Tutorial

Jenkins

Purpose of Jenkins

The purpose of Jenkins is to provide an environment that makes the developing, testing, and deployment of an application easy and automatic. It does this by monitoring your code repository for changes and running any changes to the code through a pipeline of tests and/or other checks to make sure the code is ready for release. It then takes the verified software and automatically deploys it to the production environment.

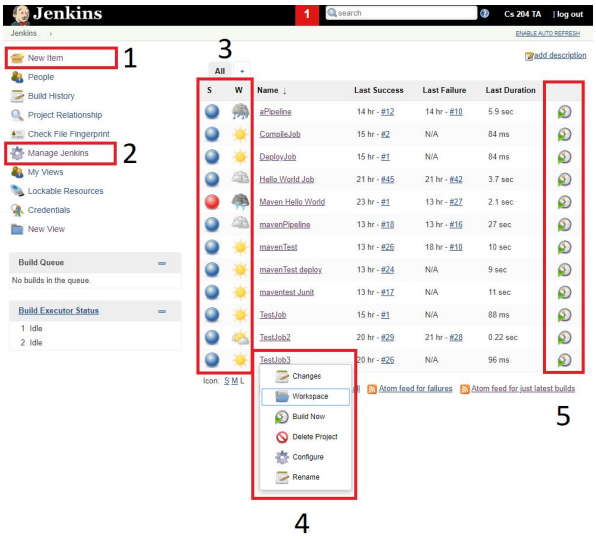
In this tutorial, you will learn/do the following:

- A basic understanding of Jenkins UI
- How to create a Jenkins Job and manually create a pipeline
- How to create a Jenkins Pipeline using a pipeline script
- How to integrate and use these tools with IntelliJ

In this tutorial, parts of the normal Jenkins process will be simplified. Code will not be deployed to separate testing and production servers. Instead the tests will be run on the same AWS EC2 instance that Jenkins is installed on, and the code will be packaged by creating a jar or war file using Maven and archiving it in Jenkins along with the source code.

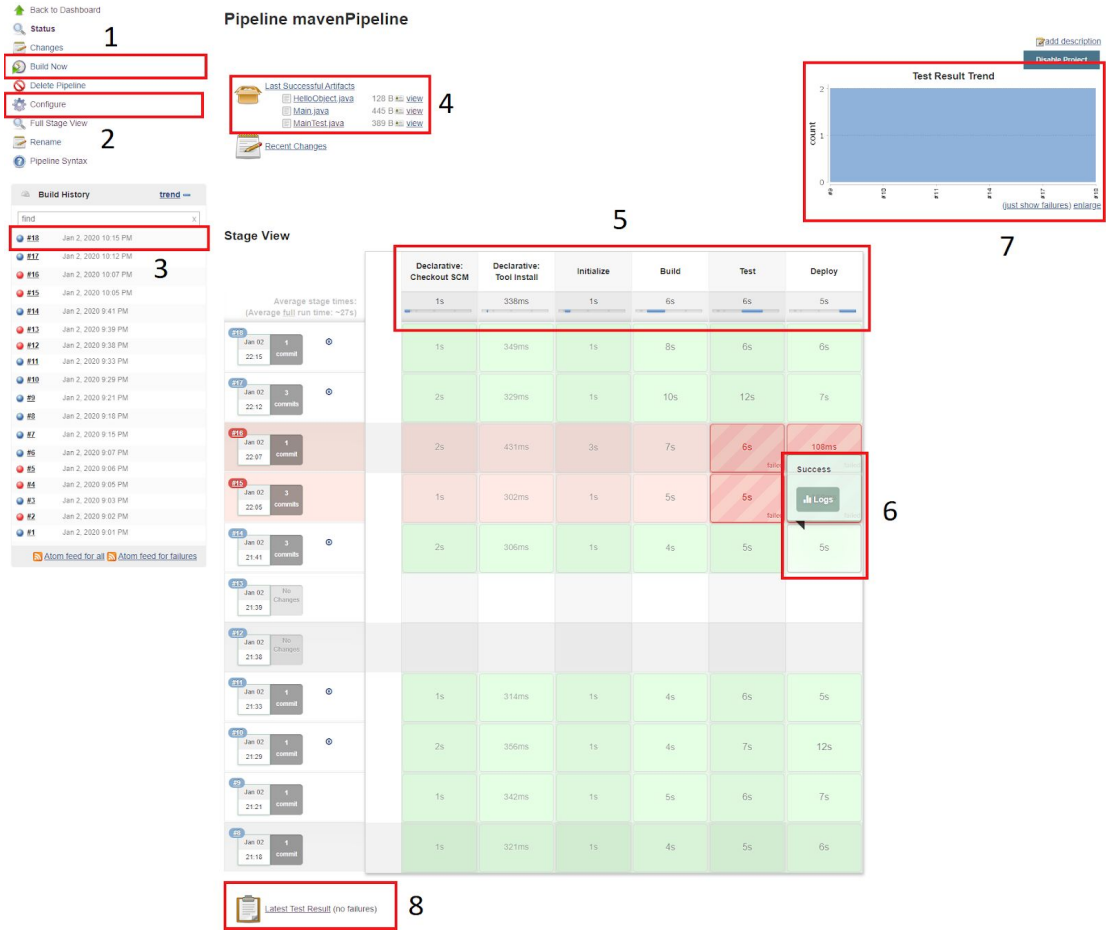
A Quick Overview of Jenkins UI

MAIN DASHBOARD



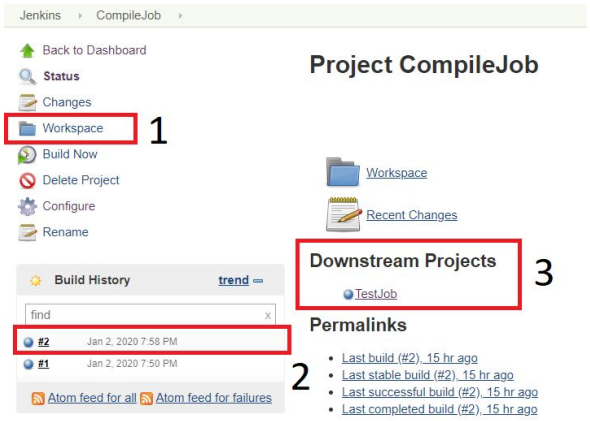
1. Directs you to the new job Menu
2. Directs you to the Jenkins settings Menu
3. Relays the status of your Build
    a. Blue means the build was successful
    b. Red means the build failed
4. Options available from the main dashboard
5. Allows you to run the job from the menu

PIPELINE MENU



1. Allows you to build the pipeline from the pipeline menu
2. Allows you to configure the pipeline
3. Allows you to access data about specific builds
4. Allows you to download artifacts from the menu that you chose to archive
5. Shows the different stages of the pipeline
6. Allows you to show logs for specific stages of the pipeline
7. Shows a graphic of the test results
8. Allows you to get a detailed report of any testing that was done

NORMAL JOB MENU



1. Allows you to view the files Jenkins used for the job
2. Allows you to access specific data about each build
3. Shows which jobs are triggered by running this job

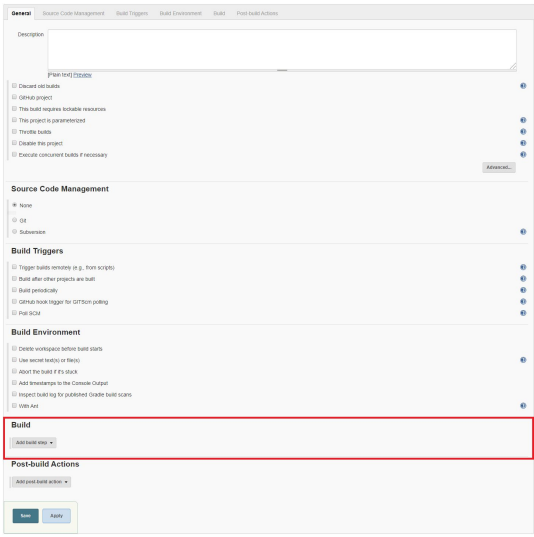
Start your AWS instance that has Jenkins Installed

1. Navigate to your EC2 AWS Dashboard https://aws.amazon.com/ec2/
2. Find the instance you installed Jenkins on, right click it , go to Instance state, Click Start

Note: *There can be a cost associated with using EC2. AWS allows certain services to be run within a "free tier" for the first year of use of your AWS account, so most students can run an EC2 instance for free. However, if you have used up your free tier eligibility (by having created an AWS account more than a year ago, possibly for CS 260 or another class) you could be charged. However, the charges are small. If you create a EC2 t2.micro server for use in this lab, and leave it running for an entire week, you will be charged approximately $1.95 if you are not still eligible for the free tier. We recommend creating an EC2 t2.micro instance as you do this pre-class assignment and then stopping (but not terminating) your instance when you are done. You can then restart it when you need for the next two weeks when you do the two Jenkins tutorials for this class.*

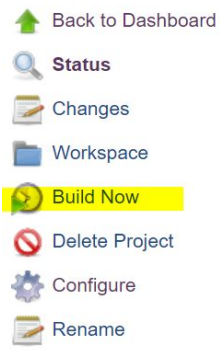
Managing and Creating Jobs

A Jenkins job is the basic setup for any Jenkins action. Jobs tell Jenkins what to , when to do it , and how to do it.

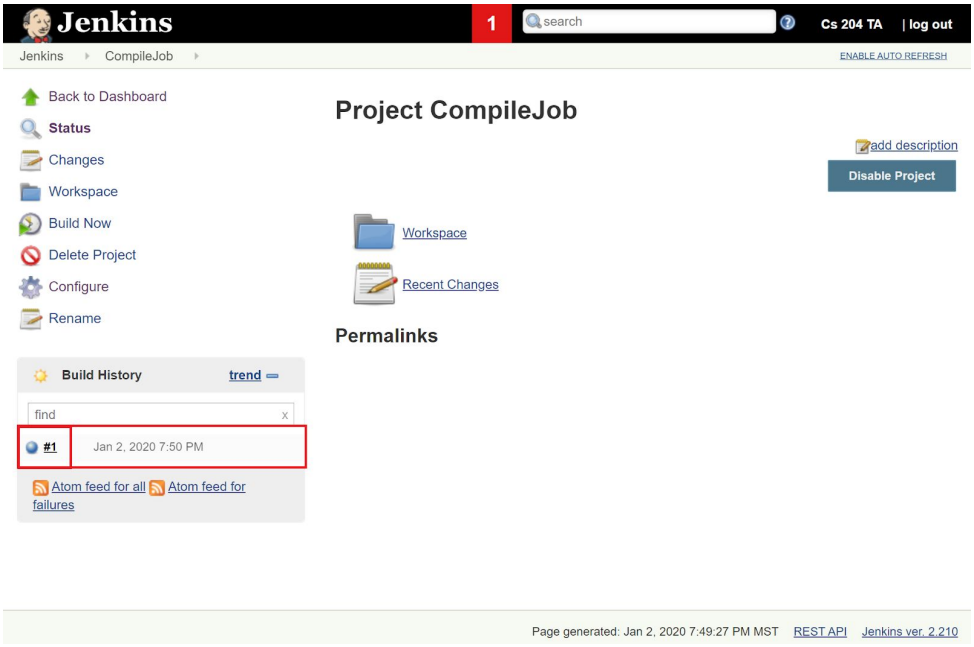


1. **Create a simple job**
    a. Navigate to http://your-EC2-IPt:8080/ and sign in using the credentials you signed up with when you initially installed jenkins
    b. **Click on "new item"** located on the top left part of the menu
    c. Type "CompileJob" for the item name
    d. Click Freestyle project, then ok

    e. While in the project configuration -> **Go to Build**
    f. Click **Add build step**
        i. Click **Execute Shell**
    g. Type "echo Compile"to output "Compile" on the terminal when the job is built.
    h. Apply and save.
    i. **Click on the "Build Now"** button to run the job.



j. Your screen should now look like the screenshot below.

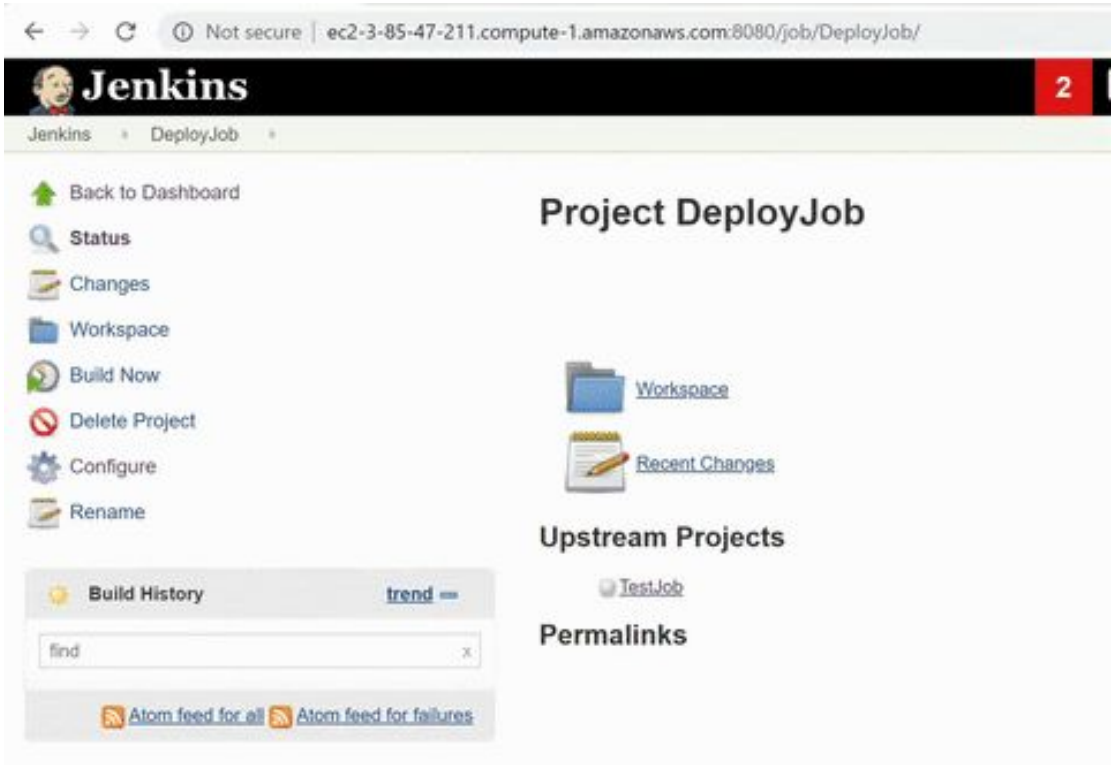


k. If the status light next to the build number is blue, it means the build was a success
    i. If it is red, it means the build failed

l. To see the console output for each build attempt, click on the build number then click on “Console Output” found on the left menu of each build.

2. **Modifying Build Triggers**
    a. **Navigate back to the main dashboard**
    b. **Create 2 new FreeStyle Jobs** named “TestJob” and “DeployJob”
        i. In the build instructions , TestJob should execute the command *“echo Test”* and DeployJob should execute *“echo Deploy”*
    c. **Navigate to TestJob -> Configure**
    d. **Navigate to Build Triggers** -> Check “Build after other projects are built”
    e. Type in the name of the first job , CompileJob, into the “Projects to watch” field.
    f. Save
    g. Navigate to DeployJob -> configure

h. Check “Build after other projects are built”
i. Type in the name of the second job , “TestJob”
j. Now when the job “CompileJob” runs , it will trigger “TestJob” to run and then trigger DeployJob
k. Verify it by:
    i. **navigating to the Dashboard**
    ii. **Clicking** “CompileJob” then clicking “Build Now”
    iii. All 3 jobs should be shown in the Build Queue and should have an updated “Last Success” time
    iv. Refer to the gif below to see what a successful setup looks like, look at the gif in the following link: https://imgur.com/JCIPNzA

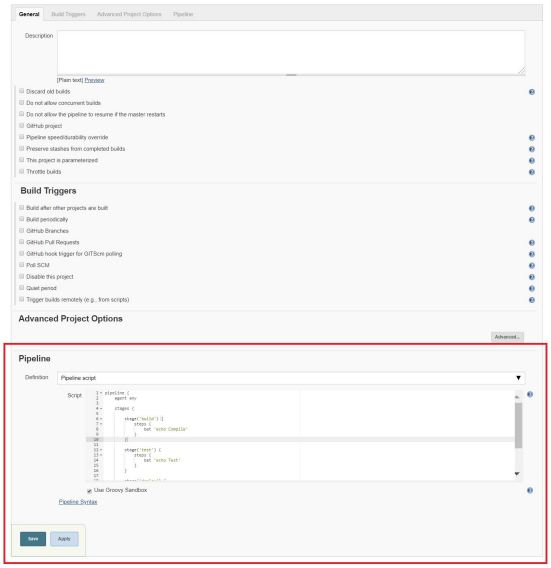


Managing and Creating Jenkins Pipelines

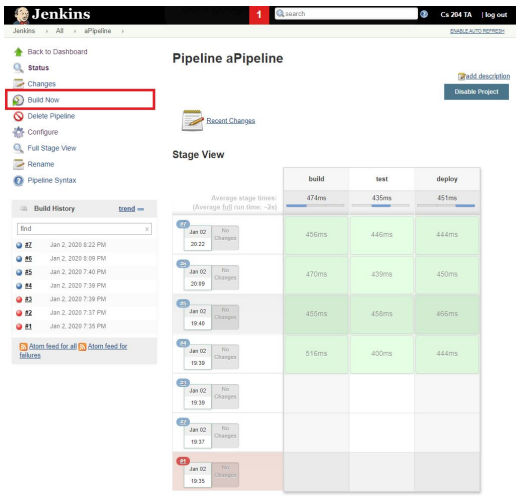
In this section you will learn how to create and manage build triggers and pipelines using a Pipeline Script and a Jenkinsfile. We will re-create the pipeline defined manually in the previous phase.

1. **Building a basic pipeline with a pipeline script**
    a. **Navigate to the main dashboard** -> new item
    b. Give your new item a name “aPipeline” -> **Click pipeline** -> click Ok
    c. Now **scroll to the bottom where it says Pipeline**
    d. We will now write a pipeline script that will do everything we did in the last stage

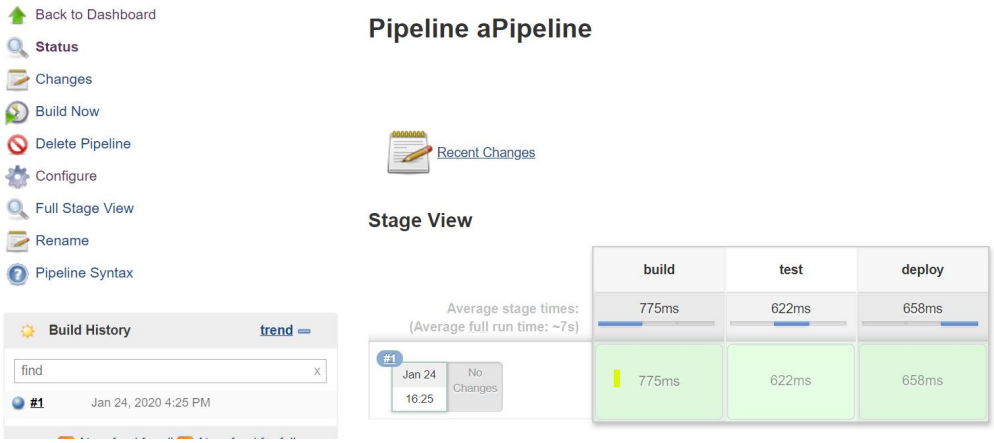
e. The script I used is shown in the screenshot below.



```
pipeline {
    agent any

    stages {
        stage ('build') {
            steps {
                sh 'echo Compile'

            }
        }

        stage ('test') {
            steps {
                sh 'echo Test'
            }
        }

        stage ('deploy') {
            steps {
                sh 'echo Deploy'
            }
        }
    }
}
```



f. After writing the script, hit apply and save
g. In the pipeline job dashboard, **Click "Build Now"**
h. The time it took and the success of the pipeline stages will be shown on a graphic as seen in the screenshot below.



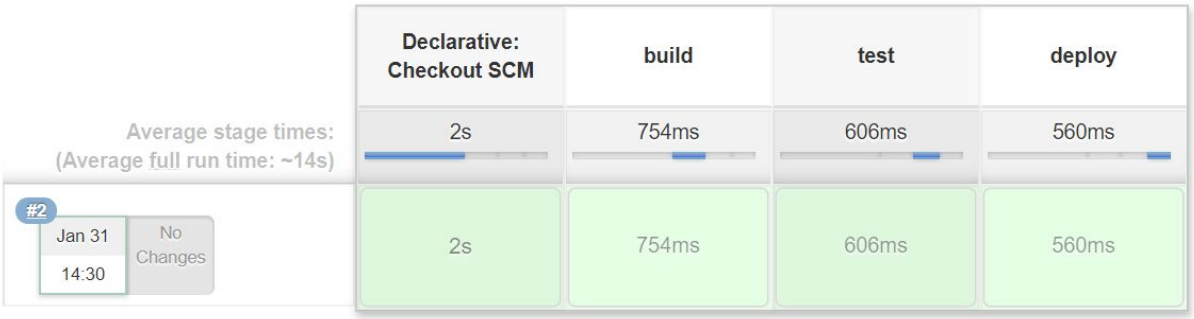
i. To see each output, click on the stage then click on "logs"

2. Jenkins files creation

A Jenkins file contains the pipeline script for a project. Creating this file allows you to define the pipeline in the project itself. Allowing the pipeline to be changed easier.

a. **Create a new public repository** on your Github account
b. **Create a new file named "Jenkinsfile"** in this new repository

c. **Copy and paste your pipeline script** from the previous step into the Jenkinsfile. You can get this by clicking your pipeline -> configure and then scrolling to the section with the script.
d. Push your changes to your GitHub repo.
e. Copy the git URL from the clone button
f. Navigate to you pipeline job -> configure -> Pipeline
g. In the definition field, **change "Pipeline Script" to "Pipeline script from SCM"**
h. Select "Git" for SCM type
i. Paste the **GitHub URL that contains the Jenkinsfile into the Source Repository URL field**
j. Save and Apply
k. Build the job
l. Jenkins now downloads the Github repository, looks for a Jenkinsfile in the home directory, then uses the script within the Jenkinsfile to build a pipeline.
m. The build pipeline should look just like the previous one, but with an extra "Checkout SCM" step



Creating a Delivery Pipeline to Build, Test, and Deploy an application with IntelliJ

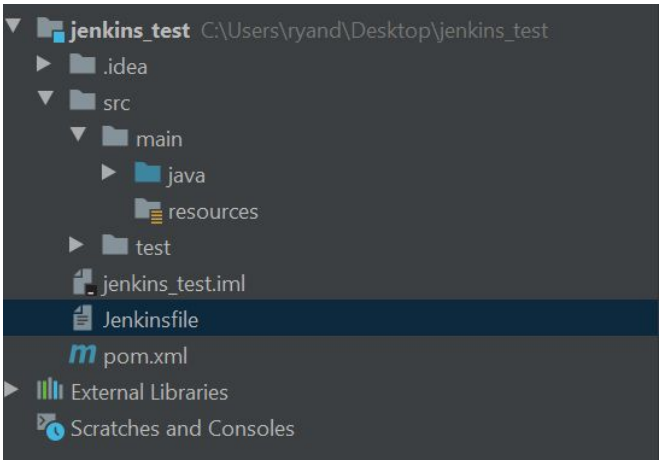
We will now connect an IntellJ project to Jenkins using Github and a Jenkins file

1. **Setup IntelliJ**
    a. Download the project files from this link: https://drive.google.com/file/d/1z3tas-Sk-WBicWUKakd1yt1RYccQ0O9X/view?usp=sharing
    b. Open the project in IntellJ
    c. Build to verify that the project is working
    Note: You will get a build error that requires you to specify a **<maven.compile.source>** and **<maven.compiler.target>** properties according to the instructions in this document: https://docs.google.com/document/d/1eUYOSPOKUCNxM5U56yDZ16FF1Pgss-Q0iisgqdo8tic/edit?usp=sharing. Enter 1.8 as the value for these two properties

to **match the Java version you used in the setup of your Jenkins server** in the pre-class assignment.

d. Connect your IntelliJ project to a **new Github repository**
   i. If you don't know how to do this, follow the steps given by IntellJ https://www.jetbrains.com/help/idea/set-up-a-git-repository.html
e. Commit and Push everything to the Github Repository

2. **Set up Jenkins**
   a. **Create a new pipeline job** on Jenkins
      i. Under the "Build Triggers" set-up option , Enable **Poll SCM**
   b. Set the **Poll SCM schedule** to check every minute with the following code (5 stars without the quotes):
      i. "* * * * *"
      ii. You can press the ? button next to Schedule for more details
   c. Enable **"Pipeline script from SCM"** -> Set SCM to "Git"
   d. Paste the Github URL into the repository URL field
   e. Apply and Save
3. **Input the Jenkinsfile that will define the Jenkins pipeline**
   a. Download the Jenkinsfile from this link: https://drive.google.com/file/d/1q0UOF589EK4krhOWJh3rpXSAdcNTXQm9/view?usp=sharing
   b. Place the Jenkinsfile in the top level folder of your project, your project files should look like the screenshot below:



   c. Open the Jenkinsfile - This Jenkinsfile consists of three parts. The agent, the tools, and the stages
      i. The agent

```
pipeline {
    agent any
    tools {
        maven 'apache maven 3.6.3'
        jdk 'JDK 8'
    }
```

a. The agent defines where Jenkins will execute in the Jenkins environment. We defined the Agent as any allowing the pipeline to execute on any environment Jenkins has access to.

ii. The Tools

```
pipeline {
    agent any
    tools {
        maven 'apache maven 3.6.3'
        jdk 'JDK 8'
    }
```

a. In tools, we tell the Jenkins pipeline which global tools it will have access to.
b. In this Jenkinsfile we are saying that this Jenkins pipeline can use the maven we identified as 'apache maven 3.6.3' and the jdk we identified as 'JDK 8.'

iii. The Stages

These stages represent the different pipeline stages. If any step fails in the process, the whole pipeline will stop. Within each stage, we must have at least one step. These steps contain the commands that Jenkins will run.

1. Clean

```
    stage ('Clean') {
        steps {
            sh 'mvn clean'
        }
    }
```

a. Here, the step will run the maven command clean on the project it downloads from Github. Removing any pre-compiled code

2. Build

```
    stage ('Build') {
        steps {
            sh 'mvn compile'
        }
    }
```

a. Here, the step will compile our project code

3. Short Tests

```
stage ('Short Tests') {
    steps {
        sh 'mvn -Dtest=CalculatorTest test'
    }
}
```

   a. Here, the step will run the test files in the project, but only the file that has the name “CalculatorTest”

4. Long Tests

```
stage ('Long Tests') {
    steps {
        sh 'mvn -Dtest=CalculatorTestThorough test'
    }
    post {
        success {
            junit 'target/surefire-reports/**/*.xml'
        }
    }
}
```

   a. Here, the step will run the test files in the project, but only the file that has the name “CalculatorTestThorough”

5. Post

```
stage ('Long Tests') {
    steps {
        sh 'mvn -Dtest=CalculatorTestThorough test'
    }
    post {
        success {
            junit 'target/surefire-reports/**/*.xml'
        }
    }
}
```

   a. A Post section runs all the commands contained within it after all of the steps in parent stage are completed.
      i. The success section sets a condition that causes the contained commands to only run if the parent stage’s steps finished successfully.
      ii. Junit - is a command that takes the junit reports generated by maven and outputs them to a graph on the pipeline dashboard

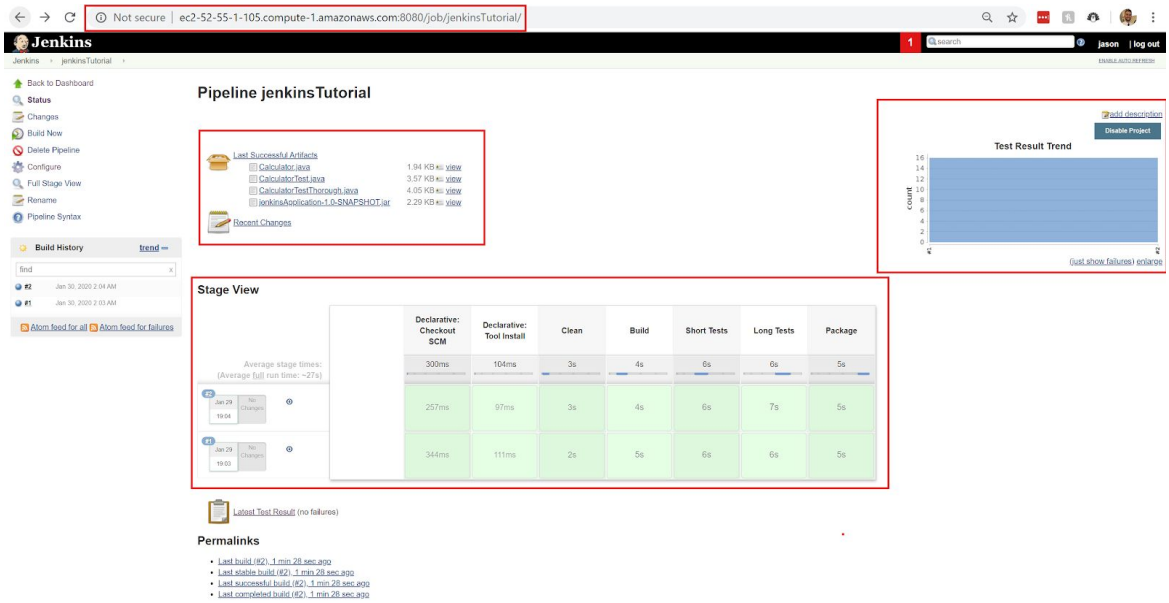
6. Package

```
stage ('Package') {
    steps {
        sh 'mvn package'
        archiveArtifacts artifacts: 'src/**/*.java'
        archiveArtifacts artifacts: 'target/*.jar'
    }
}
```

   a. Here, the step will use maven to package our code into a jar
   b. The archiveArtifacts command will get all the .java and .jar files from the specified locations and store them in Jenkins. This saves the code for each build you run.

iv. Once you've reviewed the Jenkinsfile, make sure you have added your Jenkinsfile to git and then Commit and Push your project files to your Github repository.

4. **Finish Writing the Program**
   Practice using Jenkins by finishing the code for each method. **After finishing each method Commit and Push** your program to see how Jenkins responds to changes in the code repository.
   a. Finish writing these functions: fibonacciNumberFinder, intToBinaryNumber, and createUniqueID.
      Feel free to look up code that fulfils the requirements of these methods.
      This tutorial is to teach Jenkins not how to write these methods.
      i. Make sure to follow the instructions in the comments above each function
      ii. Make sure that they pass the tests from the given Test files
      iii. Do not modify the Test Files
      iv. Once your program passes 100% of the tests, Commit and Push your code one last time. This will cause your Jenkins job to run on your server within one minute.
5. **Take a screenshot of your dashboard once you have all the tests working, it should look like the screenshot below :**
   a. Make sure you run the working pipeline at least twice
   b. Refresh your page to get the graph at the top right



What to Turn in

1. **A screenshot of your working Jenkins pipeline**
    a. Make sure your screenshot shows the following :
        i. **Your AWS Ec2 URL**
        ii. **The Pipeline Stage View**
        iii. **The Archived files**
        iv. **The Test results (with all tests passing)**

After the Tutorial

We will use this same Jenkins AWS Ec2 instance in the next tutorial. We will also build on the code that you built for this tutorial.

1. **DO NOT TERMINATE YOUR JENKINS AWS INSTANCE**
2. **REMEMBER YOUR JENKINS LOGIN CREDENTIALS**
3. **Do not delete your GitHub repository with your working Jenkinsfile and Code**
4. Stop your Jenkins AWS EC2 instance
    a. We will use this instance in the next tutorial