EX. NO.	DATE	LIST OF EXPERIMENTS	PAGE NO.	SIGNATURE OF FACULTY IN-CHARGE
1		Create a virtual machine		
2		Installation of Platforms		
3		Deploying existing Apps		
4		Create a drop box using Google AP		
5		Transfer Data using Google APPs		
6		Upload and download using Google APPs		
7		Encryption and Decryption of Text		
8		Create a datacenter with one host and run one cloudlet on it		
9 (a)		To create a datacenter with one host and run two cloudlet on it		
9 (b)		To create a datacenter with two hosts and run two cloudlets		
10		To create two datacenters with one host and run two cloudlets		
11		To create two datacenters with one host and run cloudlets of two users		
12		To pause and resume the simulation and create simulation entities dynamically		
13		Create a Warehouse Application in Salesforce.Com		
14		Create a Warehouse Application in Salesforce.Com using Apex prog Lang		
15		Implementation of SOAP Web Services		

Ex No: 1 **Register No:**
Date: **Name:**

To create Virtual Machine in an Ubuntu Operating System

Aim:
To create virtual machine in an Ubuntu operating system.

Procedure:

Step 1: Check Virtualization Support in Ubuntu
Before installing **KVM** on **Ubuntu**, first verify if the hardware supports **KVM**. A minimum requirement for installing **KVM** is the availability of CPU virtualization extensions such as **AMD-V** and **Intel-VT**.

To check whether the Ubuntu system supports virtualization, run the following command.

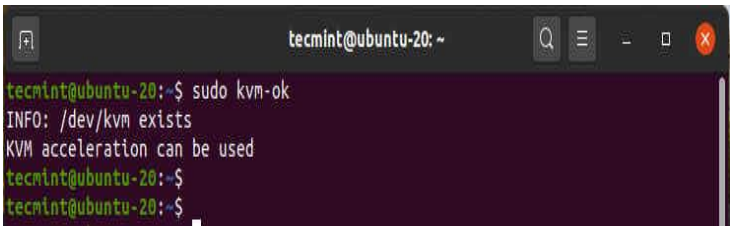
```
$ egrep -c '(vmx|svm)' /proc/cpuinfo
```

An outcome greater than **0** implies that virtualization is supported.

To check if your system supports KVM virtualization execute the command:
`$ sudo kvm-ok`

If the "kvm-ok" utility is not present on your server, install it by running the apt command:
`$ sudo apt install cpu-checker`

Now execute the "kvm-ok" command to probe your system.
`$ sudo kvm-ok`



The output clearly indicates that we are on the right path and ready to proceed with the installation of KVM.

Step 2: Install KVM on Ubuntu 20.04 LTS

With the confirmation that our system can support KVM virtualization, install KVM.
To install KVM, **virt-manager**, **bridge-utils** and other dependencies, run the following command:

```
$ sudo apt install -y qemu qemu-kvm libvirt-daemon libvirt-clients bridge-utils virt-manager
```

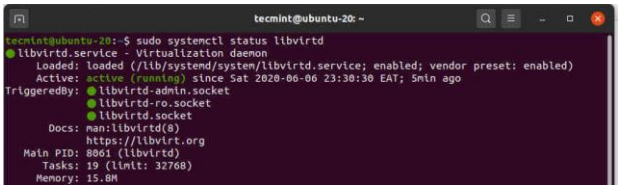
Explanation of Packages used.

- The qemu package (quick emulator) is an application that allows you to perform hardware virtualization.
- The qemu-kvm package is the main KVM package.
- The libvritd-daemon is the virtualization daemon.
- The bridge-utils package helps you create a bridge connection to allow other users to access a virtual machine other than the host system.
- The virt-manager is an application for managing virtual machines through a graphical user interface.

Before proceeding further, we need to confirm that the virtualization **daemon – libvritd-daemon – is running.**

To do so, execute the command.

```
$ sudo systemctl status libvirtd
```



You can enable it to start on boot by running:

```
$ sudo systemctl enable --now libvirtd
```

To check if the KVM modules are loaded, run the command:

```
$ lsmod | grep -i kvm
```

- From the output, you can observe the presence of the kvm_intel module. This is the case for Intel processors. For AMD CPUs, you will get the kvm_intel module instead.

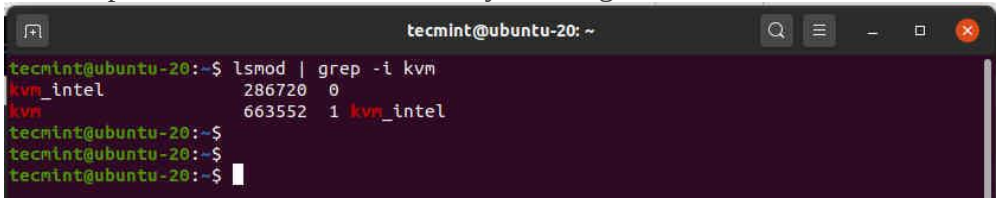


Fig.Check KVM Modules in Ubuntu

Step 3: Creating a Virtual Machine in Ubuntu

- With **KVM** successfully installed, We are now going to create a virtual machine. There are 2 ways to go about this: You can create a virtual machine on the command-line or using the KVM **virt-manager** graphical interface.

Create a Virtual Machine via Command Line

- The **virt-install** command-line tool is used for creating virtual machines on the terminal. A number of parameters are required when creating a virtual machine.
- Here's the full command I used when creating a virtual machine using a **Deepin ISO** image:

```
$ sudo virt-install --name=deepin-vm --os-variant=Debian10 --vcpu=2 --ram=2048 --graphics spice --location=/home/Downloads/deepin-20Beta-desktop-amd64.iso --network bridge:vibr0
```

- The --name option specifies the name of the virtual machine – **deepin-vm** The --os-variant flag indicates the OS family or derivate of the VM. Since **Deepin20** is a derivative of Debian, I have specified **Debian 10** as the variant.
- To get additional information about OS variants, run the command
  `$ osinfo-query os`

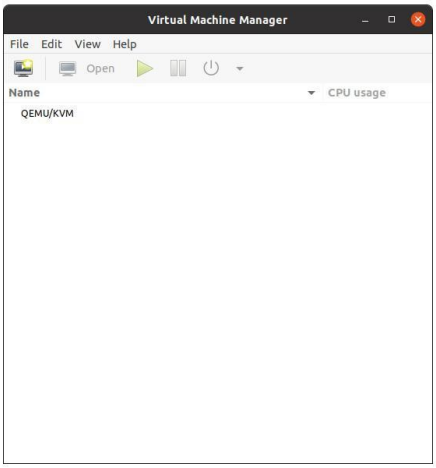
  The --vcpu option indicates the CPU cores in this case 2 cores,
  the --ram indicates the RAM capacity which is **2048MB**.
  The --location flag point to the absolute path of the ISO image and
  the --network bridge specifies the adapter to be used by the virtual machine. Immediately after executing the command, the virtual machine will boot up and the installer will be launched ready for the installation of the virtual machine.

Create a Virtual Machine via virt-manager

- The virt-manager utility allows users to create virtual machines using a GUI. To start off, head out to the terminal and run the command.

```
$ virt-manager
```

- The virtual machine manager window will pop open as shown.



- KVM Virtual Machine Manager
- Now click the monitor icon to start creating a virtual machine.

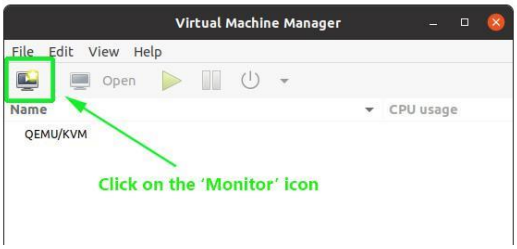


Fig.Create a Virtual Machine in KVM

- On the pop-up window, specify the location of your ISO image. In our case, the ISO image is located in the 'Downloads' folder in the home directory, so we'll select the first option – Local Install Media (ISO image or CDROM). Next, click the 'Forward' button to continue.

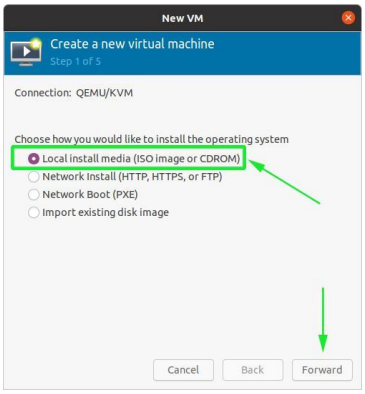
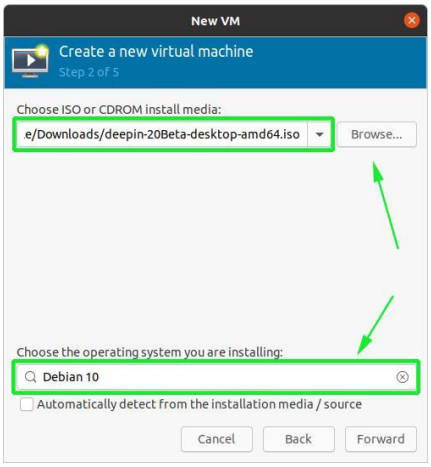


Fig.Choose Local Install Media

In the next step, browse to the ISO image on your system and directly below, specify the OS family that your image is based on.



Choose ISO Image

Next, select the memory capacity and the number of CPUs that your virtual machine will be allocated, and click 'Forward'.

Fig: Choose Memory and CPU for VM

And finally, in the last step, specify a name for your virtual machine and click on the 'Finish' button.

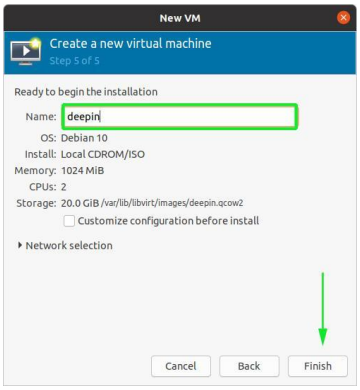


Fig: Set Virtual Machine Name

The creation of the virtual machine will take a few minutes upon which the installer of the OS you are installing will pop open.

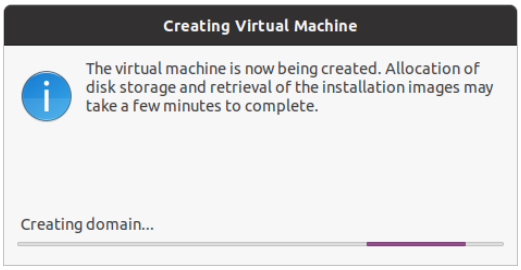


Fig: Creating Virtual Machine

- At this point, you can proceed with the installation of the virtual machine.

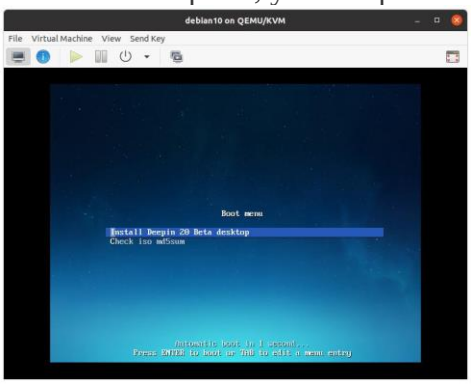


Fig: Virtual Machine Installation

Result:

Hence KVM hypervisor is installed sucessfully on Ubuntu 20.04 LTS.

Ex No: 2 **Register No:**
Date: **Name**

Installing the Platform on Google Cloud Platform (GCP)

Aim: To study about Installation of Platforms on GCP

Procedure:

Deployment Steps

To deploy an instance of the platform to an AWS cloud, execute the following steps.

Step 1: Create a Service Account

Create a service account with the required credentials for performing the installation.

Step 2: Configure the Installation Environment

Step 3: Run the Platform Installer

Run the platform installer, Provazio, by entering the following command from a command-line shell:

```
docker pull gcr.io/iguazio/provazio-dashboard:stable && docker run --rm --name provazio-dashboard \
  -v /tmp/env.yaml:/tmp/env.yaml \
  -e PROVAZIO_ENV_SPEC_PATH=/tmp/env.yaml \
  -p 8060:8060 \
  gcr.io/iguazio/provazio-dashboard:stable
```

Step 4: Access the Installer Dashboard

In a web browser, browse to `localhost:8060` to view the Provazio dashboard.

Select the plus-sign icon (+) to create a new system.

Step 5: Choose the GCP Scenario

In the Installation Scenario page, check GCP, and then click Next.

Step 6: Configure General Parameters

On the General page, fill in the configuration parameters, and then click Next.

Description

A free-text string that describes the platform instance.

System Version

The platform version. Insert the release build number that you received from Iguazio (for example, "3.0_b51_20210308021033").

Owner Full Name

An owner-name string, containing the full name of the platform owner, for bookkeeping.

Owner Email

An owner-email string, containing the email address of the platform owner, for bookkeeping.

Username

The username of a platform user to be created by the installation. This username will be used together with the configured password to log into the platform dashboard. You can add additional users after the platform is provisioned.

User Password

A platform password for the user generated by the installation — to be used with the configured username to log into platform dashboard; see the password restrictions. You can change this password after the platform is provisioned.

Region

The region in which to install the platform.

System Domain

A custom platform domain (for example, "customer.com"). The installer prepends the value of the System Name parameter to this value to create the full platform domain.

Allocate Public IP Addresses

Check this option to allocate public IP addresses to all of the platform nodes.

System Name

A platform name (ID) of your choice (for example, "my-platform-0"). The installer prepends this value to the value of the System Domain parameter to create the full platform domain.

- **Valid Values:** A string of 1–12 characters; can contain lowercase letters (a–z) and hyphens (-); must begin with a lowercase letter.
- **Default Value:** A randomly generated lowercase string.

Step 7: Configure Cluster Parameters

Common Parameters (Data and Application Clusters)

The following parameters are set for both the data and application clusters. Node references in the parameter descriptions apply to the platform's data nodes for the data cluster and application nodes for the application cluster (GKE).

Data-Cluster Parameters

On the Data Cluster page, fill in the configuration parameters, and then select Next.

# of Nodes

The number of nodes to allocate for the cluster.

Node Size

The instance type, which determines the size of the clusters' nodes.

Root Block Device Size

The size of the OS disk.

Application-Cluster Parameters

On the App Cluster page, fill in the configuration parameters, and then select Next.

Node Groups

The installer predefines a node group named, by default, "initial". You can configure the following parameters:

- **Name**—the name of the node group
- **Lifecycle**—the lifecycle of the node group (spot or on-demand)
- **# of instances**—the number of node instances in the group

- **Min. # of instances**—the minimum number node instances in the group when the group scales down
- **Max. # of instances**—the maximum number node instances in the group when the group scales up
- **# of GPUs**—the number of GPUs to be used in the group
- **Custom Labels**—user defined labels for the resources in the group
- **Custom Tags**—user defined tags for the resources in the group
- **Size**—the desired size of the node group

Step 8: Configure Cloud Parameters

On the Cloud page, fill in the configuration parameters, and then click Next. These parameters are relevant for new and existing VPC mode. There are additional parameters for New VPC mode and Existing VPC mode modes.

Step 9: Review the Settings

On the Review page, review and verify your configuration; go back and make edits, as needed; and then select Create to provision a new instance of the platform.

Step 10: Wait for Completion

It typically takes around 30–40 minutes to provision a new platform instance, regardless of the cluster sizes. You can download the provisioning logs, at any stage, by selecting Download logs from the instance's action menu.

When the installation completes, you should have a running instance of the platform in your cloud. You can use the Provazio dashboard to view the installed nodes. Then, proceed to the post-deployment steps.

Result: Thus platforms in cloud are successfully installed.

Ex No: 3 **Register No:**
Date : **Name**

Deploying existing Apps

Aim: To study about deployment of existing Apps in cloud.

Procedure:

Application Deployement

The combination of virtualization and self service facilitate application deployment. A two-tier Web application deployment using cloud.

A. Steps for deployment

The following steps comprise the deployment of the application

1. A load balancer, Web server, and database server appliances should be selected from a library of preconfigured virtual machine images.
2. Configuring each component to make a custom image should be made. Load balancer is configured accordingly; web server should be populated with the static contents by uploading them to the storage cloud where as the database servers are populated with the dynamic content of the site.
3. The developer then feeds the custom code in to the new architecture making components meet their specific requirements.
4. The developer chooses a pattern that takes the images for each layer and deploys them, handling networking, security, and scalability issues.

The secure, high-availability Web application is up and running. When the application needs to be updated, the virtual machine images can be updated, copied across the development chain, and the entire infrastructure can be redeployed. In this example, a standard set of components can be used to quickly deploy an application. With this model, enterprise business needs can be met quickly, without the need for the time-consuming, manual purchase, installation, cabling, and configuration of servers, storage, and network infrastructure

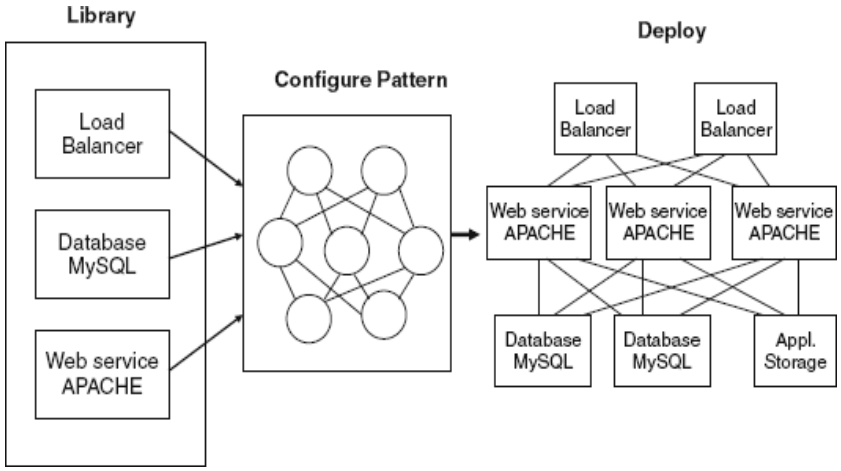


Figure. Deployment Strategy on Cloud for two tier architecture

B. Deployment on azure cloud

1) Step-1

Initially start visual studio in the administrator mode then go to file select new file. Select cloud service from project types and from template select web cloud service. In the solution explorer double click on default.aspx. Develop and press f5 to compile and debug the application.

In the solution explorer, right click on the application and then click publish. A publish folder gets opened which contains service package file and cloud service configuration file.

2) Step-2

Log in to the windows azure portal using your windows live id to deploy the application on the cloud

3) Step-3

In the portal, click on the hosted services, storage accountsand CDN

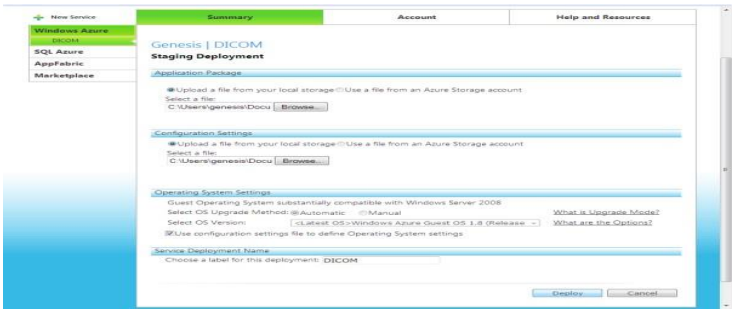
Click new hosted service. Select a subscription that will beused for application

4) Step-4

Enter the name of the application, enter URL for yourapplication, and then choose a region from the list of regions.

Select deploy to stage environment.

Ensure that Start after successful deployment is checked. Specify a name for the deployment.



5) Step-5

Figure 5. staging deployment

6) Step-6

For Package location**,** click the corresponding **Browse locally…** button, navigate to the folder where your <*Your Project Name*>.cspkg file is, and select the file.

For Configuration file, click the corresponding **Browse locally…** button, navigate to the folder where your Service Configuration.cscfg is, and select the file.

7) Step-7

Click OK. You will receive a warning after you click OK because there is only one instance of the web role defined for your application (this setting is contained in the Service Configuration. Cscfg file). For purposes of this walk-through, override the warning by clicking yes, but realize that you likely will want more than one instance of a web role for a robust application.

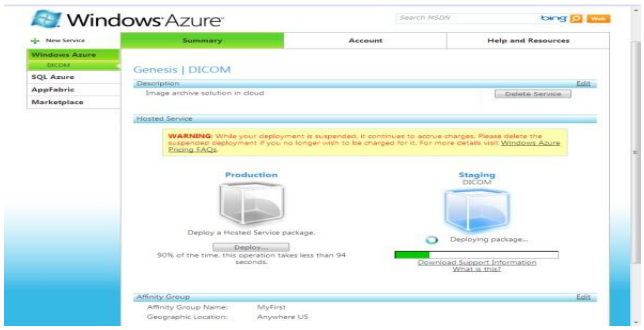


Figure 6. Staging Of An Application

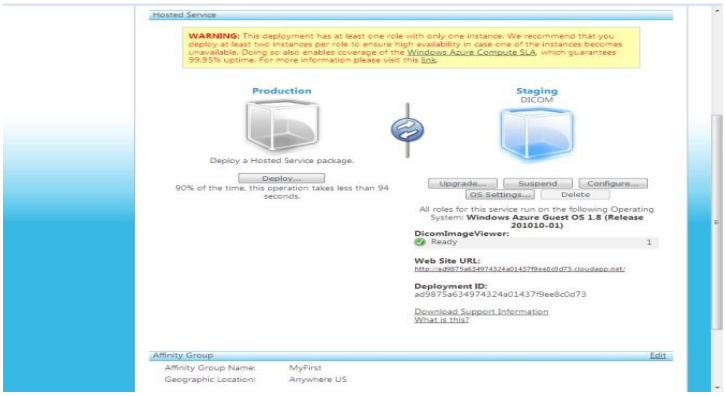


Figure 7. Final deployement screen of application on azure

We can monitor the status of the deployment in the Windows Azure management portal by navigating to the Hosted Services section

Result:

Hence, we have successfully deployed using existing app in Cloud

Ex No: 4 **Register No:**
Date : **Name:**

To create a Drop Box using Google APP

Aim:
To create a drop box using GoogleAPP

Procedure:

How to create and share Google Docs, Sheets, and Slides in Dropbox

Dropbox for Google Workspace lets you create, organize, and share Google Docs, Sheets, and Slides on dropbox.com.

Any Google Docs, Sheets, and Slides created in Dropbox save to your Dropbox account and count toward your storage space. Changes made to these Google Docs, Sheets, and Slides automatically save back to your Dropbox account. They do not save back to your Google Drive or Google account in any way.

Create Google Docs, Sheets, and Slides on dropbox.com

1. Sign in to dropbox.com.
2. Click the folder you'd like to store your file in.
3. Click Create.
4. Hover over Document, Spreadsheet, or Presentation depending on the type of file you'd like to create.
5. Click Google Docs, Google Sheets, or Google Slides.

6. The file (and any changes made to it) will save back to your Dropbox account.

Open Google Docs, Sheets, and Slides on the Dropbox mobile app

On the Dropbox mobile app, you can open previews of Google Docs, Sheets, and Slides and save them for offline viewing, but you can't create or edit them.

Share Google Docs, Sheets, and Slides with Dropbox

You can share Google Docs, Sheets, and Slides exactly the same way you would share any file stored in Dropbox.

You can choose to give Can edit or Can view access to your Google Docs, Sheets, and Slides, even when sharing with a link. **You can further limit access to your shared links in your file's link settings or deactivate a link after you've created it**.

Open and edit Microsoft Word, Excel, and PowerPoint files with Google

You can open and edit Microsoft Office files (Word, Excel, and PowerPoint) with Google (Docs, Sheets, and Slides) right from Dropbox. To do so:

1. Sign in to dropbox.com.
2. Hover over any Word (.docx), Excel (.xlsx), or PowerPoint (.pptx) file and click "..." (ellipsis).

Note: This doesn't apply to .doc, .xls, and .ppt files.

3. Hover over Open and click Google Docs, Google Sheets, or Google Slides.

Any changes you make to these files will automatically save back to the Microsoft Office file in Dropbox.

Result:

Hence drop box installed and files are shared using GoogleAP successfully.

Ex No: 5 **Register No:**
Date : **Name**

Transfer Datausing Google APPs

Aim: To study the principles of Transfer Datausing Google APPs.

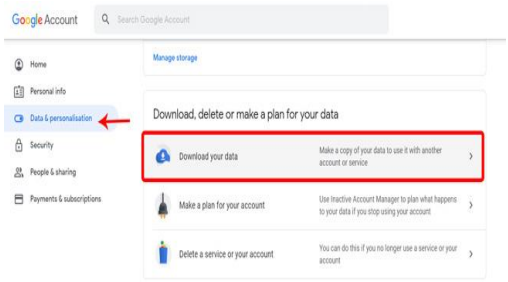
Procedure:

Transferring Data from one Google Account to another

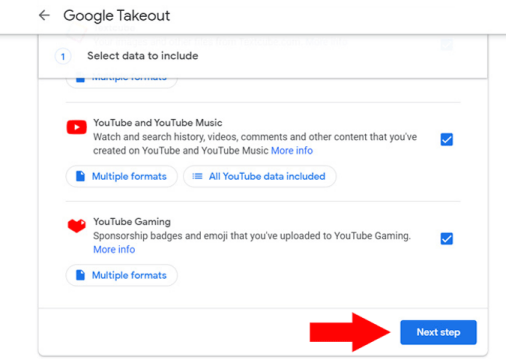
1. Create your new Google account. Take time and choose it wisely, because you know changing Gmail ID often is not an easy process.

2. Open your old Google account in a new tab. Here, we need to download the data linked with your Google account, this is known as Google Takeout Archive and you can include everything in it.

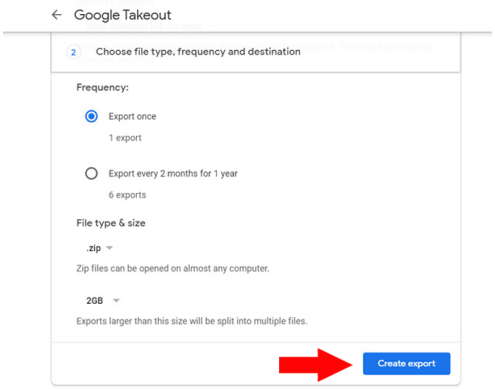
3. Log-in to your old Google account, go to the account settings > **Data & Personalisation > Download your data** in the **'Download or Delete' section**. Alternatively, you can click on this link to directly go to the final page.



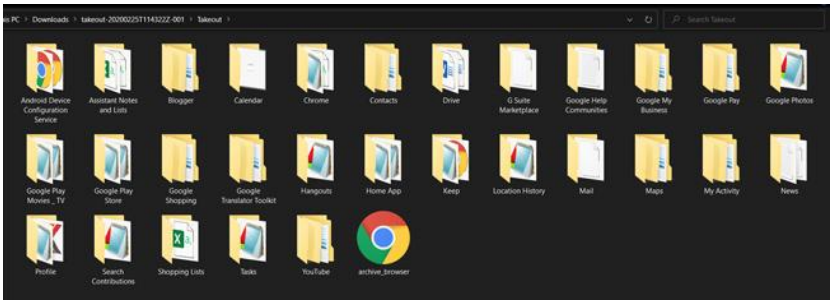
4. This page will show all the data linked to your old account ranging from auto-fill, location history, shopping list, to your contacts. **Check all the data** you want to transfer and **click the 'next step' button**.



5. **Choose file type 'Zip',** select the download destination, and **click Create export**. It may even take days if you have a lot of data on Google.



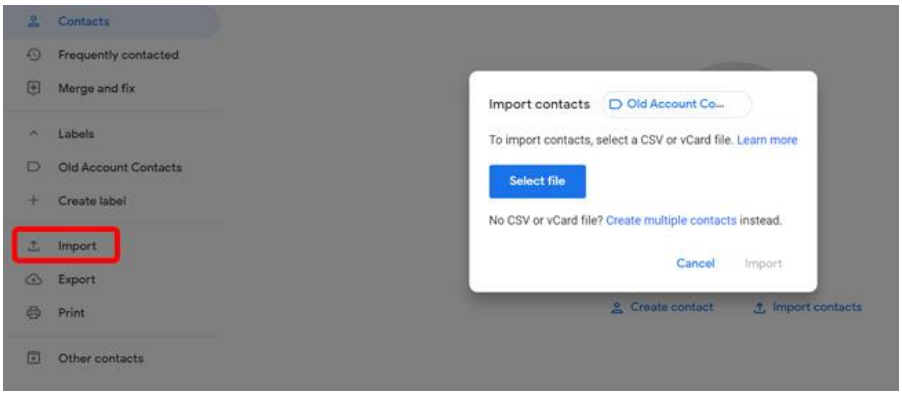
6. Once the export is completed, **download and extract the zip file** on your computer.
7. This Google Takeout Archive has all the data you need to seamlessly migrate to a new Google account.
8. Once you have extracted the zip, you can find the Google Takeout Archive folder like this.



9. Upload these data to the new Google account. However, since Google doesn't let you import all the data at once, so you'd need to import it to each service individually.

Import Contacts to New Google Account

1. In your new Google account, go to Google Contacts,
2. Click on import on the left sidebar. Select the **'.vcf'** file **in the Contacts folder** of the Google Takeout archive.
3. All your contacts will be imported to the new account. Easy.



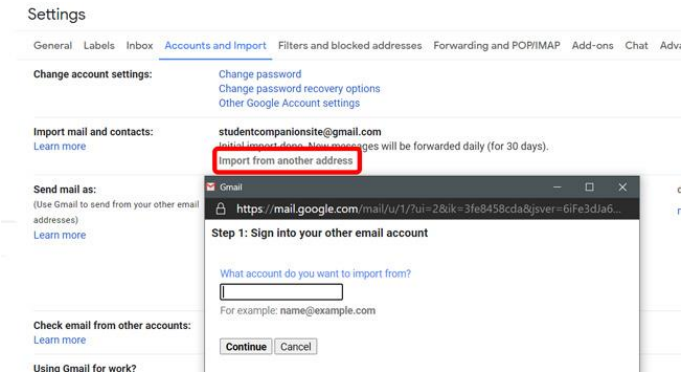
Importing Emails to the New Gmail Account

1. Technically the Google Takeout Archive has all the emails and contact information from your old account but you would have to use Thunderbird service to import all that data.

2. To import emails on your new Gmail account, open Gmail on a web browser, and log in with your new Google account.

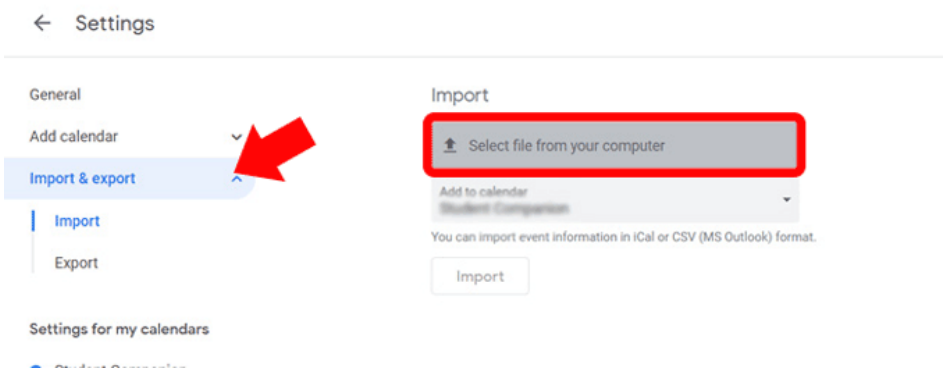
3. Click on the **Settings** button in the upper-right corner > **Accounts and Import > Import mails and contacts**. It will prompt you to enter and sign-in to your old account in the pop-up. Once you do that, it will sync all the emails, contacts, etc to the new Gmail account.

4. We will also receive emails on your old account as well as the new account for the next 30 days. You can, of course, disable this option in Settings.



Importing Calendar Events & Reminders

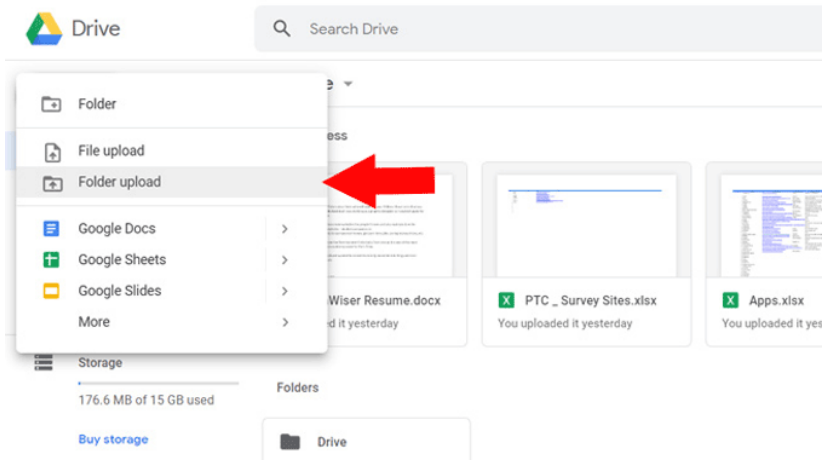
1. To import Calendar events and Reminders, go to Google Calendar on your new account > **Settings** in upper-right corner > **Import and export and select the calendar file** in the Google Takeout archive.
2. Click the Import button and all your events, reminders, birthdays, goals, etc will show up on the new account.



Importing Google Drive Files

First Method:

1. The files are downloaded as-is from the old Google Drive account and retain the hierarchy as well. This makes importing the old Drive account data to the new account effortless.
2. To import Drive files, log-in to your Drive account linked with the new Google account. **Click on New** in the top-left corner > **Folder Upload** and select the Drive folder in the Google Takeout Archive. All your old files will be uploaded to your new account.



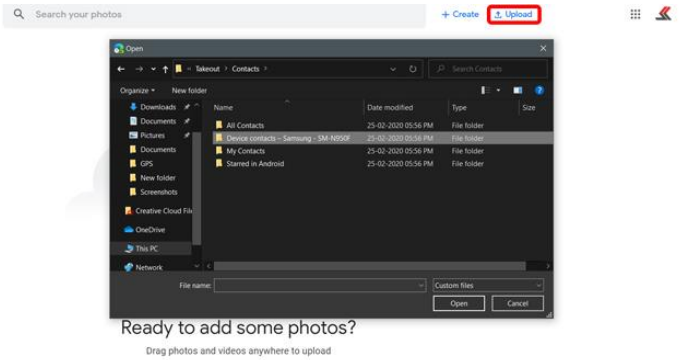
But if you are having a lot of data on the Drive, uploading them isn't an easy task. It takes a lot of time, data and your PC has to be turned on until complete data is uploaded.

Second Method:

Open Drive with your old Google account and click "Ctrl + A" to select all the files. Now click on the Share option at the top right corner. Enter your new email ID and make sure the role is selected as "Editor". Now click on send and all those files can accessible by the new account. Now again open the share menu and select the "Make Owner" option in the drop-down menu beside the added email-ID. That's it, you have total control over your data from the new account.

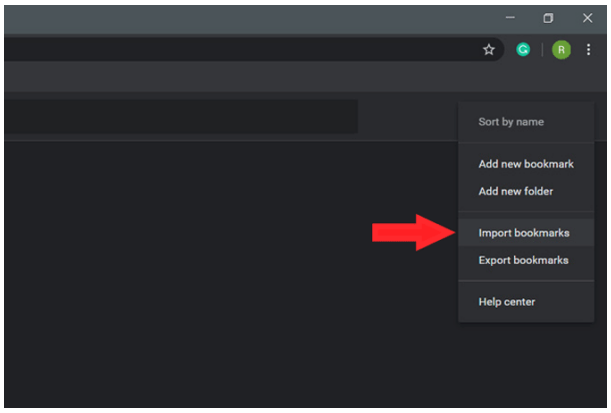
Importing Photos to Google Photos

1. Coming to the memorable Google Photos, click the upload button right at the top of the Google Photos homepage.
2. Select the Google Photos folder in the Google Takeout Archive and select all the photos enclosed in that folder.
3. It may take a lot of time to upload depending on the number of photos. Once that is done, you would have successfully migrated your Google Photos to the new account.



Importing Bookmarks on Chrome Browser

1. To import the bookmarks to your new account, just open your browser > **click the Options button on the top right corner> Bookmarks > Import bookmarks and Settings.**
2. Choose the Bookmarks Document file from the drop-down menu and upload the file in the Google Takeout Archive.
3. Google Takeout Archive only downloads bookmarks from the Chrome Browser. So if you use any other browser, export the bookmarks manually and then import it to your current browser.

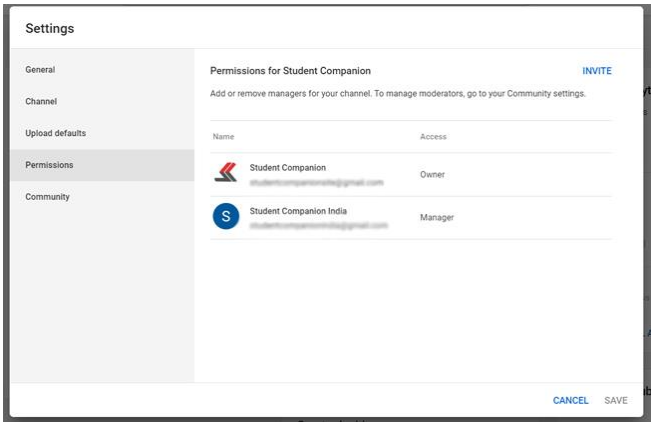


Importing Google's Autofill Passwords

1. Import Autofill data from the browser.
2. Go to **browser settings > passwords under the Autofill section** and click on import and choose the file autofill in the Chrome folder of Google Takeout archive.
3. If the import feature is not available on the Autofill page, then turn on the Password import flag in Chrome flags.

Changing your YouTube Channel

1. If you have a YouTube channel and any videos linked with your old account, then you can transfer the ownership to the new channel.
2. On the old Google account, go to **YouTube Studio > Settings > Permissions > click on Invite** and invite your new account ID as a Manager and Click Save.
3. An email will be recieved on your new account, accept the invitation and you are now the manager of your channel.
4. We can post videos, invite other people, remove and edit stuff, etc but the only caution is that you cannot delete the channel using the new account. For that, we can use old Google account.



Ex No: 6 **Register No:**
Date : **Name**

Upload and Download files and folders Using Google APPs

Aim:
To upload and download files and folders using GoogleAPPs

Procedure:

1. Upload files and folders to Google Drive
2. We can upload, view, share, and edit files with Google Drive. When you upload a file to Google Drive, it will take up space in your Drive, even if you upload to a folder owned by someone else.

Types of files

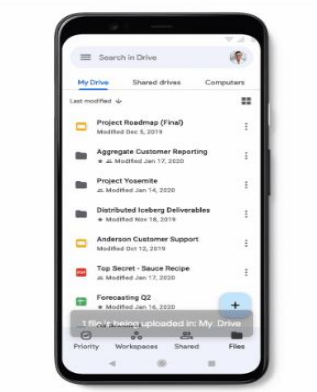
- Documents
- Images
- Audio
- Video

Important: You can upload up to 750GB a day per account.

Android ComputeriPhone & iPad

Upload & view files

1. On your Android phone or tablet, open the Google Drive app.
2. Tap Add .
3. Tap **Upload**.
4. Find and tap the files you want to upload.
5. View uploaded files in My Drive until you move them.



Convert documents into Google formats

If you want to upload Word documents, you can change a setting to convert files.

Important: You can only change Google Drive settings from your computer.

Turn mobile data usage on or off

You can choose to use your mobile data or only use Wi-Fi to transfer files.

1. On your Android phone or tablet, open the Google Drive app.
2. At the top right, click Menu ☰ › **Settings**.
3. Under "Data usage," turn **Transfer files only over Wi-Fi** on or off.

Result:

Hence documents and files are upload and download using GoogleAPPs successfully.

Ex No: 7 **Register No:**
Date : **Name**

Encryption and Decryption of Text

Aim:

To encrypt and decrypt for any given Text.

Procedure:

```
#include<stdio.h>

int main()
{
char message[100], ch;
int i, key;
printf("Enter a message to encrypt: ");
gets(message);
printf("Enter key: ");
scanf("%d", &key);
for(i = 0; message[i] != '\0'; ++i){
ch = message[i];
if(ch >= 'a' && ch <= 'z'){
ch = ch + key;
if(ch > 'z')
{
ch = ch - 'z' + 'a' - 1;
}
message[i] = ch;
}
else if(ch >= 'A' && ch <= 'Z')
{
ch = ch + key;
if(ch > 'Z'){
ch = ch - 'Z' + 'A' - 1;
}
message[i] = ch;
}
}
printf("Encrypted message: %s", message);
return 0;
}
```

OUTPUT

Enter a message to encrypt: SRMIST
Enter key: 2
Encrypted message: UTOKUV

Result:

Hence, any given text can be encrypted and decrypted successfully.

Ex No: 8 **Register No:**
Date : **Name**

Create a datacenter with one host and run one cloudlet on it.

Aim:

To create a datacenter with one host and run one cloudlet on it.

Procedure:

Step 1: Extract cloudsim 6.0 and cloudsim 3.0 to files.
Step 2: Open NetBeansIDE and open new file project and select javapplication and next select the project name and finish the initial process.
Step 3: Now by right-clicking the JavaApplication1 and select new option then create java class.
Step 4: Enter the class name of your program and finish the process.
Step5: Code your program in the workspace created with your class name.
Step 6: Right click on to the LIBRARIES on the left side and click on the ADD JAR/FOLDER.
Step 7: Add the jar files from the cloudsim 3.0 and open the file.
Step 8: Run the program your desired output will be given below.

Program:

```
package org.cloudbus.cloudsim.examples;

/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
```

```
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.VmSchedulerTimeShared;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

/**
 * A simple example showing how to create a datacenter with one host and run one
 * cloudlet on it.
 */
public class CloudSimExample1 {

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

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

	/**
	 * Creates main() to run this example.
	 *
	 * @param args the args
	 */
	@SuppressWarnings("unused")
	public static void main(String[] args) {

		Log.printLine("Starting CloudSimExample1...");

		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
Datacenter datacenter0 = createDatacenter("Datacenter_0");

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

// Fourth step: Create one virtual machine
vmlist = new ArrayList<Vm>();

// VM description
int vmid = 0;
int mips = 1000;
long size = 10000; // image size (MB)
int ram = 512; // vm memory (MB)
long bw = 1000;
int pesNumber = 1; // number of cpus
String vmm = "Xen"; // VMM name

// create VM
Vm vm = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new
 CloudletSchedulerTimeShared());

// add the VM to the vmList
vmlist.add(vm);

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

// Fifth step: Create one Cloudlet
cloudletList = new ArrayList<Cloudlet>();

// Cloudlet properties
int id = 0;
long length = 400000;
long fileSize = 300;
long outputSize = 300;
UtilizationModel utilizationModel = new UtilizationModelFull();

Cloudlet cloudlet = new Cloudlet(id, length, pesNumber, fileSize, outputSize,
utilizationModel, utilizationModel, utilizationModel);
cloudlet.setUserId(brokerId);
cloudlet.setVmId(vmid);

// add the cloudlet to the list
cloudletList.add(cloudlet);
```

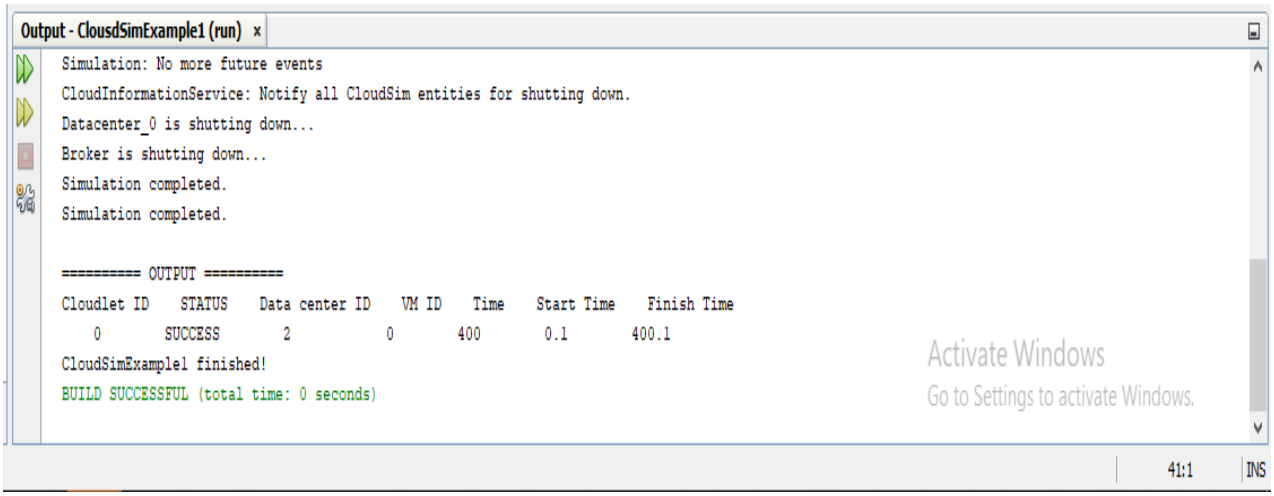
```
                // submit cloudlet list to the broker
                broker.submitCloudletList(cloudletList);

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

                CloudSim.stopSimulation();

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

                Log.printLine("CloudSimExample1 finished!");
        } catch (Exception e) {
                e.printStackTrace();
                Log.printLine("Unwanted errors happen");
        }
}

/**
 * Creates the datacenter.
 *
 * @param name the name
 *
 * @return the datacenter
 */
private static Datacenter createDatacenter(String name) {

        // Here are the steps needed to create a PowerDatacenter:
        // 1. We need to create a list to store
        // our machine
        List<Host> hostList = new ArrayList<Host>();

        // 2. A Machine contains one or more PEs or CPUs/Cores.
        // In this example, it will have only one core.
        List<Pe> peList = new ArrayList<Pe>();

        int mips = 1000;

        // 3. Create PEs and add these into a list.
        peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS
                                                              Rating

        // 4. Create Host with its id and list of PEs and add them to the list
        // of machines
        int hostId = 0;
```

```
        int ram = 2048; // host memory (MB)
        long storage = 1000000; // host storage
        int bw = 10000;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList,
                        new VmSchedulerTimeShared(peList)
                )
        ); // This is our machine

        // 5. Create a DatacenterCharacteristics object that stores the
        // properties of a data center: architecture, OS, list of
        // Machines, allocation policy: time- or space-shared, time zone
        // and its price (G$/Pe time unit).
        String arch = "x86"; // system architecture
        String os = "Linux"; // operating system
        String vmm = "Xen";
        double time_zone = 10.0; // time zone this resource located
        double cost = 3.0; // the cost of using processing in this resource
        double costPerMem = 0.05; // the cost of using memory in this resource
        double costPerStorage = 0.001; // the cost of using storage in this
                                                                        // resource
        double costPerBw = 0.0; // the cost of using bw in this resource
        LinkedList<Storage> storageList = new LinkedList<Storage>(); // we are not adding
                                                                        SAN

// devices by now

        DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
                        arch, os, vmm, hostList, time_zone, cost, costPerMem,
                        costPerStorage, costPerBw);

        // 6. Finally, we need to create a PowerDatacenter object.
        Datacenter datacenter = null;
        try {
                datacenter = new Datacenter(name, characteristics, new
               VmAllocationPolicySimple(hostList), storageList, 0);
        } catch (Exception e) {
                e.printStackTrace();
        }

        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
/**
 * Creates the broker.
 *
 * @return the datacenter broker
 */
private static DatacenterBroker createBroker() {
	DatacenterBroker broker = null;
	try {
		broker = new DatacenterBroker("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()));
                        }
                }
        }

}
```

Output



Result :

Hence, a datacenter has been created with one host with one cloudlet has been added successfully.

Ex No: 9(a) **Register No:**
Date : **Name :**

To create a datacenter with one host and run two cloudlet on it

Aim:

To create a datacenter with one host and run two cloudlets on it

Procedure:

Step 1: Extract cloudsim 6.0 and cloudsim 3.0 to files.
Step 2: Open NetBeansIDE and open new file project and select javapplication and next select the project name and finish the initial process.
Step 3: Now by right-clicking the JavaApplication1 and select new option then create java class.
Step 4: Enter the class name of your program and finish the process.
Step 5: Code your program in the workspace created with your class name.
Step 6: Right click on to the LIBRARIES on the left side and click on the ADD JAR/FOLDER.
Step 7: Add the jar files from the cloudsim 3.0 and open the file.
Step 8: Run the program your desired output will be given below**.**

Program:

```
/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */

package org.cloudbus.cloudsim.examples;


import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
```

```
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.VmSchedulerTimeShared;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;


/**
 * A simple example showing how to create
 * a datacenter with one host and run two
 * cloudlets on it. The cloudlets run in
 * VMs with the same MIPS requirements.
 * The cloudlets will take the same time to
 * complete the execution.
 */
public class CloudSimExample2 {

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

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

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

		Log.printLine("Starting CloudSimExample2...");

	    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
DatacenterBroker broker = createBroker();
int brokerId = broker.getId();

//Fourth step: Create one virtual machine
vmlist = new ArrayList<Vm>();

//VM description
int vmid = 0;
int mips = 250;
long size = 10000; //image size (MB)
int ram = 512; //vm memory (MB)
long bw = 1000;
int pesNumber = 1; //number of cpus
String vmm = "Xen"; //VMM name

//create two VMs
Vm vm1 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new
 CloudletSchedulerTimeShared());

vmid++;
Vm vm2 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());

//add the VMs to the vmList
vmlist.add(vm1);
vmlist.add(vm2);

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


//Fifth step: Create two Cloudlets
cloudletList = new ArrayList<Cloudlet>();

//Cloudlet properties
int id = 0;
pesNumber=1;
long length = 250000;
long fileSize = 300;
```

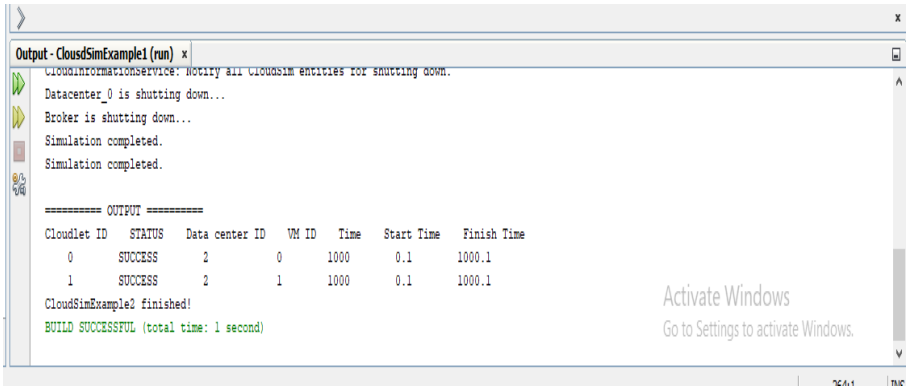
```
        long outputSize = 300;
        UtilizationModel utilizationModel = new UtilizationModelFull();

        Cloudlet cloudlet1 = new Cloudlet(id, length, pesNumber, fileSize, outputSize,
        utilizationModel, utilizationModel, utilizationModel);
        cloudlet1.setUserId(brokerId);

        id++;
        Cloudlet cloudlet2 = new Cloudlet(id, length, pesNumber, fileSize, outputSize,
        utilizationModel, utilizationModel, utilizationModel);
        cloudlet2.setUserId(brokerId);

        //add the cloudlets to the list
        cloudletList.add(cloudlet1);
        cloudletList.add(cloudlet2);

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


        //bind the cloudlets to the vms. This way, the broker
        // will submit the bound cloudlets only to the specific VM
        broker.bindCloudletToVm(cloudlet1.getCloudletId(),vm1.getId());
        broker.bindCloudletToVm(cloudlet2.getCloudletId(),vm2.getId());

        // 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("CloudSimExample2 finished!");
    }
    catch (Exception e) {
       e.printStackTrace();
       Log.printLine("The simulation has been terminated due to an unexpected error");
    }
}

        private static Datacenter createDatacenter(String name){

    // Here are the steps needed to create a PowerDatacenter:
```

```
// 1. We need to create a list to store
    //    our machine
    List<Host> hostList = new ArrayList<Host>();

// 2. A Machine contains one or more PEs or CPUs/Cores.
    // In this example, it will have only one core.
    List<Pe> peList = new ArrayList<Pe>();

    int mips = 1000;

// 3. Create PEs and add these into a list.
peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating

//4. Create Host with its id and list of PEs and add them to the list of machines
int hostId=0;
int ram = 2048; //host memory (MB)
long storage = 1000000; //host storage
int bw = 10000;

hostList.add(
                    new Host(
                            hostId,
                            new RamProvisionerSimple(ram),
                            new BwProvisionerSimple(bw),
                            storage,
                            peList,
                            new VmSchedulerTimeShared(peList)
                    )
            ); // This is our machine


// 5. Create a DatacenterCharacteristics object that stores the
//    properties of a data center: architecture, OS, list of
//    Machines, allocation policy: time- or space-shared, time zone
//    and its price (G$/Pe time unit).
String arch = "x86";      // system architecture
String os = "Linux";          // operating system
String vmm = "Xen";
double time_zone = 10.0;         // time zone this resource located
double cost = 3.0;              // the cost of using processing in this resource
double costPerMem = 0.05;                // the cost of using memory in this resource
double costPerStorage = 0.001;       // the cost of using storage in this resource
double costPerBw = 0.0;                  // the cost of using bw in this resource
LinkedList<Storage> storageList = new LinkedList<Storage>();       //we are not adding
SAN devices by now

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
```

```
        arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);


    // 6. Finally, we need to create a PowerDatacenter object.
    Datacenter datacenter = null;
    try {
        datacenter = new Datacenter(name, characteristics, new
       VmAllocationPolicySimple(hostList), storageList, 0);
    } catch (Exception e) {
        e.printStackTrace();
    }

    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 DatacenterBroker createBroker(){

        DatacenterBroker broker = null;
    try {
                broker = new DatacenterBroker("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()));
                }
             }

          }
}
```

Output:



Result:

Hence, a datacenter has been created with one host with two cloudlets has been added successfully.

Ex No: 9(b) **Register No:**
Date : **Name :**

To create a datacenter with two hosts and run two cloudlets

Aim:

To create a datacenter with two hosts and run two cloudlets on it

Procedure:

Step 1: Extract cloudsim 6.0 and cloudsim 3.0 to files.
Step 2: Open NetBeansIDE and open new file project and select javapplication and next select the project name and finish the initial process.
Step 3: Now by right-clicking the JavaApplication1 and select new option then create java class.
Step 4: Enter the class name of your program and finish the process.
Step 5: Code your program in the workspace created with your class name.
Step 6: Right click on to the LIBRARIES on the left side and click on the ADD JAR/FOLDER.
Step 7: Add the jar files from the cloudsim 3.0 and open the file.
Step 8: Run the program your desired output will be given below.

Program:

```
/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */

package org.cloudbus.cloudsim.examples;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
```

```
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.VmSchedulerTimeShared;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;


/**
 * A simple example showing how to create
 * a datacenter with two hosts and run two
 * cloudlets on it. The cloudlets run in
 * VMs with different MIPS requirements.
 * The cloudlets will take different time
 * to complete the execution depending on
 * the requested VM performance.
 */
public class CloudSimExample3 {

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

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

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

		Log.printLine("Starting CloudSimExample3...");

		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
DatacenterBroker broker = createBroker();
int brokerId = broker.getId();

//Fourth step: Create one virtual machine
vmlist = new ArrayList<Vm>();

//VM description
int vmid = 0;
int mips = 250;
long size = 10000; //image size (MB)
int ram = 2048; //vm memory (MB)
long bw = 1000;
int pesNumber = 1; //number of cpus
String vmm = "Xen"; //VMM name


//create two VMs
Vm vm1 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());

//the second VM will have twice the priority of VM1 and so will receive twice
 CPU time
vmid++;
Vm vm2 = new Vm(vmid, brokerId, mips * 2, pesNumber, ram, bw, size, vmm,
 new CloudletSchedulerTimeShared());

//add the VMs to the vmList
vmlist.add(vm1);
vmlist.add(vm2);

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


//Fifth step: Create two Cloudlets
cloudletList = new ArrayList<Cloudlet>();

//Cloudlet properties
```

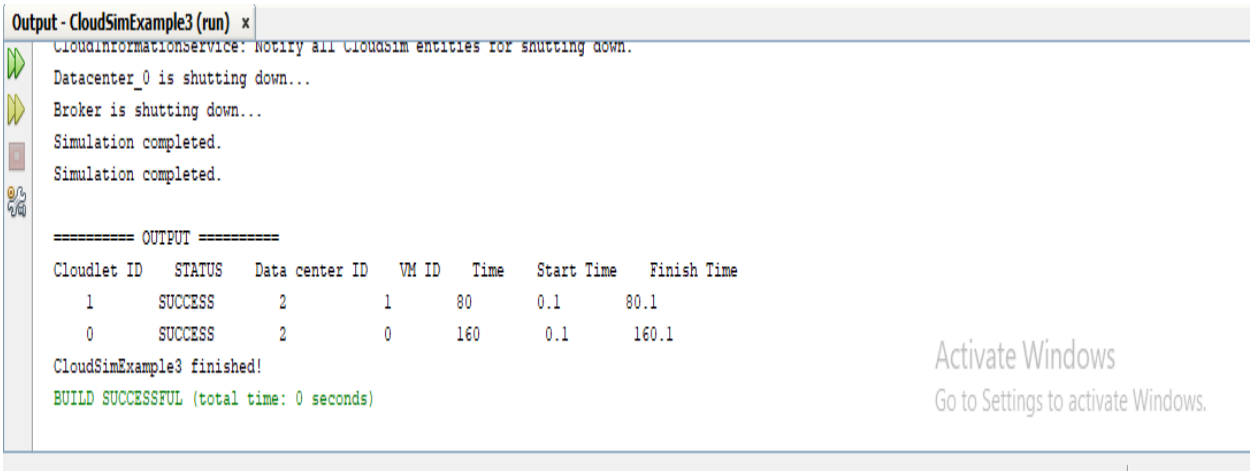
```
                int id = 0;
                long length = 40000;
                long fileSize = 300;
                long outputSize = 300;
                UtilizationModel utilizationModel = new UtilizationModelFull();

                Cloudlet cloudlet1 = new Cloudlet(id, length, pesNumber, fileSize, outputSize,
                  utilizationModel, utilizationModel, utilizationModel);
                cloudlet1.setUserId(brokerId);

                id++;
                Cloudlet cloudlet2 = new Cloudlet(id, length, pesNumber, fileSize, outputSize,
                 utilizationModel, utilizationModel, utilizationModel);
                cloudlet2.setUserId(brokerId);

                //add the cloudlets to the list
                cloudletList.add(cloudlet1);
                cloudletList.add(cloudlet2);

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


                //bind the cloudlets to the vms. This way, the broker
                // will submit the bound cloudlets only to the specific VM
                broker.bindCloudletToVm(cloudlet1.getCloudletId(),vm1.getId());
                broker.bindCloudletToVm(cloudlet2.getCloudletId(),vm2.getId());

                // 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("CloudSimExample3 finished!");
        }
        catch (Exception e) {
                e.printStackTrace();
                Log.printLine("The simulation has been terminated due to an unexpected error");
        }
}
```

```
private static Datacenter createDatacenter(String name){

        // Here are the steps needed to create a PowerDatacenter:
        // 1. We need to create a list to store
        //    our machine
        List<Host> hostList = new ArrayList<Host>();

        // 2. A Machine contains one or more PEs or CPUs/Cores.
        // In this example, it will have only one core.
        List<Pe> peList = new ArrayList<Pe>();

        int mips = 1000;

        // 3. Create PEs and add these into a list.
peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating

        //4. Create Hosts with its id and list of PEs and add them to the list of machines
        int hostId=0;
        int ram = 2048; //host memory (MB)
        long storage = 1000000; //host storage
        int bw = 10000;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList,
                        new VmSchedulerTimeShared(peList)
                )
        ); // This is our first machine

        //create another machine in the Data center
        List<Pe> peList2 = new ArrayList<Pe>();

        peList2.add(new Pe(0, new PeProvisionerSimple(mips)));

        hostId++;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
```

```
                        peList2,
                        new VmSchedulerTimeShared(peList2)
                    )
            ); // This is our second machine



            // 5. Create a DatacenterCharacteristics object that stores the
            //    properties of a data center: architecture, OS, list of
            //    Machines, allocation policy: time- or space-shared, time zone
            //    and its price (G$/Pe time unit).
            String arch = "x86";      // system architecture
            String os = "Linux";          // operating system
            String vmm = "Xen";
            double time_zone = 10.0;         // time zone this resource located
            double cost = 3.0;              // the cost of using processing in this resource
            double costPerMem = 0.05;                   // the cost of using memory in this resource
            double costPerStorage = 0.001;  // the cost of using storage in this resource
            double costPerBw = 0.0;                                 // the cost of using bw in this resource
            LinkedList<Storage> storageList = new LinkedList<Storage>();  //we are not adding
                                                                              SAN devices by now

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
      arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);

            // 6. Finally, we need to create a PowerDatacenter object.
            Datacenter datacenter = null;
            try {
                    datacenter = new Datacenter(name, characteristics, new
           VmAllocationPolicySimple(hostList), storageList, 0);
            } catch (Exception e) {
                    e.printStackTrace();
            }

            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 DatacenterBroker createBroker(){

            DatacenterBroker broker = null;
            try {
                    broker = new DatacenterBroker("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()));
                }
            }

        }
}
```

Output:



Result:

Hence, a datacenter has been created with two hosts and runs with two cloudlets successfully.

Ex No: 10 **Register No:**
Date : **Name :**

To create two datacenters with one host and run two cloudlets

Aim:

To create two datacenters with one host and run two cloudlets on them

Procedure:

Step 1: Extract cloudsim 6.0 and cloudsim 3.0 to files.
Step 2: Open NetBeansIDE and open new file project and select javapplication and next select the project name and finish the initial process.
Step 3: Now by right-clicking the JavaApplication1 and select new option then create java class.
Step 4: Enter the class name of your program and finish the process.
Step 5: Code your program in the workspace created with your class name.
Step 6: Right click on to the LIBRARIES on the left side and click on the ADD JAR/FOLDER.
Step 7: Add the jar files from the cloudsim 3.0 and open the file.
Step 8: Run the program your desired output will be given below.

Program:

```
/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */
package org.cloudbus.cloudsim.examples;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
```

```
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.VmSchedulerSpaceShared;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

/**
 * A simple example showing how to create
 * two datacenters with one host each and
 * run two cloudlets on them.
 */
public class CloudSimExample4 {

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

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

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

		Log.printLine("Starting CloudSimExample4...");

		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 GridSim 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");
@SuppressWarnings("unused")
Datacenter datacenter1 = createDatacenter("Datacenter_1");

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

//Fourth step: Create one virtual machine
vmlist = new ArrayList<Vm>();

//VM description
int vmid = 0;
int mips = 250;
long size = 10000; //image size (MB)
int ram = 512; //vm memory (MB)
long bw = 1000;
int pesNumber = 1; //number of cpus
String vmm = "Xen"; //VMM name

//create two VMs
Vm vm1 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());

vmid++;
Vm vm2 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());

//add the VMs to the vmList
vmlist.add(vm1);
vmlist.add(vm2);

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


//Fifth step: Create two Cloudlets
cloudletList = new ArrayList<Cloudlet>();

//Cloudlet properties
int id = 0;
long length = 40000;
long fileSize = 300;
long outputSize = 300;
UtilizationModel utilizationModel = new UtilizationModelFull();

Cloudlet cloudlet1 = new Cloudlet(id, length, pesNumber, fileSize, outputSize,
```

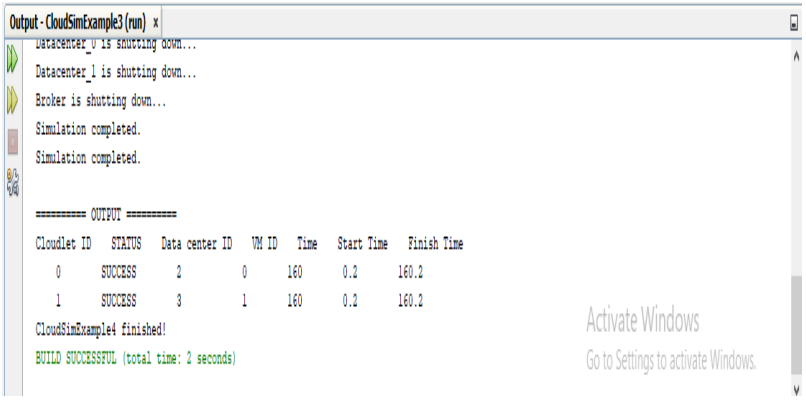
```
                utilizationModel, utilizationModel, utilizationModel);
                cloudlet1.setUserId(brokerId);

                id++;
                Cloudlet cloudlet2 = new Cloudlet(id, length, pesNumber, fileSize, outputSize,
                utilizationModel, utilizationModel, utilizationModel);
                cloudlet2.setUserId(brokerId);

                //add the cloudlets to the list
                cloudletList.add(cloudlet1);
                cloudletList.add(cloudlet2);

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


                //bind the cloudlets to the vms. This way, the broker
                // will submit the bound cloudlets only to the specific VM
                broker.bindCloudletToVm(cloudlet1.getCloudletId(),vm1.getId());
                broker.bindCloudletToVm(cloudlet2.getCloudletId(),vm2.getId());

                // 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("CloudSimExample4 finished!");
        }
        catch (Exception e) {
                e.printStackTrace();
                Log.printLine("The simulation has been terminated due to an unexpected error");
        }
}

private static Datacenter createDatacenter(String name){

        // Here are the steps needed to create a PowerDatacenter:
        // 1. We need to create a list to store
        //    our machine
        List<Host> hostList = new ArrayList<Host>();
```

```
// 2. A Machine contains one or more PEs or CPUs/Cores.
// In this example, it will have only one core.
List<Pe> peList = new ArrayList<Pe>();

int mips = 1000;

// 3. Create PEs and add these into a list.
peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS
Rating

//4. Create Host with its id and list of PEs and add them to the list of machines
int hostId=0;
int ram = 2048; //host memory (MB)
long storage = 1000000; //host storage
int bw = 10000;


//in this example, the VMAllocatonPolicy in use is SpaceShared. It means that only one
   VM
//is allowed to run on each Pe. As each Host has only one Pe, only one VM can run on
each Host.
hostList.add(
        new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
                peList,
                new VmSchedulerSpaceShared(peList)
        )
); // This is our first machine

// 5. Create a DatacenterCharacteristics object that stores the
//    properties of a data center: architecture, OS, list of
//    Machines, allocation policy: time- or space-shared, time zone
//    and its price (G$/Pe time unit).
String arch = "x86";      // system architecture
String os = "Linux";          // operating system
String vmm = "Xen";
double time_zone = 10.0;         // time zone this resource located
double cost = 3.0;              // the cost of using processing in this resource
double costPerMem = 0.05;                   // the cost of using memory in this resource
double costPerStorage = 0.001;  // the cost of using storage in this resource
double costPerBw = 0.0;                                 // the cost of using bw in this resource
LinkedList<Storage> storageList = new LinkedList<Storage>();  //we are not adding
                                                                SAN devices by now
```

```
        DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
                arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);


                // 6. Finally, we need to create a PowerDatacenter object.
                Datacenter datacenter = null;
                try {
                        datacenter = new Datacenter(name, characteristics, new
               VmAllocationPolicySimple(hostList), storageList, 0);
                } catch (Exception e) {
                        e.printStackTrace();
                }

                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 DatacenterBroker createBroker(){

                DatacenterBroker broker = null;
                try {
                        broker = new DatacenterBroker("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()));
                        }
                }

        }
}
```

Output:



Result:

Hence, a datacenter has been created with two hosts and runs with two cloudlets successfully.

Ex No: 11 **Register No:**
Date : **Name:**

To create two datacenters with one host and run cloudlets of two users

Aim:

To create two datacenters with one host and run cloudlets of two users on them.

Procedure:

Step 1: Extract cloudsim 6.0 and cloudsim 3.0 to files.

Step 2: Open NetBeansIDE and open new file project and select javapplication and next select the project name and finish the initial process.
Step 3: Now by right-clicking the JavaApplication1 and select new option then create java class.
Step 4: Enter the class name of your program and finish the process.
Step 5: Code your program in the workspace created with your class name.
Step 6: Right click on to the LIBRARIES on the left side and click on the ADD JAR/FOLDER.
Step 7: Add the jar files from the cloudsim 3.0 and open the file.
Step 8: Run the program your desired output will be given below.

Program:

```
/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */

package org.cloudbus.cloudsim.examples;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.DatacenterBroker;
```

```
import org.cloudbus.cloudsim.DatacenterCharacteristics;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.VmSchedulerSpaceShared;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;


/**
 * A simple example showing how to create
 * two datacenters with one host each and
 * run cloudlets of two users on them.
 */
public class CloudSimExample5 {

	/** The cloudlet lists. */
	private static List<Cloudlet> cloudletList1;
	private static List<Cloudlet> cloudletList2;

	/** The vmlists. */
	private static List<Vm> vmlist1;
	private static List<Vm> vmlist2;

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

		Log.printLine("Starting CloudSimExample5...");

		try {
			// First step: Initialize the CloudSim package. It should be called
			// before creating any entities.
			int num_user = 2;   // 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");
@SuppressWarnings("unused")
Datacenter datacenter1 = createDatacenter("Datacenter_1");

//Third step: Create Brokers
DatacenterBroker broker1 = createBroker(1);
int brokerId1 = broker1.getId();

DatacenterBroker broker2 = createBroker(2);
int brokerId2 = broker2.getId();

//Fourth step: Create one virtual machine for each broker/user
vmlist1 = new ArrayList<Vm>();
vmlist2 = new ArrayList<Vm>();

//VM description
int vmid = 0;
int mips = 250;
long size = 10000; //image size (MB)
int ram = 512; //vm memory (MB)
long bw = 1000;
int pesNumber = 1; //number of cpus
String vmm = "Xen"; //VMM name

//create two VMs: the first one belongs to user1
Vm vm1 = new Vm(vmid, brokerId1, mips, pesNumber, ram, bw, size, vmm,
new CloudletSchedulerTimeShared());

//the second VM: this one belongs to user2
Vm vm2 = new Vm(vmid, brokerId2, mips, pesNumber, ram, bw, size, vmm,
new CloudletSchedulerTimeShared());

//add the VMs to the vmlists
vmlist1.add(vm1);
vmlist2.add(vm2);

//submit vm list to the broker
broker1.submitVmList(vmlist1);
broker2.submitVmList(vmlist2);

//Fifth step: Create two Cloudlets
cloudletList1 = new ArrayList<Cloudlet>();
```

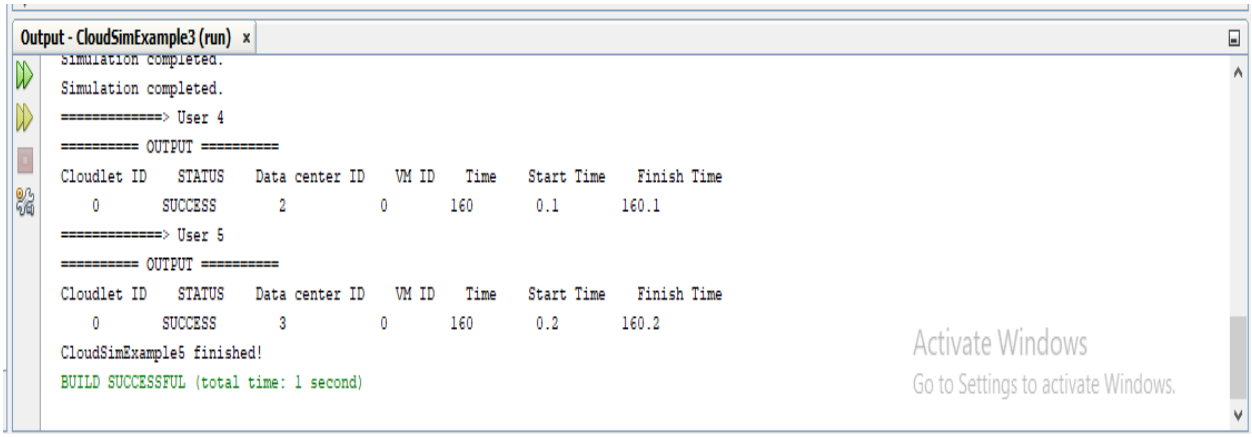
```
                cloudletList2 = new ArrayList<Cloudlet>();

                //Cloudlet properties
                int id = 0;
                long length = 40000;
                long fileSize = 300;
                long outputSize = 300;
                UtilizationModel utilizationModel = new UtilizationModelFull();

                Cloudlet cloudlet1 = new Cloudlet(id, length, pesNumber, fileSize, outputSize,
                 utilizationModel, utilizationModel, utilizationModel);
                cloudlet1.setUserId(brokerId1);

                Cloudlet cloudlet2 = new Cloudlet(id, length, pesNumber, fileSize, outputSize,
                utilizationModel, utilizationModel, utilizationModel);
                cloudlet2.setUserId(brokerId2);

                //add the cloudlets to the lists: each cloudlet belongs to one user
                cloudletList1.add(cloudlet1);
                cloudletList2.add(cloudlet2);

                //submit cloudlet list to the brokers
                broker1.submitCloudletList(cloudletList1);
                broker2.submitCloudletList(cloudletList2);

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

                // Final step: Print results when simulation is over
                List<Cloudlet> newList1 = broker1.getCloudletReceivedList();
                List<Cloudlet> newList2 = broker2.getCloudletReceivedList();

                CloudSim.stopSimulation();

                Log.print("==============> User "+brokerId1+"    ");
                printCloudletList(newList1);

                Log.print("==============> User "+brokerId2+"    ");
                printCloudletList(newList2);

                Log.printLine("CloudSimExample5 finished!");
        }
        catch (Exception e) {
                e.printStackTrace();
                Log.printLine("The simulation has been terminated due to an unexpected error");
        }
}
```

```
private static Datacenter createDatacenter(String name){

        // Here are the steps needed to create a PowerDatacenter:
        // 1. We need to create a list to store
        //    our machine
        List<Host> hostList = new ArrayList<Host>();

        // 2. A Machine contains one or more PEs or CPUs/Cores.
        // In this example, it will have only one core.
        List<Pe> peList = new ArrayList<Pe>();

        int mips=1000;

        // 3. Create PEs and add these into a list.
    peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating

        //4. Create Host with its id and list of PEs and add them to the list of machines
        int hostId=0;
        int ram = 2048; //host memory (MB)
        long storage = 1000000; //host storage
        int bw = 10000;


//in this example, the VMAllocatonPolicy in use is SpaceShared. It means that only one VM
    //is allowed to run on each Pe. As each Host has only one Pe, only one VM can run on each Host.
        hostList.add(
            new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
                peList,
                new VmSchedulerSpaceShared(peList)
            )
        ); // This is our first machine

        // 5. Create a DatacenterCharacteristics object that stores the
        //    properties of a data center: architecture, OS, list of
        //    Machines, allocation policy: time- or space-shared, time zone
        //    and its price (G$/Pe time unit).
        String arch = "x86";      // system architecture
        String os = "Linux";          // operating system
        String vmm = "Xen";
        double time_zone = 10.0;         // time zone this resource located
        double cost = 3.0;              // the cost of using processing in this resource
        double costPerMem = 0.05;                // the cost of using memory in this resource
```

```
                double costPerStorage = 0.001;  // the cost of using storage in this resource
                double costPerBw = 0.0;                         // the cost of using bw in this resource
                LinkedList<Storage> storageList = new LinkedList<Storage>();  //we are not adding
               SAN devices by now

                DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
                arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);


                // 6. Finally, we need to create a PowerDatacenter object.
                Datacenter datacenter = null;
                try {
                        datacenter = new Datacenter(name, characteristics, new
                  VmAllocationPolicySimple(hostList), storageList, 0);
                } catch (Exception e) {
                        e.printStackTrace();
                }

                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 DatacenterBroker createBroker(int id){

                DatacenterBroker broker = null;
                try {
                        broker = new DatacenterBroker("Broker"+id);
                } 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()));
                        }
                }

        }
}
```

Output:



Result:

Hence, a datacenter has been created with two hosts and runs cloudlets with two users successfully.

Ex No: 12 **Register No:**
Date : **Name :**

To pause and resume the simulation and create simulation entities dynamically

Aim:

To pause and resume the simulation and create simulation entities dynamically.

Procedure:

Step 1: Extract cloudsim 6.0 and cloudsim 3.0 to files.
Step 2: Open NetBeansIDE and open new file project and select javapplication and next select the project name and finish the initial process.
Step 3: Now by right-clicking the JavaApplication1 and select new option then create java class.
Step 4: Enter the class name of your program and finish the process.
Step 5: Code your program in the workspace created with your class name.
Step 6: Right click on to the LIBRARIES on the left side and click on the ADD JAR/FOLDER.
Step 7: Add the jar files from the cloudsim 3.0 and open the file.
Step 8: Run the program your desired output will be given below.

Program:

```
/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */
package org.cloudbus.cloudsim.examples;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
import org.cloudbus.cloudsim.Host;
```

```
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.VmSchedulerTimeShared;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

/**
 * An example showing how to pause and resume the simulation,
 * and create simulation entities (a DatacenterBroker in this example)
 * dynamically.
 */
public class CloudSimExample7 {

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

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

	private static List<Vm> createVM(int userId, int vms, int idShift) {
		//Creates a container to store VMs. This list is passed to the broker later
		LinkedList<Vm> list = new LinkedList<Vm>();

		//VM Parameters
		long size = 10000; //image size (MB)
		int ram = 512; //vm memory (MB)
		int mips = 250;
		long bw = 1000;
		int pesNumber = 1; //number of cpus
		String vmm = "Xen"; //VMM name

		//create VMs
		Vm[] vm = new Vm[vms];

		for(int i=0;i<vms;i++){
			vm[i] = new Vm(idShift + i, userId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());
			list.add(vm[i]);
		}
```

```
        return list;
    }


    private static List<Cloudlet> createCloudlet(int userId, int cloudlets, int idShift){
        // Creates a container to store Cloudlets
        LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

        //cloudlet parameters
        long length = 40000;
        long fileSize = 300;
        long outputSize = 300;
        int pesNumber = 1;
        UtilizationModel utilizationModel = new UtilizationModelFull();

        Cloudlet[] cloudlet = new Cloudlet[cloudlets];

        for(int i=0;i<cloudlets;i++){
            cloudlet[i] = new Cloudlet(idShift + i, length, pesNumber, fileSize, outputSize,
utilizationModel, utilizationModel, utilizationModel);
            // setting the owner of these Cloudlets
            cloudlet[i].setUserId(userId);
            list.add(cloudlet[i]);
        }

        return list;
    }


    ////////////////////////// STATIC METHODS ///////////////////////

    /**
     * Creates main() to run this example
     */
    public static void main(String[] args) {
        Log.printLine("Starting CloudSimExample7...");

        try {
            // First step: Initialize the CloudSim package. It should be called
            // before creating any entities.
            int num_user = 2;   // number of grid 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");
@SuppressWarnings("unused")
Datacenter datacenter1 = createDatacenter("Datacenter_1");

//Third step: Create Broker
DatacenterBroker broker = createBroker("Broker_0");
int brokerId = broker.getId();

//Fourth step: Create VMs and Cloudlets and send them to broker
vmlist = createVM(brokerId, 5, 0); //creating 5 vms
cloudletList = createCloudlet(brokerId, 10, 0); // creating 10 cloudlets

broker.submitVmList(vmlist);
broker.submitCloudletList(cloudletList);

// A thread that will create a new broker at 200 clock time
Runnable monitor = new Runnable() {
        @Override
        public void run() {
                CloudSim.pauseSimulation(200);
                while (true) {
                        if (CloudSim.isPaused()) {
                                break;
                        }
                        try {
                                Thread.sleep(100);
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }

Log.printLine("\n\n\n" + CloudSim.clock() + ": The simulation is paused for 5 sec \n\n");

                try {
                        Thread.sleep(5000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }

                DatacenterBroker broker = createBroker("Broker_1");
                int brokerId = broker.getId();

                //Create VMs and Cloudlets and send them to broker
```

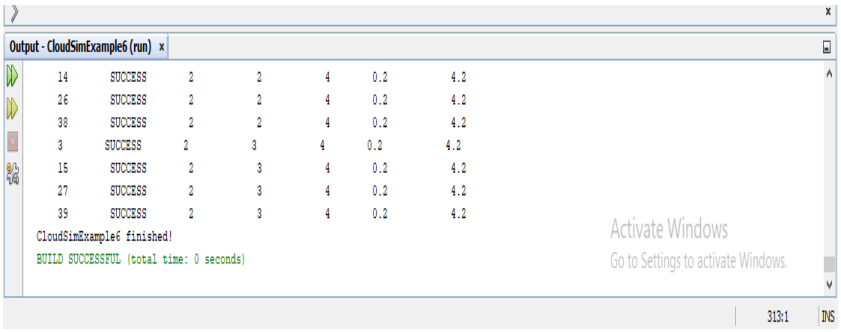
```
                        vmlist = createVM(brokerId, 5, 100); //creating 5 vms
                cloudletList = createCloudlet(brokerId, 10, 100); // creating 10 cloudlets

                        broker.submitVmList(vmlist);
                        broker.submitCloudletList(cloudletList);

                        CloudSim.resumeSimulation();
                    }
                };

                new Thread(monitor).start();
                Thread.sleep(1000);

                // Fifth 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("CloudSimExample7 finished!");
        }
        catch (Exception e)
        {
                e.printStackTrace();
                Log.printLine("The simulation has been terminated due to an unexpected error");
        }
}

private static Datacenter createDatacenter(String name){

        // Here are the steps needed to create a PowerDatacenter:
        // 1. We need to create a list to store one or more
        //    Machines
        List<Host> hostList = new ArrayList<Host>();

        // 2. A Machine contains one or more PEs or CPUs/Cores. Therefore, should
        //    create a list to store these PEs before creating
        //    a Machine.
        List<Pe> peList1 = new ArrayList<Pe>();

        int mips = 1000;

        // 3. Create PEs and add these into the list.
```

```
        //for a quad-core machine, a list of 4 PEs is required:
peList1.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating
        peList1.add(new Pe(1, new PeProvisionerSimple(mips)));
        peList1.add(new Pe(2, new PeProvisionerSimple(mips)));
        peList1.add(new Pe(3, new PeProvisionerSimple(mips)));

        //Another list, for a dual-core machine
        List<Pe> peList2 = new ArrayList<Pe>();

        peList2.add(new Pe(0, new PeProvisionerSimple(mips)));
        peList2.add(new Pe(1, new PeProvisionerSimple(mips)));

        //4. Create Hosts with its id and list of PEs and add them to the list of machines
        int hostId=0;
        int ram = 16384; //host memory (MB)
        long storage = 1000000; //host storage
        int bw = 10000;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList1,
                        new VmSchedulerTimeShared(peList1)
                )
        ); // This is our first machine

        hostId++;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList2,
                        new VmSchedulerTimeShared(peList2)
                )
        ); // Second machine

        // 5. Create a DatacenterCharacteristics object that stores the
        //    properties of a data center: architecture, OS, list of
        //    Machines, allocation policy: time- or space-shared, time zone
        //    and its price (G$/Pe time unit).
        String arch = "x86";      // system architecture
```

```
		String os = "Linux";		// operating system
		String vmm = "Xen";
		double time_zone = 10.0;		// time zone this resource located
		double cost = 3.0;			// the cost of using processing in this resource
		double costPerMem = 0.05;		// the cost of using memory in this resource
		double costPerStorage = 0.1;	// the cost of using storage in this resource
		double costPerBw = 0.1;			// the cost of using bw in this
resourceLinkedList<Storage> storageList = new LinkedList<Storage>(); //we are not adding SAN
devices by now

		DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
	  arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);


		// 6. Finally, we need to create a PowerDatacenter object.
		Datacenter datacenter = null;
		try {
			datacenter = new Datacenter(name, characteristics, new
VmAllocationPolicySimple(hostList), storageList, 0);
		} catch (Exception e) {
			e.printStackTrace();
		}

		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 DatacenterBroker createBroker(String name){

		DatacenterBroker broker = null;
		try {
			broker = new DatacenterBroker(name);
		} 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 + 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 + indent +
dft.format(cloudlet.getActualCPUTime()) +
                                                indent + indent +
dft.format(cloudlet.getExecStartTime())+ indent + indent + indent +
dft.format(cloudlet.getFinishTime()));
                        }
                }

        }
}
```

Output:



Result:

Hence simulation entities dynamically has been paused and resumed successfully.

Ex No: 13 **Register No:**
Date : **Name :**

To Create a Warehouse Application in Salesforce.Com

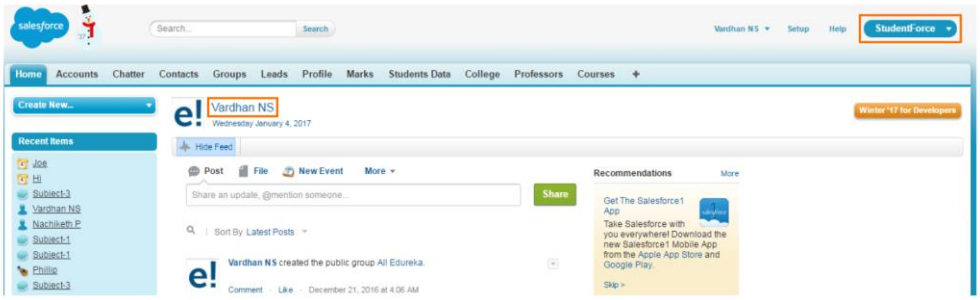
Aim:

To Create a Warehouse Application in Salesforce.Com

Procedure:

Salesforce Apps

1. The primary function of a Salesforce app is to manage customer data. Salesforce apps provide a simple UI to access customer records stored in objects (tables). Apps also help in establishing relationship between objects by linking fields.
2. Apps contain a set of related tabs and objects which are visible to the end user. The below screenshot shows, how the *StudentForce* app looks like.



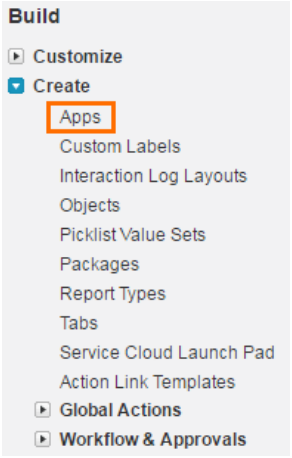
The highlighted portion in the top right corner of the screenshot displays the app name: *StudentForce*. The text highlighted next to the profile pic is my username: *Vardhan NS*.

Before you create an object and enter records, you need to set up the skeleton of the app. You can follow the below instructions to set up the app.

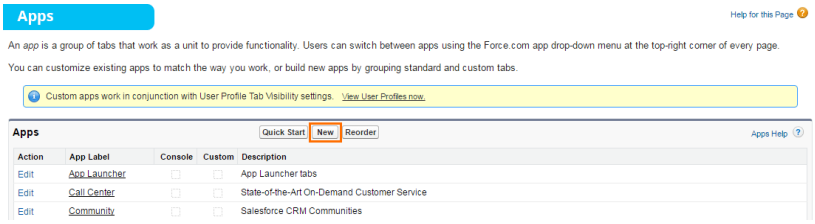
Steps To Setup The App

1. Click on *Setup* button next to app name in top right corner.

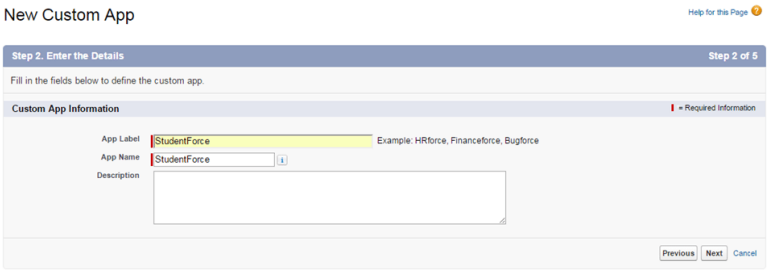
2. In the bar which is on the left side, go to *Build* → select *Create* → select *Apps* from the drop down menu.



3. Click on *New* as shown in the below screenshot.



4. Choose *Custom App*.
5. Enter the *App Label*. *StudentForce* is the label of my app. Click on *Next*.



6. Choose a profile picture for your app. Click *Next*.
7. Choose the tabs you deem necessary. Click *Next*.
8. Select the different profiles you want the *app* to be assigned to. Click *Save*.

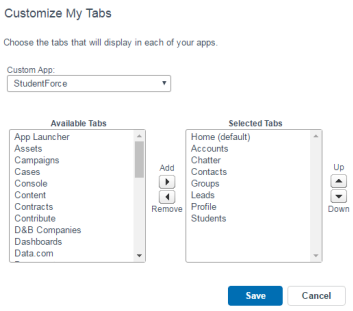
In steps 7 and 8, you were asked to choose the relevant tabs and profiles. Tabs and profiles are an integral part of Salesforce Apps because they help you to manage objects and records in Salesforce.

Salesforce Tabs

1. Tabs are used to access objects (tables) in the Salesforce App. They appear on top of the screen and are similar to a toolbar. It contains shortcut links to multiple objects.
2. On clicking the object name in a tab, records in that object will be displayed. Tabs also contain links to external web content, custom pages and other URLs.
3. All applications will have a *Home* tab by default. Standard tabs can be chosen by clicking on '+' in the Tab menu. Accounts, Contacts, Groups, Leads, Profile are the standard tabs offered by Salesforce. For example, *Accounts* tab will show you the list of accounts in the SFDC org and *Contacts* tab will show you the list of contacts in the SFDC org.

Steps To Add Tabs

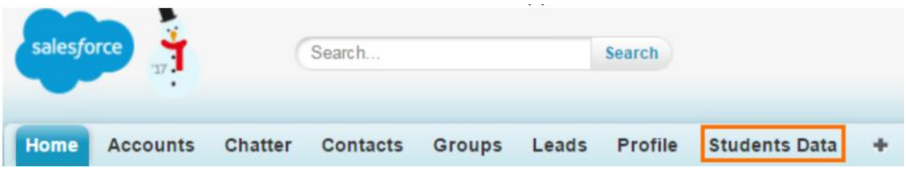
1. Click on '+' in the tab menu.
2. Click on *Customize tabs*, which is present on the right side.
3. Choose the tabs of your choice and click on
4. *Save*.



Besides standard tabs, you can also create custom tabs. *Students* tab that you see in the above screenshot is a custom tab that I have created. This is a shortcut to reach the custom object: *Students*.

Steps To Create Custom Tabs

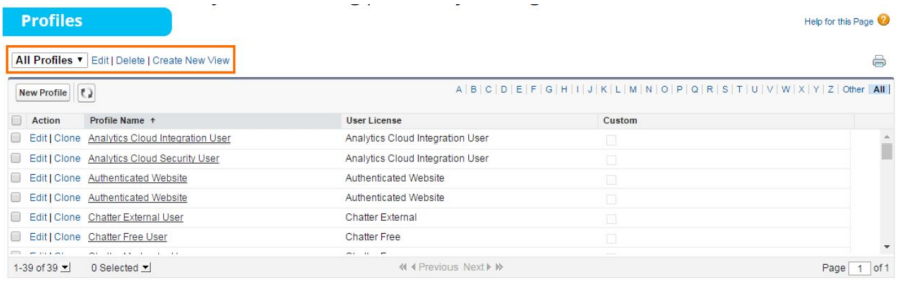
1. Navigate to Setup → Build → Create → Tabs.
2. Click on *New*.
3. Select the object name for which you are creating a tab. In my case, it is *Students Data*. This is a custom object which I have created (the instructions to create this object is covered later in this blog).
4. Choose a tab style of your preference and enter a description.
5. Click on Next → Save. The new *Students Data* tab will be created.



Salesforce Profiles

1. Every user who needs to access the data or SFDC org will be linked to a profile. A profile is a collection of settings and permissions which controls what a user can view, access and modify in Salesforce.
2. A profile controls user permissions, object permissions, field permissions, app settings, tab settings, apex class access, Visualforce page access, page layouts, record types, login hour and login IP addresses.
3. You can define profiles based on the background of the user. For example, different levels of access can be set for different users like system administrator, developer and sales representative.

Output



Result

Hence, a Warehouse Application in Salesforce.Com is created successfully.

Ex No: 14 **Register No:**
Date : **Name :**

Case Study - To Create a Warehouse Application in Sales force.Com using Apex prog Lang

Aim:

To Create a Warehouse Application in Sales force.Com using Apex prog Lang

Procedure:

Salesforce Application

1. A salesforce application is a logical container for all of the objects, tabs, process and services associated with a given business function
2. A salesforce application is a group of tabs that work as a unit to provide functionality
3. We can customize existing app to match the way to work or build new apps by grouping standard and custom tabs.
4. A force.com custom app consists of name, description, an ordered list of tabs and optionally a custom logo and a landing page.
5. Salesforce provides standard apps such as Sales, Call center, Marketing and Community etc….
6. Users can switch between apps using the force.com app drop-down menu at the top right corner of every page.
7. There are two types of salesforce application one is Custom App and other one is Service cloud console.

Apex:

1. Apex is an object-oriented and strongly typed programming language developed by Salesforce for building Software as a Service (SaaS) and Customer Relationship Management (CRM). Apex helps developers to create third-party SaaS applications and add business logic to system events by providing back-end database support and client-server interfaces.
2. Apex helps developers to add business logic to the system events like button clicks, related record updates, and Visualforce pages.
3. Apex executes in a multi-tenant environment, and Salesforce has defined some governor limits that prevent a user from controlling the shared resources. Any code that crosses the salesforce governor limit fails, an error shows up.
4. Salesforce object can be used as a datatype in apex.

Account acc = new Account();

Here Account is a standard salesforce object.

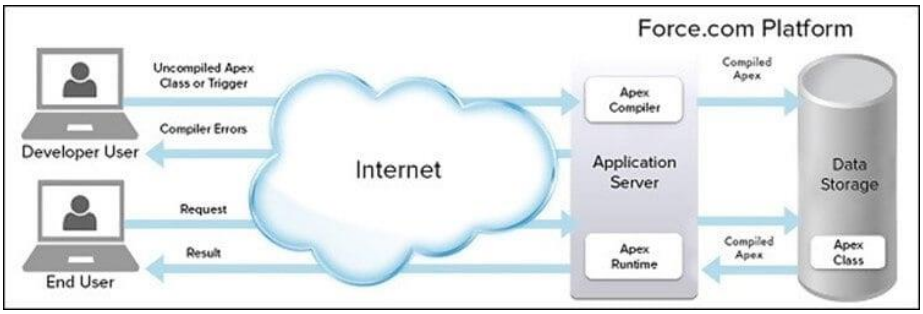
Apex automatically upgrades with every Salesforce release.

Working Structure Of Apex

Flow of actions for an apex code:

Developer Action: All the apex code written by a developer is compiled into a set of instructions that can be understood by apex runtime interpreter when the developer saves the code to the platform and these instructions then save as metadata to the platform.

End User Action: When the user event executes an apex code, the platform server gets the compiled instructions from metadata and runs them through the apex interpreter before returning the result.



Apex Syntax

Variable Declaration:

As apex is strongly typed language, it is mandatory to declare a variable with datatype in apex.

For example

contact con = new contact();
here the variable con is declared with contact as a datatype.

SOQL Query:

SOQL stands for salesforce object query language. SOQL is used to fetch sObject records from Salesforce database. For example-

Account acc = [select id, name from Account Limit 1];
The above query fetches account record from salesforce database.

Loop Statement:

Loop statement is used to iterate over the records in a list. The number of iteration is equal to the number of records in the list. For example:

```
list<Account>listOfAccounts = [select id, name from account limit 100];
// iteration over the list of accounts
for(Account acc : listOfAccounts){
        //your logic
}
```

In the above snippet of code, listOfAccounts is a variable of list datatype.

Flow Control Statement:

Flow control statement is beneficial when you want to execute some lines of the code based on some conditions.

For example:

```
list<Account>listOfAccounts = [select id, name from account limit 100];
// execute the logic if the size of the account list is greater than zero
if(listOfAccounts.size() >0){
        //your logic
}
```

The above snippet of code is querying account records from the database and checking the list size.

DML statement:

DML stands for data manipulation language. DML statements are used to manipulate data in the Salesforce database. For example –

```
Account acc = new Account(Name = ' Test Account');
Insert acc; //DML statement to create account record.
```

Apex Development Environment

Apex code can be developed either in sandbox and developer edition of Salesforce.

It is a best practice to develop the code in the sandbox environment and then deploys it to the production environment.

Keywords in Apex

If a class is defined with this keyword, then all the sharing rules apply to the current user is enforced and if this keyword is absent, then code executes under system context.

For Example:

```
public with sharing class MyApexClass{
// sharing rules enforced when code in this class execute
}
```

Without sharing:

If a class is defined with this keyword, then all the sharing rules apply to the current user is not enforced.

For Example:

```
public without sharing class MyApexClass{
// sharing rules is not enforced when code in this class execute
}
```

Static:

A variable, Method is defined with the static keyword is initialized once and associated with the class. Static variables, methods can be called by class name directly without creating the instance of a class.

Final:

A constant, Method is defined with the final keyword can't be overridden. For example:

```
public class myCls {
static final Integer INT_CONST = 10;
}
```

If you try to override the value for this INT_CONST variable, then you will get an exception – System.FinalException: Final variable has already been initialized.

Return:

This keyword returns a value from a method. For example:

```
public String getName() {
return  'Test' ;
}
```

Null:

It defines a null constant and can be assigned to a variable. For example

```
Boolean b = null;
```

Result:

Hence a case study for creating a Warehouse Application in Sales force.Com using Apex prog Lang is explained successfully.

Ex No: 15 **Register No:**
Date : **Name :**

Implementation of SOAP Web Services

Aim:

To study the implementation of SOAP web services.

Procedure:

SOAP

1. SOAP is an XML-based protocol for accessing web services over HTTP. It has some specification which could be used across all applications.
2. SOAP is known as the Simple Object Access Protocol, but in later times was just shortened to SOAP v1.2. SOAP is a protocol or in other words is a definition of how web services talk to each other or talk to client applications that invoke them.

A simple SOAP service example of a complex type is shown below.

- Suppose we wanted to send a structured data type which had a combination of a "Tutorial Name" and a "Tutorial Description," then we would define the complex type as shown below.
- The complex type is defined by the element tag <xsd:complexType>.
- All of the required elements of the structure along with their respective data types are then defined in the complex type collection.

```
<xsd:complexType>
 <xsd:sequence>
        <xsd:element name="Tutorial Name" type="string"/>
        <xsd:element name="Tutorial Description"  type="string"/>
  </xsd:sequence>
</xsd:complexType>
```

SOAP Message Structure

1. One thing to note is that SOAP messages are normally auto-generated by the web service when it is called.
2. Whenever a client application calls a method in the web service, the web service will automatically generate a SOAP message which will have the necessary details of the data which will be sent from the web service to the client application.

SOAP Message has the following elements –

1. The Envelope element
2. The header element and
3. The body element

4. The Fault element (Optional)

Below is an SOAP API example of version 1.2 of the SOAP envelope element.

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
SOAP-ENV:encodingStyle=" http://www.w3.org/2001/12/soap-encoding">
        <soap:Body>
       <Guru99WebService xmlns="http://tempuri.org/">
              <TutorialID>int</TutorialID>
            </Guru99WebService>
        </soap:Body>
</SOAP-ENV:Envelope>
```

Example for Fault Message

An example of a fault message is given below. The error is generated if the scenario wherein the client tries to use a method called TutorialID in the class GetTutorial.

The below fault message gets generated in the event that the method does not exist in the defined class.

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelopexmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
     <SOAP-ENV:Body>
       <SOAP-ENV:Fault>
       <faultcode xsi:type="xsd:string">SOAP-ENV:Client</faultcode>
       <faultstring xsi:type="xsd:string">
          Failed to locate method (GetTutorialID) in class (GetTutorial)
       </faultstring>
   </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Output:

When you execute the above code, it will show the error like “Failed to locate method (GetTutorialID) in class (GetTutorial)”

Result:
Hence, the implementation of SOAP web services is studied successfully.