EX.NO:1 INSTALLATION OF VIRTUALBOX WITH DIFFERENT FLAVOURS OF LINUX/WINDOWS

DATE:

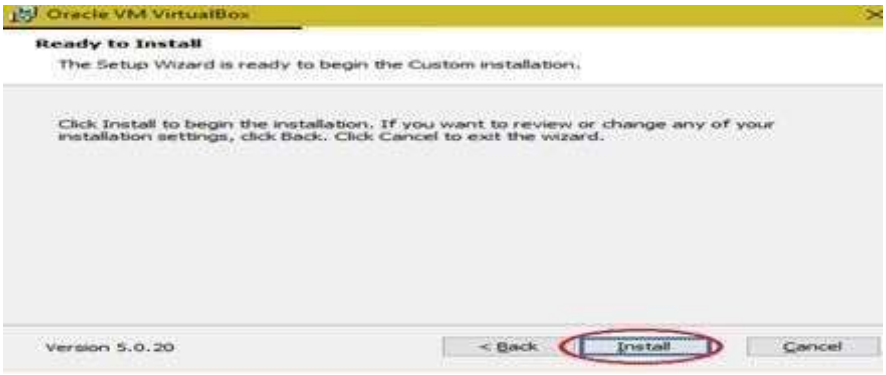
AIM:

To write the procedure to install the virtualbox with different flavours of Linux/Windows.

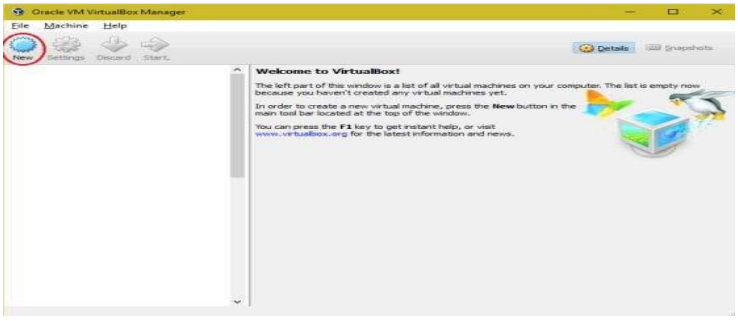
PROCEDURE:

Step 1: Run the Virtualbox steup and click the "Next" option.

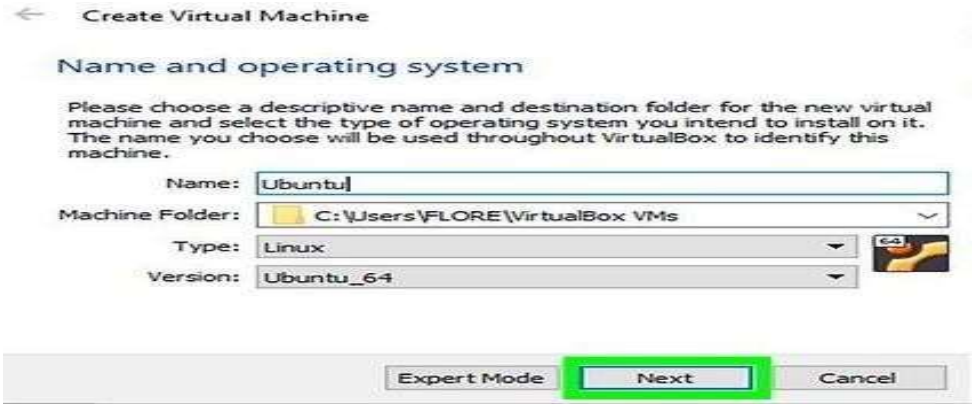
Step 2: Repeat Click on "Next" option and Finally Click the option"Install"



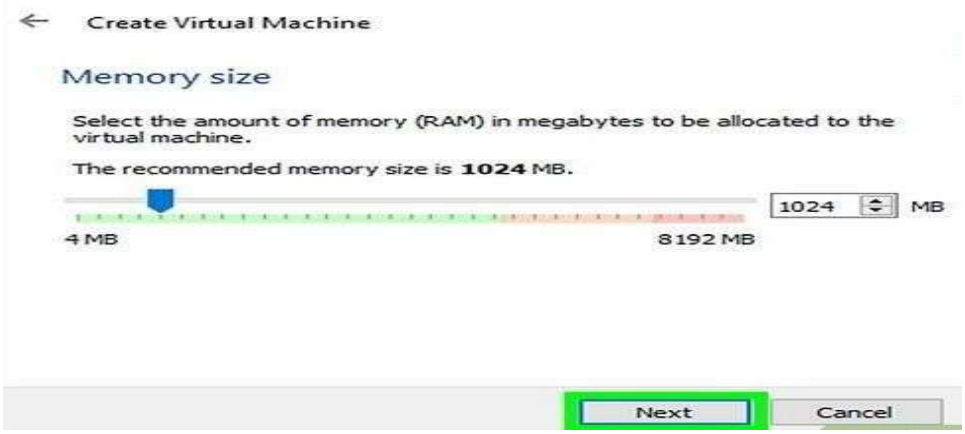
Step 3: To Install "Ubuntu" as virtual machine in "Oracle VM Virtualbox", Open the Oracle VM Virtualbox Manager



Step 4: Create new virtual machine



Step 5: Select the memory size



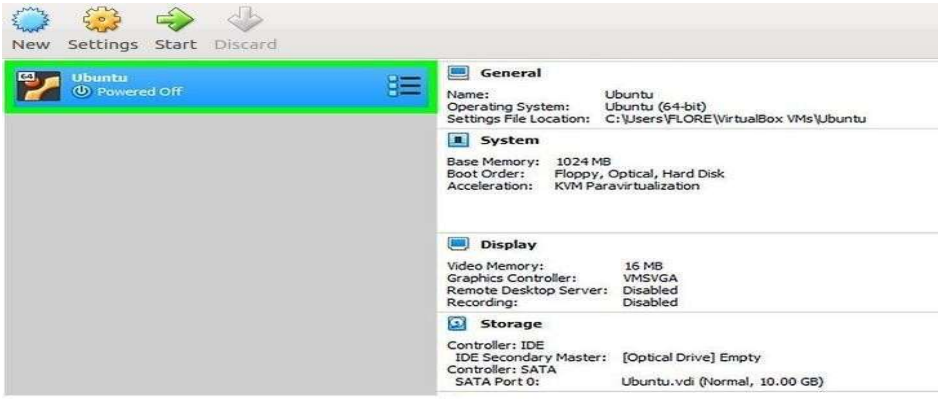
Step 6: Create a virtual hard drive

Step 7: Select the type of hardware file

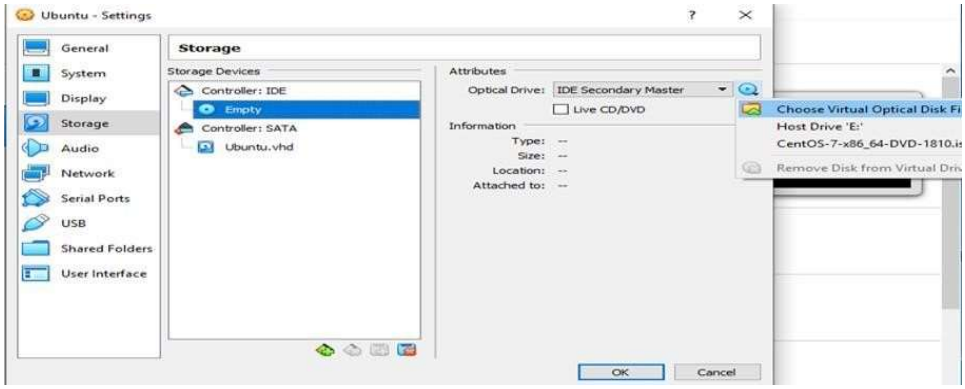
Step 8: Select the type of storage on physical hard drive

Step 9: Select the size of virtual hard drive

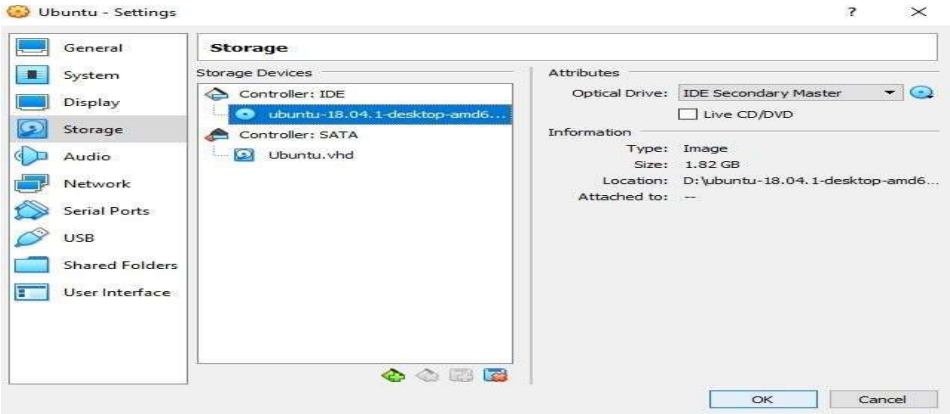
Step 10: Select the new virtual OS created and click on “Settings”



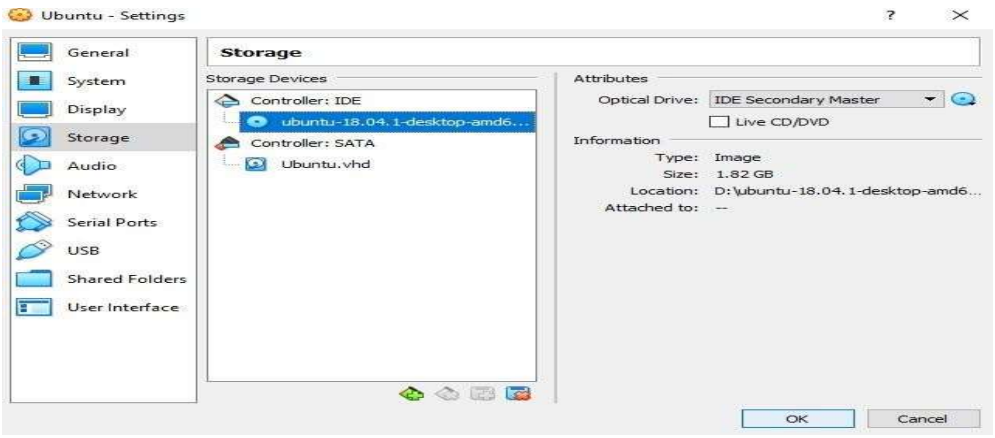
Step 11: Select “Storage” from the left panel of the window



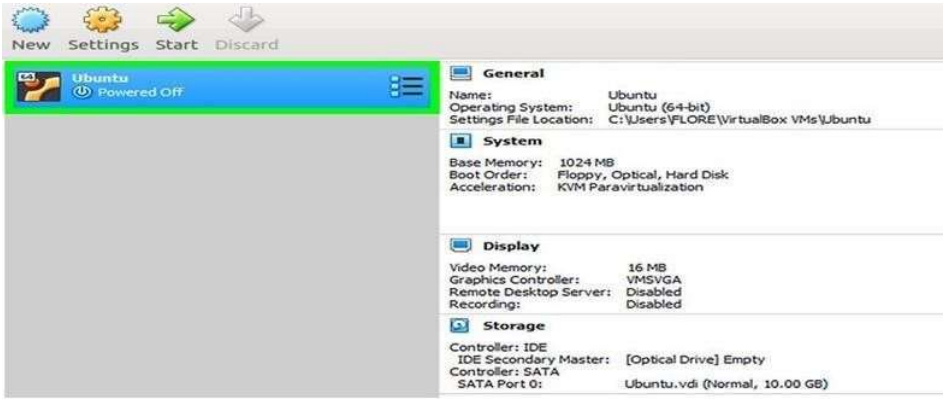
Step 12: Click on the first icon “Add CD/DVD device” in Controller:IDE



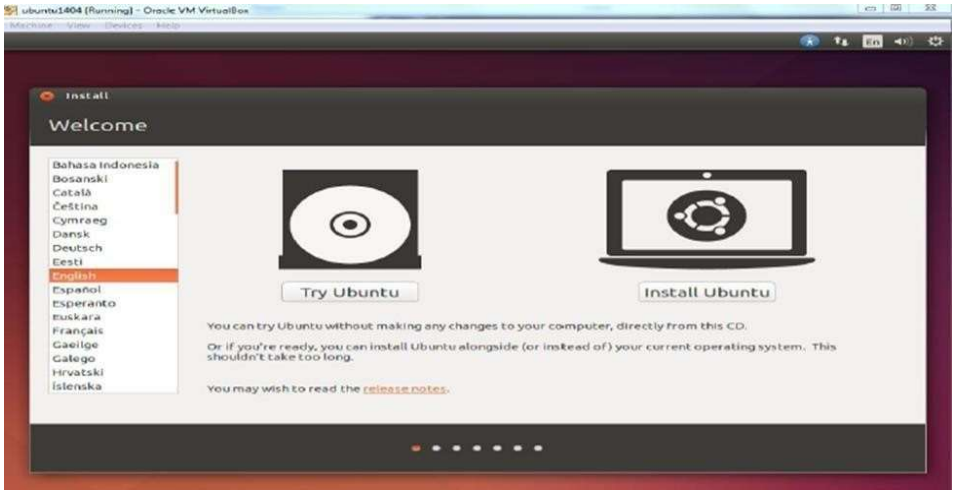
Step 13: Select "Choose Disk" and Choose the virtual machine to be used and click "Open"



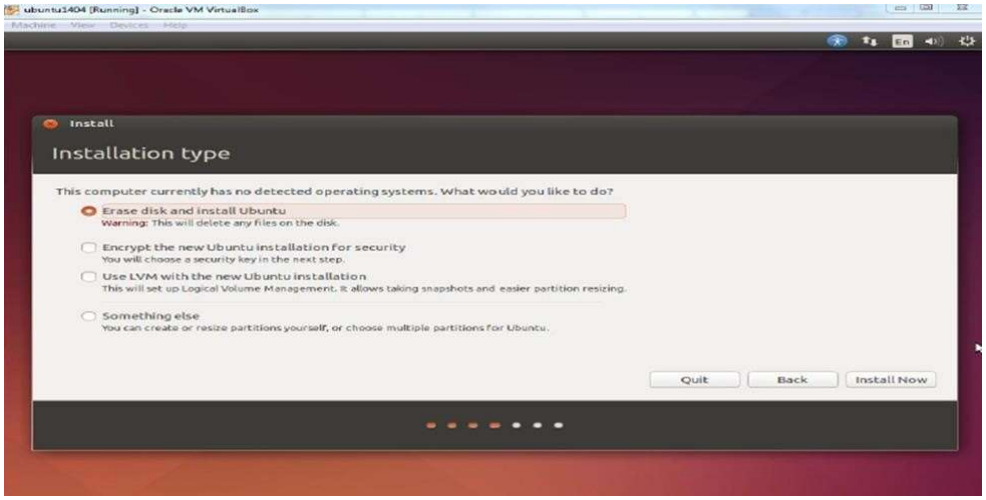
Step 14: Click "OK" and select "Start" to run the virtual machine



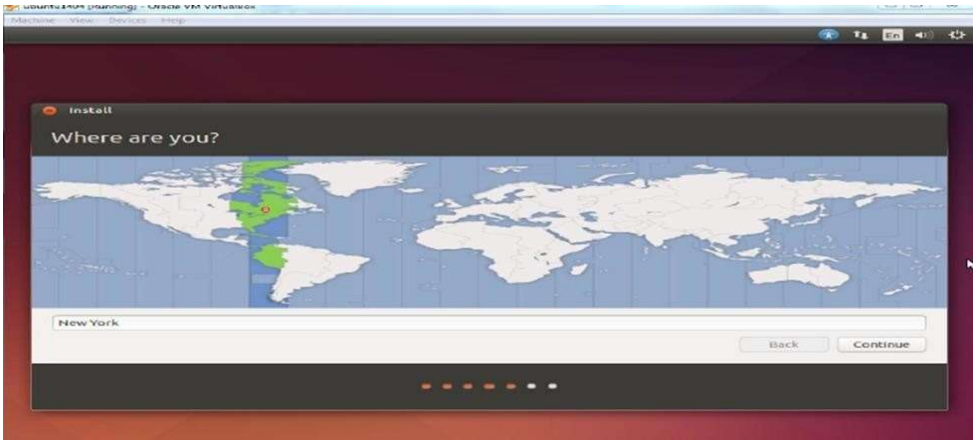
Step 15: To install ubuntu, Click 'Install Ubuntu' button



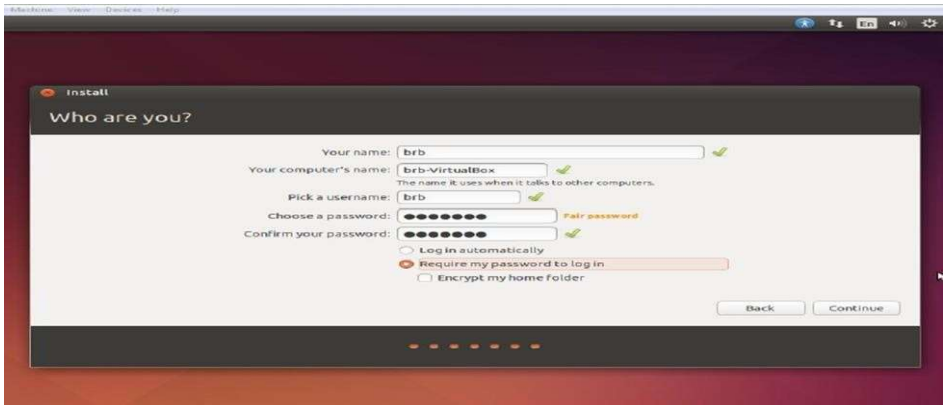
Step 16: Select 'Erase disk and install Ubuntu' option is selected and click 'Install Now' button



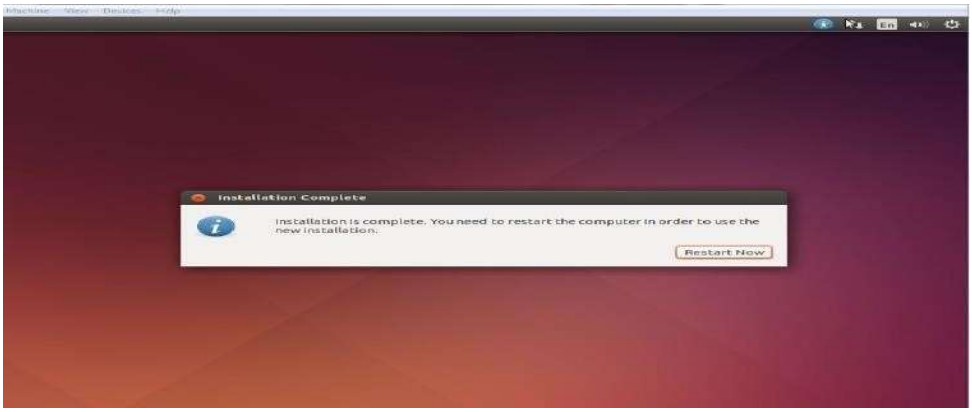
Step 17: Click 'Continue' button for upcoming dialogue box



Step 18: Enter the name, username and password. Click 'Continue' button



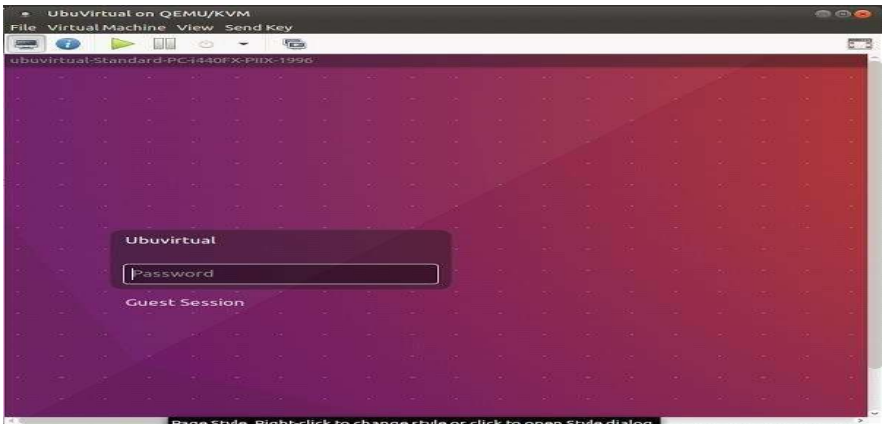
Step 19: After completion of installation process, click on 'Restart Now' button.



Step 20: The same way, we can install windows OS .

OUTPUT:

(i) Ubuntu Operating System in Virtual Machine



(ii) Windows7 Operating System in Virtual Machine

RESULT:

Thus the virtualbox with different flavours of Linux/Windows has been Installed Sucessfully.

EX.NO: 2 INSTALLATION OF A C COMPILER IN THE VIRTUAL MACHINE AND EXECUTING A SIMPLE PROGRAM

DATE:

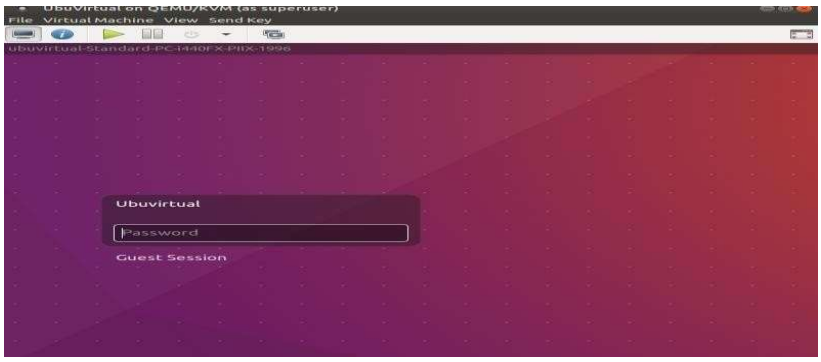
AIM:

To install a C compiler in the virtual machine and execute a simple program.

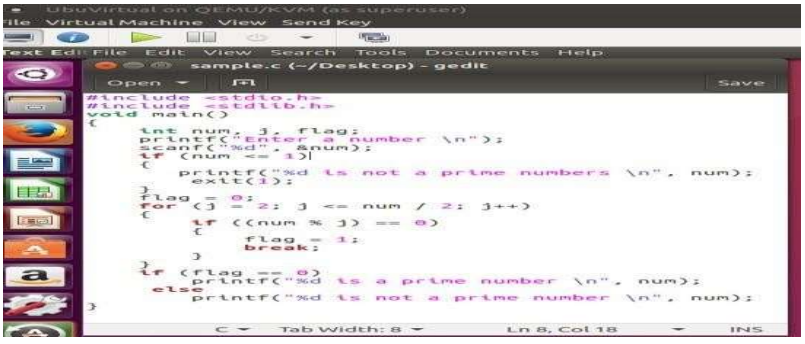
PROCEDURE:

STEP:1 Before install a C compiler in a virtual machine, we have Create a virtual machine by opening the Kernal Virtual Machine (KVM). Open the installed virtual manager. As well as install different Ubuntu and Windows OS in that virtual machine with different names.

STEP:2 Open the Ubuntu OS in our Virtual Machine.



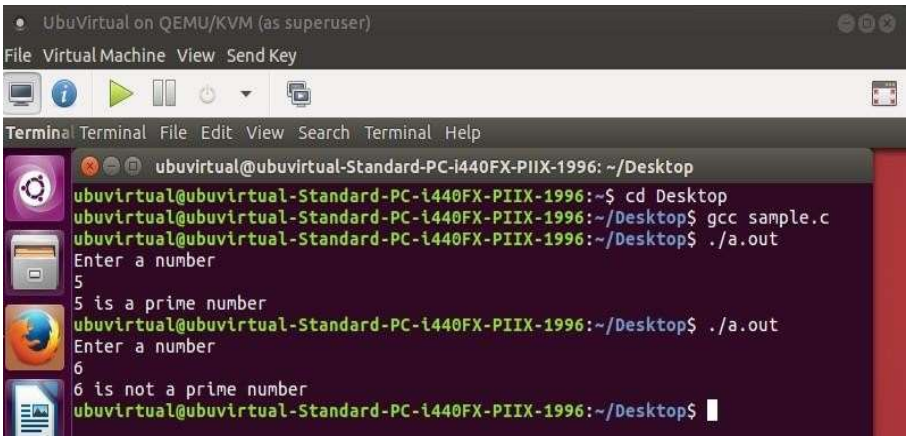
STEP:3 Open TextEditor in Ubuntu OS and type a C program and save it in desktop to execute.



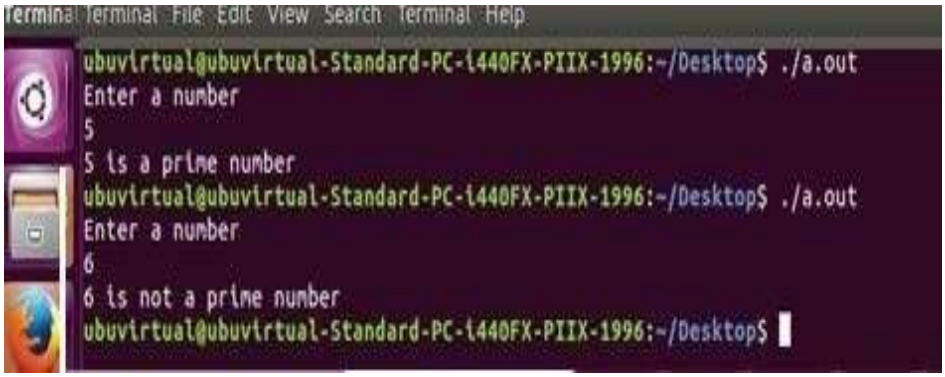
STEP:4 To install the C compiler in Ubuntu OS, open the terminal and type the command.

$sudo apt-get install gcc

STEP:5 And then compile the C program and execute it.



OUTPUT:



RESULT:

Thus the C compiler is installed in the virtual machine and executed the program Successfully.

EX.NO: 3 INSTALLATION OF GOOGLE APP ENGINE AND CREATE HELLOWORLD APP USING JAVA

DATE:

AIM:

To install google app engine and create hello world app using java.

PROCEDURE:

- **Create a project**

  Projects bundle code, VMs, and other resources together for easier development and monitoring.

- **Build and run your "Hello, world!" app**

  You will learn how to run your app using Cloud Shell, right in your browser. At the end, you'll deploy your app to the web using the App Engine Maven plugin.

  GCP (Google Cloud Platform) organizes resources into projects, which collect all of the related resources for a single application in one place.

  Begin by creating a new project or selecting an existing project for this tutorial.

  Step1:

  Select a project, or create a new one

  Step2:

  Using Cloud Shell

  Cloud Shell is a built-in command-line tool for the console. You're going to use Cloud Shell to deploy your app.

  Open Cloud Shell

  Open Cloud Shell by clicking the

  Activate Cloud Shell button in the navigation bar in the upper-right corner of the console
  Clone the sample code

  Use Cloud Shell to clone and navigate to the "Hello World" code. The sample code is cloned from your project repository to the Cloud Shell.

  Note: If the directory already exists, remove the previous files before cloning:

  ```
  rm -rf appengine-try-java

  git clone \
    https://github.com/GoogleCloudPlatform/appengine-try-java

  cd appengine-try-java
  ```

Step3:

Configuring your deployment

You are now in the main directory for the sample code. You'll look at the files that configure your application.

Exploring the application

Enter the following command to view your application code:

```
cat \
 src/main/java/myapp/DemoServlet.java
```

This servlet responds to any request by sending a response containing the message Hello, world!.

Exploring your configuration

For Java, App Engine uses XML files to specify a deployment's configuration.

Enter the following command to view your configuration file:

```
cat pom.xml
```

The helloworld app uses Maven, which means you must specify a Project Object Model, or POM, which contains information about the project and configuration details used by Maven to build the project.

Step4:

Testing your app

Test your app on Cloud Shell

Cloud Shell lets you test your app before deploying to make sure it's running as intended, just like debugging on your local machine.

To test your app enter the following:

```
mvn appengine:run
```

Preview your app with "Web preview"

Your app is now running on Cloud Shell. You can access the app by clicking the **Web preview** button at the top of the Cloud Shell pane and choosing **Preview on port 8080**.

Terminating the preview instance

Terminate the instance of the application by pressing Ctrl+C in the Cloud Shell

Step5:

Deploying to App Engine

Create an application

To deploy your app, you need to create an app in a region:

```
gcloud app create
```

Note: If you already created an app, you can skip this step.

Deploying with Cloud Shell

Now you can use Cloud Shell to deploy your app.

First, set which project to use:

```
gcloud config set project \
  <YOUR-PROJECT>
```

Then deploy your app:

```
mvn appengine:deploy
```

Visit your app

Congratulations! Your app has been deployed.

The default URL of your app is a subdomain on appspot.com that starts with your project's ID: <your-project>.appspot.com.

Try visiting your deployed application.

View your app's status

You can check in on your app by monitoring its status on the App Engine dashboard.

Open the **Navigation menu** in the upper-left corner of the console.

Then, select the **App Engine** section

RESULT:

Thus google app engine is installed and hello world app using java created successfully.

EX.NO: 4 LAUNCH THE WEB APPLICATIONS USING GAE LAUNCHER

DATE:

AIM:

To launch the web applications using gae launcher.

PROCEDURE:

Before you can host your website on Google App Engine:

1. Create a new Cloud Console project or retrieve the project ID of an existing project to use:

Go to the Projects page (https://console.cloud.google.com/project)

Tip: You can retrieve a list of your existing project IDs with the gcloud command line tool (#before_you_begin).

2. Install and then initialize the Google Cloud SDK:

Download the SDK (/sdk/docs)

Creating a website to host on Google App Engine

Basic structure for the project

This guide uses the following structure for the project:

app.yaml: Configure the settings of your App Engine application.

www/: Directory to store all of your static files, such as HTML, CSS, images, and JavaScript.

css/: Directory to store stylesheets.

style.css: Basic stylesheet that formats the look and feel of your site.

images/: Optional directory to store images.

index.html: An HTML file that displays content for your website.

js/: Optional directory to store JavaScript files.

Other asset directories.

Creating the app.yaml _le

The app.yaml file is a configuration file that tells App Engine how to map URLs to your static files. In the following steps, you will add handlers that will load www/index.html when someone visits your website, and all static files will be stored in and called from the

www directory.

Create the app.yaml file in your application's root directory:

1. Create a directory that has the same name as your project ID. You can find your project ID in the Console (https://console.cloud.google.com/).
2. In directory that you just created, create a file named app.yaml.
3. Edit the app.yaml file and add the following code to the file:

```
runtime: python27
api_version: 1
threadsafe: true
handlers:
- url: /
static_files: www/index.html
upload: www/index.html
- url: /(.*)
static_files: www/\1
upload: www/(.*)
```

More reference information about the app.yaml file can be found in the app.yaml reference documentation (/appengine/docs/standard/python/config/appref).

Creating the index.html _le

Create an HTML file that will be served when someone navigates to the root page of your website. Store this file in your www directory.

Deploying your application to App Engine

When you deploy your application files, your website will be uploaded to App Engine. To deploy your app, run the following command from within the root directory of your application where the app.yaml file is located:

Optional flags:

Include the --project flag to specify an alternate Cloud Console project ID to what you initialized as the default in the gcloud tool. Example: --project [YOUR_PROJECT_ID]

Include the -v flag to specify a version ID, otherwise one is generated for you.

Example: -v [YOUR_VERSION_ID]

```
l>
ead>
<title>Hello, world!</title>
<link rel="stylesheet" type="text/css" href="/css/style.css">
```

head>

ody>

<h1>Hello, world!</h1>

This is a simple static HTML file that will be served from Google App Engine.

body>

ml>

Deploying your application to App Engine

When you deploy your application files, your website will be uploaded to App Engine. To deploy your app, run the following command from within the root directory of your application where the app.yaml file is located:

id app deploy

Optional flags:

Include the --project flag to specify an alternate Cloud Console project ID to what you initialized as the default in the gcloud tool. Example: --project [YOUR_PROJECT_ID]

Include the -v flag to specify a version ID, otherwise one is generated for you.

Example: -v [YOUR_VERSION_ID

To learn more about deploying your app from the command line, see Deploying a Python 2 App (/appengine/docs/python/tools/uploadinganapp).

Viewing your application

To launch your browser and view the app at https://**PROJECT_ID**.**REGION_ID** (#appengine-urls).r.appspot.com, run the following command

ud app browse

RESULT:

Thus the web applications using gae launcher launched.

EX.NO: 5 SIMULATE A CLOUD SCENARIO USING CLOUDSIM AND RUN A SCHEDULING ALGORITHM THAT IS NOT PRESENT IN CLOUDSIM

DATE:

What is Cloudsim?

CloudSim is a simulation toolkit that supports the modelling and simulation of the core functionality of cloud, like job/task queue, processing of events, creation of cloud entities (datacentre, datacentre brokers, etc), communication between different entities, implementation of broker policies, etc. This toolkit allows to:

- Test application services in a repeatable and controllable environment.
- Tune the system bottlenecks before deploying apps in an actual cloud.
- Experiment with different workload mix and resource performance scenarios on simulated infrastructure for developing and testing adaptive application provisioning techniques

Core features of CloudSim are:

- The Support of modelling and simulation of large scale computing environment as federated cloud data centres, virtualized server hosts, with customizable policies for provisioning host resources to virtual machines and energy-aware computational resources
- It is a self-contained platform for modelling cloud's service brokers, provisioning, and allocation policies.
- It supports the simulation of network connections among simulated system elements.
- Support for simulation of federated cloud environment, that inter-networks resources from both private and public domains.
- Availability of a virtualization engine that aids in the creation and management of multiple independent and co-hosted virtual services on a data centre node.
- Flexibility to switch between space shared and time-shared allocation of processing cores to virtualized services.

```
import java.text.DecimalFormat;
import java.util.Calendar;
import java.util.List;
import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.Datacenter;
```

```
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.core.CloudSim;

/**
 * FCFS Task scheduling
 * @author Linda J
 */
public class FCFS {

   /** The cloudlet list. */
   private static List<Cloudlet> cloudletList;

   /** The vmlist. */
   private static List<Vm> vmlist;

   private static int reqTasks = 5;
   private static int reqVms = 2;

   /**
    * Creates main() to run this example
    */
   public static void main(String[] args) {

      Log.printLine("Starting FCFS...");

         try {
            // First step: Initialize the CloudSim package. It should be called
               // before creating any entities.
               int num_user = 1; // number of cloud users
               Calendar calendar = Calendar.getInstance();
               boolean trace_flag = false;  // mean trace events

               // Initialize the CloudSim library
               CloudSim.init(num_user, calendar, trace_flag);
```

```
        // Second step: Create Datacenters
        //Datacenters are the resource providers in CloudSim. We need at list one of
them to run a CloudSim simulation
        @SuppressWarnings("unused")
        Datacenter datacenter0 = createDatacenter("Datacenter_0");

        //Third step: Create Broker
        FcfsBroker broker = createBroker();
        int brokerId = broker.getId();

        //Fourth step: Create one virtual machine
        vmlist = new VmsCreator().createRequiredVms(reqVms, brokerId);

        //submit vm list to the broker
        broker.submitVmList(vmlist);

        //Fifth step: Create two Cloudlets
        cloudletList = new CloudletCreator().createUserCloudlet(reqTasks, brokerId);

        //submit cloudlet list to the broker
        broker.submitCloudletList(cloudletList);

        //call the scheduling function via the broker
        broker.scheduleTaskstoVms();


        // Sixth step: Starts the simulation
        CloudSim.startSimulation();

        // Final step: Print results when simulation is over
        List<Cloudlet> newList = broker.getCloudletReceivedList();

        CloudSim.stopSimulation();

        printCloudletList(newList);
```

```
            Log.printLine("FCFS finished!");
        }
        catch (Exception e) {
            e.printStackTrace();
            Log.printLine("The simulation has been terminated due to an unexpected error");
        }
    }

    private static Datacenter createDatacenter(String name){
        Datacenter datacenter=new DataCenterCreator().createUserDatacenter(name,
reqVms);

        return datacenter;

    }

    //We strongly encourage users to develop their own broker policies, to submit vms and
cloudlets according
    //to the specific rules of the simulated scenario
    private static FcfsBroker createBroker(){
        FcfsBroker broker = null;
        try {
        broker = new FcfsBroker("Broker");
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        return broker;
    }

    /**
     * Prints the Cloudlet objects
     * @param list  list of Cloudlets
     */
```

```
    private static void printCloudletList(List<Cloudlet> list) {
        int size = list.size();
        Cloudlet cloudlet;

        String indent = "    ";
        Log.printLine();
        Log.printLine("========== OUTPUT ==========");
        Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +
                "Data center ID" + indent + "VM ID" + indent + "Time" + indent + "Start Time"
+ indent + "Finish Time");

        DecimalFormat dft = new DecimalFormat("###.##");
        for (int i = 0; i < size; i++) {
            cloudlet = list.get(i);
            Log.print(indent + cloudlet.getCloudletId() + indent + indent);

            if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS){
                Log.print("SUCCESS");

                Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent +
indent + cloudlet.getVmId() +
                        indent + indent + dft.format(cloudlet.getActualCPUTime()) + indent + indent
+ dft.format(cloudlet.getExecStartTime())+
                            indent + indent + dft.format(cloudlet.getFinishTime()));
            }
        }

    }
}
```

RESULTS :

Thus simulating a cloud scenario using cloudsim is simulated successfully.

EX.NO: 6 PROCEDURE TO TRANSFER THE FILES FROM ONE VIRTUAL MACHINE TO ANOTHER VIRTUAL MACHINE

DATE:

Aim:

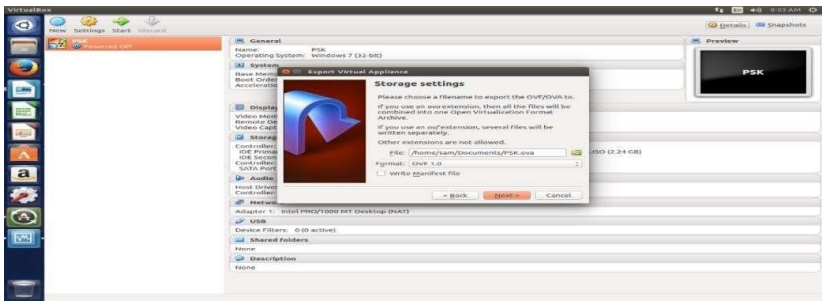
To write a procedure to transfer the files from one Virtual Machine to another Virtual Machine.

PROCEDURE:

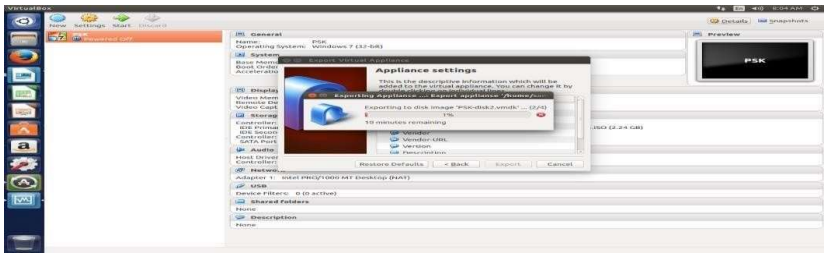
STEP:1 Select the VM and click File->Export Appliance

STEP:2 Select the VM to be exported and click NEXT.

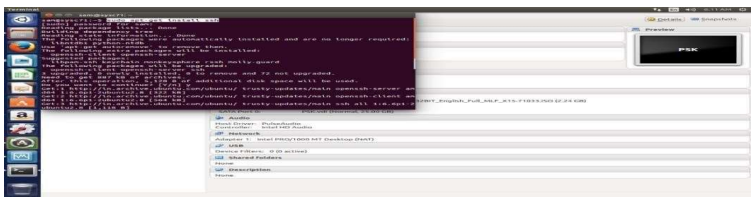
STEP:3 Note the file path and click “Next”



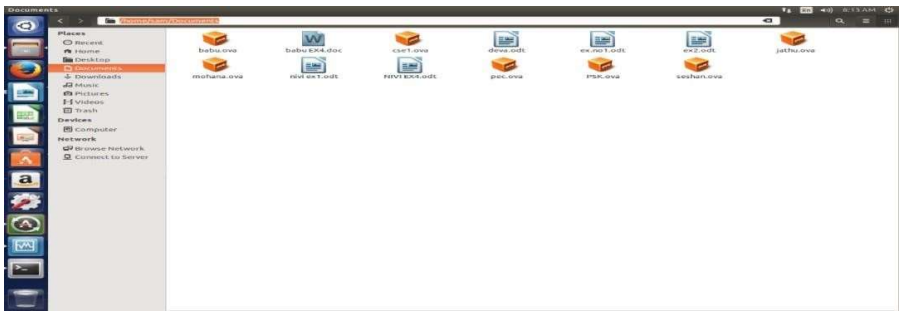
STEP:4 Click “Export” ,The Virtual machine is being exported.



STEP:5 Install “ssh” to access the neighbour's VM.

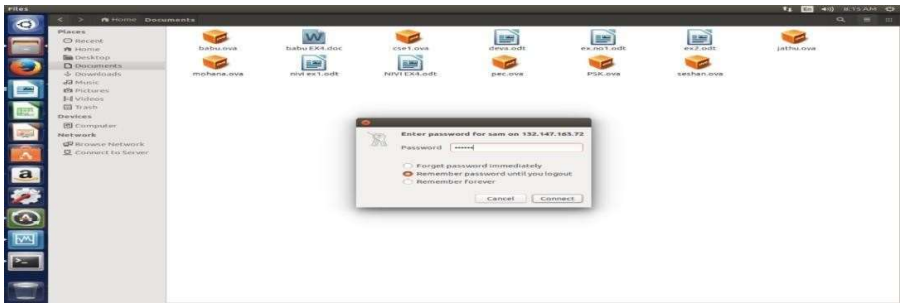


STEP:6 Go to File->Computer:/home/sam/Documents/



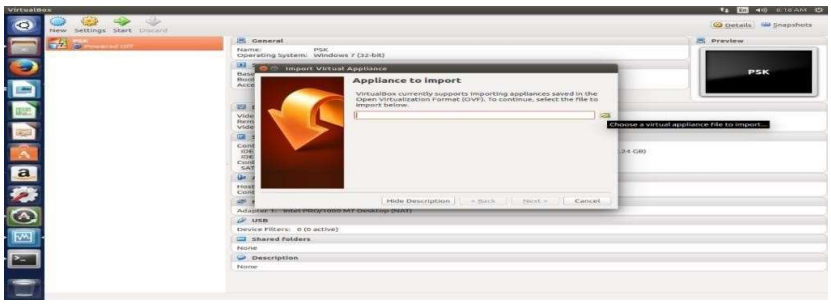
STEP:7 Type the neighbour's URL: sftp://sam@172.16.42._/

STEP:8 Give the password and get connected.

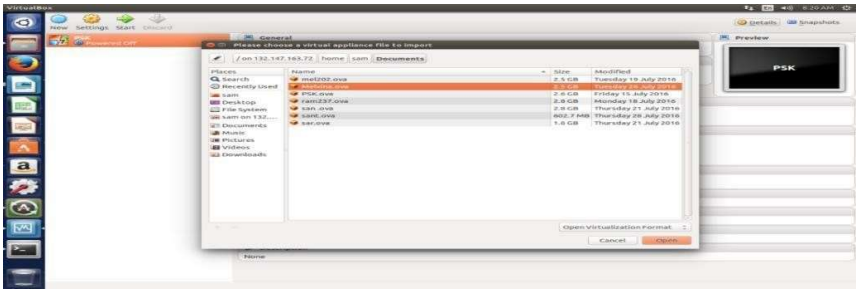


STEP:9 Select the VM and copy it in desktop.

STEP:10 Open VirtualBox and select File->Import Appliance->Browse



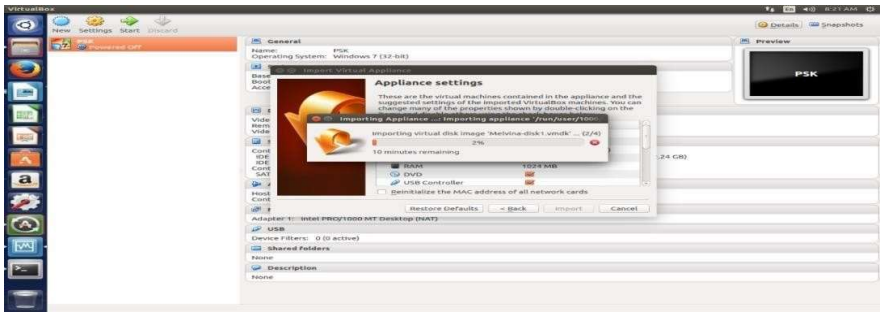
STEP:11 Select the VM to be imported and click “Open”.



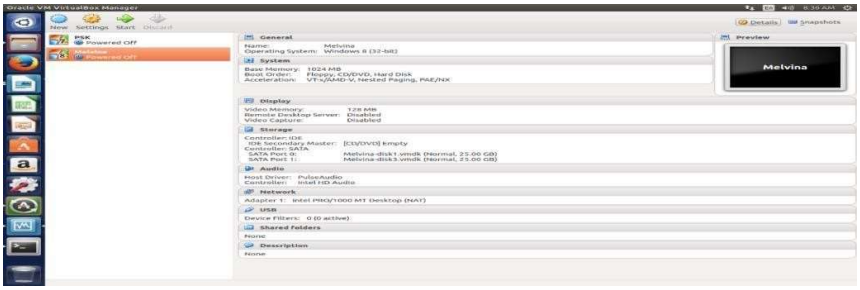
STEP:12 Click “Next”

STEP:13 Click “Import”.

STEP:14 VM is being imported.



STEP:15 VM is imported.



RESULT:

Thus the files from one Virtual Machine to another Virtual Machine is transferred Successfully..

EXP NO :7 INSTALL HADOOP SINGLE NODE CLUSTER AND RUN SIMPLE APPLICATIONS LIKE WORDCOUNT

DATE:

Aim:

To Install Hadoop single node cluster and run simple applications like wordcount.

Steps:

Install Hadoop

Step 1: Click here to download the Java 8 Package. Save this file in your home directory.

Step 2: Extract the Java Tar File.

***Command*:** tar -xvf jdk-8u101-linux-i586.tar.gz

Fig: Hadoop Installation – Extracting Java Files

Step 3: **Download the Hadoop 2.7.3 Package.**

***Command*:** wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.3/hadoop-2.7.3.tar.gz

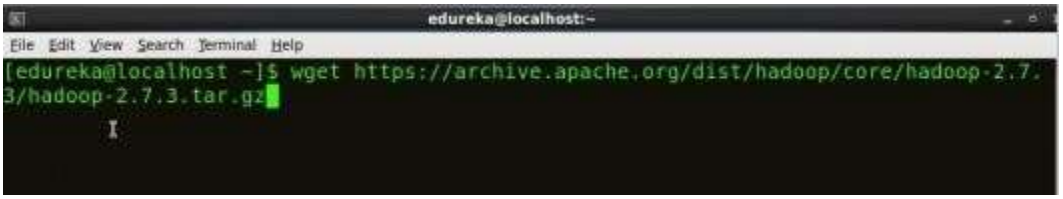
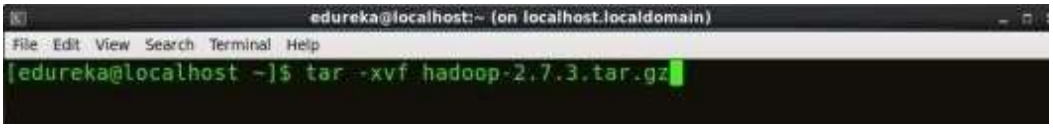


Fig: Hadoop Installation – Downloading Hadoop

Step 4: Extract the Hadoop tar File.

Command: tar -xvf hadoop-2.7.3.tar.gz



5: Add the Hadoop and Java paths in the bash file (.bashrc). Open**. bashrc**

file. Now, add Hadoop and Java Path as shown below.

***Command*:** vi .bashrc

Fig: Hadoop Installation – Setting Environment Variable

Then, save the bash file and close it.

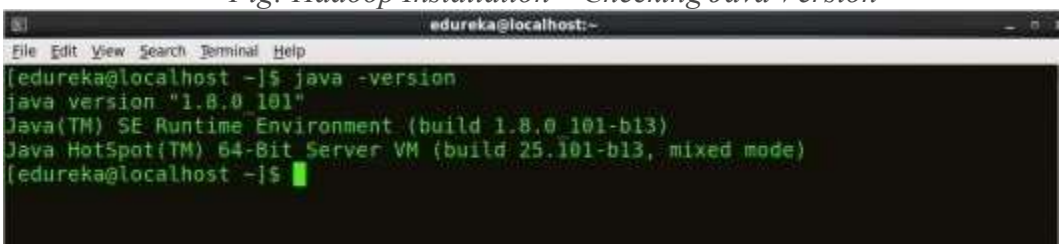
For applying all these changes to the current Terminal, execute the source command.
***Command*:** source .bashrc

Fig: Hadoop Installation – Refreshing environment variables

To make sure that Java and Hadoop have been properly installed on your system and can be accessed through the Terminal, execute the java -version and hadoop version commands.

***Command*:** java -version

Fig: Hadoop Installation – Checking Java Version



***Command*:** hadoop version

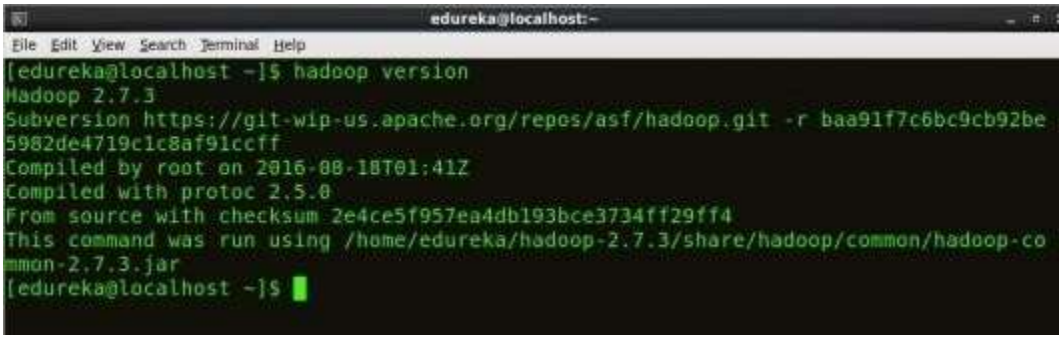


Fig: Hadoop Installation – Checking Hadoop Version

Step 6: Edit the **Hadoop Configuration files**.

Command: cd hadoop-2.7.3/etc/hadoop/

Command: ls

All the Hadoop configuration files are located in **hadoop-2.7.3/etc/hadoop** directory as you can see in the snapshot below:

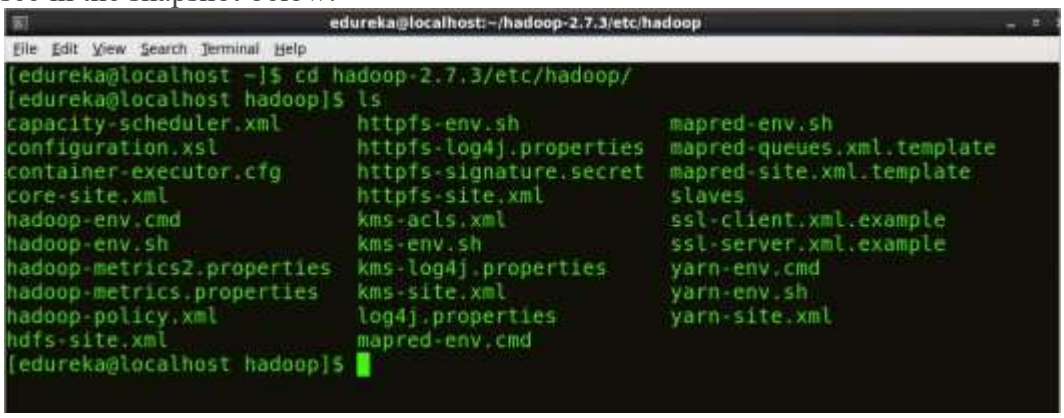


Fig: Hadoop Installation – Hadoop Configuration Files

Step 7: Open *core-site.xml* and edit the property mentioned below inside configuration tag:

core-site.xml informs Hadoop daemon where NameNode runs in the cluster. It contains configuration settings of Hadoop core such as I/O settings that are common to HDFS & MapReduce.

***Command*:** vi core-site.xml

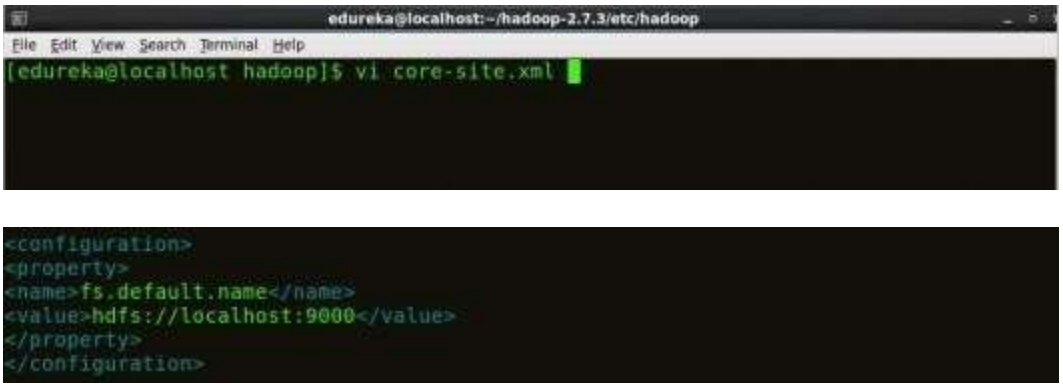


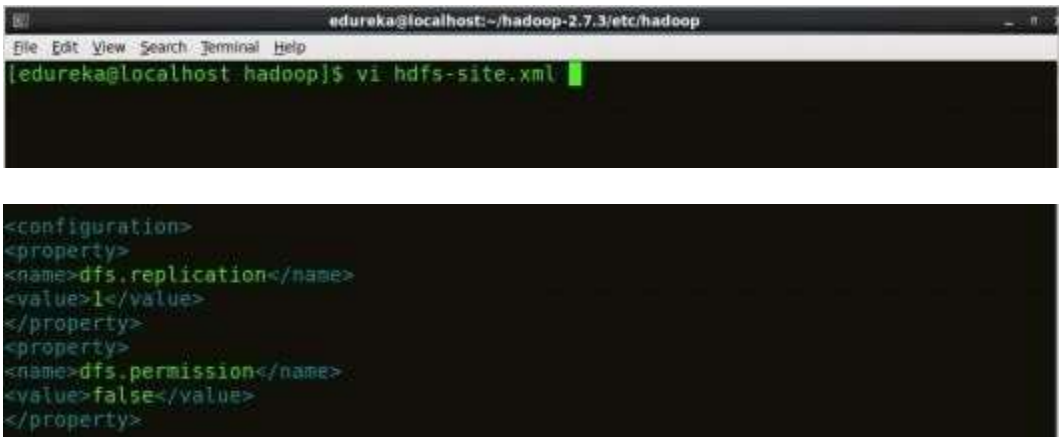
Fig: Hadoop Installation – Configuring core-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
```

Step 8: Edit *hdfs-site.xml* and edit the property mentioned below inside **configuration tag:**

hdfs-site.xml contains configuration settings of HDFS daemons (i.e. NameNode, DataNode, Secondary NameNode). It also includes the replication factor and block size of HDFS.

***Command*:** vi hdfs-site.xml



z

Fig: Hadoop Installation – Configuring hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.permission</name>
<value>false</value>
</property>
</configuration>
```

Step 9: Edit the *mapred-site.xml* file and edit the property mentioned below

inside configuration tag:

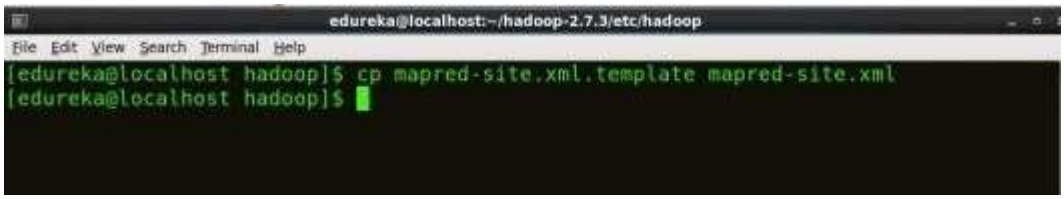
mapred-site.xml contains configuration settings of MapReduce application like number of JVM that can run in parallel, the size of the mapper and the reducer process, CPU cores available for a process, etc.

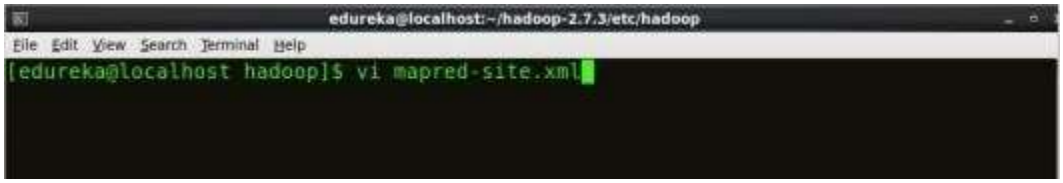
In some cases, mapred-site.xml file is not available. So, we have to create the mapred- site.xml file using mapred-site.xml template.

***Command*:** cp mapred-site.xml.template mapred-site.xml

***Command*:** vi mapred-site.xml.

Fig: Hadoop Installation – Configuring mapred-site.xml





```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
</configuration>
```

Step 10: Edit *yarn-site.xml* and edit the property mentioned below inside configuration tag:

yarn-site.xml contains configuration settings of ResourceManager and NodeManager like application memory management size, the operation needed on program & algorithm, etc.

***Command*:** vi yarn-site.xml

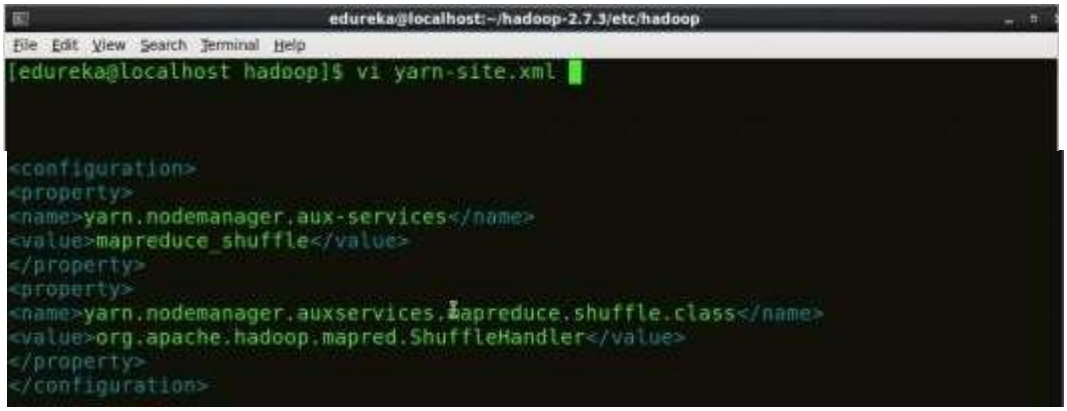


Fig: Hadoop Installation – Configuring yarn-site.xml

Step 11: Edit *hadoop-env.sh* and add the Java Path as mentioned below:

```
<?xml version="1.0">
<configuration>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</
name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

hadoop-env.sh contains the environment variables that are used in the script to run Hadoop like Java home path, etc.

***Command*:** vi hadoop–env.sh

Fig: Hadoop Installation – Configuring hadoop-env.sh **Step**

12: Go to Hadoop home directory and format the NameNode.

***Command*:** cd

***Command*:** cd hadoop-2.7.3

***Command*:** bin/hadoop namenode -format

Fig: Hadoop Installation – Formatting NameNode

This formats the HDFS via NameNode. This command is only executed for the first time. Formatting the file system means initializing the directory specified by the dfs.name.dir variable.

Never format, up and running Hadoop filesystem. You will lose all your data stored in the HDFS.

Step 13: **Once the NameNode is formatted, go to hadoop-2.7.3/sbin directory and start all the daemons.**

Command: cd hadoop-2.7.3/sbin

Either you can start all daemons with a single command or do it individually.

Command: ./start-all.sh

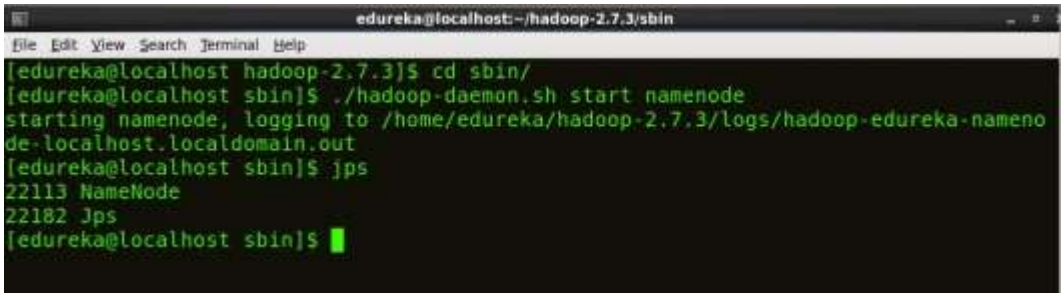
The above command is a combination of ***start-dfs.sh, start-yarn.sh*** & ***mr-jobhistory-daemon.sh***

Or you can run all the services individually as below:

Start NameNode:

The NameNode is the centerpiece of an HDFS file system. It keeps the directory tree of all files stored in the HDFS and tracks all the file stored across the cluster.

Command: ./hadoop-daemon.sh start namenode



Start DataNode:

On startup, a DataNode connects to the Namenode and it responds to the requests from the Namenode for different operations.

Command: ./hadoop-daemon.sh start datanode

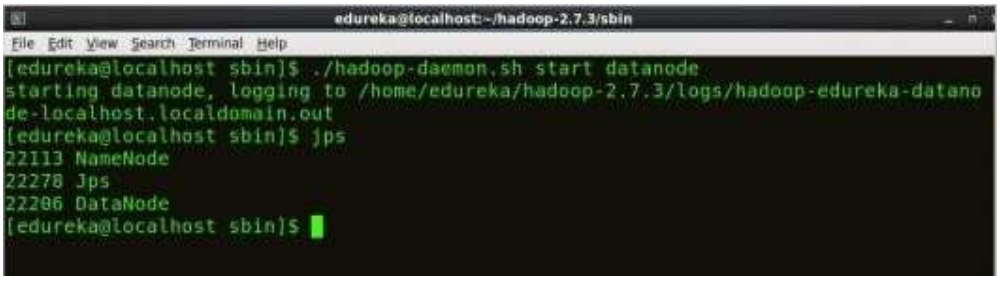


Fig: Hadoop Installation – Starting DataNode

Start ResourceManager:

ResourceManager is the master that arbitrates all the available cluster resources and thus helps in managing the distributed applications running on the YARN system. Its work is to manage each NodeManagers and the each application's ApplicationMaster.

Command: ./yarn-daemon.sh start resourcemanager

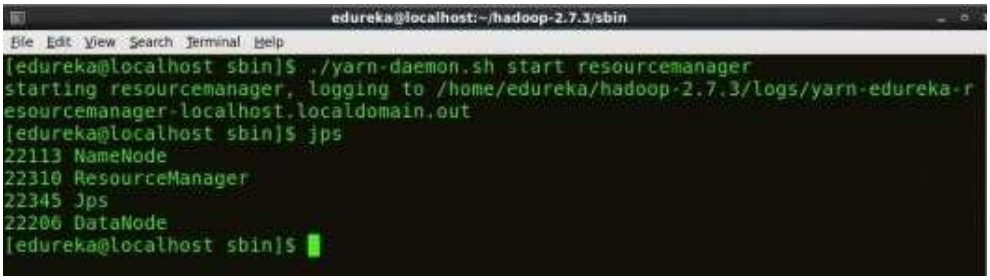


Fig: Hadoop Installation – Starting ResourceManager

Start NodeManager:

The NodeManager in each machine framework is the agent which is responsible for managing containers, monitoring their resource usage and reporting the same to the ResourceManager.

Command: ./yarn-daemon.sh start nodemanager

See Batch Details

Fig: Hadoop Installation – Starting NodeManager

Start JobHistoryServer:

JobHistoryServer is responsible for servicing all job history related requests from client.

***Command*:** ./mr-jobhistory-daemon.sh start historyserver

Step 14: **To check that all the Hadoop services are up and running, run the below command.**

Command: jps

Fig: Hadoop Installation – Checking Daemons

Step 15: Now open the Mozilla browser and go to **localhost**:**50070/dfshealth.html** to check the NameNode interface.

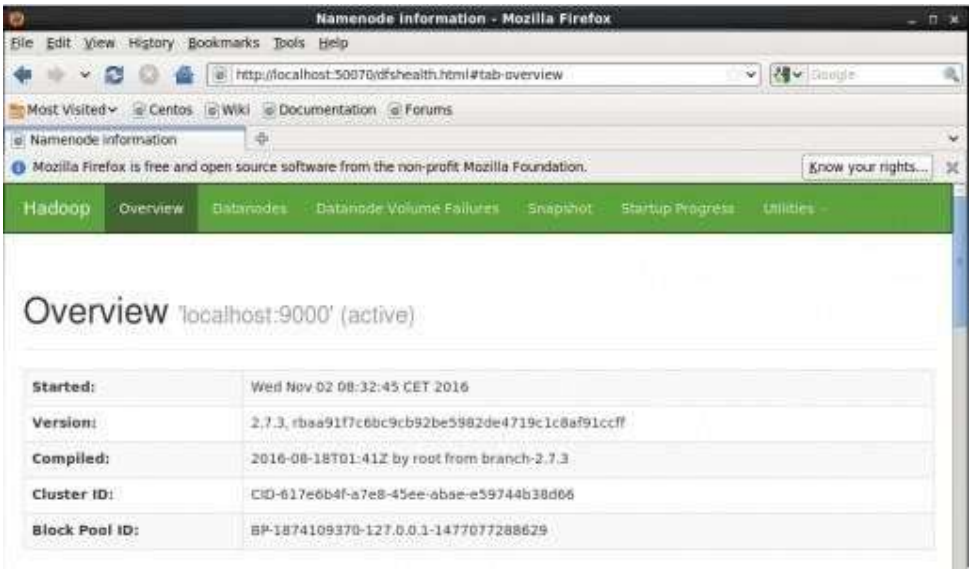


Fig: Hadoop Installation – Starting WebUI

Congratulations, you have successfully installed a single node Hadoop cluster.

Result:

Thus the Hadoop one cluster was installed and simple applications executed successfully.

EXP NO :8 CREATING AND EXECUTING YOUR FIRST CONTAINER USING DOCKER.

DATE:

AIM:

To create and execute your first container using Docker.

PROCEDURE

To follow the instructions specific to your operating system. Here are the general steps for the most popular operating systems:

Windows:

a. Visit the Docker website (https://www.docker.com/) and navigate to the "Get Docker" section.

b. Click on the "Download for Windows" button to download the Docker Desktop installer.

c. Run the installer and follow the prompts. During the installation, Docker may require you to enable Hyper-V and Containers features, so make sure to enable them if prompted.

d. Once the installation is complete, Docker Desktop will be installed on your Windows machine. You can access it from the Start menu or the system tray.

Mac:

a. Visit the Docker website (https://www.docker.com/) and navigate to the "Get Docker" section.

b. Click on the "Download for Mac" button to download the Docker Desktop installer.

c. Run the installer and drag the Docker icon to the Applications folder to install Docker Desktop.

d. Launch Docker Desktop from the Applications folder or the Launchpad. It will appear in the status bar at the top of your screen.

Linux:

Docker supports various Linux distributions. The exact installation steps may vary based on your distribution. Here's a general outline:

a. Visit the Docker website (https://www.docker.com/) and navigate to the "Get Docker" section.

b. Click on the "Download for Linux" button.

c. Docker provides installation instructions for various Linux distributions such as Ubuntu, CentOS, Debian, Fedora, and more. Follow the instructions specific to your distribution.

d. Once Docker is installed, start the Docker service using the appropriate command for your Linux distribution.

After completing the installation, you can open a terminal or command prompt and run the docker --version command to verify that Docker is installed correctly. It should display the version of Docker installed on your system.

That's it! You now have Docker installed on your machine and can start using it to manage containers.

Install Docker: First, you need to install Docker on your machine. Docker provides platform-specific installation instructions on their website for different operating systems. Follow the instructions to install Docker for your particular OS.

Docker Image: Docker containers are created based on Docker images. An image is a lightweight, standalone, and executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, and system tools. Docker Hub (hub.docker.com) is a popular online repository of Docker images. You can search for existing images on Docker Hub or create your own. For this example, we'll use an existing image.

Pull an Image: Open your terminal or command prompt and execute the following command to pull an existing Docker image from Docker Hub. We'll use the official hello-world image as an example:

Copy code
docker pull hello-world
Docker will download the image from the Docker Hub repository.

Run a Container: Once you have the Docker image, you can create and run a container based on that image. Execute the following command to run the hello-world container:

arduino
Copy code
docker run hello-world
Docker will create a container from the image and execute it. The container will print a "Hello from Docker!" message along with some information about your Docker installation.

Note: If you haven't pulled the hello-world image in the previous step, Docker will automatically download it before running the container.

Congratulations! You've created and executed your first Docker container. Docker will handle the container lifecycle, including starting, stopping, and managing resources for you. This simple example demonstrates the basic concept of running a container using Docker.

You can explore further by trying out different Docker images and running more complex applications within containers.

RESULT:

Thus the first container using docker is created and executed successfully.

EXP NO :9 **RUN A CONTAINER FROM DOCKER HUB**

AIM:

To run a container from docker hub

PROCEDURE:

Run a Container from Docker Hub

To run a container from Docker Hub, you need to follow these steps:

Search for an Image: Visit the Docker Hub website (https://hub.docker.com/) and use the search bar to find the image you want to run. You can search for popular images like nginx, mysql, redis, etc., or specific images based on your requirements.

Pull the Image: Once you've found the desired image, open a terminal or command prompt and execute the following command to pull the image from Docker Hub:

php
Copy code
docker pull <image_name>
Replace <image_name> with the name of the image you want to pull. For example, if you want to pull the nginx image, you would use:

Copy code
docker pull nginx
Docker will download the image and store it on your local machine.

Run the Container: After pulling the image, you can create and run a container based on that image. Use the following command:

arduino
Copy code
docker run <image_name>
Replace <image_name> with the name of the image you pulled. For example:

arduino
Copy code
docker run nginx
Docker will create a container from the image and start it. The container will run the default command specified in the image, such as starting a web server, database, or any other application.

Note: By default, Docker will allocate a random port on your host machine and map it to the container's exposed ports. If you want to specify a specific port mapping, you can use the -p option. For example:

arduino
Copy code
docker run -p 8080:80 nginx
This will map port 8080 on your host machine to port 80 inside the container.

Interact with the Container: Once the container is running, you can interact with it as needed. For example, if you ran an nginx container, you can access it in your web browser by visiting http://localhost or http://<your_host_ip> (if you specified a port mapping).

To stop the container, you can use the docker stop command followed by the container ID or name:

arduino
Copy code
docker stop <container_id_or_name>
You can list the running containers using the docker ps command and stop them as necessary.

(or)

Flow-1: Pull Docker Image from Docker Hub and Run it
Step-1: Verify Docker version and also login to Docker Hub

docker version
docker login
Step-2: Pull Image from Docker Hub

docker pull stacksimplify/dockerintro-springboot-helloworld-rest-api:1.0.0-RELEASE
Step-3: Run the downloaded Docker Image & Access the Application
Copy the docker image name from Docker Hub

docker run --name app1 -p 80:8080 -d stacksimplify/dockerintro-springboot-helloworld-rest-api:1.0.0-RELEASE
http://localhost/hello
Step-4: List Running Containers

docker ps
docker ps -a
docker ps -a -q
Step-5: Connect to Container Terminal

docker exec -it <container-name> /bin/sh
Step-6: Container Stop, Start

docker stop <container-name>
docker start <container-name>

RESULT:

Thus the program ran a container from docker hub