Ex.No:1 Date :	Use gcc to compile c-programs. Split the programs to different modules and create an application using make command.

AIM

To use a gcc compiler for compiling C programs and creating an application using make command.

make - utility for building and maintaining groups of programs.

DESCRIPTION

1. **make** The purpose of the make utility is to determine automatically which pieces of a large program need to be recompiled, and issue the commands to recompile them.
2. You can use **make** with any programming language whose compiler can be run with a shell command.
3. In fact, make is not limited to programs. You can use it to describe any task where some files must be updated automatically from others whenever the others change.
4. To prepare to use make, you must write a file called the makefile that describes the relationships among files in your program, and the commands for updating each file.
5. In a program, typically the **executable file is updated from object files**, which are in turn made by compiling source files.

PROCEDURE

Procedure to execute make command:

1. Open text editor in Ubuntu.
2. Create files for any application (Note : You can use any programming language) for example here I have created 4 files to find factorial. Files include three cpp files and one header file.

Function1.cpp

```
# include "functions.h"
int factorial(int n)
```

```
{
        if(n!=1)
        {
                return(n*factorial(n-1));
        }
        else return 1;
}
```

Function2.cpp

```
# include<iostream>
# include "functions.h"
void print_hello()
{
std::cout<<"Hello World";
}
```

Main.cpp

```
# include <iostream>
# include "functions.h"
int main()
{
        print_hello();
        std::cout<<std::endl;
        std::cout<<"The factorial of 5 is"<<factorial(5)<<std::endl; return 0;
}
```

Function.h

```
void print_hello();
int factorial(int n);
```

3. Create Makefile : Syntax: target: (use tab space) prereq1 prereq2..

Open new text editor and do the following
Makefile:

```
#target: pre-req1 pre-req2...
```

all: hello //first target

hello: main.o function1.o function2.o

g++ main.o function1.o function2.o -o hello

main.o: main.cpp // main.o dependency to main.cpp g++ -c main.cpp //compiling main.cpp

function1.o: function1.cpp

g++ -c function1.cpp

function2.o: function2.cpp

g++ -c function2.cpp

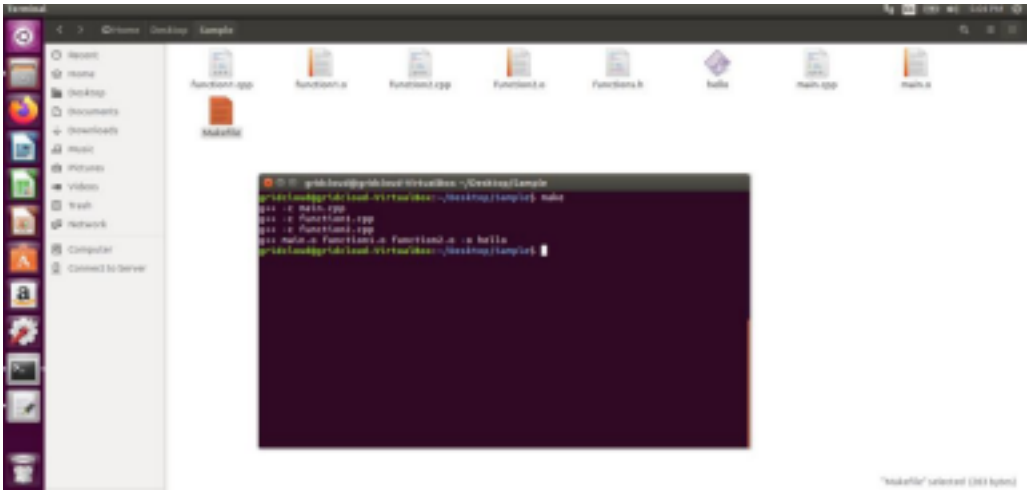
clean: // used to remove object and exe files rm -rf *o hello

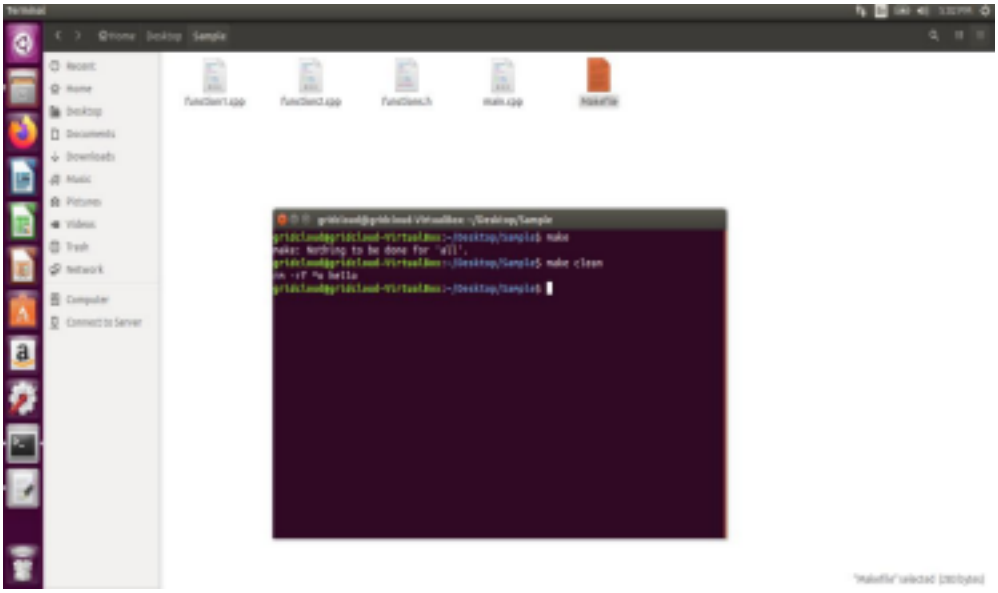
you can create an executable file for the above application (Factorial) desktop/sample/$ make all

or

desktop/sample/$ make all

OUTPUT





RESULT

Application with make commands has been created and executed successfully using gcc compiler.

Ex.No:2 Date :	Use version control systems command to clone, commit, push, fetch, pull, checkout, reset, and delete

AIM

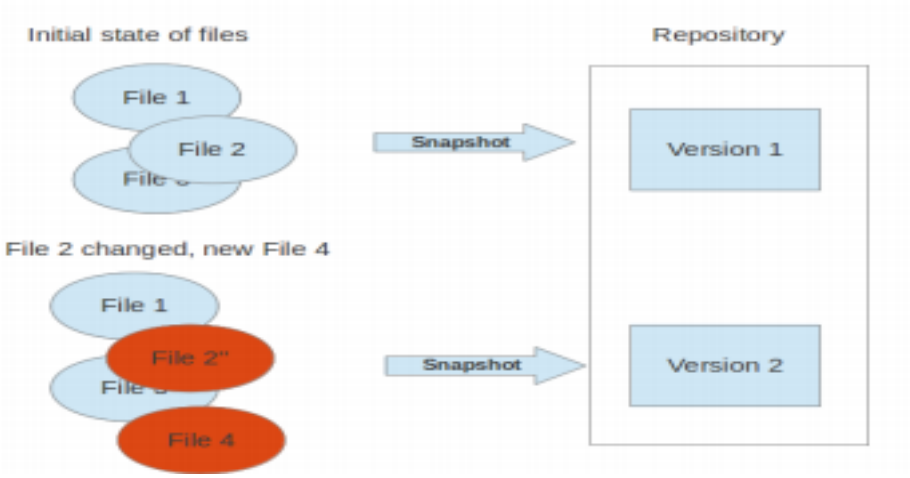
To use version control systems commands to clone, commit, push, fetch, pull, checkout, reset and delete repositories.

DESCRIPTION

What is a version control system?

A version control system (VCS) allows you to track the history of a collection of files. It supports creating different versions of this collection. Each version captures a snapshot of the files at a certain point in time and the VCS allows you to switch between these versions. These versions are stored in a specific place, typically called a *repository*.

You may, for example, revert the collection of files to a state from 2 days ago. Or you may switch between versions of your files for experimental features. The process of creating different versions (snapshots) in the repository is depicted in the following graphic. Please note that this picture fits primarily to Git. Other version control systems like *Concurrent Versions System* (CVS) don't create snapshots of the files but store file deltas.



What is Git?

Git is currently the most popular implementation of a distributed version control system.
Git originates from the Linux kernel development and was founded in 2005 by Linus Torvalds. Nowadays it is used by many popular open source projects, e.g., the Android or the Eclipse developer teams, as well as many commercial organizations.

The core of Git was originally written in the programming language *C*, but Git has also been re-implemented in other languages, e.g., Java, Ruby and Python.

PROCEDURE

2.1 Setting Up Git

You need to setup Git on your local machine, as follows:

1. Download & Install:
   · For Windows and Mac, download the installer from http://gitscm.com/downloads and run the downloaded installer.
   · For Ubuntu, issue command "sudo apt-get install git".

For Windows, use the "Git Bash" command shell bundled with Git Installer to issue commands. For Mac/Ubuntu, use the "Terminal".

2. Customize Git:

Issue "git config" command (for Windows, run "Git Bash" from the Git installed directory. For Ubuntu/Mac, launch a "Terminal"):

```
// Set up your username and email (to be used in labeling your commits) $ git confi
--global user.name "umamageswaran"
$ git config --global user.email "j.umamageswaran@gmail.com"
```

The settings are kept in "<GIT_HOME>/etc/gitconfig" (of the GIT installed directory) and "<USER_HOME>/.gitconfig" (of the user's home directory.

You can issue "git config --list" to list the settings:

```
$ git config --list
user.email= j.umamageswaran@gmail.com
user.name=umamageswaran
```

Git Basics

Git Commands

Git provides a set of simple, distinct, standalone commands developed according to the "Unix toolkit" philosophy - build small, interoperable tools.

To issue a command, start a "Terminal" (for Ubuntu/Mac) or "Git Bash" (for Windows):

```
$ git <command> <arguments>
```

The commonly-used commands are:

1. **init**, **clone**, **config**: for starting a Git-managed project.
2. **add**, **mv**, **rm**: for staging file changes.
3. **commit**, **rebase**, **reset**, **tag**:
4. **status**, **log**, **diff**, **grep**, **show**: show status
5. **checkout**, **branch**, **merge**, **push**, **fetch**, **pull**

Help and Manual

```
$ git help <command>
// or
$ git <command> --help
```

Getting Started with Local Repo

There are 2 ways to start a Git-managed project:

1. Starting your own project;
2. Cloning an existing project from a GIT host.

Setup the Working Directory for a New Project

Let's start a programming project under the *working directory* called "hello-git", with one source file "Hello.java" (or "Hello.cpp", or "Hello.c") as follows:

```
// Hello.java
public class Hello {
 public static void main(String[] args) {
 System.out.println("Hello, world from GIT!");
 }
```

```
}
```

Compile the "Hello.java" into "Hello.class" (or "Hello.cpp" or "Hello.c" into "Hello.exe").

Command to compile : javac Hello.java

Now, we have 2 files in the *working tree*: "Hello.java", "Hello.class" . We do not wish to track the ".class" as they can be reproduced from ".java".

Initialize a new Git Repo (git init)

To manage a project under Git, run "git init" at the project *root* directory

```
// Initialize Git repo for this project
$ git init
Initialized empty Git repository in /path-to/hello-git/.git/

$ ls -al
drwxr-xr-x 1 xxxxx xxxxx 4096 Sep 14 14:58 .git
-rw-r--r-- 1 xxxxx xxxxx 426 Sep 14 14:40 Hello.class
-rw-r--r-- 1 xxxxx xxxxx 142 Sep 14 14:32 Hello.java
```

Staging File Changes for Tracking (git add <file>...)

```
$ git status
On branch master
Initial commit

Untracked files:
 (use "git add <file>..." to include in what will be committed)
 Hello.class
 Hello.java
 nothing added to commit but untracked files present (use "git add" to track)
```

In Git, the files in the working tree are either *untracked* or *tracked*. Currently, all 2 files are *untracked*. To stage a new file for tracking, use "git add <file>..." command.

```
//Add all Java source files into Git repo
$ git add *.java

// You can also include multiple files in the "git add"
// E.g.,
// git add Hello.java README.md

$ git status
On branch master
Initial commit

Changes to be committed:
 (use "git rm --cached <file>..." to unstage)
 new file: Hello.java
 new file: README.md

Untracked files:
 (use "git add <file>..." to include in what will be committed)
 Hello.class
```

The command "git add <file>..." takes one or more filenames or pathnames with possibly wildcards pattern. You can also use "git add ." to add all the files in the current directory (and all sub-directories). But this will include "Hello.class", which we do not wish to be tracked. When a new file is added, it is *staged* (or *index*ed, or *cached*) in the *staging area* (as shown in the GIT storage model), but NOT yet *committed*.

Git uses two stages to commit file changes:

1. "git add <file>" to stage file changes into the *staging area*, and
2. "git commit" to commit ALL the file changes in the *staging area* to the *local repo*. The staging area allows you to group related file changes and commit them together. ***Committing File Changes (git commit)***

The "git commit" command commits ALL the file changes in the *staging area*. Use a - m option to provide a *message* for the commit.

```
$ git commit -m "First commit" // -m to specify the commit message
[master (root-commit) 858f3e7] first commit
 1 files changed, 5 insertions(+)
 create mode 100644 Hello.java
create mode 100644 Hello.class

// Check the status
$ git status
On branch master
nothing added to commit but untracked files present (use "git add" to track)
```

2.2 Commit Command

Viewing the Commit Data (git log)

Git records several pieces of metadata for every commit, which includes a log message, timestamp, the author's username and email (set during customization).

You can use "git log" to list the commit data; or "git log --stat" to view the file statistics:

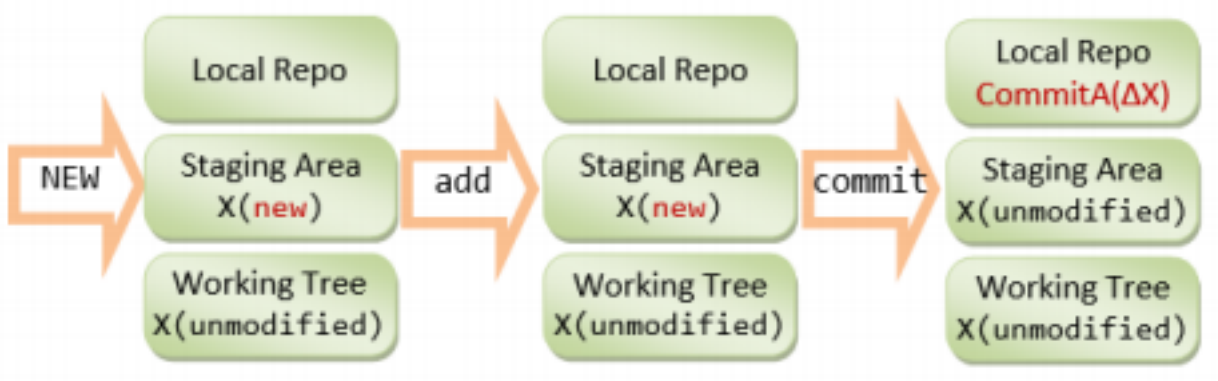
```
$ git log
commit 858f3e71b95271ea320d45b69f44dc55cf1ff794
Author: Umamageswaran <j.Umamageswaran@gmail.com>
Date: Fri Aug 14 18:58:07 2020 +0530
 First commit

$ git log --stat
commit 858f3e71b95271ea320d45b69f44dc55cf1ff794
Author: Umamageswaran <j.Umamageswaran@gmail.com>
Date: Fri Aug 14 18:58:07 2020 +0530
 First commit
 Hello.java | 5 ++++++
 Hello.class | Bin 0 -> 426 bytes
 2 files changed, 5 insertions(+)
```

File Status (git status)

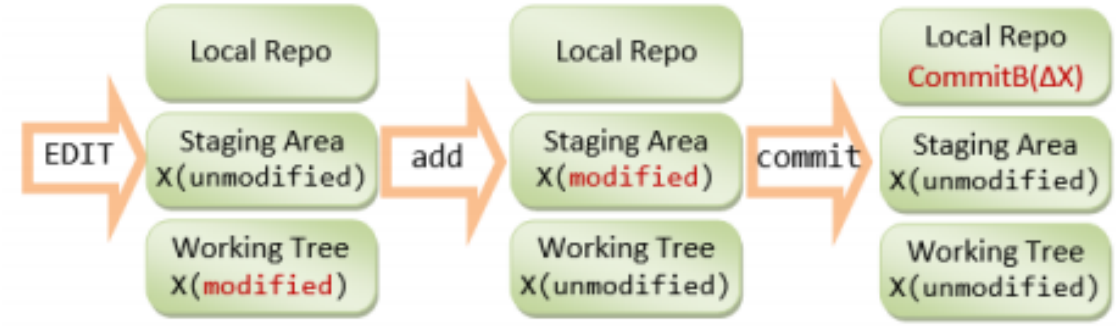
A file could be *untracked* or *tracked*.

As mentioned, Git tracks file changes at commits. In Git, changes for a *tracked* file could be: 1. *unstaged* (in *Working Tree*) - called *unstaged changes*, *staged* (in *Staging Area* or *Index* or *Cache*) - called *staged changes*, or *committed* (in *local repo object database*).The files in "working tree" or "staging area" could have status of *unmodified*, *added*, *modified*, *deleted*, *renamed*, *copied*, as reported by "git status".

2. The "git status" output is divided into 3 sections: "Changes not staged for commit" for the unstaged changes in "working tree", "Changes to be committed" for the staged changes in the "staging area", and "Untracked files". In each section, It lists all the files that have been changed, i,e., files having status other than *unmodified*.
3. When a new file is created in the working tree, it is marked as *new* in working tree and shown as an untracked file. When the file change is staged, it is marked as *new* (*added*) in the staging area, and *unmodified* in working tree. When the file change is committed, it is marked as *unmodified* in both the working tree and staging area.



When a committed file is modified, it is marked as *modified* in the working tree and *unmodified* in the staging area. When the file change is staged, it is marked as *modified* in the staging area and *unmodified* in the working tree. When the file change is committed, it is marked as *unmodified* in both the working tree and staging area.



For example, made some changes to the file "Hello.java", and check the status again:

```
// Hello.java
public class Hello {
 public static void main(String[] args) {
```

```
 System.out.println("Hello, world from GIT!");
 System.out.println("Changes after First commit!");
 }
}
```

```
$ git status
On branch master
Changes not staged for commit:
 (use "git add <file>..." to update what will be committed)
 (use "git checkout -- <file>..." to discard changes in working directory)
 modified: Hello.java
 modified: Hello.class

no changes added to commit (use "git add" and/or "git commit -a")
```

The "Hello.java" is marked as *modified* in the working tree (under "Changes not staged for commit"), but *unmodified* in the staging area (not shown in "Changes to be committed"). You can inspect all the unstaged changes using "git diff" command (or "git diff <file>" for the specified file). It shows the file changes in the working tree since the last commit:

```
$ git diff
diff --git a/Hello.java b/Hello.java
index dc8d4cf..f4a4393 100644
--- a/Hello.java
+++ b/Hello.java
@@ -2,5 +2,6 @@
 public class Hello {
 public static void main(String[] args) {
 System.out.println("Hello, world from GIT!");
+ System.out.println("Changes after First commit!");
 }
 }
```

The older version (as of last commit) is marked as --- and new one as +++. Each chunk of changes is delimited by "@@ -<old-line-number>,<number-of-lines> +<new-line number>,<number-of-lines> @@". Added lines are marked as + and deleted as -. In the above output, older version (as of last commit) from line 2 for 5 lines and the modified version from line 2 for 6 lines are compared. One line (marked as +) is added. Stage the

changes of "Hello.java" by issuing the "git add <file>...":

```
$ git add Hello.java
$ git add Hello.class

$ git status
On branch master
Changes to be committed:
 (use "git reset HEAD <file>..." to unstage)
 modified: Hello.java
 modified: Hello.class
```

Now, it is marked as *modified* in the staging area ("Changes to be committed"), but *unmodified* in the working tree (not shown in "Changes not staged for commit"). Now, the changes have been staged. Issuing an "git diff" to show the unstaged changes results in empty output.

You can inspect the staged change (in the staging area) via "git diff --staged" command:

```
// List all "unstaged" changes for all files (in the working tree)
$ git diff
 // empty output - no unstaged change

// List all "staged" changes for all files (in the staging area)
$ git diff --staged
diff --git a/Hello.java b/Hello.java
index dc8d4cf..f4a4393 100644
--- a/Hello.java
+++ b/Hello.java
@@ -2,5 +2,6 @@
 public class Hello {
```

```
 public static void main(String[] args) {
 System.out.println("Hello, world from GIT!");
+ System.out.println("Changes after First commit!");
 }
 }
 // The "unstaged" changes are now "staged".
```

Commit ALL staged file changes via "git commit":

```
$ git commit -m "Second commit"
[master 96efc96] Second commit
 2 file changed, 2 insertion(+)
Rewrite Hello.class (96%)

$ git status
On branch master
nothing added to commit but untracked files present (use "git add" to track)
```

2.3 Setting up Remote Repo

1. Sign up for a GIT host, such as Github https://github.com/signup/free (Unlimited for public projects; fee for private projects);
2. Login to the GIT host. Create a new remote repo called "test".
3. On your local repo (let's continue to work on our "hello-git" project), set up the remote repo's *name* and *URL* via "git remote add <remote-name> <remote-url>" command.

   By convention, we shall name our remote repo as "origin". You can find the URL of a remote repo from the Git host. The URL may take the form of HTTPS or SSH. Use HTTPS for simplicity.

   https://github.com/Umamageswaran/test1.git

```
// Add a remote repo called "origin" via "git remote add <remote-name> <remote-url>" // Fo
examples,
$ git remote add origin https://github.com/your-username/test1.git // for GitHub
```

You can list all the remote names and their corresponding URLs via "git remote -v", for example,

```
$ git remote -v
origin https://github.com/Umamageswaran /test1.git (fetch)
origin https://github.com/ Umamageswaran /test1.git (push)
```

4. Push the commits from the local repo to the remote repo via "git push -u <remote name> <local-branch-name>".

By convention, the main branch of our local repo is called "master" (as seen from the earlier "git status" output). We shall discuss "branch" later.

```
// Push all commits of the branch "master" to remote repo "origin" $ git pus
origin master
Username for 'https://github.com': j.umamageswaran@gmail.com Password fo
'https://your-username@github.com': *******
Counting objects: 8, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 1.13 KiB | 0 bytes/s, done.
Total 10 (delta 1), reused 0 (delta 0)
```

```
To https://github.com/your-username/test.git
 * [new branch] master -> master
Branch master set up to track remote branch master from origin.
```

5. Login to the GIT host and select the remote repo "test", you shall find all the committed

files.

6. On your local system, make some change (e.g., on "Hello.java"); stage and commit the changes on the local repo; and push it to the remote. This is known as the "Edit/Stage/Commit/Push" cycle

```
// Hello.java
public class Hello {
 public static void main(String[] args) {
 System.out.println("Hello, world from GIT!");
 System.out.println("Changes after First commit!");
 System.out.println("Changes after Pushing to remote!");
 }
}
```

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
```

```
// Stage file changes
$ git add *.java

$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
 (use "git reset HEAD <file>..." to unstage)
```

```
 modified: Hello.java

// Commit all staged file changes
$ git commit -m "Third commit"
[master 744307e] Third commit
 1 file changed, 1 insertion(+)

// Push the commits on local master branch to remote
$ git push origin master
Username for 'https://github.com': ******
Password for 'https://username@github.com': ******
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 377 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://github.com/your-username/test.git
 711ef4f..744307e master -> master
```

Again, login to the remote to check the committed files.

2.4 Cloning a Project from a Remote Repo

(git clone <remote-url>)

As mentioned earlier, you can start a local GIT repo either running "git init" on your own project, or "git clone <remote-url>" to copy from an existing project.

```
SYNTAX
git clone <remote-url>
```

// <url>: can be https (recommended), ssh or file.

// Clone the project UNDER the current directory

// The name of the "working directory" is the same as the remote project name

git clone <remote-url> <working-directory-name>

// Clone UNDER current directory, use the given "working directory" name

$ **git clone https://github.com/*Umamageswaran*/test1.git hello-git-cloned** Cloning int 'hello-git-cloned'...

remote: Counting objects: 13, done.

remote: Compressing objects: 100% (11/11), done.

remote: Total 13 (delta 2), reused 13 (delta 2)

Unpacking objects: 100% (13/13), done.

Checking connectivity... done.

// Verify

$ **cd hello-git-cloned**

$ **ls -a**

. .. .git .gitignore Hello.java README.md

$ **git status**

On branch master

Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean

The "git clone" automatically creates a remote name called "origin" mapped to the cloned remote-URL. You can check via "git remote -v":

```
// List all the remote names
```

```
$ git remote -v
origin https://github.com/your-username/test.git (fetch)
origin https://github.com/your-username/test.git (push)
```

Summary of Basic "Edit/Stage/Commit/Push" Cycle

```
// Edit (Create, Modified, Rename, Delete) files,
// which produces "unstaged" file changes.

// Stage file changes, which produces "Staged" file changes
$ git add <file> // for new and modified files
$ git rm <file> // for deleted files
$ git mv <old-file-name> <new-file-name> // for renamed file

// Commit (ALL staged file changes)
$ git commit -m "message"

// Push
$ git push <remote-name> <local-branch-name>
```

OR,

```
// Stage ALL files with changes
$ git add -A // OR, 'git add --all'

$ git commit -m "message"
$ git push
```

OR,

```
// Add All and Commit in one command
$ git commit -a -m "message"

$ git push
```

2.5 Check OUT

Switching to a Branch (git checkout <branch-name>)

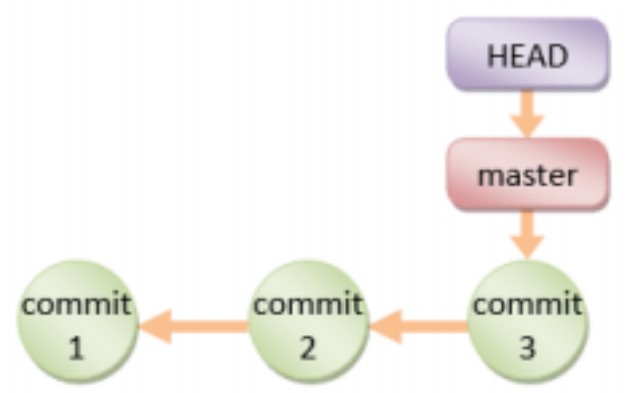
Branching allows you and your team members to work on different aspects of the software *concurrently* (on so-called feature branches), and merge into the master branch as and when they completes. Branching is the most important feature in a *concurrent* version control system.

A branch in Git is a lightweight movable pointer to one of the commits. For the initial commit, Git assigns the default branch name called master and sets the master branch pointer at the initial commit. As you make further commits on the master branch, the master branch pointer move forward accordingly. Git also uses a special pointer called HEAD to keep track of the branch that you are currently working on. The HEAD always refers to the latest commit on the current branch. Whenever you switch branch, the HEAD also switches to the latest commit on the branch switched.

Example

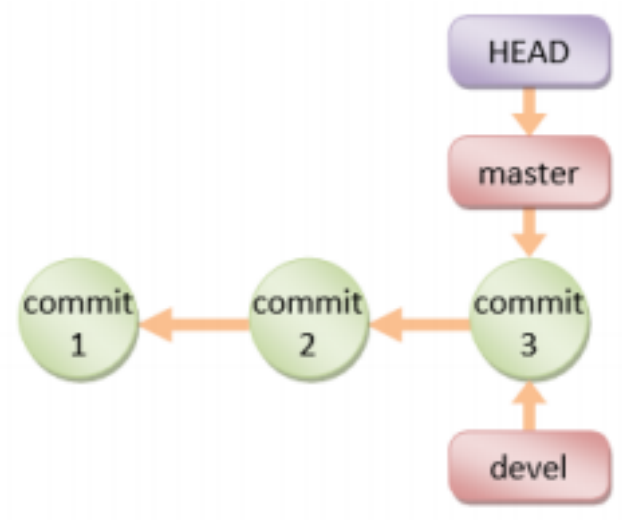
For example, let's create a Git-managed project called git_branch_test with only the a single line README.md file:

```
$ git init
$ git add README.md
$ git commit -m "Commit 1"

// Append a line in README.md: This line is added after Commit 1
$ git status
$ git add README.md
$ git commit -m "Commit 2"

// Append a line in README.md: This line is added after Commit 2
$ git status
$ git add README.md
$ git commit -m "Commit 3"

// Show all the commits (oneline each)
$ git log --oneline
44fdf4c Commit 3
51f6827 Commit 2
fbed70e Commit 1
```



Creating a new Branch (git branch <branch-name>)

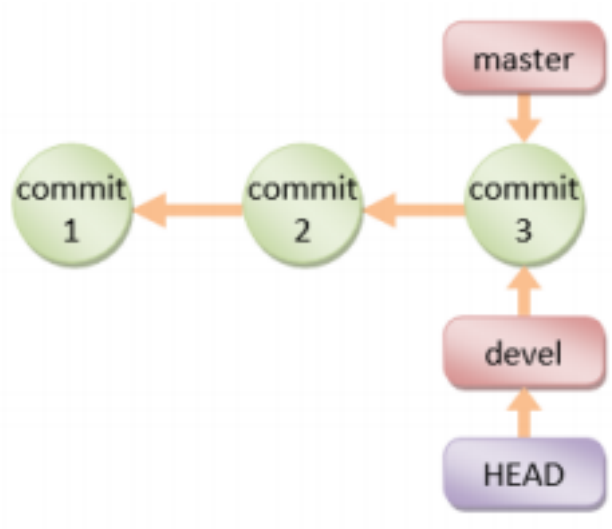
```
$ git branch devel
```



Switching to a Branch (git checkout <branch-name>)

Git uses a special pointer called HEAD to keep track of the branch that you are working on. The "git branch <branch-name>" command simply create a branch, but does not switch to the new branch. To switch to a branch, use "git checkout <branch-name>" command. The HEAD pointer will be pointing at the switched branch (e.g., devel).

```
$ git checkout devel
Switched to branch 'devel'
```

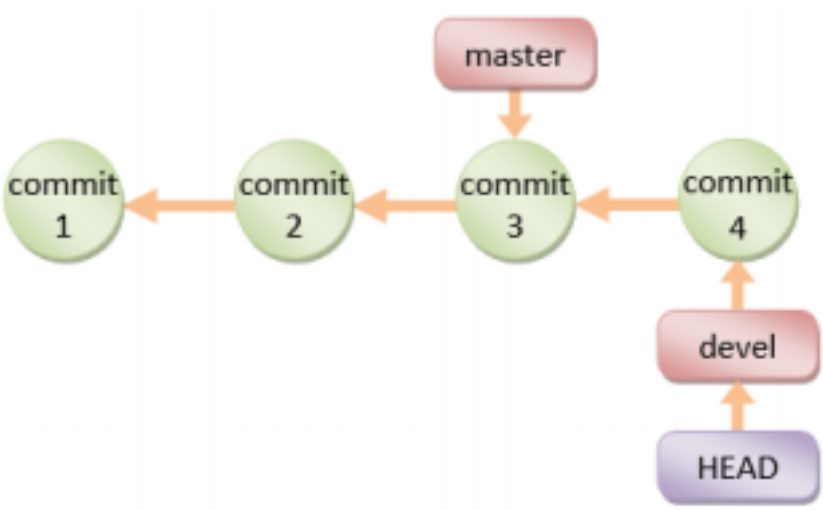


Alternatively, you can use "git checkout -b <branch-name>" to create a new branch and switch into the new branch.

If you switch to a branch and make changes and commit. The HEAD pointer moves forward

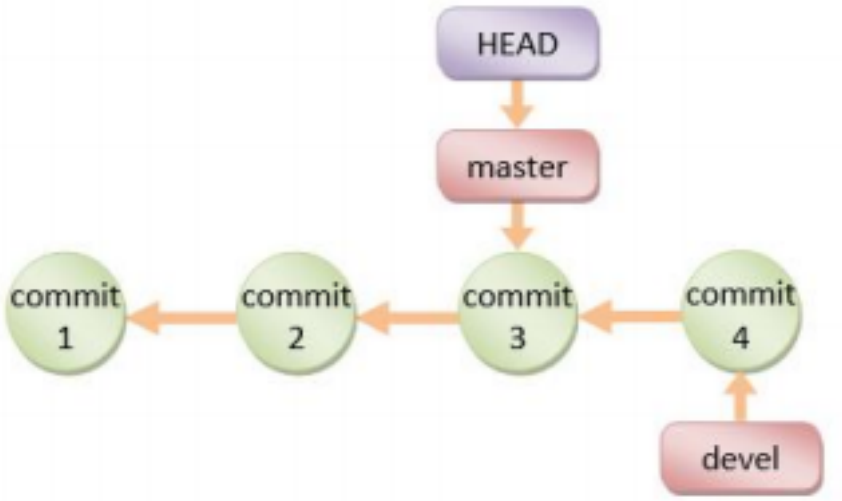
in that branch.

```
// Append a line in README.md: This line is added on devel branch after Commit 3 $ gi
status // NOTE "On branch devel"
$ git add README.md
$ git commit -m "Commit 4"
[devel c9b88d9] Commit 4
```



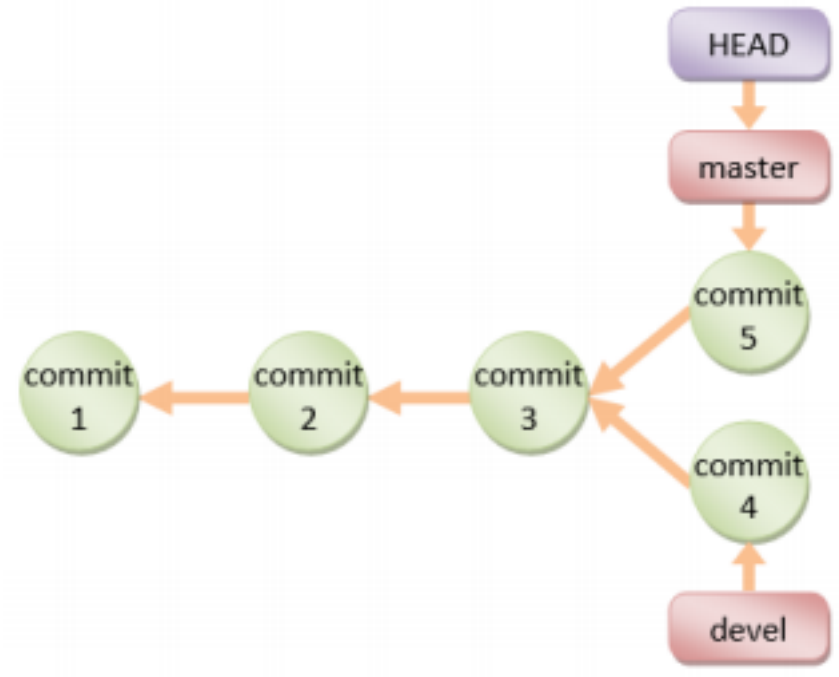
ou can switch back to the master branch via "git checkout master". The HEAD pointer moves back to the last commit of the master branch, and the working directory is *rewinded* back to the latest commit on the master branch.

```
$ git checkout master
Switched to branch 'master'
// Check the content of the README.md, which is reminded back to Commit 3
```



f you continue to work on the master branch and commit, the HEAD pointer moves forward on the master branch. The two branches now *diverge*.

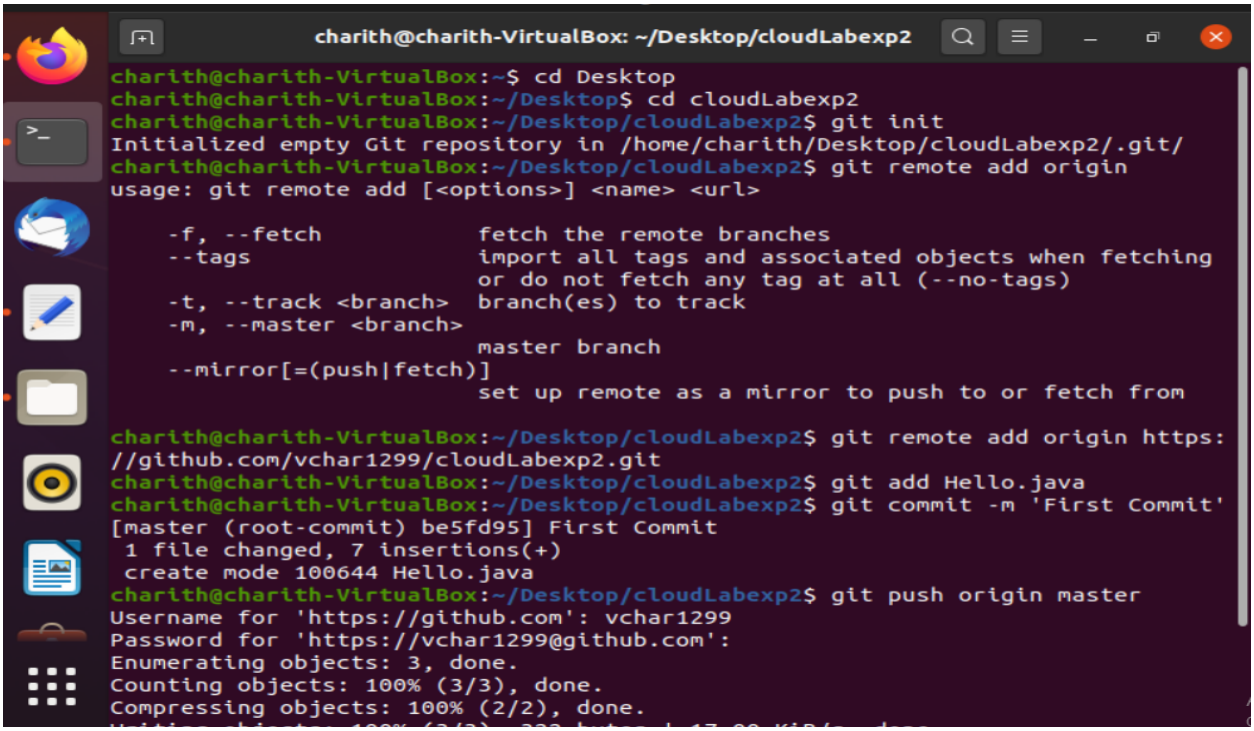
```
// Append a line in README.md: This line is added on master branch after Commit 4 $ gi
status // NOTE "On branch master"
$ git add README.md
$ git commit -m "Commit 5"
[master 6464eb8] Commit 5
```



If you check out the devel branch, the file contents will be rewinded back to Commit-4.

```
$ git checkout devel
// Check file contents
```

OUTPUT



RESULT

Version control system commands has been executed successfully.

Ex.No:3 Date :	Install VirtualBox workstation with different flavours of linux or windows on top of windows.

AIM

To install VirtualBox on top of windows OS.

PROCEDURE

1. Open the VirtualBox website. Go to https://www.virtualbox.org/ in your computer's Internet browser. This is the website from which you'll download the VirtualBox setup file.
2. Click Download VirtualBox button**.** It's a blue button in the middle of the page. Doing so will open the downloads page.
3. Click Windows hosts. You'll see this link below the "VirtualBox 6.1.14 platform packages" heading. The VirtualBox EXE file will begin downloading onto your computer.
4. Open the VirtualBox EXE file. Go to the location to which the EXE file downloaded and double-click the file. Doing so will open the VirtualBox installation window.

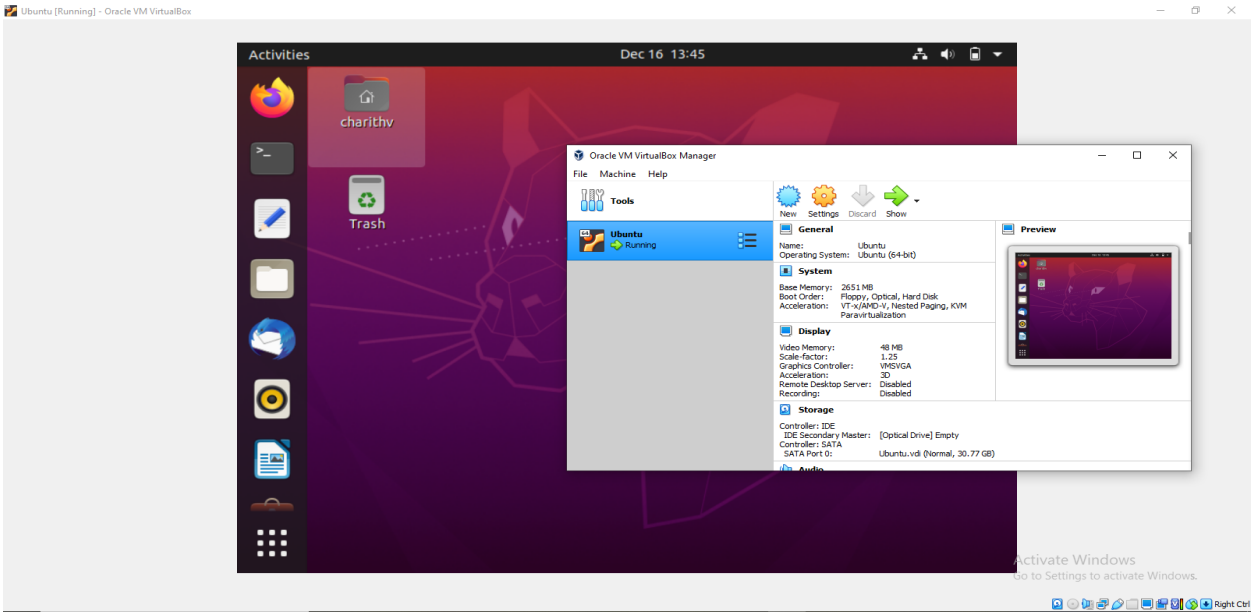
5. Navigate through the installation prompts. Do the following:
    - Click Next on the first three pages.
    - Click Yes when prompted.
    - Click Install
    - Click Yes when prompted.
6. Click Install when prompted. Doing so will allow VirtualBox to begin installing on your computer.

7. Click Finish when prompted. It's in the lower-right side of the window. Doing so will close the installation window and open VirtualBox. Now that you've installed and opened VirtualBox, you can create a virtual machine in order to run any operating system on your PC.

- Make sure that you don't uncheck the "Start" box before doing this.

OUTPUT

Ubuntu	8/24/2020 8:42 AM	File folder	
VM	8/17/2020 10:12 PM	File folder	
dataiku-dss-8.0.0.tar	8/20/2020 3:37 PM	WinRAR archive	646,447 KB
eula.1028	11/7/2007 8:00 AM	Text Document	18 KB
eula.1031	11/7/2007 8:00 AM	Text Document	18 KB
eula.1033	11/7/2007 8:00 AM	Text Document	10 KB
eula.1036	11/7/2007 8:00 AM	Text Document	18 KB
eula.1040	11/7/2007 8:00 AM	Text Document	18 KB
eula.1041	11/7/2007 8:00 AM	Text Document	1 KB
eula.1042	11/7/2007 8:00 AM	Text Document	18 KB
eula.2052	11/7/2007 8:00 AM	Text Document	18 KB
eula.3082	11/7/2007 8:00 AM	Text Document	18 KB
globdata	11/7/2007 8:00 AM	Configuration sett...	2 KB
install	11/7/2007 8:03 AM	Application	550 KB
install	11/7/2007 8:00 AM	Configuration sett...	1 KB
install.res.1028.dll	11/7/2007 8:03 AM	Application exten...	75 KB
install.res.1031.dll	11/7/2007 8:03 AM	Application exten...	95 KB
install.res.1033.dll	11/7/2007 8:03 AM	Application exten...	90 KB
install.res.1036.dll	11/7/2007 8:03 AM	Application exten...	96 KB
install.res.1040.dll	11/7/2007 8:03 AM	Application exten...	94 KB
install.res.1041.dll	11/7/2007 8:03 AM	Application exten...	80 KB
install.res.1042.dll	11/7/2007 8:03 AM	Application exten...	79 KB
install.res.2052.dll	11/7/2007 8:03 AM	Application exten...	75 KB
install.res.3082.dll	11/7/2007 8:03 AM	Application exten...	95 KB
ubuntu-20.04.1-desktop-amd64	8/18/2020 12:51 AM	Disc Image File	2,719,744 KB



RESULT

Virtual box has been installed and a linux OS was run successfully.

Ex.No:4 Date :	Compile and run atleast 3 C programs.

AIM

To compile and run atleast 3 C programs

PROCEDURE

1. Open text editor in virtual box.
2. Type the code.
3. Save the file with a .C extension.
4. Compile the program using gcc filename and then see the output using ./a.out

PROGRAM

Hello.c

```
#include<stdio.h>
int main()
{
  printf(“Hello world”):
}
```

Palindrome.c

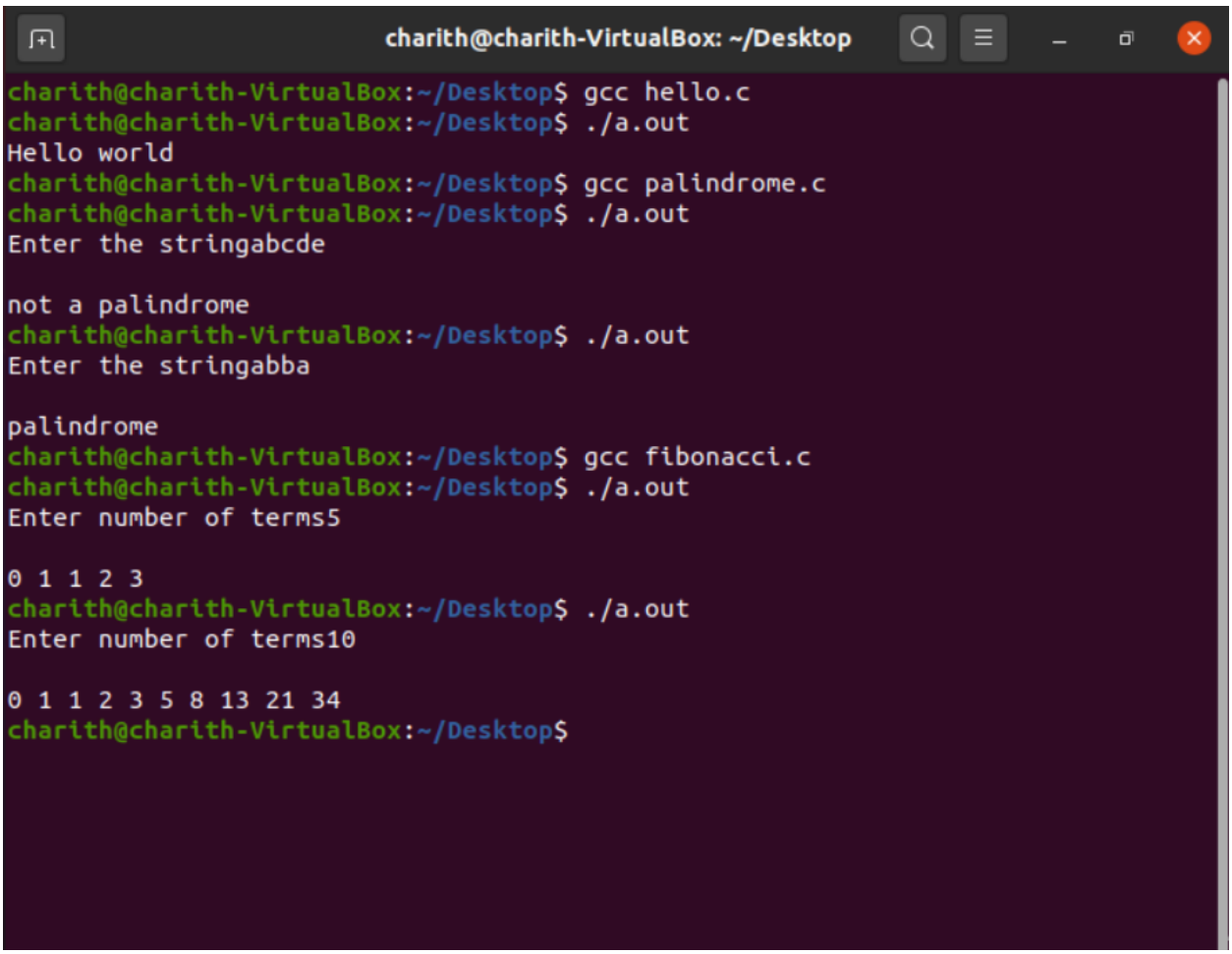
```
#include<stdio.h>
int main()
{
 int n,r,sum=0,temp;
 printf("enter the number=");
 scanf("%d",&n);
 temp=n;
 while(n>0)
 {
  r=n%10;
```

```
   sum=(sum*10)+r;
   n=n/10;
  }
  if(temp==sum)
    printf("palindrome number ");
  else
    printf("not palindrome");
  return 0;
}
```

Fibonacci.c

```
#include<stdio.h>
int main()
{
  int n1=0,n2=1,n3,i,number;
  printf("Enter the number of elements:");
  scanf("%d",&number);
  printf("\n%d %d",n1,n2);
  for(i=2;i<number;++i)
  {
    n3=n1+n2;
    printf(" %d",n3);
    n1=n2;
    n2=n3;
  }
}
```

OUTPUT



RESULT

Hello world, Fibonacci and Palindrome C programs have been executed in a virtual box with linux OS successfully.

Ex.No:5 Date :	Install Google App Engine. Create a hello world app and other simple web applications using python/java.

AIM

To install google app engine and create Hello world app using Python.

PROCEDURE

1. Install notepad++ editor or any other editor to write a program.
2. Download latest version of Python for windows and Install it from https://www.python.org/downloads/
3. Download and install Google App Engine from https://cloud.google.com/appengine/downloads?csw=1 and Download the Cloud SDK installer. For windows.
4. Write a program for hello world in notepad++ editor as follows

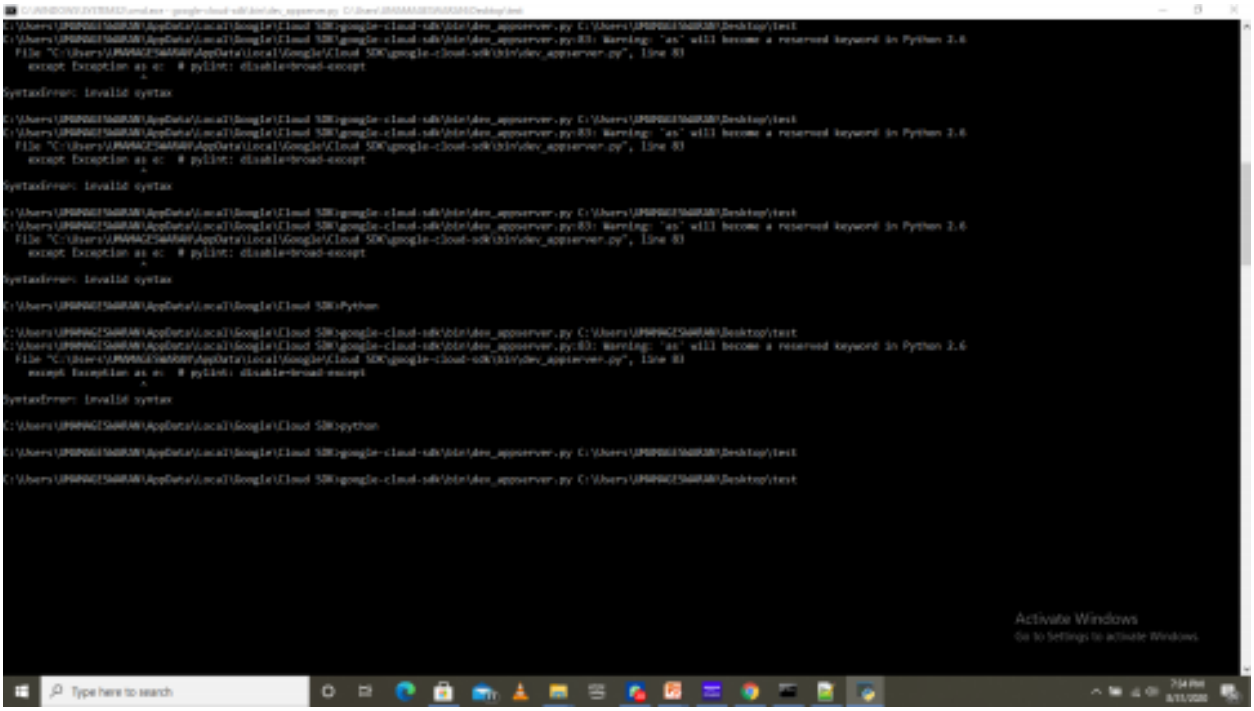
   Program 1: test.py

   ```
   class MainPage(webapp2.RequestHandler):
           def get(self):
                   self.response.write("Hello World")
   app=webapp2.WSGIApplication([('/', MainPage), ], debug=True)
   ```

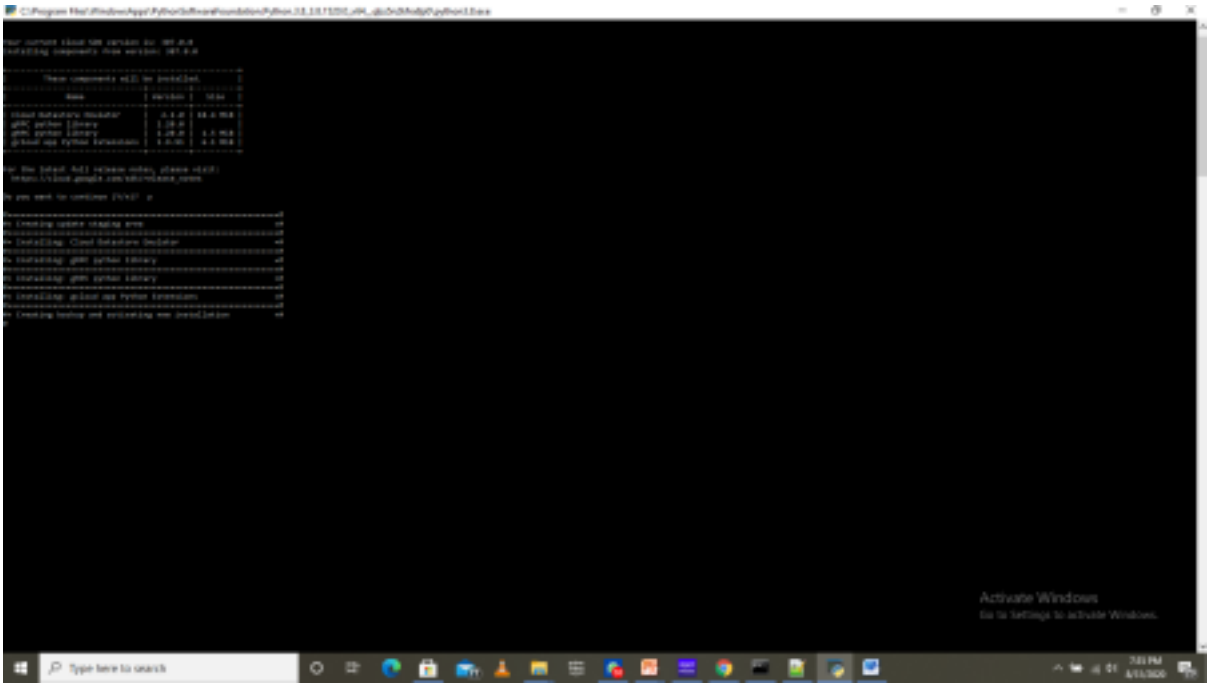
   Program 2. app.yaml

   ```
   runtime: python38
   api_version: 1
   threadsafe: true
   handler:
   - url: /
   script: test.app
   ```

5. Open google cloud SDK and set the path as c:\User\UMAMAGESWARAN\Appdata\Local\Google\Cloud SDK> google-cloud sdk\bin\dev_appserver.py
   and drag the folder that contains Hello world app. The screen looks like below. Then press Enter.



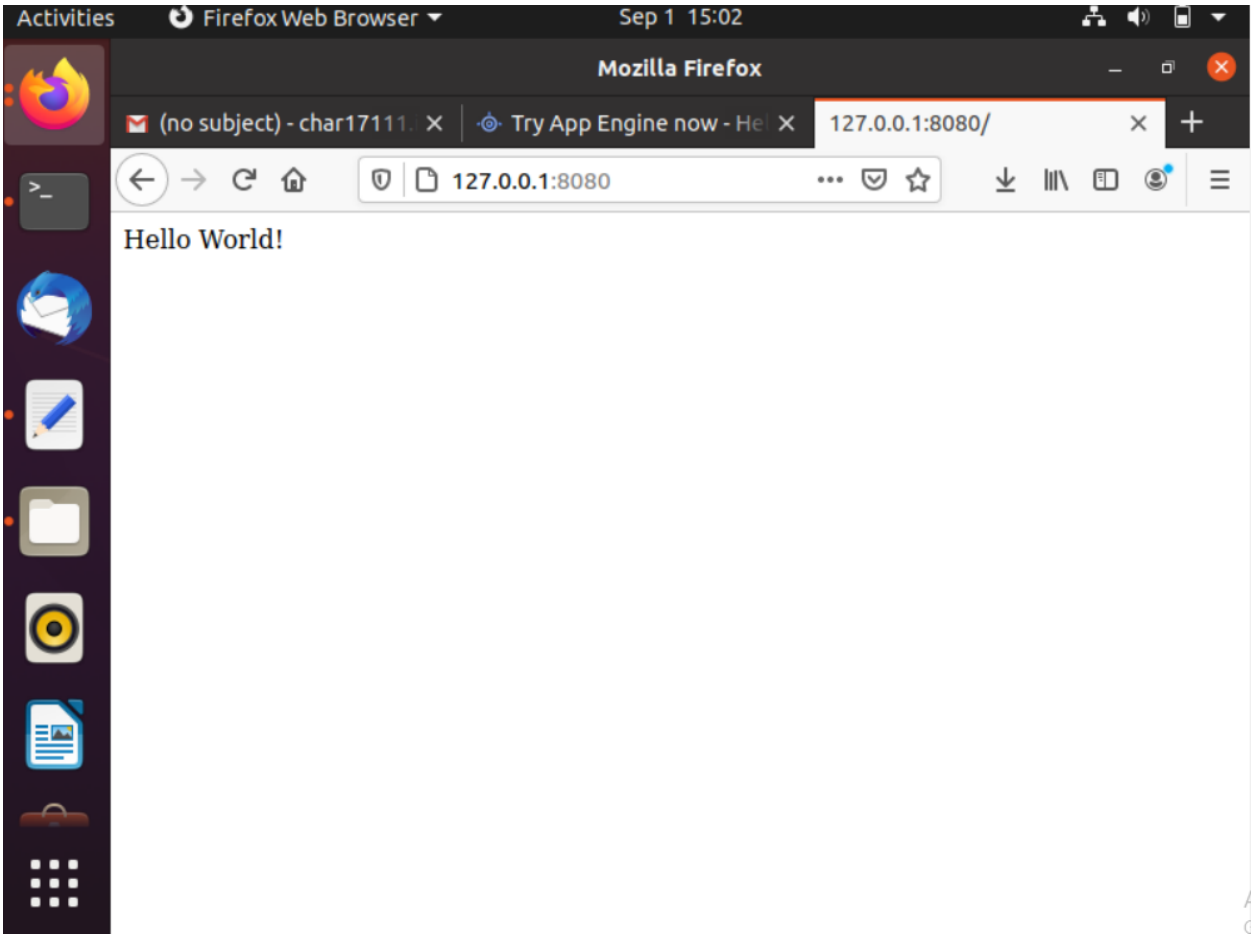
6. A new window will get open and if you are using Python for the first time make sure that you are selecting “Yes” and the press enter .



7. You can see the Server is running at the local host like : http://localhost:8080

8. Open a web browser and type in the address bar as : http://localhost:8080

OUTPUT



RESULT

Google app engine has been installed and a python program was executed successfully

Ex.No:6 Date :	Use GAE launcher to launch web applications.

AIM

To use a Google App Engine launcher to launch web applications.

PROCEDURE

Step 1. Download the basic housekeeping stuff

1. Download Python 2.7

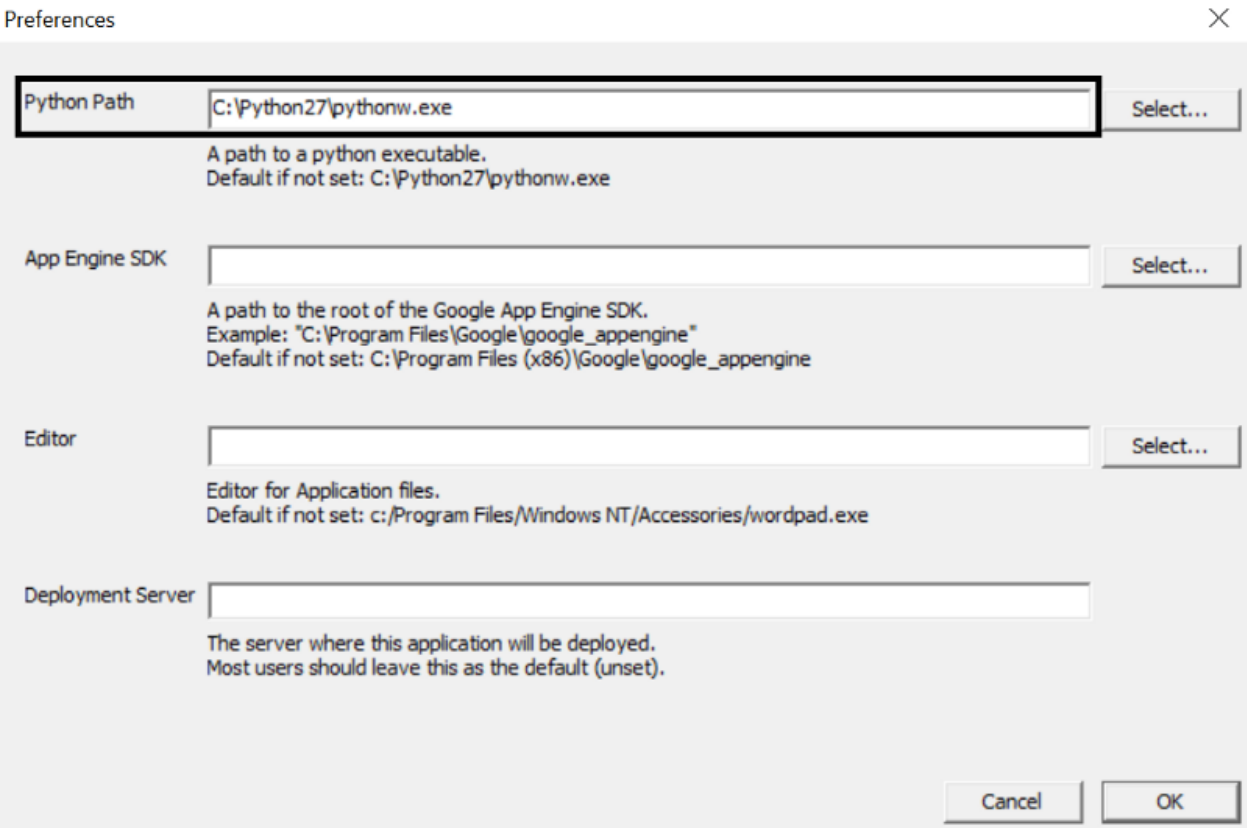
   As of when this article was written, the Google App Engine standard environment supports Python only upto version 2.7. However, it is only a matter of time before support for Python 3.x is added. You can check the App Engine docs for the latest info.

2. Download Google Cloud SDK

   This will allow you to fork apps onto your local machine, make changes (edit and develop the app), and deploy your app back to the cloud.

3. Set the Python path in the Google App Engine launcher

   After downloading the SDK, launch the App Engine launcher, go to Edit -> Preferences and make sure you set the path for where you installed Python in step 1 above.



Set the Python path in Google App Engine launcher

Step 2. App Engine sign-up

Things to should know when you sign-up:

1. Currently, App Engine offers a free trial for one year.
2. The trial includes $300 of credit that can be used during the one year trial period.
3. You will need to add a credit card to sign-up (for verification purposes).
4. You will not be charged during the sign-up process.
5. You will not be charged during the trial period as long as you do not cross the credit limit offered.

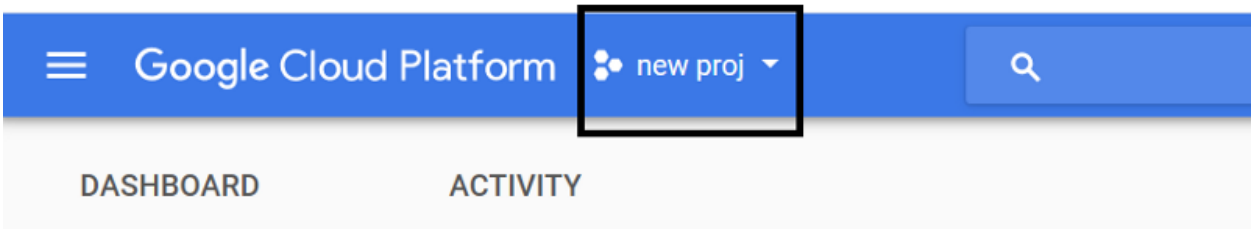
Here are the steps you need to follow to sign-up:

1. Go to the Google Cloud landing page
2. Follow the sign-up process and go to your App Engine dashboard.

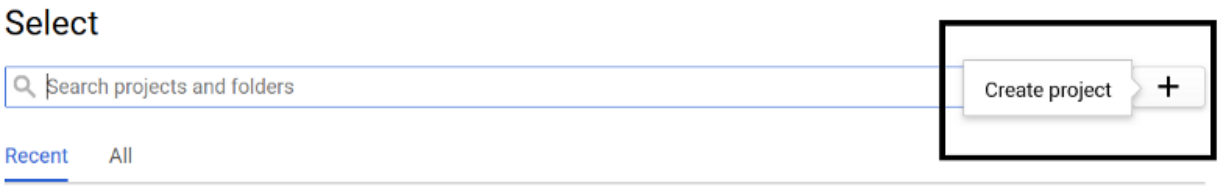
Step 3. Create a new project

The next step is to create a new Python project that you can work on.

Launch the new project wizard.



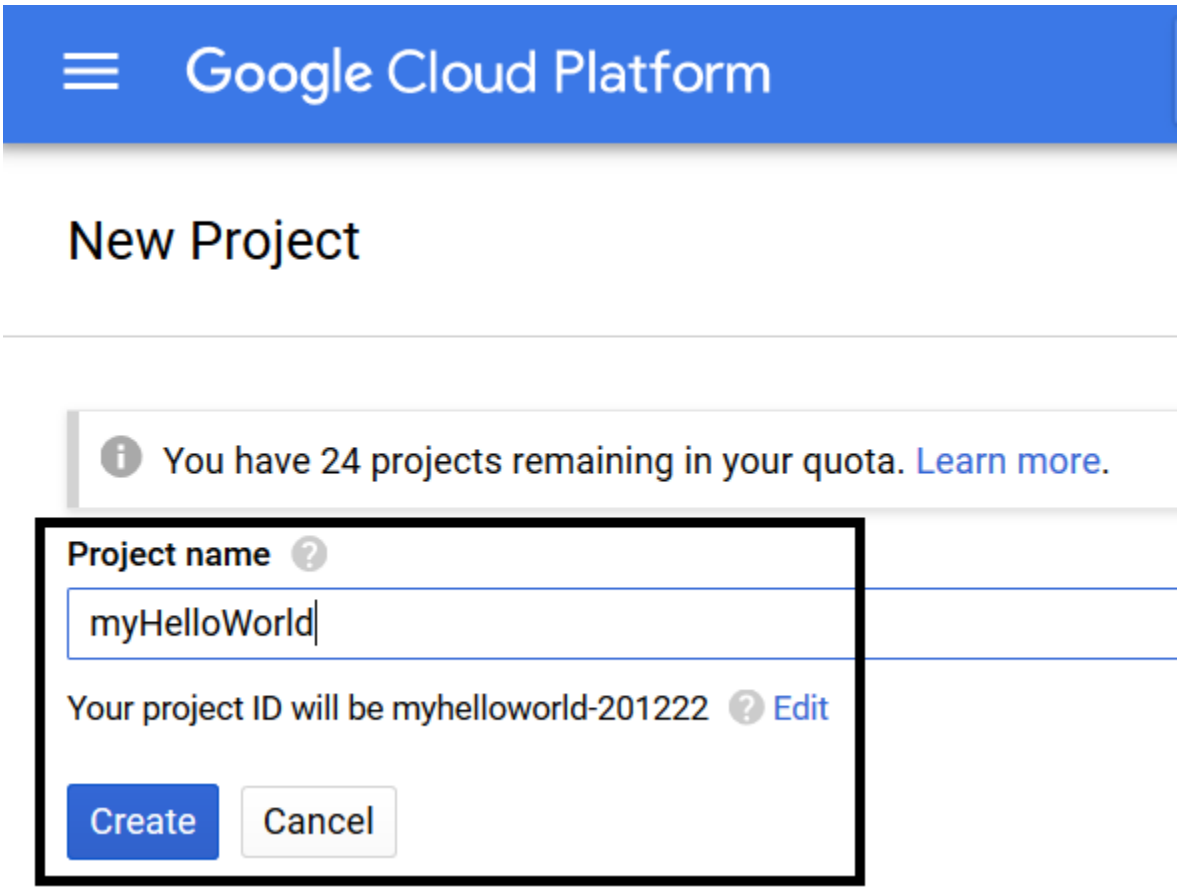
https://console.cloud.google.com/home



---

https://console.cloud.google.com/home

Give your app a name and make a note of your project ID.



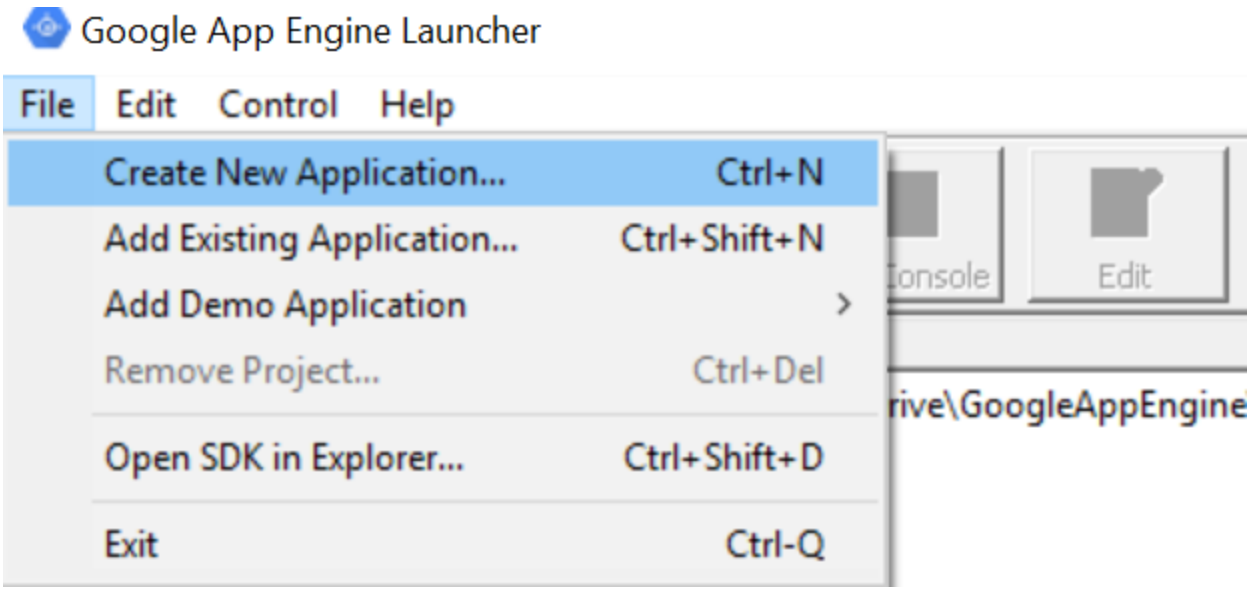
https://console.cloud.google.com/home

Hit the create button and Google should take a few minutes to set up all that is necessary for your newly created app.

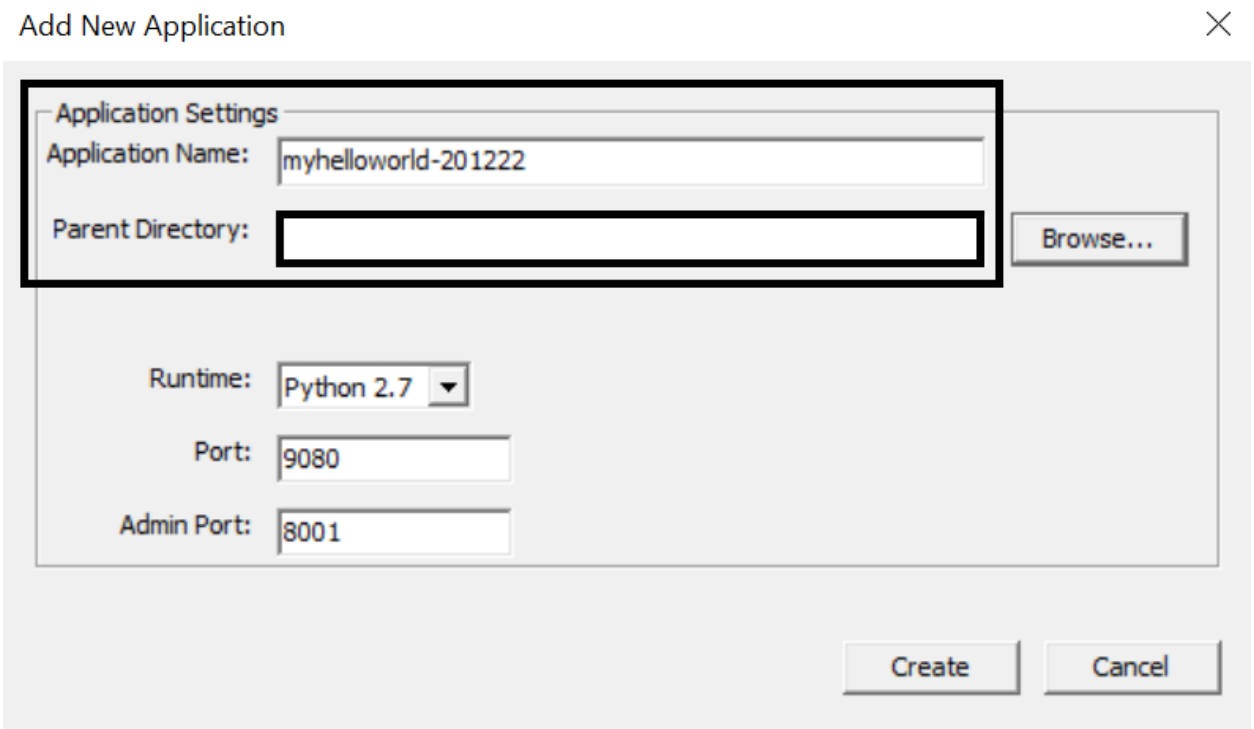
Step 4. Fork the app to develop it locally

The next step in the process is to fork the app on your local machine. This will allow you to make changes to the app locally and deploy it whenever you wish to.

Go to Google App Engine launcher and create a new application.



Enter the project ID of your newly created app. Also, provide the folder (local destination) where you wish to store the app locally. Make sure you select the Python 2.7 as your runtime engine.

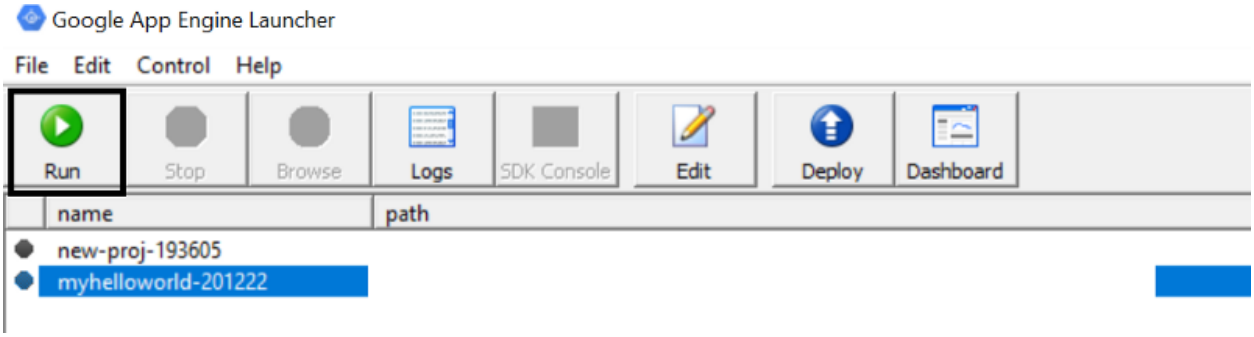


Hit the create button, and you should see your app listed on the window that follows. You should also check that you now see some files in your local storage (the directory you chose in the screenshot above) after this step.

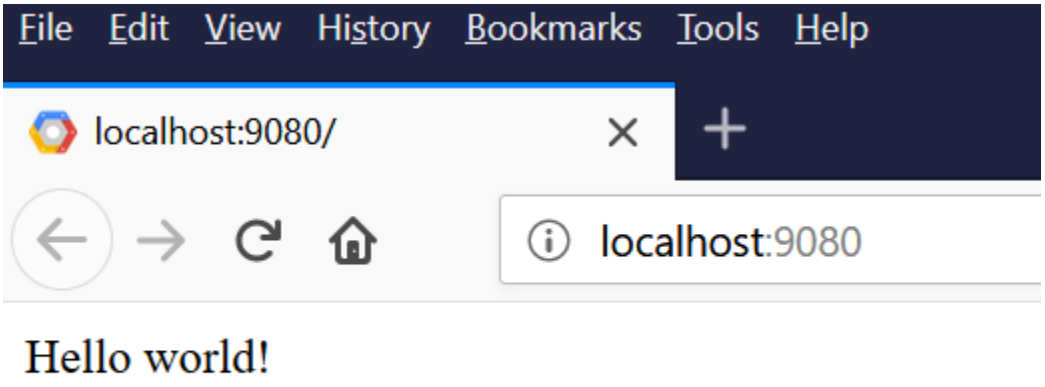
Step 5. Run the app locally

Before you go ahead and make some changes to the app, it is important to check whether or not you have executed all the above steps correctly. This can be done by simply running the app locally.

Select the app and hit the run button on the window.



Wait for a few seconds until you can hit the **Browse** button. Once the **Browse** button becomes clickable, click it. This should take you to the browser, and you should see the hello world text appear in your browser window. Alternatively, you can manually go to the browser and use the port specified to access the app.



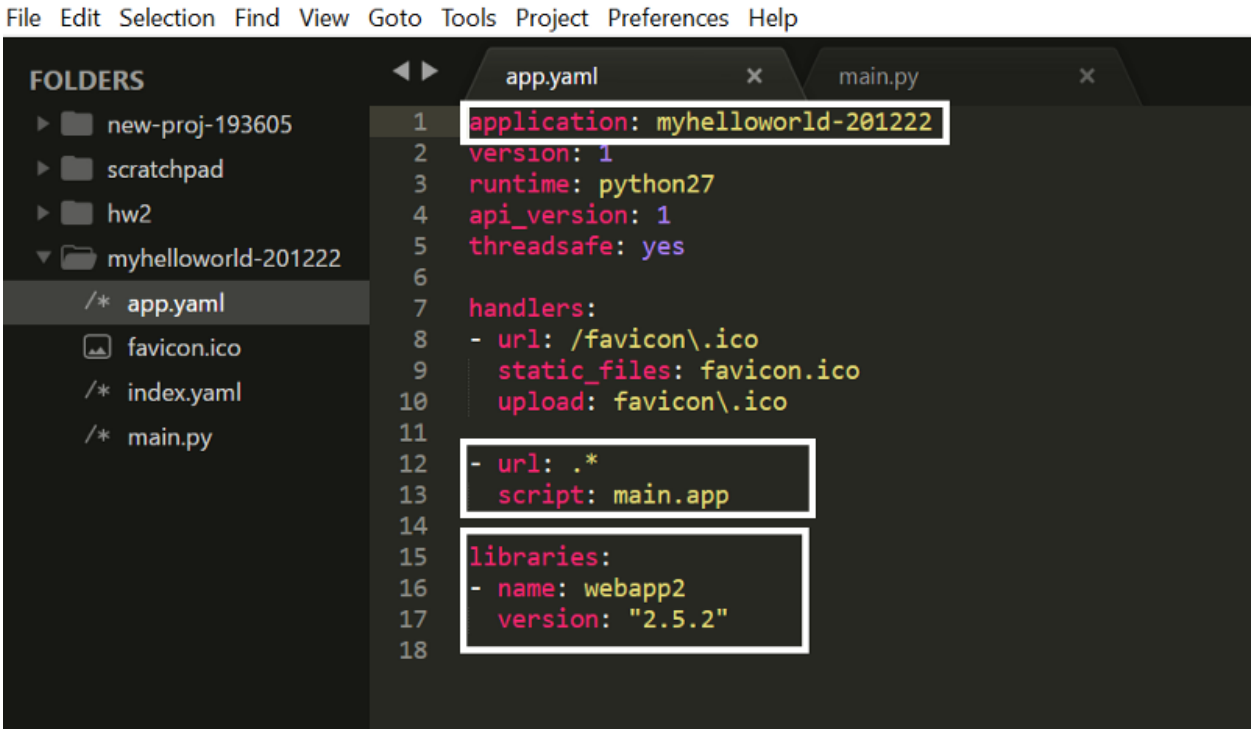
Step 6. Understand the app structure

Open your app folder in the text editor of your choice.

app.yaml

This file is a basic markup file that stores information (some metadata) about the app. It is important to note the following crucial parts of the file.

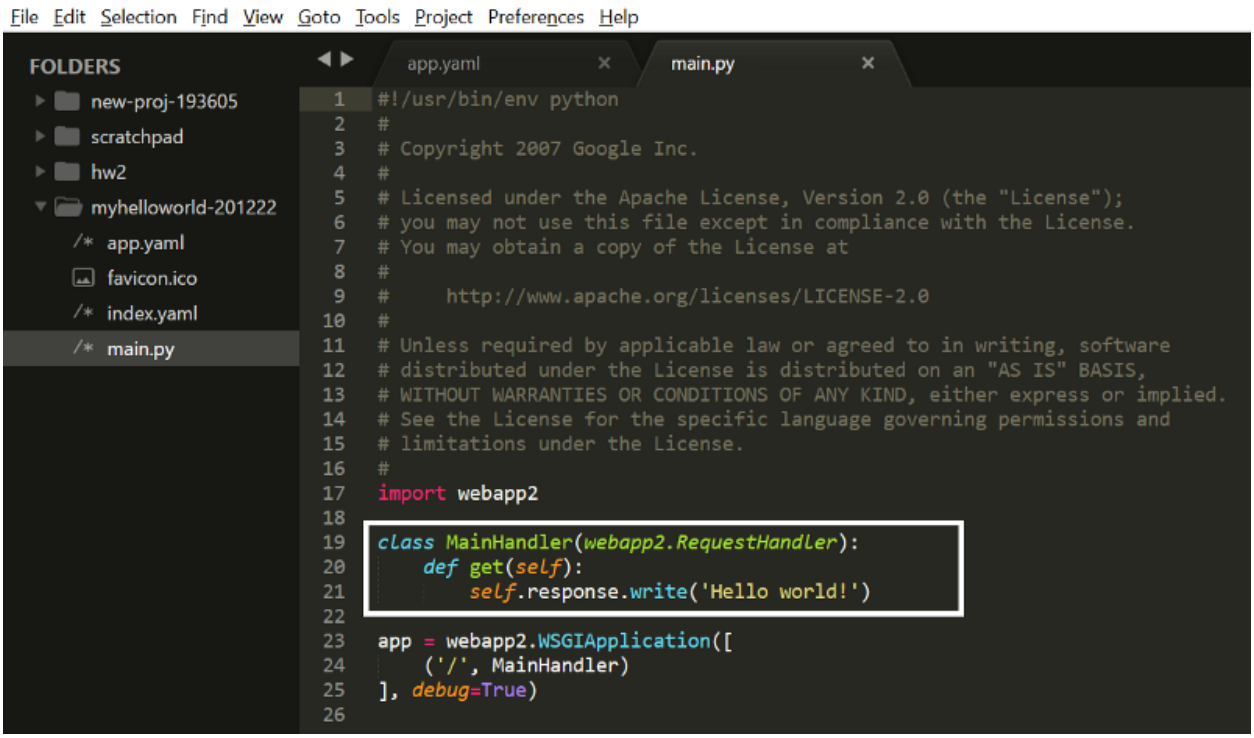
1. **application**
   This is the project ID which you should never change. This is the unique identifier for the app
2. **url -> scr**ipt
   This is the homepage for the app. In other words, this file will be rendered in your browser when you launch the app
3. **libraries**
   This is where you can include external libraries to use within the webapp



app.yaml file in the webapp folder

main.py

This is the homepage of the app



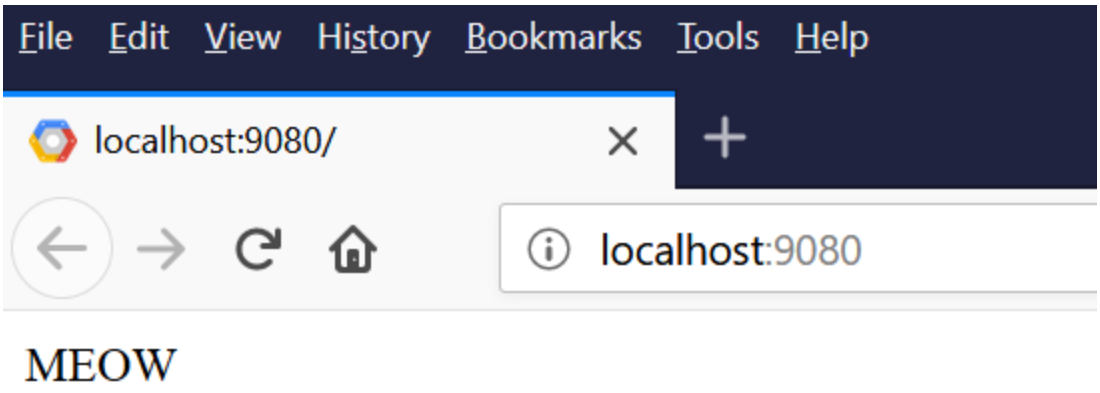
main.py file in the webapp folder

Step 7. Make your changes and deploy the new app

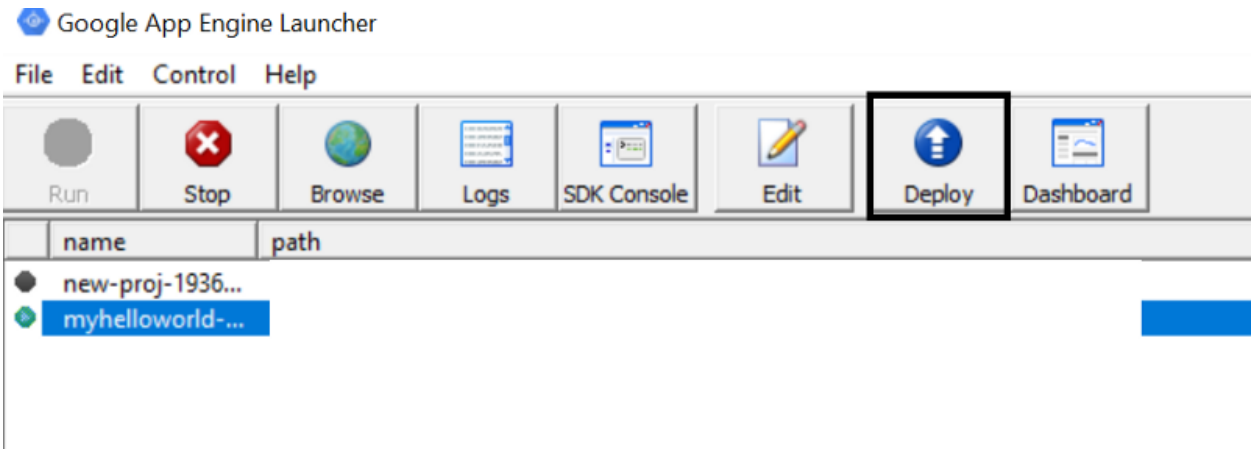
Go ahead and change the text in the above screenshot to something else.

main.py file in the webapp folder

Save the changes, go to the browser and refresh the page.



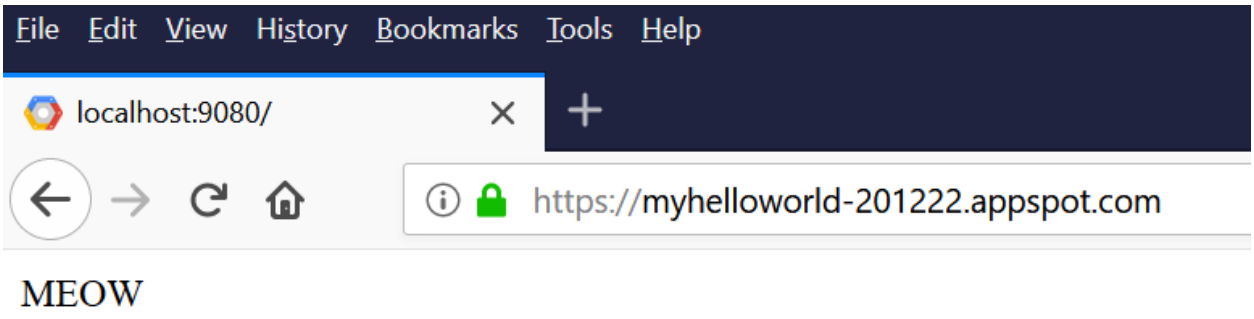
Finally, deploy your changes to the cloud to make them globally accessible via a URL. Go to the App Engine launcher, select the app, and hit the **Deploy** button.



This will ensure your app gets deployed onto Google Cloud. To check whether or not everything worked just fine, go to the URL below:

https://<yourProjectID>.appspot.com/

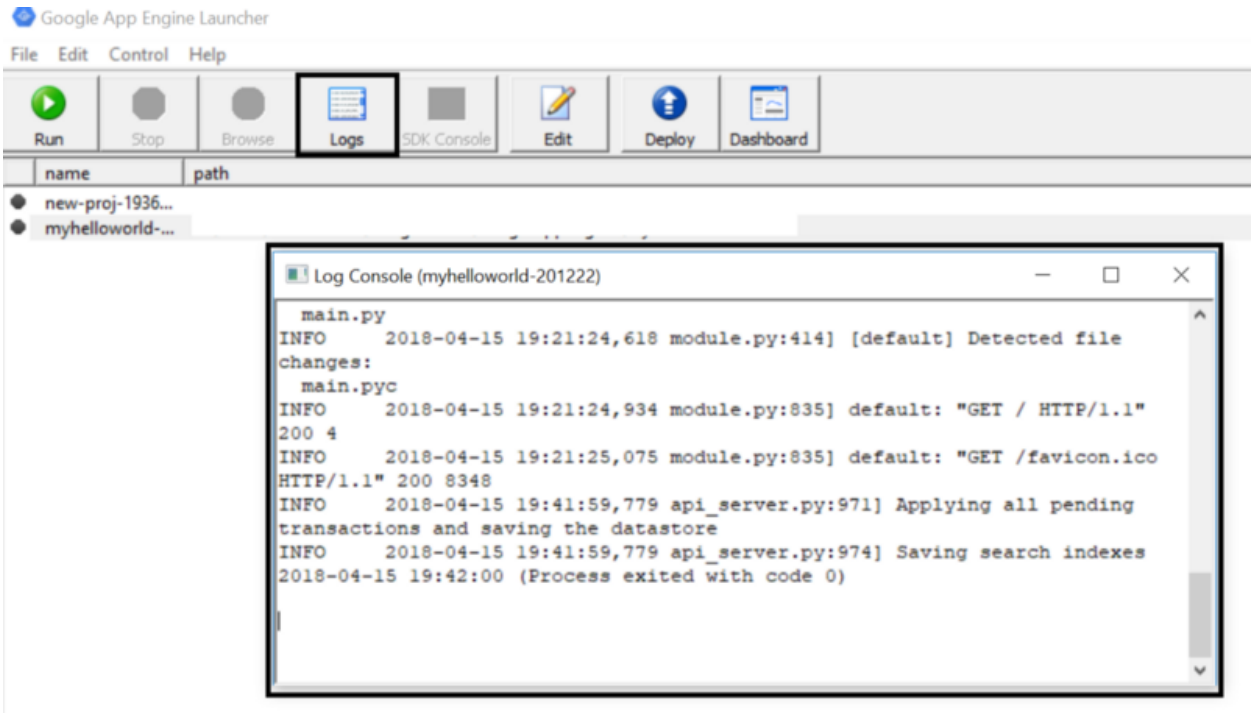
You should see the exact same window as above, expect now, it is a URL that is globally accessible.



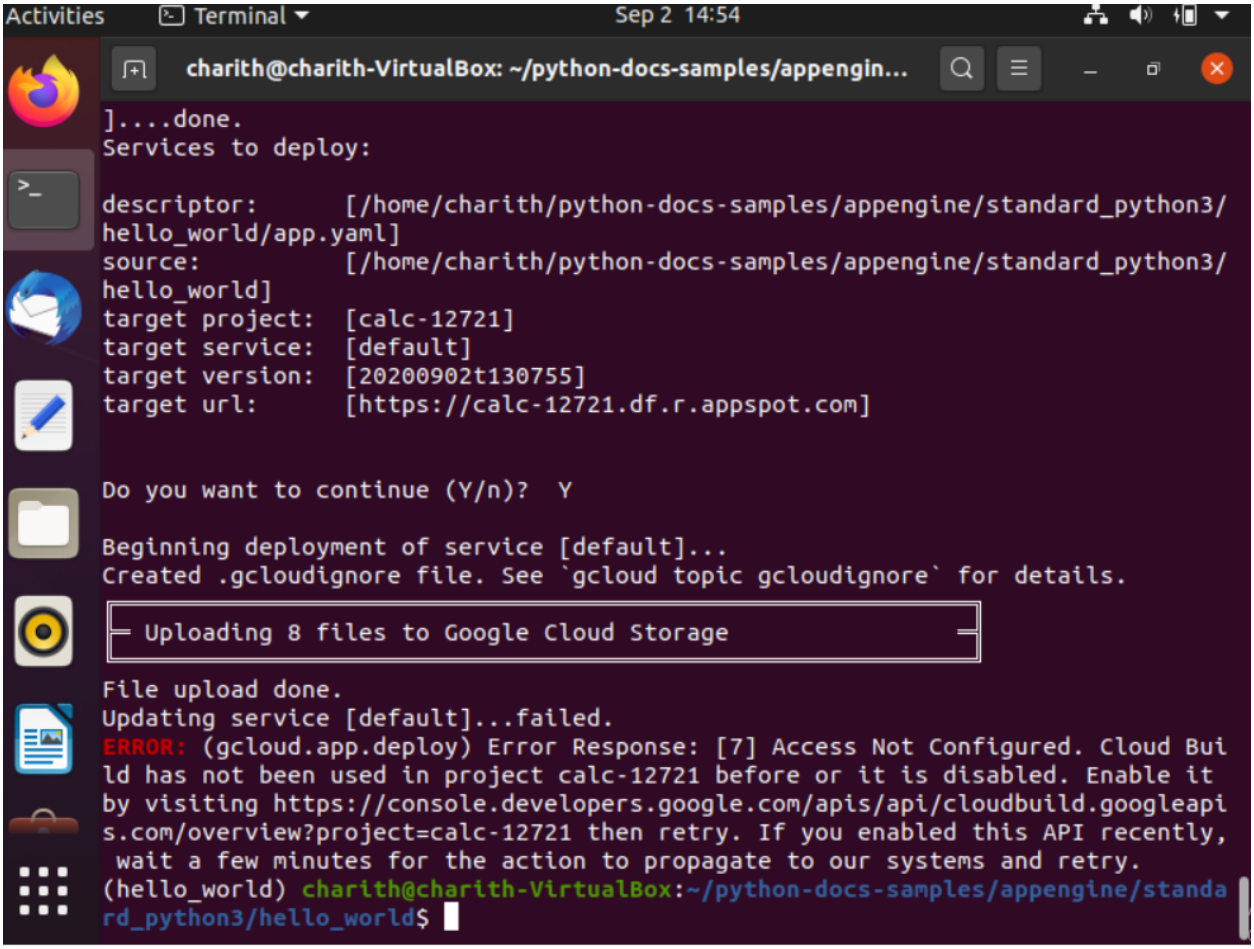
Step 8. Misc

Congratulations, you've finally gotten your first Python webapp deployed on the Google App Engine. Here are some other points which you may find useful.

1. Jinja 2 is an amazing front end templating library for Python that can do some cool stuff, such as passing objects form Python to HTML, using for loops, if conditions, and so on directly out of the box
2. Here's a very useful Udacity course on web development that I have personally found quite resourceful
3. Viewing the logs while running your webapp can be handy to debug and also discover some bugs on the fly



OUTPUT



RESULT

Web Applications were launched using the Google App Engine Launcher successfully.

Ex.No:7 Date :	CloudSim Installation and Simulation of a scheduling algorithm in CloudSim.

AIM

To install a cloud simulator

PROCEDURE

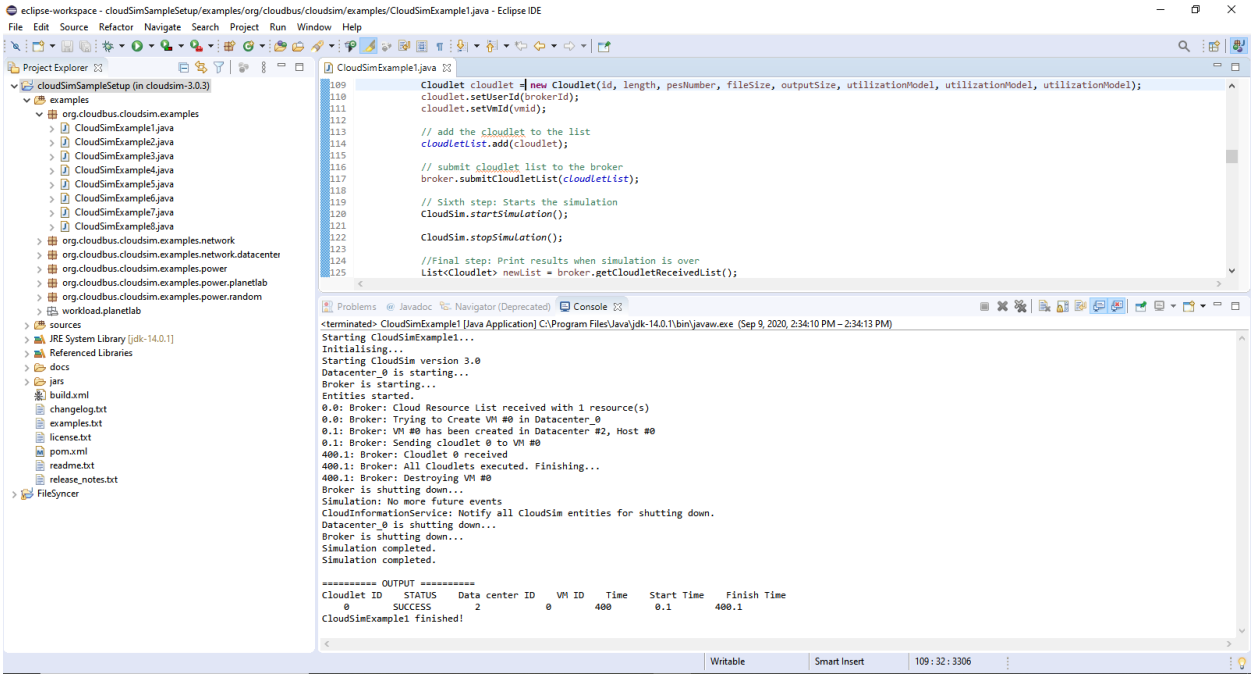
CloudSim installation

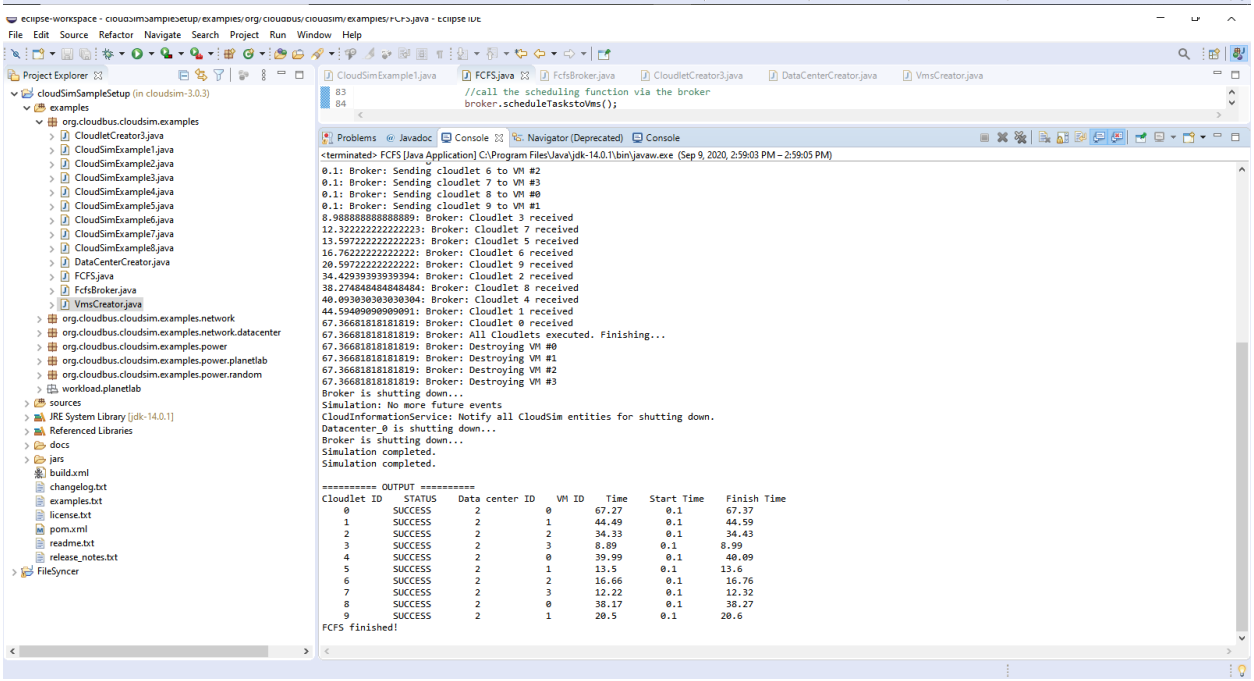
1. Go to https://github.com/Cloudslab/cloudsim/releases/tag/cloudsim-3.0.3
2. Download cloudsim-3.0.3.zip in this link
3. Extract the downloaded zip file.
4. Have an eclipse IDE installed.
5. Go to https://commons.apache.org/proper/commons-math/download_math.cgi
6. Download commons-math3-3.6.1-bin.zip from the above link.
7. Extract the downloaded folder.
8. Open Eclipse SDK and select File--->New--->Java Project.
9. Give a project name and then uncheck the Use Default Location checkbox.
10. Click on browse and select the extracted cloudsim-3.0.3 folder as location.
11. Select Next.
12. In the next window select the Libraries tab.
13. Click on add external JARs button and select commons-math3-3.6.1.jar file from the extracted commons-math3-3.6.1 folder.
14. The existing and added JAR files will be displayed in the libraries tab. Check if the selected file has been added and click finish.
15. Now if there are errors in the project, then right click on project name and select properties.
16. Select the java compiler option in the properties window. Enable the project specific settings checkbox. Change the compiler compliance level to 1.7 and allow rebuilding of the project.

Algorithm simulation

1. Initialize the cloudSim package.
2. Create DataCenters to act as resource providers.
3. Create a data center broker. This will help in selecting a data center for usage.
4. Create a list of virtual machines to help in the execution of scheduling algorithm. Submit the list of virtual machines to the broker.
5. Create a list of cloudlet. A cloudlet specifies a set of user requests using an ID and also keeps track of the user to whom the responses has to be sent after processing the request. Submit the list of cloudlets to the broker. Call the required scheduling algorithm using the broker.
6. Now the tasks will get scheduled in the virtual machines.
7. Execute the tasks by starting simulation.
8. Print the results after execution.

OUTPUT





RESULT

A Cloud simulator was installed successfully.

Ex.No:8 Date :	ONE NODE HADOOP CLUSTER

AIM

To find procedure to set up the one node Hadoop cluster.

PROCEDURE

This document describes how to set up and configure a single-node Hadoop installation so that you can quickly perform simple operations using Hadoop MapReduce and the Hadoop Distributed File System (HDFS).

Required software for Linux include:

1. Java™ must be installed.
2. ssh must be installed and sshd must be running to use the Hadoop scripts that manage remote Hadoop daemons.

```
$ sudo apt-get install ssh
$ sudo apt-get install rsync
```

Download Hadoop

Apache Hadoop 2.8.0

Prepare to Start the Hadoop Cluster

Unpack the downloaded Hadoop distribution. In the distribution, edit the file

etc/hadoop/hadoop-env.sh

to define some parameters as follows:

```
# set to the root of your Java installation
  export JAVA_HOME=”java path”
```

enter into the hadoop folder

```
$ cd hadoop 2.8.0
```

Try the following command:

```
$ bin/hadoop
```

This will display the usage documentation for the hadoop script.

Now you are ready to start your Hadoop cluster in one of the three supported modes:

- Local (Standalone) Mode
- Pseudo-Distributed Mode
- Fully-Distributed Mode

Standalone Operation

By default, Hadoop is configured to run in a non-distributed mode, as a single Java process. This is useful for debugging.

The following example copies the unpacked conf directory to use as input and then finds and displays every match of the given regular expression. Output is written to the given output directory.

```
$ mkdir input
$ cp etc/hadoop/*.xml input
$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.0.jar grep
input output 'dfs[a-z.]+'
$ cat output/*
```

Pseudo-Distributed Operation

Hadoop can also be run on a single-node in a pseudo-distributed mode where each Hadoop

daemon runs in a separate Java process.

Configuration:

Use the following:

etc/hadoop/core-site.xml:

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

etc/hadoop/hdfs-site.xml:

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```

Setup passphraseless ssh

Now check that you can ssh to the localhost without a passphrase:

```
$ ssh localhost
```

If you cannot ssh to localhost without a passphrase, execute the following commands:

```
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
```

Execution

The following instructions are to run a MapReduce job locally.

1. Format the filesystem:

```
$ bin/hdfs namenode -format
```

2. Start NameNode daemon and DataNode daemon:

```
$ sbin/start-dfs.sh
```

The hadoop daemon log output is written to the $HADOOP_LOG_DIR directory (defaults to $HADOOP_HOME/logs).

3. Browse the web interface for the NameNode; by default it is available at:

```
NameNode - http://localhost:50070/
```

4.Make the HDFS directories required to execute MapReduce jobs:

```
$ bin/hdfs dfs -mkdir /user
$ bin/hdfs dfs -mkdir /user/<username>
```

5. Copy the input files into the distributed file system:

```
$ bin/hdfs dfs -put etc/hadoop input
```

6.Run some of the examples provided:

```
$  bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.0.jar grep input output 'dfs[a-z.]+'
```

7.Examine the output files:Copy the output files from the distributed file system to the local filesystem and examine them:

```
$ bin/hdfs dfs -get output output
$ cat output/*
```

or

View the output files on the distributed file system

```
$ bin/hdfs dfs -cat output/*
```

8.When you're done, stop the daemons with:

```
$ sbin/stop-dfs.sh
```

Ex.No:9 Date :	Use of Map And Reduce Tasks

AIM:

To write a word count program to demonstrate the use of Map and Reduce tasks.

PROCEDURE:

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

A MapReduce *job* usually splits the input data-set into independent chunks which are processed by the *map tasks* in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the *reduce tasks*. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks

on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

The MapReduce framework consists of a single master ResourceManager, one slave NodeManager per cluster-node, and MRAppMaster per application (see YARN Architecture Guide).

Minimally, applications specify the input/output locations and supply *map* and *reduce* functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the *job configuration*.

The Hadoop *job client* then submits the job (jar/executable etc.) and configuration to the ResourceManager which then assumes the responsibility of distributing the software/ configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

YARN on a Single Node

You can run a MapReduce job on YARN in a pseudo-distributed mode by setting a few parameters and running ResourceManager daemon and NodeManager daemon in addition.

Configure parameters as follows:etc/hadoop/mapred-site.xml:

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

etc/hadoop/yarn-site.xml:

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

Start ResourceManager daemon and NodeManager daemon:

```
$ sbin/start-yarn.sh
```

Browse the web interface for the ResourceManager; by default it is available at:

ResourceManager - http://localhost:8088/

Run a MapReduce job.

wordCount is a simple application that counts the number of occurrences of each word in a given input set.

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
```

```
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Assuming environment variables are set as follows:

```
    export JAVA_HOME=/usr/java/default
    export PATH=${JAVA_HOME}/bin:${PATH}
    export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
```

Compile WordCount.java and create a jar:

```
$ bin/hadoop com.sun.tools.javac.Main WordCount.java
$ jar cf wc.jar WordCount*.class
```

Assuming that:

/user/joe/wordcount/input - input directory in HDFS

/user/joe/wordcount/output - output directory in HDFS

```
$ bin/hadoop fs -ls /user/joe/wordcount/input/
/user/joe/wordcount/input/file01
```

/user/joe/wordcount/input/file02

$ bin/hadoop fs -cat /user/joe/wordcount/input/file01
Hello World Bye World

$ bin/hadoop fs -cat /user/joe/wordcount/input/file02
Hello Hadoop Goodbye Hadoop

Run the application:

$ bin/hadoop jar wc.jar WordCount /user/joe/wordcount/input /user/joe/wordcount/output

OUTPUT

$ bin/hadoop fs -cat /user/joe/wordcount/output/part-r-00000

Bye 1

Goodbye 1

Hadoop 2

Hello 2

World 2

stop the daemons with

$ sbin/stop-yarn.sh

RESULT

One Node Hadoop Cluster was set up successfully**.**

Ex.No:10 Date :	Study of Transferring files to and from virtual machines

AIM

To study about transferring files to and from various virtual machines

PROCEDURE

This page provides information on transferring files between different types of virtual machines which are provided by the SCS. Often times it is useful to be able to transfer files between the **guest**machine and the **host**machine. On this page you will find instructions for the following methods:

- Creating a Shared Folder in VirtualBox
- Dragging and Dropping Files in VirtualBox

- Managing Files with NextCloud

Creating a Shared Folder in VirtualBox

A shared folder is a folder which makes its files available on both the guest machine *and* the host machine **at the same time**. Creating a shared folder between the guest and the host allows you to easily manage files which should be present on both machines. The course virtual machines are ready to use shared folders right away, but if you are using the virtual machine on your personal computer you will need to specify which folder to use as shared storage.

Shared Folders on SCS Lab Computers using Course VMs

If you are using a course VM on a lab computer, it is likely that a shared folder has already been setup for you. On the desktop of your course VM you should notice a folder titled *SharedFolders*. Inside of this you will find any folders that have been shared between the course VM and lab computers.

You should see two folders that have already been configured for you: **Z_DRIVE** and **Temp**.

Z_DRIVE gives you access to your Windows Account Z:\drive. This is storage that is persistent to your SCS account and available as a network drive on the lab computers.

Temp gives you access to the folder found at D:\temp on the lab computer. Files stored in this folder are local to the machine, meaning that they can be accessed **faster**, but will **delete** from the system when you log out.

If you are working with data that you will need to use again, use the *Z_DRIVE* for your shared folder. If you need faster read/write speed, use the *Temp* folder, but remember to backup your files or they will be deleted when you log off the computer.

Shared Folders on Personal Computers

If you are using your own personal machine, you will need to configure VirtualBox to look in the right place for your shared files.

- Video Tutorial (Windows)
- Quick Start Guide

First, click on the guest machine you intend to share files with. From there, you can select the guest *Settings*and navigate to *Shared Folders*on the left side menu. To create a new shared folder, either click the *New Folder*icon on the right menu **or**right click the empty list of shared folders and click *Add Shared Folder*.From here, there are six options:

- **Folder Path:** The folder name on the **host** machine. Click the drop down menu and navigate to the folder you would like to share.
- **Folder Name:**This is the name of the folder as it will appear on the **guest** machine.
- **Read-Only:** If you check read-only, the **guest**machine will be unable to write changes to the folder. This is valuable when you only want to send files *to*the virtual machine, but do not want to risk having the files modified by the guest.
- **Auto-Mount:** When any external storage is connected to a computer it must be *mounted* in order to be used. It is recommended that you turn on auto-mounting, unless you are familiar with the process of mounting a drive yourself.
- **Mount Point:**Unless you already know about mount points, leave this blank.
- **Make Permanent:** If you check this, the shared folder will be a permanent **machine folder**. If it is not checked, the folder will not be shared after a shutdown

On the course virtual machines, when you load into the desktop, you should see a folder labelled *SharedFolders*. In there you will see any folders that are currently mounted and being shared.

Dragging and Dropping Files in VirtualBox

If you only need to transfer a few files quickly, you can simply drag and drop the files in. On the top bar of the running guest machine, click on *Devices* > *Drag and Drop*and make sure that *Bidirectional*is selected. This means that you will be able to drag files from the host to the guest and from the guest to the host. Once bidirectional drag and drop is checked, you should be able to begin dragging and dropping files.

NOTE: Sometimes when dragging files ***into*the course VM**, you may not be able to drag into the file browser directly. If you encounter this issue, you should drag your files onto the *Desktop* and move the files around from there. You should see the cursor change when it is ready to drop files.

You can also drag files from the guest machine into the host. To do this, simply open the file browser on the host to where you would like to drop the files and drag the files from the virtual machine into the file browser of the host. File transfers should be pretty quick; if the virtual machine seems stuck when transferring, simply cancel the transfer and try again.

Managing Files with NextCloud

On any virtual machine, including VirtualBox, VMWare, or the virtual machines hosted on the SCS OpenStack, you can access the SCS NextCloudservices to move files between multiple machines and your SCS Windows Account storage. NextCloud offers you all of your SCS storage in one remote location, similar to how you might use other file hosting services like Dropbox or Google Drive. Before trying to use NextCloud, you should check that you can access the service by logging in here.

If you can access the NextCloud services, you can browse the various file storage services available to you

- **Linux Home:**These are the files from your SCS Linux Account
- **Windows Home:**These are the files from your SCS Windows Accountand your lab Z:\ drive.

- **NextCloud:** In addition to the other storage accounts provided to you by the SCS, you can also upload up to 20GB of files directly to NextCloud.

With NextCloud, you can upload your files from any machine with an internet connection and download them onto any other machine with an internet connection. For example, you can move project files off of your virtual machine, onto the NextCloud storage, and then download them on your personal laptop. Alternatively, you can upload files from your personal PC onto the NextCloud storage, place it into the *Windows Home*folder, and access those files from either the lab Z:\drive or download them on a virtual machine like VirtualBox or OpenStack.

Uploading Files to NextCloud from a Lab Computer

If you would like to upload files from a lab computer, the easiest way to do this is to place the files you would like to transfer into your Z:\drive. These files will be automatically backup into your NextCloud storage under the *Windows Home*folder. After that, you can move them into the main NextCloud storage or choose to keep them in your Z:\drive.

Uploading Files to NextCloud from a VM or Other PC

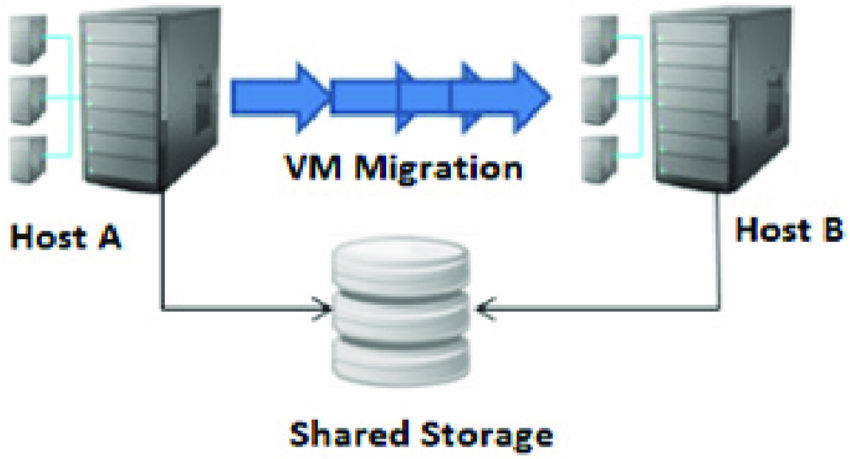
If you would like to upload files from either a VM or any other computer, you can login to the NextCloud service using any of the available interfaces, such as the web interface. Press the "+" icon in the top left of the file browser and select *UploadFile*. From here, you can choose to keep it in the main NextCloud storage, move it into your Windows Account storage (the *Windows Home*folder), or into your Linux Account storage (the *Linux Home* folder)

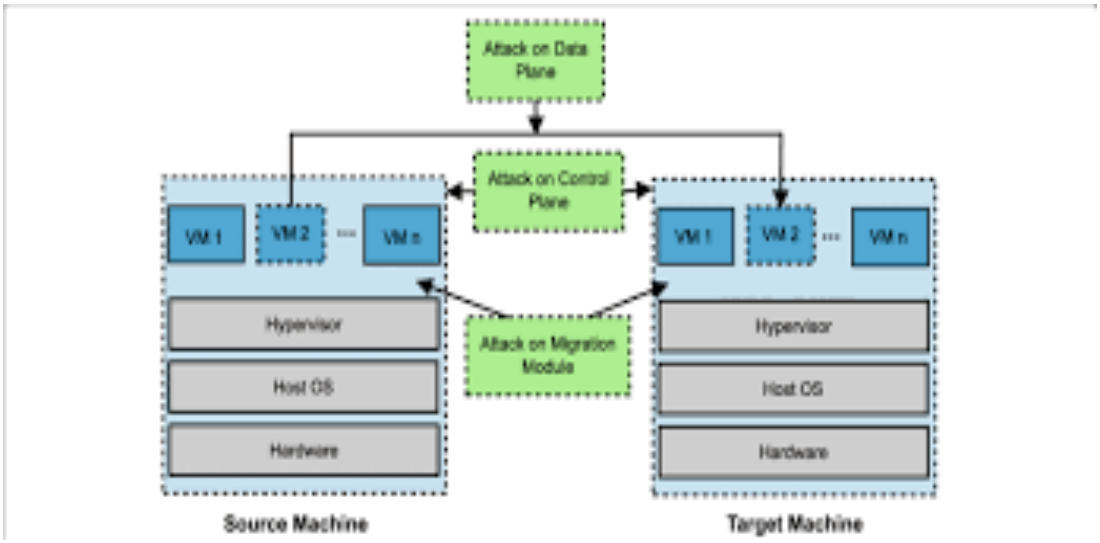
Downloading NextCloud Files to a VM or Other PC

Once your files are uploaded you will be able to download those files onto any machine which can connect to NextCloud. First, log in to your preferred NextCloud interface (eg. the web interface). N to the folder which contains the files you would like to download. Once you are in the target folder, click the checkbox next to each file you would like to download. Above the file listing

you should notice the context bar changing to tell you how many files you have selected and a button labelled *Actions*. Click *Actions* > *Download*.

If you have selected a single file, it will prompt you to confirm the download. If you have chosen more than one file, NextCloud will place all of the selected files into a *zip archive*. Before you can use the files, you will need to extract them from the archive. Once you have downloaded your file, or extracted your archive, you are ready to use your files on your machine.





RESULT

A study of transferring files from one VM to another was successfully implemented

Ex.No:11 Date :	Study of launching Virtual machines with tryStack

AIM

To study about the use of tryStack for launching virtual machines.

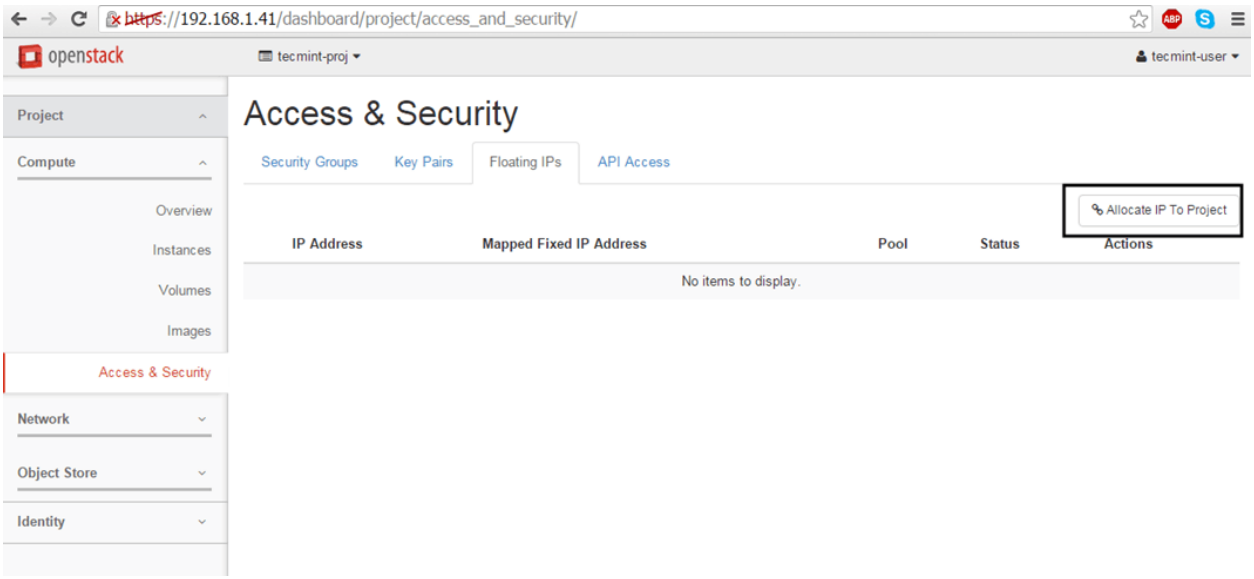
PROCEDURE

Step 1: Allocate Floating IP to OpenStack

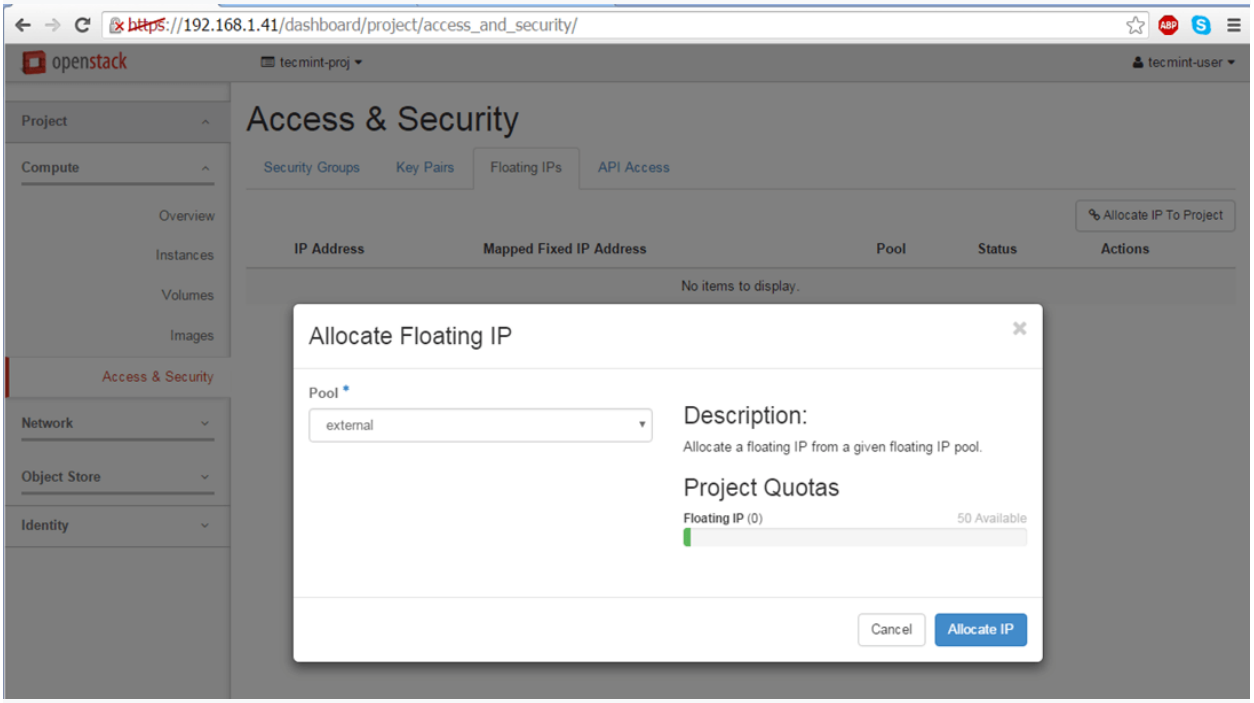
1. Before you deploy an **OpenStack** image, first you need to assure that all pieces are in place and we'll start by allocating floating IP.

Floating IP allows external access from outside networks or internet to an Openstack virtual machine. In order to create floating IPs for your project, login with your **user** credentials and go to **Project** -> **Compute** -> **Access & Security** -> **Floating IPs** tab and click on **Allocate IP** to The Project.

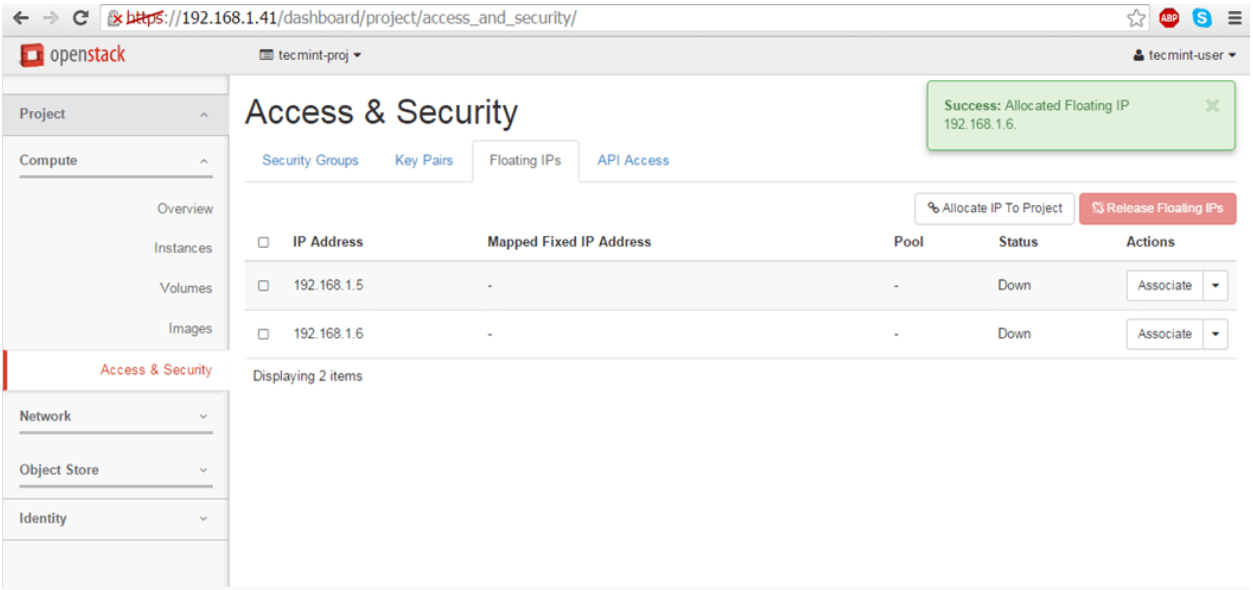
Choose external **Pool** and hit on **Allocate IP** button and the IP address should appear in dashboard. It's a good idea to allocate a Floating IP for each instance you run.



Allocate Floating IP to Project in OpenStack



Allocate Floating IP to External Pool



Confirmation of Adding Floating IP

Step 2: Create an OpenStack Image

2. OpenStack images are just virtual machines already created by third-parties. You can create your own customized images on your machine by installing an Linux OS in a virtual machine using a virtualization tool, such as KVM, VirtualBox, VMware or Hyper-V.

Once you have installed the OS, just convert the file to raw and upload it to your OpenStack cloud infrastructure.

To deploy official images provided by major Linux distributions use the following links to download the latest packaged images:

1. **CentOS 7** – http://cloud.centos.org/centos/7/images/
2. **CentOS 6** – http://cloud.centos.org/centos/6/images/
3. **Fedora 23** – https://download.fedoraproject.org/pub/fedora/linux/releases/23/Cloud/
4. **Ubuntu** – http://cloud-images.ubuntu.com/
5. **Debian** – http://cdimage.debian.org/cdimage/openstack/current/
6. **Windows Server 2012 R2** – https://cloudbase.it/windows-cloud-images/#download

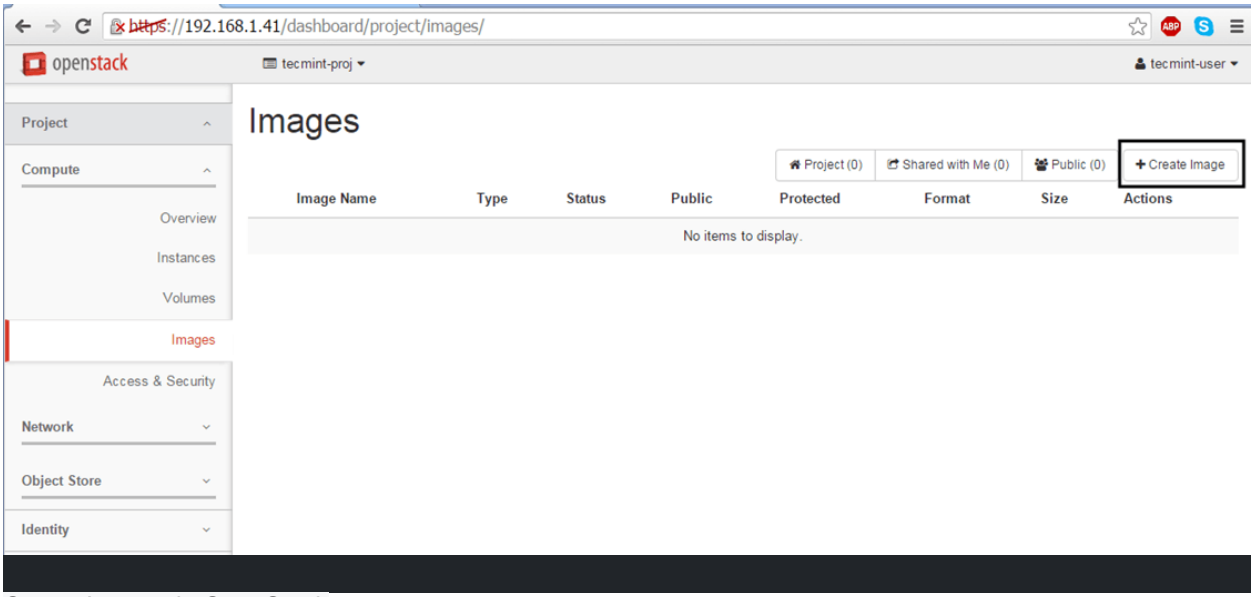
Official images additionally contain the **cloud-init** package which is responsible with SSH key pair and user data injection.

On this guide we'll deploy a test image, for demonstration purposes, based on a lightweight Cirros cloud image which can be obtained by visiting the following link http://download.cirros-cloud.net/0.3.4/.

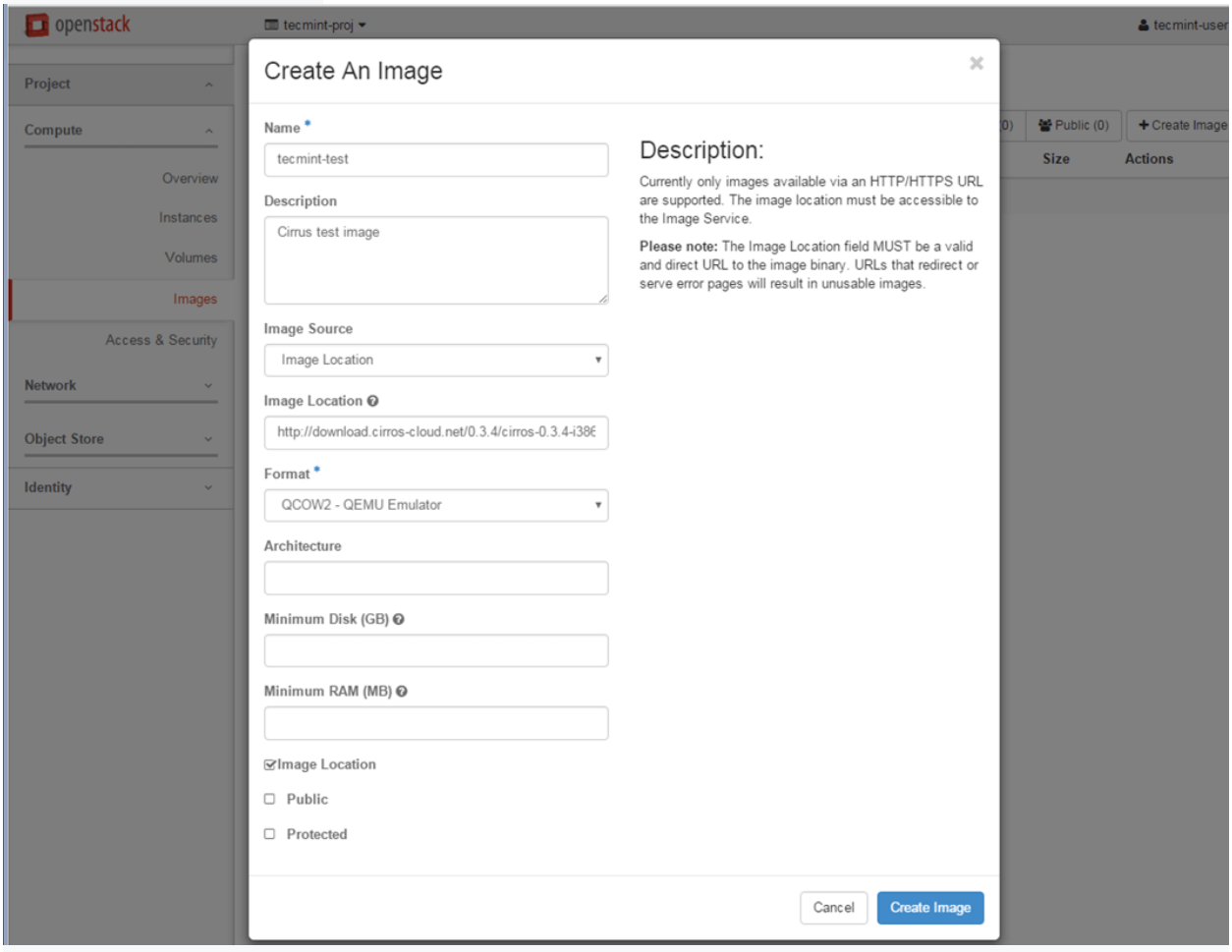
The image file can be used directly from the HTTP link or downloaded locally on your machine and uploaded to OpenStack cloud.

To create an image, go **OpenStack** web panel and navigate to **Project** -> **Compute** -> **Images** and hit on **Create Image** button. On the image prompt use the following settings and hit on **Create Image** when done.

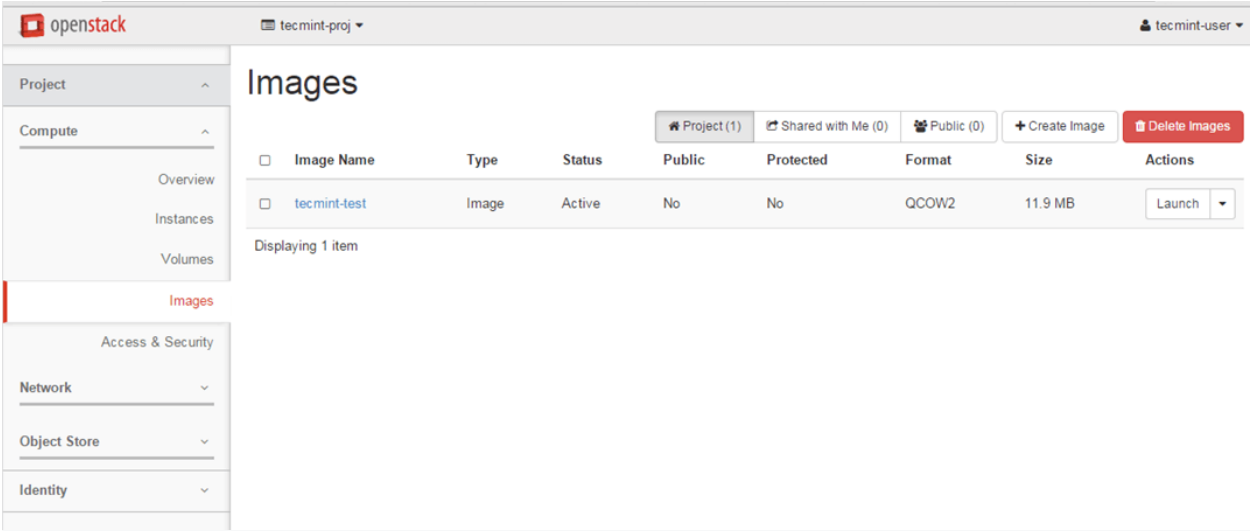
```
Name: tecmint-test
Description: Cirros test image
Image Source: Image Location  #Use Image File if you've
downloaded the file locally on your hard disk
Image Location: http://download.cirros-cloud.net/0.3.4/
cirros-0.3.4-i386-disk.img
Format: QCOWW2 - QEMU Emulator
Architecture: leave blank
Minimum Disk: leave blank
Minimum RAM: leave blank
Image Location: checked
Public: unchecked
Protected: unchecked
```



Create Images in OpenStack



Add OpenStack Image Details

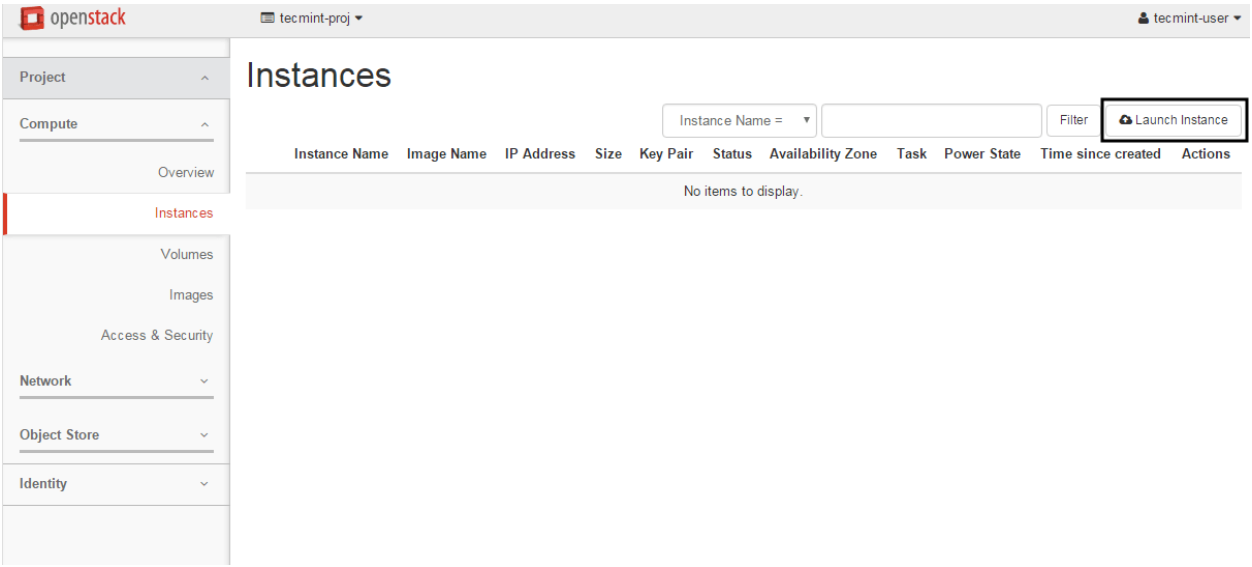


OpenStack Images

Step 3: Launch an Image Instance in OpenStack

3. Once you've created an image you're good to go. Now you can run the virtual machine based on the image created earlier in your cloud environment.

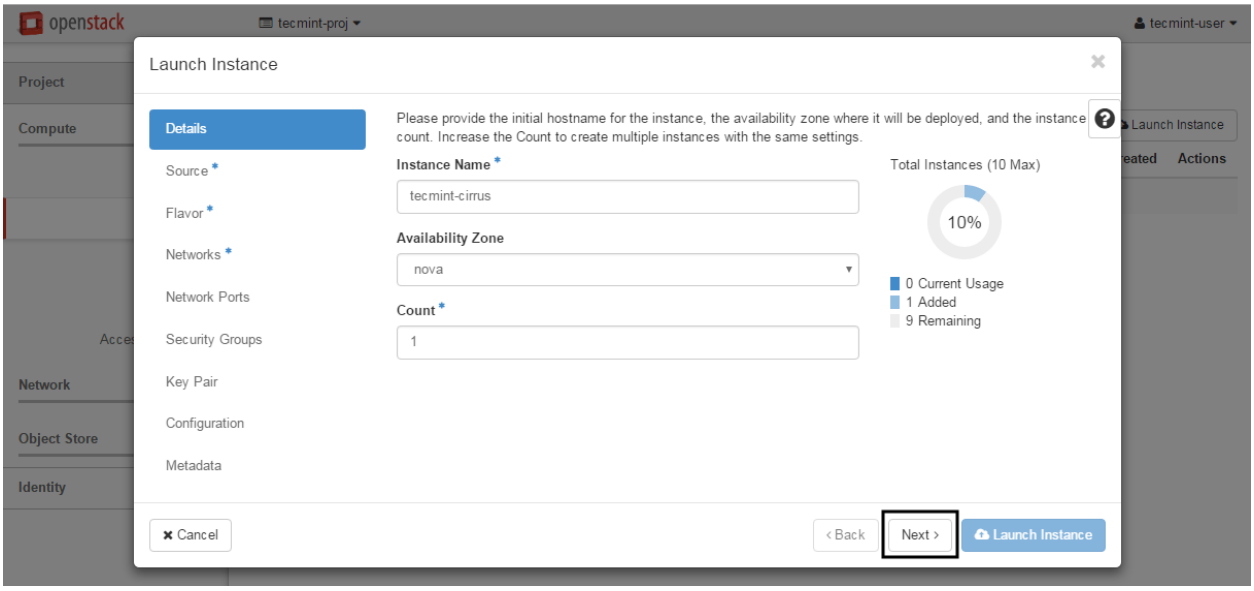
Move to **Project** -> **Instances** and hit on **Launch Instance** button and a new window will appear.



Launch Image Instance in Openstack

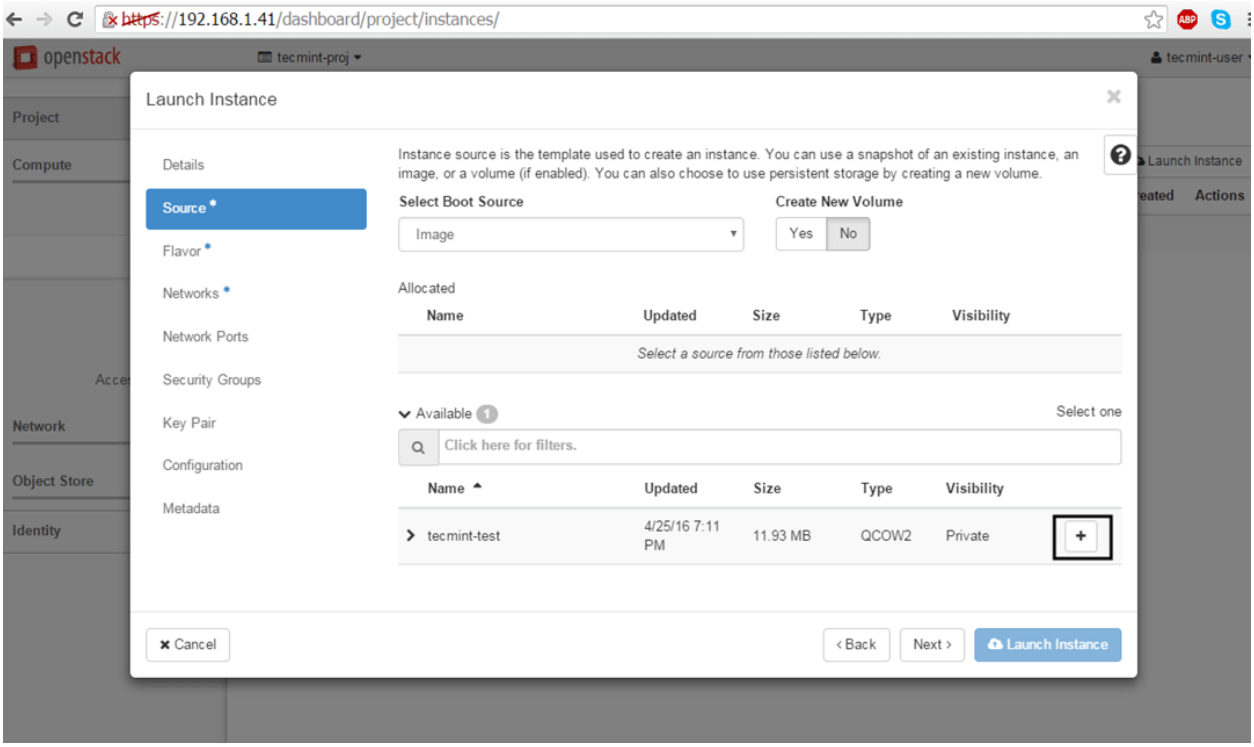
4. On the first screen add a name for your instance, leave the **Availability Zone** to nova, use one instance count and hit on **Next** button to continue.

Choose a descriptive **Instance Name** for your instance because this name will be used to form the virtual machine hostname.

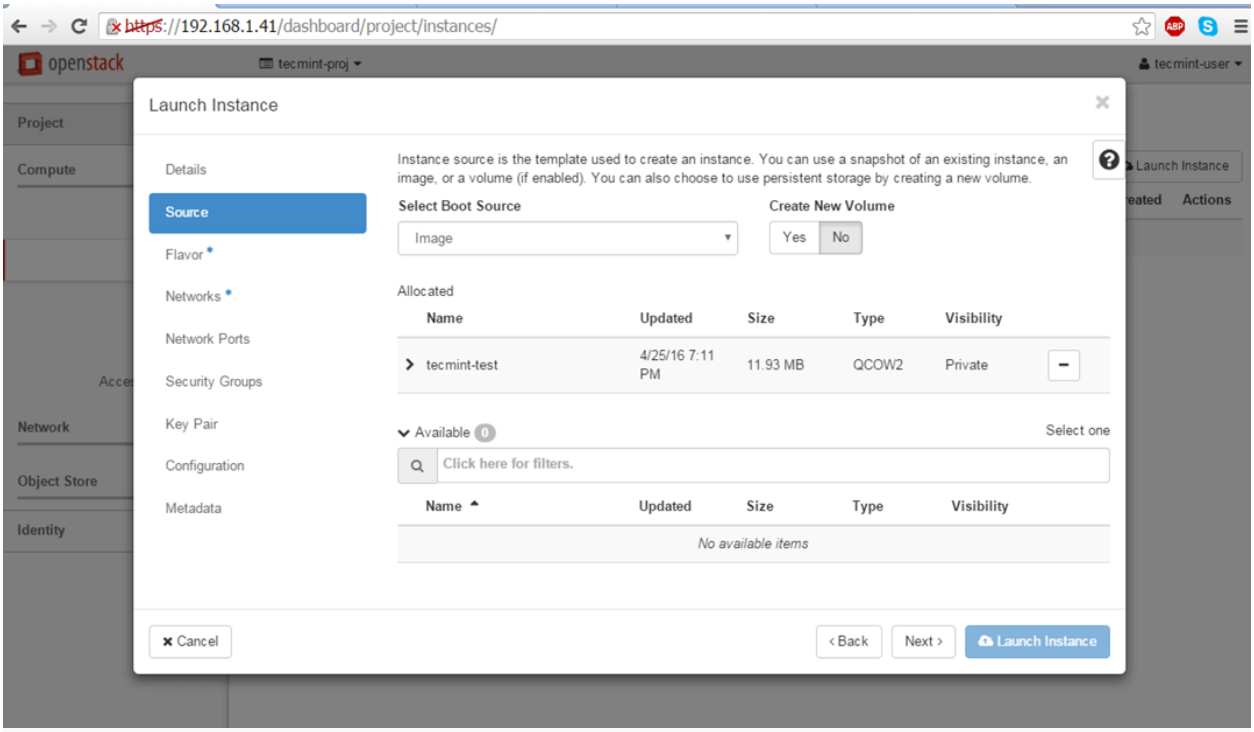


Add Hostname to OpenStack Instance

5. Next, select Image as a **Boot Source**, add the **Cirros** test image created earlier by hitting the + button and hit **Next** to proceed further.

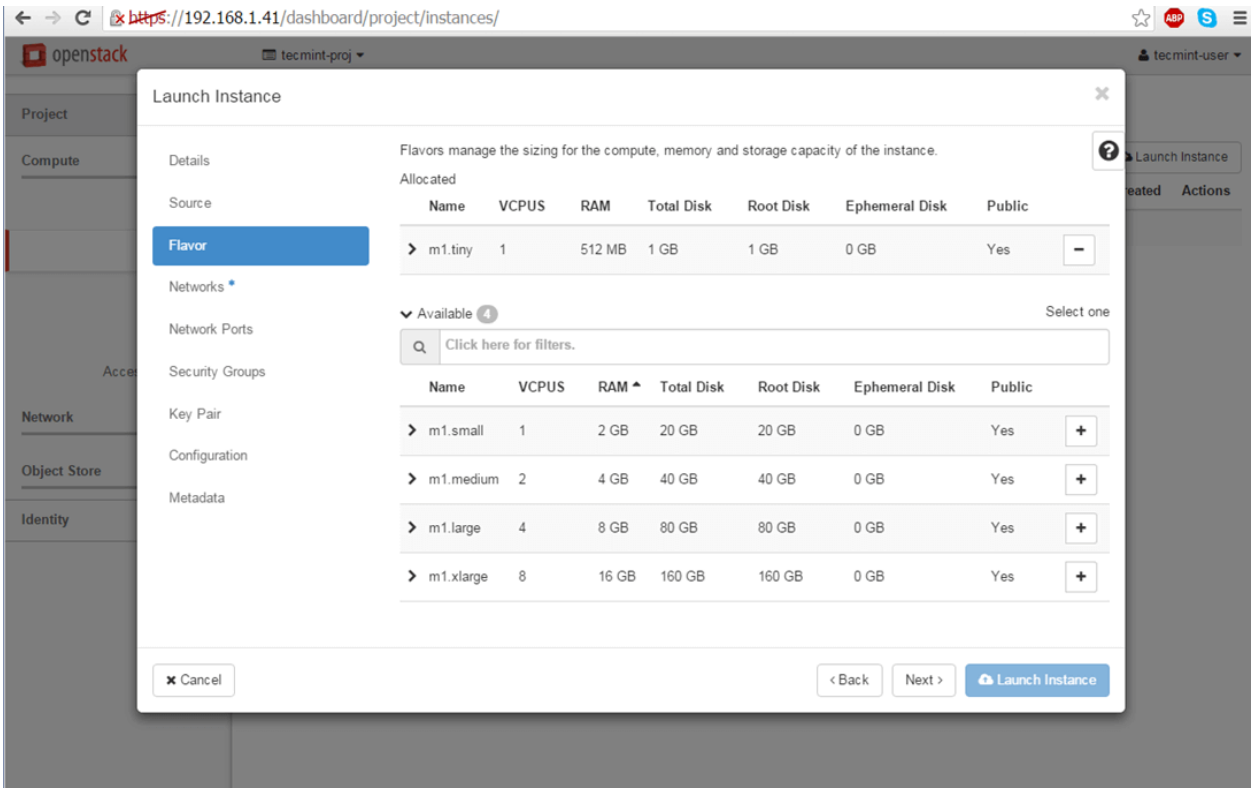


Select OpenStack Instance Boot Source



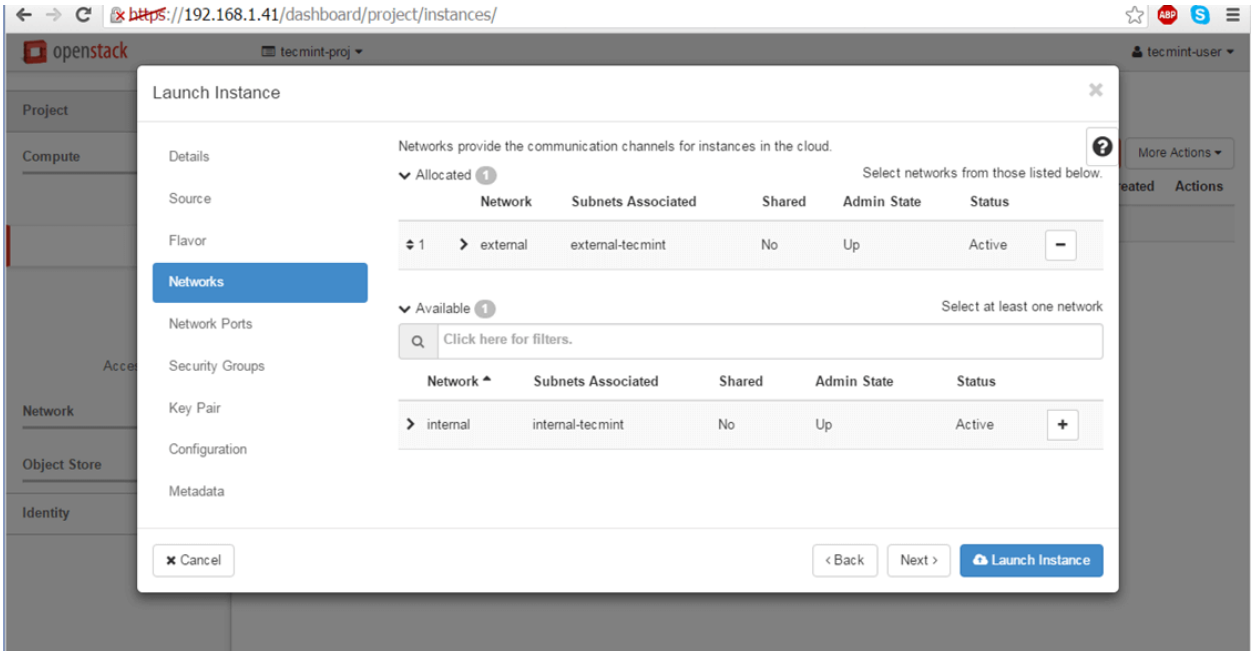
Add Cirros Text Image

6. Allocate the virtual machine resources by adding a flavor best suited for your needs and click on **Next** to move on.



Add Resources to OpenStack Instance

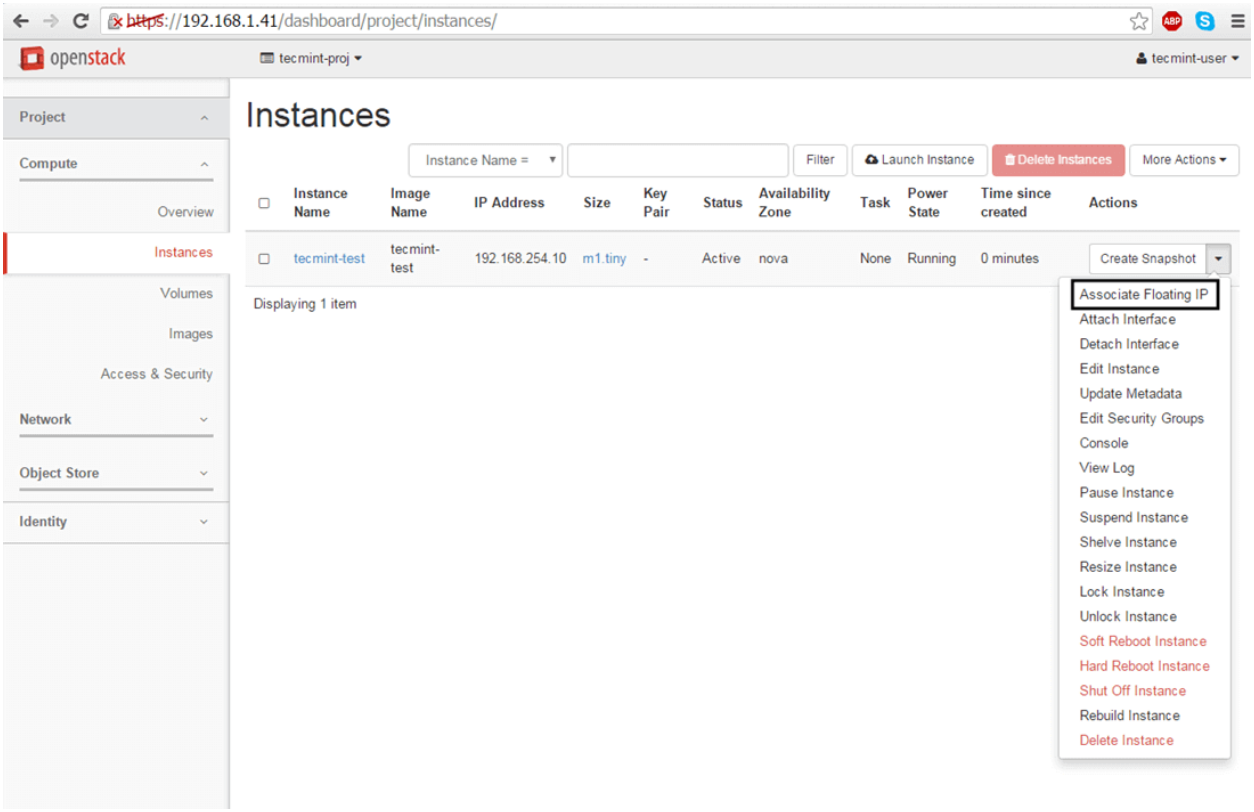
7. Finally, add one of the OpenStack available networks to your instance using the + button and hit on **Launch Instance** to start the virtual machine.



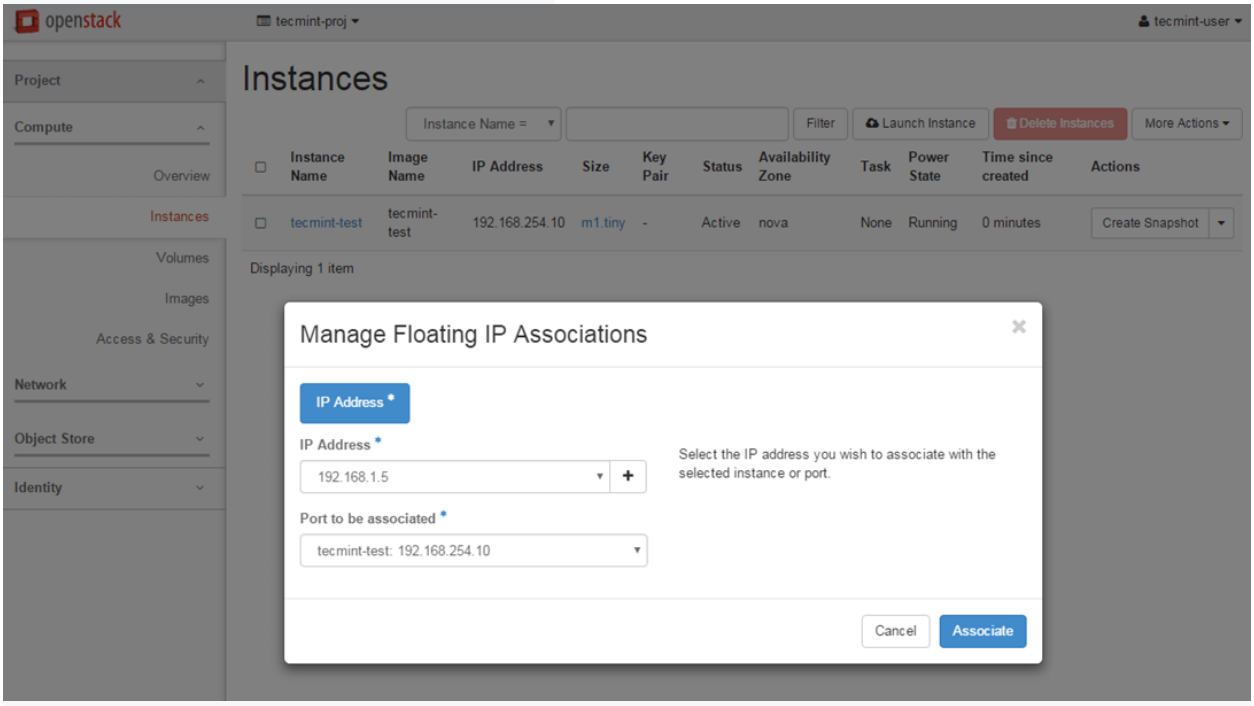
Add Network to OpenStack Instance

8. Once the instance has been started, hit on the right arrow from **Create Snapshot** menu button and choose **Associate Floating IP**.

Select one of the floating IP created earlier and hit on **Associate** button in order to make the instance reachable from your internal LAN.

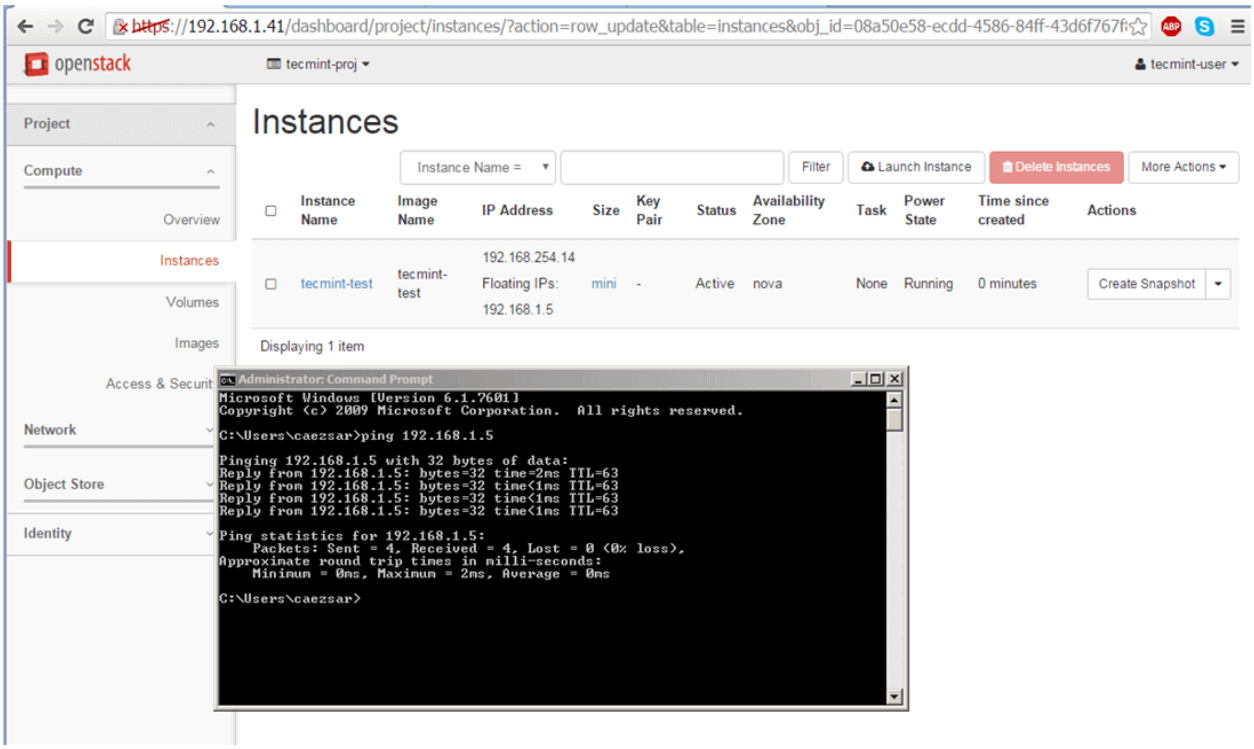


Add Associate Floating IP to OpenStack Instance



Manage Floating IP Associations

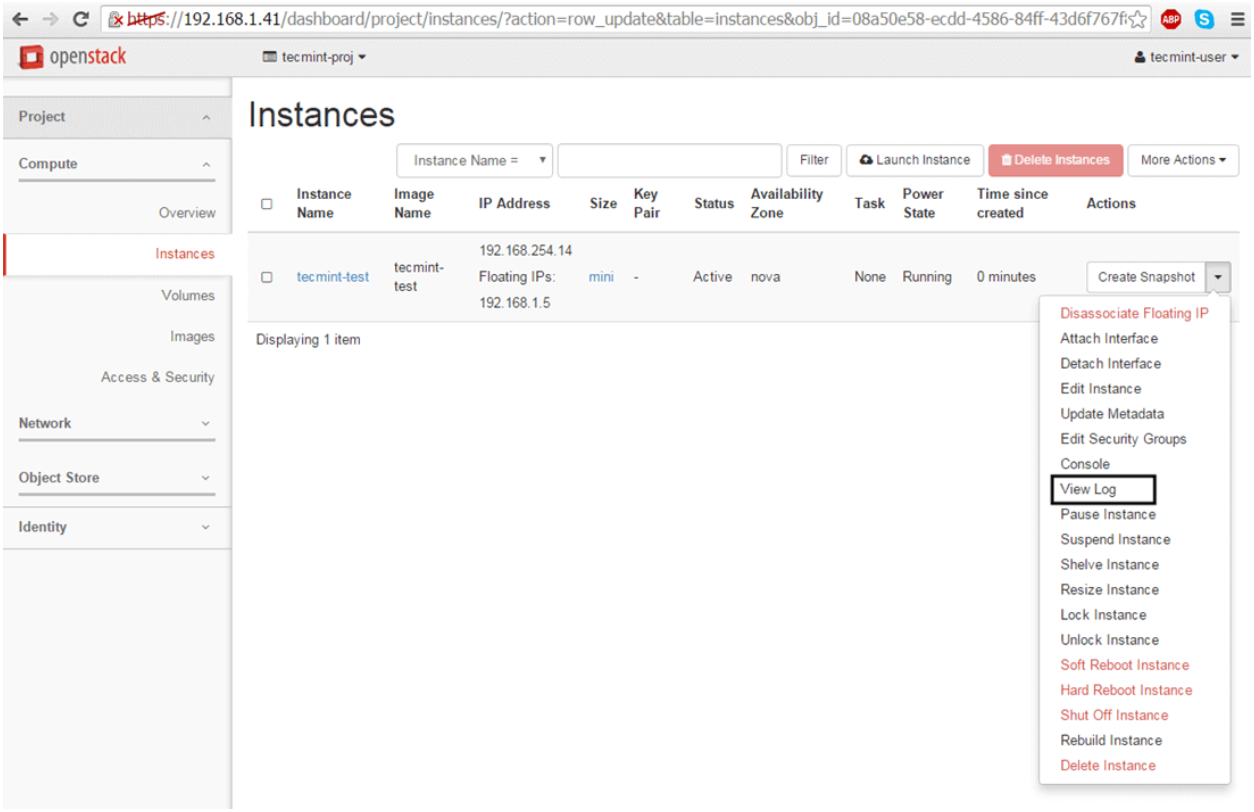
9. To test the network connectivity for your active virtual machine issue a **ping** command against the instance floating IP address from a remote computer in your LAN.



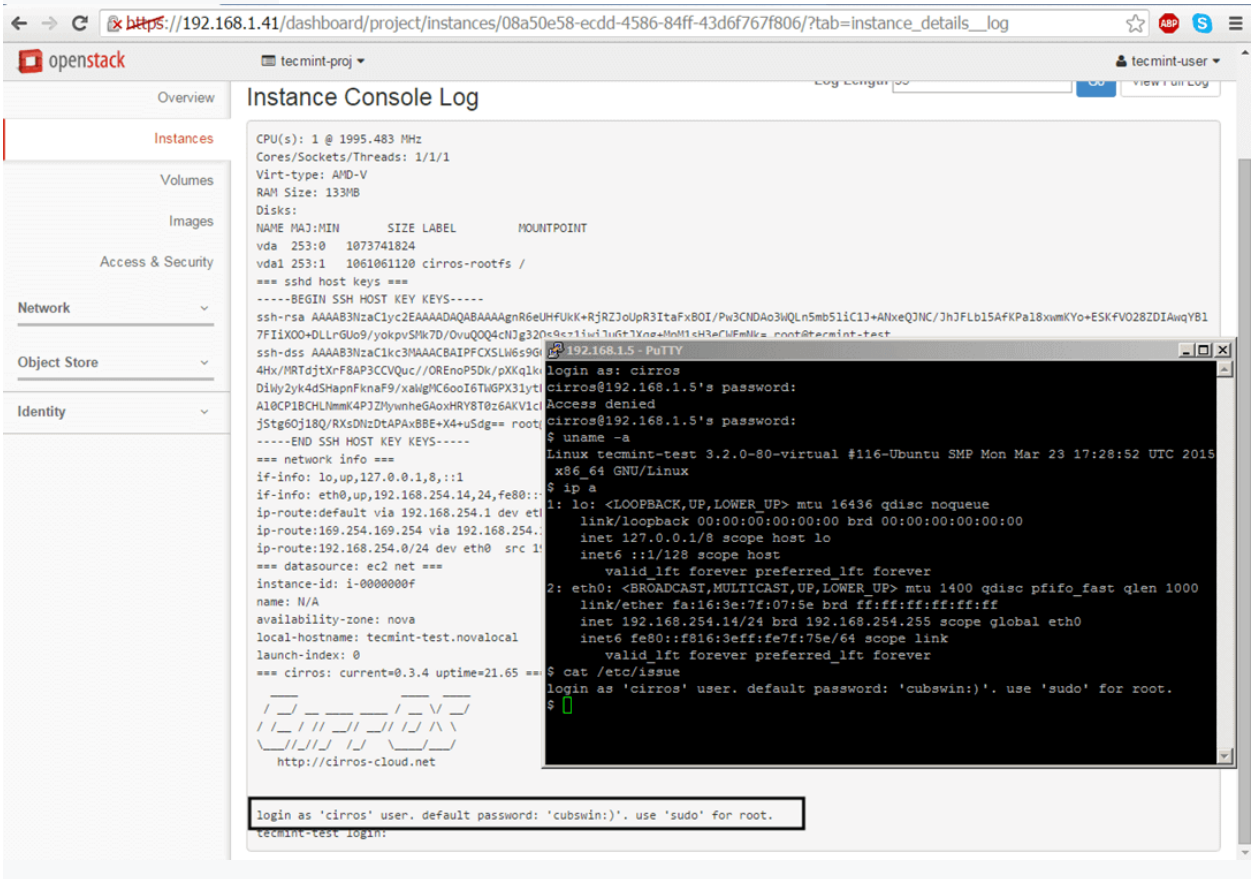
Check Network of Virtual Machine in OpenStack

10. In case there's no issue with your instance and the **ping** command succeeds you can remotely login via SSH on your instance.

Use the instance **View Log** utility to obtain **Cirros** default credentials as illustrated on the below screenshots.



Instance View Log Utility

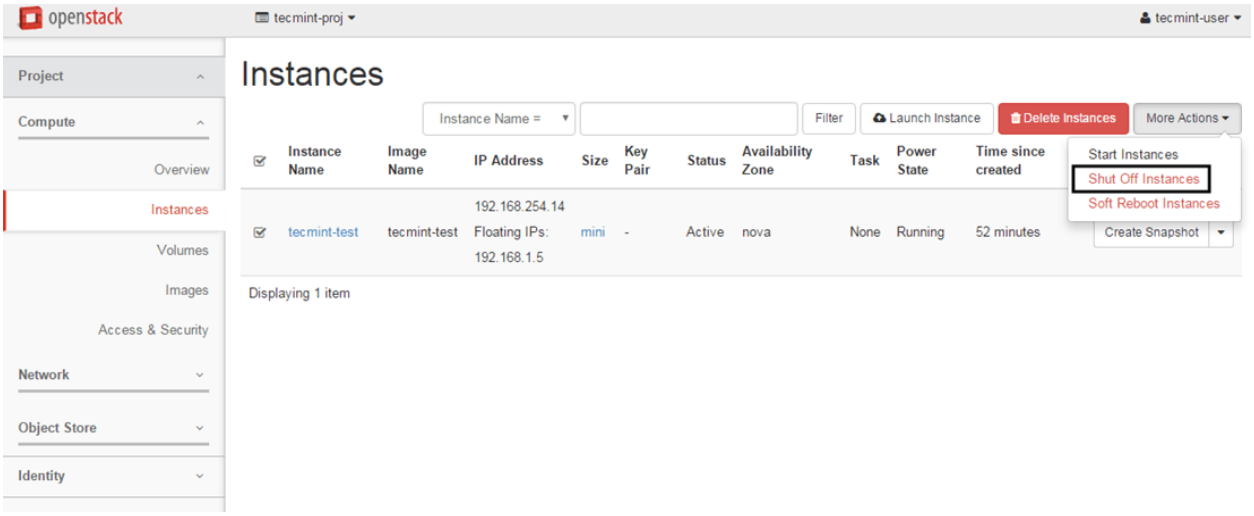


Instance Login Credentials

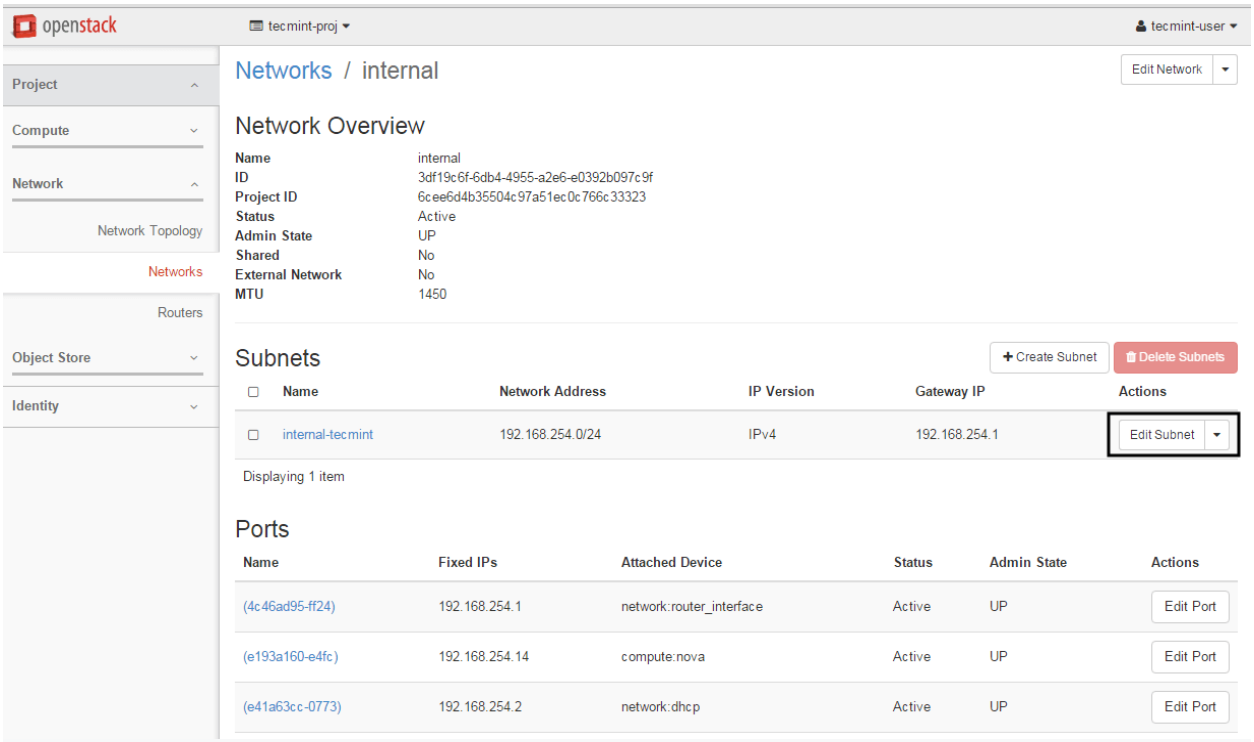
11. By default, no DNS name servers will be allocated from the internal network DHCP server for your virtual machine. This problem leads to domain connectivity issues from instance counterpart.

To solve this issue, first stop the instance and go to **Project** -> **Network** -> **Networks** and edit the proper subnet by hitting the **Subnet Details** button.

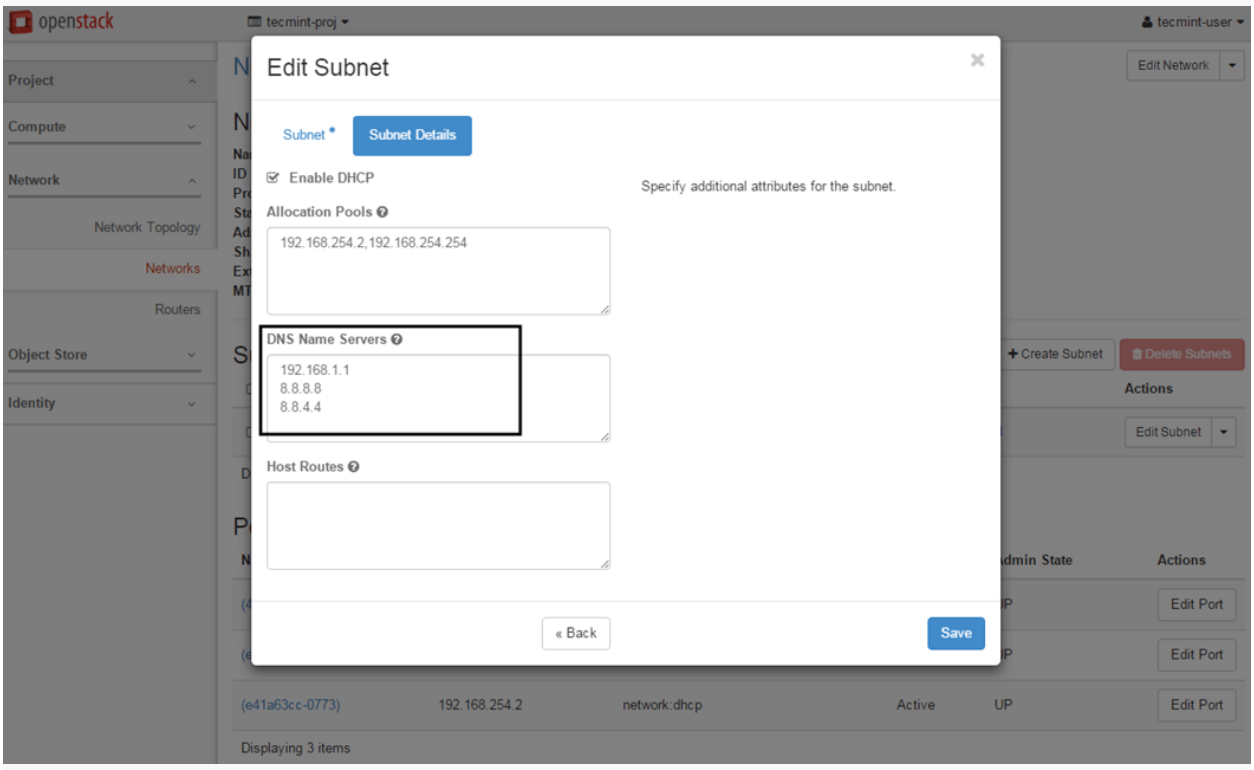
Add the required DNS name servers, save the configuration, start and connect to the instance console to test if the new configuration has been applied by pinging a domain name. Use the following screenshots as a guide.



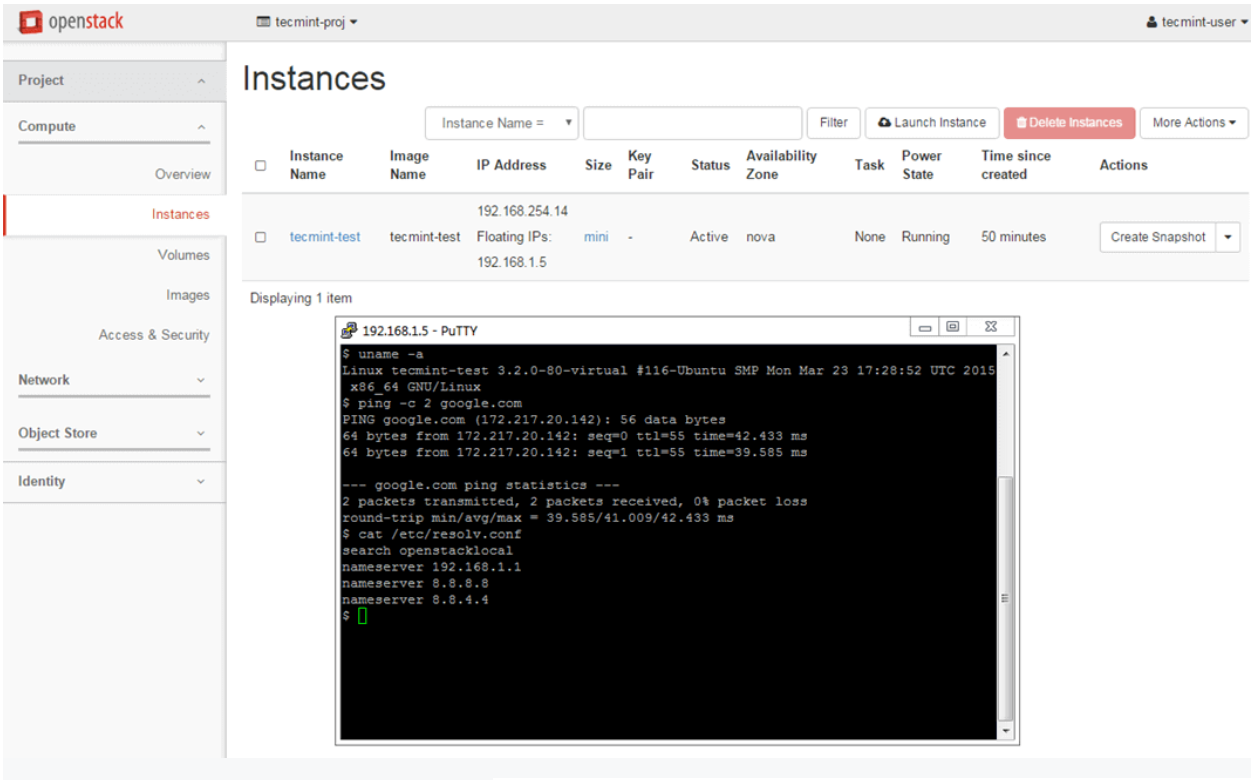
Shutdown Instance



Modify Instance Network Subnet



Add DNS Servers to Instance



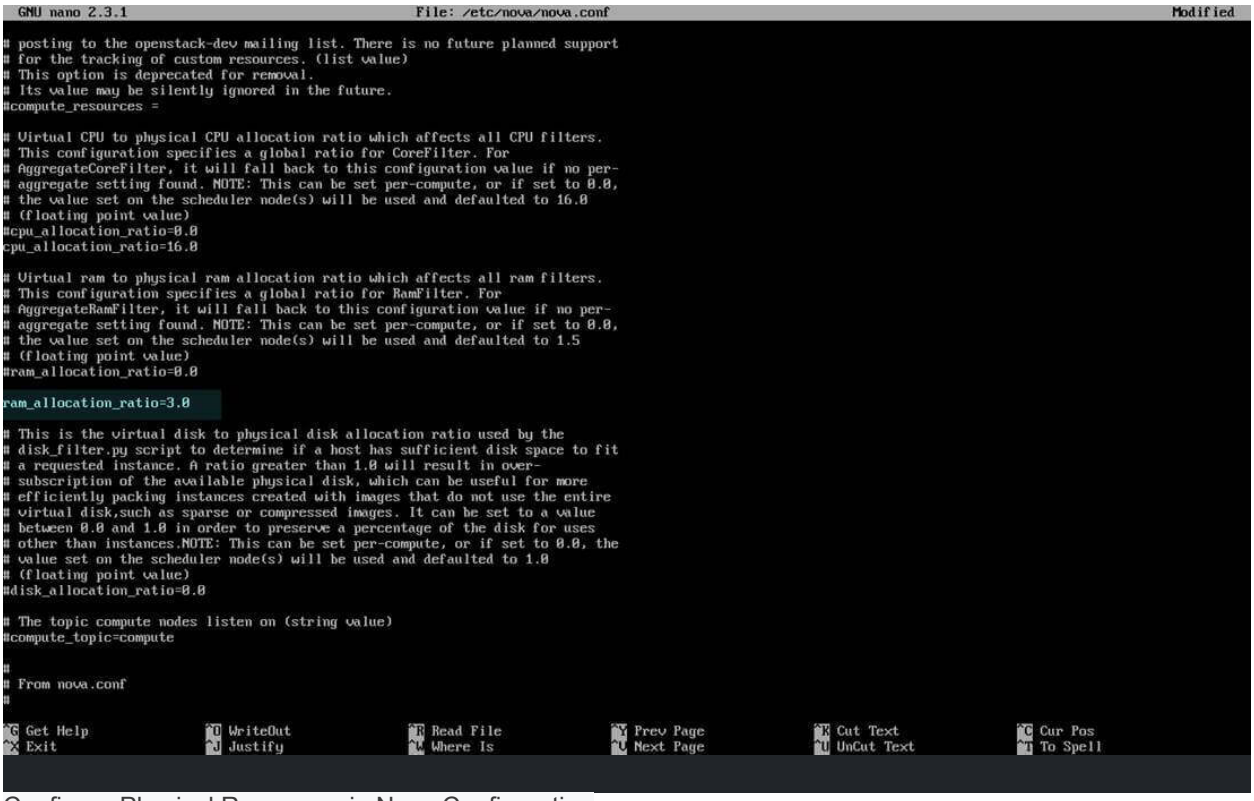
Check Instance Network Connectivity

In case you have limited physical resources in your infrastructure and some of your instances refuse to start, edit the following line from nova configuration file and restart the machine in order to apply changes.

```
# vi /etc/nova/nova.conf
```

Change the following line to look like this:

```
ram_allocation_ratio=3.0
```



Configure Physical Resources in Nova Configuration

That’s all! Although this series of guides just scratched the surface of **OpenStack** mammoth, now you have the basic knowledge to start create new tenants and use real Linux OS images in order to deploy virtual machines in your own OpenStack cloud infrastructure.

RESULT

A study of launching virtual machines using trystack was successfully implemented.