

**Practice for Lesson 4:  
Integrating Jenkins Pipelines  
with Jobs**

## Practices for Lesson 4

---

### Overview

In these practices, you will trigger the Jenkins Jobs from one job to another automatically in the Jenkins Dashboard, create the pipeline using Groovy script in Jenkins instance and learn to integrate the GitHub source code to the Jenkins pipeline, and further delegate a pipeline Job using Agent in Jenkins Instance.

## Practice 4-2: Create Pipeline Using Groovy Script in Jenkins

### Overview

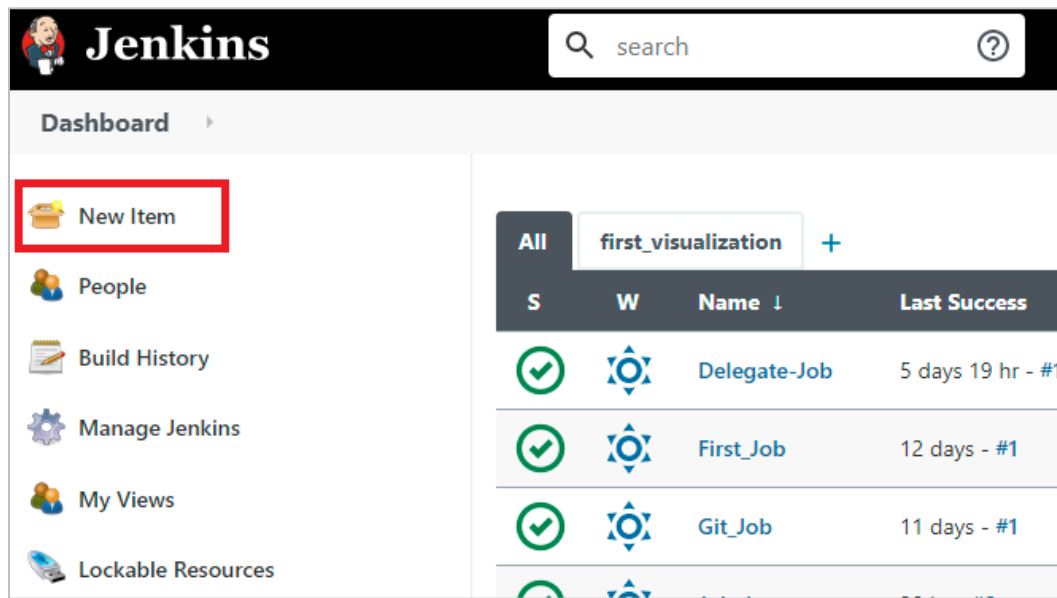
In this practice, you will learn how to create the pipeline using Groovy scripting in Jenkins instance using a sample example.

### Assumptions

You should have completed the Practice of Lesson 4-1.

### Tasks

1. Create a scripted pipeline using the Groovy scripting in Jenkins instance.
  - a. In the Jenkins Dashboard, navigate to main menu and select **New Item** to create a Groovy Scripted Pipeline as shown below.



- b. Provide the **name** for the Pipeline, select **Pipeline** and click **OK** as shown below.

**Enter an item name**

Scripted\_Pipeline

» 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.

**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.

**OK**

- c. Scroll down to **Pipeline**, select **Pipeline script** under **Definition** as shown below.

General Build Triggers **Advanced Project Options** Pipeline

**Advanced Project Options**

Advanced...

**Pipeline**

Definition

Pipeline script

Script

1

try sample Pipeline...

- d. Copy the Groovy script provided below and paste it in the **Script** block as shown below. Select the checkbox of **Use Groovy Sandbox** and click **Save**.

```

pipeline {
    agent any
    stages {
        stage('Compile') {
            steps {
                echo "Compiled Successfully!!";
            }
        }
        stage('JUnit') {
            steps {
                echo "JUnit Passed Successfully!";
            }
        }
        stage('Quality-Gate') {
            steps {
                echo "SonarQude Quality Gate passed
successfully!!";

                /*sh exit ("1");*/
            }
        }
        stage('Deploy') {
            steps {
                echo "Pass!";
            }
        }
    }
    post {
        always {
            echo 'This will always run'
        }
        success {
            echo 'This will run only if successful'
        }
        failure {
            echo 'This will run only if failed'
        }
        unstable {
            echo 'This will run only if the run was marked as
unstable'
        }
        changed {

```

```

        echo 'This will run only if the state of the pipeline
has changed'

        echo 'For example, if the Pipeline was previously
failing but is now successful'
    }
}
}
}

```

General Build Triggers Advanced Project Options **Pipeline**

### Pipeline

Definition

Pipeline script

Script

```

1 pipeline {
2   agent any
3   stages {
4     stage('Compile') {
5       steps {
6         echo "Compiled Successfully!!";
7       }
8     }
9     stage('JUnit') {
10      steps {
11        echo "JUnit Passed Successfully!";
12      }
13    }
14    stage('Quality-Gate') {
15      steps {
16        echo "SonarQude Quality Gate passed successfully!!";
17        /*sh exit ("1");*/
18      }
19    }
20  }
21 }

```

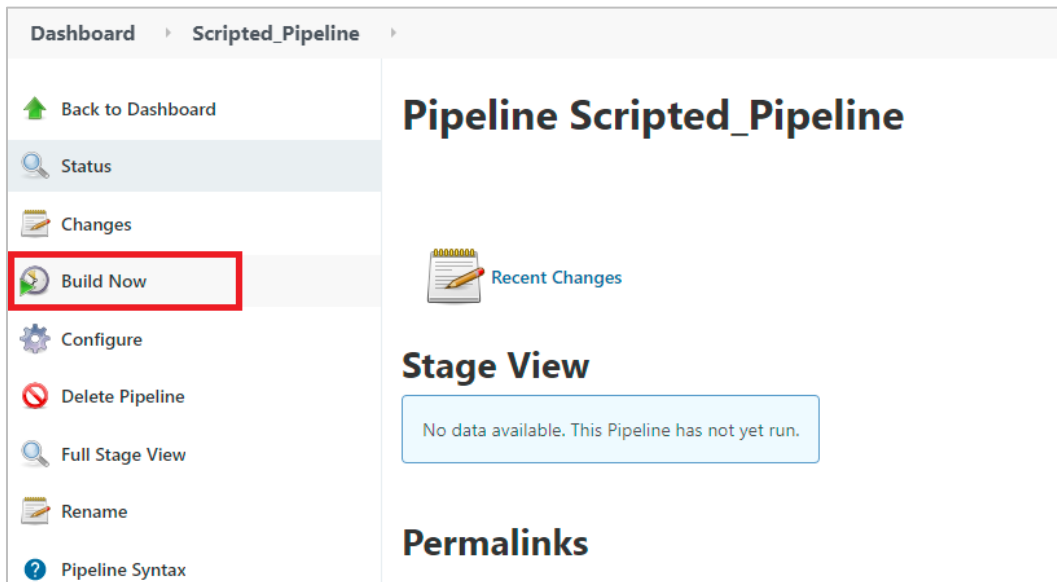
try sample Pipeline...

☒ Use Groovy Sandbox

Pipeline Syntax

**Save** Apply

- e. Scripted pipeline is created successfully. Click **Build Now** to execute the pipeline as shown below.



- f. As shown below, **view** the **stages** of **pipeline** getting executed and click on the link under **Build History** as shown below.

Compile	JUnit	Quality-Gate	Deploy	Declarative: Post Actions
306ms	96ms	85ms	85ms	116ms

- g. Navigate to **Console Output** to view the execution output of the scripted pipeline.

Dashboard
Scripted\_Pipeline
#1

Back to Project
Status
Changes

Console Output

View as plain text

Edit Build Information
Delete build '#1'
Restart from Stage
Replay
Pipeline Steps
Workspaces

## Console Output

Started by user [oralabuser](#)  
Running in Durability level: MAX\_SURVIVABILITY  
[Pipeline] Start of Pipeline  
[Pipeline] node  
Running on **Jenkins** in /var/lib/jenkins/workspace/Scripted\_Pipeline  
[Pipeline] {  
[Pipeline] stage  
[Pipeline] { (Compile)  
[Pipeline] echo  

Compiled Successfully!!

  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] stage  
[Pipeline] { (JUnit)  
[Pipeline] echo  

JUnit Passed Successfully!

  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] stage  
[Pipeline] { (Quality-Gate)  
[Pipeline] echo  

SonarQube Quality Gate passed successfully!!

  
[Pipeline] }  
[Pipeline] // stage  
[Pipeline] stage

h. As shown below, the scripted pipeline sample is executed successfully.

```

[Pipeline] echo
Pass!
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
This will always run
[Pipeline] echo
This will run only if the state of the pipeline has changed
[Pipeline] echo
For example, if the Pipeline was previously failing but is now successful
[Pipeline] echo
This will run only if successful
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline


Finished: SUCCESS


```

- Keep the Jenkins Dashboard and the AWS Management Console open for the next practice.