

# Jenkins CI/CD

By Satya

# What is CI/CD

- ▶ CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment
- ▶ CI/CD is the most important part of DevOps that is used to integrate various DevOps stages. Jenkins is the most famous Continuous Integration tool

# What is Jenkins

- ▶ Jenkins is an open-source automation tool written in Java
- ▶ Jenkins achieves Continuous Integration with the help of plugins. Plugins allow the integration of Various DevOps stages. If you want to integrate a particular tool, you need to install the plugins for that tool.
- ▶ **Advantages of Jenkins:**
  - It is an open-source tool with great community support.
  - It is easy to install.
  - It has 1000+ plugins to ease your work.
  - It is free of cost.
  - It is built with Java and hence, it is portable to all the major platforms.

## ► What is a Jenkins pipeline

A pipeline is a collection of jobs that brings the software from version control into the hands of the end users by using automation tools.

The key feature of this pipeline is to define the entire deployment flow through code.

It basically follows the '**pipeline as code**' discipline. Instead of building several jobs for each phase, you can now code the entire workflow and put it in a **Jenkinsfile**. Below is a list of reasons why you should use the Jenkins Pipeline.

## ► Jenkins Pipeline Advantages

It models simple to complex pipelines as code by using **Groovy DSL** (Domain Specific Language)

The code is stored in a text file called the Jenkinsfile which can be **checked into a SCM** (Source Code Management)

Improves user interface by incorporating **user input** within the pipeline

It is durable in terms of unplanned restart of the Jenkins master

It can restart from saved **checkpoints**

It supports complex pipelines by incorporating conditional loops, fork or join operations and allowing tasks to be performed in parallel

It can integrate with several other plugins

## ► What is a Jenkinsfile

A **Jenkinsfile** is a text file that stores the entire workflow as code

The Jenkinsfile is written using the Groovy DSL. It is written based on two syntaxes

- Declarative pipeline syntax

```
pipeline {  
}
```

- Scripted pipeline syntax

```
node {  
}
```

This is a user defined block. All the stages and steps are defined within this block. It is the key block for a declarative pipeline

A node is a machine that executes an entire workflow. It is a key part of the scripted pipeline syntax.

#Declarative syntax

```
pipeline {  
  agent any  
  
  stages {  
    stage("Build") {  
      steps {  
        echo "Some code compilation here..."  
      }  
    }  
  
    stage("Test") {  
      steps {  
        echo "Some tests execution here..."  
        echo 1  
      }  
    }  
  }  
}
```

#Scripted syntax

```
node {  
    stage("Build") {  
        echo "Some code compilation here..."  
    }  
  
    stage("Test") {  
        echo "Some tests execution here..."  
        echo 1  
    }  
}
```

- Once Jenkins installed successfully, Lunch Jenkins

`http://<Instance ip address>:8080/`

# Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (**not sure where to find it?**) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

**Administrator password**

```
# cat /var/lib/jenkins/secrets/initialAdminPassword  
bd64c9d595c54b3eab2e32027a15cf15
```



# Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

## Install suggested plugins

Install plugins the Jenkins community finds most useful.

## Select plugins to install

Select and install plugins most suitable for your needs.

# Create First Admin User

Username:

Password:

Confirm password:

Full name:

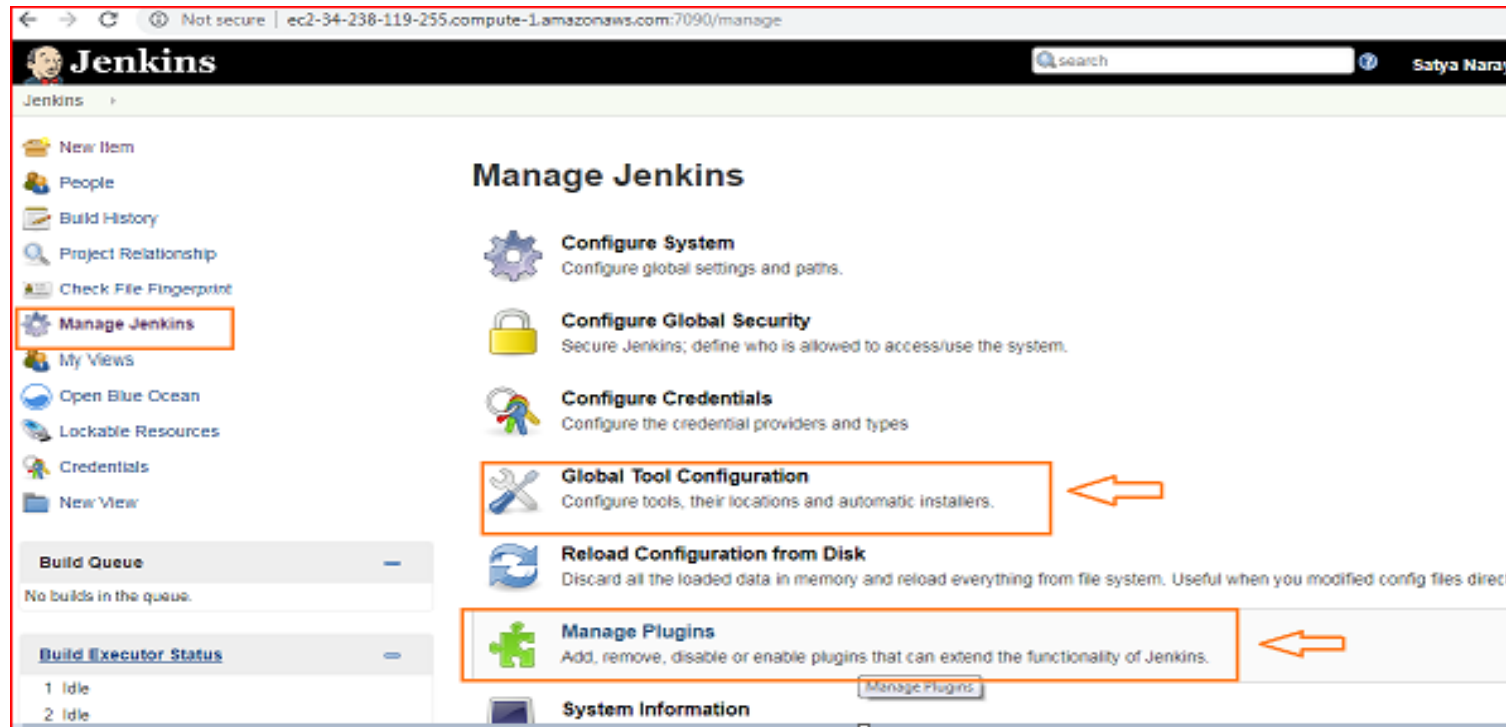
E-mail address:

## ► Configure Jenkins

- After Jenkins is installed, JDK, Maven and Githas to be configured

Follow the below steps to start with configuration.

Go to Jenkins >> Manage Jenkins >> Global Tool configuration >>



Add JDK, Git, Maven path as below >> Apply and Save

Jenkins > Global Tool Configuration

**JDK**

JDK installations

Add JDK

[[ JDK

Name JAVA

JAVA\_HOME /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.181-3.b13.el7\_5.x86\_64/

☐ Install automatically

Delete JDK

Add JDK

List of JDK installations on this system

**Git**

Git installations

[[ Git

Name Default

Path to Git executable git

☐ Install automatically

Delete Git

**Maven**

Maven installations

Add Maven

[[ Maven

Name maven

MAVEN\_HOME /opt/maven/

☐ Install automatically

Delete Maven

Save Apply

# Jenkins Jfrog integration

Manage Jenkins -> Manage Plugins -> Available -> Artifactory

## Artifactory

Artifactory servers

☐ Use the Credentials Plugin 

 Artifactory

Server ID  

URL  

Advanced...

### Default Deployer Credentials

Username  

Password  

Found Artifactory 4.15.0

Test Connection

☐ Use Different Resolver Credentials

Delete

Save

Apply

## ► Installing Plugins

Go to Jenkins >> Manage Jenkins >> Manage plugins >>

 New Item

 People

 Build History

 **Manage Jenkins**

 Credentials

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

## Manage Jenkins



[Configure System](#)

Configure global settings and paths.



[Configure Global Security](#)

Secure Jenkins; define who is allowed to access/use the system.



[Configure Credentials](#)

Configure the credential providers and types



[Reload Configuration from Disk](#)

Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.



[Manage Plugins](#)

Add, remove, disable or enable plugins that can extend the functionality of Jenkins. **(updates available)**

## ► Creating your first Jenkins pipeline.

**Step 1:** Log into Jenkins and select 'New item' from the dashboard.




**Step 2:** Next, enter a name for your pipeline and select 'pipeline' project. Click on 'ok' to proceed.

Jenkins > All >


### Enter an item name

PIPELINE DEMO


*\* Required field*

**Freestyle project**

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

**Pipeline**

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

**Multi-configuration project**

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



**Step 3:** Scroll down to the pipeline and choose if you want a declarative pipeline or a scripted one.

Jenkins > PIPELINE DEMO >

GeneralBuild TriggersAdvanced Project OptionsPipeline

Pipeline

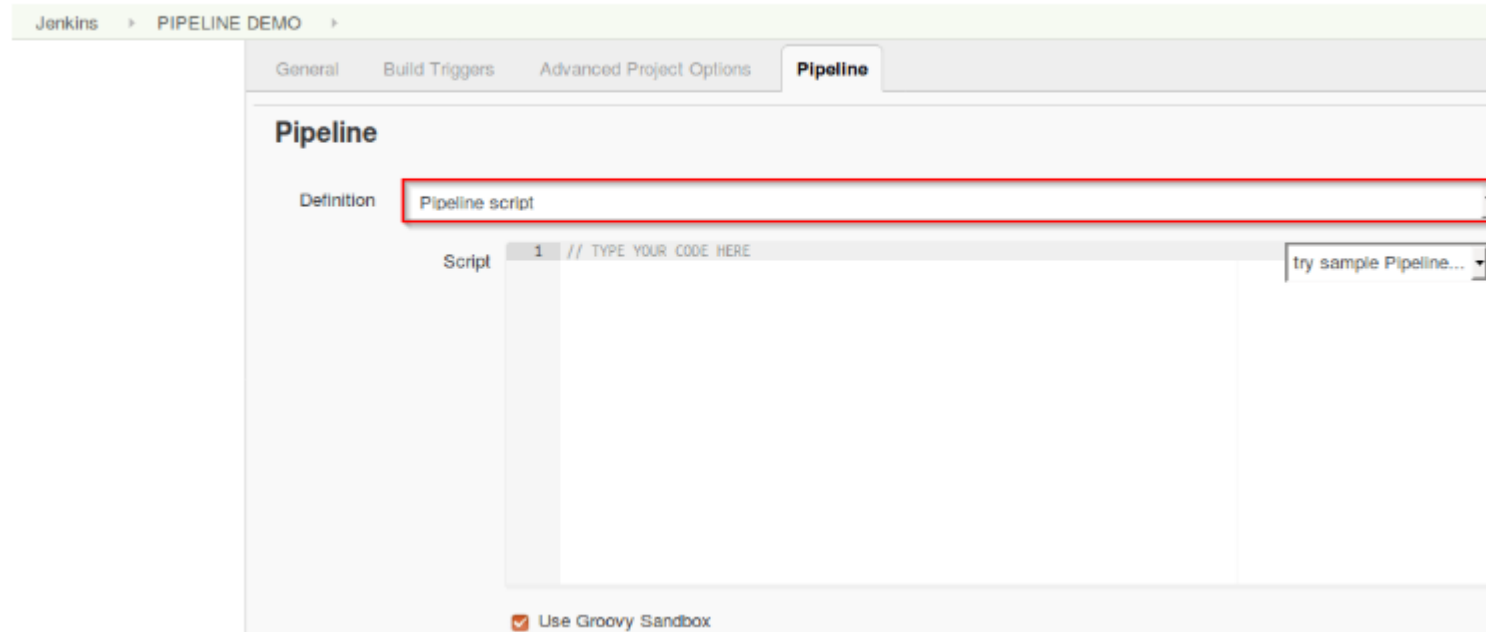
Definition

Pipeline script

Pipeline script from SCM

☒ Use Groovy Sandbox

**Step 4a:** If you want a scripted pipeline then choose 'pipeline script' and start typing your code.



The screenshot shows the Jenkins configuration page for a project named 'PIPELINE DEMO'. The 'Pipeline' tab is selected, and the 'Definition' dropdown menu is set to 'Pipeline script', which is highlighted with a red rectangle. Below this, the 'Script' section contains a text area with the placeholder text '// TYPE YOUR CODE HERE'. To the right of the text area is a button labeled 'try sample Pipeline...'. At the bottom of the page, there is a checkbox labeled 'Use Groovy Sandbox' which is checked.

Jenkins > PIPELINE DEMO >

General Build Triggers Advanced Project Options **Pipeline**

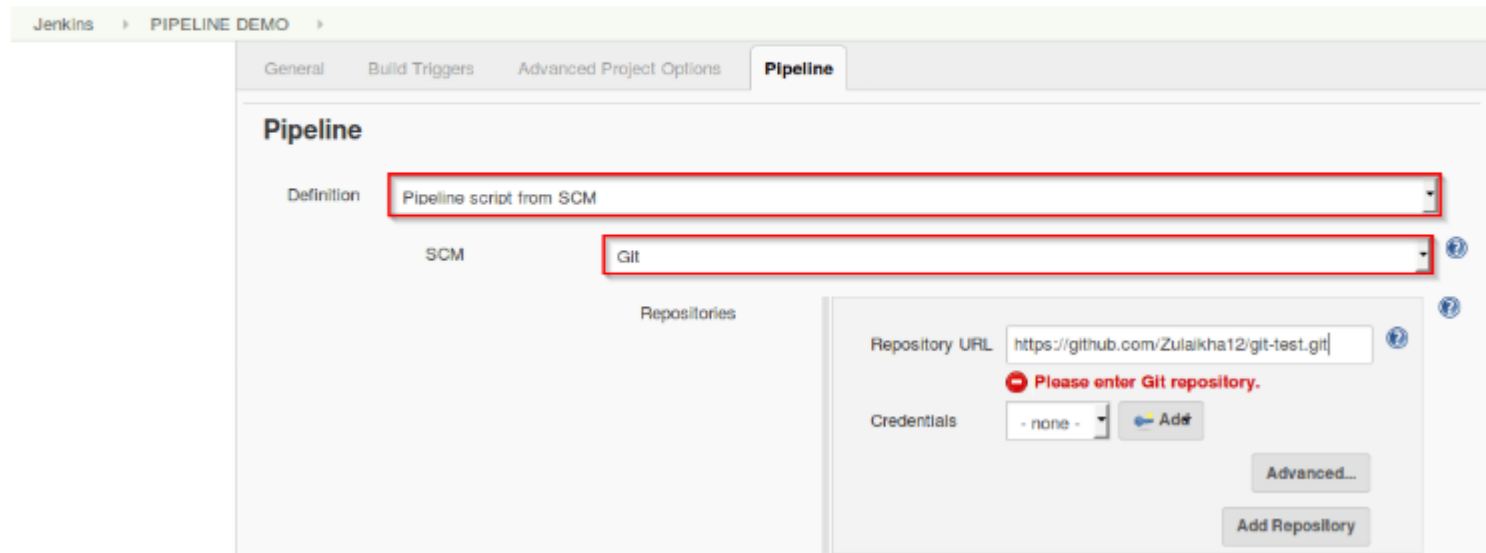
**Pipeline**

Definition Pipeline script

Script 1 // TYPE YOUR CODE HERE try sample Pipeline...

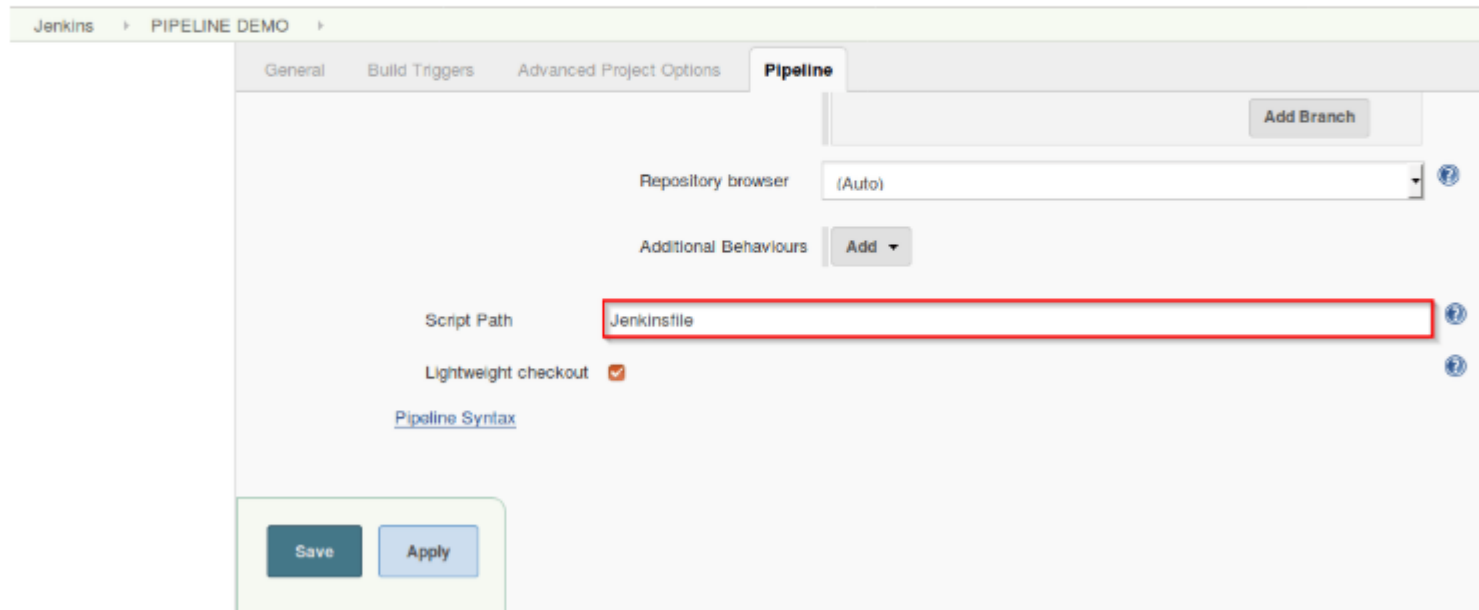
☒ Use Groovy Sandbox

**Step 4b:** If you want a declarative pipeline then select 'pipeline script from SCM' and choose your SCM. In my case I'm going to use Git throughout this demo. Enter your repository URL.



The screenshot shows the Jenkins web interface for configuring a pipeline. The breadcrumb navigation at the top indicates the path: Jenkins > PIPELINE DEMO >. Below this, there are tabs for 'General', 'Build Triggers', 'Advanced Project Options', and 'Pipeline', with 'Pipeline' being the active tab. The main section is titled 'Pipeline'. Under the 'Definition' label, a dropdown menu is set to 'Pipeline script from SCM'. Below this, under the 'SCM' label, a dropdown menu is set to 'Git'. Under the 'Repositories' label, there is a form with a 'Repository URL' field containing 'https://github.com/Zulaikha12/git-test.git'. Below the URL field is a red error message: 'Please enter Git repository.' with a red circle icon. There is also a 'Credentials' dropdown set to '- none -' with an 'Add' button next to it. At the bottom right of the form are two buttons: 'Advanced...' and 'Add Repository'.

**Step 5:** Within the script path is the name of the Jenkinsfile that is going to be accessed from your SCM to run. Finally click on 'apply' and 'save'. You have successfully created your first Jenkins pipeline.



The screenshot shows the Jenkins web interface for configuring a pipeline. The breadcrumb navigation at the top reads 'Jenkins > PIPELINE DEMO >'. Below this, there are tabs for 'General', 'Build Triggers', 'Advanced Project Options', and 'Pipeline', with the 'Pipeline' tab currently selected. The 'Pipeline' tab contains several configuration options: a 'Repository browser' dropdown menu set to '(Auto)', an 'Additional Behaviours' section with an 'Add' button, a 'Script Path' text input field containing 'Jenkinsfile' (highlighted with a red border), and a 'Lightweight checkout' checkbox which is checked. There is also a link for 'Pipeline Syntax'. At the bottom left, there are two buttons: 'Save' and 'Apply'.

## ► Pipeline code validation at startup

```
pipeline {  
  agent any  
  
  stages {  
    stage("Build") {  
      steps {  
        echo "Some code compilation here..."  
      }  
    }  
  
    stage("Test") {  
      steps {  
        echo "Some tests execution here..."  
        echo 1  
      }  
    }  
  }  
}
```

Branch: —

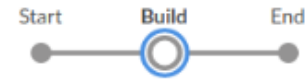
<1s

No changes

Commit: —

a minute ago

Started by user Szymon Stepniak



## Build



No steps This stage has no steps

```

1 Started by user Szymon Stepniak
2 Running in Durability level: MAX_SURVIVABILITY
3 org.codehaus.groovy.control.MultipleCompilationErrorsException: startup failed:
4 WorkflowScript: 14: Expecting "class java.lang.String" but got "1" of type class java.lang.Integer instead @ line 14, column 22.
5     echo 1
6         ^
7
8 1 error
9
10 at org.codehaus.groovy.control.ErrorCollector.failIfErrors(ErrorCollector.java:310)
11 at org.codehaus.groovy.control.CompilationUnit.applyToPrimaryClassNodes(CompilationUnit.java:1085)
12 at org.codehaus.groovy.control.CompilationUnit.doPhaseOperation(CompilationUnit.java:603)
13 at org.codehaus.groovy.control.CompilationUnit.processPhaseOperations(CompilationUnit.java:581)
  
```

```
node {  
    stage("Build") {  
        echo "Some code compilation here..."  
    }  
  
    stage("Test") {  
        echo "Some tests execution here..."  
        echo 1  
    }  
}
```

pipeline-sandbox < 72

Pipeline Changes Tests Artifacts

Branch: — <1s No changes  
Commit: — a few seconds ago Started by user Szymon Stepniak

Start Build Test End

Test - <1s

✓ > Some tests execution here... — Print Message

✗ 1 — Print Message

1 Could not instantiate {message=1} for org.jenkinsci.plugins.workflow.steps.EchoStep: java.lang.ClassCastException: org.jenkinsci.plugins.workflow.steps.EchoStep.message expects class java.lang.String but received class java.lang.Integer

The declarative pipeline in this case handles such use case much better.

## ► Restart from stage

Another cool feature that only declarative pipeline has is "Restart from stage"

```
pipeline {  
  agent any  
  
  stages {  
    stage("Build") {  
      steps {  
        echo "Some code compilation here..."  
      }  
    }  
  
    stage("Test") {  
      steps {  
        echo "Some tests execution here..."  
      }  
    }  
  }  
}
```





Jenkins ▶ scripted-vs-declarative-pipeline ▶ #5 ▶ Restart from Stage





- 📈 Back to Project
- 🔍 Status
- 📝 Changes
- 💻 Console Output
- 📝 Edit Build Information
- 🚫 Delete build '#5'
- 🌊 Open Blue Ocean
- 🎀 Build timeline
- 🔄 **Restart from Stage**
- 🔄 Replay


## Restart #5 from Stage

Stage Name 


Build ▼  
Build  
Test

Run


-  Full Stage View
-  Open Blue Ocean
-  Rename
-  Pipeline Syntax

 **Build History** [trend](#)


x

 **#6**

Jun 1, 2020 1:13 AM

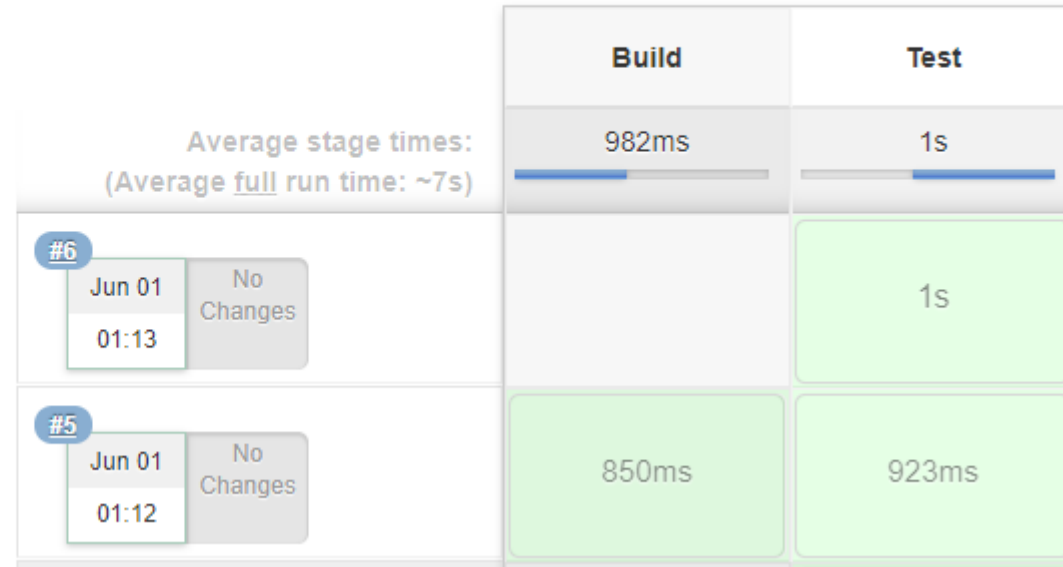
 **#5**

Jun 1, 2020 1:12 AM

 **#4**

Jun 1, 2020 1:12 AM

## Stage View



No restart option you can see in scripted pipeline.

---

# Q&A