenkins can be used to automate all types of software development tasks such as building, testing, and delivering or deploying software.
- Jenkins can be installed via native system packages, Docker, or it can be run standalone on any computer.

# How Jenkins Evolved

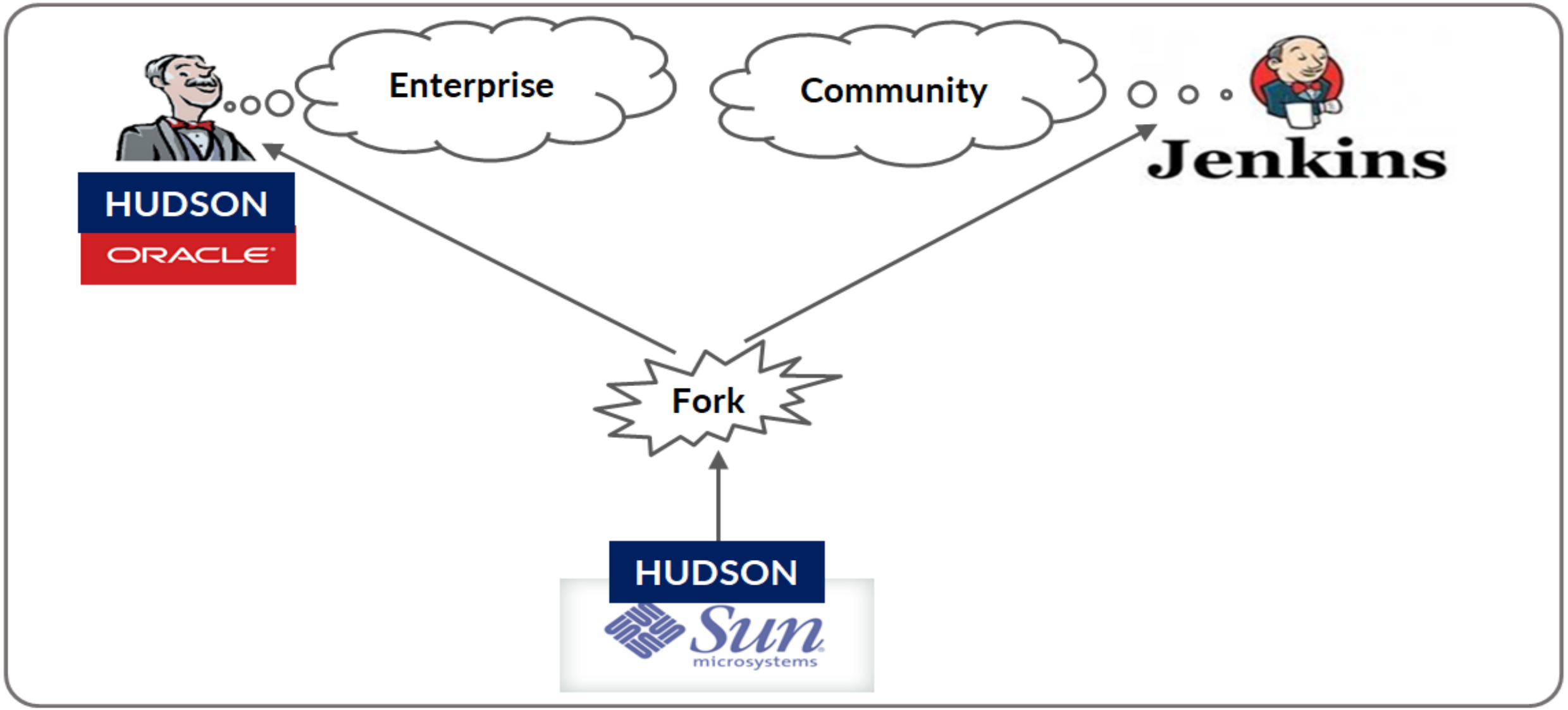
- Jenkins is the outcome of an innovative developer, Kohsuke Kawaguchi, who began this project as a hobby under the name Hudson in late 2004 while working at Sun Microsystems.

- In 2009, Sun Microsystems was acquired by Oracle. At the end of 2010, tensions arose between the Hudson developer community and Oracle. Initially, the problem was with the Java.net infrastructure, which later worsened due to issues related to Oracle's claim to the Hudson trademark.

- In January 2011, the Hudson developer community decided to rename the project as Jenkins. Subsequently, they migrated the original Hudson code base to a new GitHub project and continued their work there.

- After this, the majority of users joined the Jenkins developer community, migrating to Jenkins.

# History of Jenkins



# Features of Jenkins

- Jenkins, originally called Hudson, is an open-source CI tool written in Java.
- Jenkins is a very user-friendly tool, with a simple, intuitive and visually appealing user interface.
- Jenkins has a very low learning curve.
- Jenkins is an extremely flexible tool and hundreds of open-source plugins are available, with more coming up every week.

# Features of Jenkins

- These plugins cover everything from version-control systems, build tools, code quality metrics and build notifiers to integration with external systems, UI customisations, and much more.

- The Jenkins community is a large, dynamic, reactive and welcoming bunch, with passionate champions, active mailing lists, IRC channels and a very vocal blog and Twitter account.

.

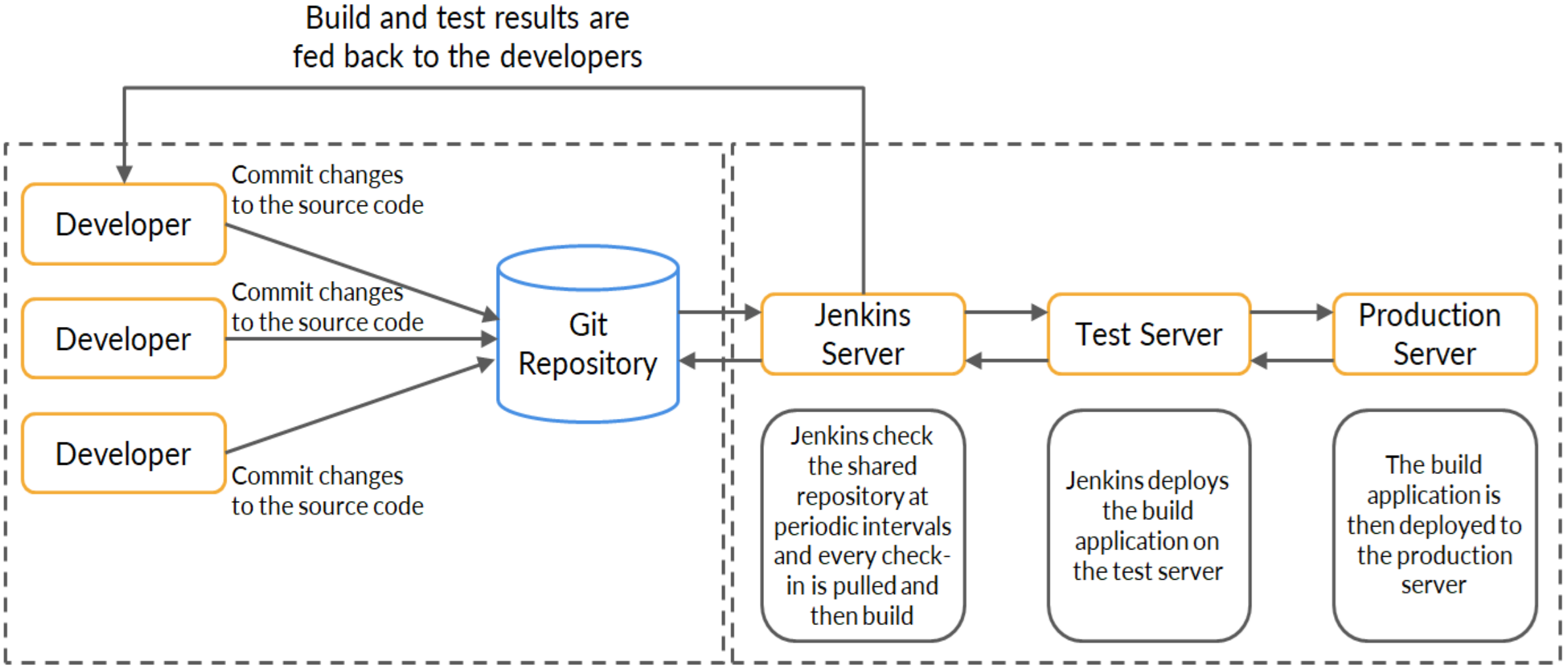
- The pace of development is quite fast, with the latest new features, bug fixes, and plugin updates being released on a weekly basis.

- Jenkins is a JAVA application, so it is platform independent.

# Comparison of CI Tools

| | Jenkins | TeamCity | Bamboo | Travis | Circle | Codeship |
|---|---|---|---|---|---|---|
| Pricing | Free | $299-$1999 | $10-$800 | $69-$489 | $50-$3150 | $75-$1500 |
| Operating system | Windows, Linux, macOS, any Unix-like OS | Windows, Linux, macOS, Solaris, FreeBSD and more | Windows, Linux, macOS, Solaris | Linux, macOS | Linux, iOS, Android | Windows, macOS |
| Hosting | On premise/cloud | On premise | On premise/ Bitbucket as cloud | On premise/cloud | Cloud | Cloud |
| Container support | ✓ | ✓ | ✓ | ✓ | ✓ | Yes for Pro version |
| Plugins | ***** | **** | ** | **** | *** | **** |
| Docs and support | Adequate | Good | Good | Poor | Good | Poor |
| Learning curve and usability | Easy | Medium | Medium | Easy | Easy | Easy |
| Use case | For big Projects | For enterprise needs | For Atlassian integrations | For small projects and startups | For fast development and high budget | For any project |

# Jenkins Architecture



# Companies That Use Jenkins

**Companies: 2836** companies reportedly use **Jenkins** in their tech stacks, including Facebook, Netflix, and Udemy

**Developers: 33308** developers on StackShare have stated that they use **Jenkins.**


Facebook

Netflix


Udemy


Instacart


Robinhood


Twitch

Lyft

Delivery Hero


LinkedIn

Slack, Datadog, BrowserStack, Azure DevOps, and SonarQube are some of the popular **144 tools that integrate with Jenkins.**


Slack


Datadog


BrowserStack


Azure DevOPs


SonarQube

Rancher


AWS CodePipeline


Mattermost


AWS CodeCommit

# Configuring and Setting Up Jenkins

# Jenkins Installation

**Install Java**

sudo apt-get update
sudo apt install -y openjdk-11-jdk


**To check: java - - version**

openjdk version "11.0.11" 2021-04-20
OpenJDK Runtime Environment (build 11.0.11+9-Ubuntu-0ubuntu2.18.04)
OpenJDK 64-Bit Server VM (build 11.0.11+9-Ubuntu-0ubuntu2.18.04, mixed mode, sharing)

**Install Jenkins**

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

**Install ca-certificates** (man 8 update-ca-certificates)

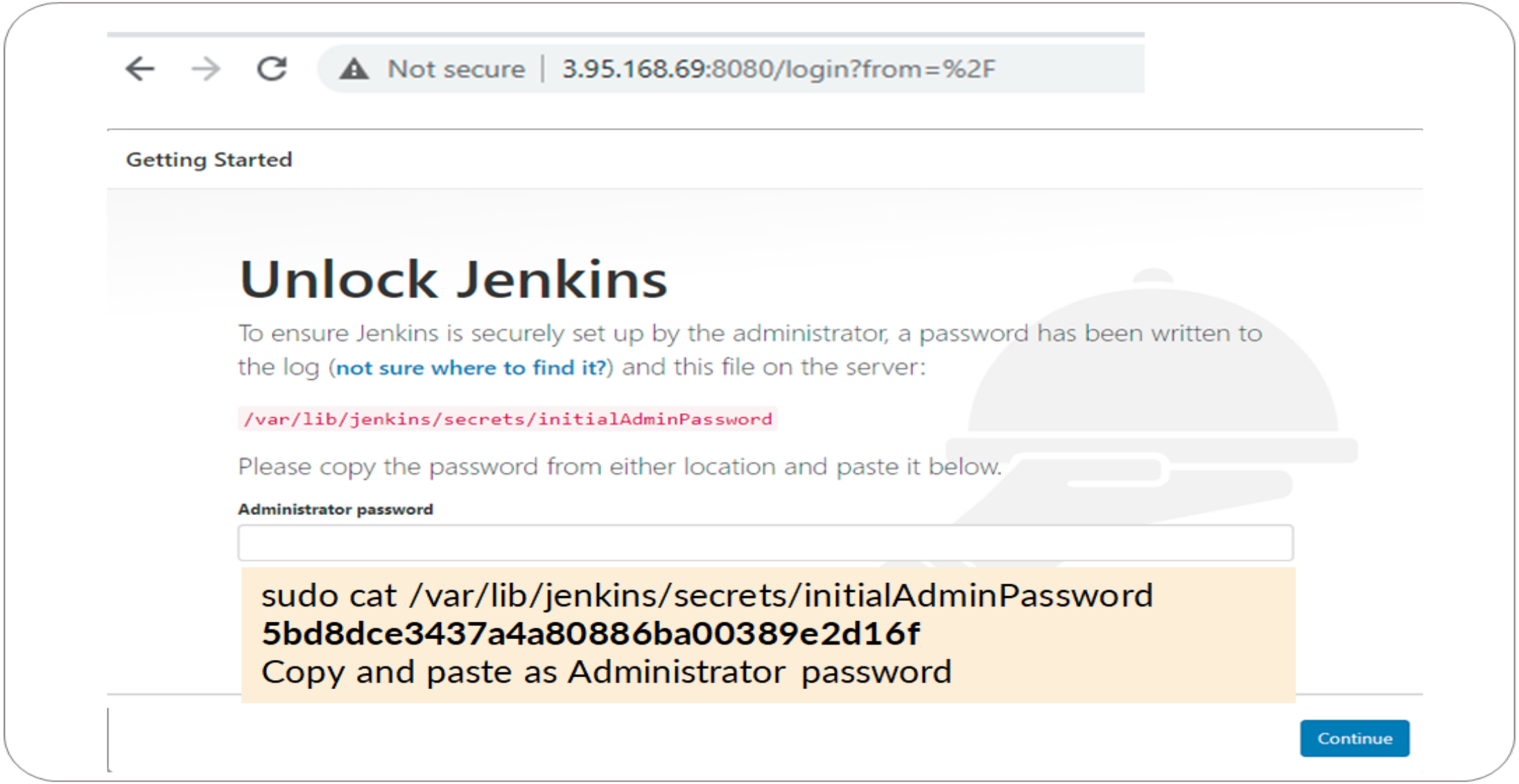
sudo apt install ca-certificates

**Install Jenkins**

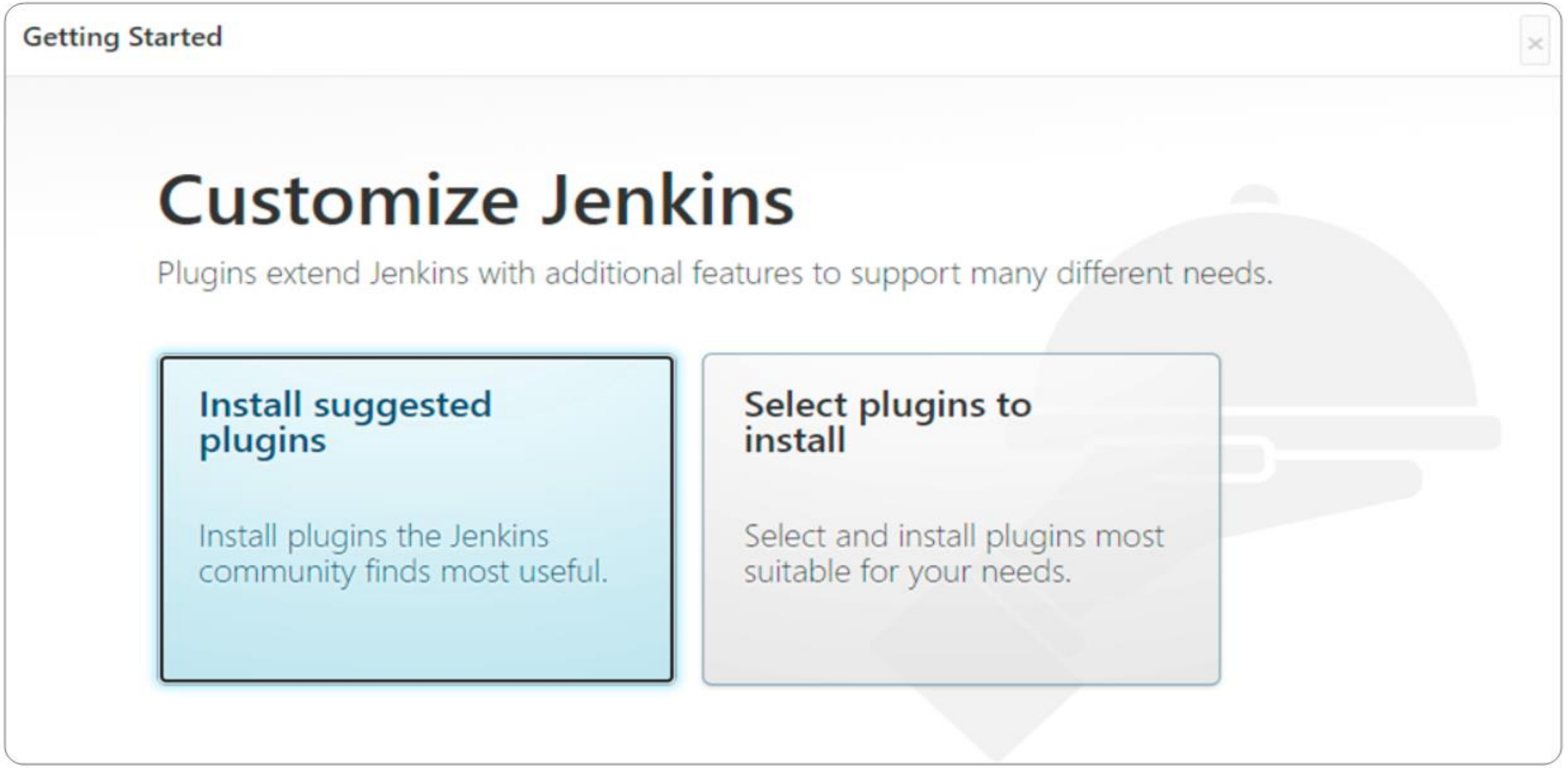
sudo apt-get update
sudo apt-get install jenkins

**To check Jenkins version**: vim /var/lib/jenkins/config.xml
**To copy admin password**: sudo cat /var/lib/jenkins/secrets/initialAdminPassword

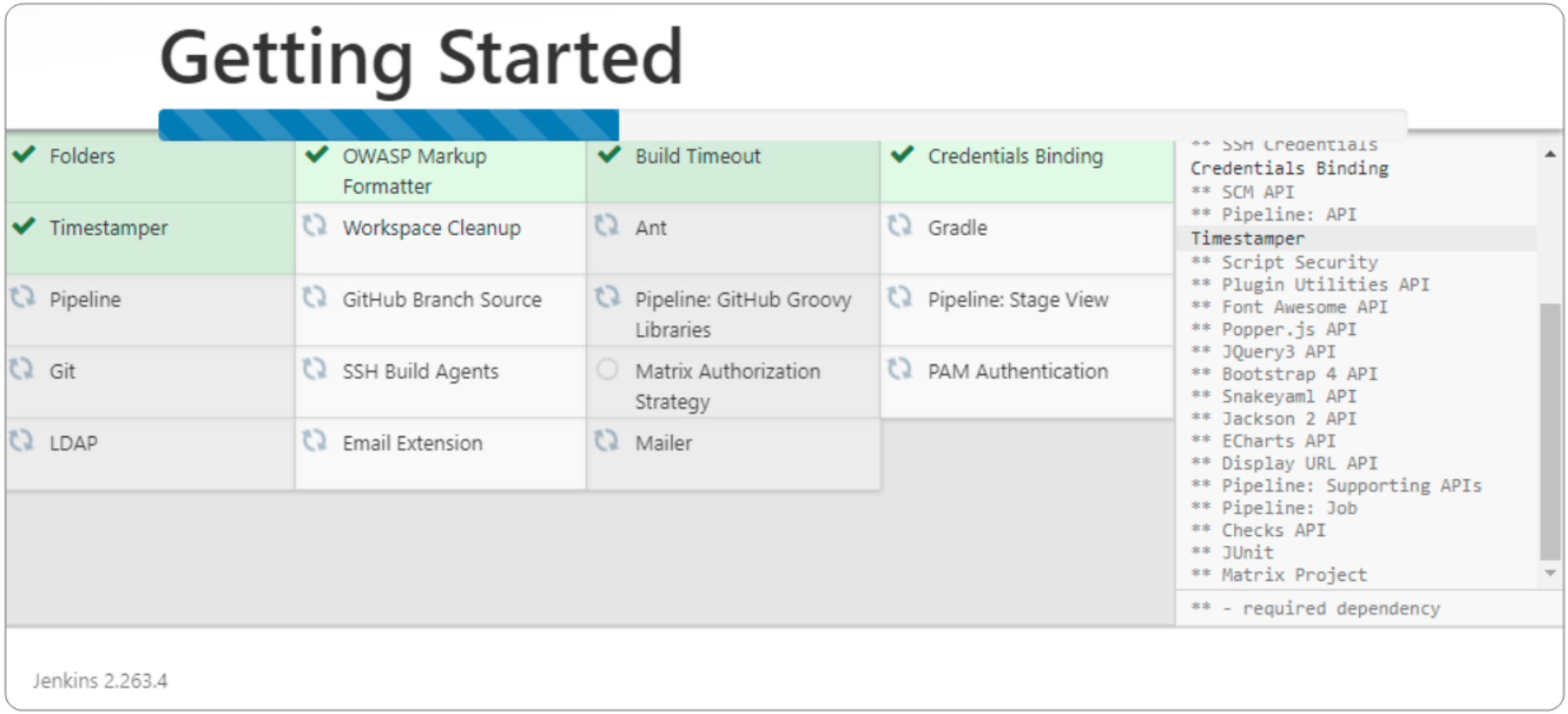
# Execute https://<ipaddress>:8080



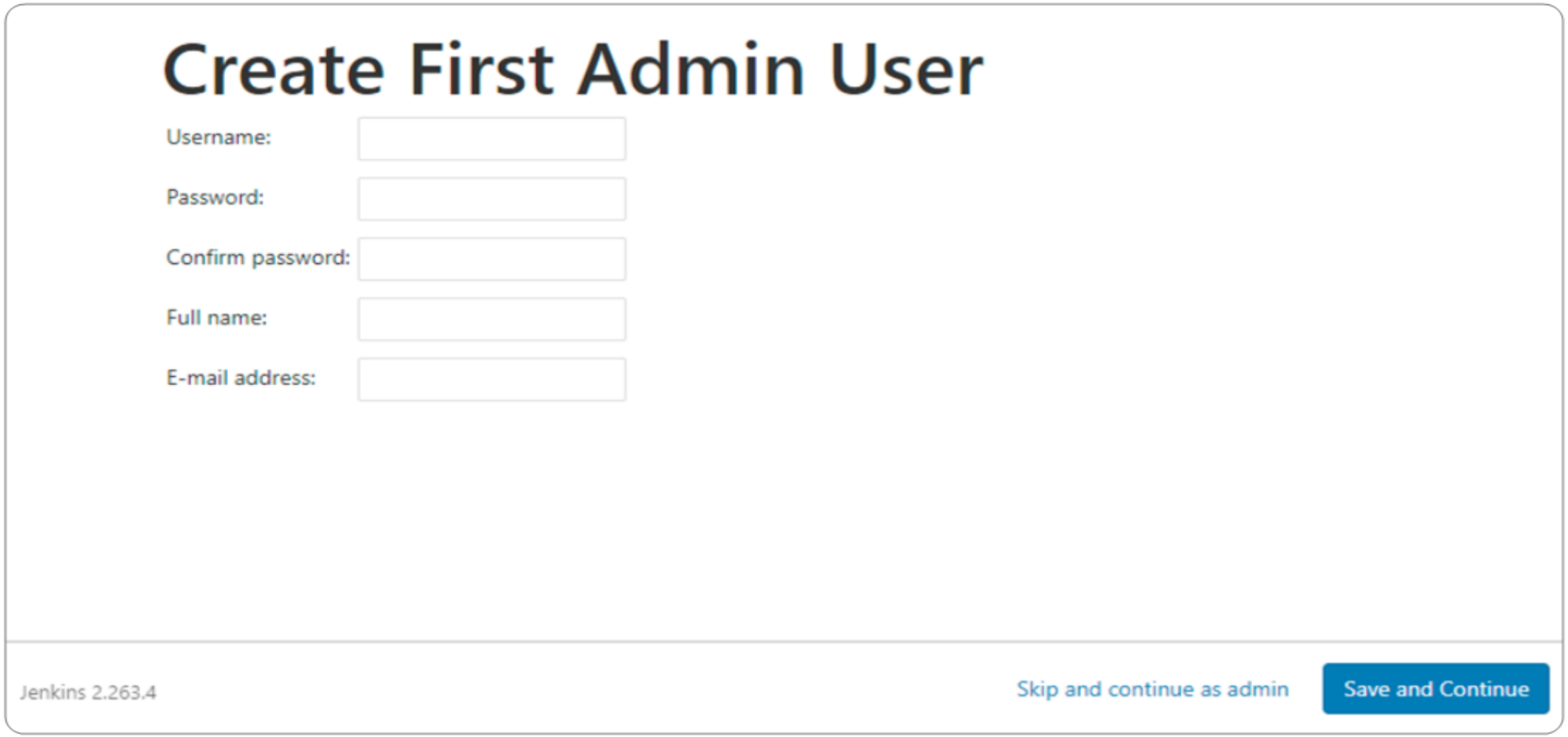
# Installation of Default Plugins



# Install Suggested Plugins

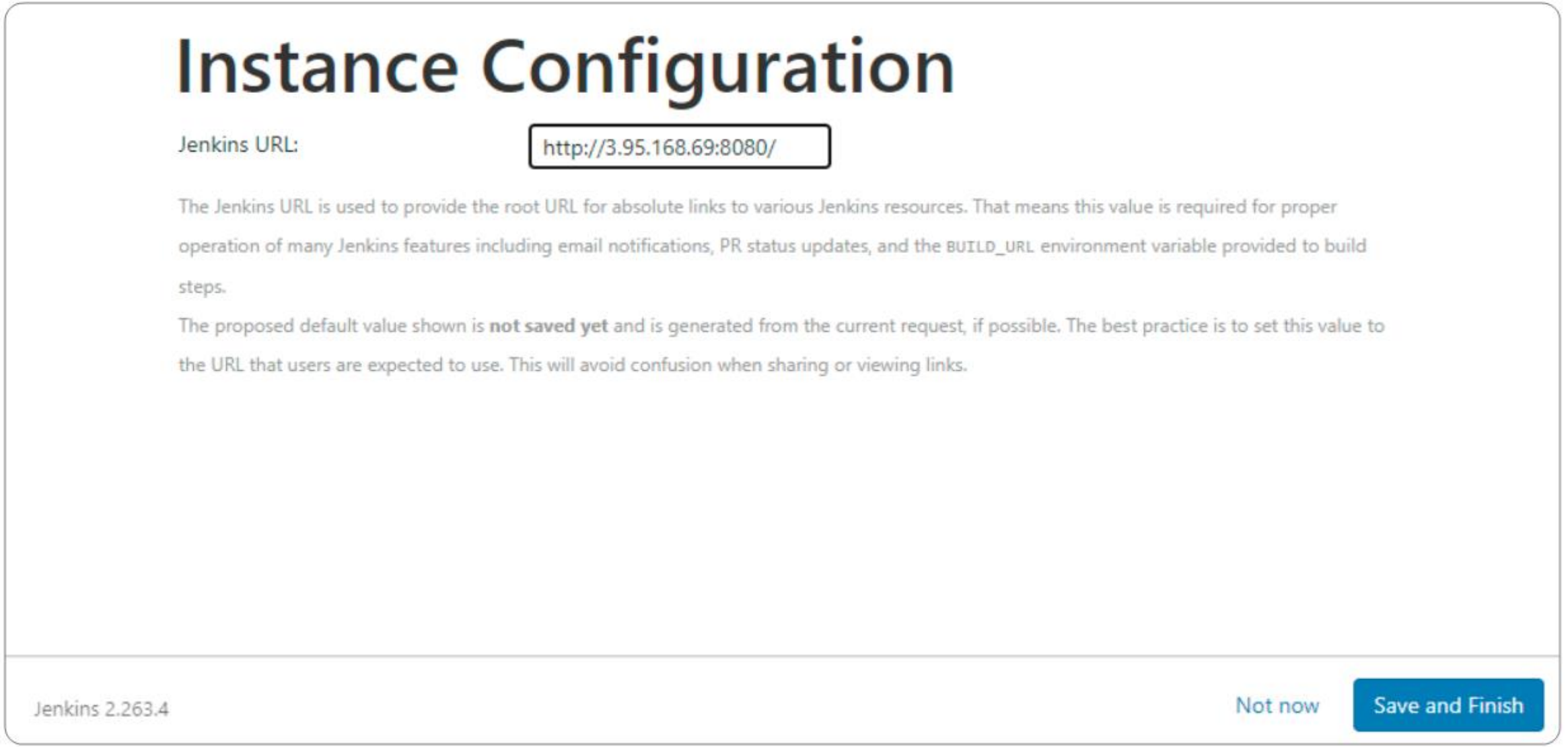


# Continue as Admin

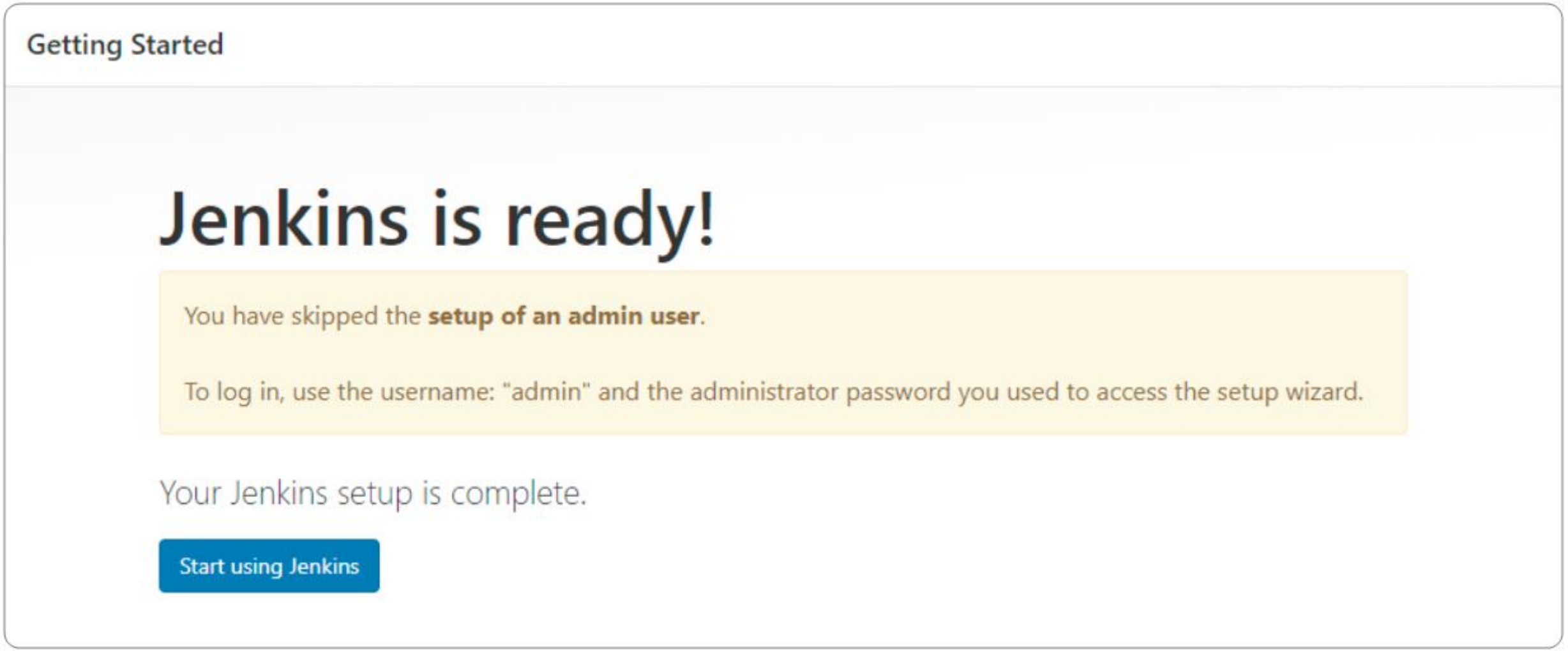



**Create User Accounts Later**

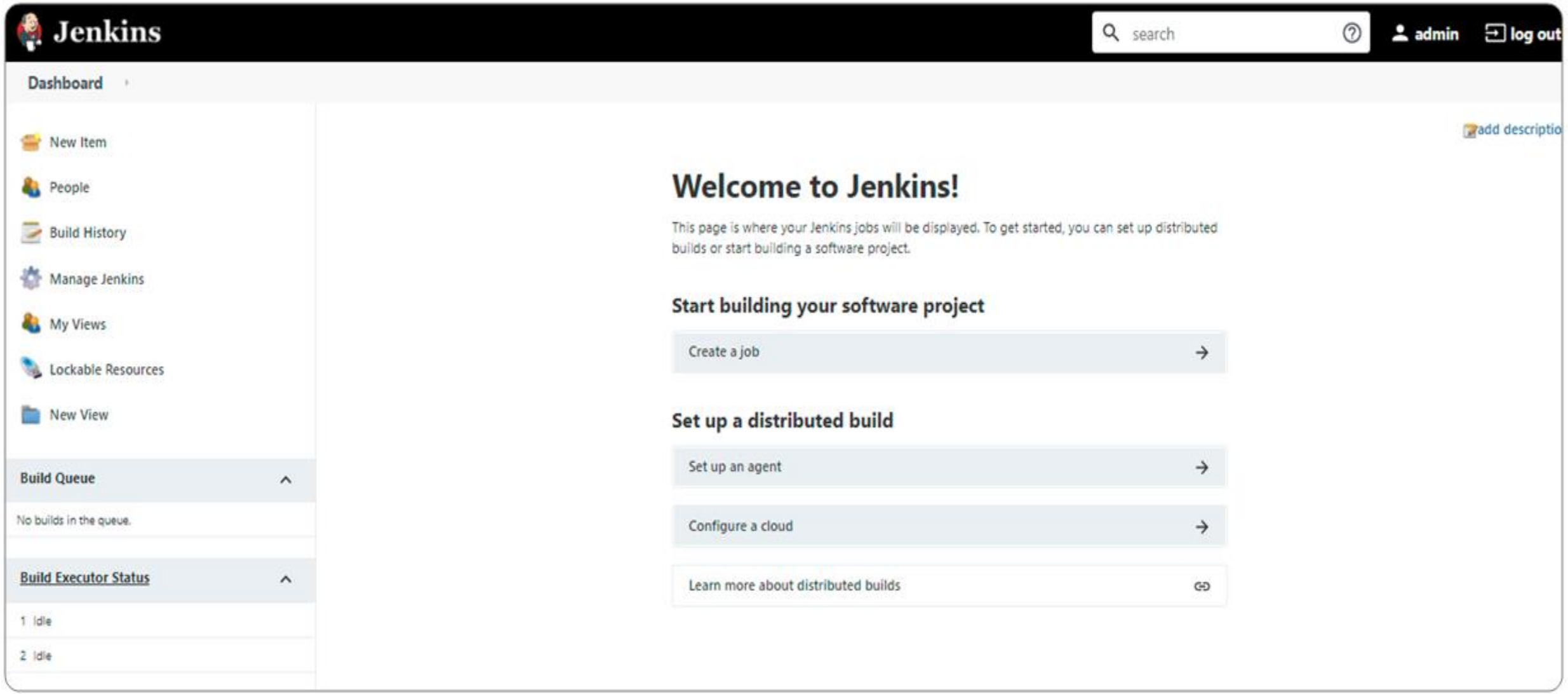
# Save Jenkins Configuration



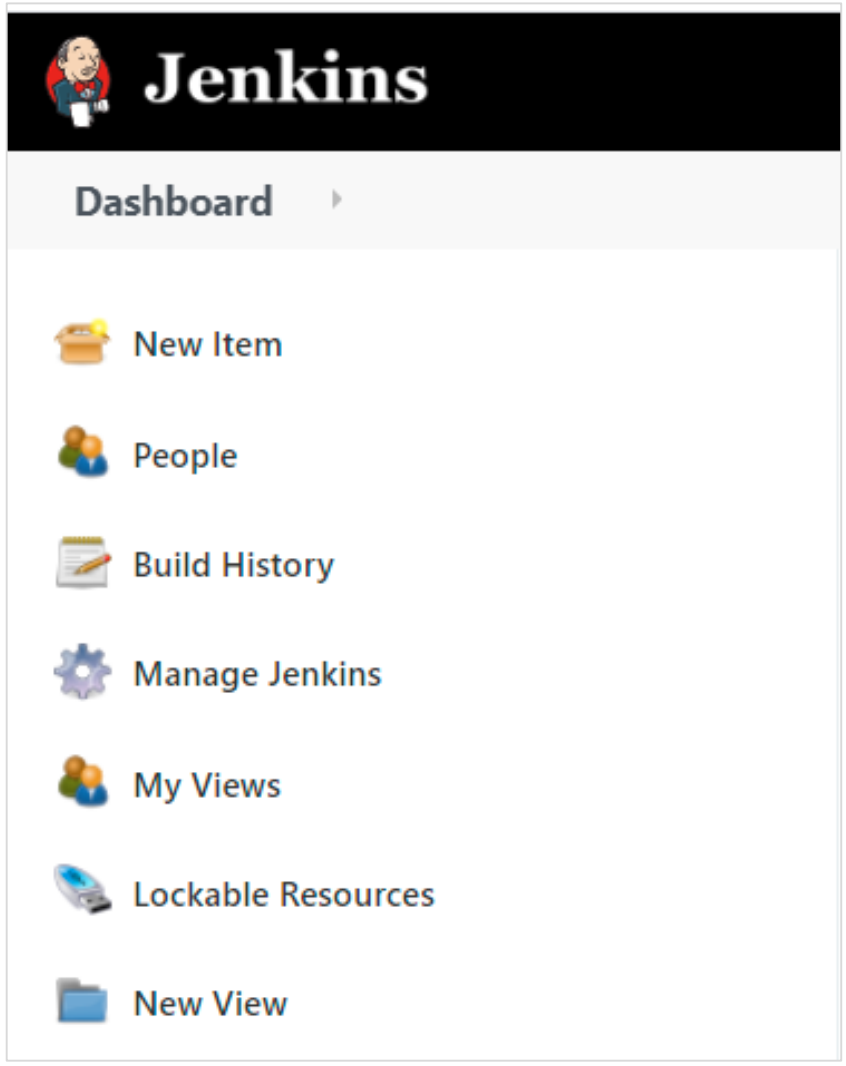
# Start Jenkins



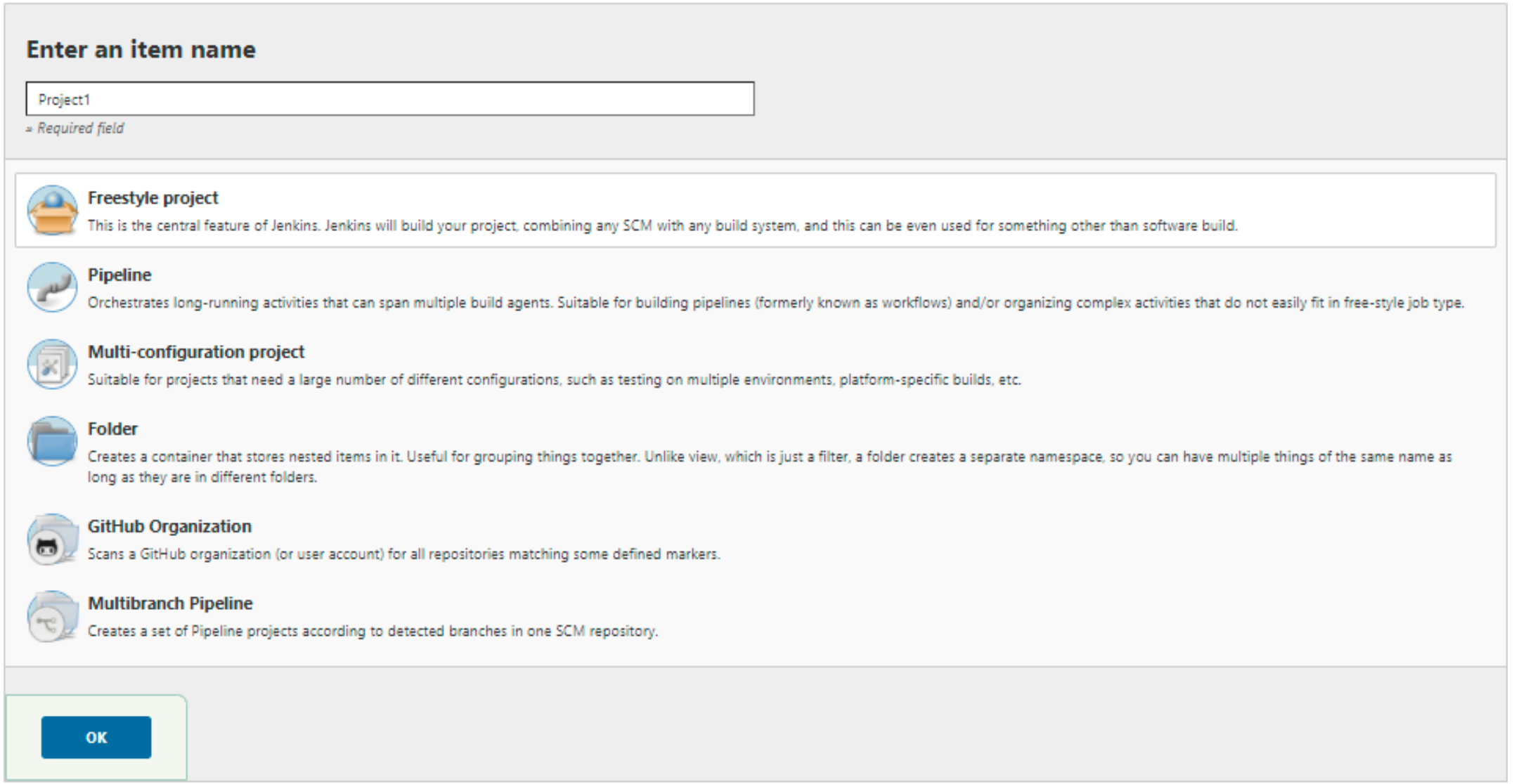
# Jenkins Dashboard



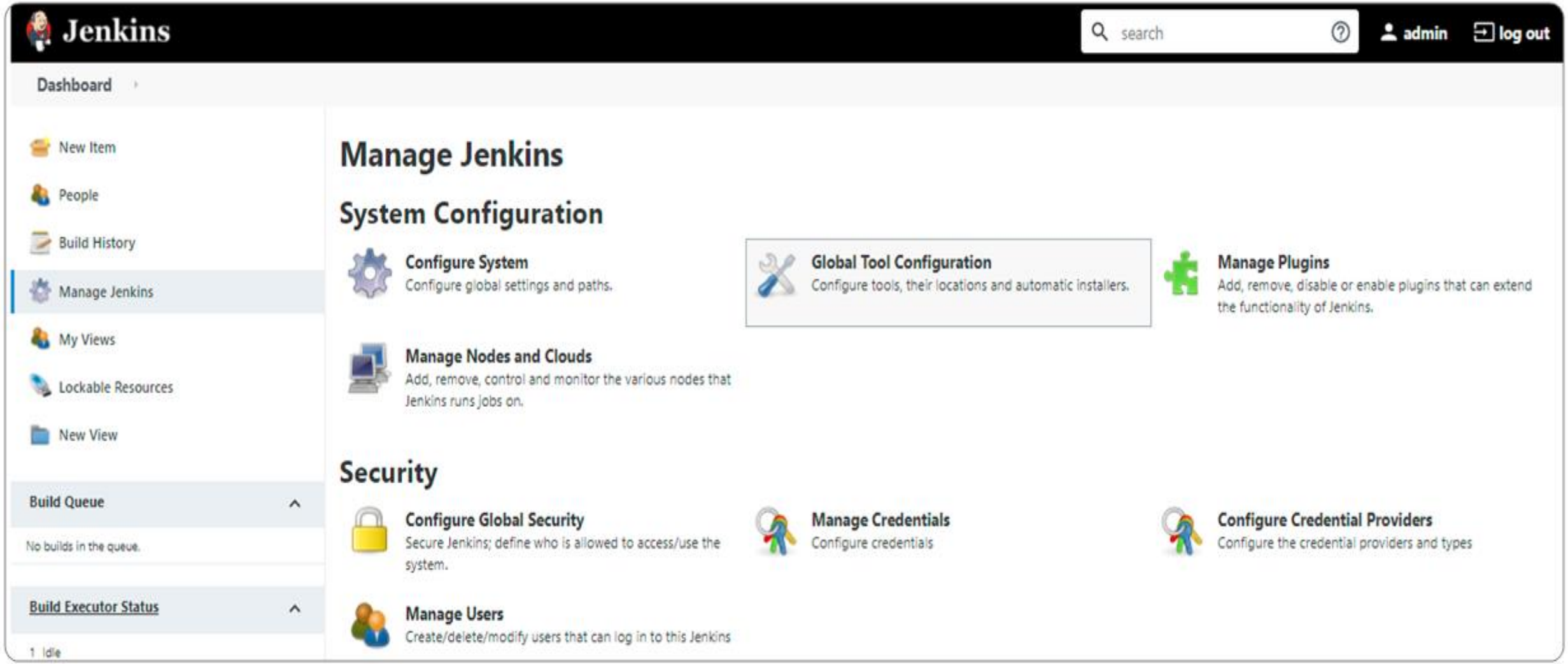
# Dashboard – Various Options



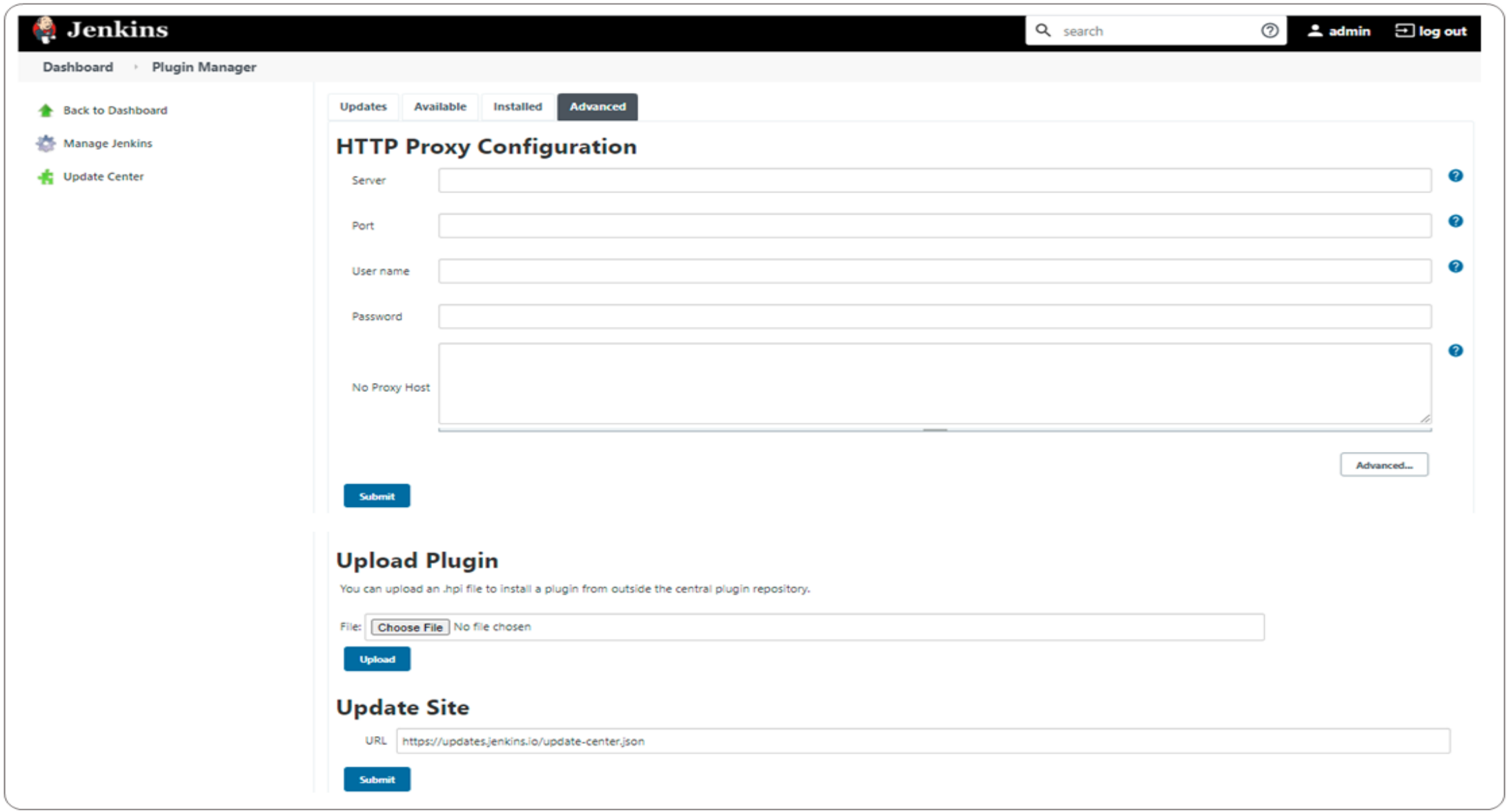
# Select New Item



# Manage Jenkins – Various Options



# Jenkins Plugin Manager – Advanced



# Understand How to Create Jobs in Jenkins

# Start Jenkins

```
ubuntu@ip-172-31-81-117:~$ sudo service jenkins start
ubuntu@ip-172-31-81-117:~$ sudo service jenkins status
● jenkins.service - LSB: Start Jenkins at boot time
   Loaded: loaded (/etc/init.d/jenkins; generated)
   Active: active (exited) since Tue 2021-03-09 01:13:37 UTC; 2min 52s ago
     Docs: man:systemd-sysv-generator(8)
    Tasks: 0 (limit: 1140)
   CGroup: /system.slice/jenkins.service

Mar 09 01:13:34 ip-172-31-81-117 systemd[1]: Starting LSB: Start Jenkins at boot time...
Mar 09 01:13:35 ip-172-31-81-117 jenkins[842]: Correct java version found
Mar 09 01:13:35 ip-172-31-81-117 jenkins[842]:  * Starting Jenkins Automation Server jenkins
Mar 09 01:13:35 ip-172-31-81-117 su[943]: Successful su for jenkins by root
Mar 09 01:13:35 ip-172-31-81-117 su[943]: + ??? root:jenkins
Mar 09 01:13:36 ip-172-31-81-117 su[943]: pam_unix(su:session): session opened for user jenkins by (uid=0)
Mar 09 01:13:36 ip-172-31-81-117 su[943]: pam_unix(su:session): session closed for user jenkins
Mar 09 01:13:37 ip-172-31-81-117 jenkins[842]:    ...done.
Mar 09 01:13:37 ip-172-31-81-117 systemd[1]: Started LSB: Start Jenkins at boot time.
ubuntu@ip-172-31-81-117:~$
```

1. Know the status of **Jenkins:** sudo service **jenkins** status.
2. To **start Jenkins:** sudo service **jenkins start.**
3. To **stop Jenkins:** sudo service **jenkins stop.**
4. To **restart Jenkins:** sudo service **jenkins restart.** (stop and start)

# Jenkins Workspace

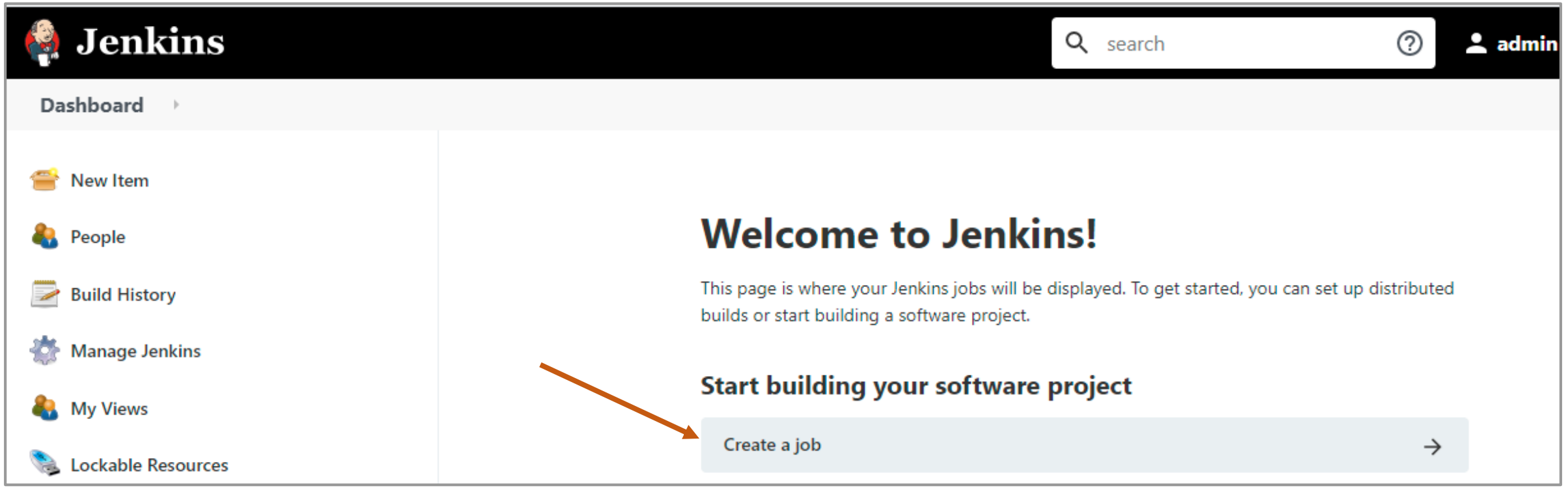
**Jenkins home directory:**

ubuntu@ip-172-31-81-117:~$ sudo su - jenkins
jenkins@ip-172-31-81-117:~$ pwd
/var/lib/jenkins
jenkins@ip-172-31-81-117:~$

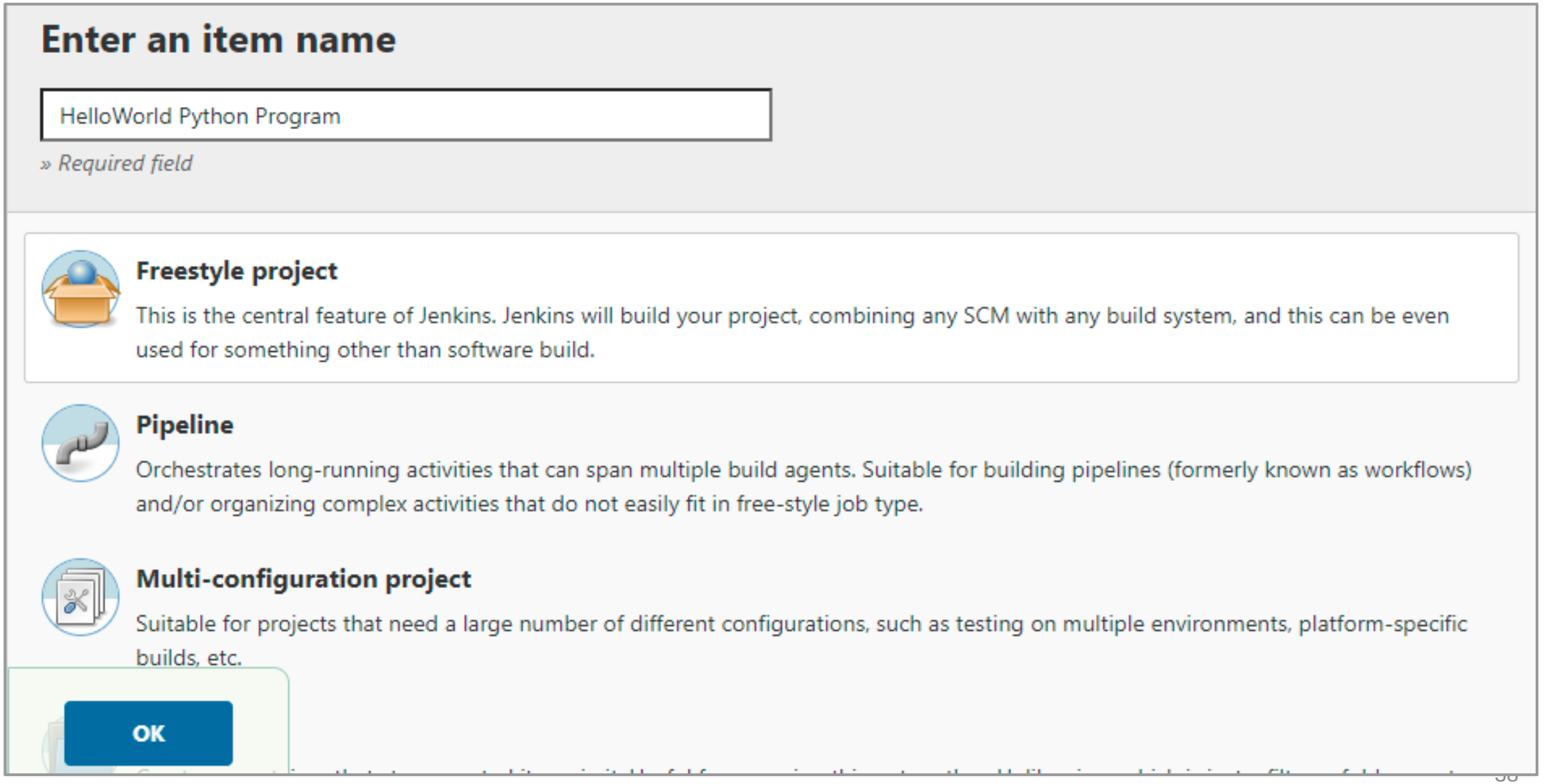
Jenkins home directory: /var/lib/jenkins
It contains the details of the Jenkins server configuration, which is used to configure in the Manage Jenkins. The core configuration files are stored in config.xml.

It contains two important subdirectories: **jobs and workspace**.
Jobs directory contains configuration details of the build job in config.xml file.
Jenkins builds the given project in the workspace directory. Each project has its own directory in the workspace.

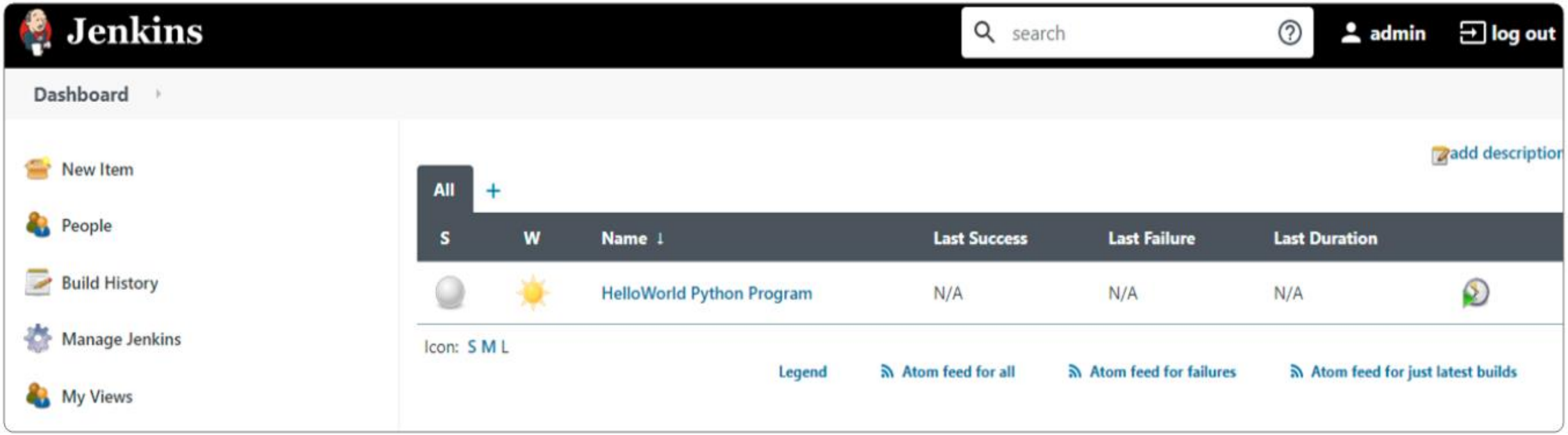
# Click to Create a Job



# Freestyle Project

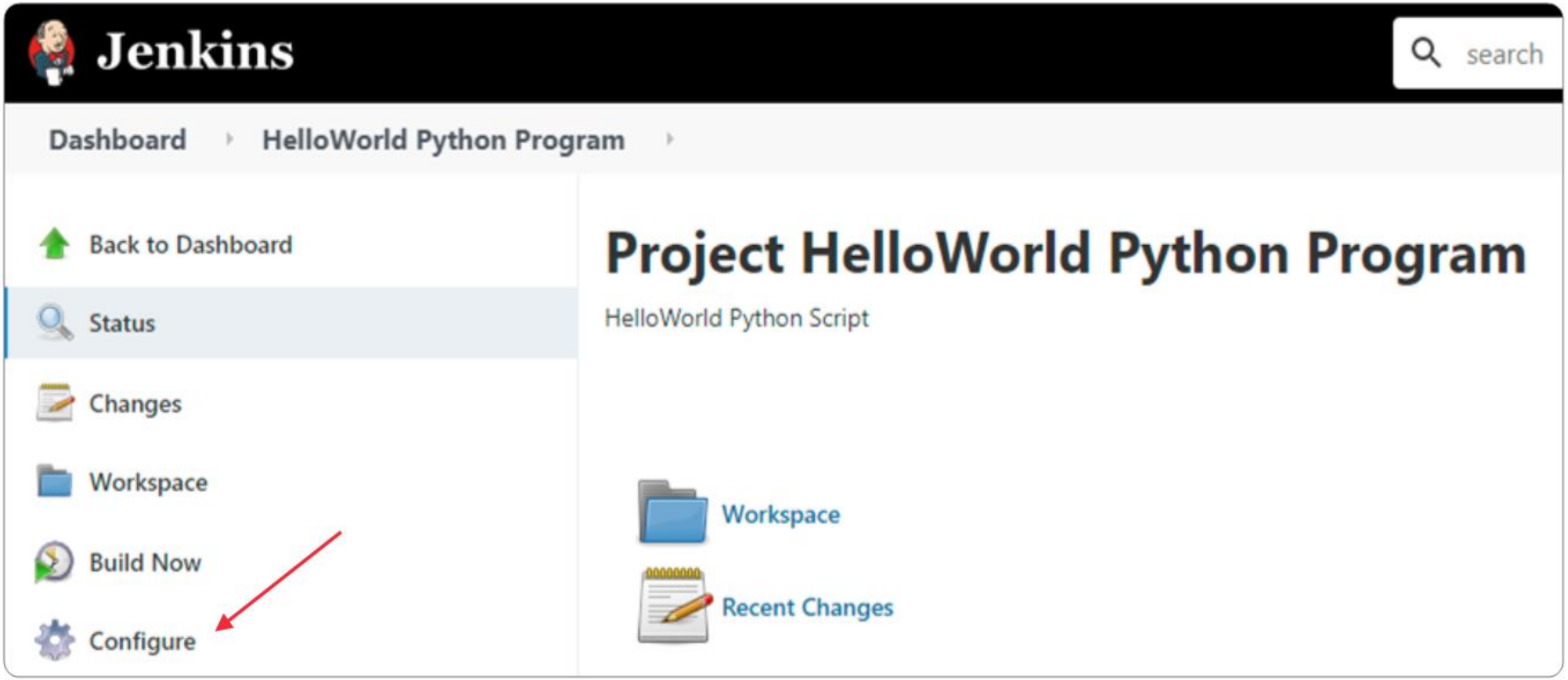


# Example: HelloWorld Python Program

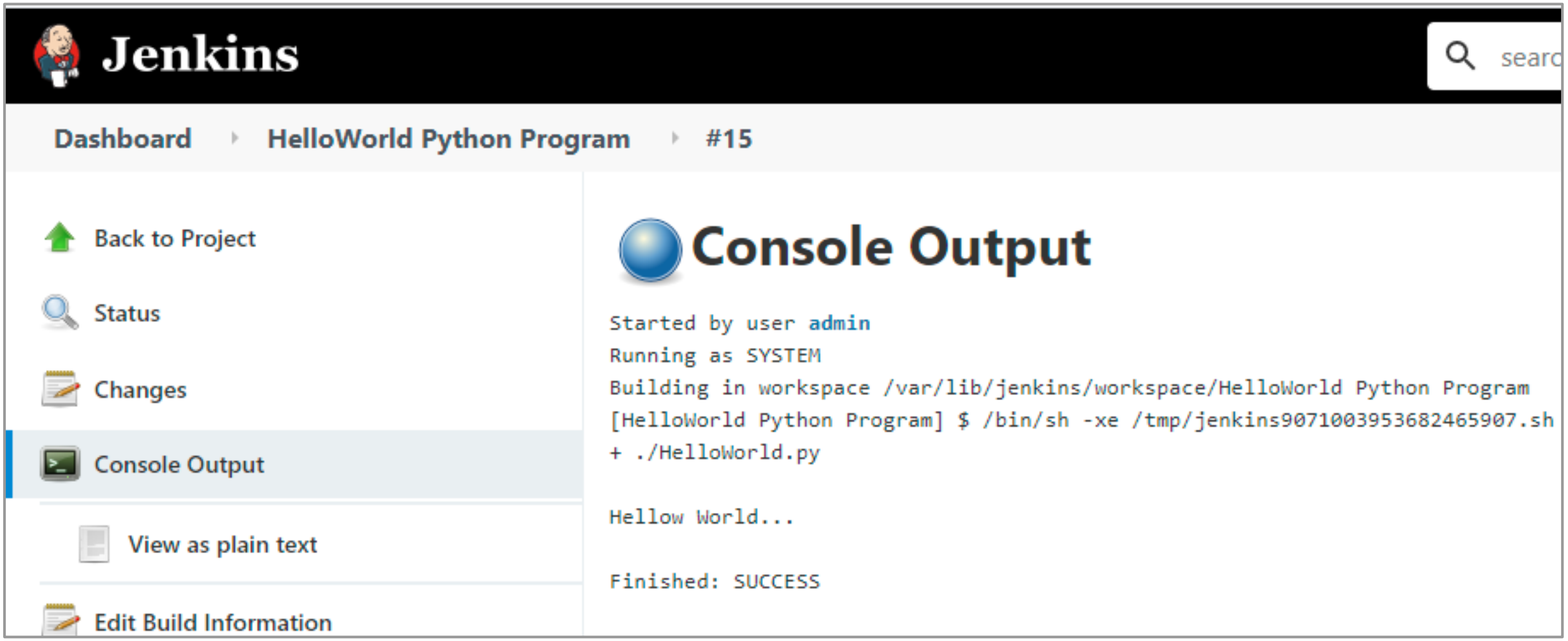



```
#! /usr/bin/python3
# This Phython program will print Hello World...
print("Hello World ...\n")
```

# Build the Project



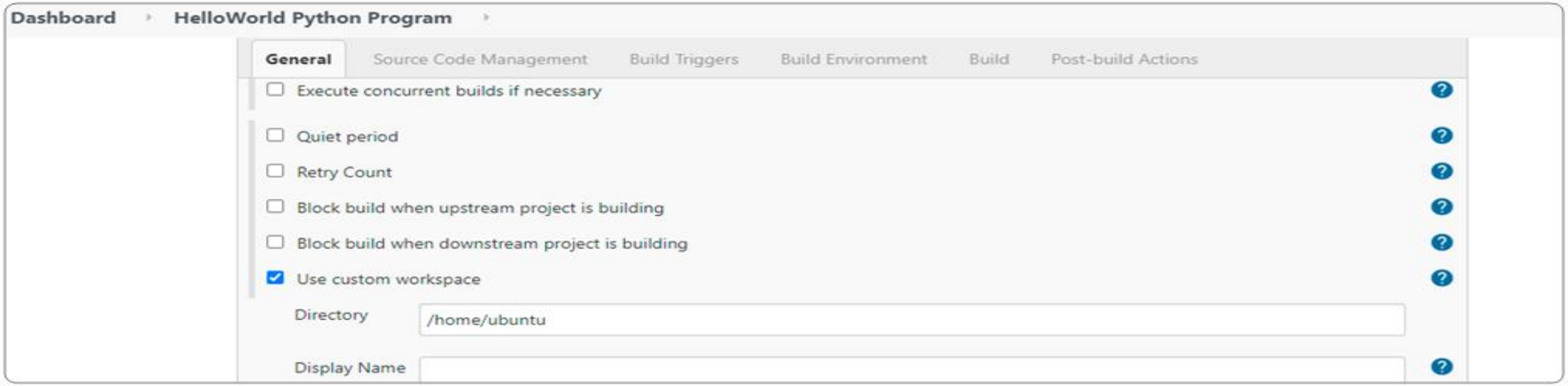
# Console Output



# Check Log Files

```
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds$ ls
1  10  2  3  4  5  6  7  8  9  legacyIds  permalinks
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds$ cd 4
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds/4$ ls
build.xml  changelog.xml  log
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds/4$ cat log
Started by user ha:////4O1Xwh1EUGVcmahxd0uDY/VpeIbwK+Jgej0qfQuwtGnlAAAAlx+LCAAAAAAAAP9b85aBtbiIQTGjNKU4P08vOT+v
1x6OyILUoJzMv2y+/JJUBAhiZGBgqihhk0NSjKDWzXb3RdlLBUSYGJk8GtpzUvPSSDB8G5tKinBIGIZ+sxLJE/ZzEvHT94JKizLx0a6BxUmjGOU
gZu/dLi1CL9xJTczDwAj6GcLcAAAAA=admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/HelloWorld Python Program
[HelloWorld Python Program] $ /bin/sh -xe /tmp/jenkins10839759464246091234.sh
+ echo Hello World
Hello World
Finished: SUCCESS
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds/4$
```

# Custom Workspace

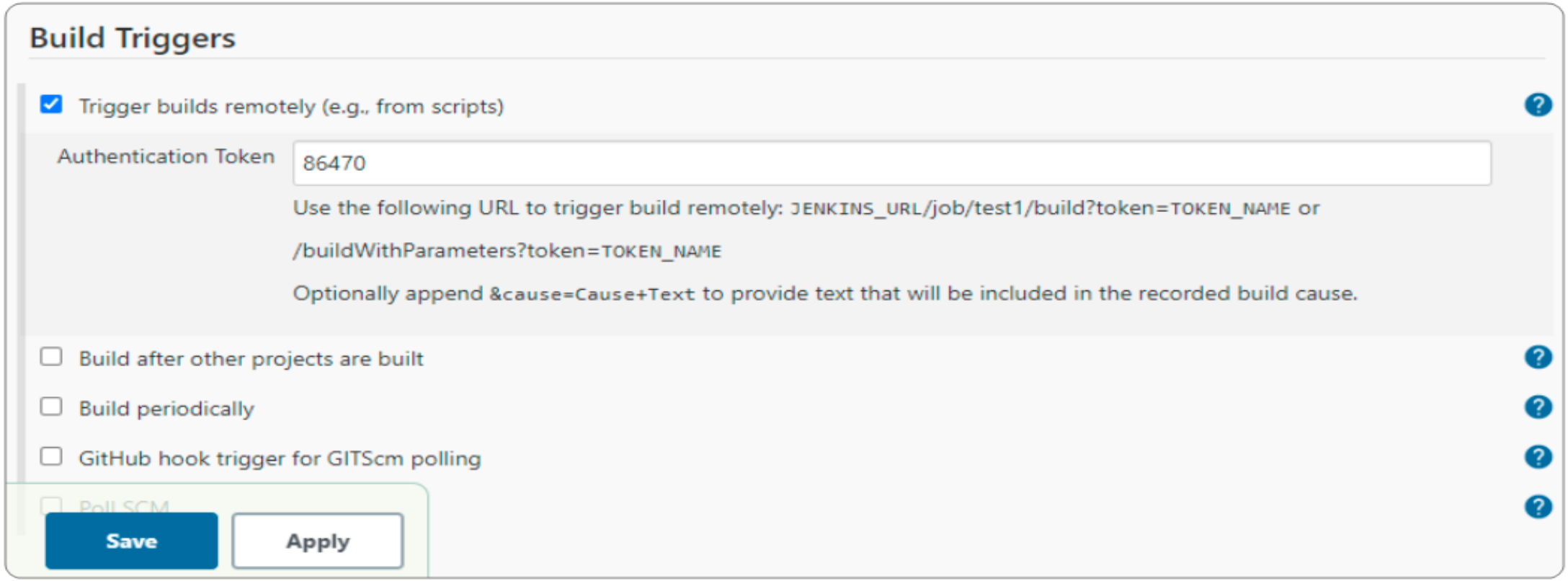



```
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds$ ls
1  10  2  3  4  5  6  7  8  9  legacyIds  permalinks
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds$ cd 9
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds/9$ cat log
Started by user ha:////4O1Xwh1EUGVcmahxd0uDY/VpeIbwK+Jgej0qfQuwtGnlAAAAlx+LCAAAAAAAAP9b85aBtbiIQTGj
1x6OyILUoJzMv2y+/JJUBAhiZGBgqihhk0NSjKDWzXb3RdlLBUSYGJk8GtpzUvPSSDB8G5tKinBIGIZ+sxLJE/ZzEvHT94JKizL
gZu/dLi1CL9xJTczDwAj6GcLcAAAAA=admin
Running as SYSTEM
Building in workspace /home/ubuntu
[ubuntu] $ /bin/sh -xe /tmp/jenkins21212885716844030 99.sh
+ ./HelloWorld.py

Hellow World...

Finished: SUCCESS
ubuntu@ip-172-31-81-117:/var/lib/jenkins/jobs/HelloWorld Python Program/builds/9$
```

# Custom Workspace: Console Output

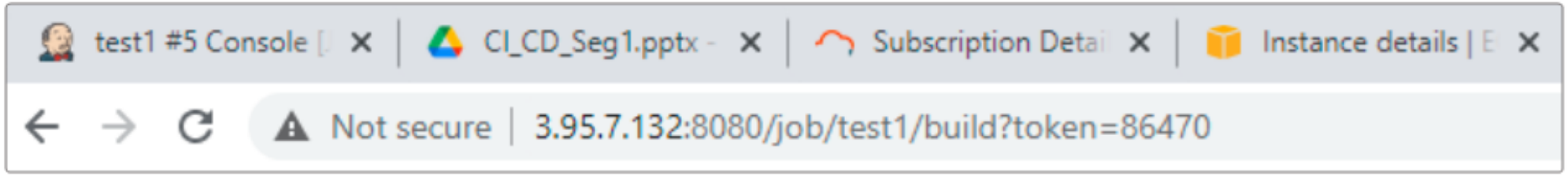
# Build Jenkins Job Remotely

Step 1: Choose the project.
Step 2: Configure – Build – enable Trigger builds remotely.
Step 3: Modify: JENKINS_URL/job/test1/build?token=TOKEN_NAME
http://3.95.7.132:8080/job/test1/build?token=86470.
Step 4: Open a browser and enter: http://3.95.7.132:8080/job/test1/build?token=86470
Step 5: Verify the build number.

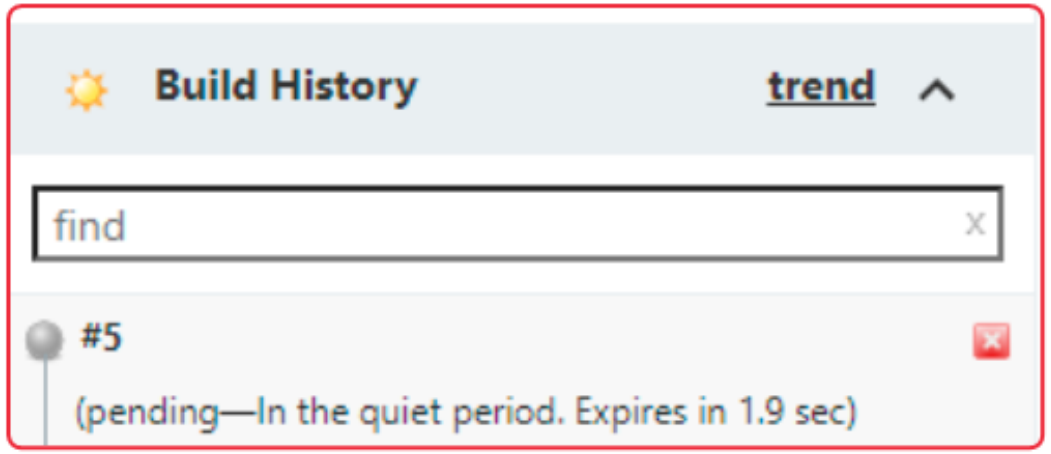


# Trigger Build Job by URL

**Open a browser, type the URL and enter**




**Check Build History in the Jenkins Server**



# Build by Remote Host – Check Console Output

Manual Build

## Console Output

```
Started by user admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/test1
[test1] $ /bin/sh -xe /tmp/jenkins17988033407760365594.sh
+ cd ../../
+ echo The total number of directories in the /var/lib/jenkins
The total number of directories in the /var/lib/jenkins
+ wc -l
+ grep ^d
+ ls -Rl
676
Finished: SUCCESS
```

Trigger Build Remotely

## Console Output

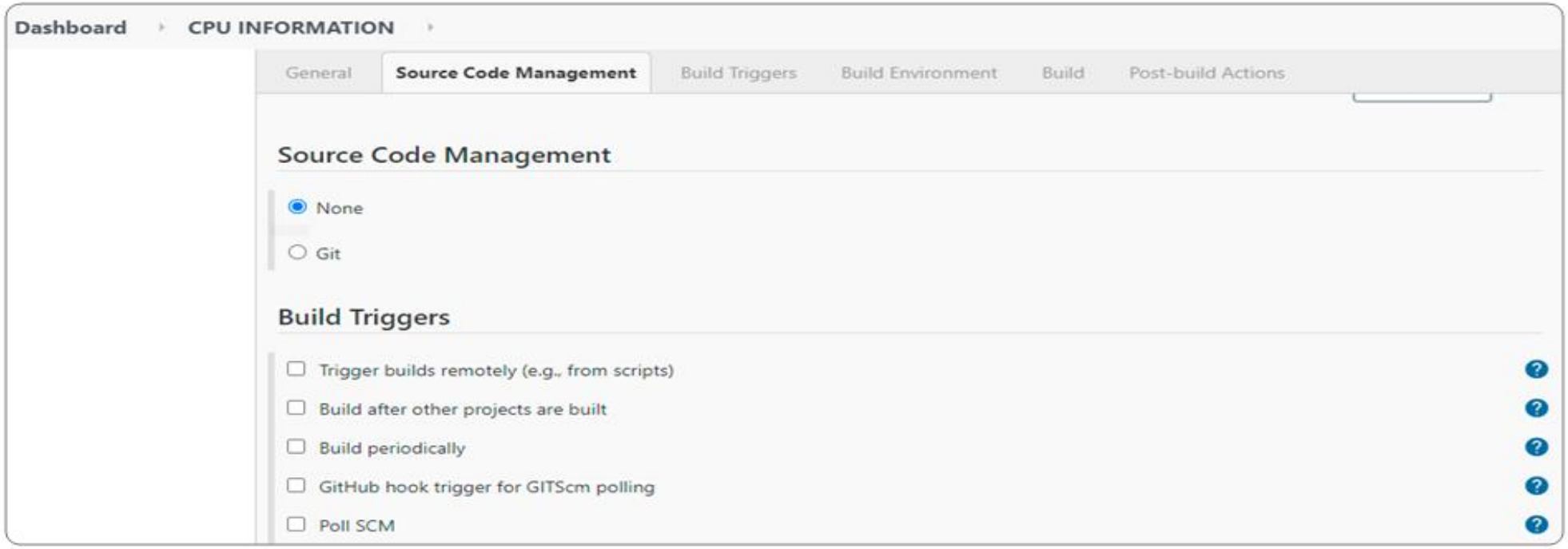
```
Started by remote host 122.181.192.30
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/test1
[test1] $ /bin/sh -xe /tmp/jenkins1040342982316183531.sh
+ cd ../../
+ echo The total number of directories in the /var/lib/jenkins
The total number of directories in the /var/lib/jenkins
+ wc -l
+ grep ^d
+ ls -Rl
678
Finished: SUCCESS
```

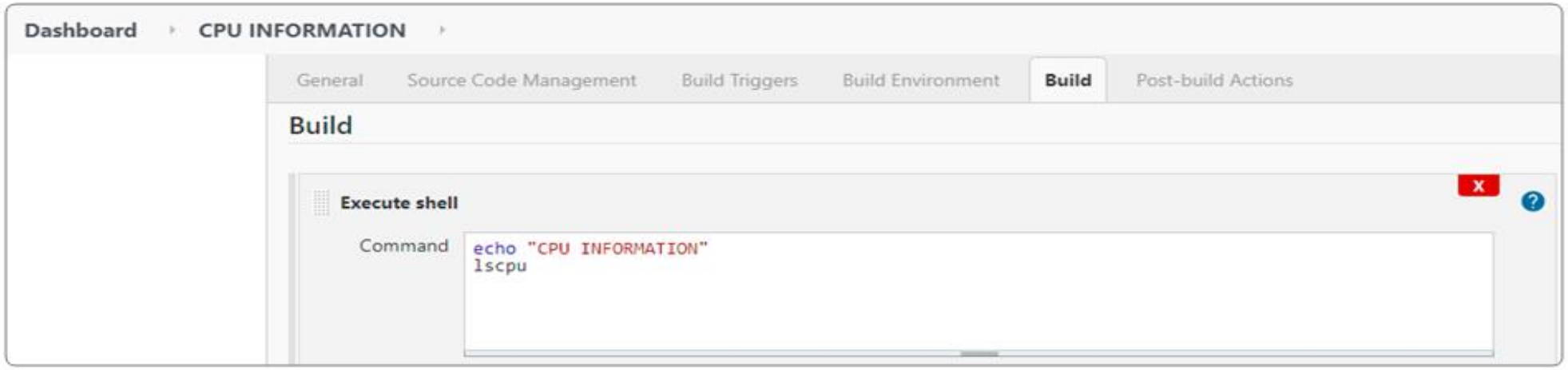
# Job Chaining in Jenkins

- Create 3 Jobs:
  - Job1: CPU INFORMATION
  - Job2: RAM INFORMATION
  - Job3: DISK INFORMATION

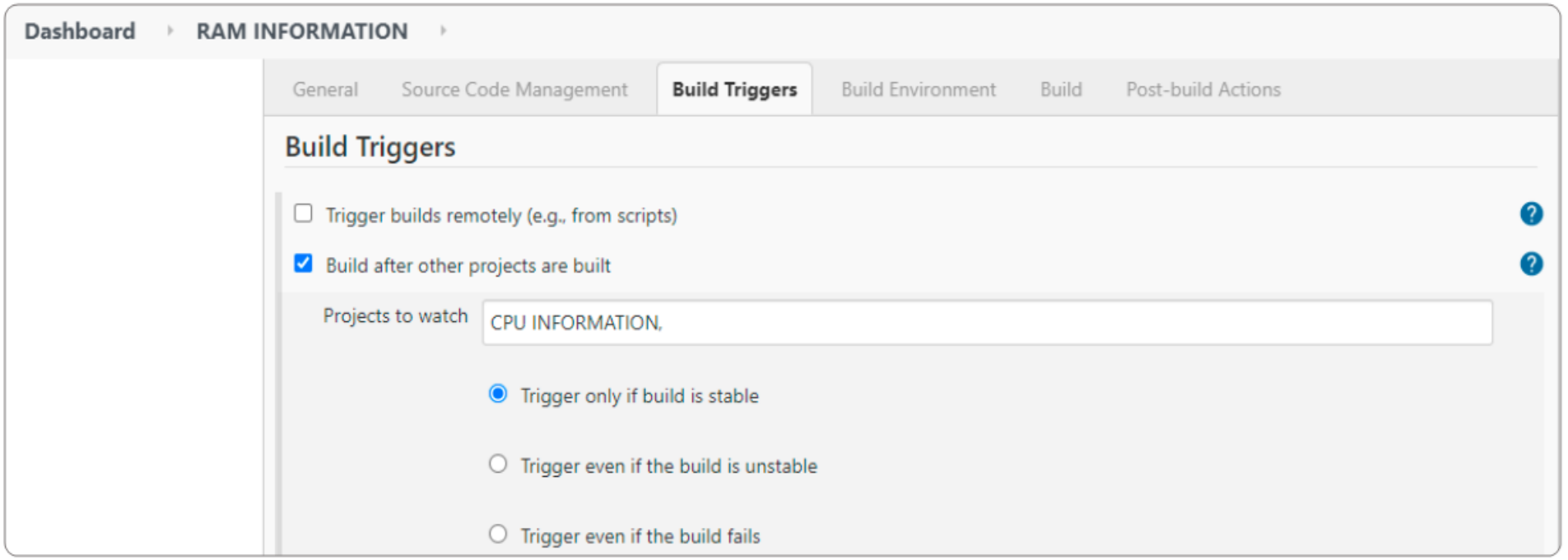
- Job2 will start after Job1 build is successful
- Job3 will start after job2 build is successful

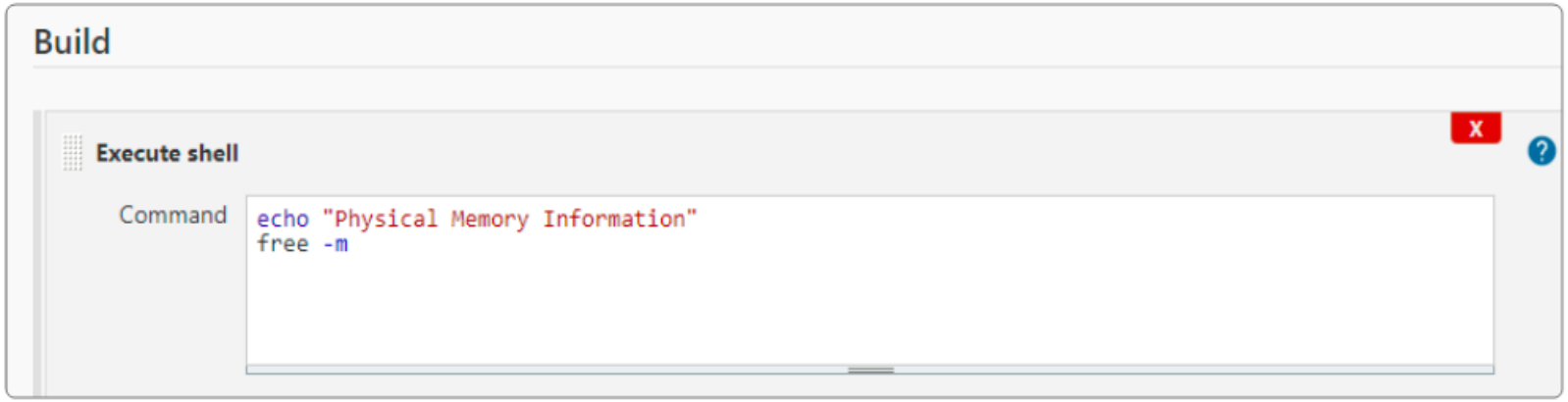
# Job Chaining in Jenkins – Job1 Configuration



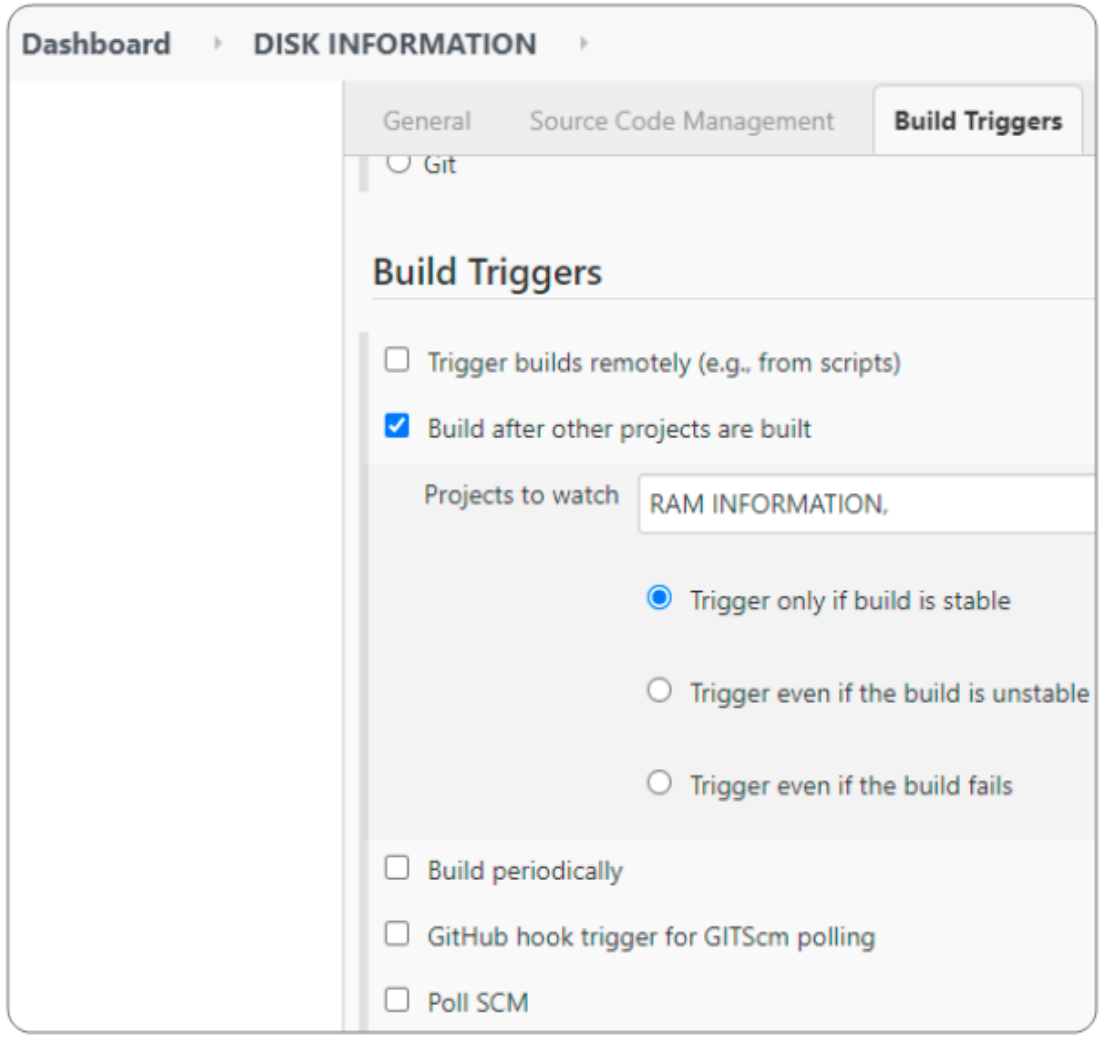



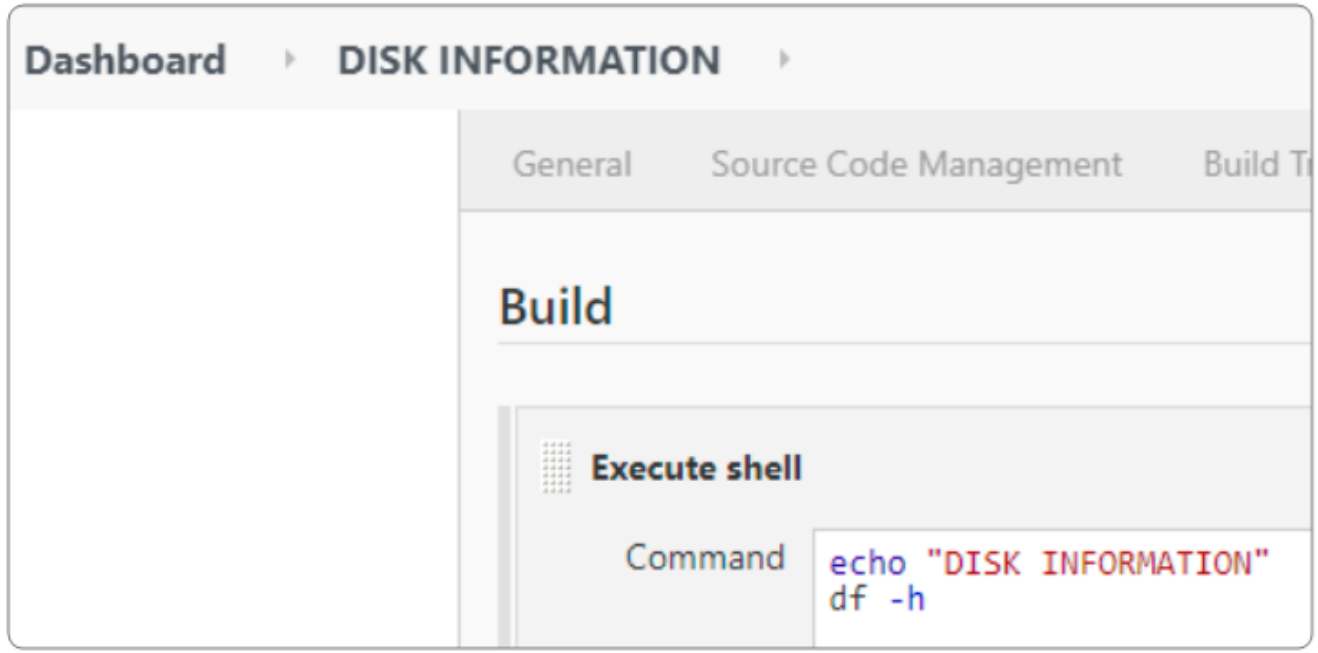
# Job Chaining in Jenkins – Job2 Configuration



# Job Chaining in Jenkins – Job3 Configuration

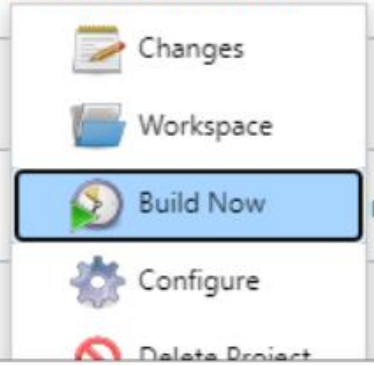


# Build Job1 – CPU INFORMATION

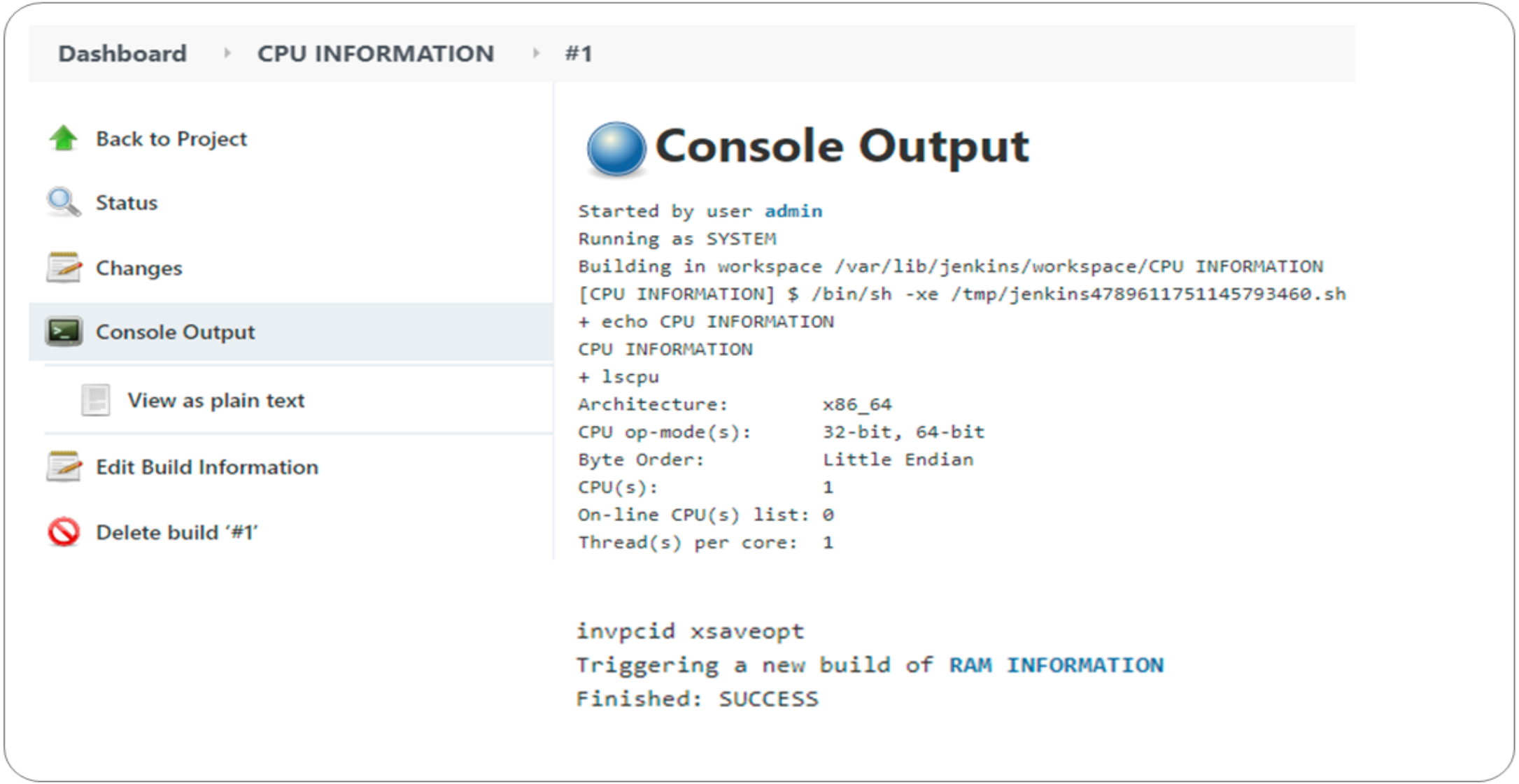
| S | W | Name ↓ |
|---|---|---|
| | | CPU INFORMATION |
| | | DISK INFORMATION |
| | | HelloWorld Python Program |
| | | RAM INFORMATION |
| | | test1 |

| S | W | Name ↓ | Last Success | Last Failure | Last Duration | |
|---|---|---|---|---|---|---|
| | | CPU INFORMATION | 20 min - #1 | N/A | 18 ms | |
| | | | 20 min - #1 | N/A | 31 ms | |
| | | ram | 2 hr 6 min - #15 | 4 hr 4 min - #8 | 43 ms | |
| | | | 20 min - #1 | N/A | 26 ms | |

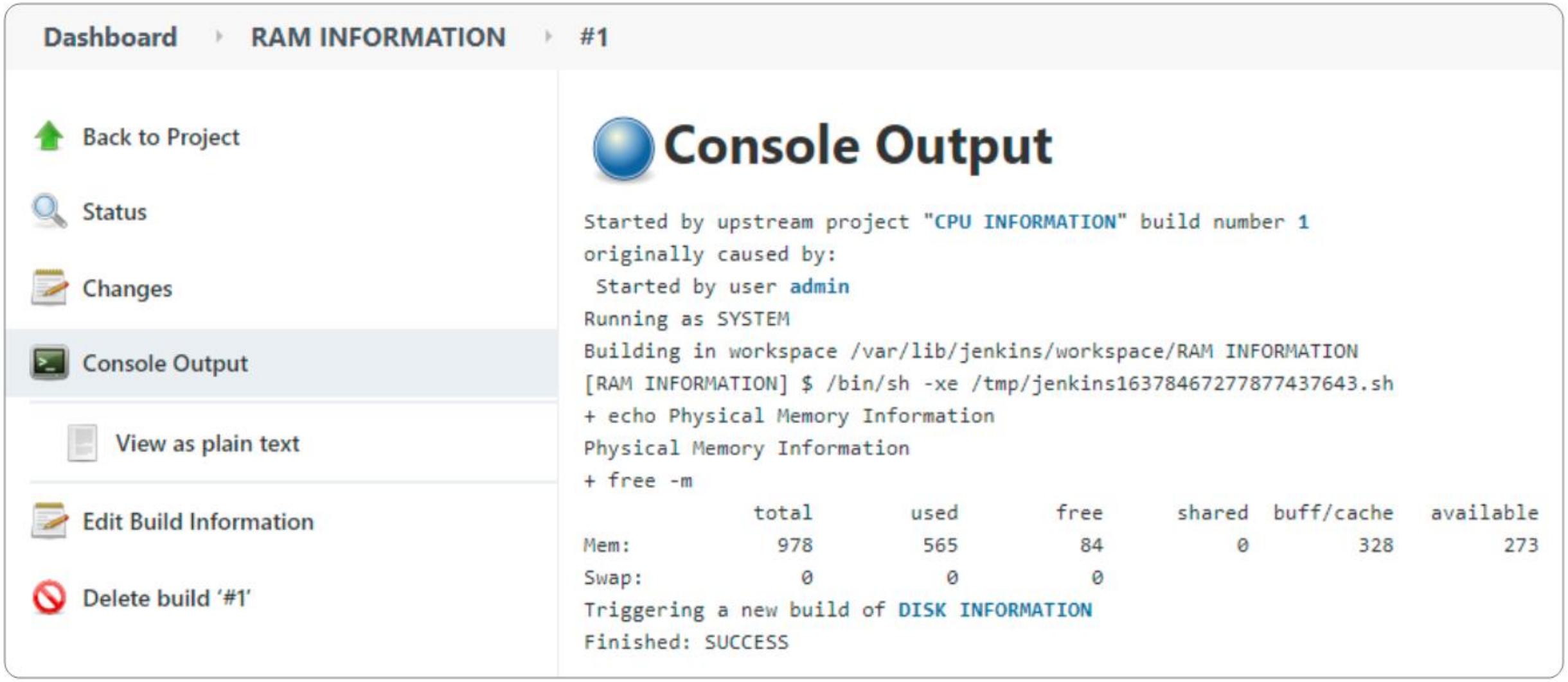


# Monitor Job Chaining Builds

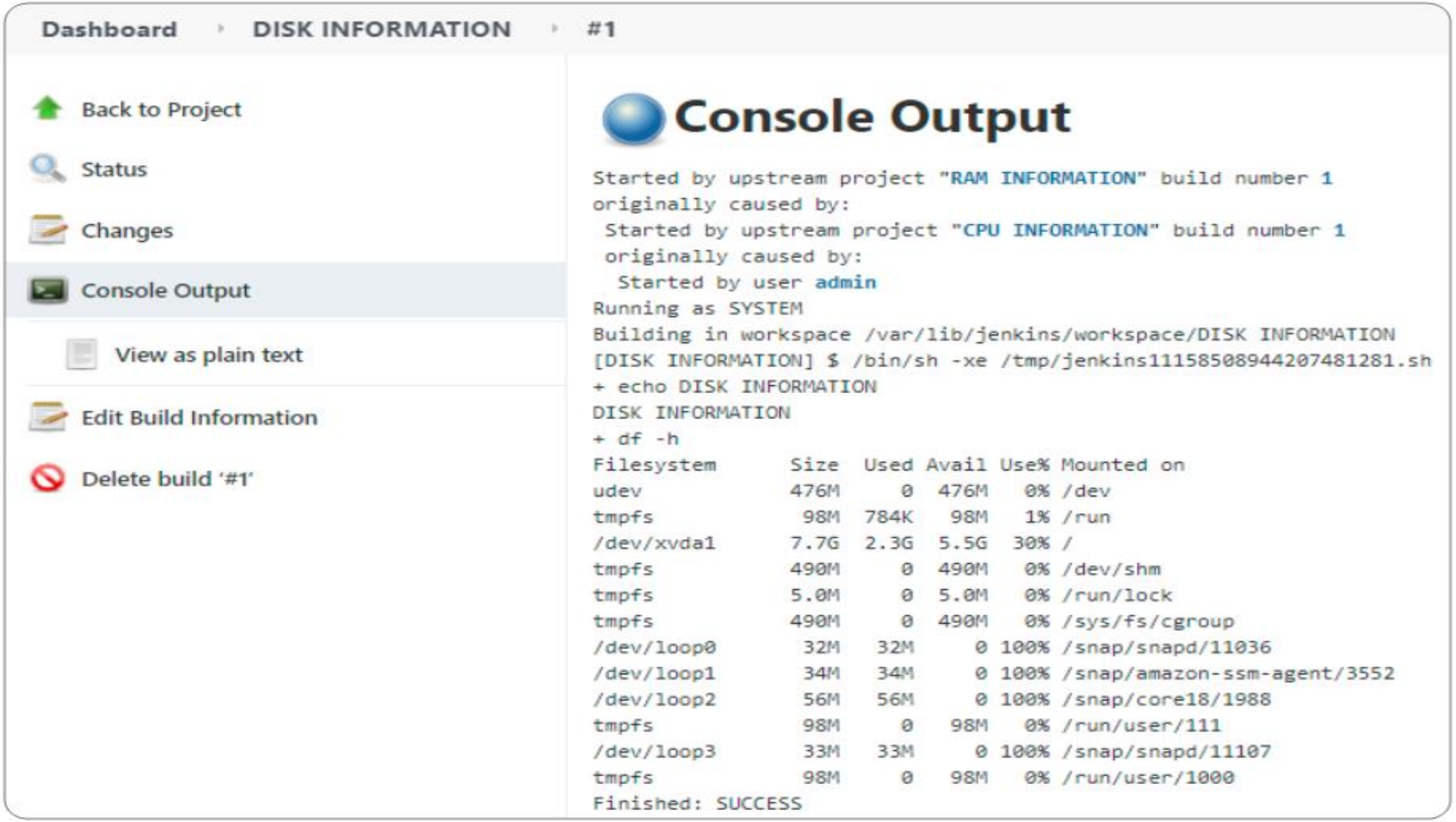
# CPU INFORMATION – Console Output



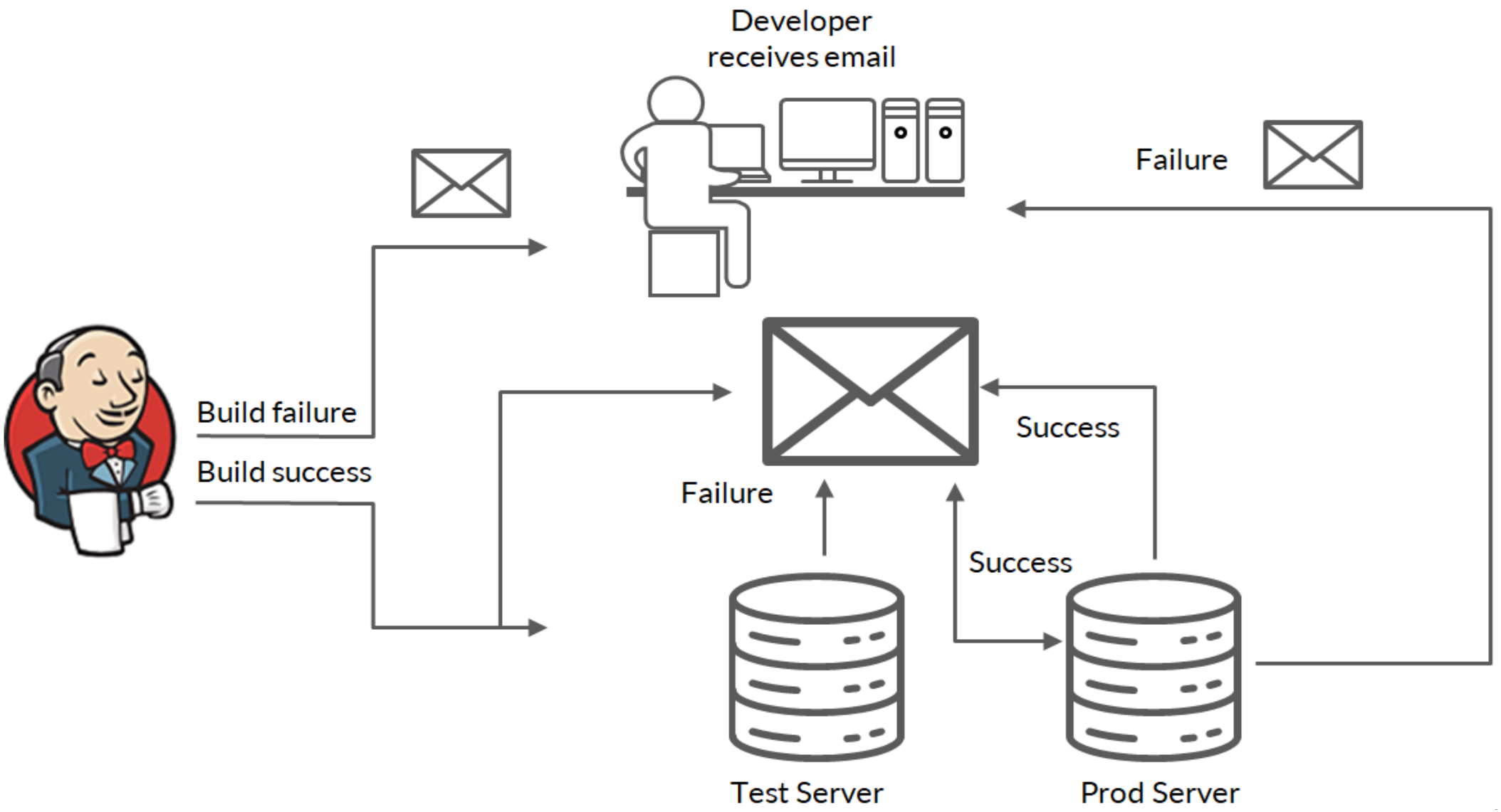
# RAM INFORMATION – Console Output



# DISK INFORMATION – Console Output



# Email Notification

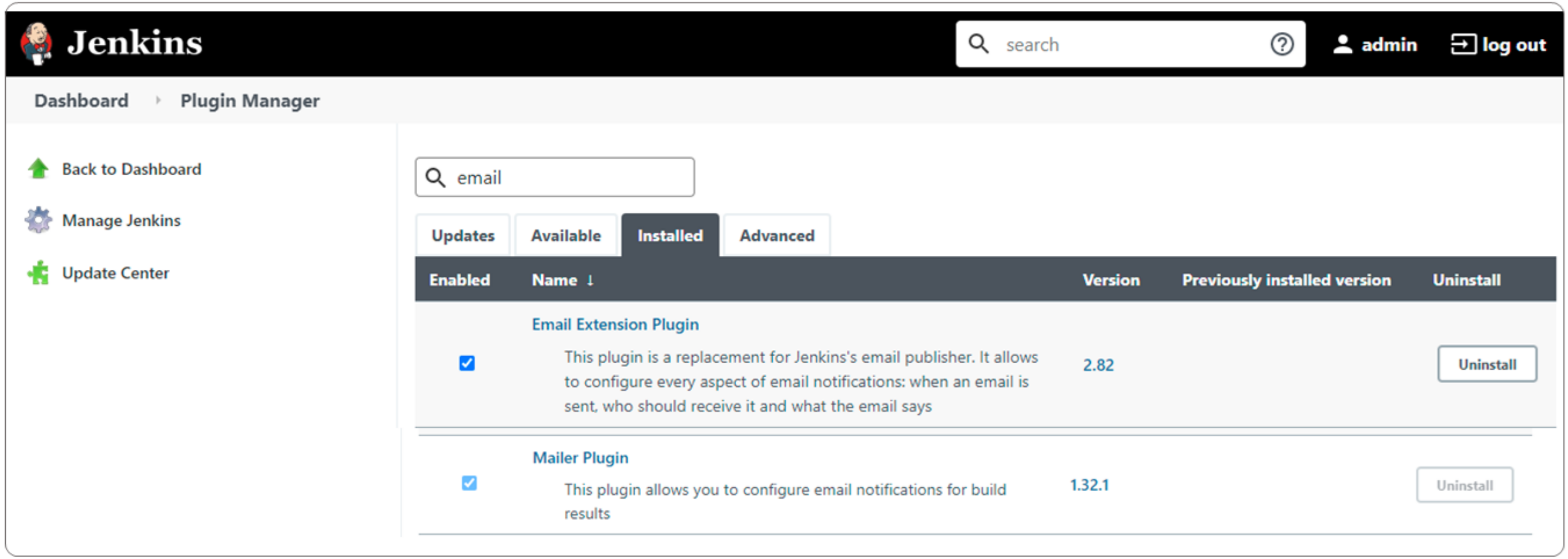


# Jenkins Email Notification Configuration

- If build failed or the build status changed from failed to success, then Jenkins has a provision to send an email notification to the given recipients.

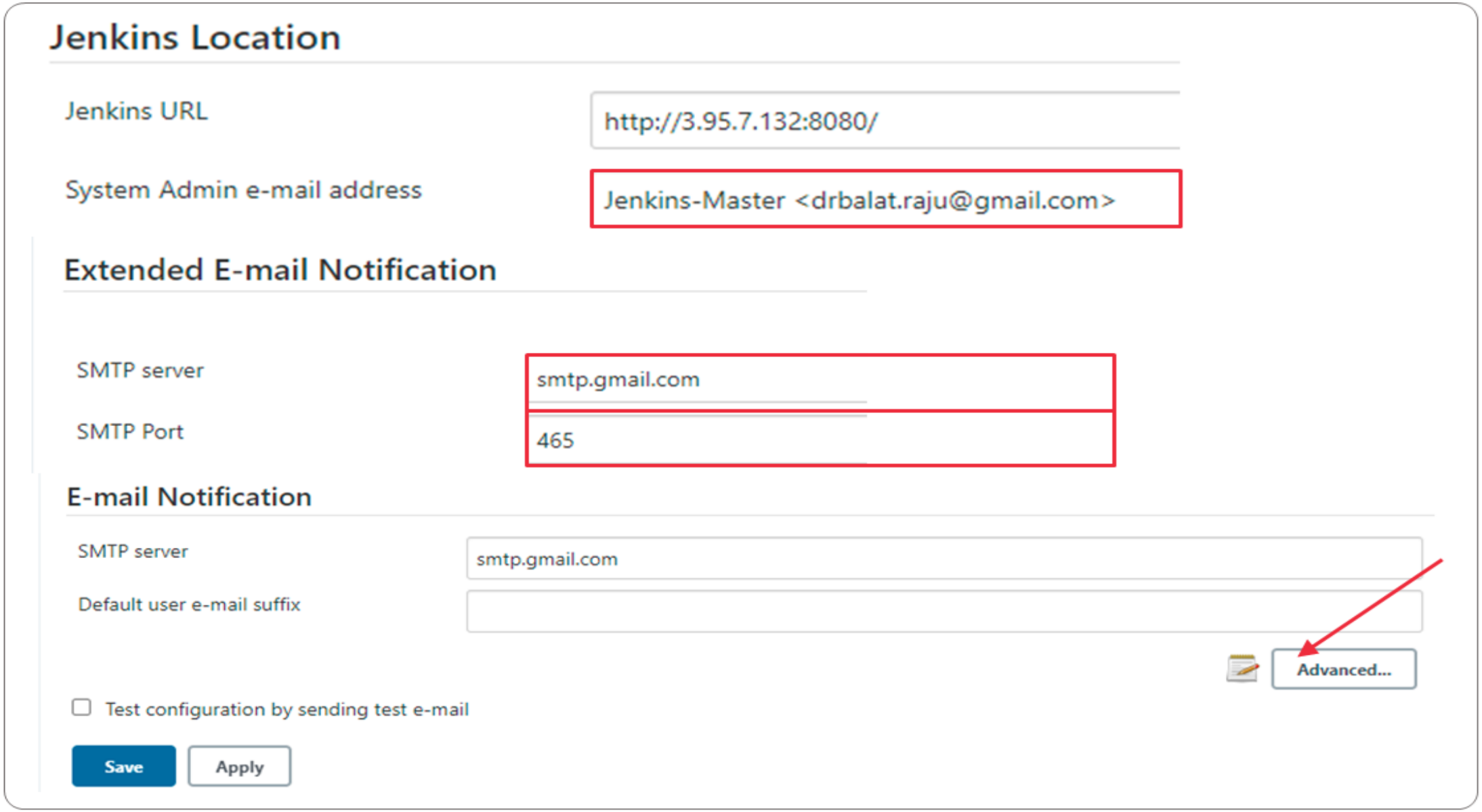
- Install Email Extension Plugin and Mailer Plugin

# Email Configuration Setup

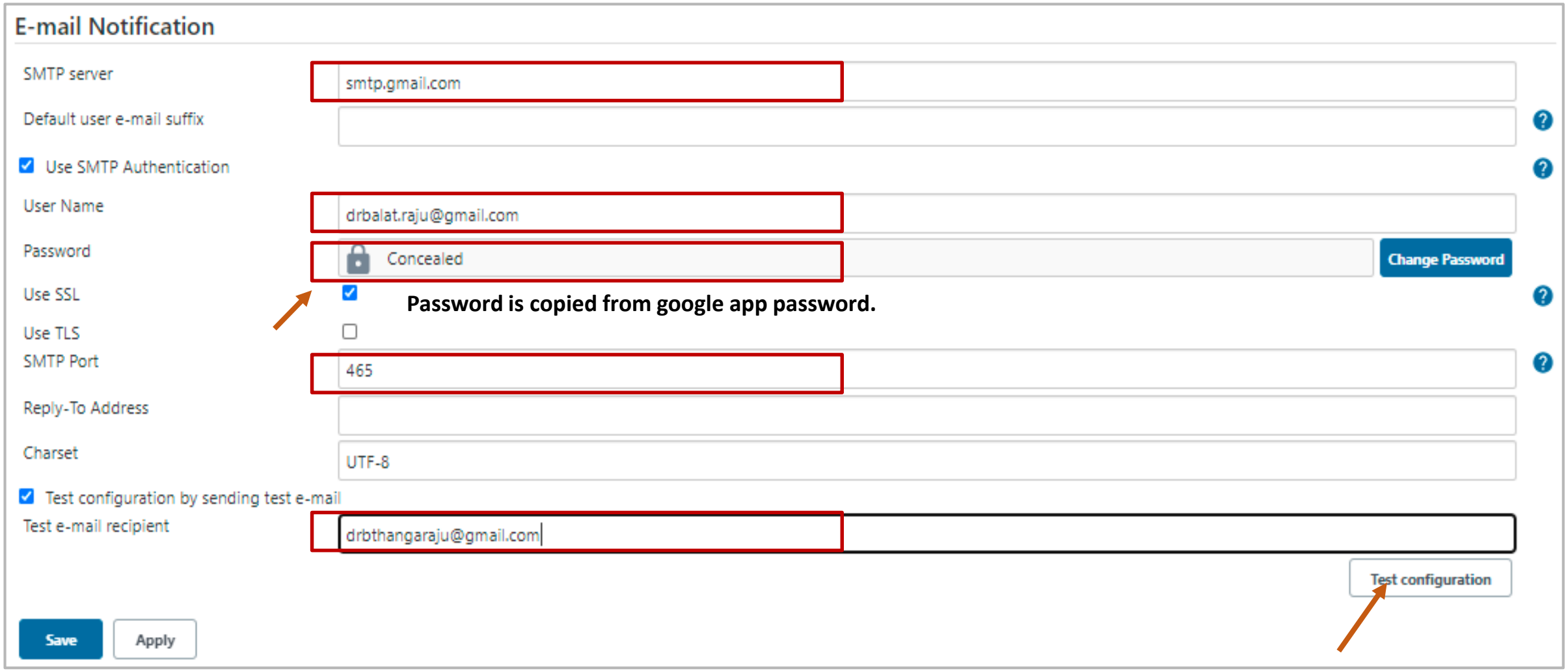



**email configuration setup** → **Manage Jenkins** → **Configure System**
Configure global settings and paths.

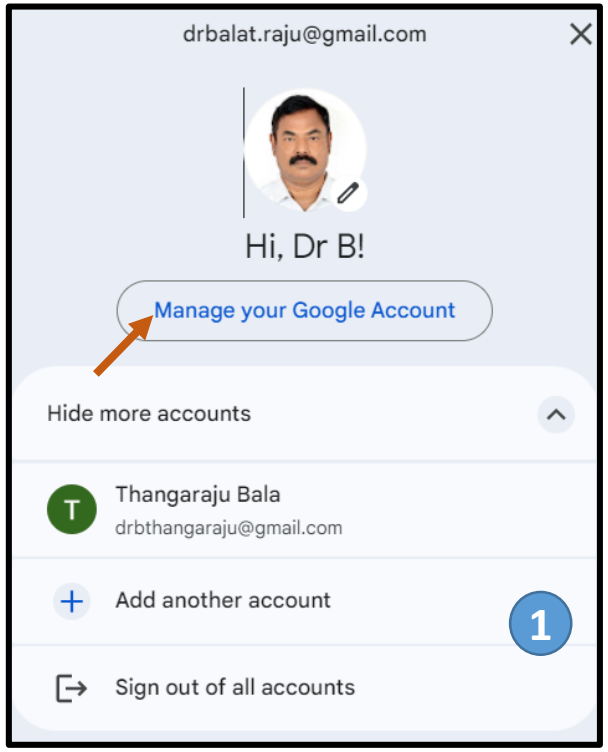
# Email Configuration Setup

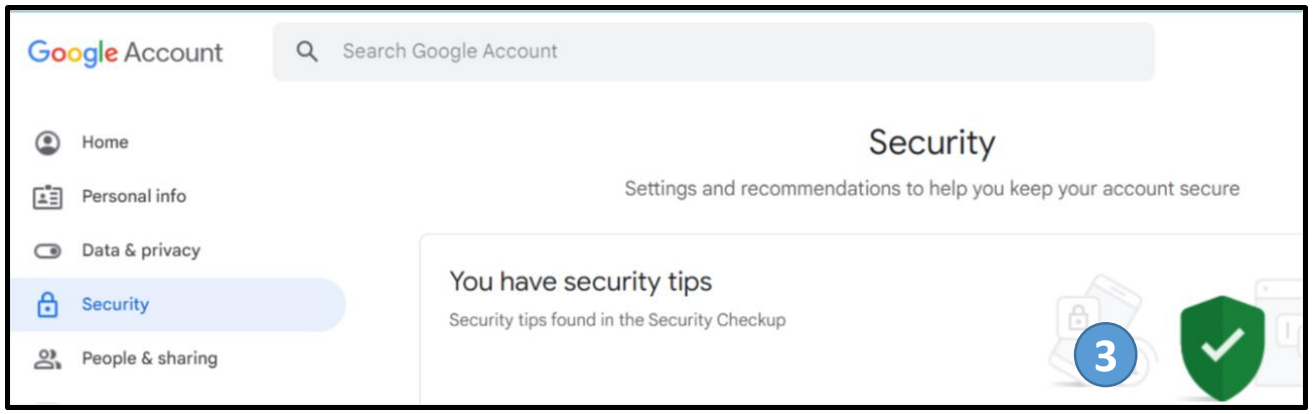


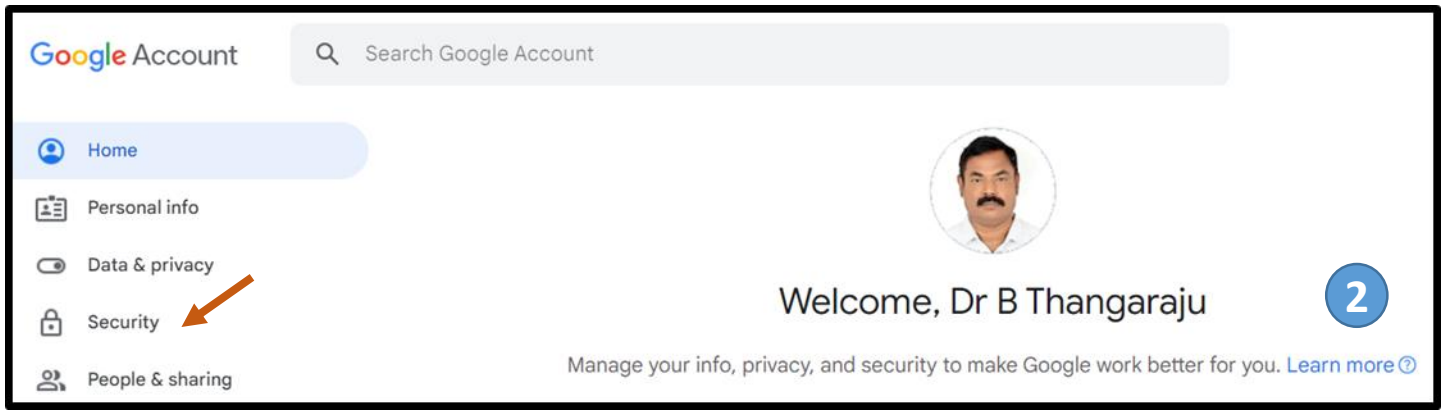
# Email Configuration Setup

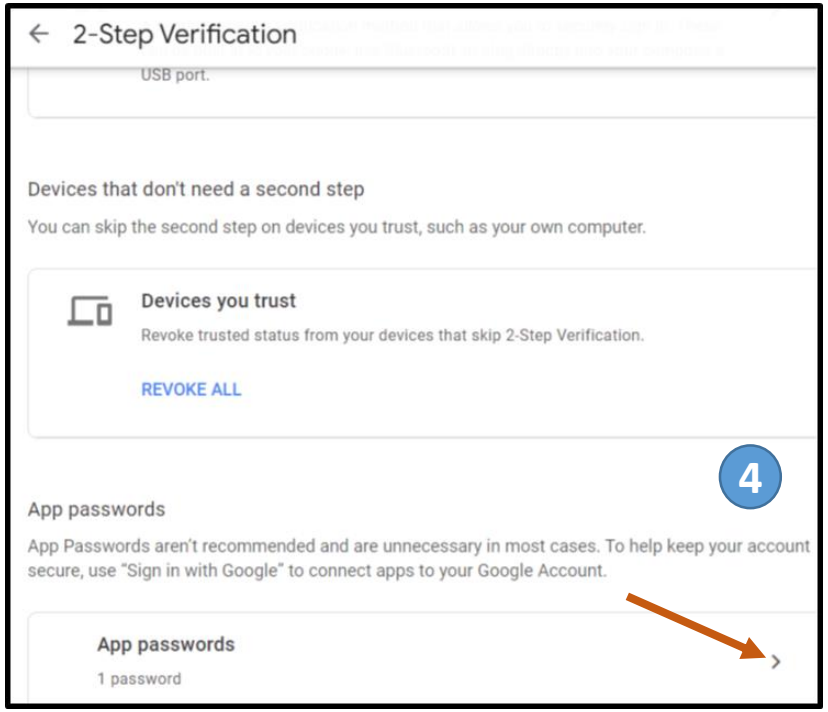


# Configure Jenkins email notification with gmail

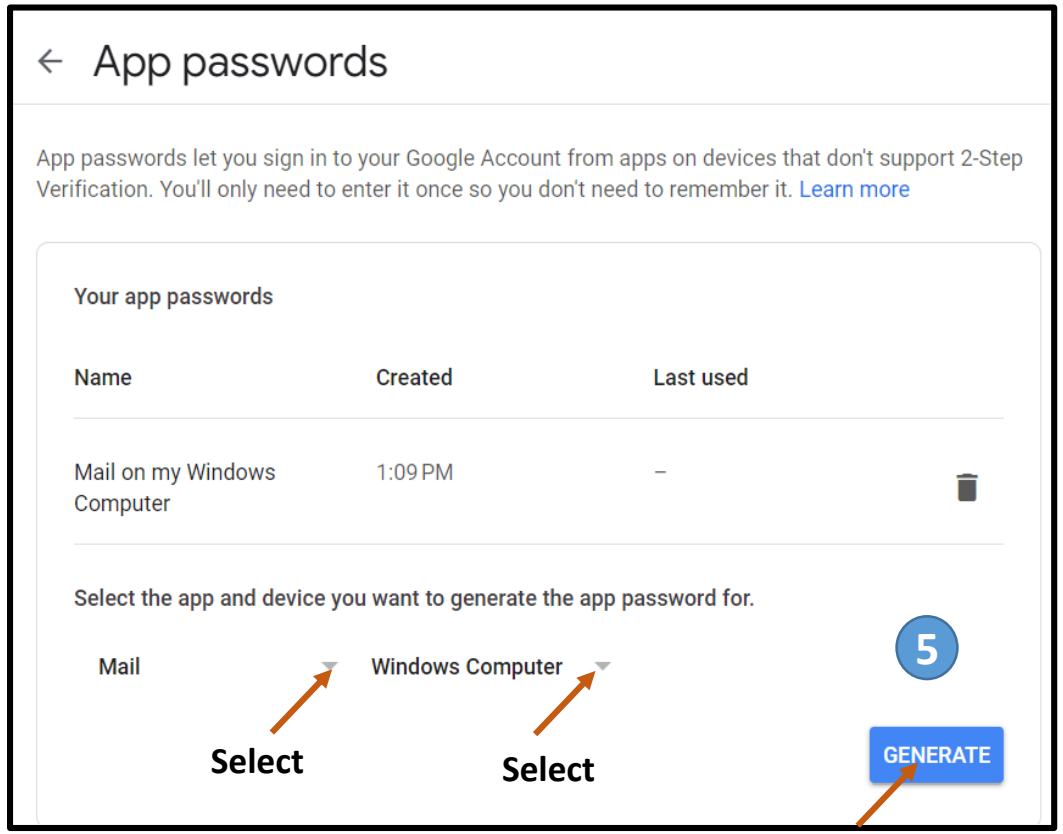


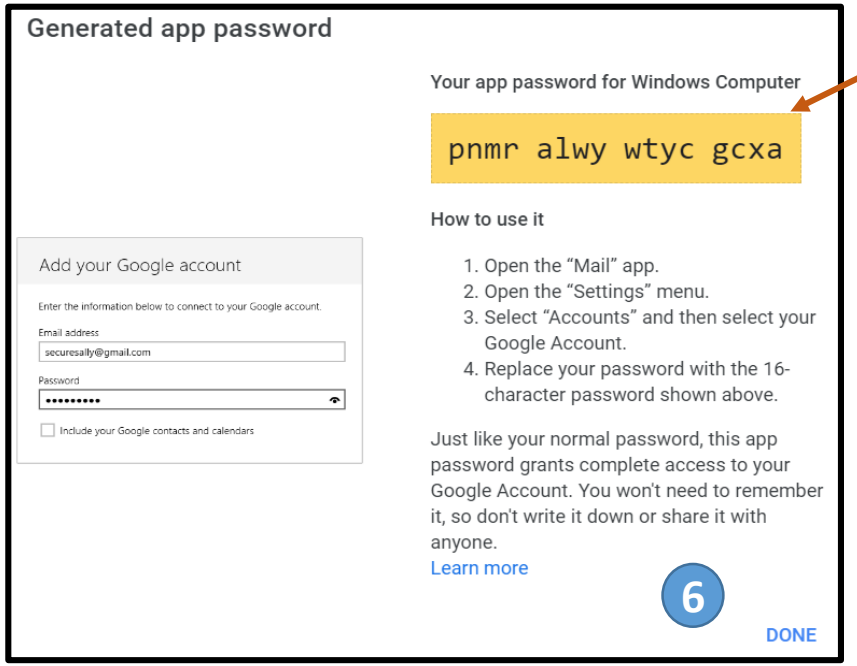



# Configure Jenkins email notification with gmail

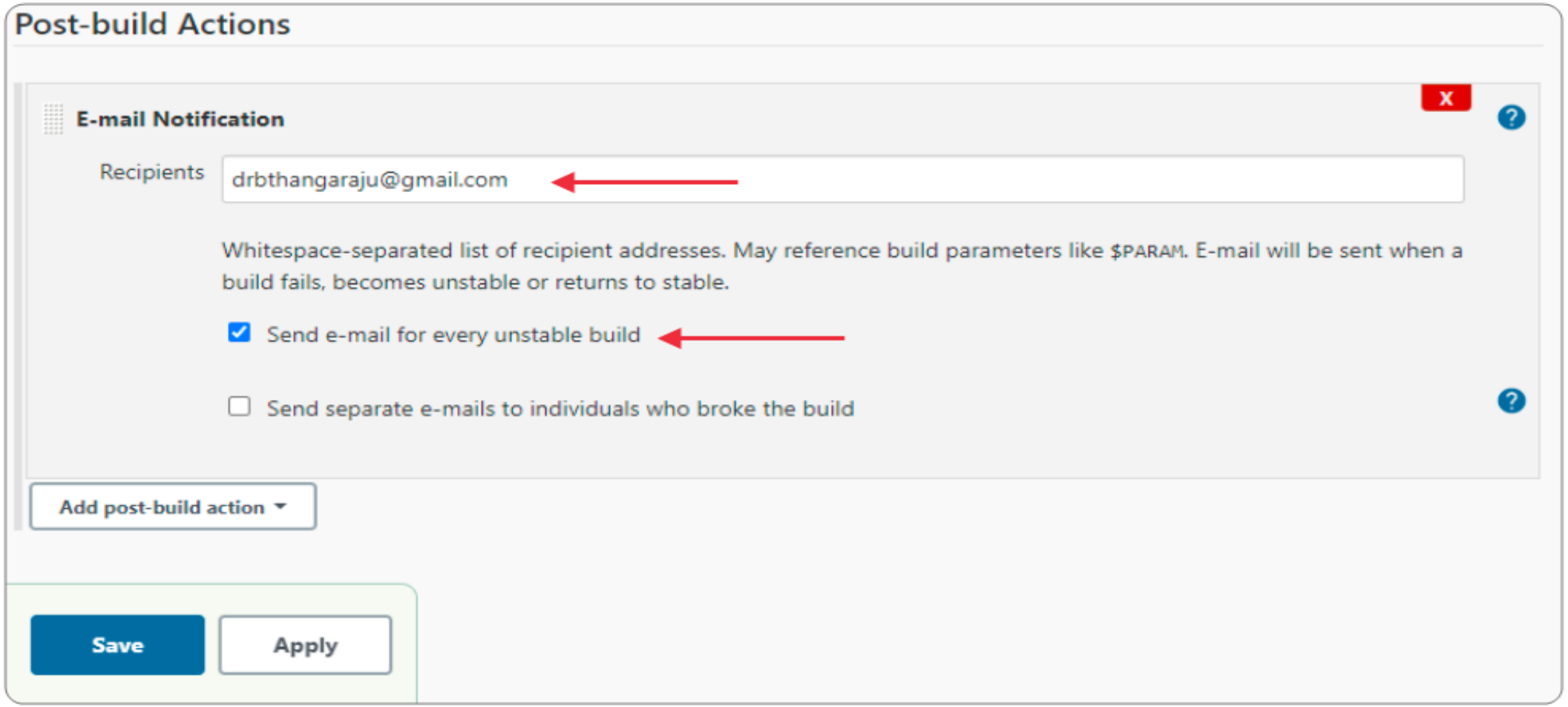


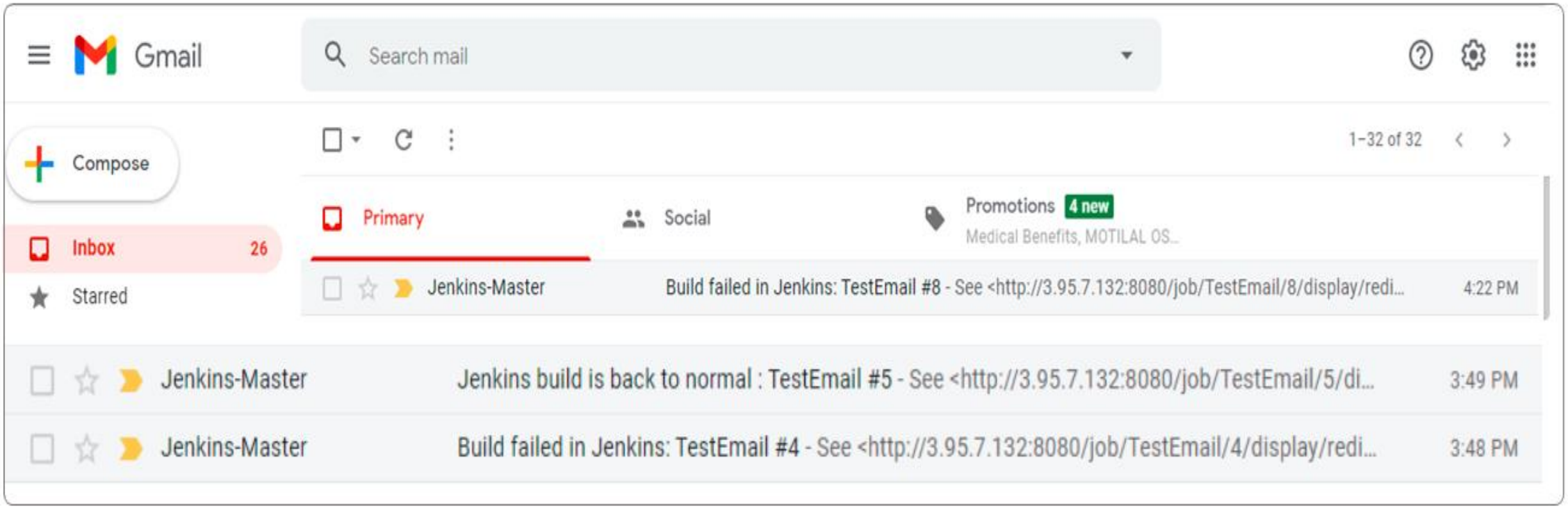

**Copy and past it in Jenkins email configuration setup**

# Post-Build Actions for a Project

Go to Dashboard
Select Project –TestEmail
Click Configure



# Email Notification



# Email Notification

## Build failed in Jenkins: TestEmail #8 Inbox ×

**Jenkins-Master** <drbalat.raju@gmail.com>
to me

See <http://3.95.7.132:8080/job/TestEmail/8/display/redirect>

Changes:

---

Started by user admin
Running as SYSTEM
Building in workspace <http://3.95.7.132:8080/job/TestEmail/ws/>
[TestEmail] $ /bin/sh -xe /tmp/jenkins8426018425159410393.sh
+ fre -m
/tmp/jenkins8426018425159410393.sh: 2: /tmp/jenkins8426018425159410393.sh: fre: not found
Build step 'Execute shell' marked build as failure

# Email Notification

Jenkins build is back to normal : TestEmail #5 Inbox ×

**Jenkins-Master** <drbalat.raju@gmail.com>

to me

See <http://3.95.7.132:8080/job/TestEmail/5/display/redirect>

## Console Output

```
Started by user admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/TestEmail
[TestEmail] $ /bin/sh -xe /tmp/jenkins11996699832483977733.sh
+ free -m
              total        used        free      shared  buff/cache   available
Mem:            978         565          70           0         342         268
Swap:             0           0           0
Sending e-mails to: drbthangaraju@gmail.com
Finished: SUCCESS
```

# Parameterized Jenkins Job

# Parameterized Project

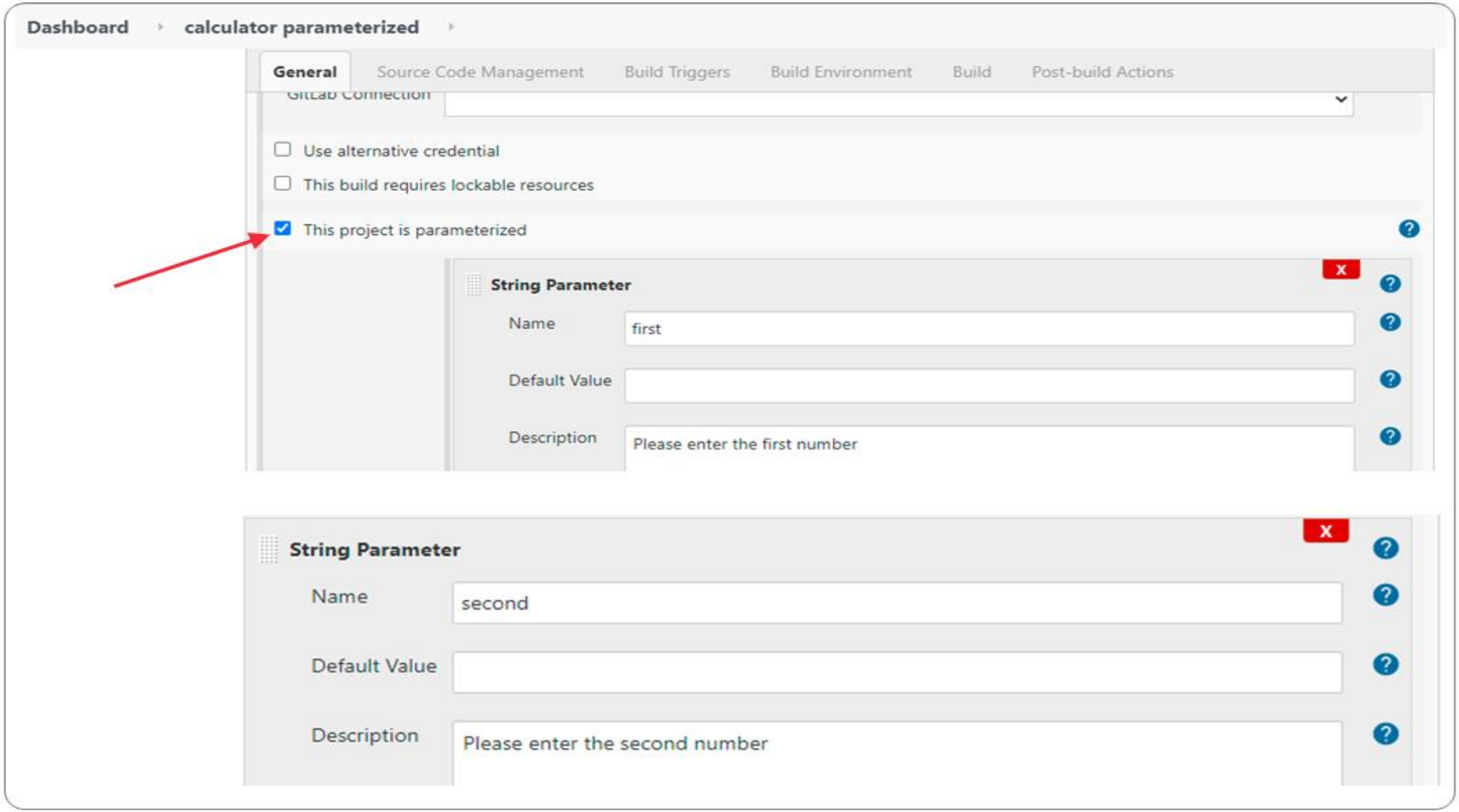
```
# !/bin/bash
# A simple calculator shell program

a=$1
b=$2

echo "Enter Choice :"
echo "1. Addition"
echo "2. Subtraction"
echo "3. Multiplication"
echo "4. Division"
ch=$3

case $ch in
  1)res=`expr $a + $b`
  ;;
  2)res=`expr $a - $b`
  ;;
  3)res=`expr $a \* $b`
  ;;
  4)res=`expr $a / $b`
  ;;
esac
echo "Result : $res"
```

Step1. Create a freestyle project: **calculator parameterized**

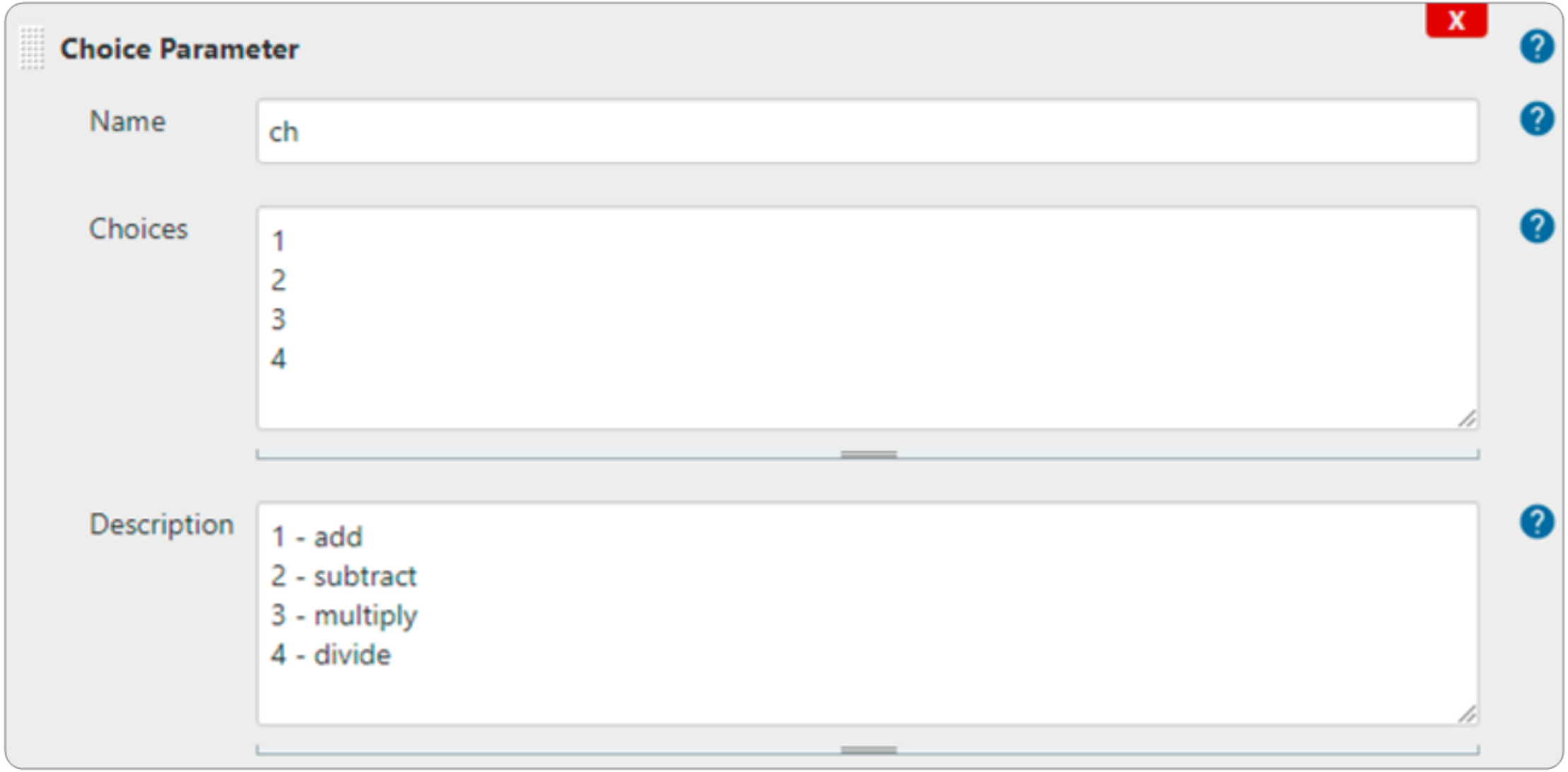
Step 2. Develop a simple calculator shell program in the default workspace

Step 3. During execution, you can choose the a, b and choice through Jenkins 'This project is parameterized' option

# Enter the First and Second Parameters



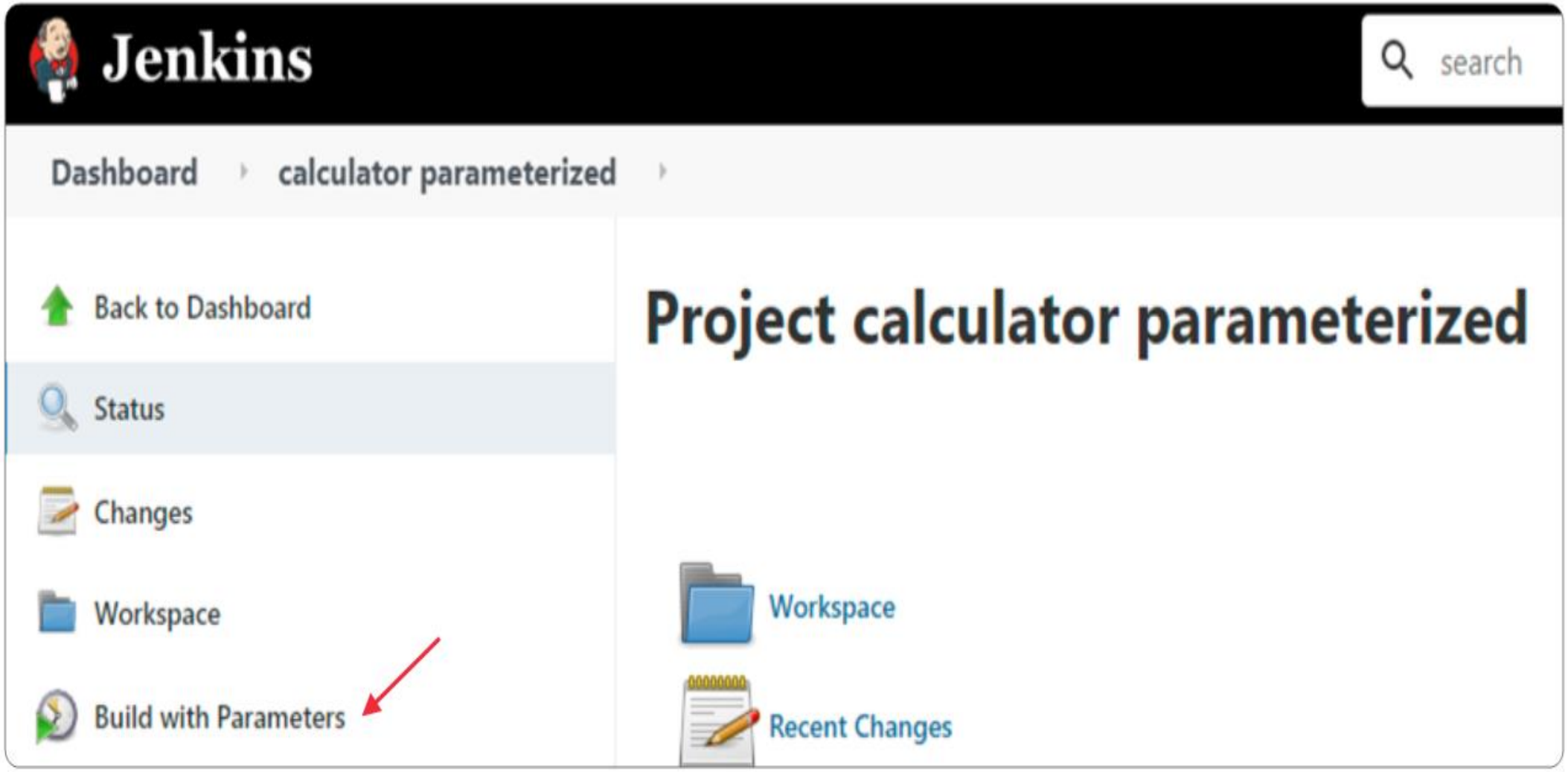
# Choice Parameter



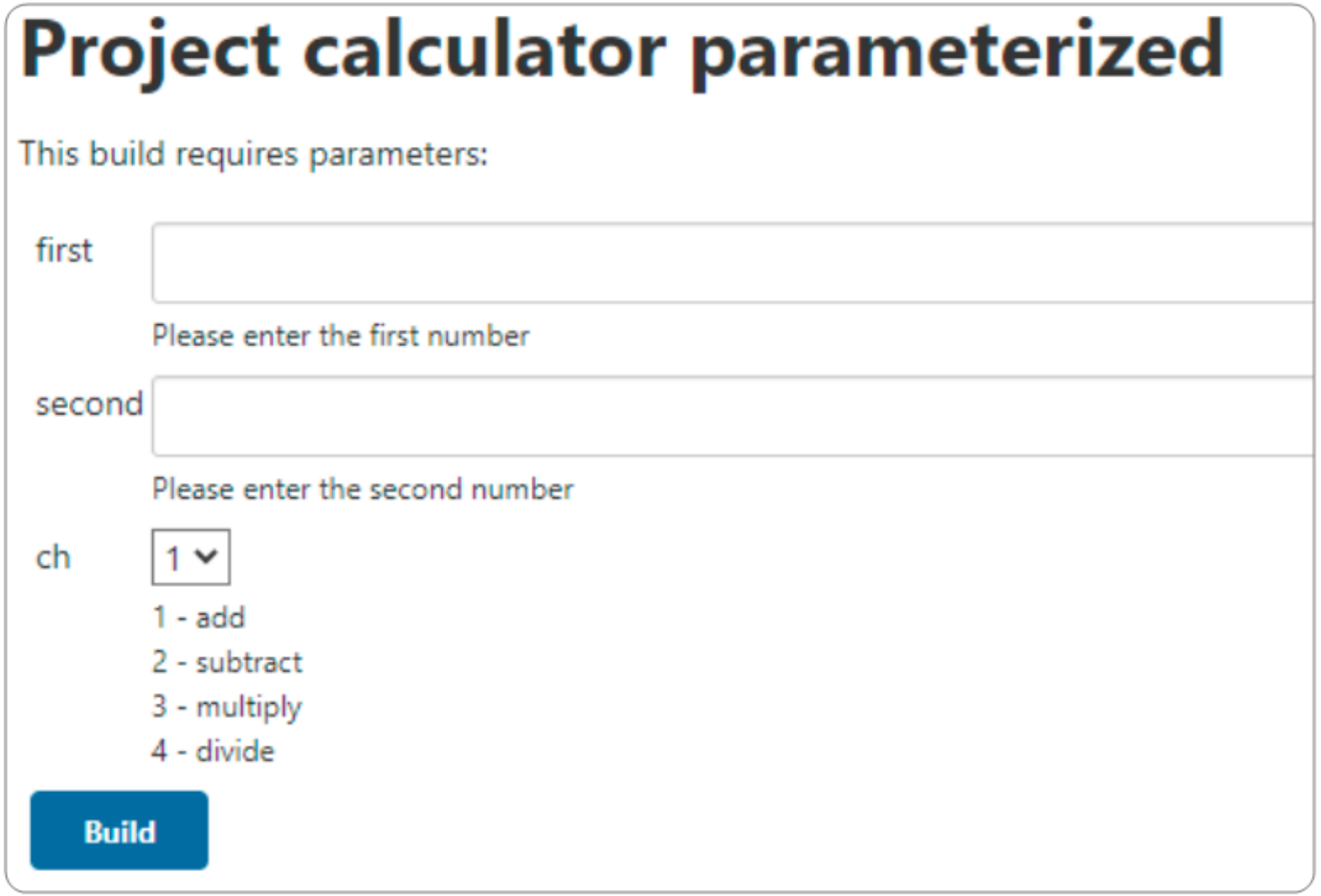
# Build Step

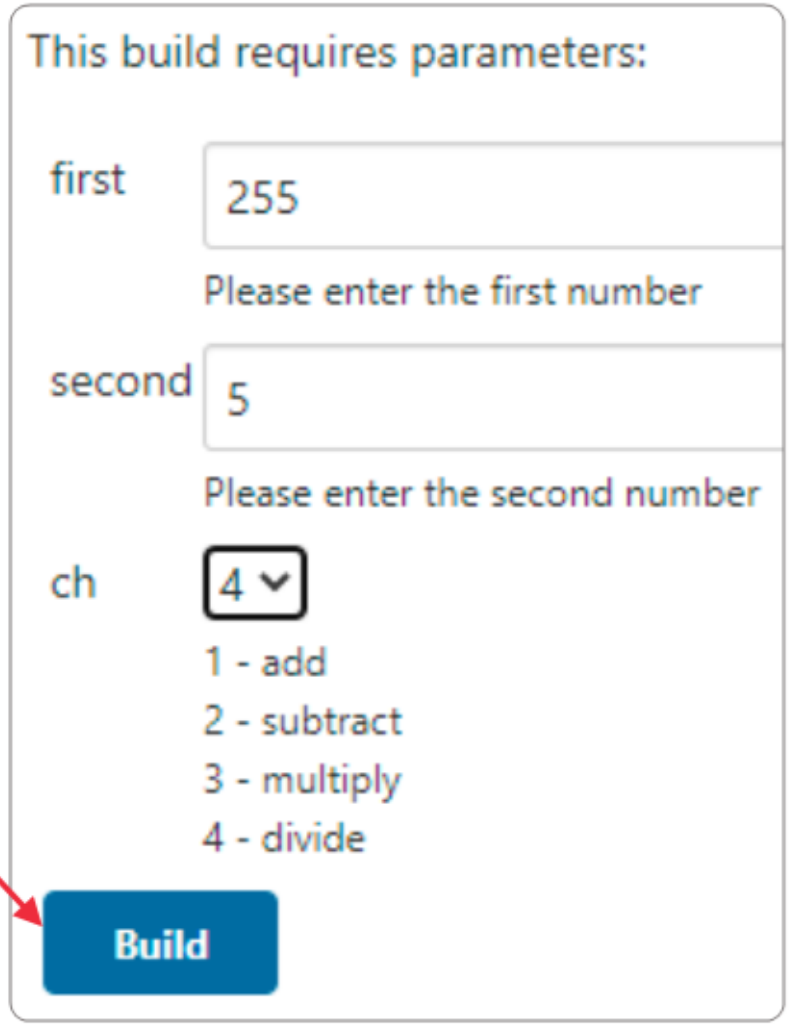

Build
Execute shell
X
Command
chmod u+x c.sh
./c.sh ${first} ${second} ${ch}
See the list of available environment variables
Advanced...
Add build step
Save
Apply

# Build Job With Parameter

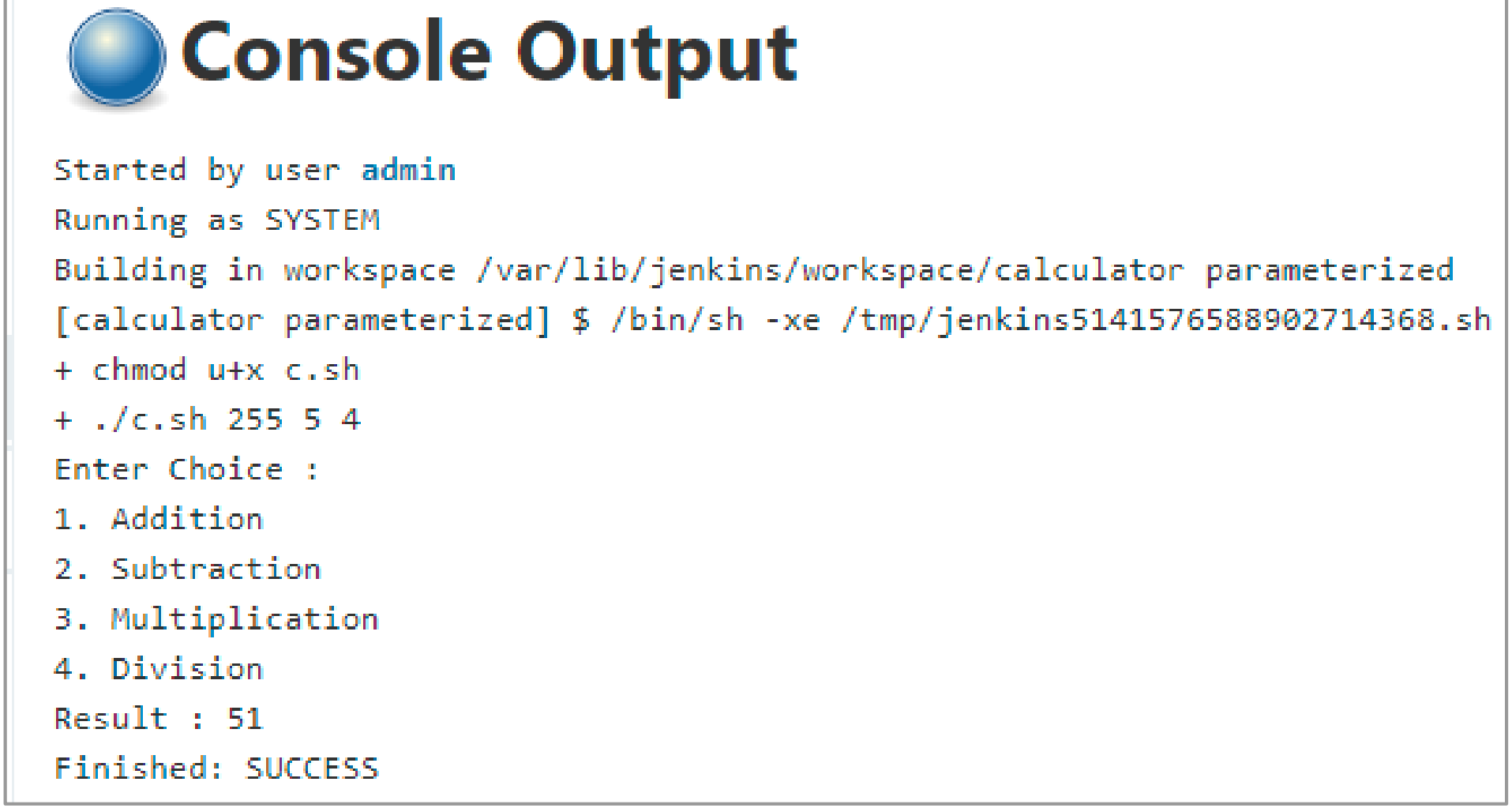


# Variables Values and Select Choice



# Console Output



# Understand How to Manage Users in Jenkins

# Importance of User Management

- In a typical project environment, many employees working on a project can access the Jenkins server to run their build or test jobs. This can create security and authorization issues.
- So, you need to give every Jenkins user appropriate permission to enable the Jenkins server's safety and security.
- In Jenkins, different configuration options are available to enable, edit or disable various security features.
- By default, anonymous users have no permissions and logged in users have complete control.

# Importance of User Management

- Jenkins admin manages these users based on their roles. Jenkins provides capabilities to add users, edit users and provide different roles to each user. For this, Jenkins provides a role-based authentication plugin.

- The ‘Configure Global Security’ option helps a Jenkins administrator to enable, configure or disable key security features to the Jenkins environment.

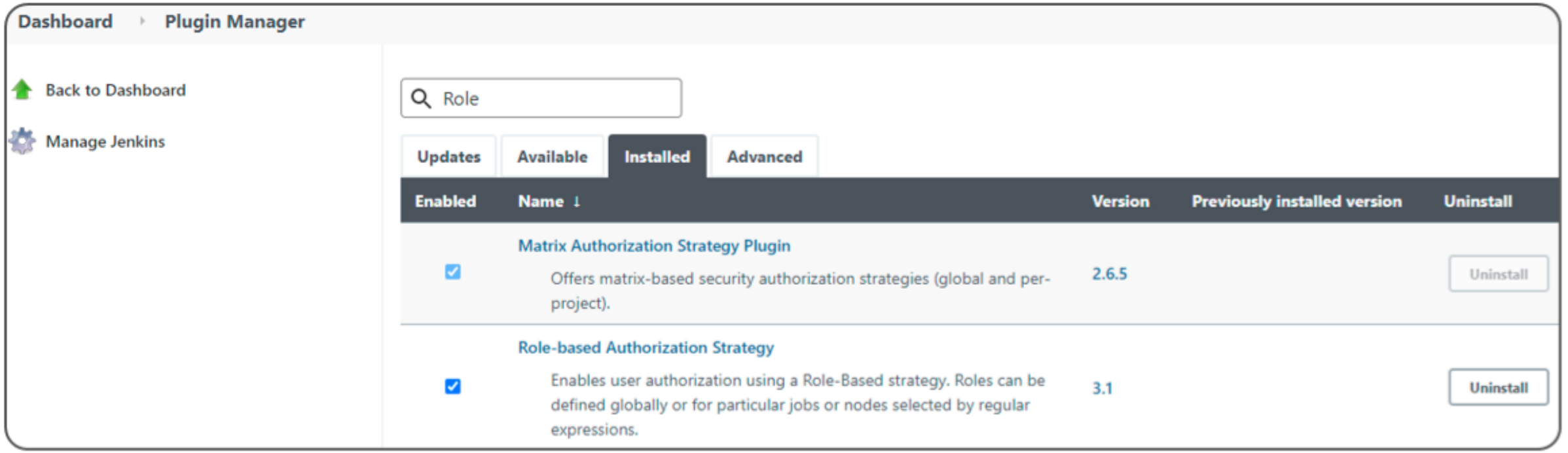
# Security Realm

- To configure authentication and authorization schemes in Jenkins, you need to use Security Realm and Authorization configurations.
- Security Realm informs the Jenkins environment how and from where to pull user information.
- Authorization configuration informs the Jenkins environment about which users can access which aspects of Jenkins and to what extent.
- The Security Realm/authentication specifies who can access the Jenkins environment, whereas Authorization specifies what they can access.
- Matrix-based security allows the administrator a granular control over assigning users:
  - Provides the most security and flexibility
  - Recommended for production environments

# Role-Based Authentication Strategy

- Used to add a new role-based mechanism to manage users' permission
- Roles can be assigned to users and user groups
- Global Roles: Admin, Developer, Tester, QA and Anonymous
- Allow to set permission: Agent, Job, Run, View and SCM
- Project/Item roles: Allow additional access control for each project separately in the Project configuration screen and give access control to specific user to access only the specified projects
- Agent roles: Set node-related permissions

# Required Plugins



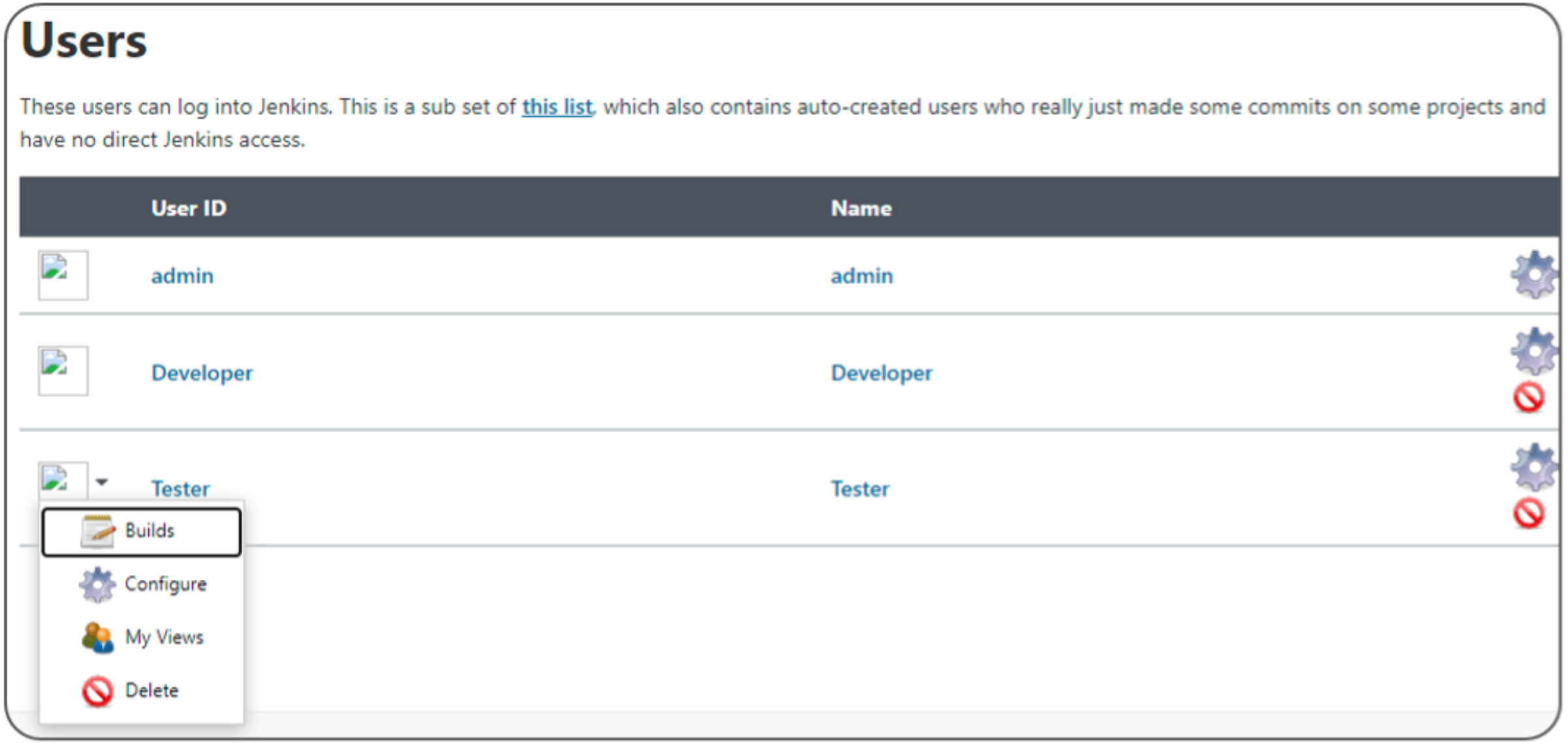
# Demo

- Create two users: Developer and Tester
- Configure Global Security: Enable Role-based Authentication strategy
- Manage and Assign roles:
  - Manage roles: Create a Global Roles - add ProjectMember and enable only required access
  - Item roles: Create two roles - Developer and Tester. Enable all the options.
  - Manage and Assign roles: Add Developer and Tester as ProjectMember
  - Item roles: Developer and Tester users should be assigned to Developer and Tester roles, respectively. Set pattern as Prog.* for Developer and Test.* for Tester
- Create two projects: Program1 and TestProject1
- Login as Developer or Tester and view/build/create/delete the Projects

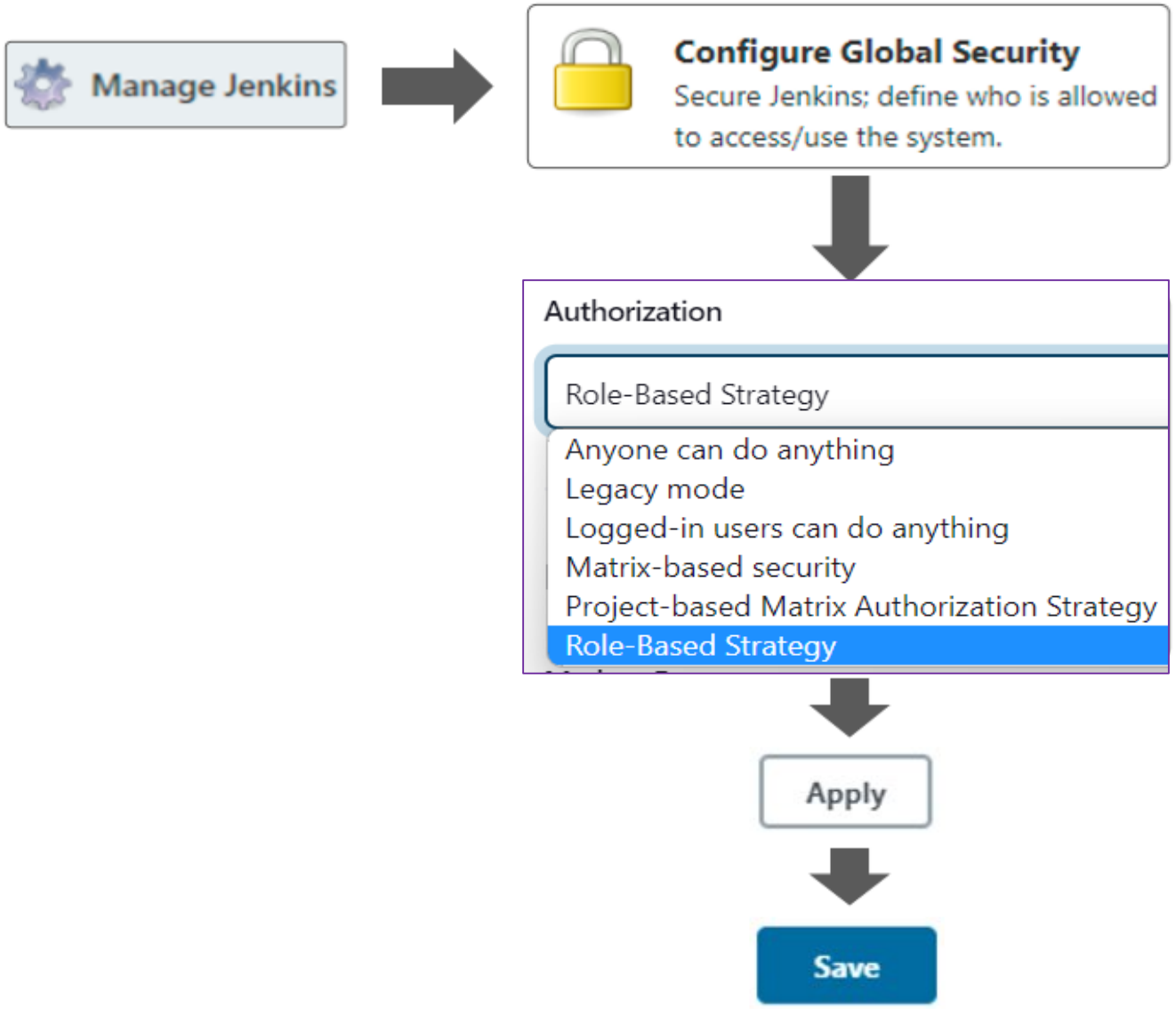
# User Account Creation



# List and Delete User

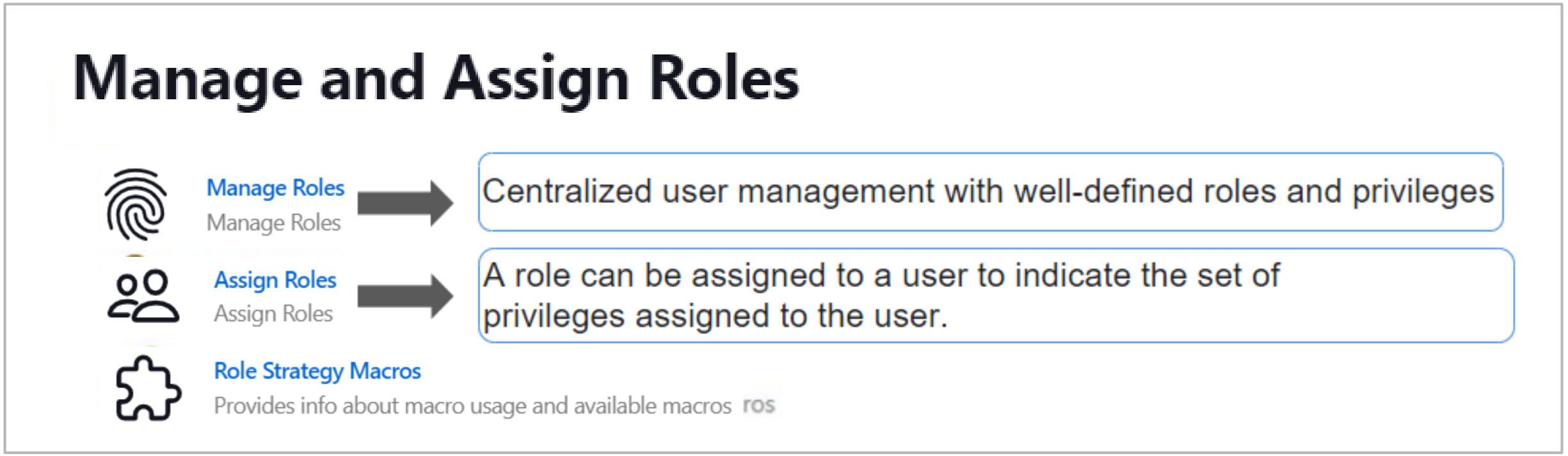


# Enable Role-Based Authentication Strategy



# Manage and Assign Roles

**Handle permissions by creating roles and assigning them to users/groups**



# Manage Roles

Global roles

| Role | Overall | | Credentials | | | | | Agent | | | | | | | Job | | | | | | | | | Run | | | View | | | | SCM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Administer | Read | Create | Delete | ManageDomains | Update | View | Build | Configure | Connect | Create | Delete | Disconnect | Provision | Build | Cancel | Configure | Create | Delete | Discover | Move | Read | Workspace | Delete | Replay | Update | Configure | Create | Delete | Read | Tag |
| ProjectMember | ☐ | ☑ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☑ | ☐ | ☑ | ☑ | ☑ | ☐ | ☐ | ☐ | ☐ | ☑ | ☐ | ☑ | ☐ | ☐ | ☐ | ☐ | ☐ |
| admin | ☑ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ |

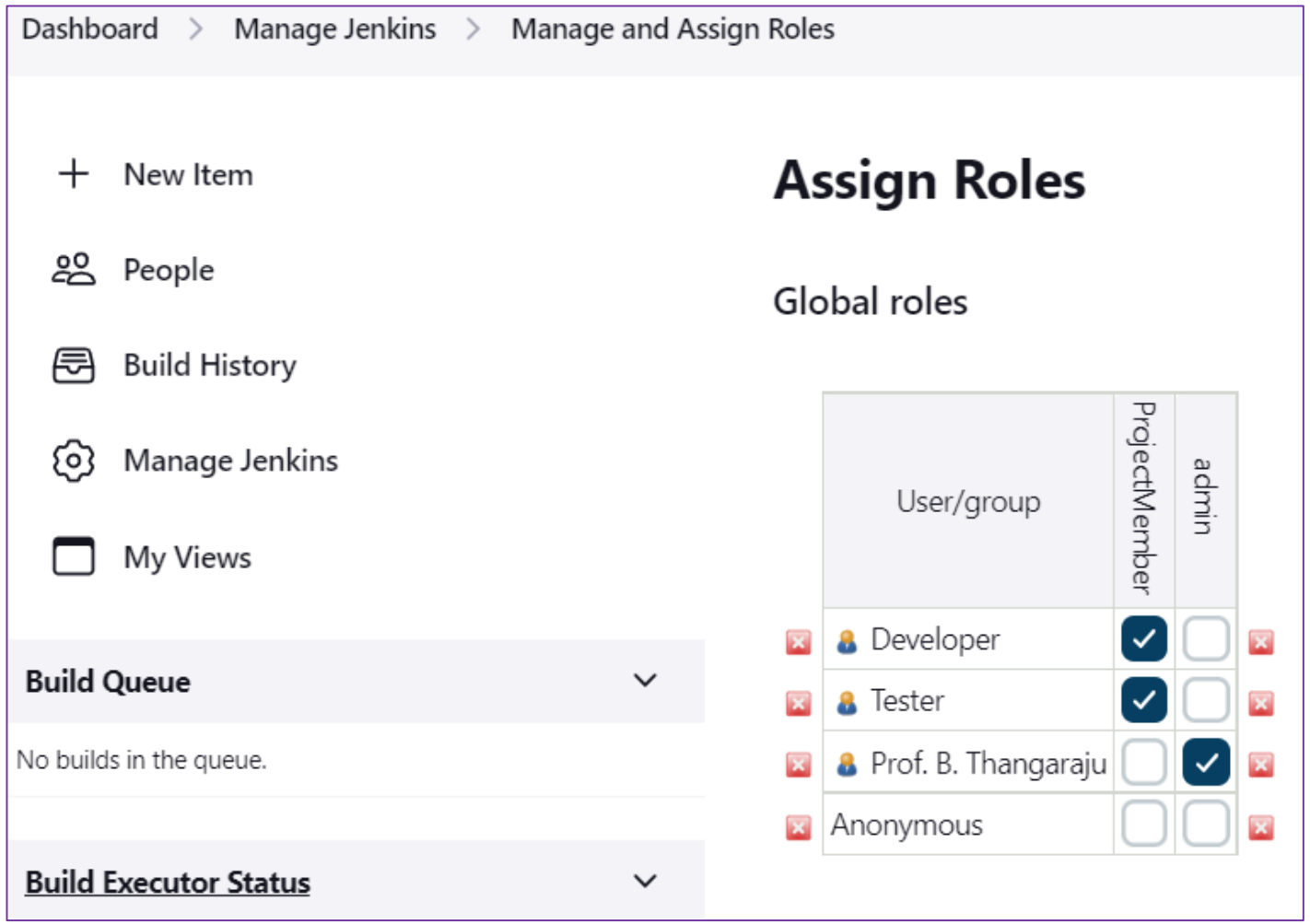
**Add ProjectMember role in Global roles and enable required permissions**

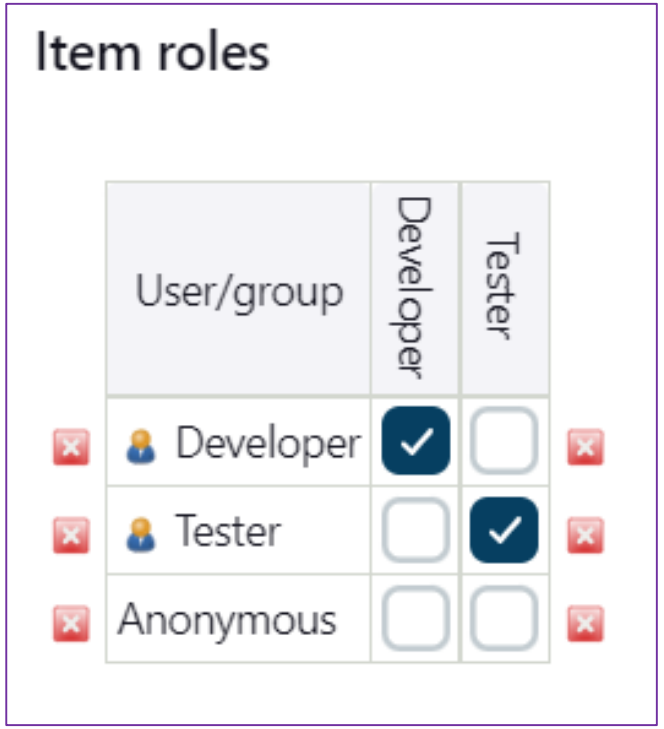
Item roles

| Role | Pattern | Credentials | | | | | Job | | | | | | | | | Run | | | View | | | | SCM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Create | Delete | ManageDomains | Update | View | Build | Cancel | Configure | Create | Delete | Discover | Move | Read | Workspace | Delete | Replay | Update | Configure | Create | Delete | Read | Tag |
| Developer | "Prog.*" | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☐ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ |
| Tester | "Test.*" | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ | ☐ | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ |

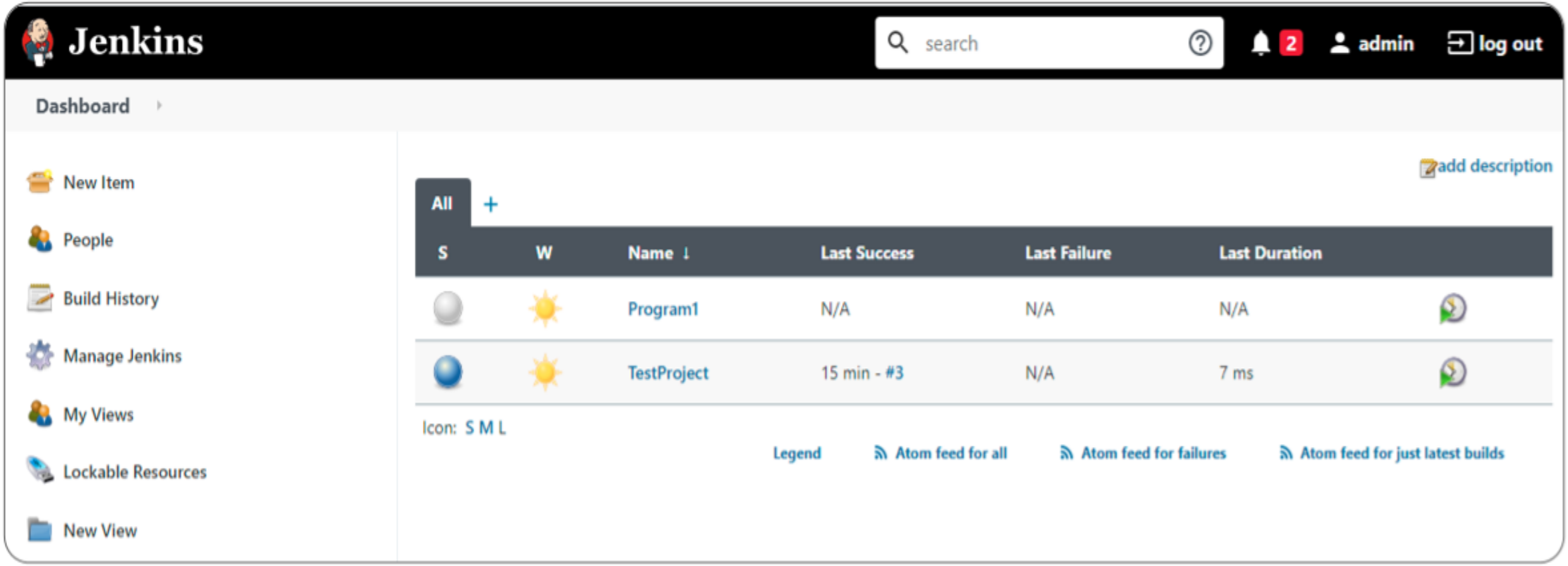
**Add Developer and Tester roles and set pattern as Prog.* for Developer and Test.* for Tester.**

# Assign Roles: Global and Item Roles

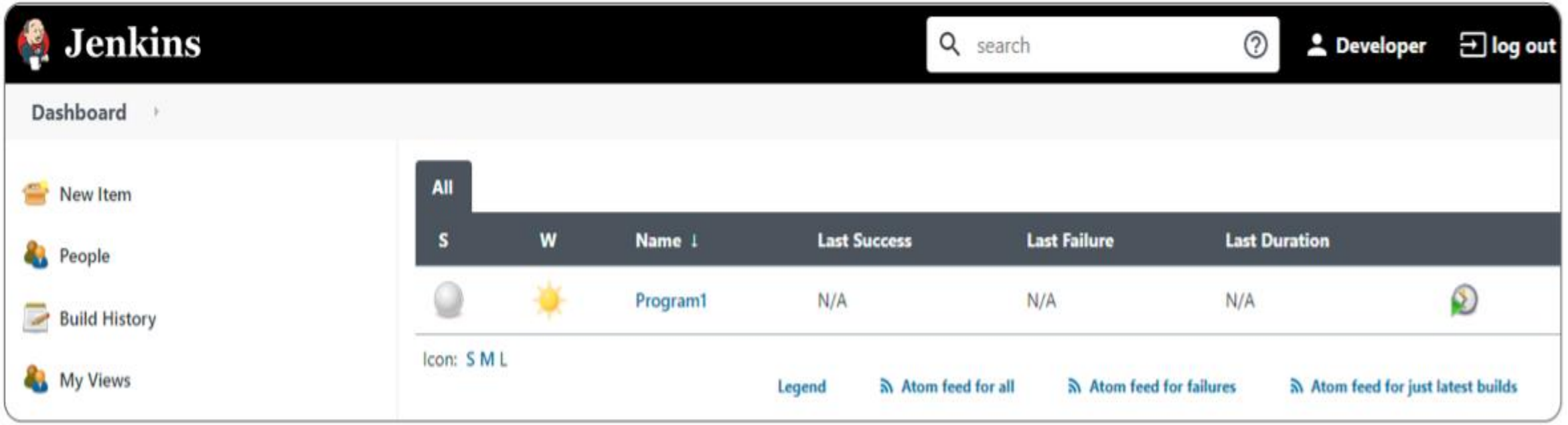


# List of Projects Created

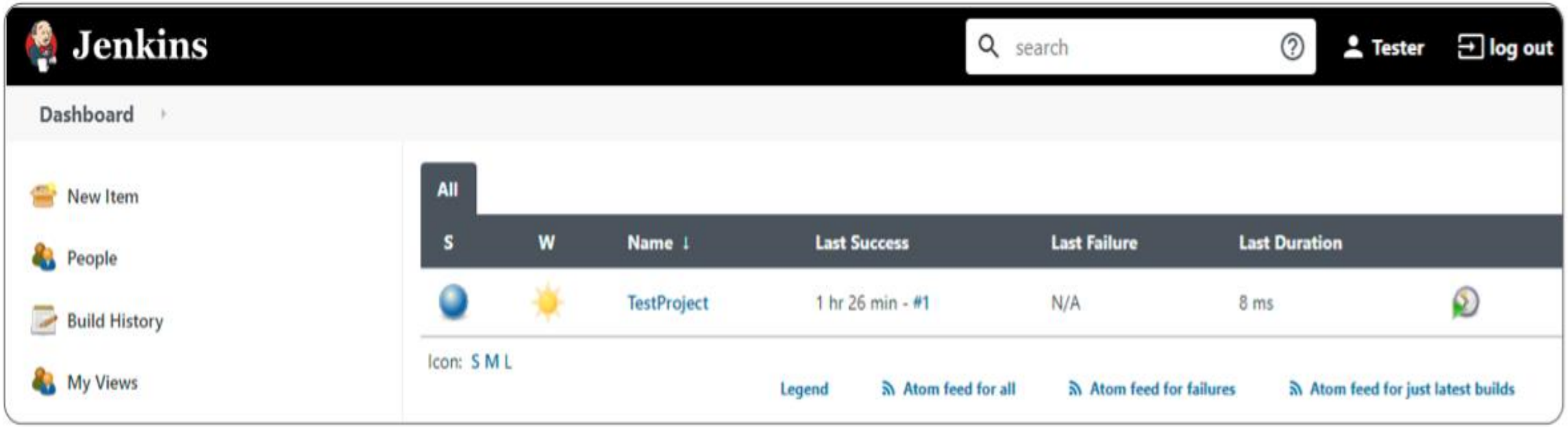


# Login as Developer and View the Dashboard



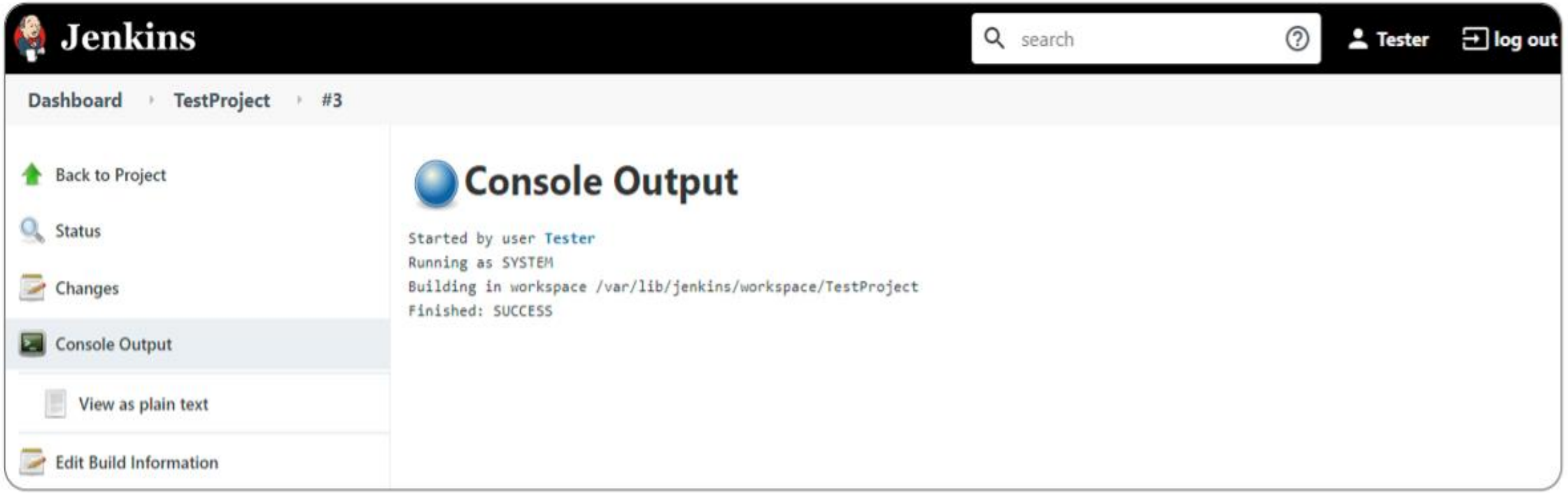

- Developer can view only the jobs started with Prog.*
- Developer can build the available Job.
- Developer can create a new development job, but the job name should starts with Prog.
- Developer cannot see: Tester projects, configure the system and manage plugin.

# Login as Tester and View the Dashboard




- Tester can view only the jobs started with Test.*
- Tester can build the available Job
- Tester can create a new test job, but the job name should starts with Test.
- Tester cannot see: Developer projects, configure the system and manage plugin

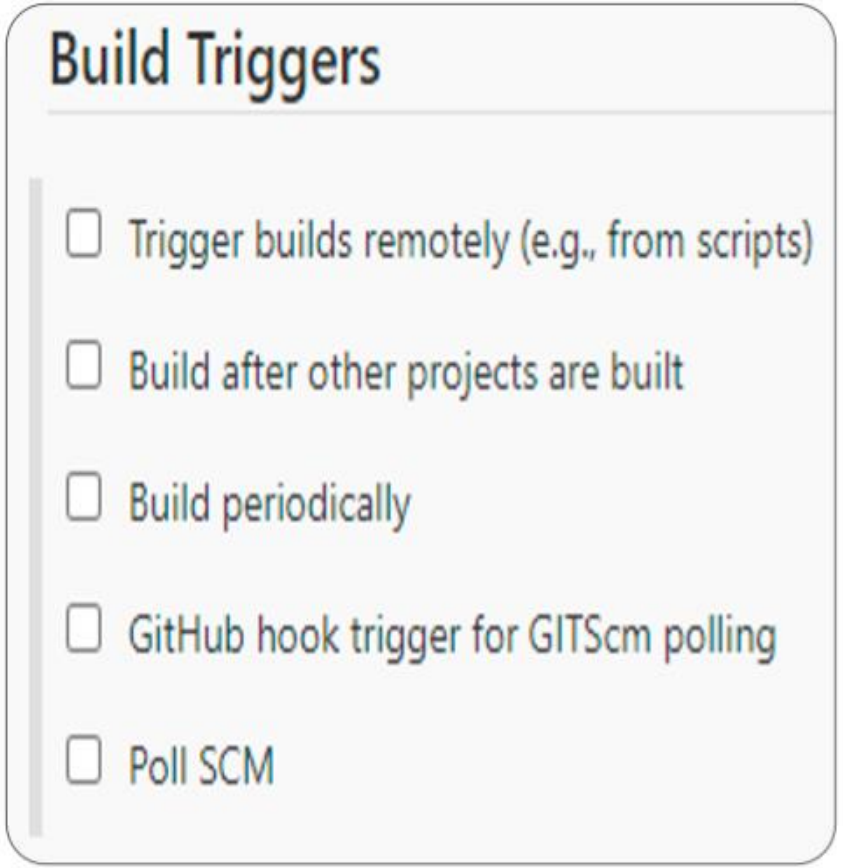
# Tester: Build and Verify



# Integrating Jenkins With Git

# Introduction

- So far, We have seen how to Trigger build remotely and build after other projects are built (chains of Jenkins job).
- Build periodically – You can trigger the jobs periodically with crontab time format.
- In this section, let's see how to GitHub hook trigger for GitScm polling and Poll SCM.
- First we need to integrate with GitHub repository.
- Next, Install Git Plugin.



# Install Git Plugin

# Poll SCM

Poll Source Code Management (SCM) vs Build Periodically

- **Build Periodically**: Jenkins builds periodically even if there are no changes in the project.

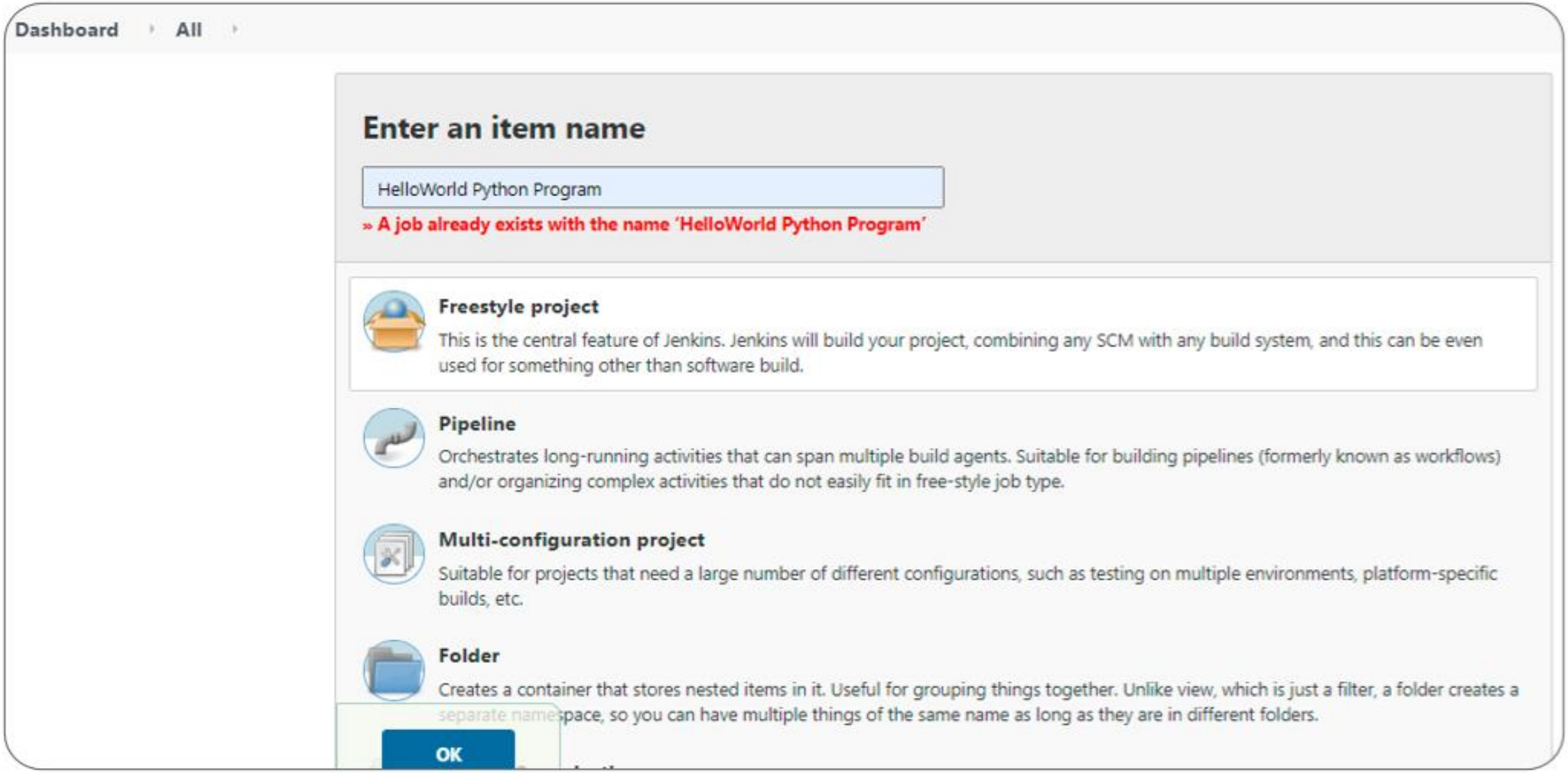
- **Poll SCM**: Jenkins builds periodically only if any new changes are made in the project.

**Poll SCM**

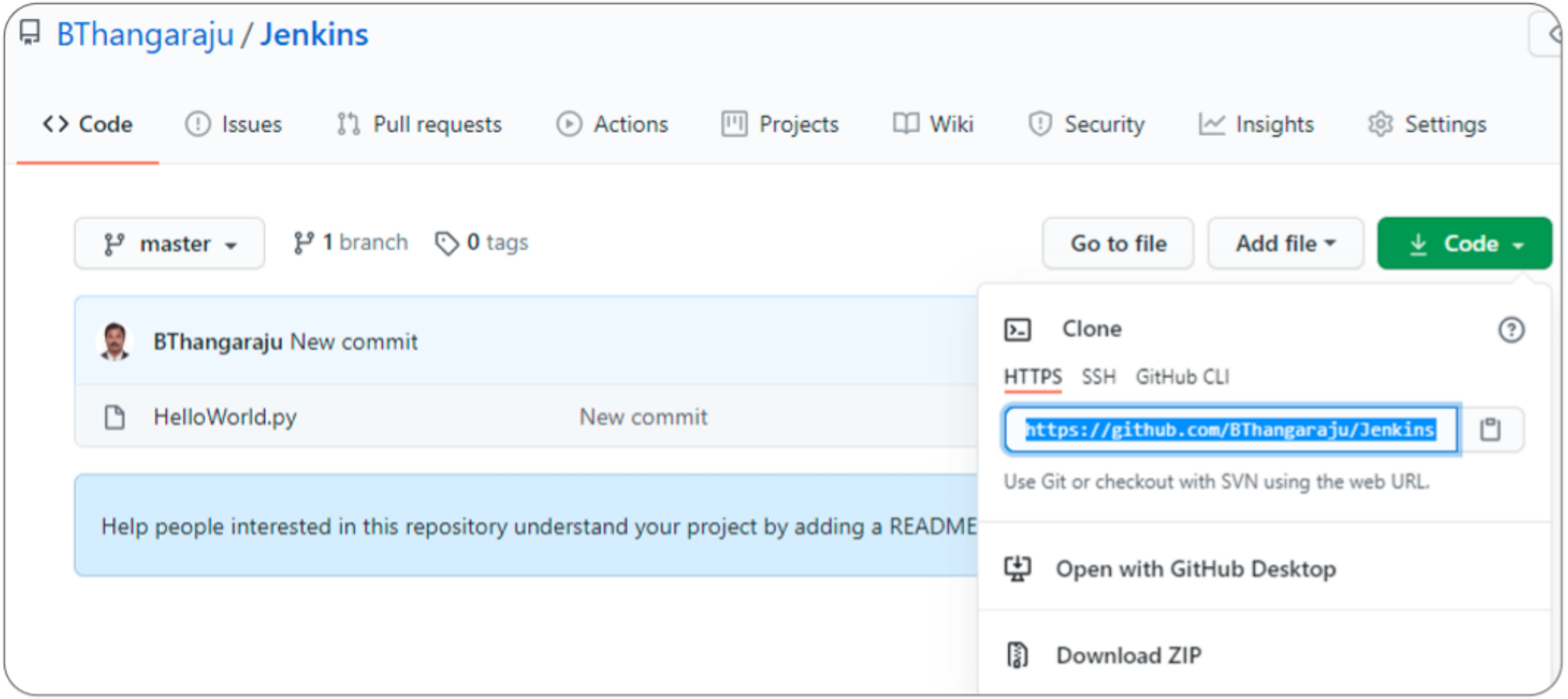
- * * * * * - for every minute, Jenkins polls periodically the GitHub to check whether any new commits were made.

- If there are any changes pushed since the last build, then Jenkins automatically builds the project.

# Create a New Freestyle Project

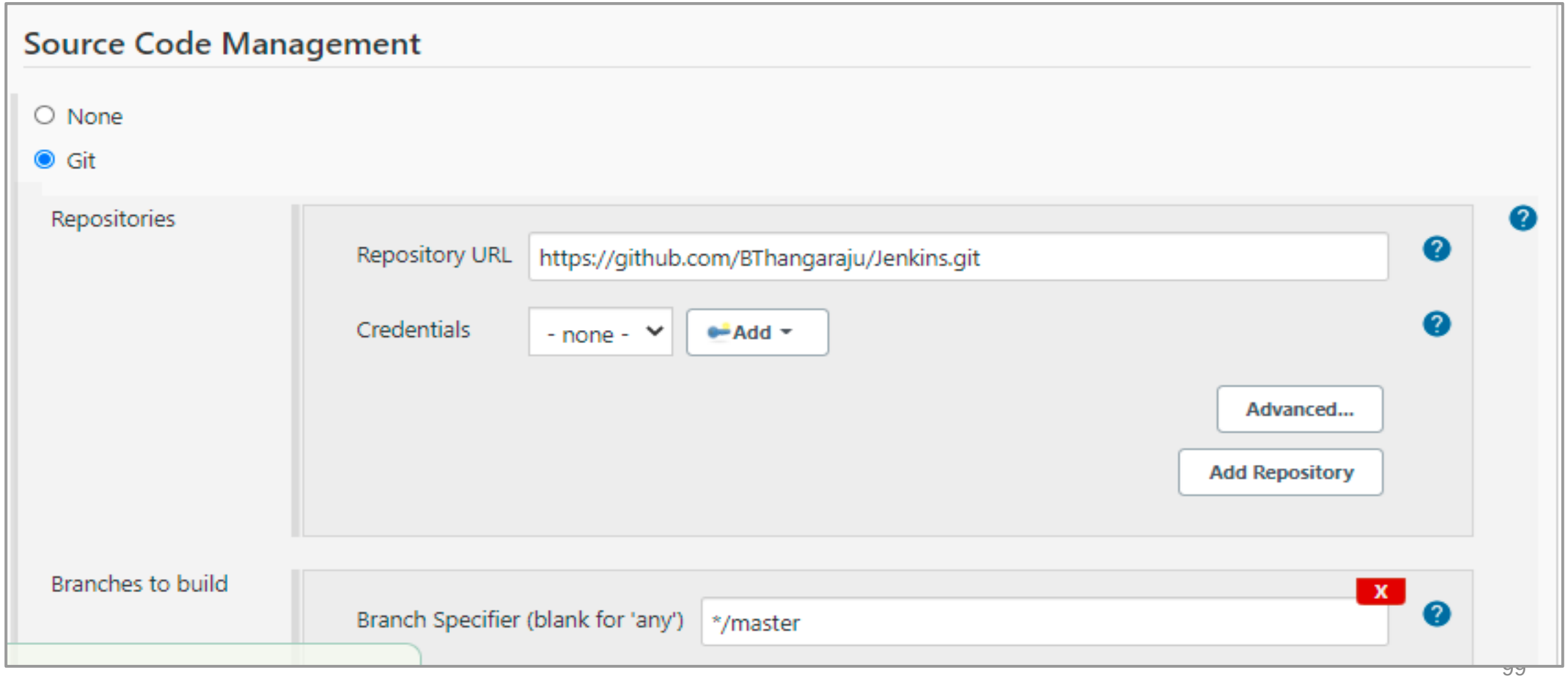


# Copy Your Project GitHub URL



# Enable Git

- Select Git in the Source Code Management and enter GitHub URL in the given Repository URL option.



# Poll SCM Fields and Format

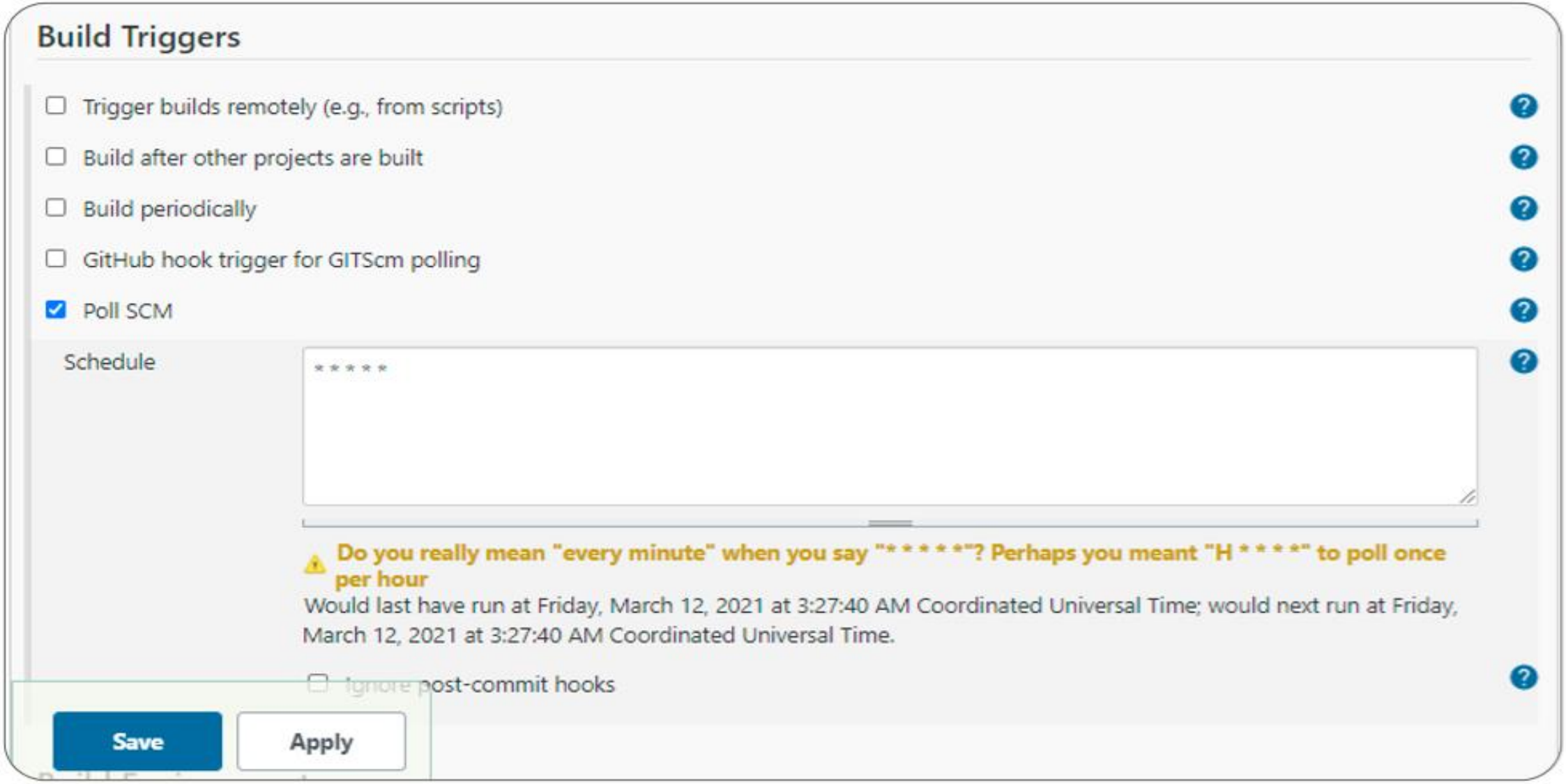
This field follows the syntax of cron (with minor differences). Specifically, each line consists of 5 fields separated by TAB or whitespace:
MINUTE HOUR DOM MONTH DOW

| field | allowed values |
| --- | --- |
| minute | 0-59 |
| hour | 0-23 |
| day of month | 1-31 |
| month | 1-12 (or names, see below) |
| day of week | 0-7 (0 or 7 is Sun, or use names) |

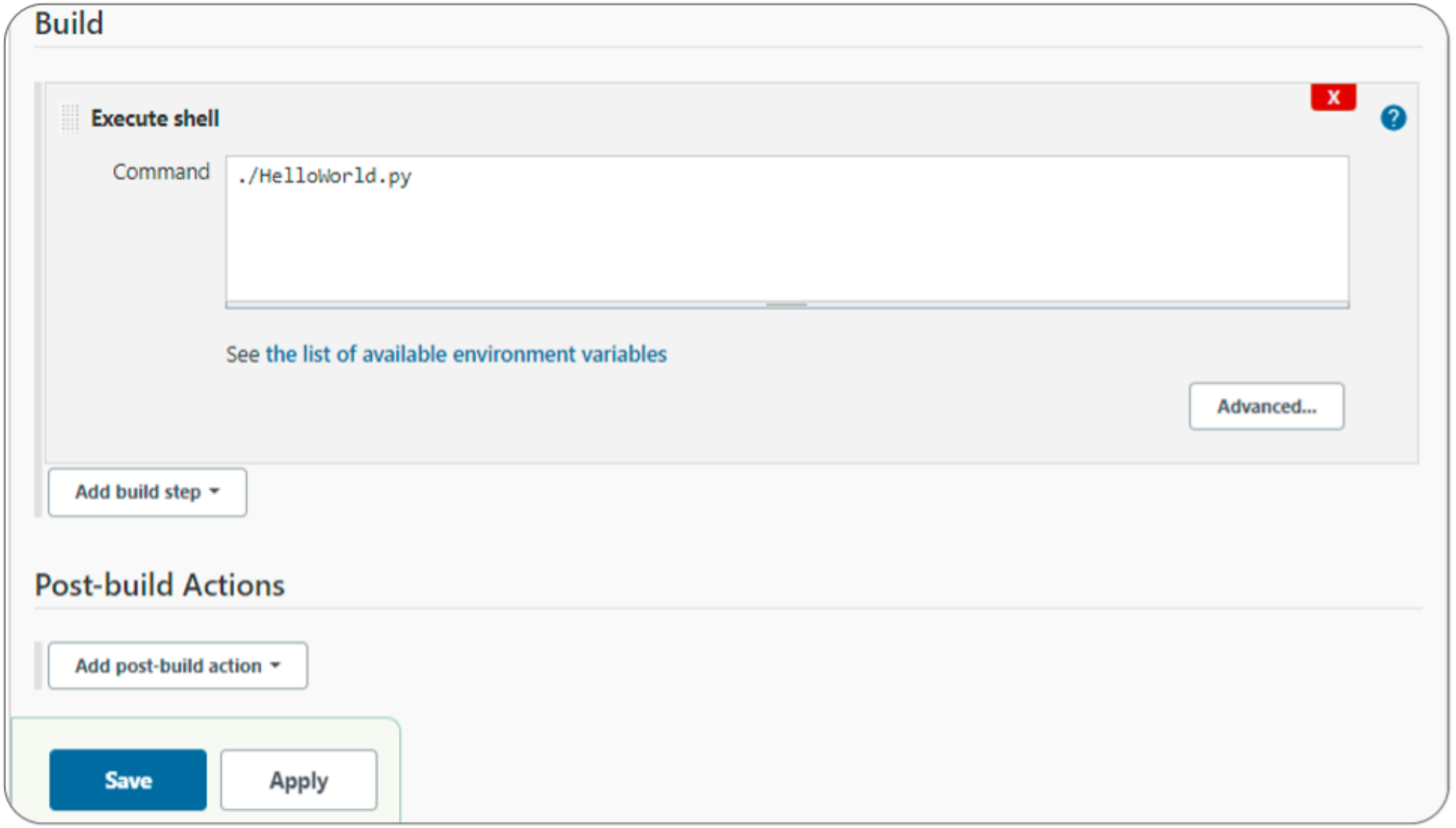
Examples:

```
# Every fifteen minutes (perhaps at :07, :22, :37, :52):
H/15 * * * *
# Every ten minutes in the first half of every hour (three times, perhaps at :04, :14, :24):
H(0-29)/10 * * * *
# Once every two hours at 45 minutes past the hour starting at 9:45 AM and finishing at 3:45 PM every weekday:
45 9-16/2 * * 1-5
# Once in every two hour slot between 8 AM and 4 PM every weekday (perhaps at 9:38 AM, 11:38 AM, 1:38 PM, 3:38
PM):
H H(8-15)/2 * * 1-5
# Once a day on the 1st and 15th of every month except December:
H H 1,15 1-11 *
```

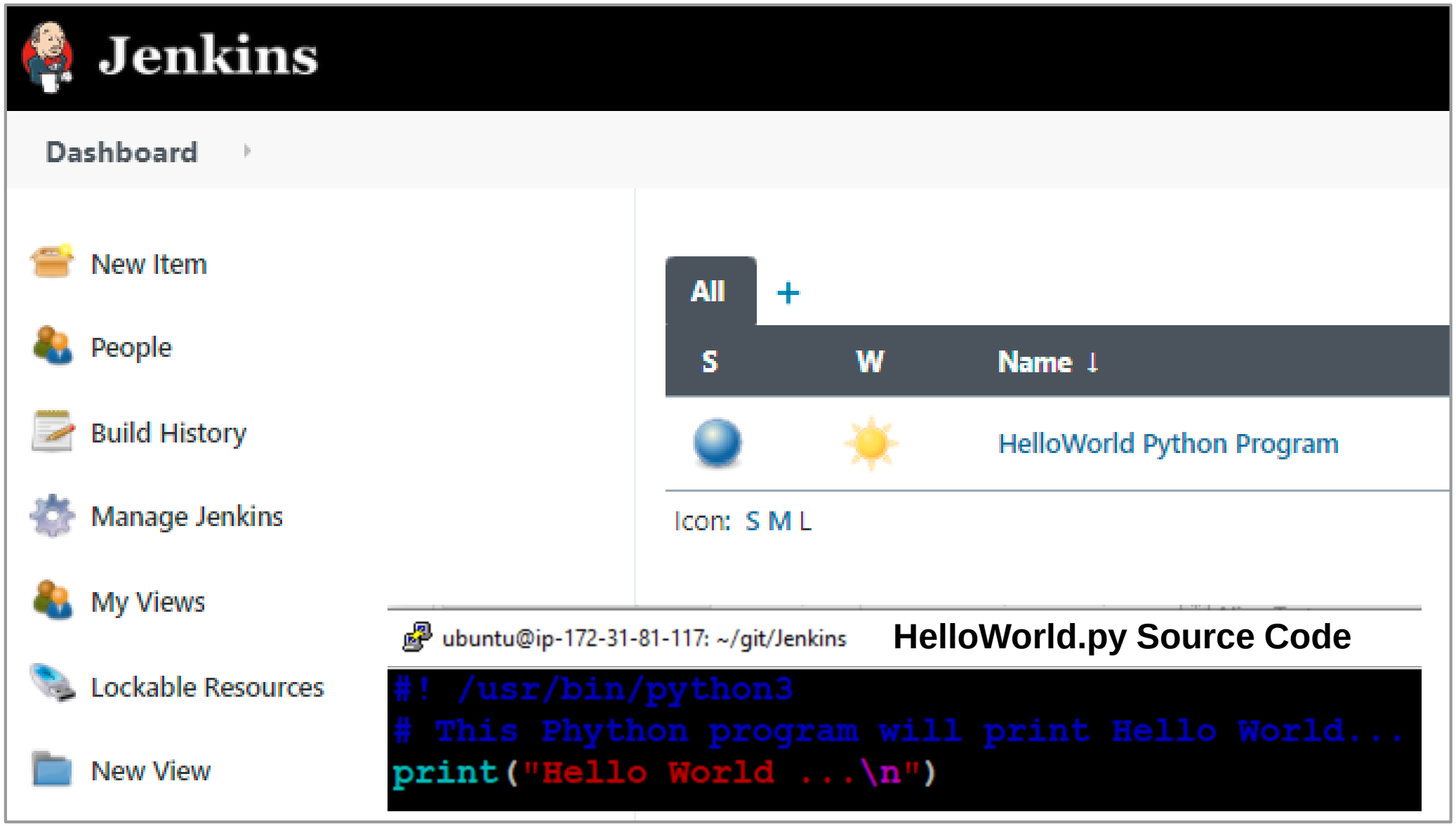
# Select Poll SCM and Enter the Value



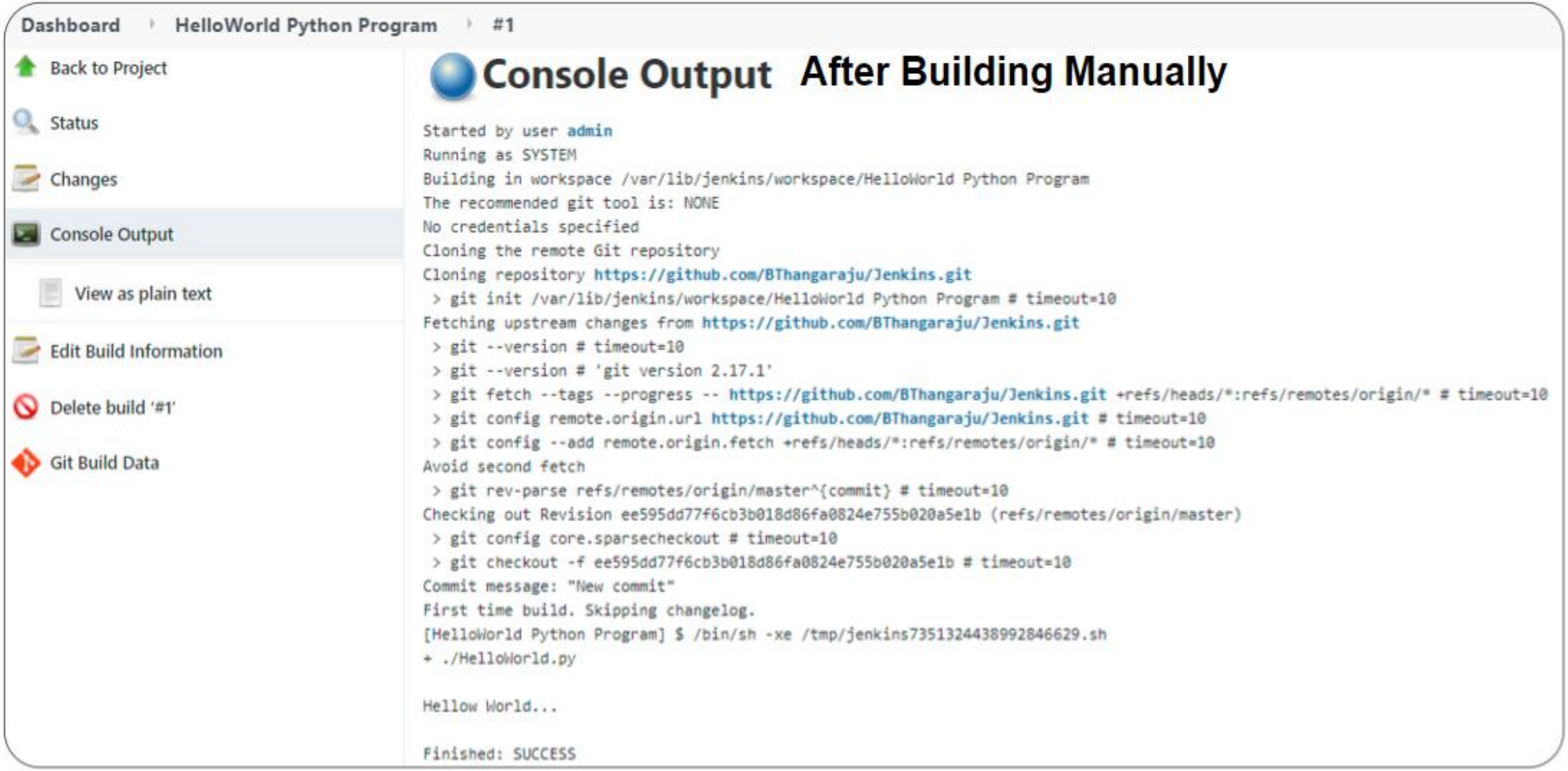
# Enter Command to Execute the Project



# Jenkins Dashboard



# Console Output



# Commit Changes and Build Automatically

ubuntu@ip-172-31-81-117: ~/git/Jenkins

```
#! /usr/bin/python3
# This Python program will print Hello World
print("\n Hello World...\n")
print("\n Hello World...\n")
```

```
ubuntu@ip-172-31-81-117:~/git/Jenkins$ git add .
ubuntu@ip-172-31-81-117:~/git/Jenkins$ git commit -m "First commit"
[master a8dc0e6] First commit
 1 file changed, 1 insertion(+)
ubuntu@ip-172-31-81-117:~/git/Jenkins$ git push origin master
Username for 'https://github.com': b.thangaraju@iiitb.ac.in
Password for 'https://b.thangaraju@iiitb.ac.in@github.com':
Counting objects: 3, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 302 bytes | 302.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/BThangaraju/Jenkins.git
   ee595dd..a8dc0e6  master -> master
ubuntu@ip-172-31-81-117:~/git/Jenkins$ vim HelloWorld.py
```

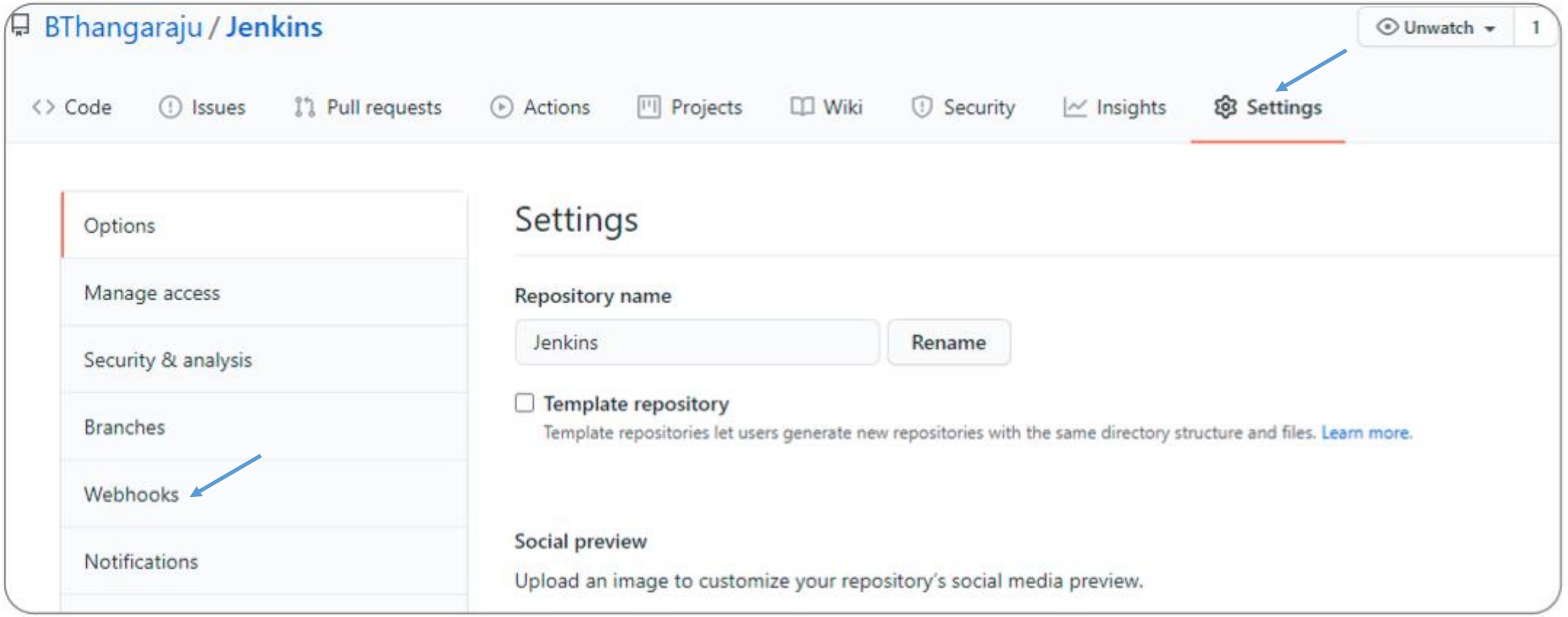


- After Committing changes in the local Git repo, we need to push the changes into our GitHub repository.
- Then Jenkins check the GitHub repository periodically and automatically build the Job.

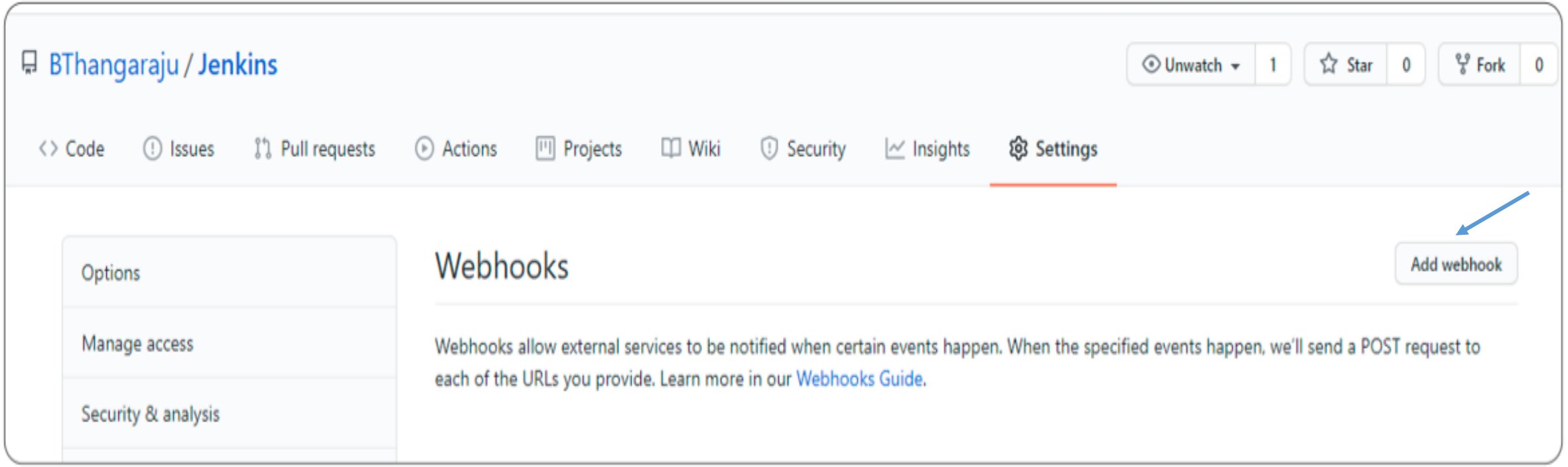
# GitHub Hook Trigger for GITScm Polling

- For GitHub hook trigger for GITScm polling, Jenkins will not build the job periodically; instead, when you commit new changes in the GitHub repository, only then will Jenkins trigger the job to build.

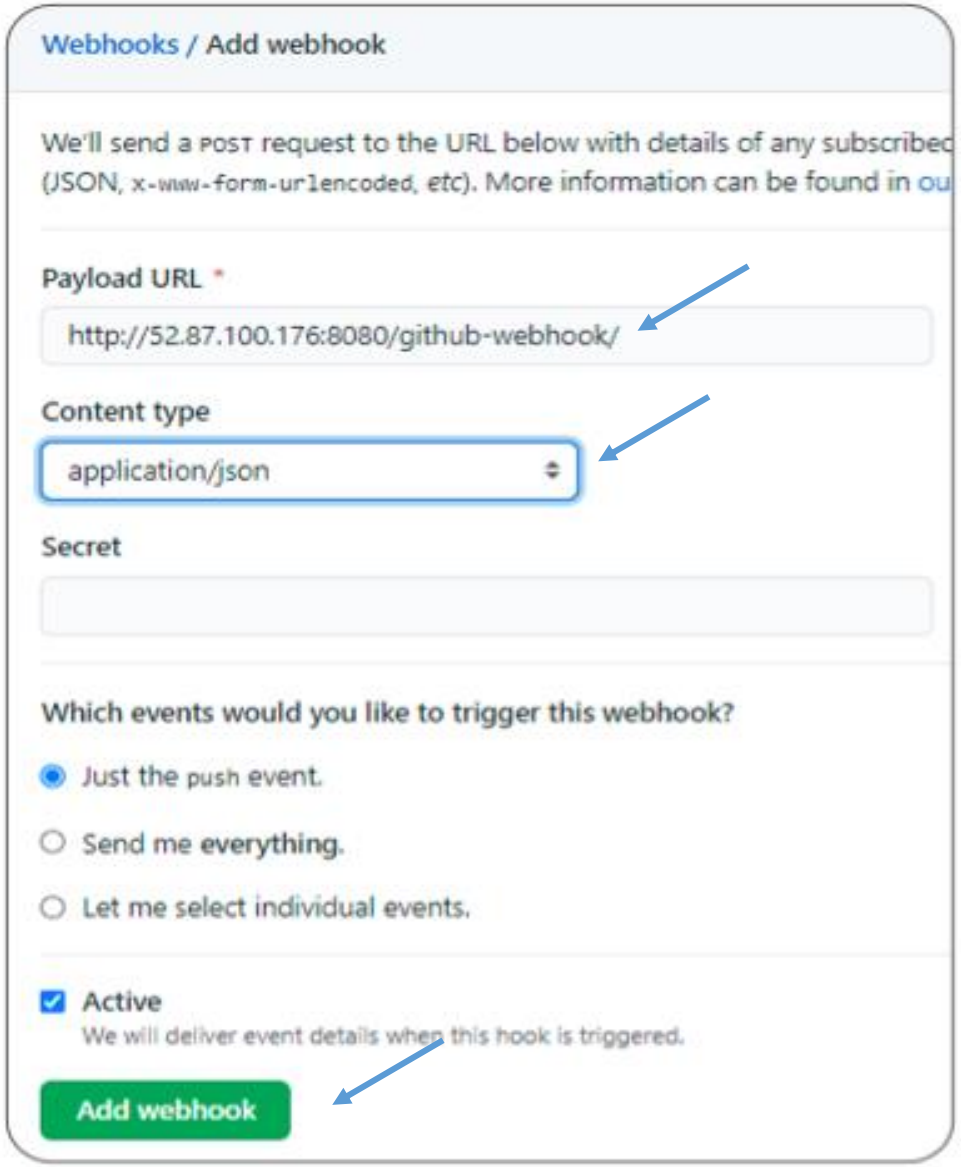
# Working With Webhooks

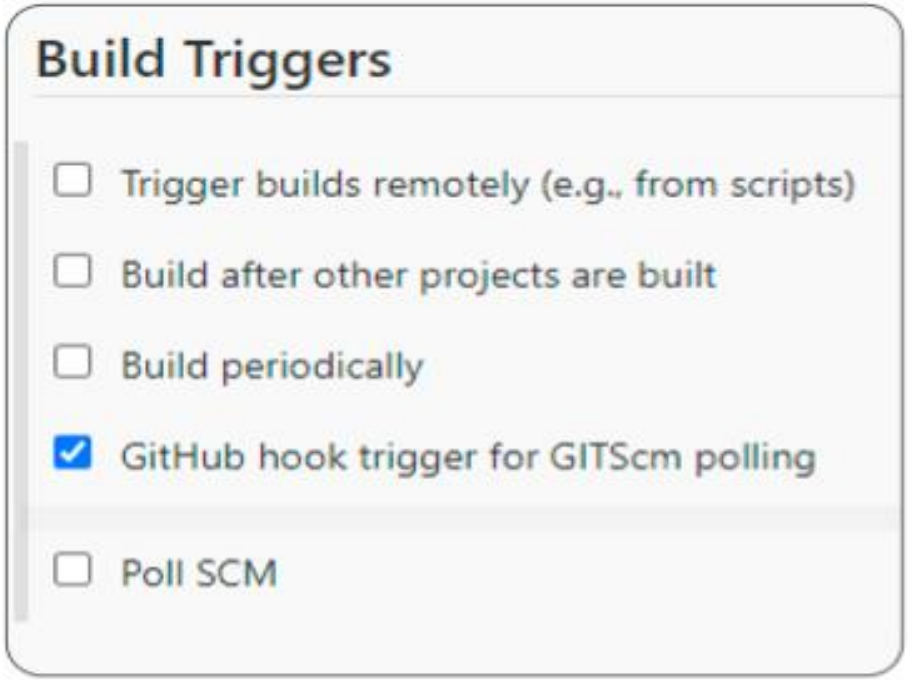


# Working With Webhooks



# Working With Webhooks



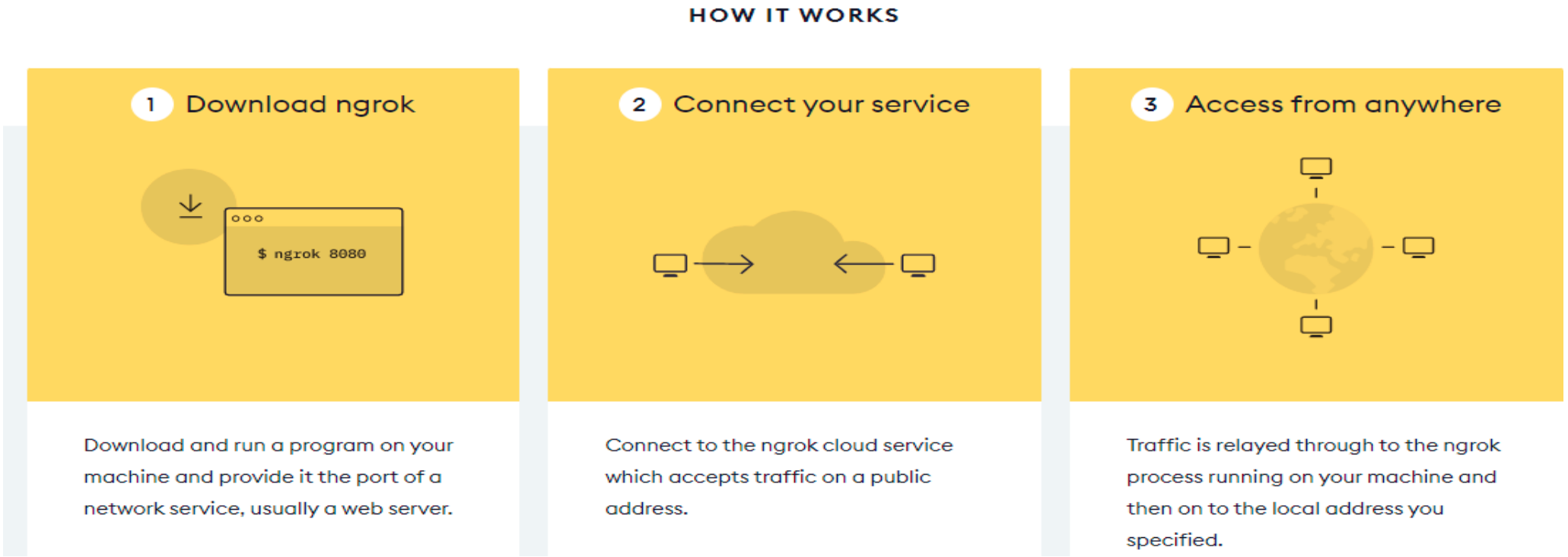




# ngrok - secure introspectable tunnels to localhost

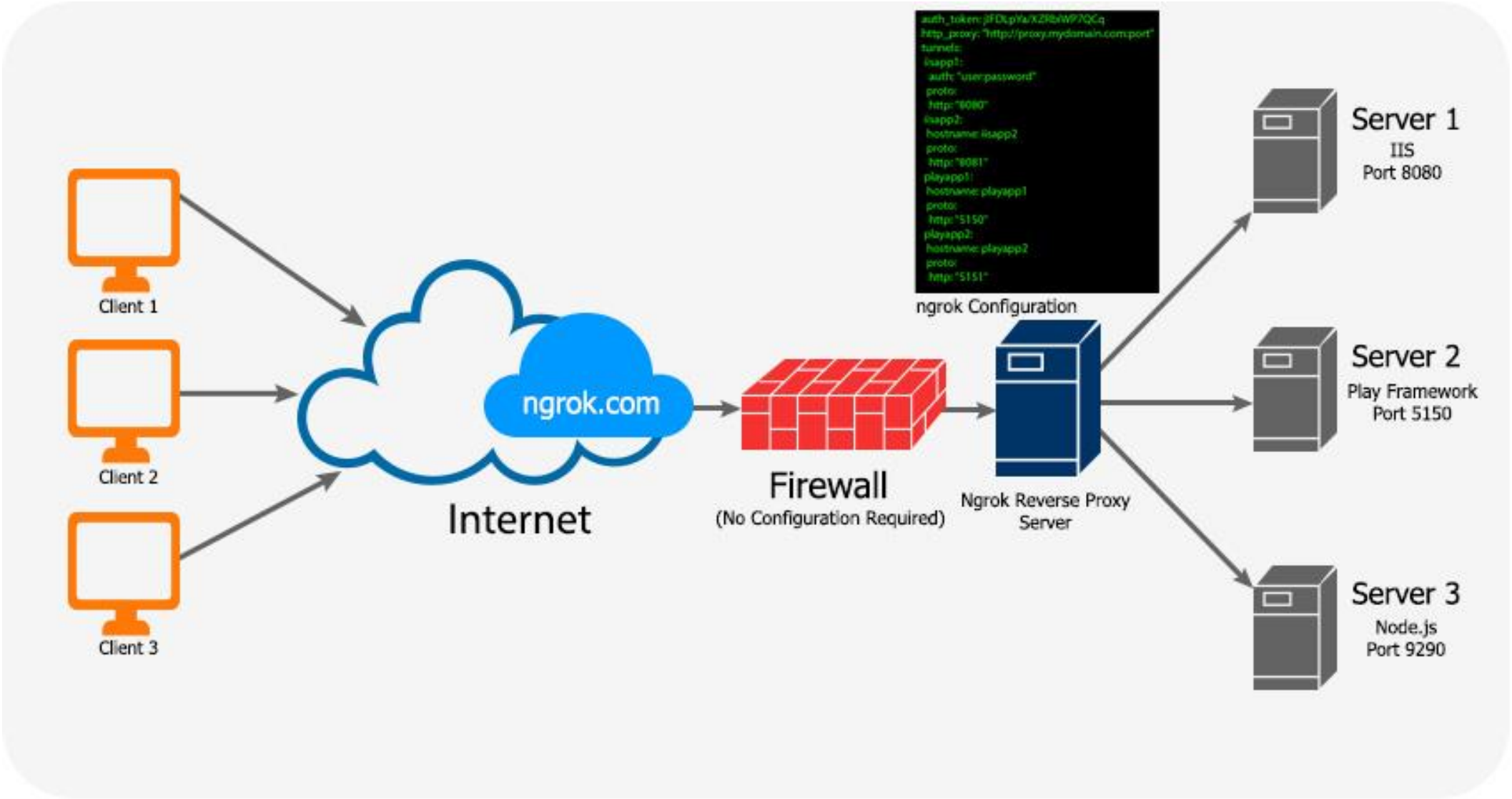
1. Ngrok exposes local servers behind NATs (Network Address Translation) and firewalls to the public internet over secure tunnels.
2. ngrok provides a real-time web UI where you can introspect all HTTP traffic running over your tunnels.
3. ngrok allows you to expose a web server running on your local machine to the internet. Just tell ngrok what port your web server is listening on.
4. Example: Expose a web server on port 80 of your local machine to the internet. $*ngrok http 8080*
5. If different port number for ex: 5001 then: *$ngrok http https://localhost:5001*

# ngrok - secure introspectable tunnels to localhost

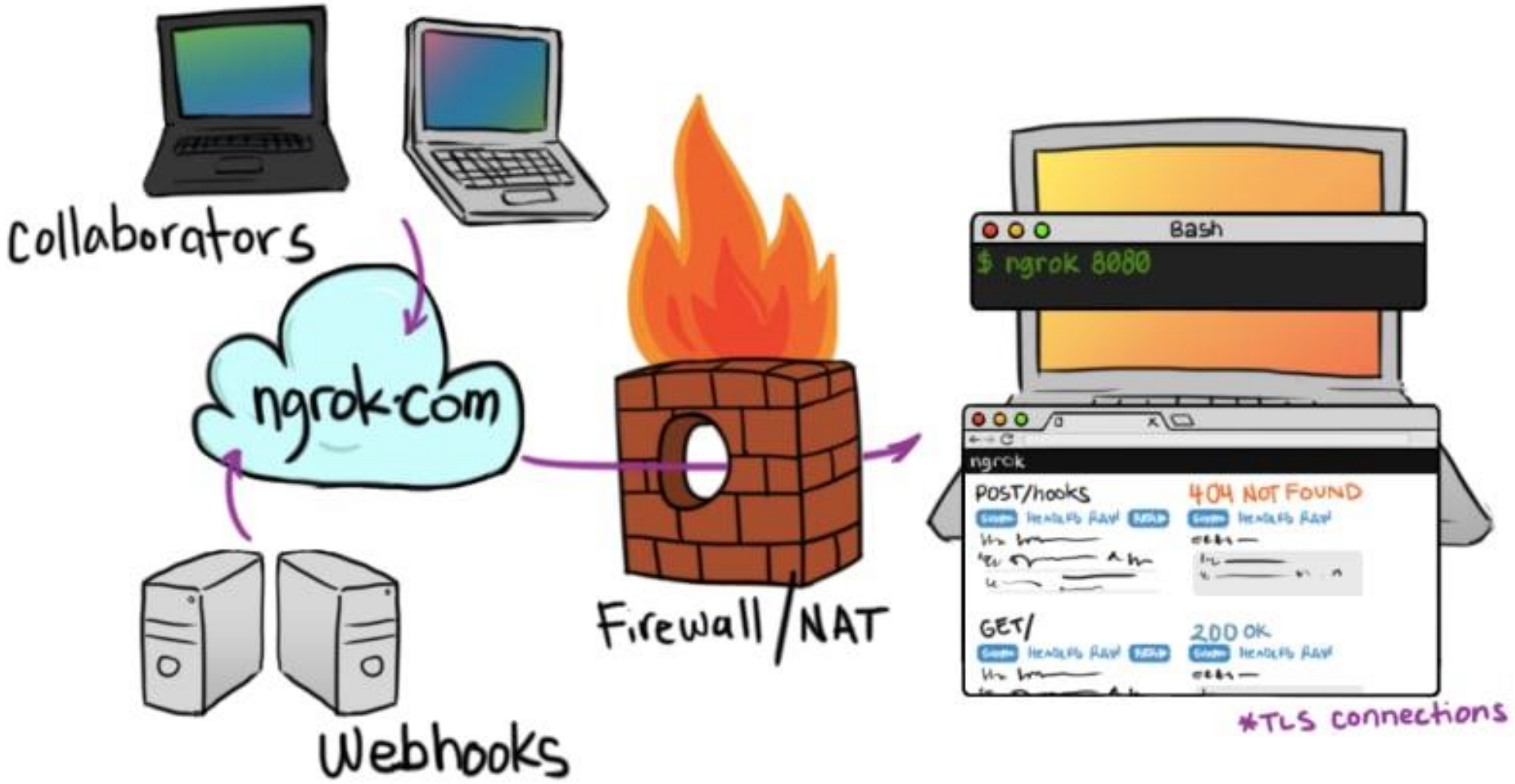
1. Ngrok exposes local servers behind NATs (Network Address Translation) and firewalls to the public internet over secure tunnels.
2. **ngrok** provides a real-time web UI where you can introspect all HTTP traffic running over your tunnels.



# Ngrok Architecture



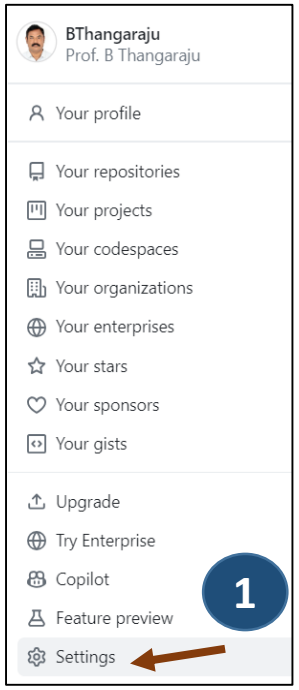
# Ngrok Connection

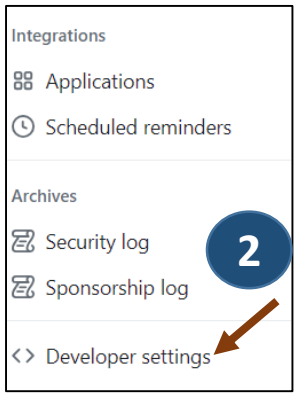


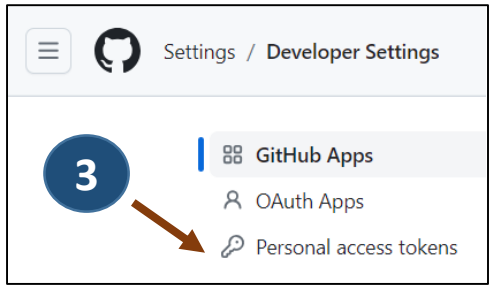
# Ngrok Installation

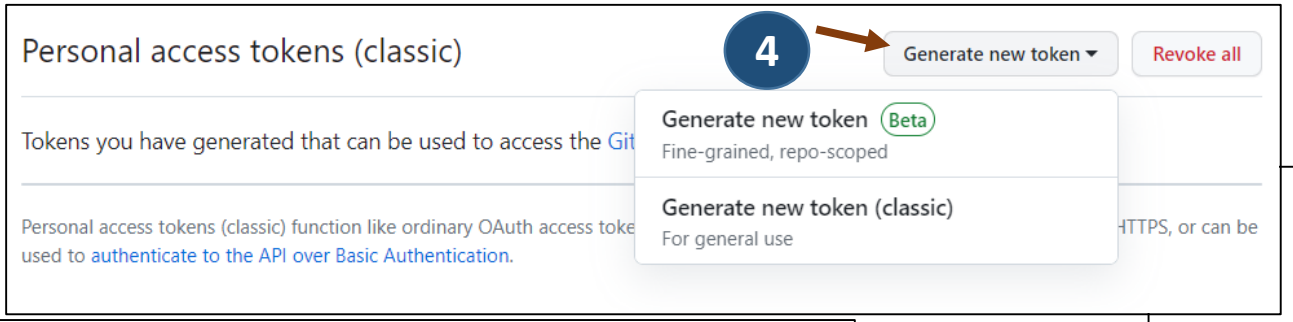
1. Sign up in https://ngrok.com/
2. Download ngrok from: https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.tgz
3. Then extract ngrok from the terminal: $sudo tar xvzf ~/Downloads/ngrok-stable-linux-amd64.tgz -C /usr/local/bin
4. Copy Authtoken from: https://dashboard.ngrok.com/get-started/your-authtoken
5. Add  Authtoken: $ngrok authtoken <token>
6. Execute $ngrok http 8080; copy the public ip address for your local host.

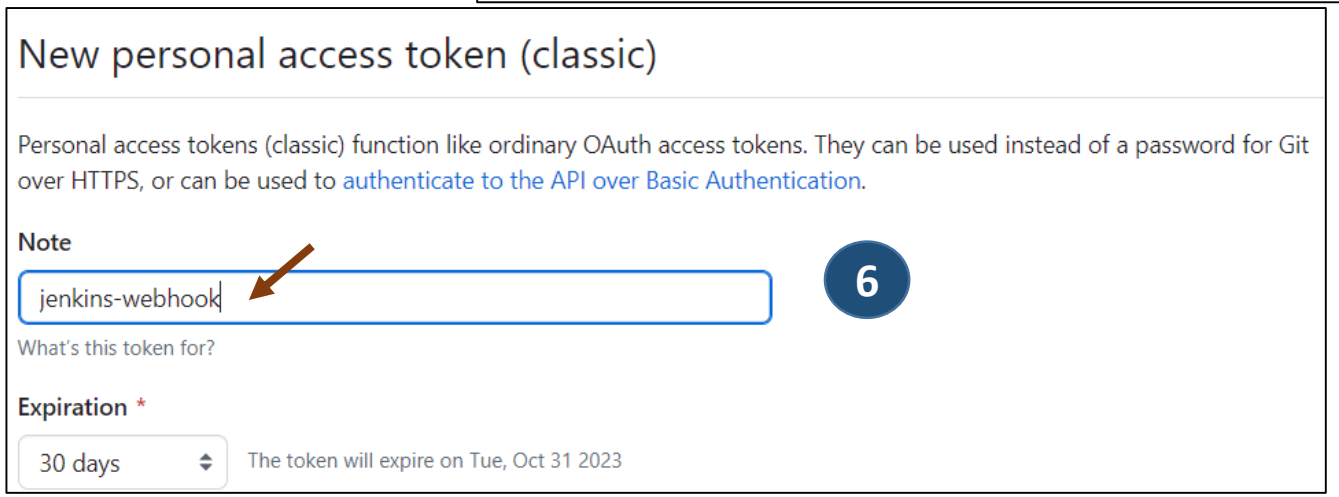
```
ngrok by @inconshreveable                                       (Ctrl+C to quit)

Session Status                online
Account                       Thangaraju (Plan: Free)
Version                       2.3.40
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://5336-122-167-77-156.ngrok.io -> http://localhost:8080
Forwarding                    https://5336-122-167-77-156.ngrok.io -> http://localhost:8080

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```


BThangaraju
Prof. B Thangaraju
Your profile
Your repositories
Your projects
Your codespaces
Your organizations
Your enterprises
Your stars
Your sponsors
Your gists
Upgrade
Try Enterprise
Copilot
Feature preview
Settings
1

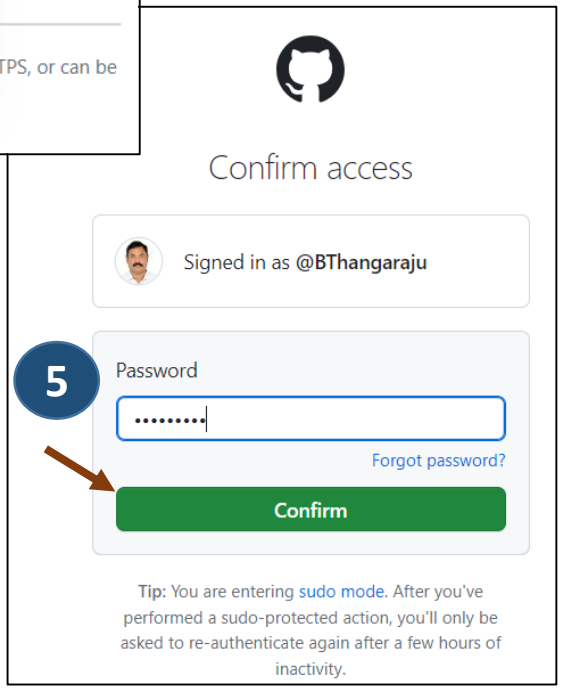

Integrations
Applications
Scheduled reminders
Archives
Security log
Sponsorship log
2
Developer settings

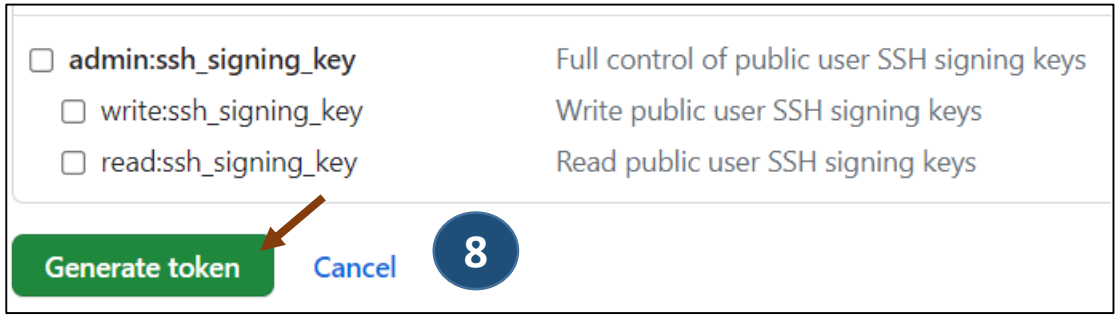

Settings / Developer Settings
3
GitHub Apps
OAuth Apps
Personal access tokens


Personal access tokens (classic)
4
Generate new token
Revoke all
Tokens you have generated that can be used to access the Git
Generate new token Beta
Fine-grained, repo-scoped
Generate new token (classic)
For general use
Personal access tokens (classic) function like ordinary OAuth access toke
used to authenticate to the API over Basic Authentication.
HTTPS, or can be


New personal access token (classic)
Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.
Note
jenkins-webhook
6
What's this token for?
Expiration *
30 days
The token will expire on Tue, Oct 31 2023


admin:repo_hook
Full control of repository hooks
write:repo_hook
Write repository hooks
read:repo_hook
Read repository hooks
7


Confirm access
Signed in as @BThangaraju
5
Password
Forgot password?
Confirm
Tip: You are entering sudo mode. After you've performed a sudo-protected action, you'll only be asked to re-authenticate again after a few hours of inactivity.


admin:ssh_signing_key
Full control of public user SSH signing keys
write:ssh_signing_key
Write public user SSH signing keys
read:ssh_signing_key
Read public user SSH signing keys
Generate token
Cancel
8

## Personal access tokens (classic)

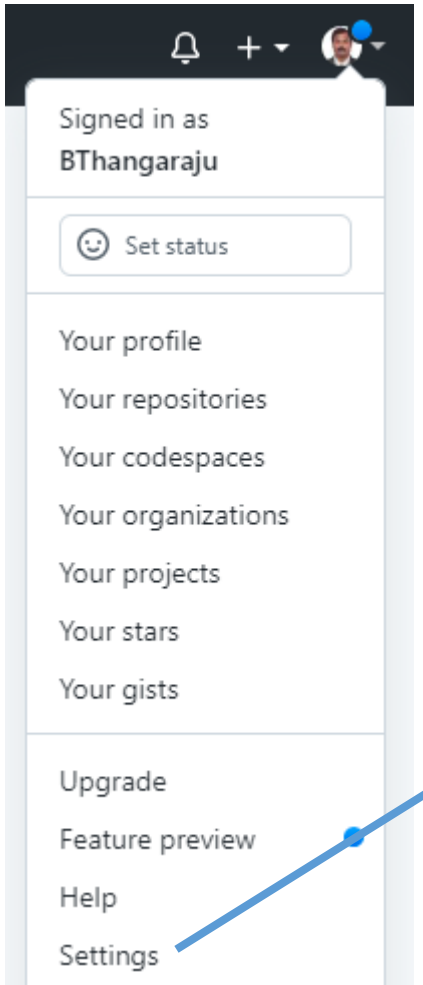
Generate new token ▾ | Revoke all

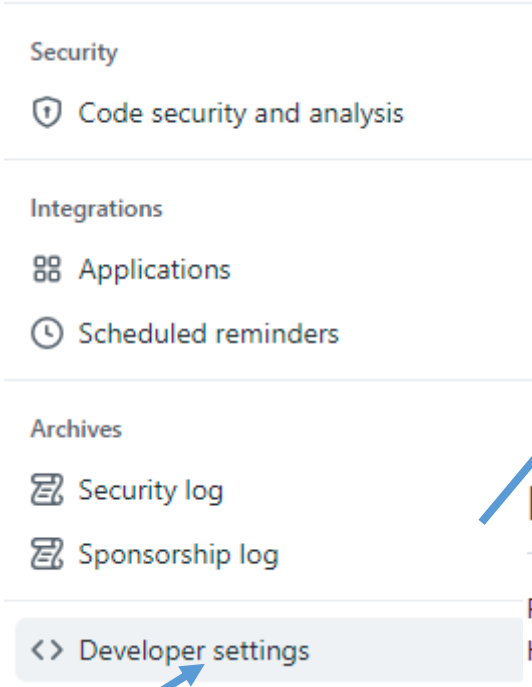
Tokens you have generated that can be used to access the GitHub API.

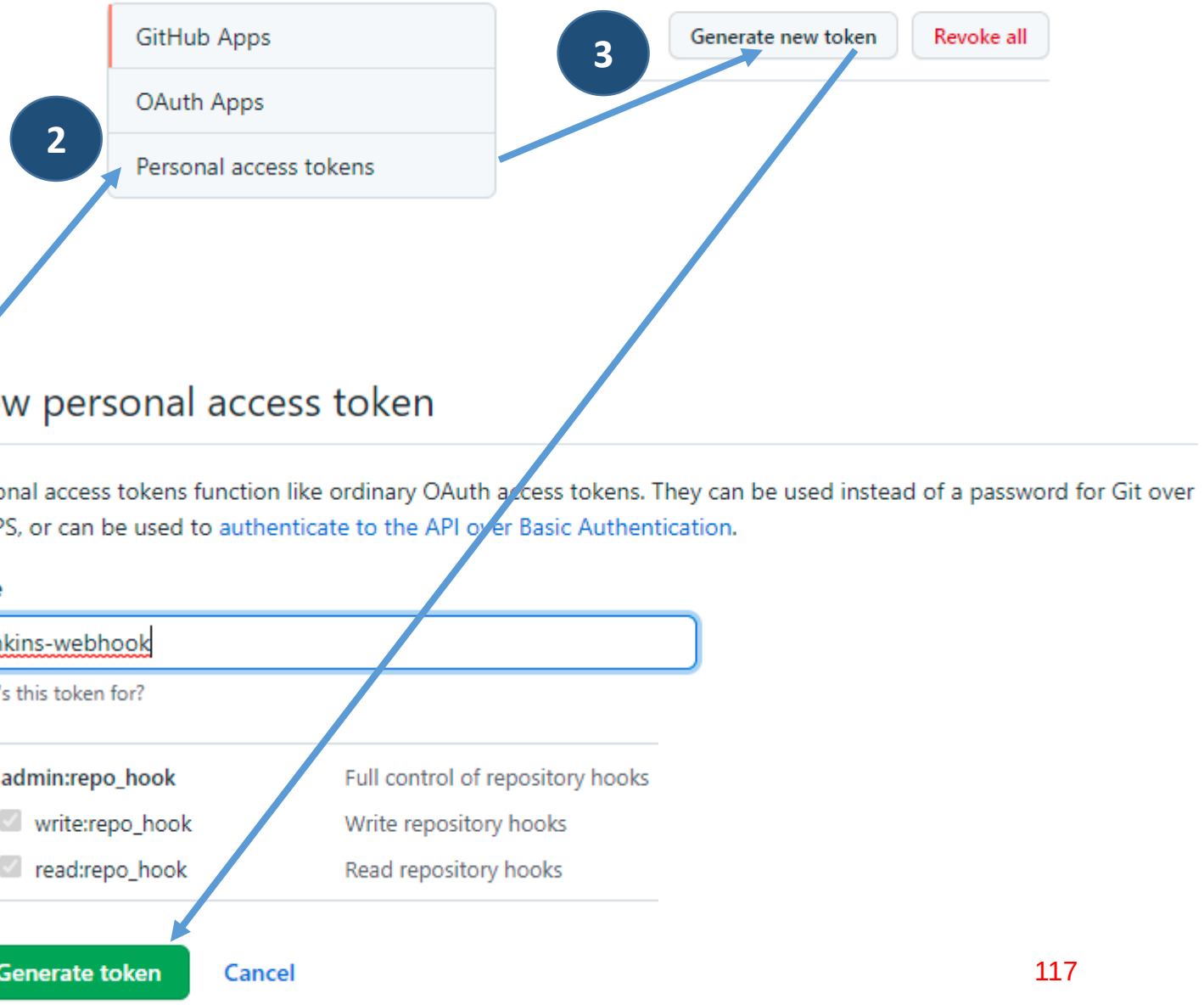
Make sure to copy your personal access token now. You won't be able to see it again!

✓ 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  Delete

# Secret Text creation -GitHub



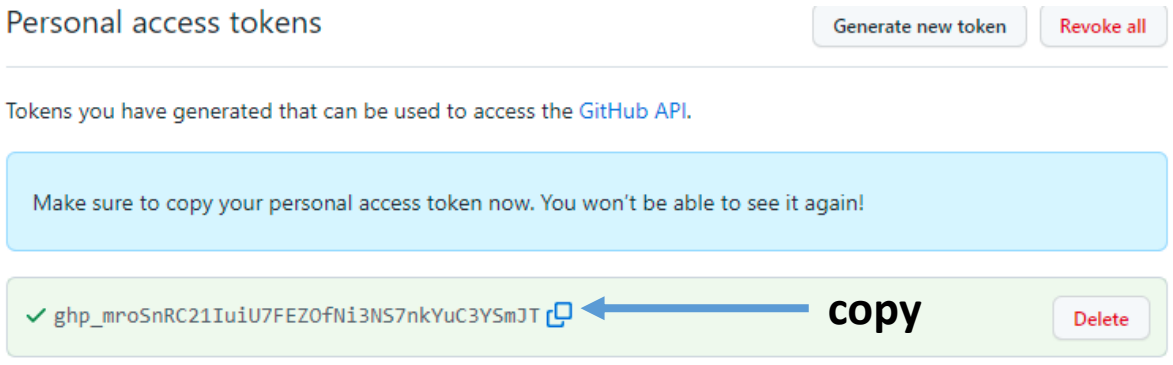

1

2

3

4

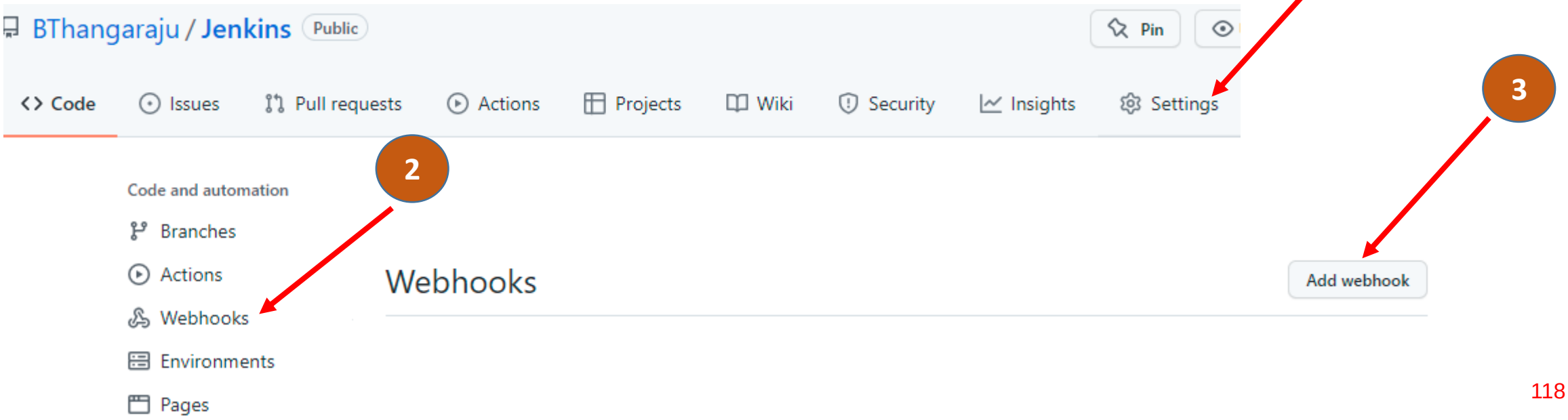
# Webhook Configuration -GitHub



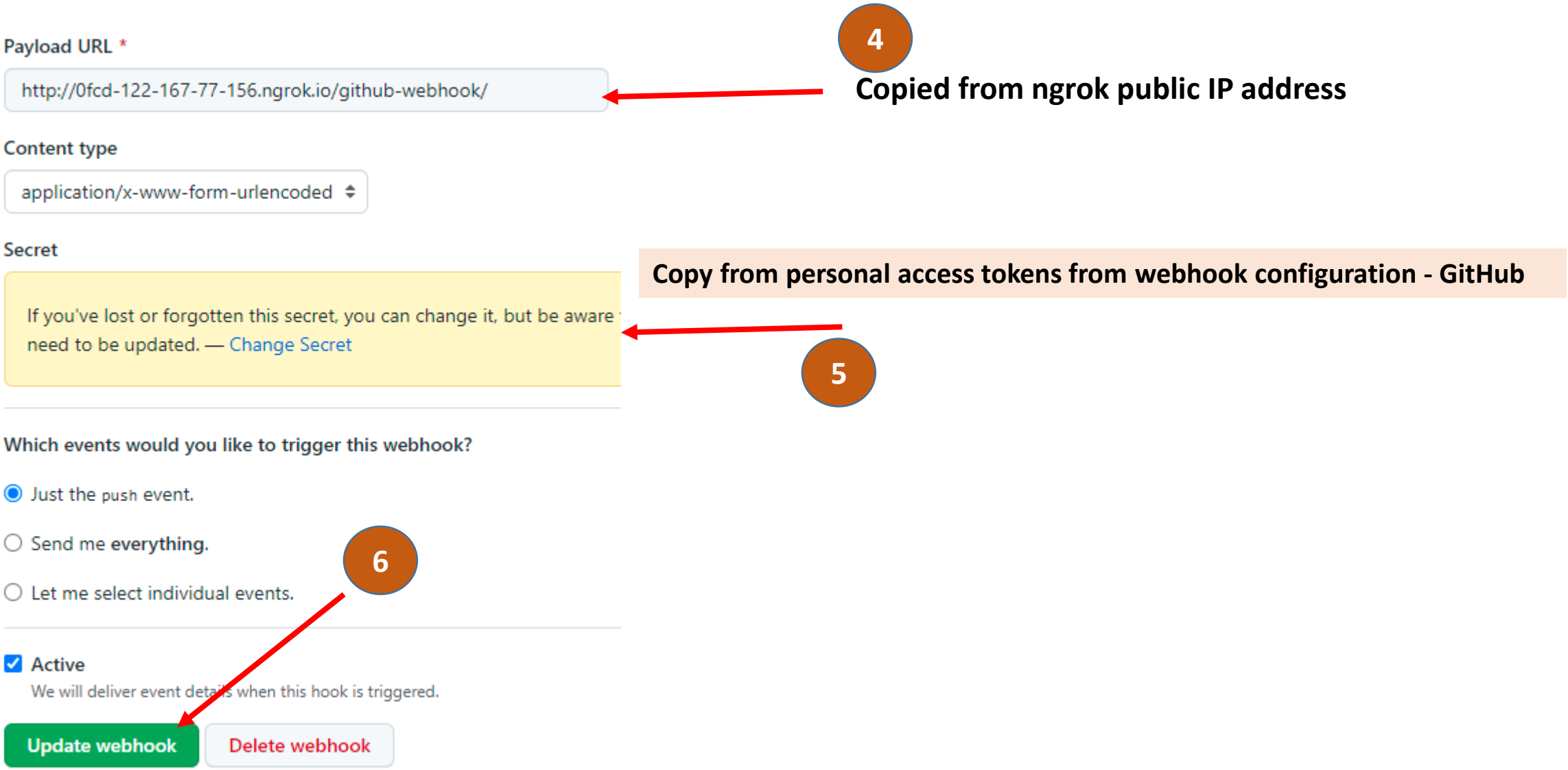

5

**Secret Text**

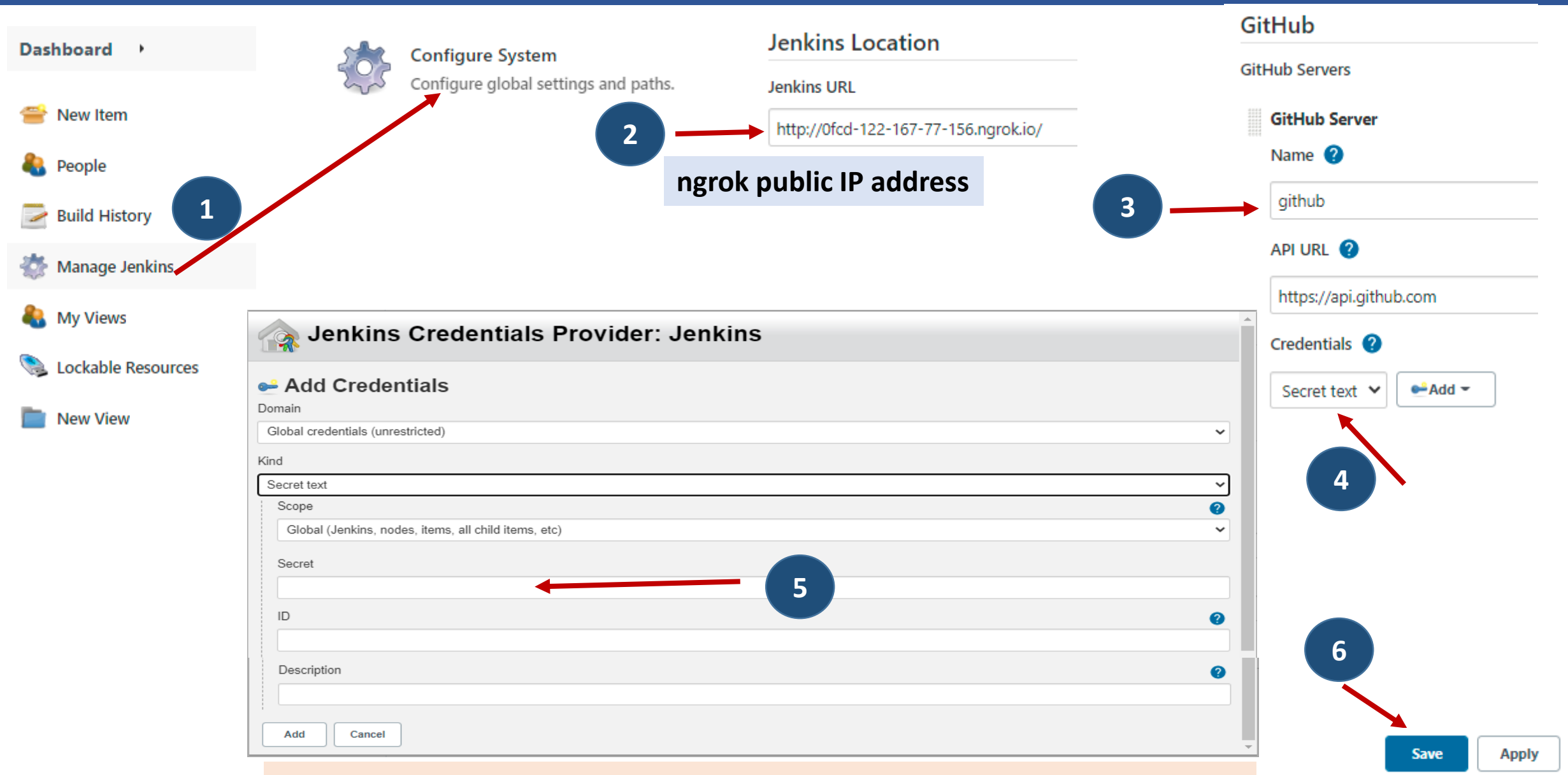
**Create Webhook from Github repository**



# Webhook Configuration -GitHub

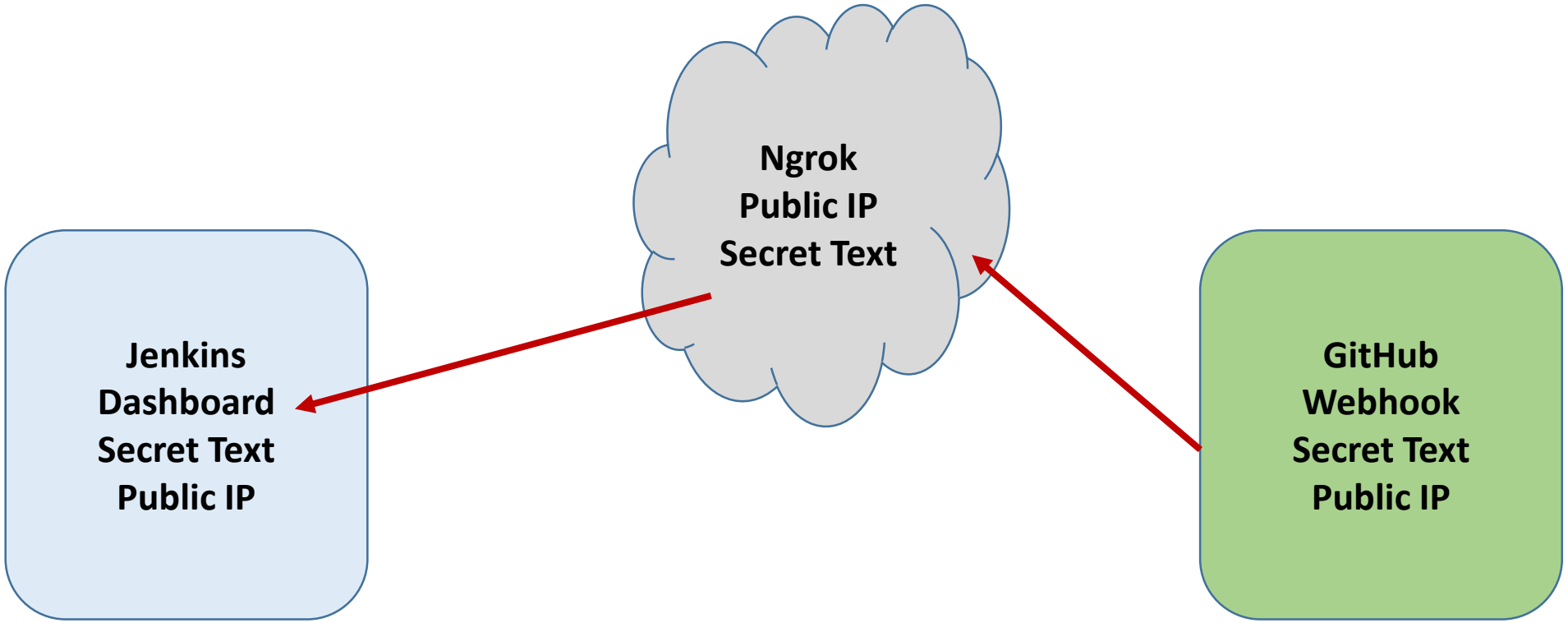


# GitHub Server Configuration –Jenkins Dashboard




**Copy from personal access tokens from webhook configuration - GitHub**

# GitSCM Polling



# Introduction to Jenkins Pipeline

# Jenkins Pipeline Introduction

**Pipeline**:

- It can be specified as code, so you can write pipeline script and maintain version control in the Git repository.
- It will provide continuous release of the application.
- Sequence of stages to perform the given tasks such as pulling code from the Git repository, static code analysis, building project, executing unit test, automated tests, performance tests and deploying application.

# Types of Pipeline

- **Declarative**
  - New method
  - Easy to use for beginners
  - Groovy language skill is desirable

- **Scripted**
  - Traditional
  - Based on Groovy Domain Specific Language (DSL)
  - Multiple features - very expressive and flexible tool
  - Difficult to use for beginners
  - Should have working experience on Groovy language

# Declarative vs Scripted Pipeline Script Template

```
pipeline {
        agent any
        stages {
stage('build code') {
        steps {
            /*write steps */
            }
        }
        stage ('test') {
        steps {
            /*write steps */
            }
        }
    }
}
```

**Declarative Pipeline Script**

```
node {
        stage ('build code' {
            /*write steps */
        }
        stage ('test') {
            /*write steps */
        }
}
```

**Scripted Pipeline Script**

**Pipeline** – contains all the script content
**Agent and Node** – defines the agent where the pipeline will run
**Stages** – contains all the stages
**Steps** – way to execute various jobs

# Pipeline Directives

Directive allows you to define a list of parameters to be used in the script. Parameters should be provided once the pipeline is triggered.

- **Environment** – defined as environment variables
- **Input** – prompt for input
- **Options** – configure pipeline-specific options like retry, timeout, etc.
- **Parallel** – list of nested stages to be run in parallel
- **Parameters** – list of parameters to provide when triggering the Pipeline (e.g., string, password)
- **Post** – run at the end of a Pipeline's execution (e.g., add some notification or other end of Pipeline tasks)

# Pipeline Directives

- **Tools** – defining tools or packages to auto-install and put on the PATH (e.g., maven. Jdk, gradle)
- **Triggers** – determines how pipelines should be triggered (e.g., cron, poll SCM)
- **When** – determine executing stage depending on the given condition

Jenkins Pipeline Syntax: https://www.jenkins.io/doc/book/pipeline/syntax/

# Jenkins Pipeline

**Pipeline:**

- Defined as a suite of plugins that helps you orchestrate simple or complex automation

**Jenkins Pipeline:**

- Provides tool for modeling delivery pipelines, “as Code” (Pipeline as Code)
- Implement and integrate CD pipeline.

**CI/CD Pipeline:**

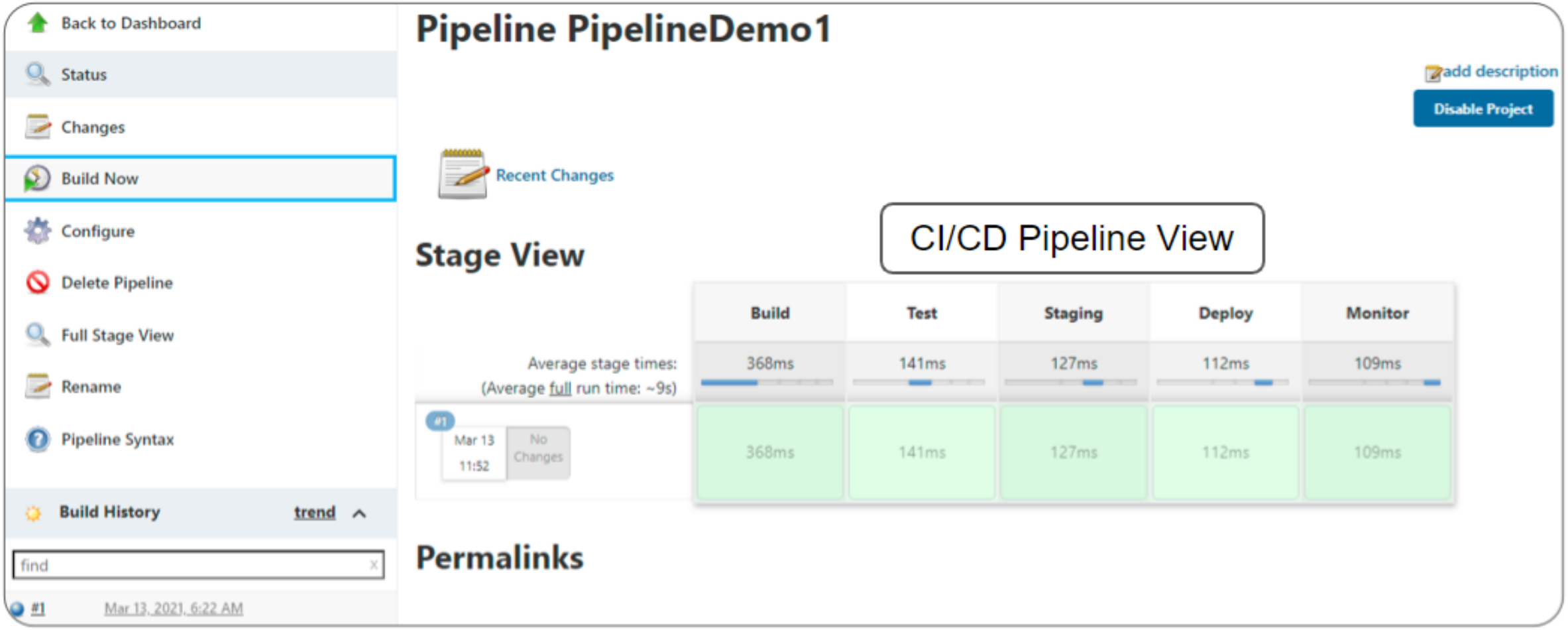
- Integrates SDLC stages, steps to execute tasks in each stage, trigger jobs for a given order and show pipeline status with logs
- Automation from Continuous build to Continuous Monitoring (build, test, staging, deploy and monitor)

# Demo – Jenkins Pipeline CI/CD

## Declarative Pipeline Script

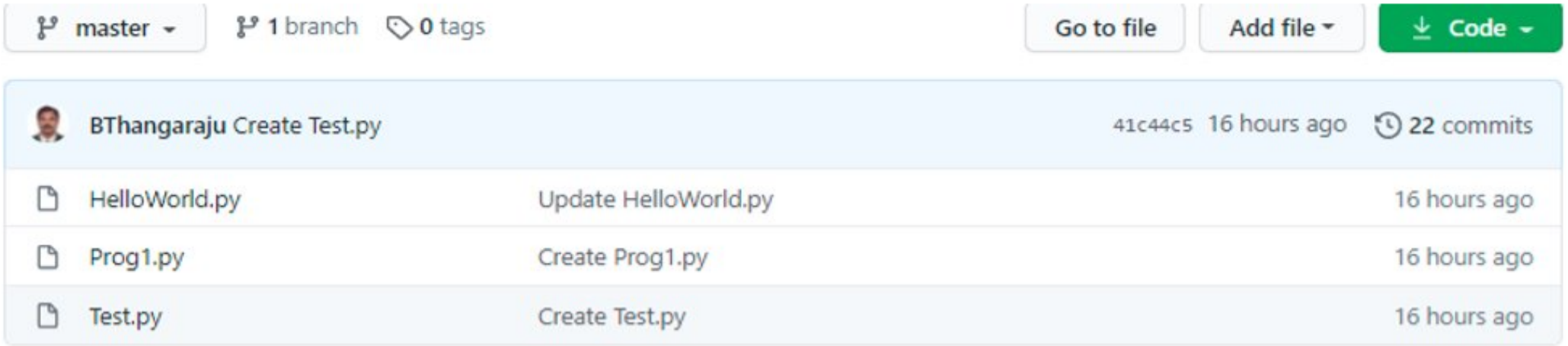
```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'This is job building stage'
            }
        }
 stage('Test') {
            steps {
                echo 'This is Testing stage'
            }
        }
stage('Staging') {
            steps {
                echo 'This is Staging environment'
            }
        }
stage('Deploy') {
            steps {
                echo 'This is Deploying stage'
            }
        }
 stage('Monitor') {
            steps {
                echo 'This is Monitoring stage'
            }
        }
    }
}
```

# Demo – Jenkins Pipeline CI/CD



# Demo Programs

- Git Repository




- HelloWorld.py –program to git clone example.
- Prog1.py – for building the code.
- Test.py –for testing the Prog1.py source code.

# Demo Programs

ubuntu@ip-172-31-57-56: ~

```
#! /usr/bin/python3
# This Phython program will print Hello World...
print("Hello World ...\n")
```

HelloWorld.py

ubuntu@ip-172-31-81-117: ~/git-demo

```
#!/usr/bin/python3
# Source code for summation of two numbers

def summation(data):
    return sum(data)
```

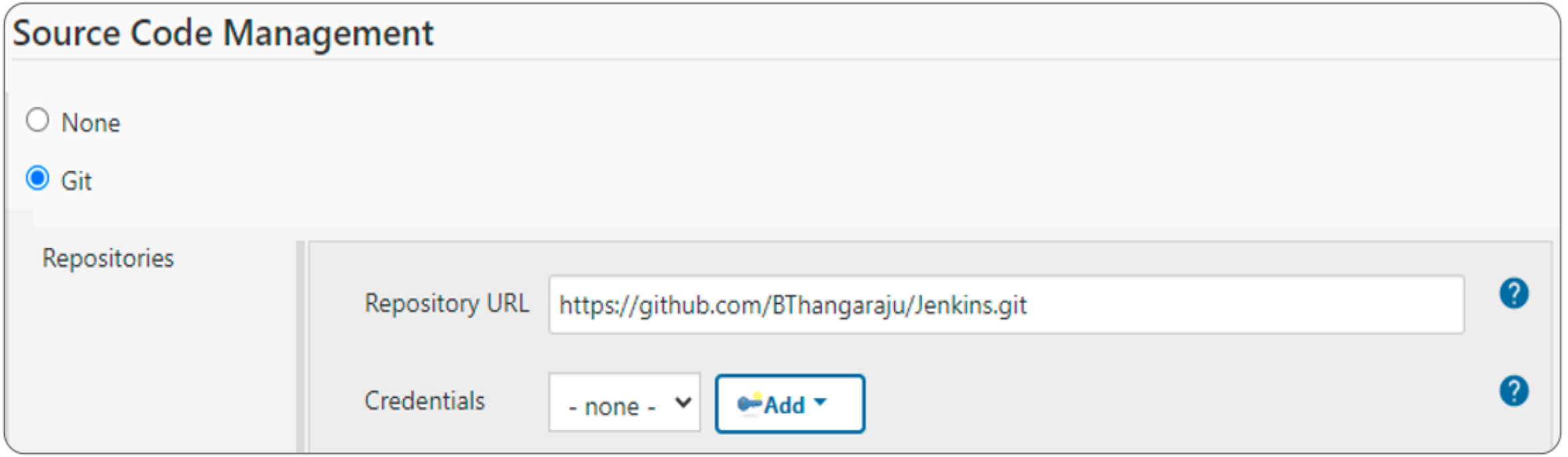
Prog1.py

ubuntu@ip-172-31-81-117: ~

```
!/usr/bin/python3
# Test case for adding two numbers
import unittest

from Prog1 import summation

class TestSum(unittest.TestCase):
    def test_list_int(self):
        """
        Test case to add two numbers
        """
        data = [23, 32]
        result = summation(data)
        self.assertEqual(result, 55)

if __name__ == '__main__':
    unittest.main()
```

Test.py

# Integration With Git

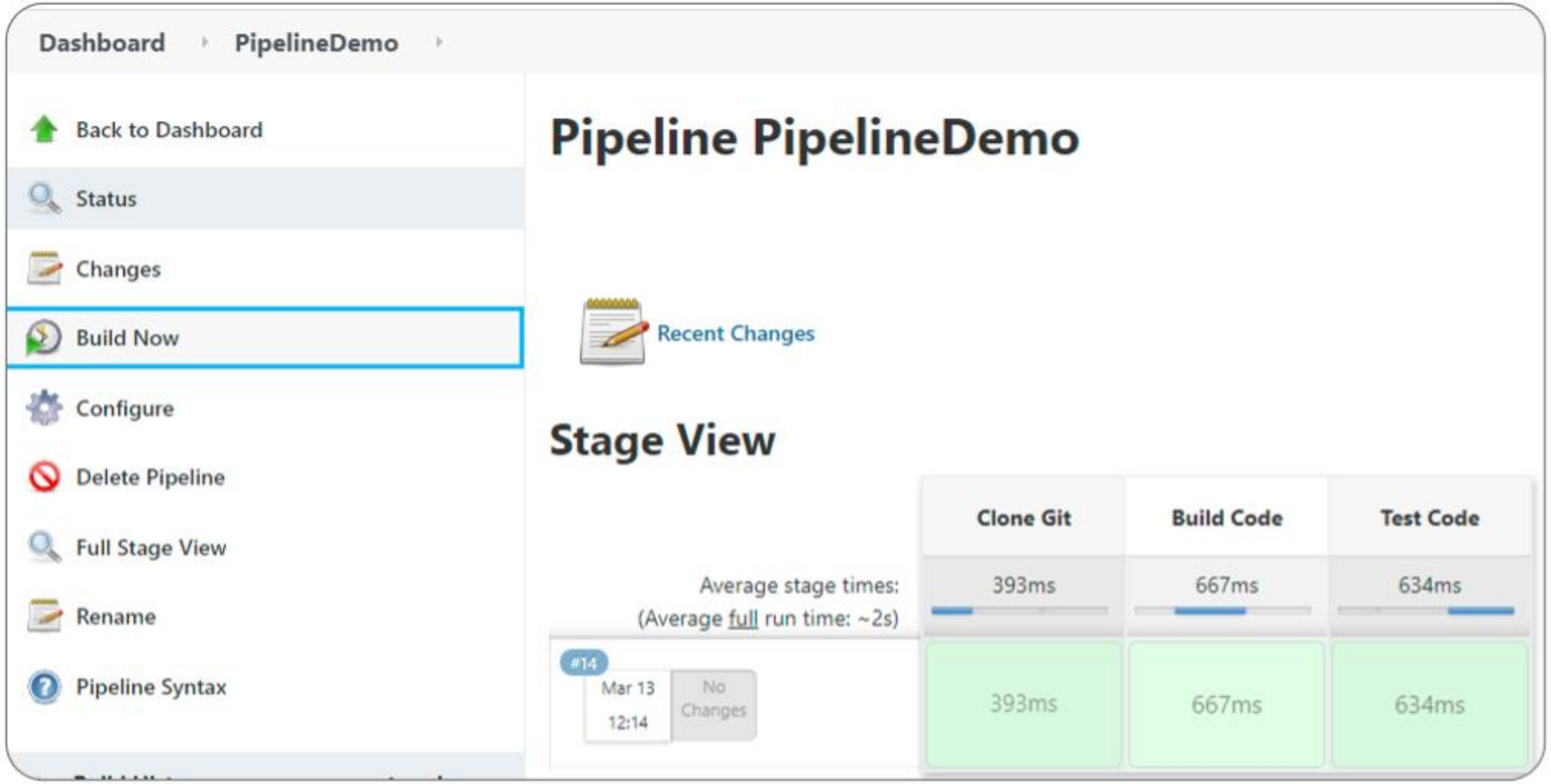


# Git Clone, Build and Test a Python Script

```
pipeline {
agent any
    stages {
        stage('Clone Git') { /*you can also specify git location */
            steps {
                git 'https://github.com/BThangaraju/Jenkins.git'
            }
        }
        stage('Build Code') {
            steps {
                sh "chmod u+x Prog1.py"
                sh "./Prog1.py"
            }
        }
        stage('Test Code') {
            steps {
                sh "chmod u+x Test.py"
                sh "./Test.py"
            }
        }
    }
}
```

You can execute any commands for example:
You can compile and execute a.c
sh “gcc a.c”
sh “./a.out”

# Pipeline View



# Jenkins Distributed Architecture

# Scaling Jenkins

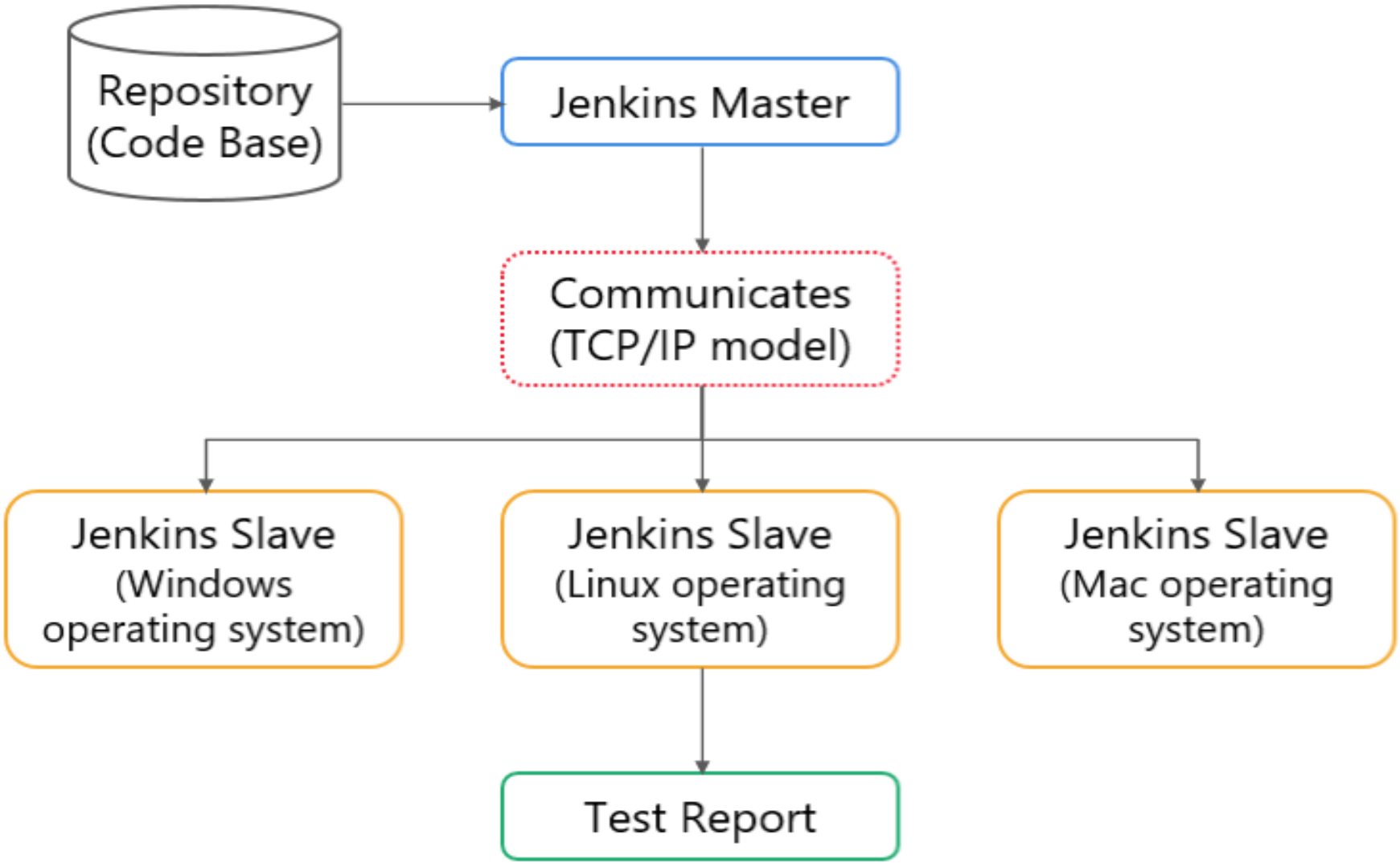
- Initially, when you start working with Jenkins, you have a single server to carry out all the tasks. The single Jenkins server is also called as Master node or Jenkins controller.

- Single Jenkins server is not enough to meet certain requirements like:
  - When you configure more jobs
  - When you orchestrate more frequent builds
  - When more developers depend on one controller
  - When you add incremental features in large and complex projects frequently
  - When you need different environments (diff OS) to test the build

# Scaling Jenkins

- Instead of adding new team members or new projects to an existing single Jenkins controller, you can create additional Jenkins controllers to accommodate new teams or projects.
- The Jenkins distributed architecture enables us to use various environments for each build project, dividing the workload across multiple agents running jobs concurrently.
- Jenkins' distributed architecture is based on the idea of ‘Master + Agent’. The master is responsible for coordination and providing the GUI and API endpoints, while the Agents perform the work.
- The Jenkins master manages the Jenkins agents and orchestrates their work by scheduling jobs on agents and monitoring them.
- Agents can link to the Jenkins controller via local or cloud computers.

# Jenkins Distributed Architecture



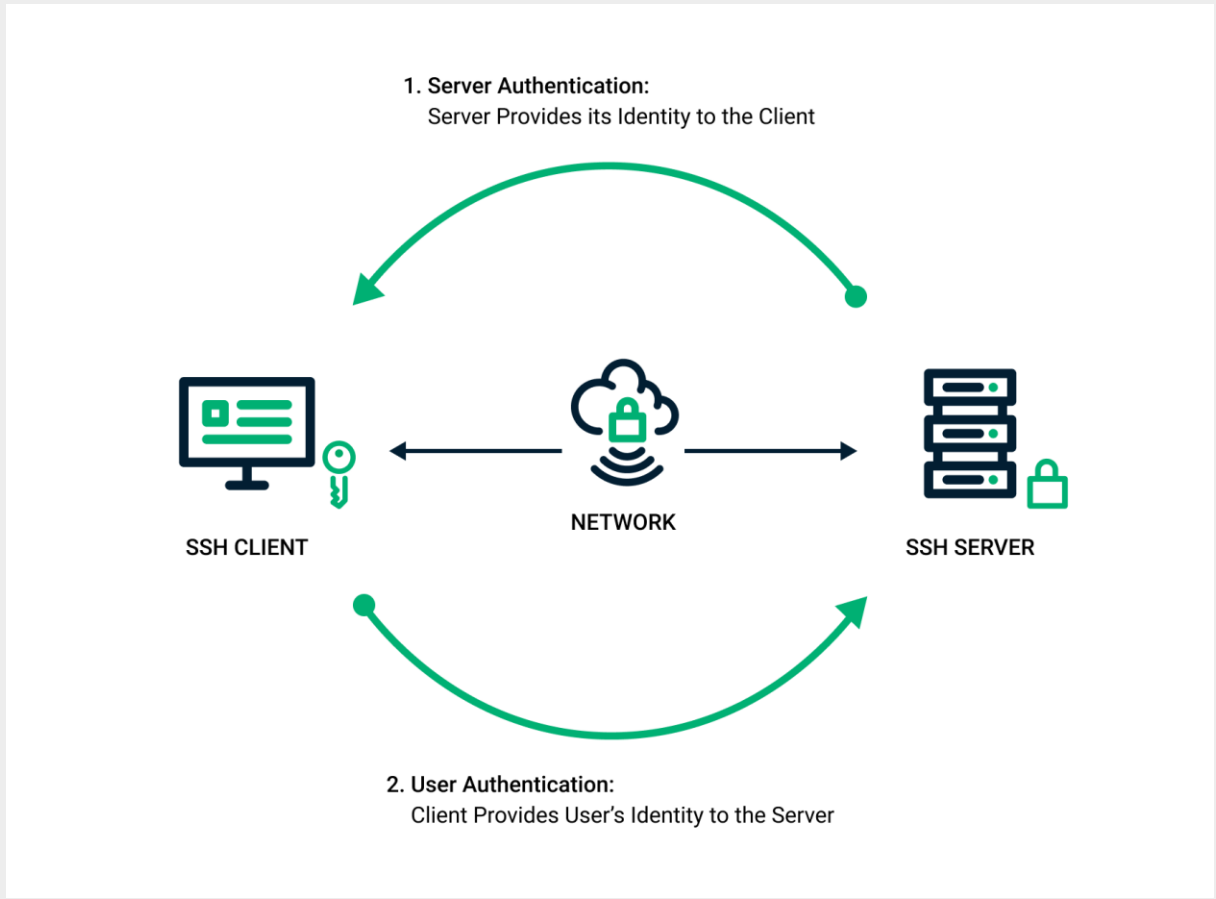
# SSH Key

- SSH key is a secure access credential used in the Secure Shell (SSH) protocol.
- SSH keys use key pairs based on public key infrastructure (PKI) technology, the gold standard for digital identity authentication and encryption, to provide a secure and scalable method of authentication.
- As the SSH protocol is widely used for communication in cloud services, network environments, file transfer tools, configuration management tools, and other computer-dependent services, to authenticate identity and protect those services from unintended use or malicious attacks.
- SSH keys not only improve security, but also enable the automation of connected processes, single sign-on (SSO), and identity and access management at scale that today's businesses require.

# SSH Key

- The SSH key pair : Private and Public key.  The public key is used by both the user and the remote server to encrypt messages.
- On the remote server side, it is saved in a file that contains a list of all authorized public keys.
- On the user's side, it is stored in SSH key management software or in a file on their computer.
- The private key remains only on the system being used to access the remote server and is used to decrypt messages.
- In computer security, challenge-response authentication is **a set of protocols used to protect digital assets and services from unauthorized users, programs or activities**.

# SSH Key




Ref: https://sectigo.com/

- When a user or process requests a connection to the remote server using the SSH client, a challenge-response sequence is initiated to complete authentication.
- The SSH server recognizes that a connection is being requested and sends an encrypted challenge request using the shared public key information.
- The SSH client then decrypts the challenge message and responds back to the server.
- This challenge-response sequence happens automatically between the SSH client and server without any manual action by the user.

# Host Setup

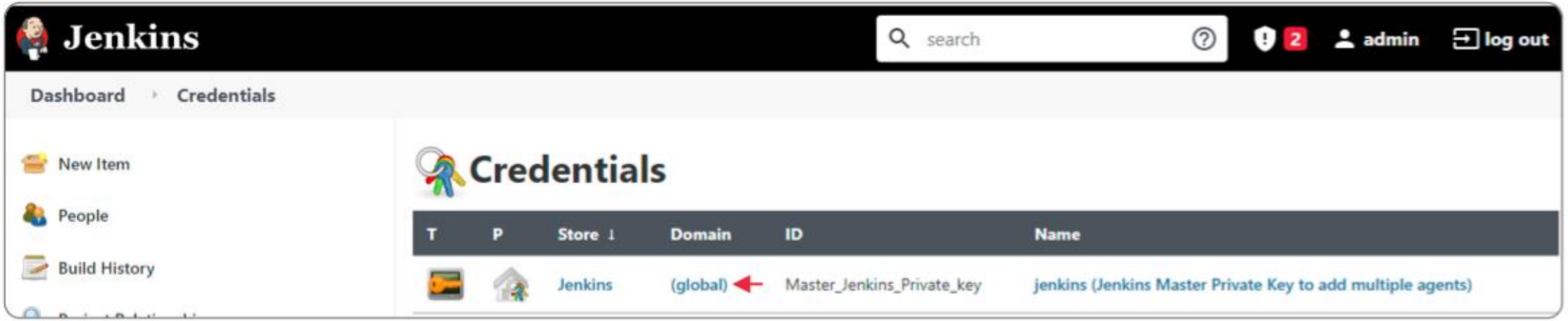
**From Host:**
#sudo su - jenkins
#ssh-keygen
#ls .ssh/; id_rsa  id_rsa.pub

**Docker Setup:**
#sudo apt-get install docker.io
#sudo service docker status
#sudo docker pull ubuntu
#sudo docker run --it --name Jenkins_Agent ubuntu /bin/bash

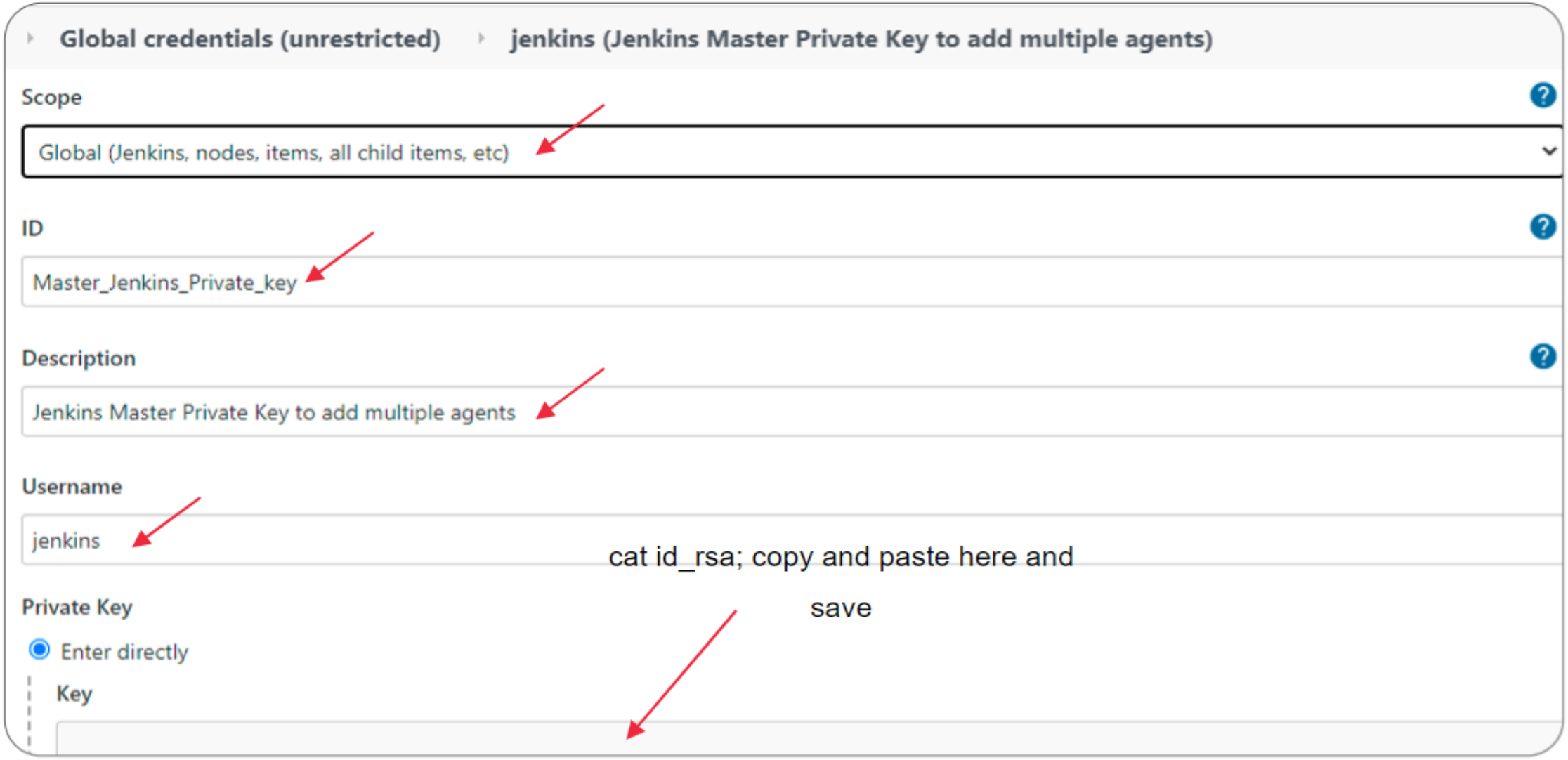
**In the browser**: Manage Jenkins -> Manage Credentials -> click global -> click Add Credentials -> for Kind - select ssh username with private key and enter the details.
**ID**: Master_Jenkins_Private_key
**Description**: Jenkins Master Private Key to Add Multiple Agents
**Username**: jenkins
**Private Key** -> Enter Directly ->copy and paste id_rsa from the server (/var/lib/jenkins/.ssh/id_rsa)

# Check the docker.service in Host

```
jenkins@ip-172-31-81-117:~/.ssh$ systemctl status docker.service
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; disabled; vendor preset: enabled)
   Active: active (running) since Sat 2021-03-20 02:16:40 UTC; 3h 49min ago
     Docs: https://docs.docker.com
 Main PID: 3527 (dockerd)
    Tasks: 10
   CGroup: /system.slice/docker.service
           └─3527 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
jenkins@ip-172-31-81-117:~/.ssh$
```



# Configure Jenkins Global Credentials



# Docker Configuration

**Inside the container:**

#adduser jenkins

#usermod -aG sudo jenkins

#apt-get update

#apt-get install sudo

#su - jenkins

**ssh server configuration**

#sudo apt-get install openssh-server

#sudo service ssh restart

#service ssh status

#sudo apt install openjdk-11-jdk

#java --version

**From Host:**

#ssh-copy-id jenkins@172.17.0.2

**To Check Docker container:**

#cat /home/jenkins/.ssh/authorized_keys

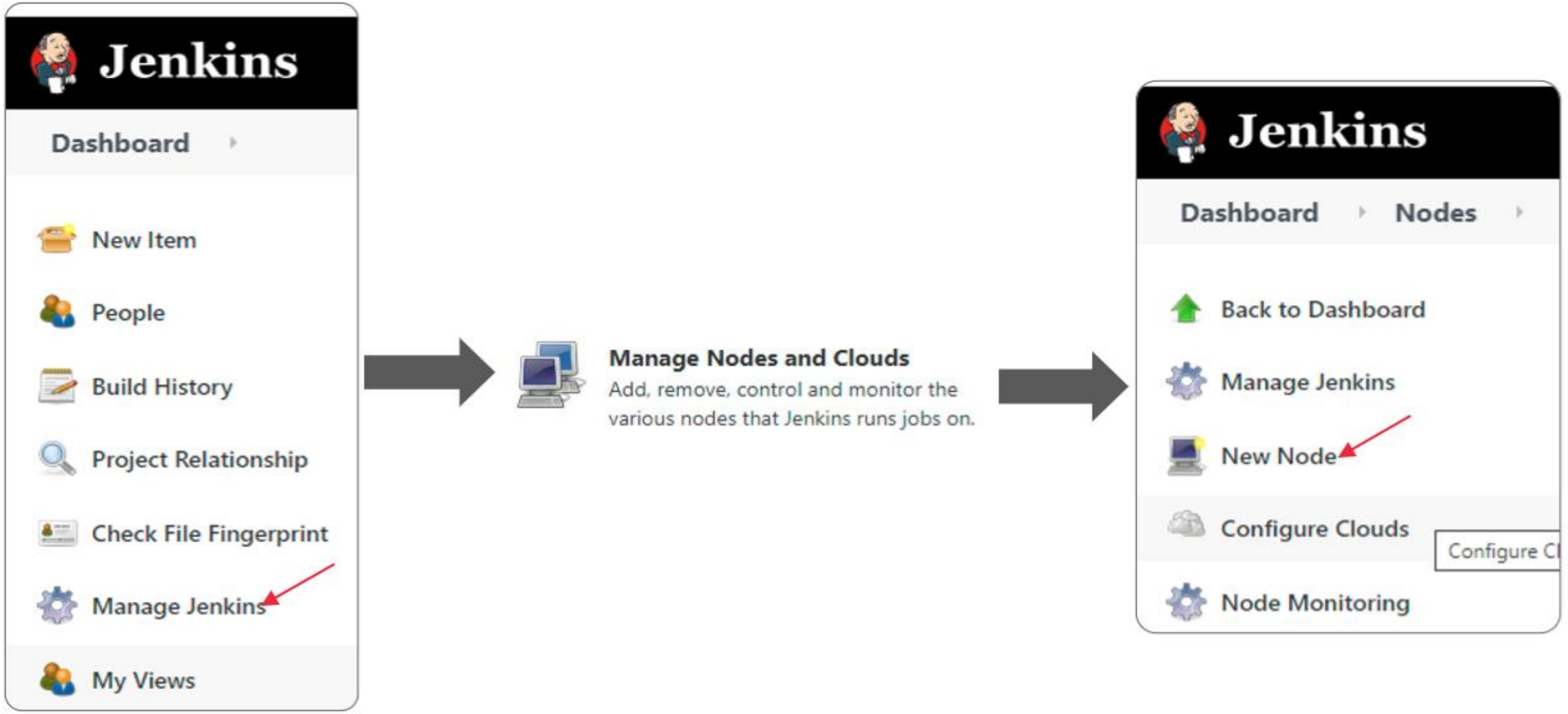
It should be same as: id_rsa.pub in host.

# Copy Authorized Keys From Host to Container

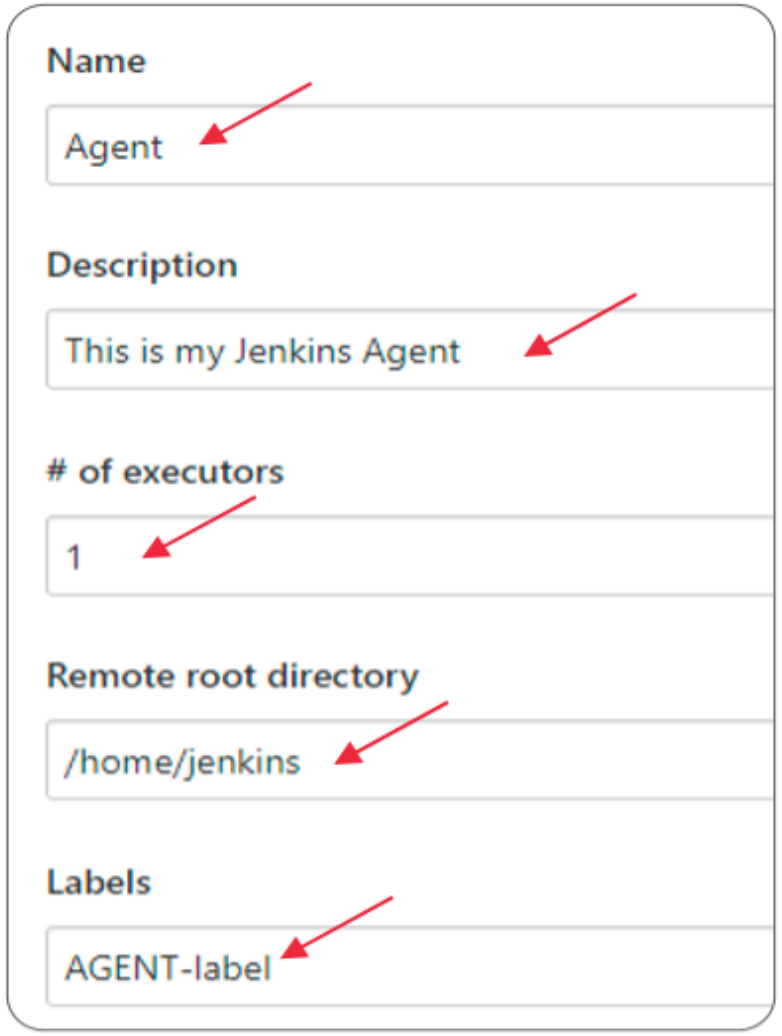
- In the Host: ssh-copy-id jenkins@172.17.0.2
- Check the copied authorized_keys in the container

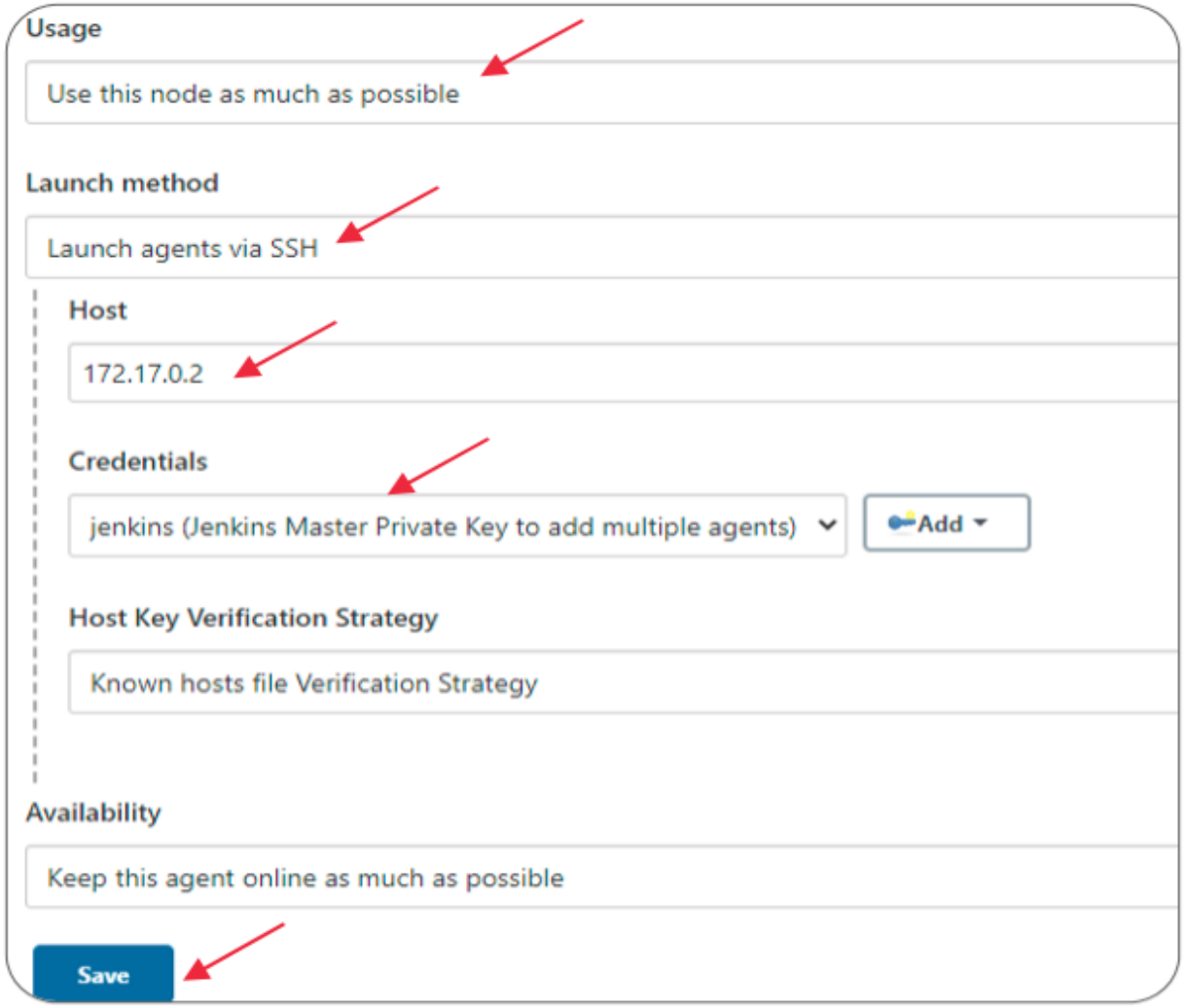
```
jenkins@9f9f046f179f:~$ pwd
/home/jenkins
jenkins@9f9f046f179f:~$ cd .ssh
jenkins@9f9f046f179f:~/.ssh$ ls
authorized_keys
jenkins@9f9f046f179f:~/.ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDj6rSfwrR3N+4
0qU3Hrpkb1hiPhYQ9otB6kGmEnTGnB5OAT/t32qsIBLVIx4zWC
ORxKqe6I6z+2CIftC+4FYSedOkFgznmOISv66FNWslUHipsPV8
2Vu/vHGaRDNuPN jenkins@ip-172-31-81-117
jenkins@9f9f046f179f:~/.ssh$
```

# Configure Agent in Jenkins

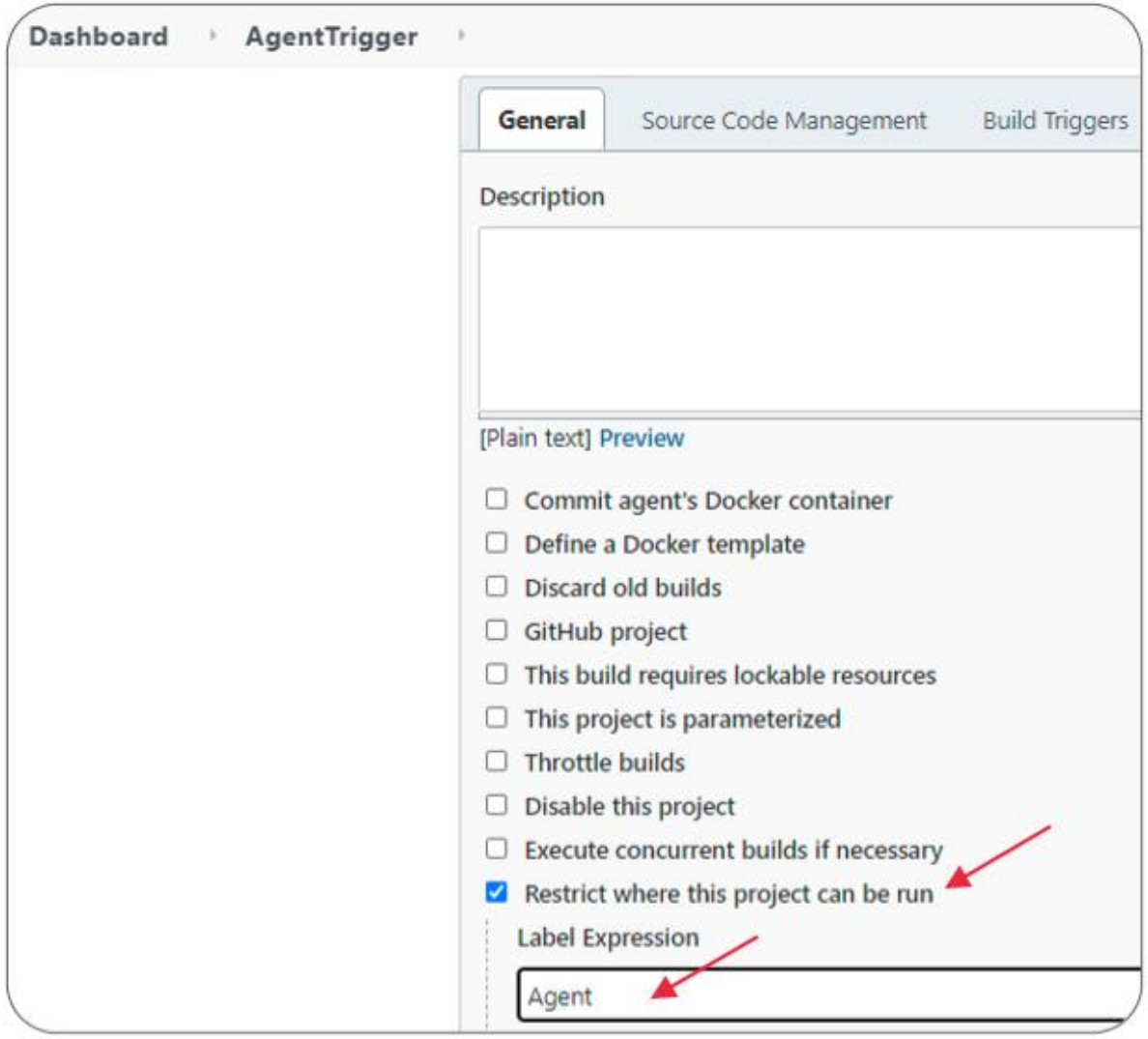


# Configure Agent in Jenkins



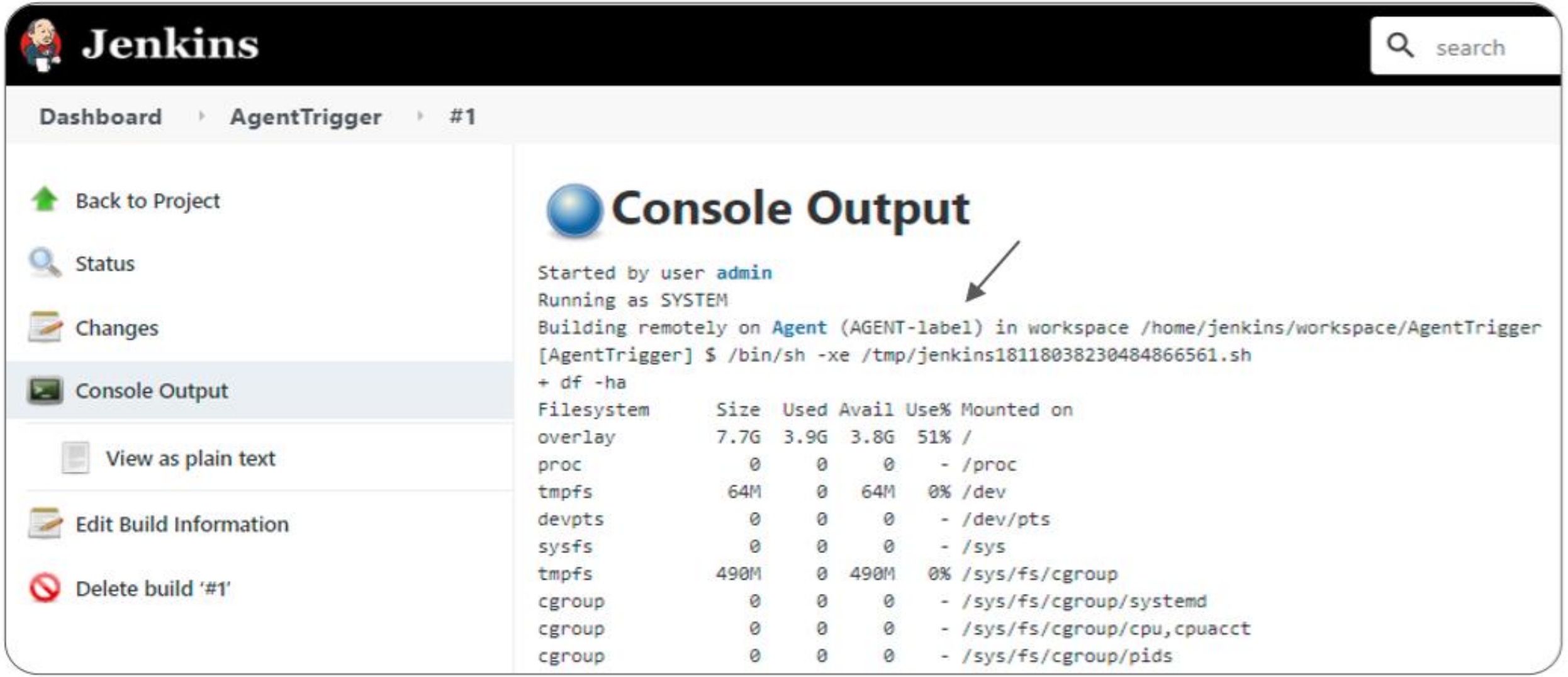



# Create a New Project as AgentTrigger




- Add build script as: df –ha
- Save and build manually

# Console Output



# Benefits of Jenkins Distributed Architecture

- Higher Performance
- High Availability
- Failover Mechanism
- Enhanced Security
- Rollback Mechanism from Machine Failure

# Thank You