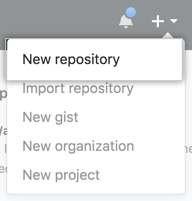
| **Expt.No:** 1 | **CREATING A NEW GIT REPOSITORY, CLONING EXISTING REPOSITORY, CHECKING CHANGES INTO A** |
| --- | --- |
| **Date:** |
|  | **GIT REPOSITORY, PUSHING CHANGES TO A GIT** |
|  | **REMOTE, CREATING A GIT BRANCH** |

# AIM:

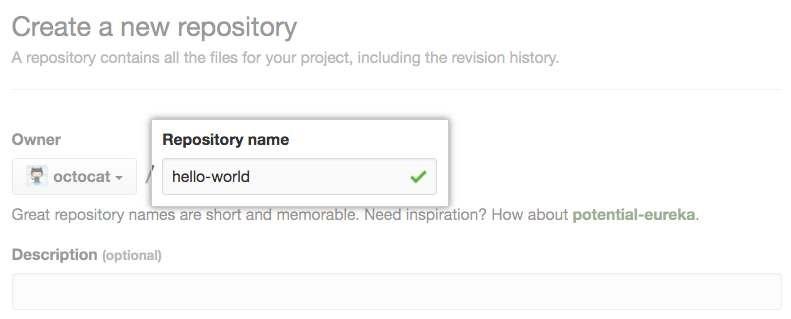
To creating a new Git repository, cloning existing repository, checkin changes into a Git repository, pushing changes to a Git remote, creating a Git branch.

# CREATING A NEW GIT REPOSITORY:

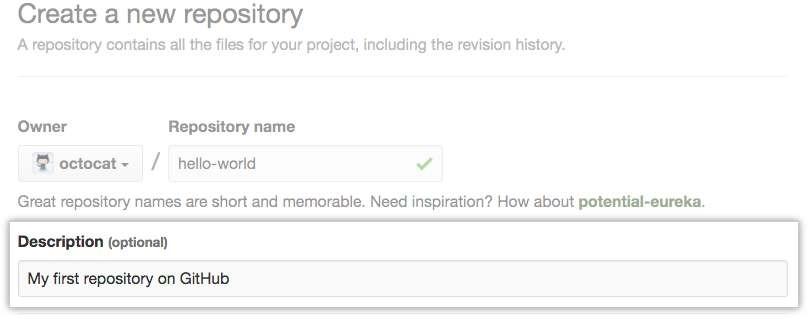
1. On GitHub.com, in the upper-right corner of any page, use the drop-down menu, and select New repository.



1. Type a short, memorable name for your repository. For example, "hello-world”.

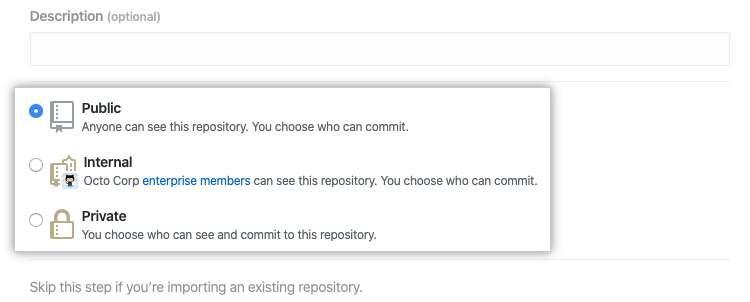


1. Optionally, add a description of your repository. For example, "My first repository on GitHub."

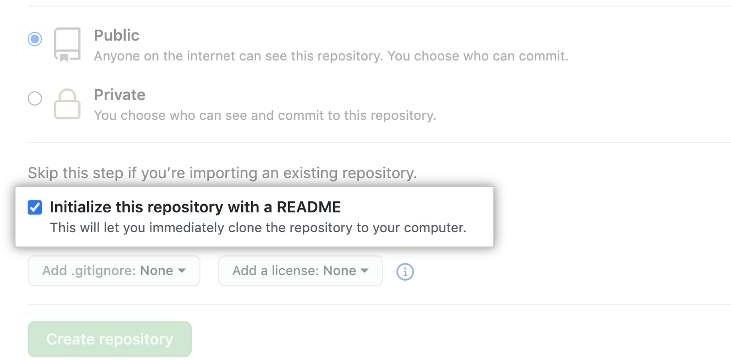


1

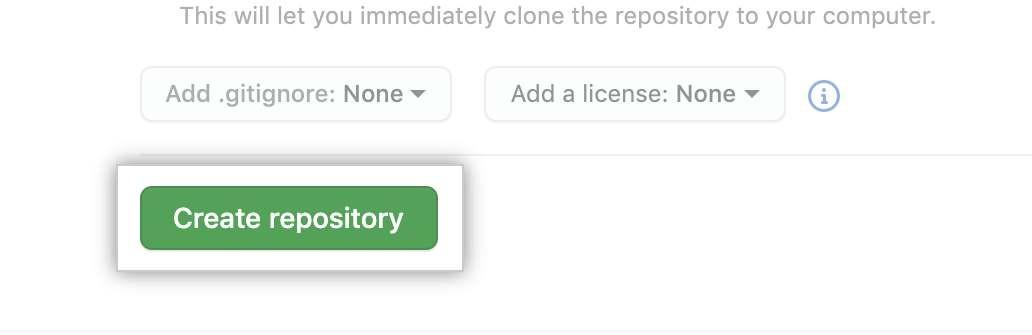
1. Choose a repository visibility.



1. Select Initialize this repository with a README.



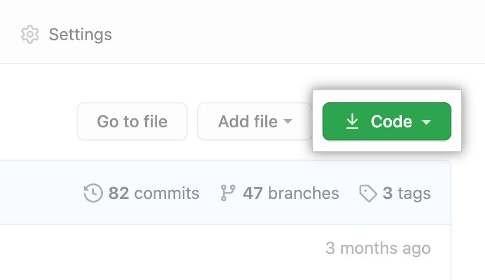
1. Click **Create repository**.



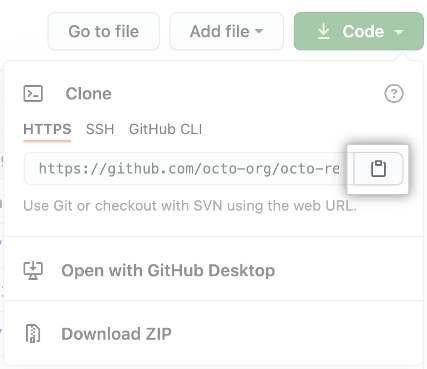
2

# CLONING EXISTING REPOSITORY

* 1. On GitHub.com, navigate to the main page of the repository.
  2. Above the list of files, click **Code**.



* 1. Copy the URL for the repository.
     + To clone the repository using HTTPS, under "HTTPS", click .
     + To clone the repository using an SSH key, including a certificate issued by your organization's SSH certificate authority, click **SSH**, then click .
     + To clone a repository using GitHub CLI, click **GitHub CLI**, then click .



* 1. Open Git Bash.
  2. Change the current working directory to the location where you want the cloned directory.
  3. Type git clone, and then paste the URL you copied earlier.

$ git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY

* 1. Press **Enter** to create your local clone.



3

# CHECKIN CHANGES INTO A GIT REPOSITORY

* 1. Fetching changes from a remote repository



* 1. Merging changes into your local branch



* 1. Pulling changes from a remote repository



# PUSHING CHANGES TO A GIT REMOTE

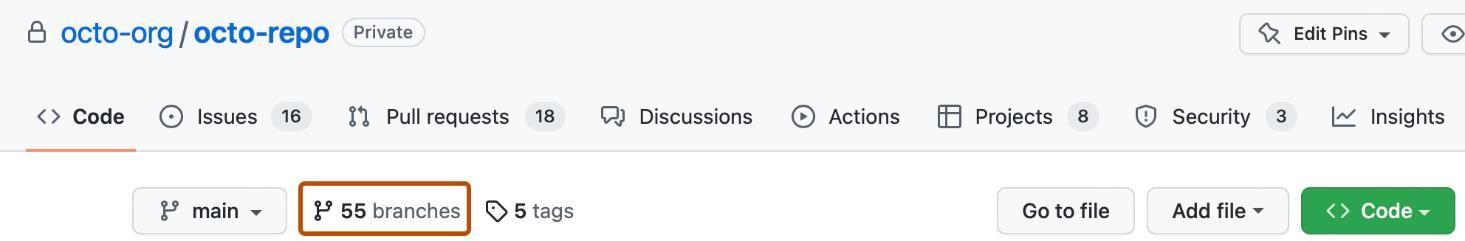
The git push command takes two arguments:

* A remote name, for example, origin
* A branch name, for example, main

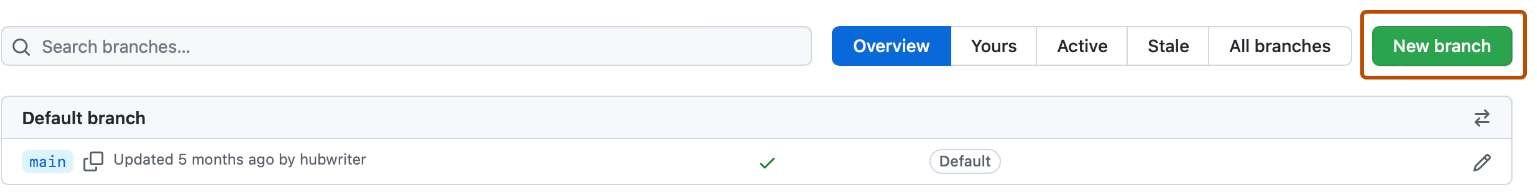


# CREATING A GIT BRANCH

* 1. On GitHub.com, navigate to the main page of the repository.
  2. Above the list of files, click **Branches**.



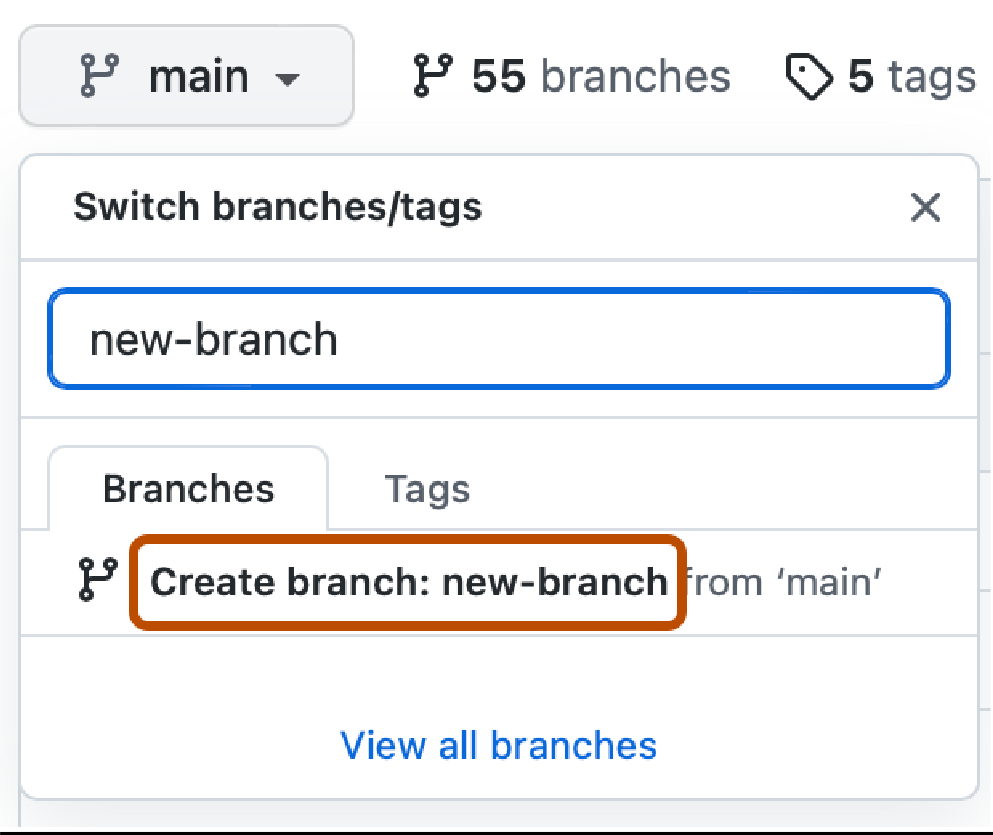
* 1. Click **New branch**.



* 1. Under "Branch name", type a name for the branch.
  2. Under "Branch source", choose a source for your branch.
     + If your repository is a fork, select the repository dropdown menu and click your fork or the upstream repository.
     + Select the branch dropdown menu and click a branch.

4

* 1. Click **Create branch**.



# CONCLUSION:

Successfully executed the above operations on local and remote Git Repositories and outputs were captured.

5

| **Expt.No:** 2 | **INSTALLING DOCKER CONTAINER ON WINDOWS/LINUX, ISSUING DOCKER COMMANDS** |
| --- | --- |
| **Date:** |

# AIM:

To install Docker container on Windows/Linux and issuing docker commands.

# INSTALLING DOCKER CONTAINER ON LINUX:

**Step 1** − Before installing Docker, you first have to ensure that you have the right Linux kernel version running. Docker is only designed to run on Linux kernel version 3.8 and higher. We can do this by running the following command.

**uname**

This method returns the system information about the Linux system.

**Syntax**

uname -a

**Options**

**a** − This is used to ensure that the system information is returned.

**Example**

uname –a

**Output**

When we run above command, we will get the following result −



From the output, we can see that the Linux kernel version is 4.2.0-27 which is higher than version 3.8, so we are good to go.

**Step 2** − Need to update the OS with the latest packages, which can be done via the following command −

apt-get This method installs packages from the Internet on to the Linux system.

**Syntax**

sudo apt-get update **Return Value** None

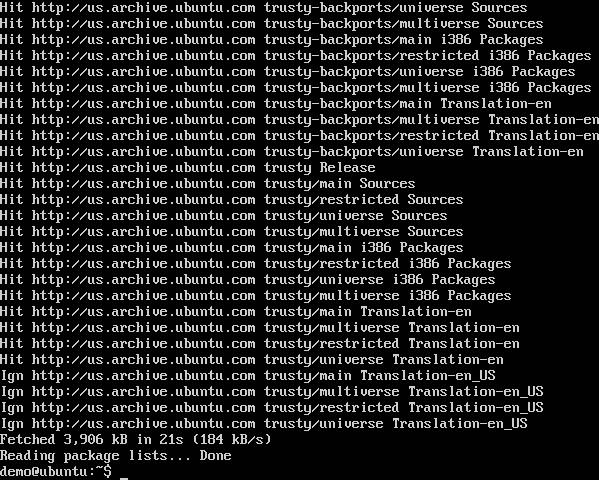
6

**Example**

sudo apt-get update

**Output**

When we run the above command, we will get the following result −



This command will connect to the internet and download the latest system packages for Ubuntu.

**Step 3** − The next step is to install the necessary certificates that will be required to work with the Docker site

later on to download the necessary Docker packages. It can be done with the following command.

sudo apt-get install apt-transport-https ca-certificates

**Step 4** − The next step is to add the new GPG key. This key is required to ensure that all data is encrypted when

downloading the necessary packages for Docker.



7

**Step 5** − Next, depending on the version of Ubuntu you have, you will need to add the relevant site to the **docker.list** for the **apt package manager**, so that it will be able to detect the Docker packages from the Docker site and download them accordingly.

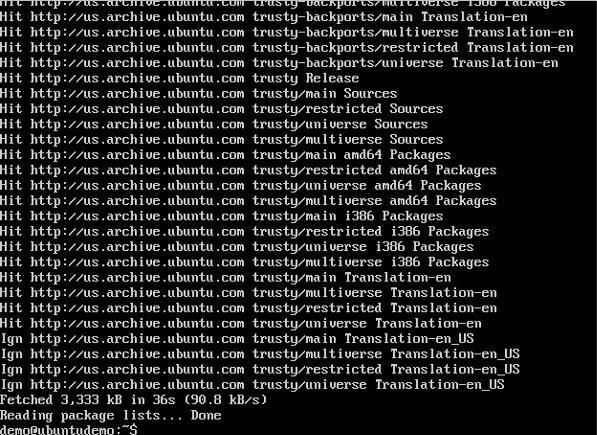
Since our OS is Ubuntu 14.04, we will use the Repository name as

“deb [https://apt.dockerproject.org/repo](https://apt.dockerproject.org/repo/)ubuntu-trusty main”.

And then, we will need to add this repository to the **docker.list** as mentioned above.

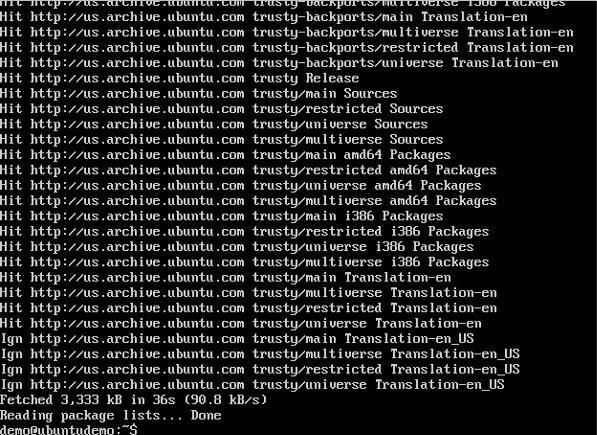


**Step 6** − Next, we issue the **apt-get update command** to update the packages on the Ubuntu system.

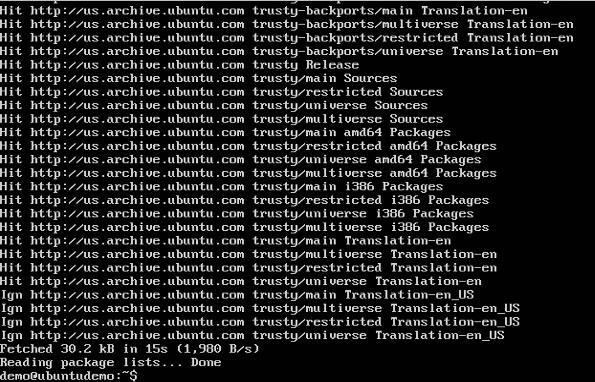


8

**Step 7** − If you want to verify that the package manager is pointing to the right repository, you can do it by issuing the **apt-cache command**.

apt-cache policy docker-engine In the output, you will get the link to <https://apt.dockerproject.org/repo/>

**Step 8** − Issue the **apt-get update command** to ensure all the packages on the local system are up to date.



9

**Step 9** − For Ubuntu Trusty, Wily, and Xenial, we have to install the linux-image-extra-\* kernel packages, which allows one to use the **aufs storage driver**. This driver is used by the newer versions of Docker.

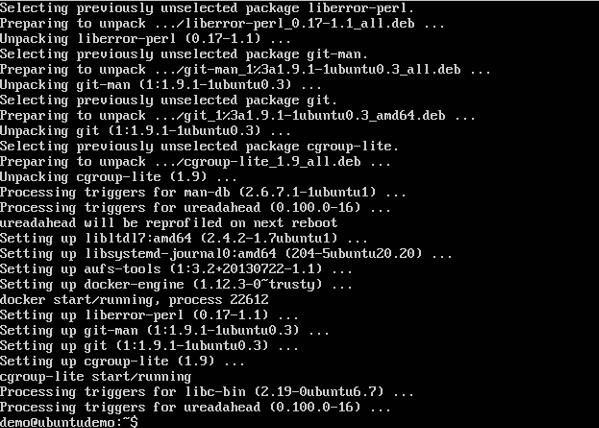


**Step 10** − The final step is to install Docker and we can do this with the following command −

sudo apt-get install –y docker-engine

Here, **apt-get** uses the install option to download the Docker-engine image from the Docker website and get Docker installed.

The Docker-engine is the official package from the Docker Corporation for Ubuntu-based systems.

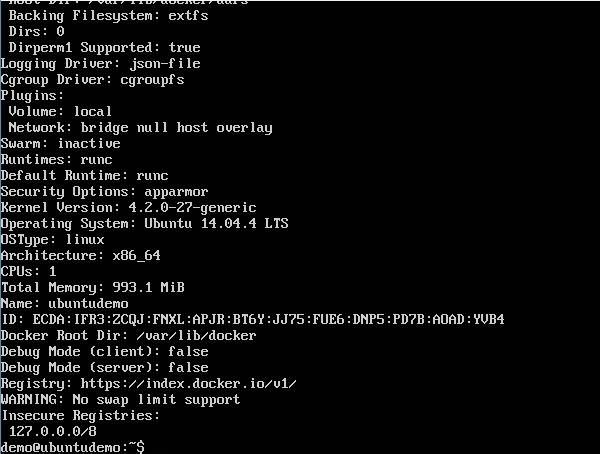


10

| 1. **ISSUING DOCKER COMMANDS**    1. **Docker Version**   To see the version of Docker running, you can issue the following command −  **Syntax**  docker version  **Return Value**  The output will provide the various details of the Docker version installed on the system.  **Example**  sudo docker version  **Output**  When we run the above program, we will get the following result −     * 1. **Docker Info**   To see more information on the Docker running on the system, you can issue the following command −  **Syntax**  docker info  **Example** | |
| --- | --- |
|  | sudo docker info |
| 11 | |

**Output**

When we run the above command, we will get the following result −



# DOCKER FOR WINDOWS

Docker has out-of-the-box support for Windows, but you need to have the following configuration in order to install Docker for Windows.

**System Requirements**

| Windows OS | Windows 10 64 bit |
| --- | --- |
| Memory | 2 GB RAM (recommended) |

Following command lines can be used to install docker desktop on windows 10, 11 or higher versions

* + 1. To run in terminal,

"Docker Desktop Installer.exe" install

12

* + 1. For Powershell,

Start-Process '.\win\build\Docker Desktop Installer.exe' -Wait install

* + 1. For windows command prompt,

start /w "" "Docker Desktop Installer.exe" install

* + 1. To add user accounts into docker user groups especially when your admin account and user account are not the same.

net localgroup docker-users <users>/add



# CONCLUSION:

Successfully installed the docker containers in Windows and Linux and verified docker commands.

13

| **Expt.No:** 3 | **BUILDING DOCKER IMAGES FOR PYTHON APPLICATION** |
| --- | --- |
| **Date:** |

# AIM:

To build docker images for Python application.

# PROCEDURE:

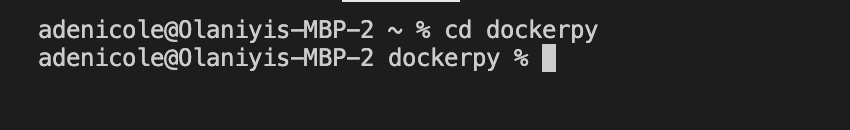
1. **Setting Up the Dockerfile**

First, going to set up the [Dockerfile,](https://docs.docker.com/engine/reference/builder/) which is a sequential set of commands used in building the Docker image. For this, you’ll use python unbuffered, a Python environmental variable that allows the Python output to be sent straight to the terminal when set to a non-empty string or executed with the -u option on the command line.

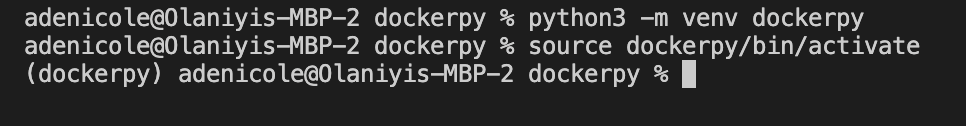
This is useful when log messages are needed in real time. It also prevents issues such as the

application crashing without giving relevant details due to the message being “stuck” in a buffer.

Create a project directory and change into the directory using cd <directory\_name>:



Run the commands below to create a virtual environment. This isolates the environment for the Python project, so that it won’t affect or be affected by other Python projects running on the local environment. Any dependencies installed won’t interfere with other Python projects.

>\_python3 -m venv <directory\_name> source <directory\_name>/bin/activate

Using the following code, create a new file called Dockerfile in the empty project directory:

>\_FROM python:3.8-slim-buster

14

ENV PYTHONUNBUFFERED=1

WORKDIR /app

COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0", "-- port=5000"]

This code pulls the base image from python:3.8-slim-buster and ensures the output is sent straight to the terminal. It confirms the current working directory location, the Python app to be copied to the current directory, and the packages in requirements.txt to be installed.

# Creating the Python App

Create an app.py file and copy the below code:

>\_ from flask import Flask

app = Flask( name )

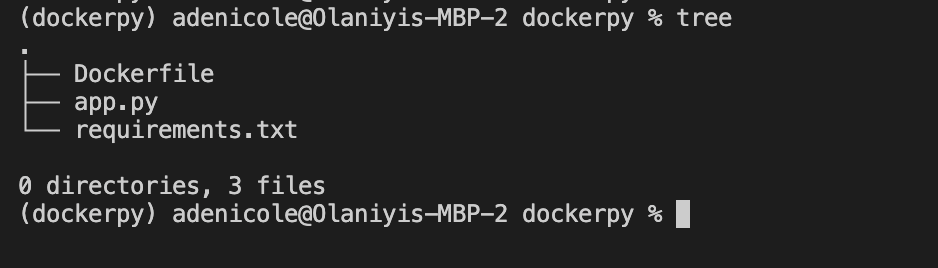
@app.route('/')

def hello\_world():

return 'Hello, Docker!'

Save and close. This creates a simple Python web app that shows Hello, Docker! text.

Create the requirements.txt file. This should contain the dependencies needed for the app to run. The working directory should now look like this:



15

Run the following command in the terminal to install the Flask framework needed to run the Python app and add them to the requirements.txt file. pip3 freeze shows all packages installed via pip.

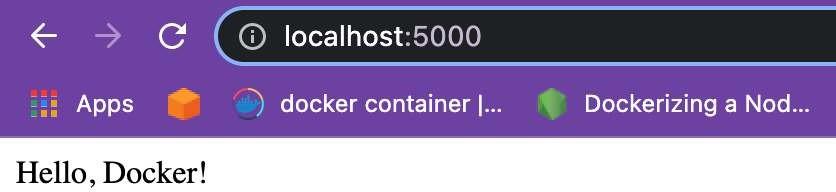
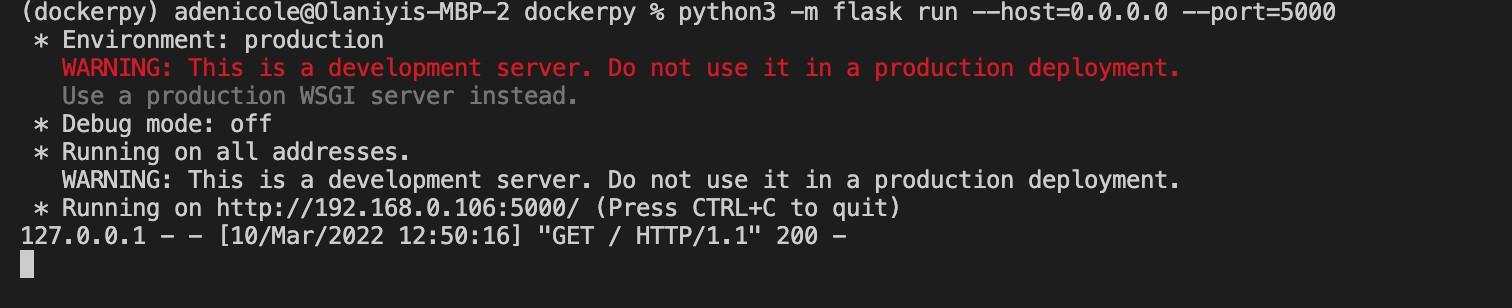
>\_pip3 install Flask

pip3 freeze | grep Flask >> requirements.txt

The requirements.txt file should no longer be empty:

requirements.txtFlask=2.0.3

pylint

Test the app to see if it works by using python3 -m flask run --host=0.0.0.0 --port=5000 and then navigating to http://localhost:5000 in your preferred browser:

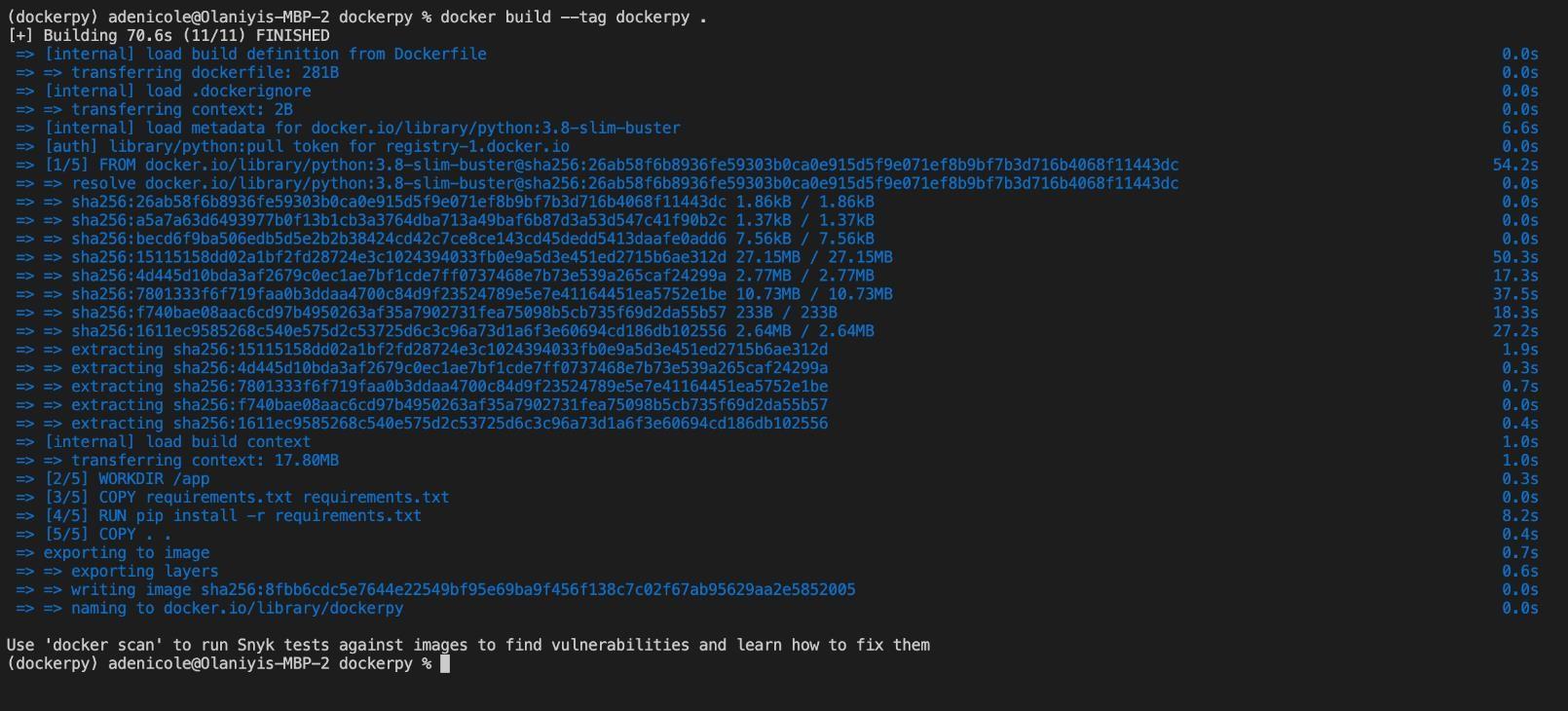
# Building the Docker Image and Container

Now that the Dockerfile, requirements.txt, and app.py have been created, you should test the Python app on your local environment to make sure it works.

Going to build the Docker image from the created Dockerfile. This image is a set of read-only commands used in the building and deployment of Docker containers.

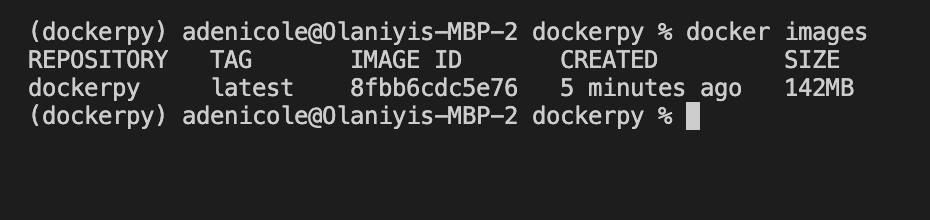
To build the Docker image, use the docker build --tag dockerpy . command. It is common practice to use tags; Docker will give the image a default latest tag.

16



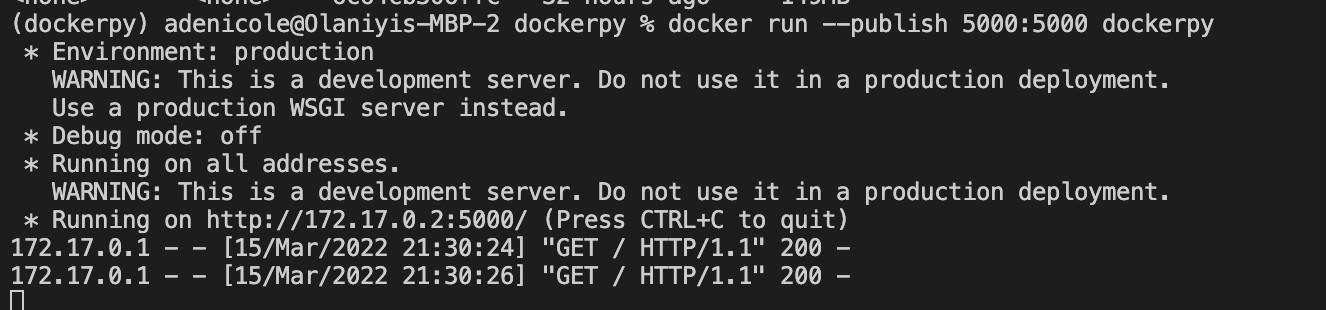
# Docker build

Type docker images into the terminal to view the newly created image:



Tag the image using docker tag <imageId> <hostname>/<imagename>:<tag>:

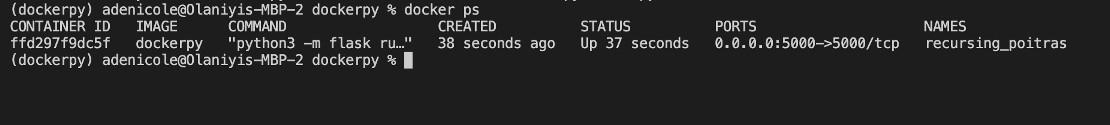
>\_$ docker tag 8fbb6cdc5e76 adenicole/dockerpy:latest



17

# Building the container

Then, use docker ps to see the list of containers present:



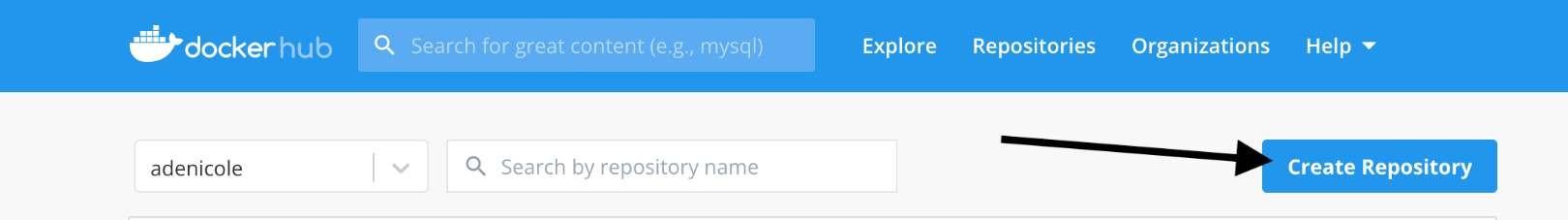
Viewing the list

Now test application using http://localhost:5000 on your preferred browser. You’ve run your Python

app inside a Docker container.

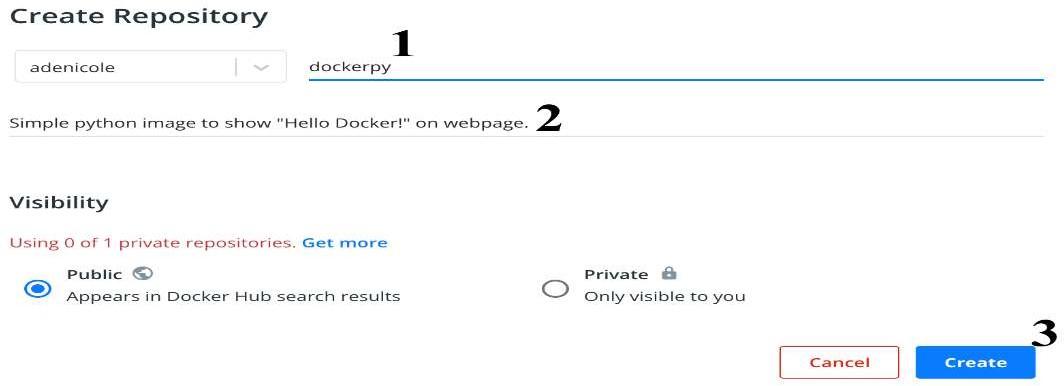
# Running Docker Push

The container image can be pushed and retrieved from the [Docker Hub](https://hub.docker.com/) registry. Docker Hub is an open source library and community for container images. Pushed images can be shared among teams, customers, and communities. It uses a single command, docker push <hub-user>/<repo- name>:<tag>.

To get a hub username, sign up [on the website](https://hub.docker.com/). Then, click **Create Repository** at the top right corner of the page:

Creating repository

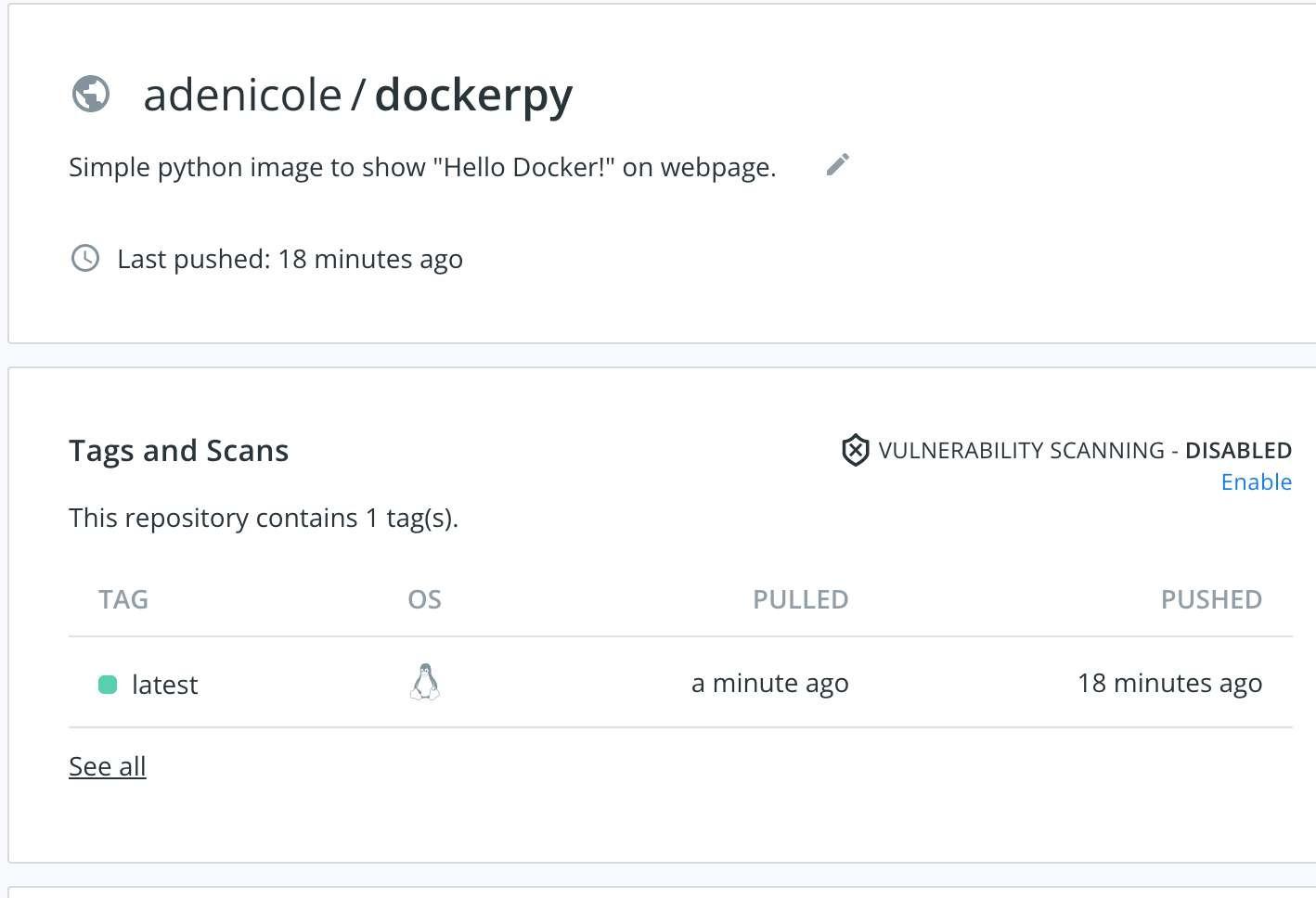
Give the repo a name and description, then click **Create**:



Creating a repo

18

# Docker push for the created Python application

Copy the command on the right side of the page to your terminal, replacing tagname with a version or with the word latest.

# CONCLUSION:

Successfully built the docker image for Python application and outputs were captured.

19

| **Expt.No:** 4 | **SETTING UP DOCKER AND MAVEN IN JENKINS AND FIRST PIPELINE RUN** |
| --- | --- |
| **Date:** |

# AIM:

To setting up Docker and Maven in Jenkins and first pipeline run.

# PREREQUISITES:

* A macOS, Linux or Windows machine with:
* 256 MB of RAM, although more than 2 GB is recommended.
* 10 GB of drive space for Jenkins and your Docker images and containers.

# PROCEDURE:

* 1. Open up a terminal window.
  2. Create a [bridge network](https://docs.docker.com/network/bridge/) in Docker using the following [docker network create](https://docs.docker.com/engine/reference/commandline/network_create/) command:
  3. docker network create jenkins
  4. In order to execute Docker commands inside Jenkins nodes, download and run the docker:dind Docker image using the following [docker run](https://docs.docker.com/engine/reference/run/) command:





20



* 1. Customise official Jenkins Docker image, by executing below two steps:



* 1. Build a new docker image from this Dockerfile and assign the image a meaningful name,



* 1. Keep in mind that the process described above will automatically download the official Jenkins

Docker image if this hasn’t been done before.

* 1. Run your own myjenkins-blueocean:2.387.1-1 image as a container in Docker using the following [docker run](https://docs.docker.com/engine/reference/run/) command:



21



* 1. Proceed to the [Post-installation setup wizard.](https://www.jenkins.io/doc/tutorials/build-a-java-app-with-maven/#setup-wizard)

The Jenkins project provides a Linux container image, not a Windows container image. Be sure that your Docker for Windows installation is configured to run Linux Containers rather than Windows Containers. See the Docker documentation for instructions to [switch to Linux containers.](https://docs.docker.com/docker-for-windows/#switch-between-windows-and-linux-containers) Once configured to run Linux Containers, the steps are:

* 1. Open up a command prompt window and similar to the [macOS and Linux](https://www.jenkins.io/doc/tutorials/build-a-java-app-with-maven/#on-macos-and-linux) instructions above do the following:
  2. Create a bridge network in Docker
  3. docker network create jenkins
  4. Run a docker:dind Docker image

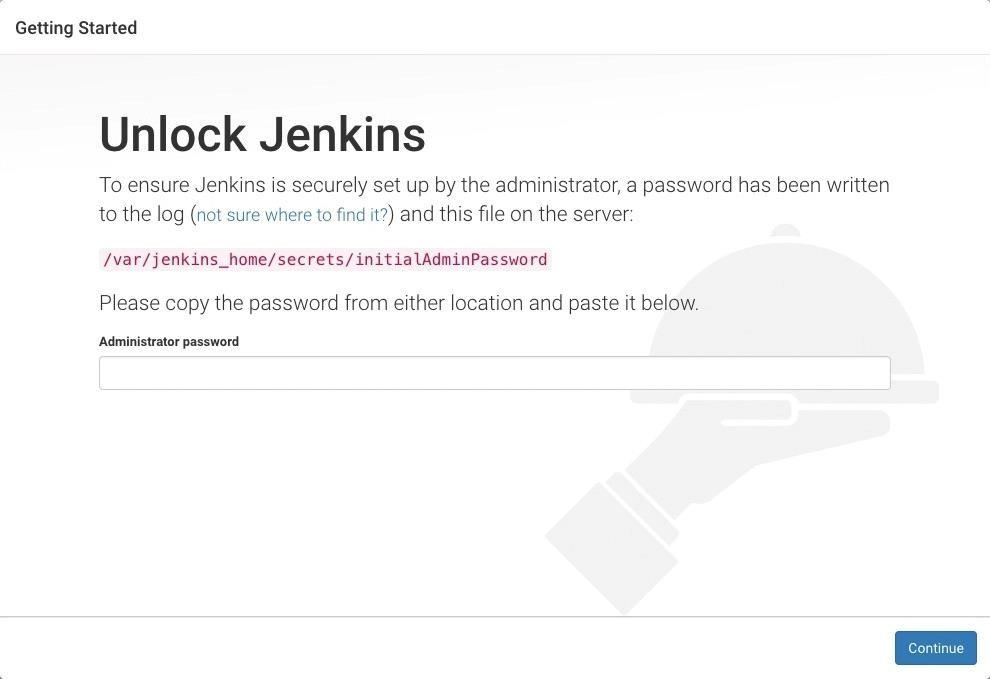


* 1. Customise official Jenkins Docker image, by executing below two steps:
  2. Create Dockerfile with the following content:

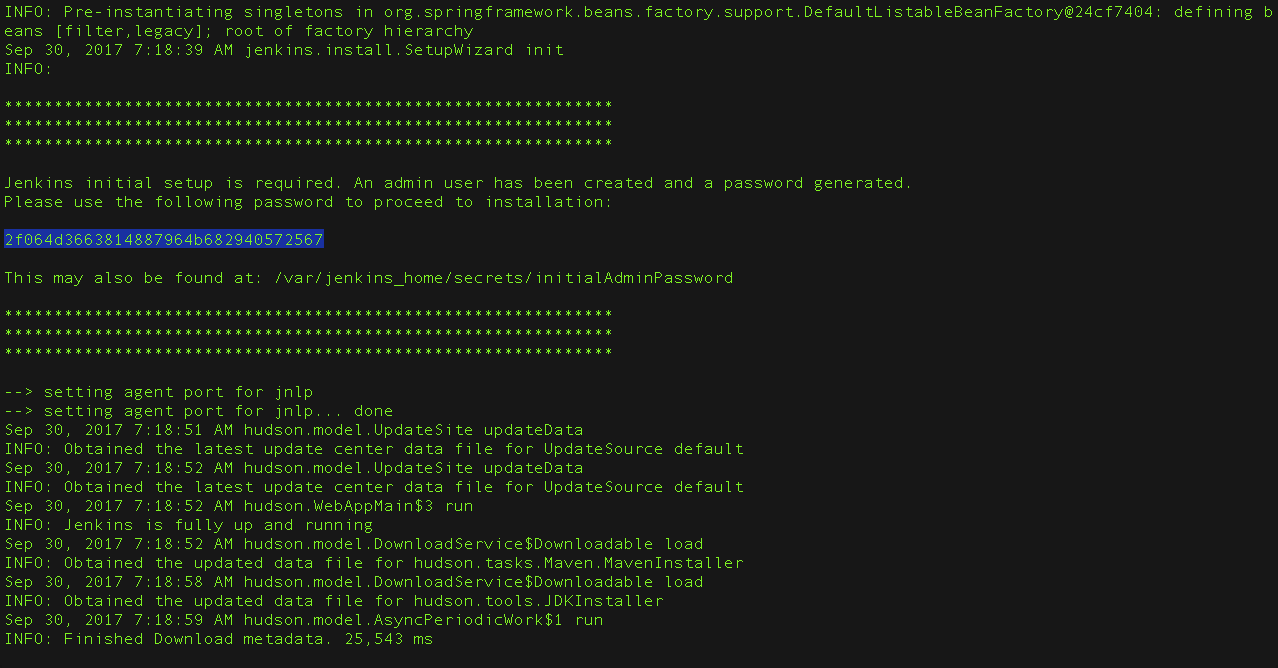


22

* 1. Proceed to the [Setup wizard.](https://www.jenkins.io/doc/tutorials/build-a-java-app-with-maven/#setup-wizard)



* 1. Display the Jenkins console log with the command:

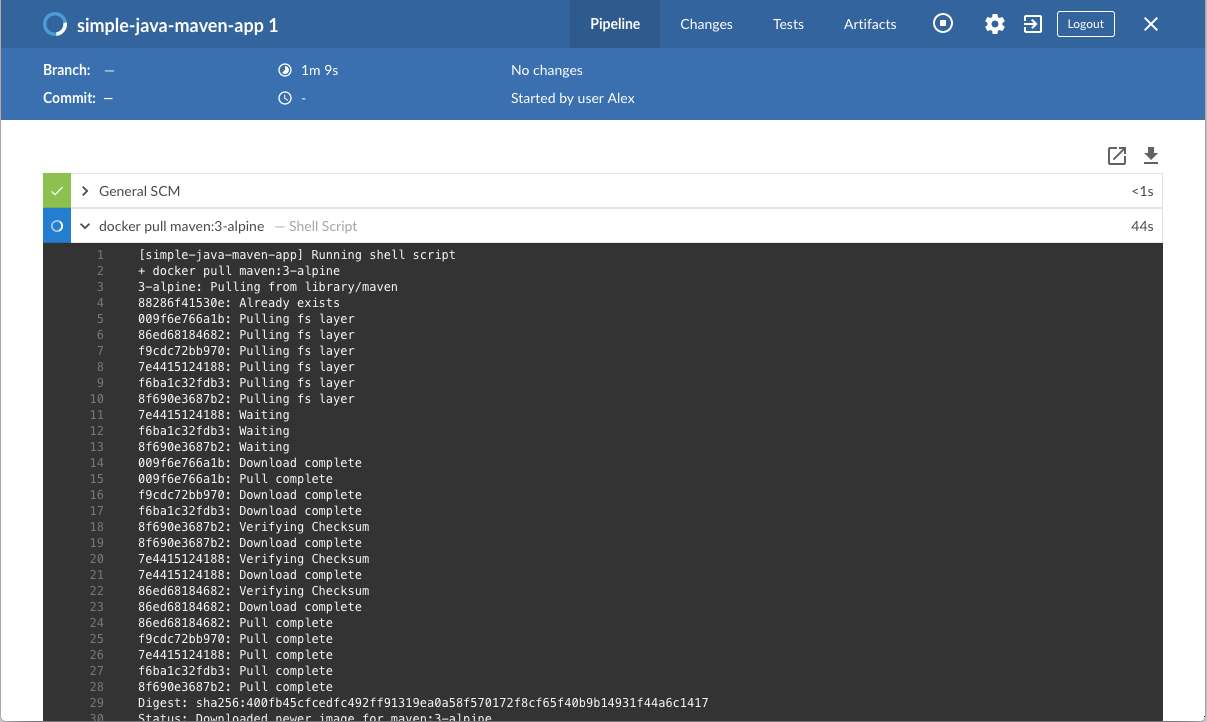


23

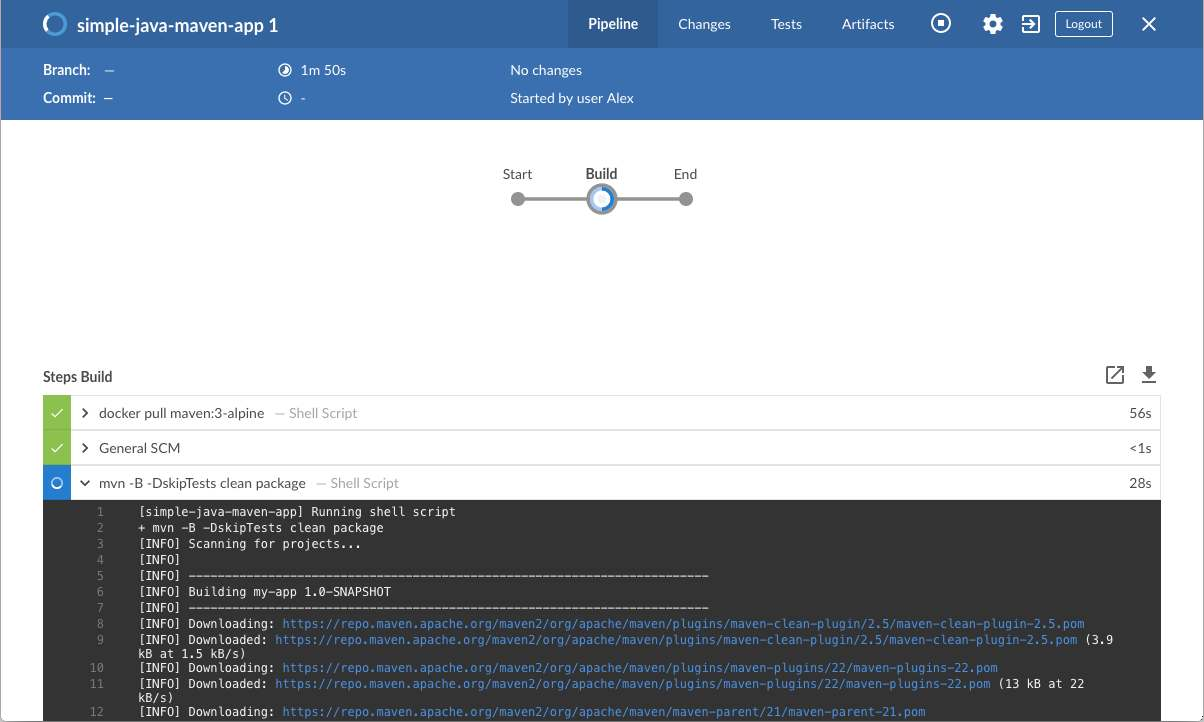
* 1. Copy the following Declarative Pipeline code and paste it into your empty Jenkinsfile:

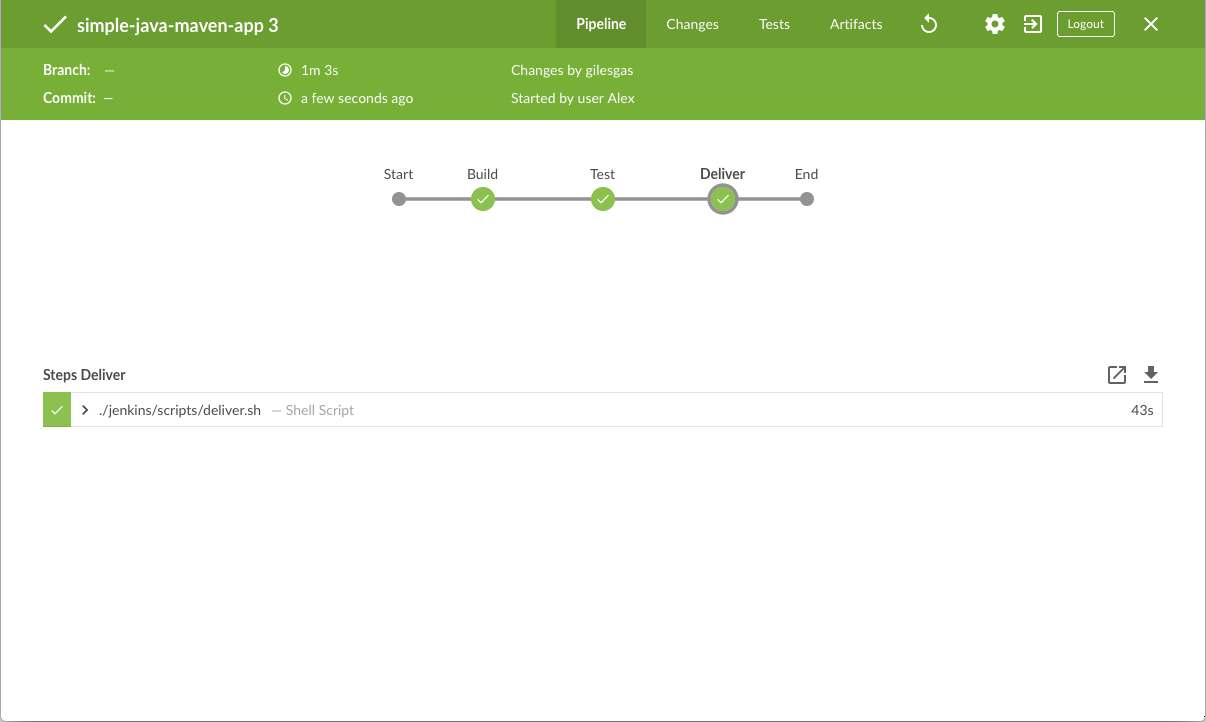


* 1. Downloads the Maven Docker image and runs it in a container on Docker.



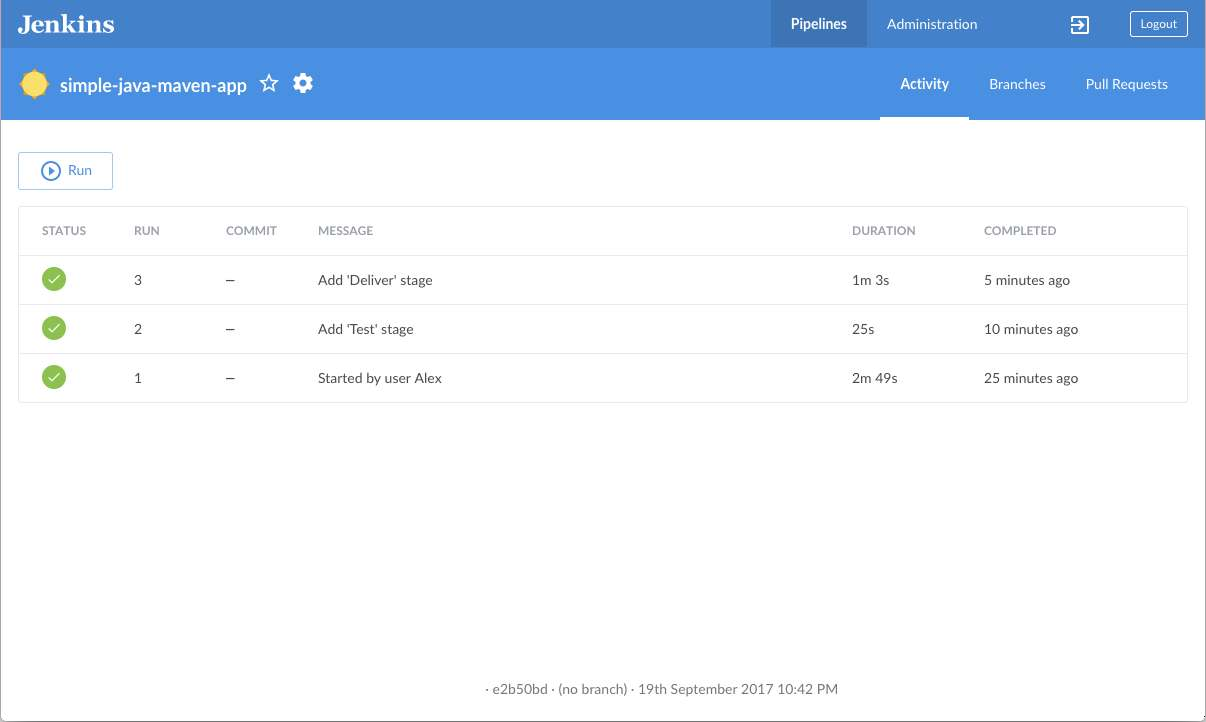
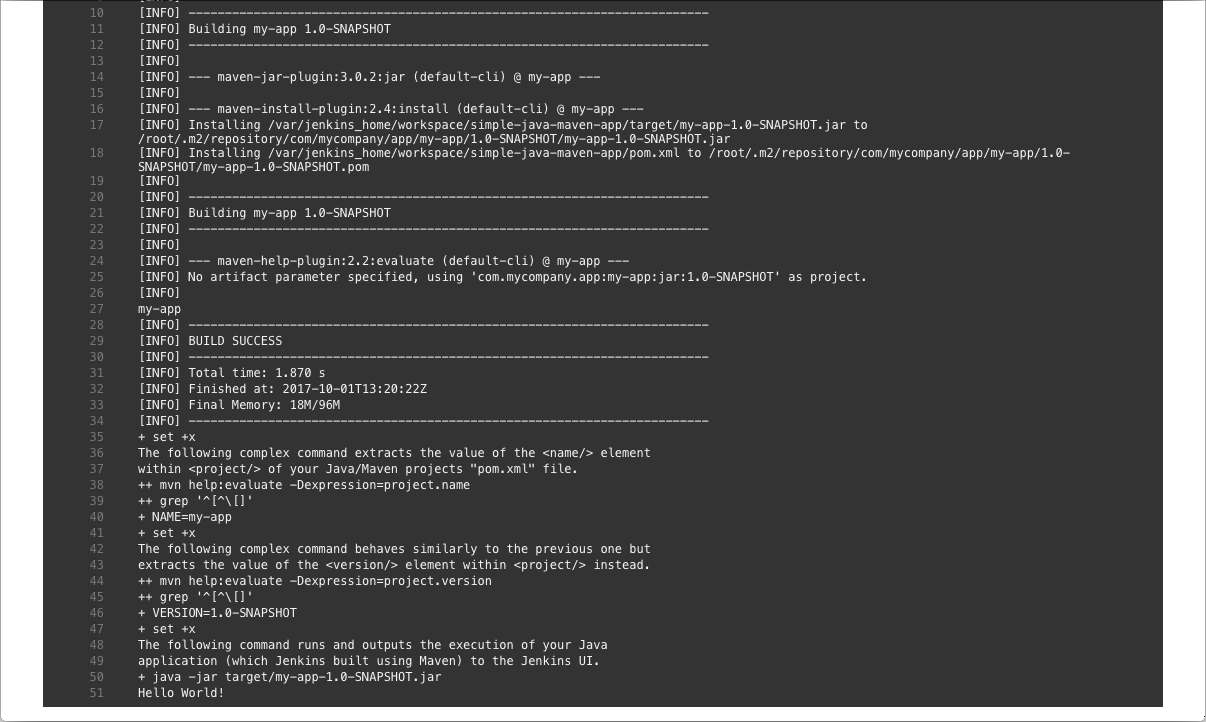
24

* 1. Runs the Build stage (defined in the Jenkinsfile) on the Maven container. During this time, Maven downloads many artifacts necessary to build your Java application, which will ultimately be stored in Jenkins’s local Maven repository (in the Docker host’s filesystem).
  2. Verify successful loading of application.



25

Output of the "Deliver" stage,



# CONCLUSION:

Successfully created Docker and Maven in Jenkins and outputs were captured.

26

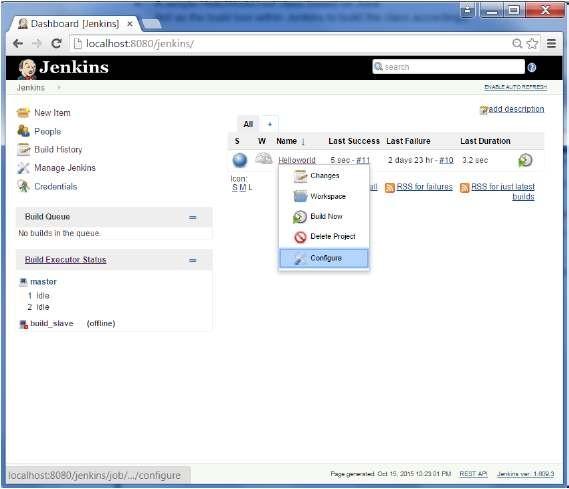
| **Expt.No:** 5 | **RUNNING UNIT TESTS AND INTEGRATION TESTS IN JENKINS PIPELINES** |
| --- | --- |
| **Date:** |

# AIM:

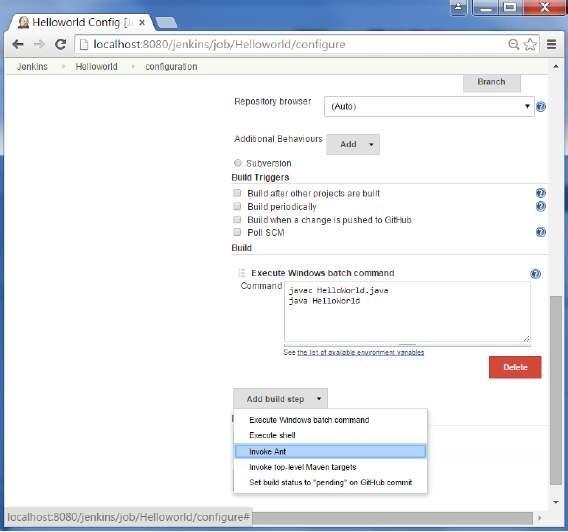
To run Unit tests and Integration tests in Jenkins pipelines.

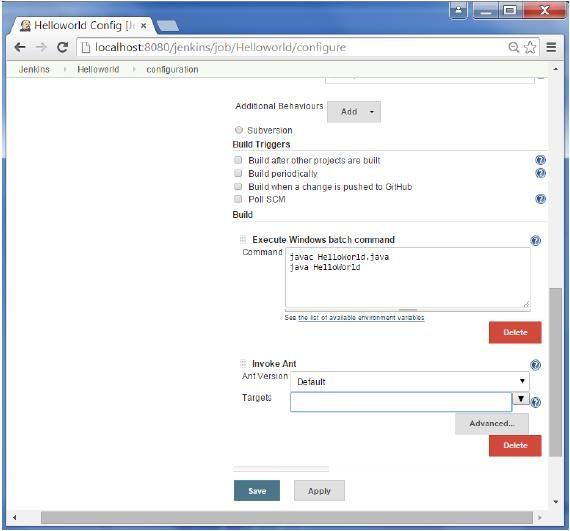
# RUNNING UNIT TESTS IN JENKINS PIPELINES:

* 1. Go to the Jenkins dashboard and Click on the existing HelloWorld project and choose the Configure option



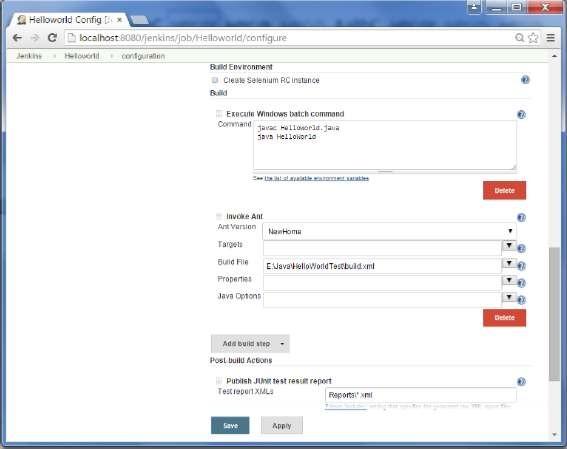
27

* 1. Browse to the section to Add a Build step and choose the option to Invoke Ant.
  2. Click on the Advanced button.

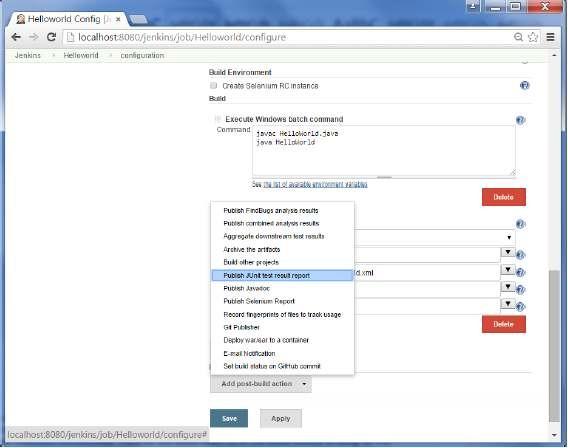


28

* 1. In the build file section, enter the location of the build.xml file.

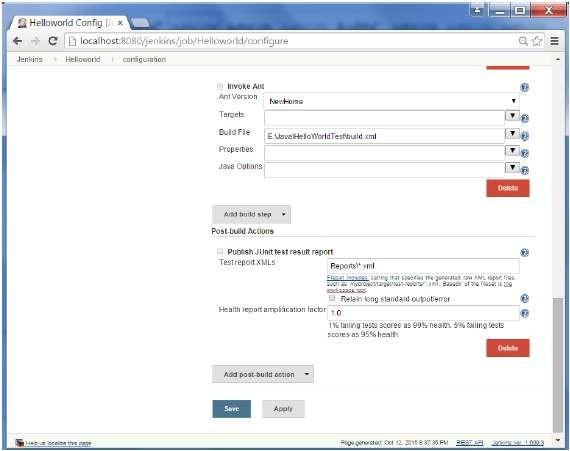


* 1. Next click the option to Add post-build option and choose the option of “Publish Junit test result report”



29

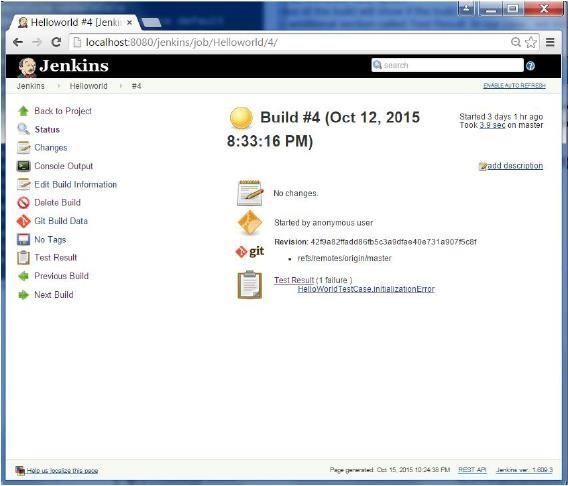
* 1. In the Test reports XML’s, enter the location as shown below. Ensure that Reports is a folder which is created in the HelloWorld project workspace. The “\*.xml” basically tells Jenkins to pick up the result xml files which are produced by the running of the Junit test cases. These xml files which then be converted into reports which can be viewed later.



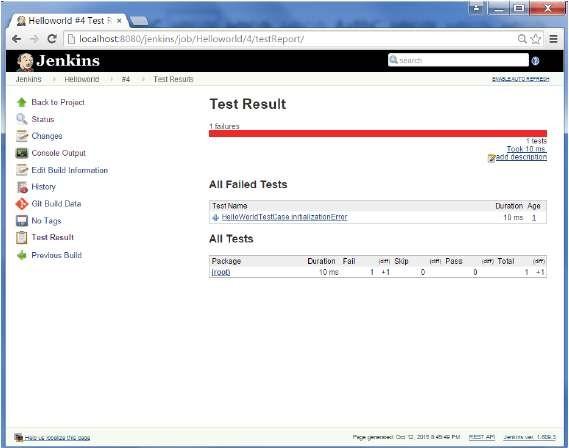
30

* 1. Once saved, you can click on the Build Now option.

Once the build is completed, a status of the build will show if the build was successful or not. In the Build output information, you will now notice an additional section called Test Result. In our case, we entered a negative Test case so that the result would fail just as an example.

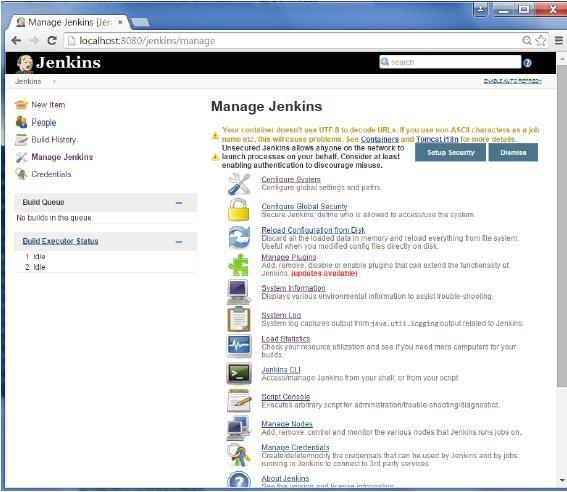


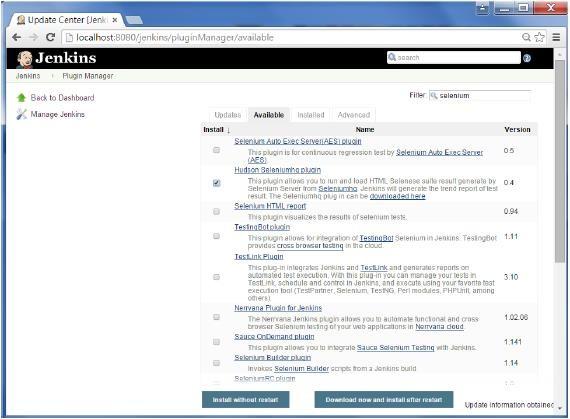
One can go to the Console output to see further information. But what’s more interesting is that if you click on Test Result, you will now see a drill down of the Test results.



31

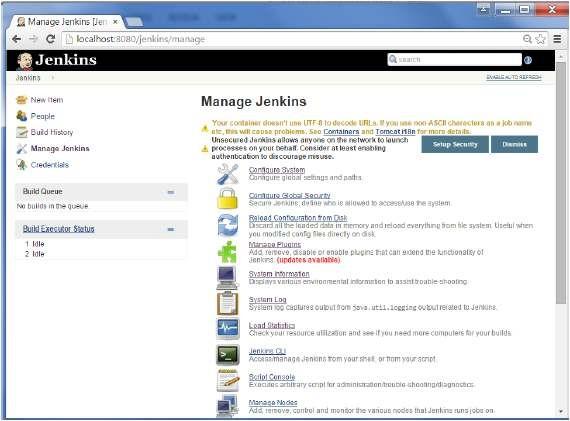
# RUNNING INTEGRATION TESTS IN JENKINS PIPELINES:

* 1. Go to Manage Plugins.
  2. Find the Hudson Selenium Plugin and choose to install. Restart the Jenkins instance.

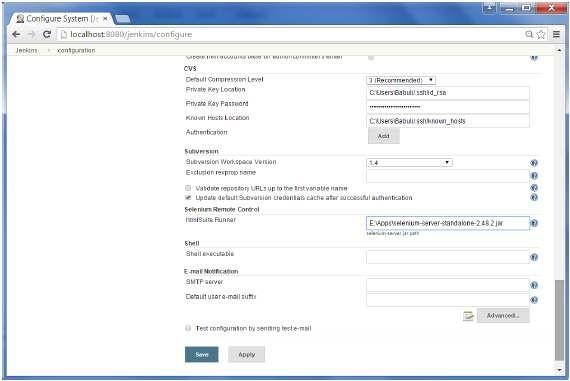


32

* 1. Go to Configure system.

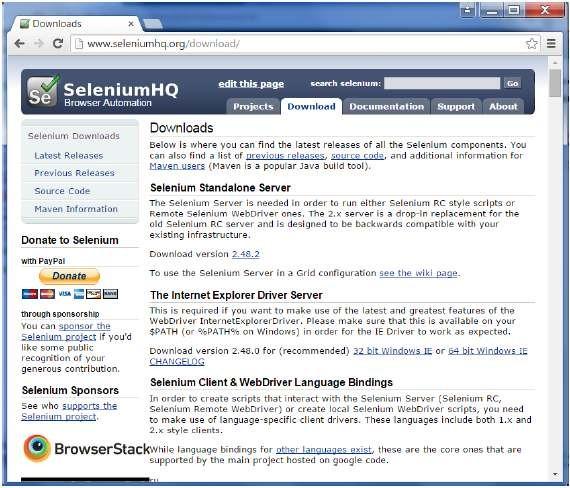


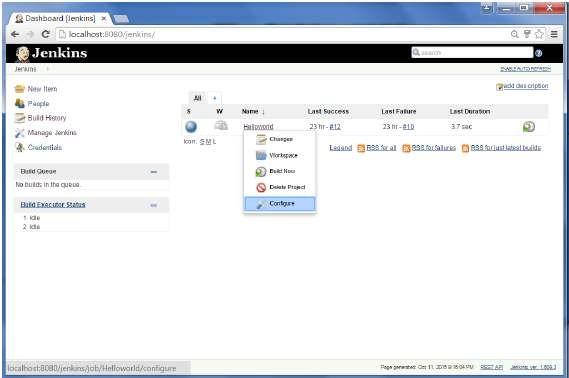
* 1. Configure the selenium server jar and click on the Save button.



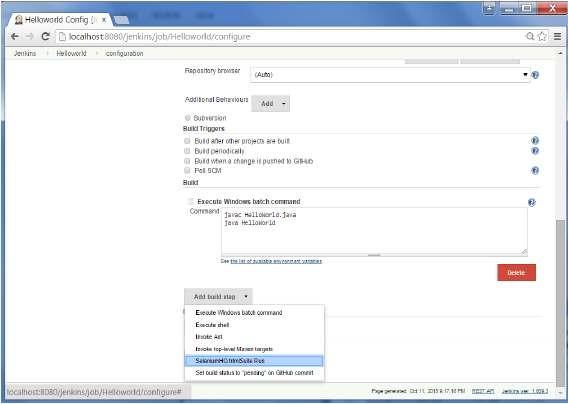
33

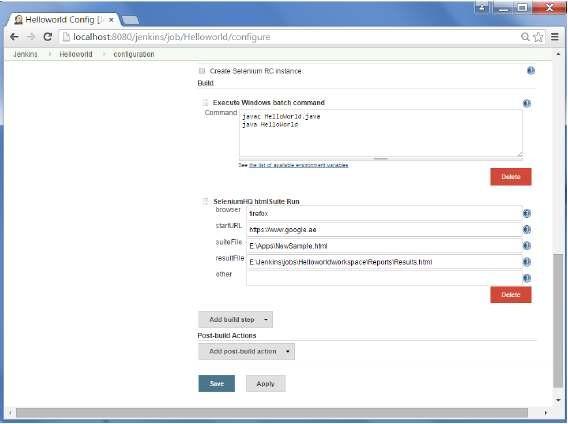
Note − The selenium jar file can be downloaded from the location [SeleniumHQ](http://www.seleniumhq.org/download/) Click on the download for the Selenium standalone server.



* 1. Go back to your dashboard and click on the Configure option for the HelloWorld project.

34

* 1. Click on Add build step and choose the optin of “SeleniumHQ htmlSuite Run”
  2. Add the necessary details for the selenium test. Click on Save and execute a build. Now the post build will launch the selenium driver, and execute the html test.



# CONCLUSION:

Successfully ran the Unit and Integration tests in Jenkins and outputs were captured.

35