



SRM

INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University u/s 3 of UGC Act, 1956)

DELHI NCR CAMPUS, GHAZIABAD, (U.P.)

S R M INSTITUTE OF SCIENCE & TECHNOLOGY

(FACULTY OF SCIENCE & HUMANITIES)

DEPARTMENT OF COMPUTER APPLICATIONS

PRACTICAL FILE

INTRODUCTION TO COMPUTING WITH

DISTRIBUTED DATA PROCESSING

[UDS23201J]

BCA 1ST YEAR, 2nd SEMESTER

Session: 2023-24

SUBMITTED TO :

Aalekh Choudhary

Assistant Professor

Department of Computer Applications

SUBMITTED BY :

Student Reg No

Student Name

Department of Computer Applications

S R M INSTITUTE OF SCIENCE & TECHNOLOGY

DELHI NCR CAMPUS, MODINAGAR

(FACULTY OF SCIENCE & HUMANITIES)

DEPARTMENT OF COMPUTER APPLICATIONS

Register No. :

BONAFIDE CERTIFICATE

Certified to be the Bonafide record of the work done by **XYZ Kumar** of BCA **1st Year/ 2nd Semester** for the Award of **Bachelor's Degree** in the **DEPARTMENT OF COMPUTER APPLICATIONS** in **Introduction to Computing with Distributed Data Processing [UDS23201J]** laboratory during the Academic Year **2023-24**.

SUBJECT IN – CHARGE

HEAD – COMPUTER APPLICATIONS

Submitted for the University Examination held on _____.

INTERNAL EXAMINER - 1

INTERNAL EXAMINER - 2

TABLE OF CONTENTS

S. No	Date	Experiment Title	Page No	Signature
1		Write a program in JAVA to implement Remote Method Invocation (RPI)	5-7	
2		Write a Program in JAVA to implement Remote Procedure Call(RPC).	8-11	
3		Write a query in SQL to create a table of name Employee with following structure.	12-13	
4		Write a query in SQL , Add a column commission with domain to the Employees table.	14-15	
5		Write a query in SQL , to perform alter (Drop Operation) command on table.	16-17	
6		Write a query in SQL , to perform alter(Modify Operation) command on table.	18	
7		Write a query in SQL , to insert the five records into the table employee.	19	
8		Write a query in SQL , to perform operation using clauses find the those employees name whose salary is greater than 7000	20	
9		Write Case study on PaaS (FaceBook, Google AppEngine)	21-23	
10		Write the steps , How to create Virtualization in Cloud by using KVM and VMware	24-32	
11		Write steps for installation of MongoDB Atlas	33-39	
12		How MongoDB CRUD operations works ? What are the syntax for CRUD operation using MongoDB database.	40-46	
13		Write a simple code for Data modeling in MongoDB	47-48	
14		Write steps for Creation of Queries in MongoDB	49	
15		Using Creation of Queries in MongoDB , write a program in MongoDB to create a Collection:	50	
16		Using Creation of Queries in MongoDB , write a program in MongoDB to insert a document based data.	51-52	
17		Using Creation of Queries in MongoDB , write a program in MongoDB to update a document .	53	

18		Write a program to sort a single field in MongoDB	54-55	
19				
20				
21				
22				
23				
24				

Experiment 1: Write a program to implement Remote Method Invocation

Aim:

Using Java Programming write a program to implement Remote Method Invocation.

Algorithm/ Procedure:

Here is the the steps to be followed sequentially to implement Interface as defined:

1. Defining a remote interface
2. Implementing the remote interface
3. Creating Stub and Skeleton objects from the implementation class using rmic (RMI compiler)
4. Start the rmiregistry
5. Create and execute the server application program
6. Create and execute the client application program.

Program:

Remote Method Invocation(RMI) is a Java API that enables communication between different Java Virtual Machine(JVMs) to invoke methods on remote objects.

We must create three Java files to execute this concept.

1. Calculator.java (interface for remote method);

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Calculator extends Remote {
    int add(int a, int b) throws RemoteException;
}
```

2. CalculatorImpl.java (implementation of the remote interface):

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class CalculatorImpl extends UnicastRemoteObject implements Calculator {

    protected CalculatorImpl() throws RemoteException {
        super();
    }

    @Override

    // function definition
```

```

public int add(int a, int b) throws RemoteException {
    return a + b;
}

public static void main(String[] args) {
    try {
        // Create an instance of the remote object
        Calculator calculator = new CalculatorImpl();

        // Bind the remote object to the RMI registry
        java.rmi.registry.LocateRegistry.createRegistry(1099);
        java.rmi.Naming.rebind("CalculatorService", calculator);

        System.out.println("CalculatorService bound in registry");
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

3. **CalculatorClient.java** (client program):

```

import java.rmi.Naming;

public class CalculatorClient {

    public static void main(String[] args) {
        try {
            // Look up the remote object in the RMI registry
            Calculator calculator = (Calculator) Naming.lookup("rmi://localhost/CalculatorService");

            // Invoke remote methods
            int result = calculator.add(5, 10);
            System.out.println("Result of remote method invocation: " + result);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```
}  
}
```

Input/ Output

1. Compile the source files:

```
javac Calculator.java CalculatorImpl.java CalculatorClient.java
```

1. Start the RMI registry (you can run it in a separate terminal):

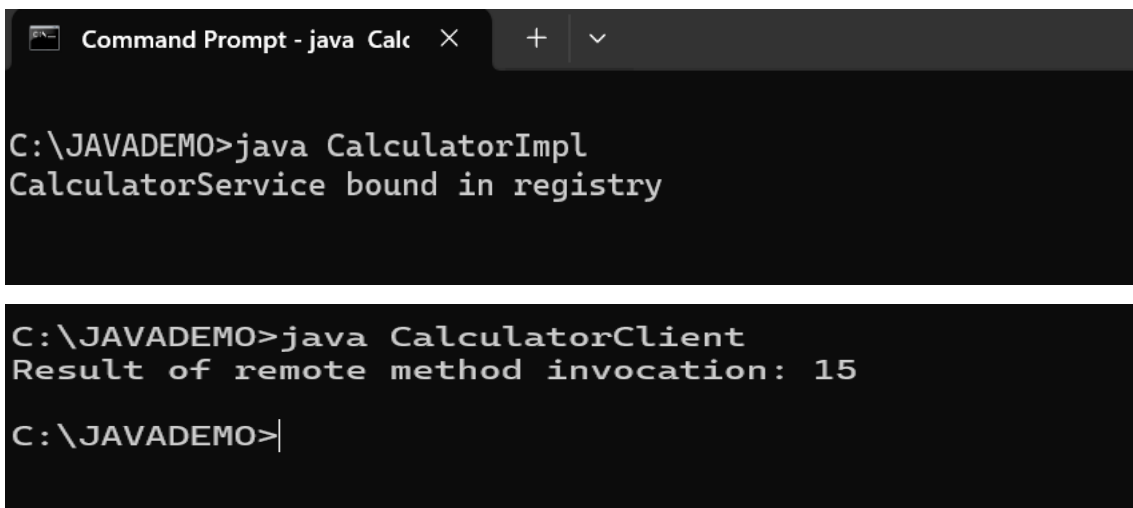
```
rmiregistry
```

2. Run the server (in a separate terminal):

```
java CalculatorImpl
```

3. Run the client (in a separate terminal):

```
java CalculatorClient
```



```
Command Prompt - java Calc  X  +  v  
  
C:\JAVADEMO>java CalculatorImpl  
CalculatorService bound in registry  
  
C:\JAVADEMO>java CalculatorClient  
Result of remote method invocation: 15  
  
C:\JAVADEMO>|
```

Result:

Thus the JAVA program to implement Remote Method Invocation was executed and verified successfully.

Experiment 2: Write a Program in JAVA to implement Remote Procedure Call

Aim:

a Program in Java to implement Remote Procedure Call by using Client Server Method.

Procedure:

- The client stub is called by the client.
- The client stub makes a system call to send the message to the server and puts the parameters in the message.
- The message is sent from the client to the server by the client's operating system.
- The message is passed to the server stub by the server operating system.
- The parameters are removed from the message by the server stub.
- Then, the server procedure is called by the server stub.

Program:

Program written at server machine: RPCServer.java

```
import java.io.*;
import java.net.*;

public class RPCServer {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(1234);
            System.out.println("Server waiting for client on port 1234...");

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Connected to client");

                // Create a new thread to handle the client
                new ClientHandler(clientSocket).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



```

class ClientHandler extends Thread {
    private Socket clientSocket;

    public ClientHandler(Socket clientSocket) {
        this.clientSocket = clientSocket;
    }

    @Override
    public void run() {
        try {
            // Setup input and output streams
            BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

            // Read the method name from the client
            String methodName = in.readLine();

            // Process the remote method
            int result = 0;
            if (methodName.equals("add")) {
                int a = Integer.parseInt(in.readLine());
                int b = Integer.parseInt(in.readLine());
                result = add(a, b);
            }

            // Send the result back to the client
            out.println(result);

            // Close the connection
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```
private int add(int a, int b) {  
    return a + b;  
}  
}
```

Program written at client machine RPCClient.java

```
import java.io.*;  
import java.net.*;  
  
public class RPCClient {  
    public static void main(String[] args) {  
        try {  
            Socket clientSocket = new Socket("localhost", 1234);  
  
            // Setup input and output streams  
            BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));  
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);  
  
            // Call the remote method 'add'  
            out.println("add");  
            out.println(5);  
            out.println(10);  
  
            // Receive the result from the server  
            int result = Integer.parseInt(in.readLine());  
            System.out.println("Result of remote method invocation: " + result);  
  
            // Close the connection  
            clientSocket.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Run Separately on client and server machine or different terminals

First Compile then run

```
javac RPCServer.java
```

```
javac RPCClient.java
```

>>Run on the server in one terminal

```
java RPCServer
```

```
C:\JAVADEMO>javac RPCClient.java

C:\JAVADEMO>java RPCServer
Server waiting for client on port 1234...
Connected to client
|
```

>> Run the client in another terminal

```
java RPCClient
```

```
C:\JAVADEMO>javac RPCClient.java

C:\JAVADEMO>java  RPCClient.java
Result of remote method invocation: 15
```

Result: Thus the JAVA program to implement Remote Procedure Call was executed and verified successfully.

Experiment 3: Write a query in SQL to create a table of name Employee with following structure.

Name	Type
Empno	Number
Ename	Varchar2(10)
Job	Varchar2(10)
Mgr	Number
Sal	Number

Aim: Using Oracle SQL, Create a table of name Employee with the column name Empno, Empname, Job, Mgr, Sal.

Procedure:

1. Open Oracle SQL command line window.
2. Insert command for create table.
3. Press button run.
4. Result: table created
5. Show with using select command.

Program:

SQL> create table Employees(empno number(5) ,ename varchar2(10), job varchar2(10), mgr number ,sal number);

Output: **Table created.**

SQL> desc employee;

```
create table Employees(empno number(5) ,ename varchar2(10), job varchar2(10 ), mgr number ,sal number);
desc employees
```

Results Explain Describe Saved SQL History

Object Type **TABLE** Object **EMPLOYEES**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPLOYEES	EMPNO	Number	-	5	0	-	✓	-	-
	ENAME	Varchar2	10	-	-	-	✓	-	-
	JOB	Varchar2	10	-	-	-	✓	-	-
	MGR	Number	-	-	-	-	✓	-	-
	SAL	Number	-	-	-	-	✓	-	-
1 - 5									

Output:

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
<u>EMPLOYEE</u> <u>S</u>	<u>EMPNO</u>	Number	-	5	0	-	✓	-	-
	<u>ENAME</u>	Varchar2	10	-	-	-	✓	-	-
	<u>JOB</u>	Varchar2	10	-	-	-	✓	-	-
	<u>MGR</u>	Number	-	-	-	-	✓	-	-
	<u>SAL</u>	Number	-	-	-	-	✓	-	-

Result: Table Created Successfully

Experiment 4: Write a query in SQL , Add a column commission with domain to the Employees table.

Aim: Using Oracle SQL , Update the table Employees , Add column commission(Alter Command).

Algorithm:

- Open the SQL command line or a development tool
- Enter the ALTER TABLE statement and specify the table
- Use the DROP COLUMN keyword and name the column
- Execute and wait until it's successful
- Verify the column is gone by checking the table's structure

COMMAND:

ALTER TABLE EMPLOYEES ADD COMMISSION NUMBER(10)

Output:

```
ALTER TABLE EMPLOYEES ADD COMMISSION NUMBER(10)
```

Results	Explain	Describe	Saved SQL	History
---------	---------	----------	-----------	---------

Table altered.

0.03 seconds

To check the updates run

desc employees

Output:

```
ALTER TABLE EMPLOYEES ADD COMMISSION NUMBER(10)
```

```
desc employees
```

Results Explain Describe Saved SQL History

Object Type TABLEObject EMPLOYEES

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPLOYEES	EMPNO	Number	-	5	0	-	✓	-	-
	ENAME	Varchar2	10	-	-	-	✓	-	-
	JOB	Varchar2	10	-	-	-	✓	-	-
	MGR	Number	-	-	-	-	✓	-	-
	SAL	Number	-	-	-	-	✓	-	-
	COMMISSION	Number	-	10	0	-	✓	-	-
									1 - 6

Result:

Column Added Successfully.

Experiment 5: Write a query in SQL , to perform alter (Drop Operation) command on table.

Aim: Using Oracle SQL , Alter table drop column Commission.

Algorithm:

- Open the SQL command line or a development tool
- Enter the ALTER TABLE statement and specify the table
- Use the DROP COLUMN keyword and name the column
- Execute and wait until it's successful
- Verify the column is gone by checking the table's structure

Command:

Before editing table the structure of table : Employees is

>> Select * From Employees

SELECT * FROM EMPLOYEES					
Results Explain Describe Saved SQL History					
EMPNO	ENAME	JOB	MGR	SAL	COMMISSION
101	abhi	manager	1234	10000	70
102	rohith	analyst	2345	9000	65
103	david	analyst	3456	9000	65
104	rahul	clerk	4567	7000	55
105	pramod	salesman	5678	5000	50
5 rows returned in 0.01 seconds			CSV Export		

Now perform Alter Command to drop column

>> ALTER TABLE EMPLOYEES DROP COLUMN COMMISSION


```
ALTER TABLE EMPLOYEES DROP COLUMN COMMISSION
```

Results Explain Describe Saved SQL History

Table dropped.

0.11 seconds

Output:

```
ALTER TABLE EMPLOYEES DROP COLUMN COMMISSION
```

```
-- CHECK DROP COLUMN(EXIST OR NOT IN TABLE)
```

```
SELECT * FROM EMPLOYEES
```

Results Explain Describe Saved SQL History

EMPNO	ENAME	JOB	MGR	SAL
101	abhi	manager	1234	10000
102	rohith	analyst	2345	9000
103	david	analyst	3456	9000
104	rahul	clerk	4567	7000
105	pramod	salesman	5678	5000

5 rows returned in 0.00 seconds

[CSV Export](#)

Result:

Table Altered(Column Dropped) Successfully

Experiment 6: Write a query in SQL , to perform alter(Modify Operation) command on table.

Aim: Using Oracle SQL , Alter table Modify column Commission.

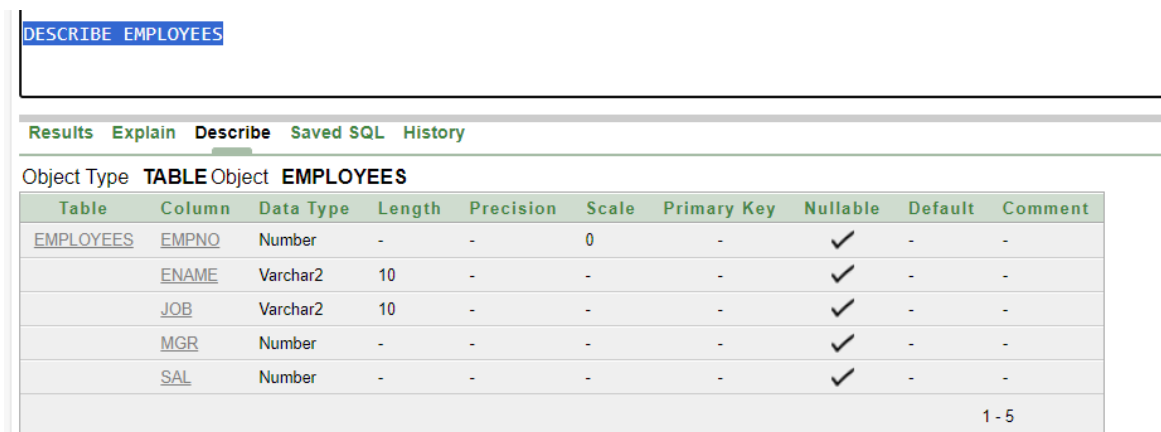
Algorithm:

- Open the SQL command line or a development tool
- Enter the ALTER TABLE statement and specify the table
- Use the MODIFY COLUMN keyword and name the column
- Execute and wait until it's successful
- Verify the column is gone by checking the table's structure

Command:

Before editing table the structure of table : Employees is

>> Describe Employees



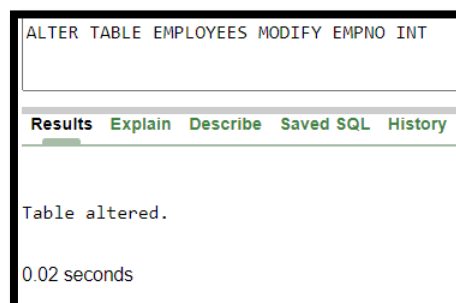
The screenshot shows the SQL Developer interface. At the top, the command 'DESCRIBE EMPLOYEES' is entered in the SQL command window. Below it, the 'Results' tab is selected, displaying the table structure for 'EMPLOYEES'. The table has five columns: EMPNO (Number), ENAME (Varchar2), JOB (Varchar2), MGR (Number), and SAL (Number). Each column is marked as nullable with a checkmark in the 'Nullable' column. The 'Primary Key' column is empty for all columns. The 'Default' and 'Comment' columns are also empty. The table is displayed in a grid format with a header row and five data rows. The page number '1 - 5' is visible at the bottom right of the results area.

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPLOYEES	EMPNO	Number	-	-	0	-	✓	-	-
	ENAME	Varchar2	10	-	-	-	✓	-	-
	JOB	Varchar2	10	-	-	-	✓	-	-
	MGR	Number	-	-	-	-	✓	-	-
	SAL	Number	-	-	-	-	✓	-	-

Now perform Alter Command to drop column

>>ALTER TABLE EMPLOYEES MODIFY EMPNO INT

Output:



The screenshot shows the SQL Developer interface. At the top, the command 'ALTER TABLE EMPLOYEES MODIFY EMPNO INT' is entered in the SQL command window. Below it, the 'Results' tab is selected, displaying the output 'Table altered.' and the execution time '0.02 seconds'.

Results	Explain	Describe	Saved SQL	History
Table altered.				
0.02 seconds				

Result:

Table Altered(Column Modified) Successfully

Experiment 7: Write a query in SQL , to insert the five records into the table employee.

Aim: Using Oracle SQL , insert five records into the table - Employee.

COMMAND:

```
insert into employees values(101,'abhi','manager',1234,10000,'70');
insert into employees values(102,'rohith','analyst',2345,9000,'65');
insert into employees values(103,'david','analyst',3456,9000,'65');
insert into employees values(104,'rahul','clerk',4567,7000,'55');
insert into employees values(105,'pramod','salesman',5678,5000,'50');
```

Output:

```
select * from employees

insert into employees values(101,'abhi','manager',1234,10000,'70')
insert into employees values(102,'rohith','analyst',2345,9000,'65');
insert into employees values(103,'david','analyst',3456,9000,'65');
insert into employees values(104,'rahul','clerk',4567,7000,'55');
insert into employees values(105,'pramod','salesman',5678,5000,'50');
```

Results Explain Describe Saved SQL History

EMPNO	ENAME	JOB	MGR	SAL	COMMISSION
101	abhi	manager	1234	10000	70
102	rohith	analyst	2345	9000	65
103	david	analyst	3456	9000	65
104	rahul	clerk	4567	7000	55
105	pramod	salesman	5678	5000	50

5 rows returned in 0.00 seconds [CSV Export](#)

Result:

Data inserted into table Employee successfully

Clauses in SQL Syntax:

SELECT column FROM table_name WHERE conditions GROUP BY column ORDER BY column

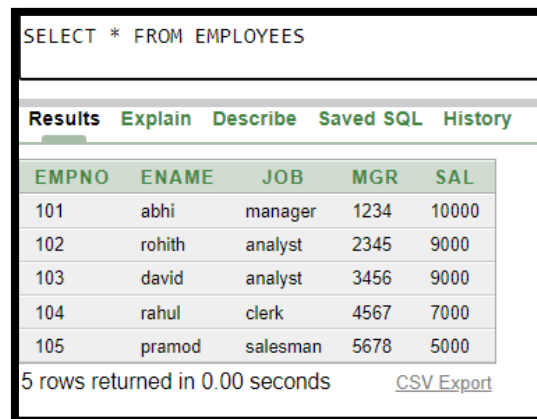
Experiment 8: Write a query in SQL , to perform operation using clauses find the those employees name whose salary is greater than 7000.

Aim: Using Oracle SQL , find record of those employees whose salary is greater than 7000.

COMMAND:

Check all record on table Employees

>> ***SELECT * FROM EMPLOYEES***



SELECT * FROM EMPLOYEES

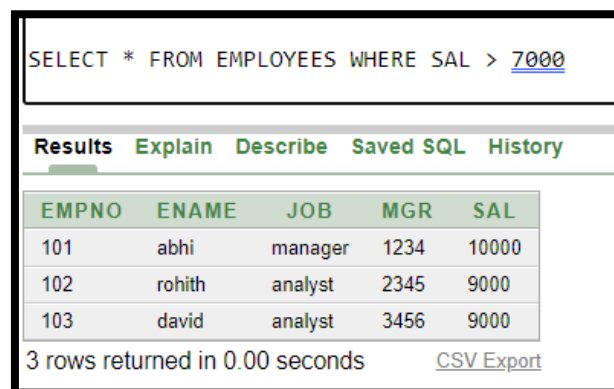
Results	Explain	Describe	Saved SQL	History
EMPNO	ENAME	JOB	MGR	SAL
101	abhi	manager	1234	10000
102	rohith	analyst	2345	9000
103	david	analyst	3456	9000
104	rahul	clerk	4567	7000
105	pramod	salesman	5678	5000

5 rows returned in 0.00 seconds [CSV Export](#)

Now find those record whose salary is greater than 7000

>> ***SELECT * FROM EMPLOYEES WHERE SAL > 7000***

Output:



SELECT * FROM EMPLOYEES WHERE SAL > 7000

Results	Explain	Describe	Saved SQL	History
EMPNO	ENAME	JOB	MGR	SAL
101	abhi	manager	1234	10000
102	rohith	analyst	2345	9000
103	david	analyst	3456	9000

3 rows returned in 0.00 seconds [CSV Export](#)

Result:

Record found successfully

Experiment 9: Write Case study on PaaS (FaceBook, Google AppEngine)

Aim:

To study about Facebook and Google AppEngine, find out how both platform work As Platform-as-a-Service).



Algorithm/ Procedure:

- Step 1: definition about topics (Facebook and Google AppEngine).
- Step 2: conclude research about both topics how it work platform as a service.
- Step 3: Make the case study generalizable
- Step 4: conclusion of study.

Case Study:

1. Facebook

Facebook is not considered a SaaS company; rather, they're identified as a PaaS (Platform as a Service) company. Below is the distinction of why Facebook is considered a PaaS vs SaaS company.

Software as a Service is really the most basic form of cloud computing and doesn't require third party developments or installations for the user. SaaS products provide the ability to collaborate with other users from any location.

Platform as a Service is a step up from SaaS as developers have the APIs to create and develop an app that will operate in a specific environment. Developers can create apps designed specifically to be used for the Facebook platform using open APIs. That app is then available to any Facebook user. This is what makes Facebook a PaaS company compared to a SaaS company.

Facebook is an example of a platform as a service (PaaS). PaaS is a cloud computing model that allows users to rent hardware and software tools to quickly develop software and applications.

Facebook's platform offers tools and programming interfaces that allow developers to integrate with the "social graph" of personal relationships and other things like songs, places, and Facebook pages. Applications on facebook.com, external websites, and devices can all access the graph.

Developers can use proprietary APIs to create specific applications for the Facebook platform and make them available to any Facebook user. For example, some applications integrate a user's Twitter and Facebook account, while others integrate a database with a Facebook profile.

Third-party app developers can use the Messenger Platform to reach Facebook's millions of users on any device they use. Apps built on the platform install the Messenger service directly into the existing ecosystem of iOS and Android apps.

There is no revenue-sharing mandate among platform partners. The developers of an application can keep whatever it makes. However, Facebook does provide some services via revenue sharing, such as Facebook Credits.



2. Google App Engine

Google App Engine is an industry-leading Platform as a Service (PaaS) from the company that pioneered much of the microservices technology we rely on today.

In this blog, we are going to cover Google App Engine, its features, advantages, and use-cases.

Google App Engine (GAE) is a cloud computing platform that allows developers to build and host web applications in Google-managed data centers. GAE is a Platform as a Service (PaaS) solution within Google Cloud Platform (GCP).

GAE allows developers to build, deploy, and scale applications without managing the underlying infrastructure. It supports various programming languages, including Python, Java, Node.js, Go, and others.

GAE requires that applications be written in Java or Python, store data in Google Bigtable, and use the Google query language.

GAE automatically scales applications in response to the amount of traffic they receive. This means that users only pay for the resources they use, from zero to millions of users.

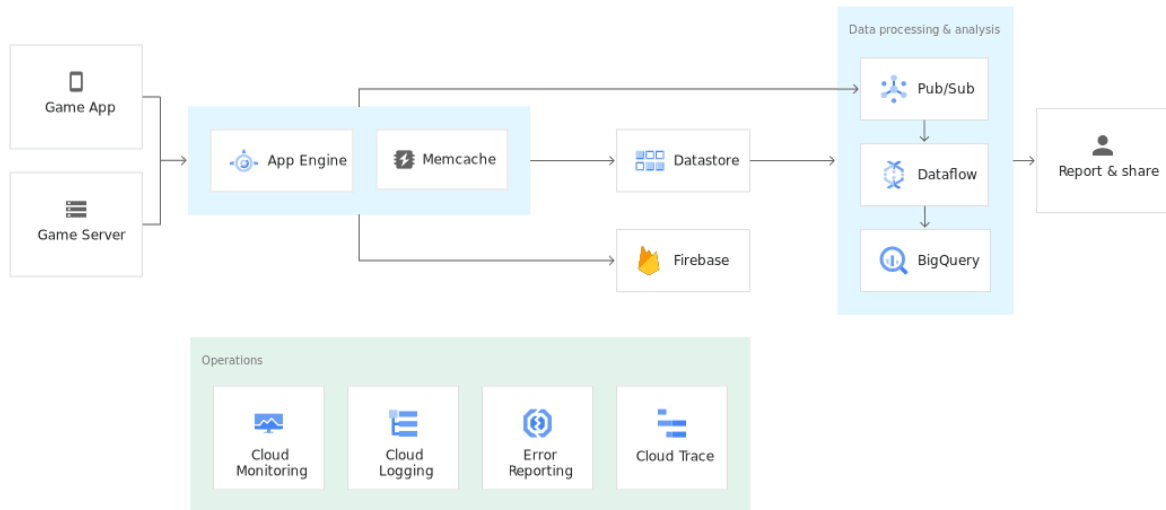
Google App Engine is a fully managed serverless platform for developing and hosting web applications at a scale. Users can choose from several popular languages, libraries, and frameworks to develop their applications and then App Engine takes care of provisioning servers and scaling app instances based on demand. It is a PaaS for building scalable applications.

Cloud Function Execution Environment

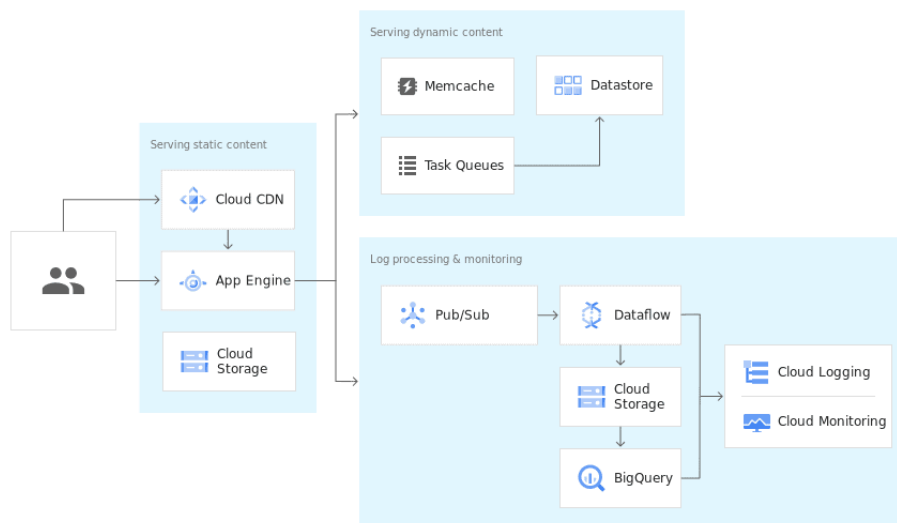
Cloud Functions run in a fully-managed, serverless environment where Google handles infrastructure, operating systems, and runtime environments completely on behalf of the users. Each Cloud Function runs in its own isolated secure execution context, scales automatically, and has a lifecycle independent from other functions. It supports multiple language runtimes like Python 3.7, Java 11, Ruby 2.7, etc.

Google App Engine Use-Cases

1.) Scalable Mobile Backends: App Engine automatically scales the hosting environment for users who are building their first mobile application or looking to reach out to existing users via a mobile experience. It offers seamless integration with Firebase which provides an easy-to-use frontend mobile platform along with a scalable and reliable back end.



2.) Modern Web-Applications: Quickly reach customers and end-users by deploying web apps on App Engine. With zero-config deployments and zero server management, It allows users to focus on just writing code. In addition to this, it automatically scales to support sudden traffic spikes without provisioning, patching, or monitoring.



Experiment 10: Write the steps , How to create Virtualization in Cloud by using KVM and VMware

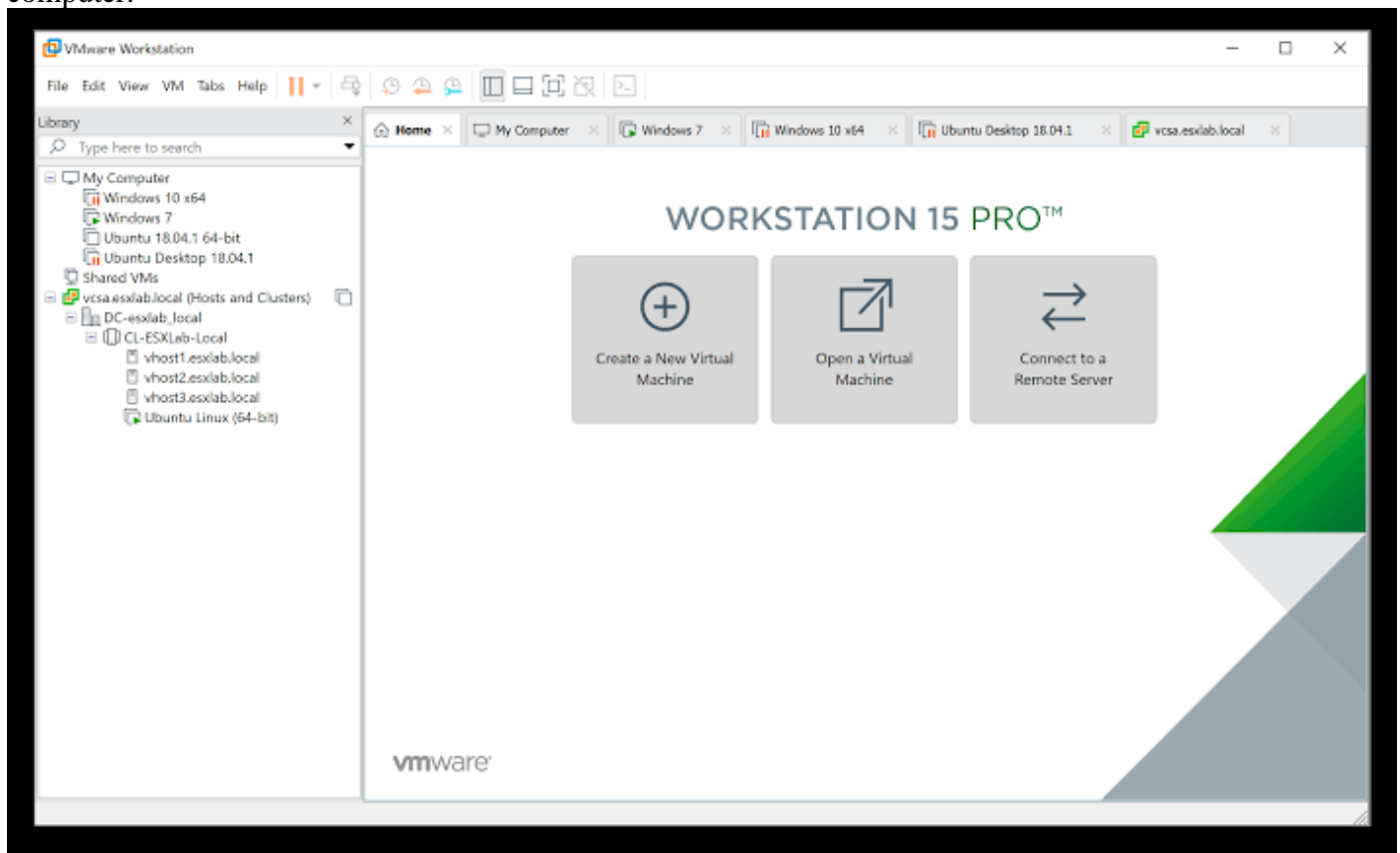
Aim: How to Create Virtual Machines in Linux Using KVM (Kernel-based Virtual Machine) and VMWare.

Procedure:

1. Log in to the VMware Cloud Console at <https://vmc.vmware.com>.
2. Click **Workloads > Create VM**.
3. Enter the VM configuration details.
4. Click **Review and Create**.

VMWare Machine

VMware Workstation is a virtualization tool that allows users to create and destroy virtual machines (VMs) on a single server. It also enables users to run multiple operating systems on a single physical host computer.



VMware Workstation is a type 2 hypervisor, which is a virtualization technology. It allows users to create multiple VMs with different operating systems across any platform. Each VM can run a single instance of any operating system, including Microsoft, Linux, BSD, and MS-DOS.

VMware Workstation helps network and system administrators check, test, and verify the client/server environment. It also offers a safe working environment and creates safe areas for new and testing projects.

The VMware Workstation product line consists of two products: Workstation Pro and Workstation Player

Create Virtual Machines

You can create a virtual machine (VM) in the VMware Cloud Console either by using an existing template or by specifying all the required configuration for your VM. By default, the virtual disks on the VM are configured with thick provisioning. If you want to use thin provisioning, use the vSphere Client to create VMs.

Prerequisites

- Set up the necessary authentication mechanism between the vCenter and VMware Cloud. See [Set Up vSphere+ Authentication](#).
- You must have either the **Cloud Administrator** or the **DevOps User** role in vSphere+.
- Ensure that you have the necessary permissions on the vCenter to view, create, and manage VMs. See [vCenter System Roles](#) and [Required Privileges for Common Tasks](#).

Procedure

5. Log in to the VMware Cloud Console at <https://vmc.vmware.com>.
6. Click **Workloads > Create VM**.
7. Enter the VM configuration details.

Option	Action
VM Location	Enter all the required information such as the name, vCenter, data center, and cluster where you want to create the VM.
Operating System and Hardware	For the Template Source you select, enter the required information for the VM. <ul style="list-style-type: none">▪ No Template. Enter the required operating system and the necessary hardware configuration for the VM.▪ Local. Select an existing template. The hardware, operating system, and other configurations on the new VM are taken from the template you select.
Storage	Select the datastore or cluster on which you want the VM to be created. Note: The vSAN default policy is applied when you select a vSAN datastore. To create a vSAN datastore with a custom policy, use the vSphere Client.
Networking	You can either select a network or proceed with the default selection.

8. Click **Review and Create**.

Results

- Creating a VM takes a while. Refresh the **Virtual Machines** page to see the VM you created.

- After you create a VM, the VM creation is registered as an event on the vCenter with the user name that you used to log in to vSphere+.
-

Kernel-Based Virtual Machine (KVM)

The purpose of Kernel-Based Virtual Machine (KVM) is to provide the ability of virtualization to Linux. It is an open-source technology which provides which allows Linux machines (host) to run different environments called virtual machines. Each virtual machine represents a different Linux process. Each Virtual Machine has its own copy of hardware such as memory, processor, and also software this allows to use of resources to a greater extent causing more reliability.

Steps for creating Virtual Machine in Linux using KVM

Note: We are going to use ubuntu os throughout this tutorial,

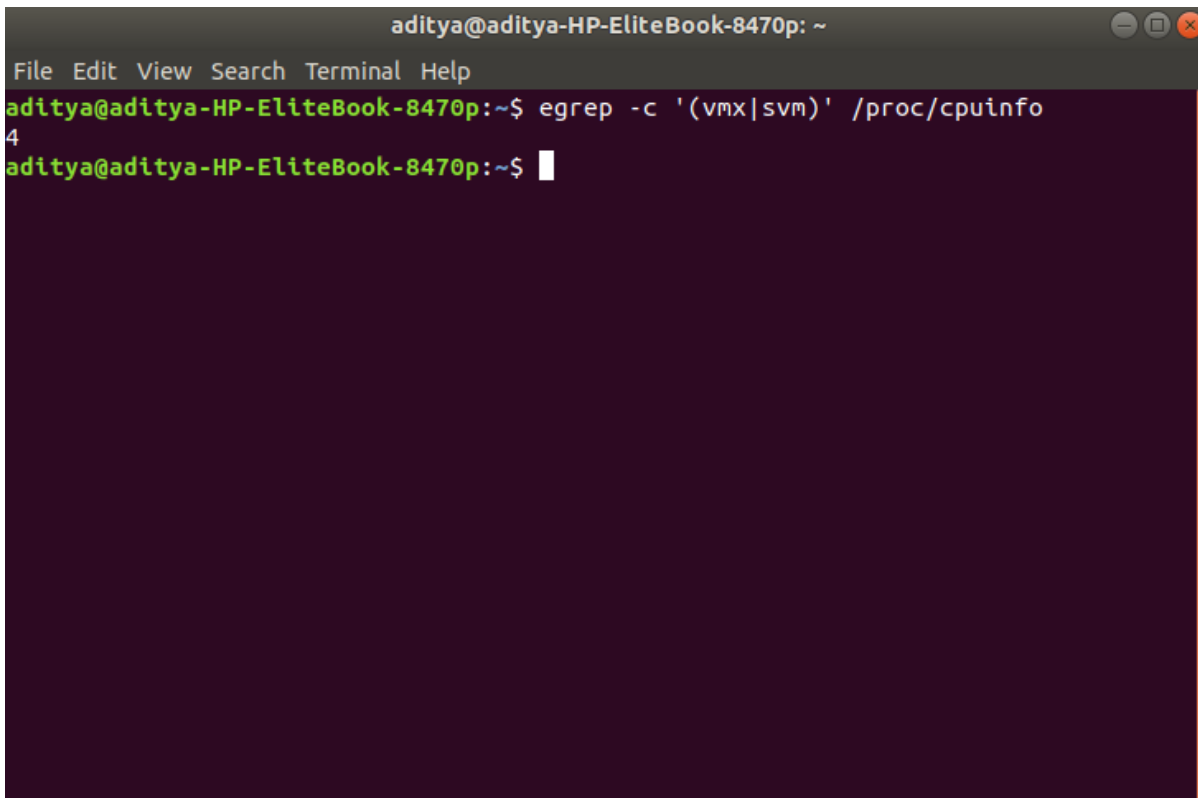
- Checking if virtualization is enabled or not
 - Installing KVM on Ubuntu
 - Adding a user to KVM
 - Creating Virtual Machine
1. Using command line
 2. Using interface

Virtualization Inability

To create a virtual machine first we need to ensure that virtualization is enabled on our system. It is mandatory to create virtual machines. There are multiple ways to check if virtualization is enabled,

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

The above command /proc/cpuinfo gives information about the processor. The output of the command will be a number. The output number 1 or more than that represents that virtualization is enabled, output 0 says you need to enable virtualization on your system.

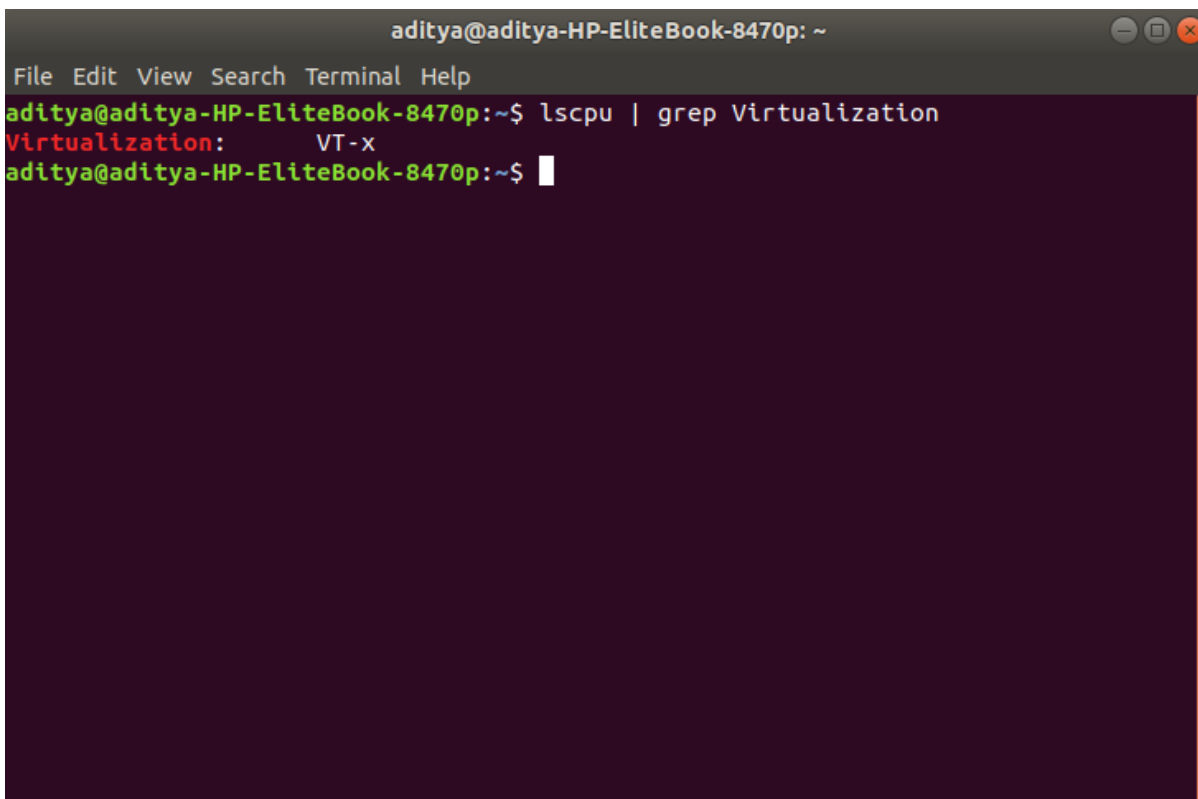


```
aditya@aditya-HP-EliteBook-8470p: ~  
File Edit View Search Terminal Help  
aditya@aditya-HP-EliteBook-8470p:~$ egrep -c '(vmx|svm)' /proc/cpuinfo  
4  
aditya@aditya-HP-EliteBook-8470p:~$
```

Virtualization Inability

\$ lscpu | grep Virtualization

This command is used to check which type of virtualization your processor supports. If the system contains a CPU with Intel VT support, the above command will provide the following output



```
aditya@aditya-HP-EliteBook-8470p: ~  
File Edit View Search Terminal Help  
aditya@aditya-HP-EliteBook-8470p:~$ lscpu | grep Virtualization  
Virtualization:      VT-x  
aditya@aditya-HP-EliteBook-8470p:~$
```

Virtualization Type

Installing KVM on Ubuntu

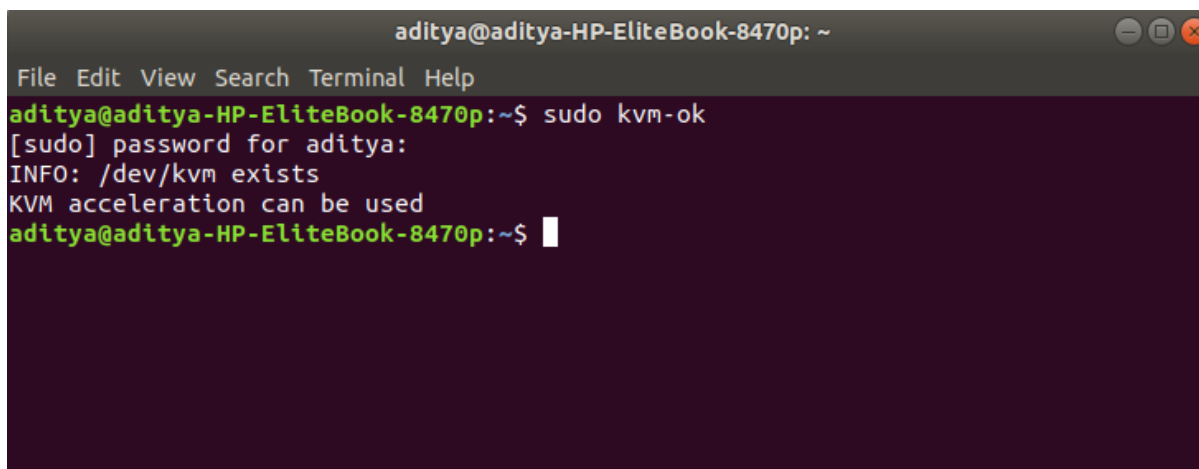
Now we know our system is capable of creating and running virtual machines, it's time to install tools which will create our virtual machines. To install KVM and other KVM dependencies such as virt-manager, bridge-utils enter command:

```
$ sudo apt -y install bridge-utils cpu-checker libvirt-clients libvirt-daemon qemu qemu-kvm
```

1. bridge-utils: The bridge-utils package contains a utility needed to create and manage bridge devices. This is useful in setting up networks for a hosted virtual machine (VM).
2. cpu-checker: Outputs the specifications of CPU
3. libvirt-clients: a toolkit to manage virtualization platforms/clients and hypervisors
4. qemu: A program that can run the operating system of the machine on different machines
5. qemu-kvm: Runs process using KVM module

All dependencies are installed now run command its time to check if KVM ins installed successfully:

```
$ sudo kvm-ok
```



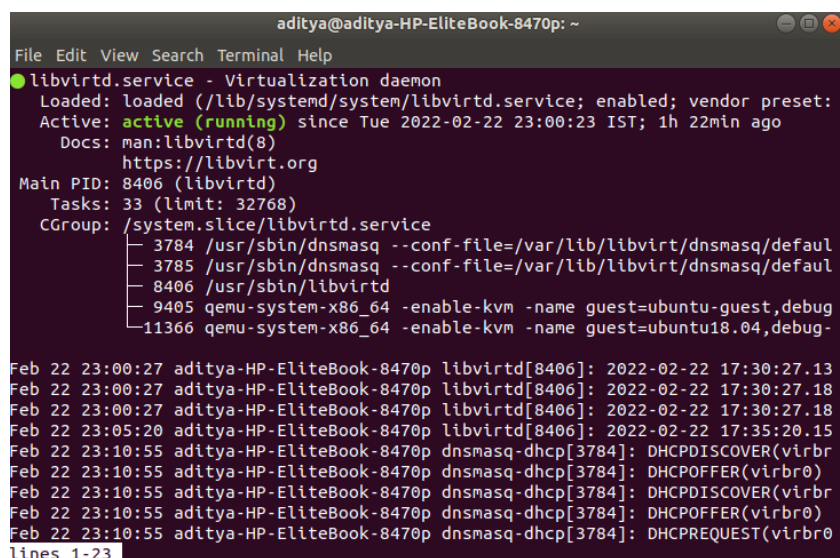
```
aditya@aditya-HP-EliteBook-8470p: ~  
File Edit View Search Terminal Help  
aditya@aditya-HP-EliteBook-8470p:~$ sudo kvm-ok  
[sudo] password for aditya:  
INFO: /dev/kvm exists  
KVM acceleration can be used  
aditya@aditya-HP-EliteBook-8470p:~$
```

Check if KVM is Installed properly

Also, we need to confirm if the virtualization daemon – libvirtd-daemon – is running, to do so enter the command.

```
$ sudo systemctl status libvirtd
```

If the output is not active: running you need to start daemon thread



```
aditya@aditya-HP-EliteBook-8470p: ~  
File Edit View Search Terminal Help  
● libvirtd.service - Virtualization daemon  
   Loaded: loaded (/lib/systemd/system/libvirtd.service; enabled; vendor preset: enabled)  
   Active: active (running) since Tue 2022-02-22 23:00:23 IST; 1h 22min ago  
     Docs: man:libvirtd(8)  
           https://libvirt.org  
  Main PID: 8406 (libvirtd)  
    Tasks: 33 (limit: 32768)  
   CGroup: /system.slice/libvirtd.service  
           └─ 3784 /usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default  
             └─ 3785 /usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/default  
               └─ 8406 /usr/sbin/libvirtd  
                 └─ 9405 qemu-system-x86_64 -enable-kvm -name guest=ubuntu-guest,debug  
                   └─ 11366 qemu-system-x86_64 -enable-kvm -name guest=ubuntu18.04,debug-  
Feb 22 23:00:27 aditya-HP-EliteBook-8470p libvirtd[8406]: 2022-02-22 17:30:27.13  
Feb 22 23:00:27 aditya-HP-EliteBook-8470p libvirtd[8406]: 2022-02-22 17:30:27.18  
Feb 22 23:00:27 aditya-HP-EliteBook-8470p libvirtd[8406]: 2022-02-22 17:30:27.18  
Feb 22 23:05:20 aditya-HP-EliteBook-8470p libvirtd[8406]: 2022-02-22 17:35:20.15  
Feb 22 23:10:55 aditya-HP-EliteBook-8470p dnsmasq-dhcp[3784]: DHCPDISCOVER(virbr  
Feb 22 23:10:55 aditya-HP-EliteBook-8470p dnsmasq-dhcp[3784]: DHCPPOFFER(virbr0  
Feb 22 23:10:55 aditya-HP-EliteBook-8470p dnsmasq-dhcp[3784]: DHCPDISCOVER(virbr  
Feb 22 23:10:55 aditya-HP-EliteBook-8470p dnsmasq-dhcp[3784]: DHCPPOFFER(virbr0  
Feb 22 23:10:55 aditya-HP-EliteBook-8470p dnsmasq-dhcp[3784]: DHCPREQUEST(virbr0  
lines 1-23
```

If the daemon thread is not running enter the following command to start the thread,

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

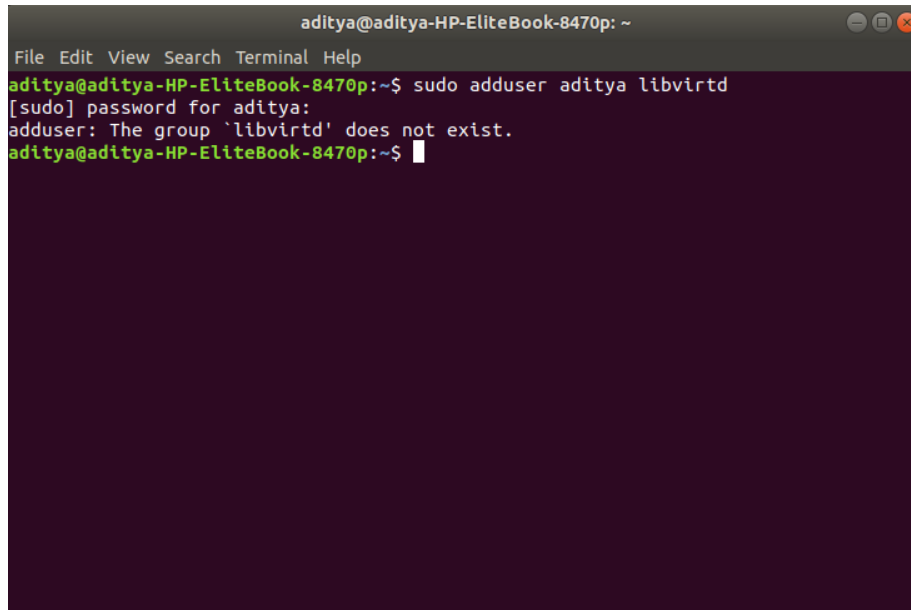
Adding a user to KVM

In this part, we are going to create a user for KVM. To prevent root user from using KVM and root user is only available when root user is a part of libvirt/libvirtd group.

To add a new user to KVM, use the following command,

```
sudo adduser [username] libvirtd
```

[username] enter the username of your choice if the output is



```
aditya@aditya-HP-EliteBook-8470p: ~  
File Edit View Search Terminal Help  
aditya@aditya-HP-EliteBook-8470p:~$ sudo adduser aditya libvirtd  
[sudo] password for aditya:  
adduser: The group 'libvirtd' does not exist.  
aditya@aditya-HP-EliteBook-8470p:~$
```

Adding User to KVM

Then your KVM is already a member of the non-root user and serves the same purpose as libvirtd group then you don't need to add yourself to the group.

Creating Virtual Machine

There are two ways to create a virtual machine

1. Using the command line
 2. Using the graphical interface
- **Create a Virtual Machine via Command Line**

virt-install is a command which is used to create virtual machines in Linux, following is the command which creates a VM.

```
sudo virt-install --name=ubuntu-guest --os-variant=ubuntu20.04 --vcpu=2 --ram=2048 --graphics none --  
location=[local path to iso file] --network bridge:vibr0
```

The above command creates a Ubuntu virtual machine with version 20.04 and the name ubuntu-guest.

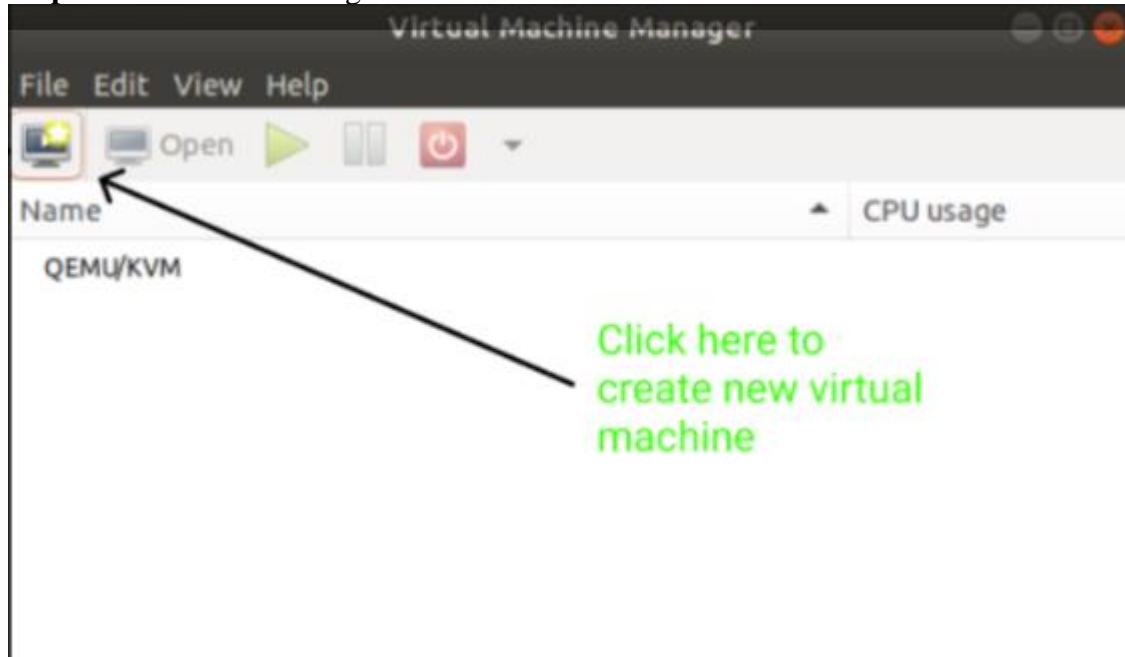
1. Name: specify the name of the virtual machine being created
2. vcpu: Number of virtual CPUs to configure for the guest.
3. Ram: Memory to allocate for guest instance in megabytes. According to your machine, you can specify the given memory of the VM.
4. Graphics spice: If no graphics option is specified, “virt-install” will default to ‘--graphics vnc’ if the display environment variable is set, otherwise ‘--graphics none’ is used.

5. Location: location of iso file on which virtual machine will be built. It can be the path to an ISO image, or to a CDROM device. It can also be a URL from which to fetch/access a minimal boot ISO image.
6. Network-bridge: Connect the guest to the host network, Connect to a bridge device in the host called "BRIDGE".

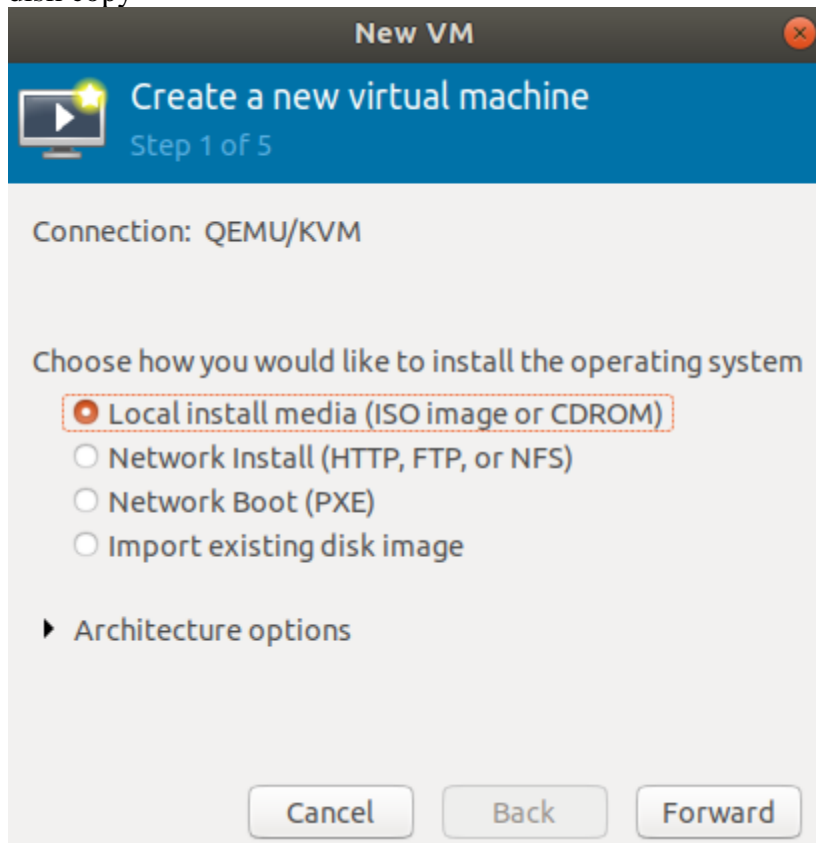
Create a Virtual machine using a graphical interface

If you are not much familiar with the command line don't worry there's another way to create a virtual machine using a tool called virt-manager you can easily create virtual machines. Steps to create VM using virt-manager,

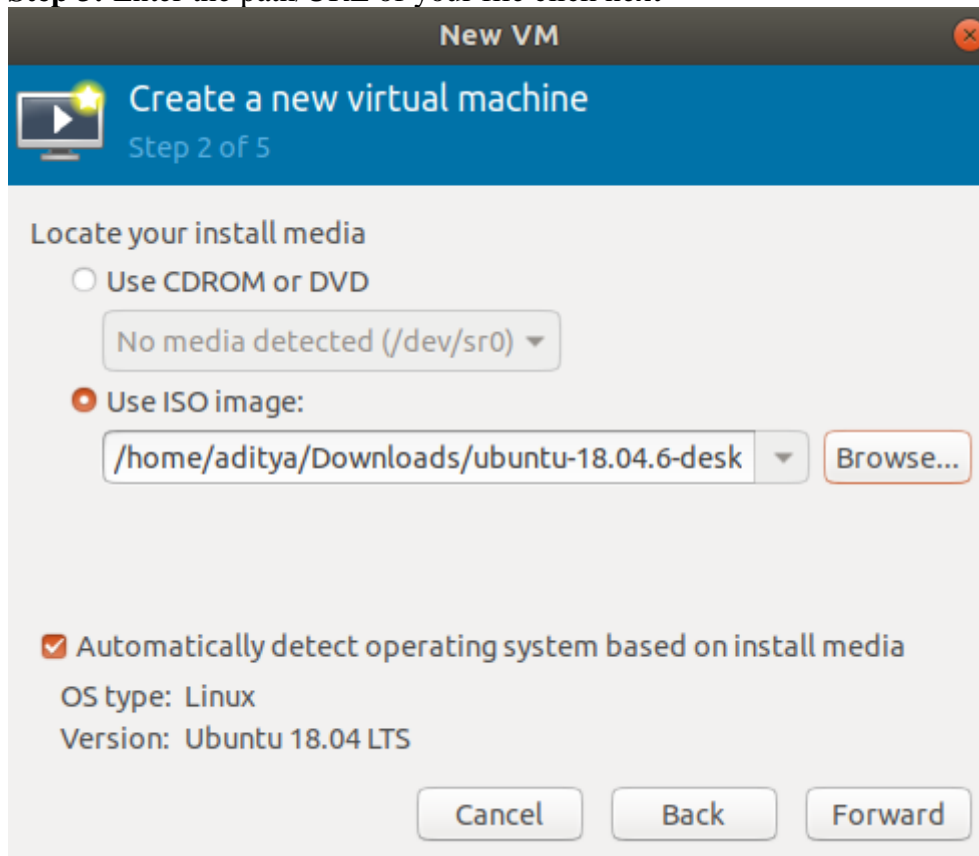
Step 1: Launch virt-manager



Step 2: Choose installation media it can be an iso file of OS, you can install from the network or can be a disk copy



Step 3: Enter the path/URL of your file click next



New VM

Create a new virtual machine
Step 2 of 5

Locate your install media

☐ Use CDROM or DVD

No media detected (/dev/sr0) ▾

☒ Use ISO image:

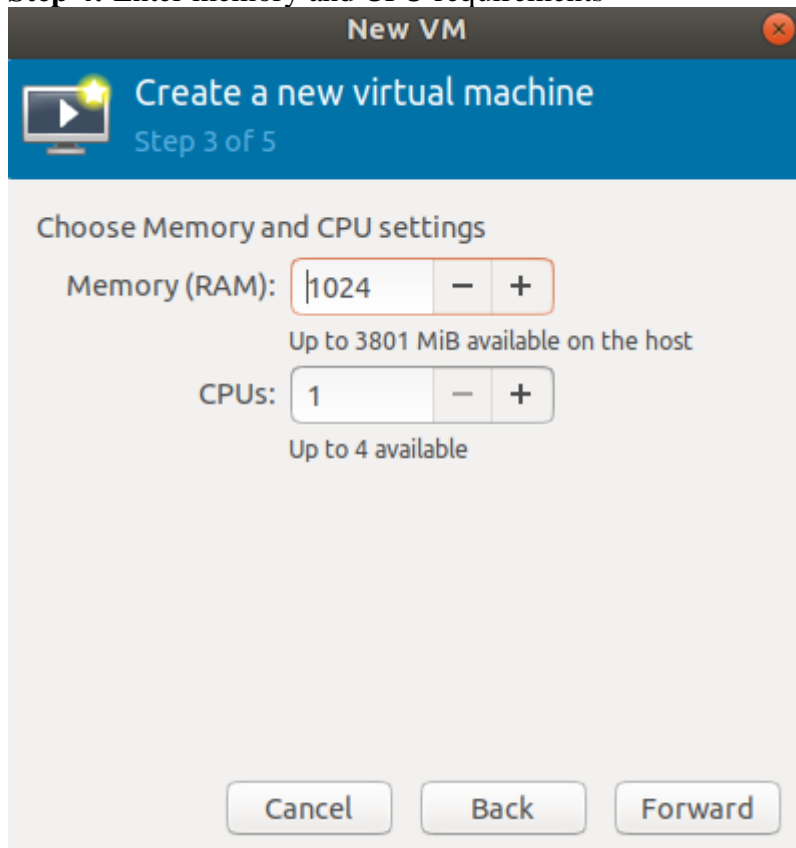
/home/aditya/Downloads/ubuntu-18.04.6-desk ▾ Browse...

☒ Automatically detect operating system based on install media

OS type: Linux
Version: Ubuntu 18.04 LTS

Cancel Back Forward

Step 4: Enter memory and CPU requirements



New VM

Create a new virtual machine
Step 3 of 5

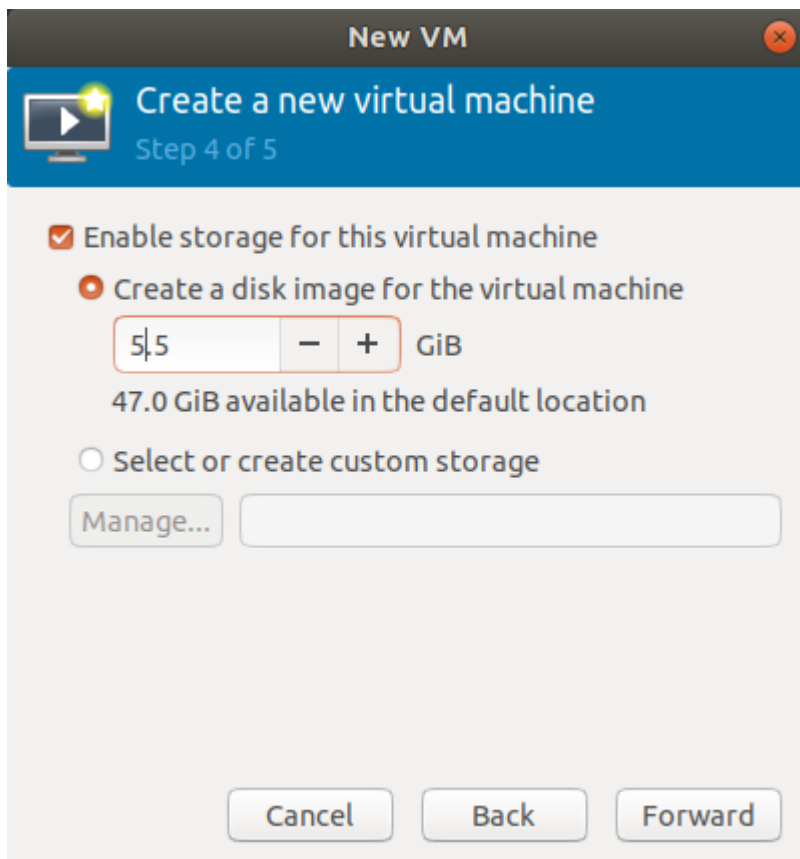
Choose Memory and CPU settings

Memory (RAM): 1024 - +
Up to 3801 MiB available on the host

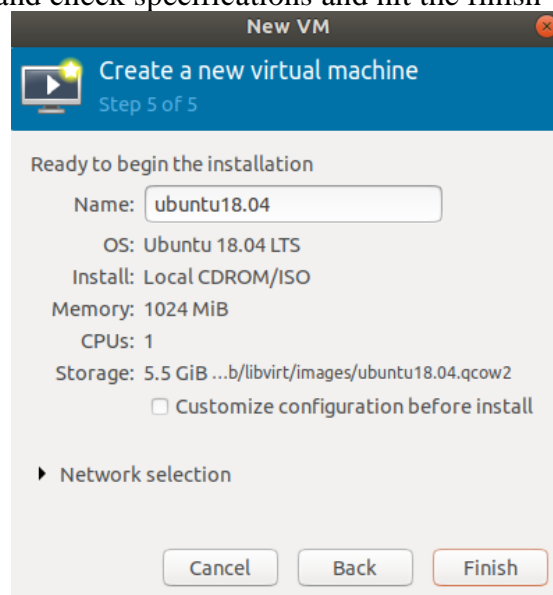
CPUs: 1 - +
Up to 4 available

Cancel Back Forward

Step 5: Enter required disk space



Step 6: Enter the name of VM and check specifications and hit the finish



Now the installations should be completed and the virtual machine should start running.

Result:

Using internet services (it may be any ISP service) , we have completed the task of implemented Virtualization in Cloud by using KVM and VMware

Experiment 11: Write steps for installation of MongoDB Atlas .

Aim: Write simple steps for installation of MongoDB Atlas.

Procedure/ Algorithm:

To begin using MongoDB Atlas, you'll need to do the following:

1. Create a MongoDB Cloud account.
2. Create a MongoDB Atlas cluster.
3. Configure network access and create a cluster user.
4. Connect to the cluster.

Solution:

MongoDB Atlas

Database-as-a-Service (DBaaS) is a service that allows you to set up, deploy, and scale a database without worrying about on-premise physical hardware, software updates, and the details of configuring for performance. With DBaaS, a cloud provider does all that for you—and gets you up and running right away.

MongoDB Atlas is a fully-managed cloud database that handles all the complexity of deploying, managing, and healing your deployments on the cloud service provider of your choice (AWS , Azure, and GCP). MongoDB Atlas is the best way to deploy, run, and scale MongoDB in the cloud. With Atlas, you will have a MongoDB database running with just a few clicks, and in just a few minutes.

Getting Started with MongoDB Atlas

To begin using MongoDB Atlas, you'll need to do the following:

1. Create a MongoDB Cloud account.
2. Create a MongoDB Atlas cluster.
3. Configure network access and create a cluster user.
4. Connect to the cluster.

MongoDB Atlas has a free tier, so you won't need any payment or credit card information.

Now, let's get started!

Creating a MongoDB Atlas Account

Registration

In order to create an Atlas account, navigate to <https://www.mongodb.com/cloud/atlas/register>.

You can sign up using your Google account. This would be the preferred method; however, you can also register using your email address.

Organizations and Projects

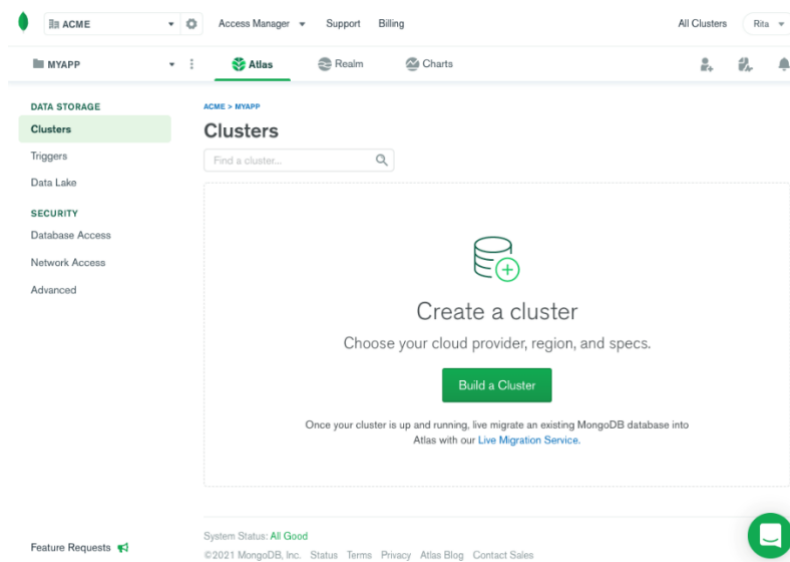
At the end of the sign-up process, you will be prompted to create an organization and a project. Organizations allow you to group and define users and teams, and grant them access to the different projects. Projects allow you to define and organize resources such as database clusters, triggers, and data lakes. A common way to use projects is to define each environment as a project. For example, you can have a separate project for development, testing, and production.

You can skip this step and go directly to the management console for MongoDB Atlas. But you will need an organization and a project in order to create a database cluster, so it makes sense to do this step now. If you decide to skip it, you'll be able to create an organization and a project later.

Setting Up a Cluster in MongoDB Atlas

Once you have an Atlas account and have created an organization and a project, you'll be able to create a database cluster.

Make sure you have the desired organization and project selected in the top navigation dropdowns. Then, select "Clusters" from the left navigation menu and click on the Build a Cluster button.



You'll be presented with a choice of Shared Cluster, Dedicated Cluster, and Multi-Cloud & Multi-Region Cluster.

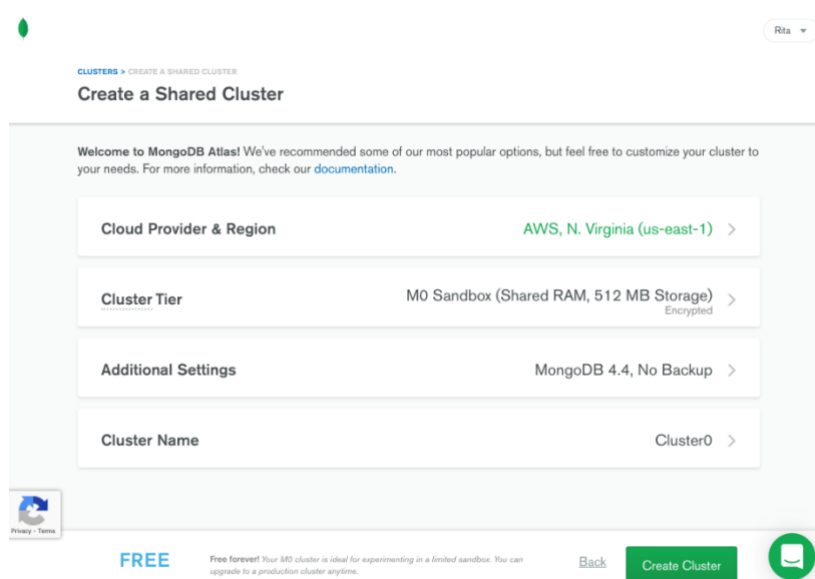
Shared Cluster is the least expensive (or free, depending on the usage) but it utilizes shared hardware resources and network.

Dedicated Cluster provides you with a dedicated set of hardware and network isolation as well as the option to scale automatically within a single region.

The Multi-Cloud & Multi-Region Cluster builds on top of what the dedicated cluster provides. It offers the best availability since it can replicate data across multiple geographic regions. It also allows the creation of multi-cloud clusters using any combination of cloud providers: AWS, Azure, and GCP.

If you'd like to explore a little with the free tier, select the Shared Cluster.

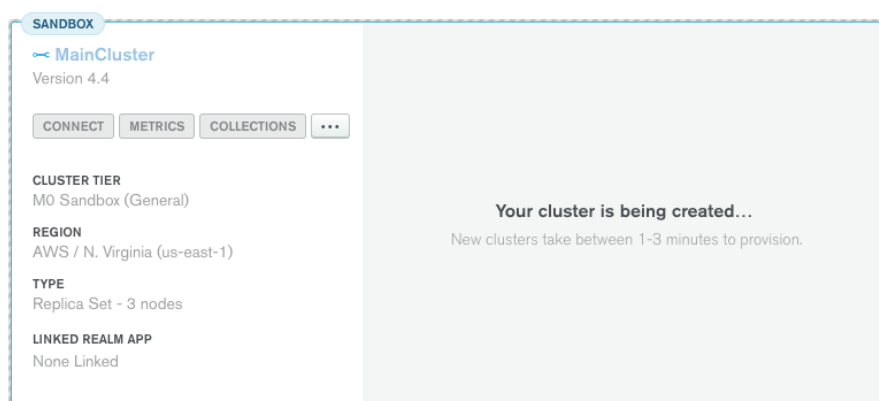
Once you have selected the type of cluster, you'll be able to choose from the top three cloud providers (Amazon Web Services, Microsoft Azure, and Google Cloud Platform) and select a region for hosting the cluster.



The screenshot shows the 'Create a Shared Cluster' form in the MongoDB Atlas console. At the top, there's a breadcrumb 'CLUSTERS > CREATE A SHARED CLUSTER' and a user profile 'Rita'. The form title is 'Create a Shared Cluster'. Below it, a welcome message states: 'Welcome to MongoDB Atlas! We've recommended some of our most popular options, but feel free to customize your cluster to your needs. For more information, check our [documentation](#).' The form contains four sections, each with a title and a value followed by a chevron icon: 'Cloud Provider & Region' with 'AWS, N. Virginia (us-east-1)', 'Cluster Tier' with 'M0 Sandbox (Shared RAM, 512 MB Storage) Encrypted', 'Additional Settings' with 'MongoDB 4.4, No Backup', and 'Cluster Name' with 'Cluster0'. At the bottom left, there are links for 'Privacy - Terms'. In the center, the word 'FREE' is displayed in large blue letters, followed by a note: 'Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.' To the right of this note are 'Back' and 'Create Cluster' buttons. A green chat icon is in the bottom right corner.

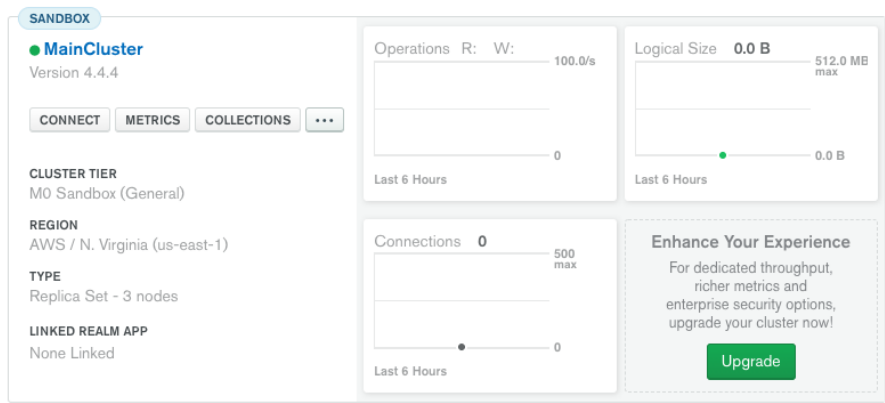
You'll also be able to select the cluster tier and additional settings, such as turn on backup and cluster name. Some options, such as the MongoDB version cloud backups, are only available with the paid cluster tiers.

Once you are satisfied with your selection, click the Create Cluster button. It may take a couple of minutes for Atlas to launch your cluster in the selected cloud hosting provider.



The screenshot shows the 'MainCluster' overview page in the MongoDB Atlas console. The page is titled 'MainCluster' and 'Version 4.4'. It has a 'SANDBOX' label in the top left corner. Below the title, there are three buttons: 'CONNECT', 'METRICS', and 'COLLECTIONS', followed by a three-dot menu. On the left side, there are several sections with labels and values: 'CLUSTER TIER' with 'M0 Sandbox (General)', 'REGION' with 'AWS / N. Virginia (us-east-1)', 'TYPE' with 'Replica Set - 3 nodes', and 'LINKED REALM APP' with 'None Linked'. The right side of the page has a large light blue area with the text 'Your cluster is being created...' and a note below it: 'New clusters take between 1-3 minutes to provision.'

When the cluster is ready, you will see the cluster name with a green circle next to it, indicating a successful setup. You will also see several metrics next to it indicating connections, operations, and the size of your cluster.



Next, let's connect to the cluster.

Accessing a MongoDB Atlas Cluster

In order to access your MongoDB Atlas cluster, you'll need to enable network access for your network or IP address and create a database user for connecting to the cluster. After that, you can generate a connection string for your application or script.

Connect to MainCluster

Setup connection security > Choose a connection method > Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more](#)

You can't connect yet. Set up your firewall access and user security permission below.

1 Add a connection IP address

2 Create a Database User

This first user will have [atlasAdmin](#) permissions for this project.
Keep your credentials handy, you'll need them for the next step.

Username

Password

Allowing Access to Your IP Address

For security reasons, new database clusters do not have network access enabled by default. You need to enable network access explicitly by whitelisting the addresses that will connect to the cluster.

Each entry can be an IP address, a subnet, or you can enable access from any location. In general, you would grant access only to a list of subnets or IP addresses rather than grant access to any location. This limits the connections your cluster accepts, making it more secure.

To enable network access to your cluster, click on the Connect button from the clusters view in the Atlas management console. This will open up the connection settings wizard.

In order to allow access from your current IP address, click on the Add your current IP address button. If you need to access it from a different IP address or subnet, click on the Add a different IP address button and enter the IP or a subnet using the CIDR notation, such as 172.10.1.0/24.

Creating a Cluster User

In order to connect to the database from a script or an application, you must first create a MongoDB database user. The database user allows you to connect and use the databases. Please note this is *separate* from the user that logs in and manages the clusters and resources in Atlas.

Database users are created per project and have access to all the clusters in the project. You can also assign different roles and privileges to the database users. Note that the first user you create will automatically be granted administrative privileges.

Right below the network access settings, you can create a database user. First enter the username and password and then click on the Create database user button.

If you later need to add more users to the project, you can do it from the Security tab.

Generating a Database Connection String

Depending on your application, you may need to install a driver (library) corresponding to your platform in order to connect to a cluster in Atlas. You can see the full list of supported drivers [here](#). If you're using Compass (the MongoDB UI application) or the mongo shell application, the drivers are already built in.

Regardless of your application, you'll need to generate a database connection string for your cluster. If you are just creating your cluster, the last step in the process will allow you to create a connection string. Once you have enabled network access and created a database user, you can click on the Choose connection method button, which will allow you to generate a connection string for your application. You can also create the connection string by clicking on the Connect button on your cluster from the Clusters tab in Atlas.

Connect to MainCluster


✓ Setup connection security


Choose a connection method


Connect

Choose a connection method [View documentation](#)

Get your pre-formatted connection string by selecting your tool below.

**Connect with the mongo shell**
Interact with your cluster using MongoDB's interactive Javascript interface

**Connect your application**
Connect your application to your cluster using MongoDB's native drivers

**Connect using MongoDB Compass**
Explore, modify, and visualize your data with MongoDB's GUI

Go Back

Close

Click on the Connect your application button and then select the driver, such as Node.js, Python, or another language. Then, select the version to generate the connection string for your application. You can even check the "Include full driver code example" option in order to generate the code to test the connectivity.

Connect to MainCluster

✓ Setup connection security

✓ Choose a connection method

Connect

1

Select your driver and version

DRIVER

VERSION

Python

3.6 or later

2

Add your connection string into your application code

☐ Include full driver code example

```
mongodb+srv://<username>:
<password>@maincluster.3pva0.mongodb.net/myFirstDatabase?
retryWrites=true&w=majority
```

Replace **<password>** with the password for the **<username>** user. Replace **myFirstDatabase** with the name of the database that connections will use by default. Ensure any option params are **URL encoded**.

Having trouble connecting? [View our troubleshooting documentation](#)

Go Back

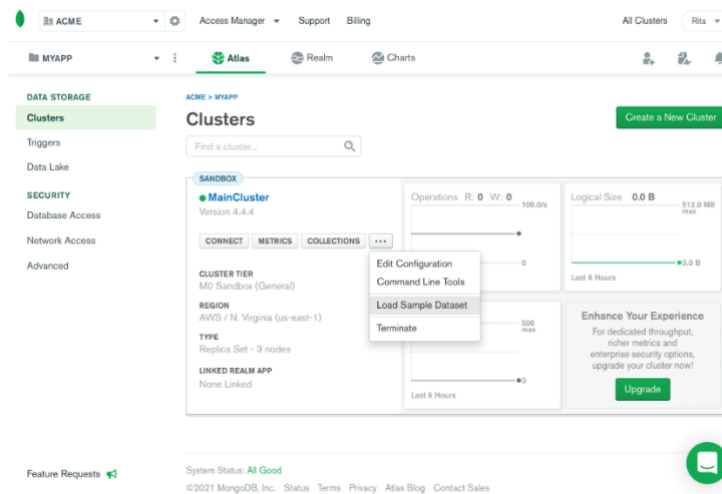
Close

Note that the connection string generated does not include the actual cluster user login. You will need to replace the **<username>** and **<password>** with your actual username and password. You will also need to replace **MyFirstDatabase** with an actual database name in your cluster.

Sample Data

If you are just starting with MongoDB, you may want to load a sample data set.

pg. 38



From the cluster view in the Atlas management console, click on the ellipsis button [...] and select “Load Sample Data” from the menu. Then confirm your selection.

This will load a few sample collections that you can use to run test queries and learn more about MongoDB. You can view the sample collections loaded by clicking on the Collections button on your cluster from the Clusters tab in Atlas.

Input/Output:

To begin using MongoDB Atlas, you need to follow these steps and give appropriate input.

1. Create a MongoDB Cloud account.
2. Create a MongoDB Atlas cluster.
3. Configure network access and create a cluster user.
4. Connect to the cluster.

Result: By using appropriate steps we have been installed MongoDB Atlas successfully and it’s working very smoothly.

Experiment 12: How MongoDB CRUD operations works? What are the syntax for CRUD operation using MongoDB database.

Aim: using MongoDB database write commands for CRUD operation.

Algorithm/Procedure:

1. Create a MongoDB database
2. Install the required dependencies (express, mongoose)
3. Set up the MongoDB database connection
4. Create a Mongoose schema for the data
5. Create routes to handle CRUD operations
6. Test the application

Program/ Solution:

CRUD is an acronym for

C=Create,
R=Read,
U= Update,
D= Delete.

CRUD operations in MongoDB are used to manipulate data in databases.

CRUD In MongoDB

CREATE, READ, UPDATE, and DELETE actions are referred to as CRUD operations in MongoDB. These are the basic operations used to alter data in databases. In MongoDB, the term "CRUD operations" refers to the standard set of database operations used to work with collections of data.

Create: To add new documents to a collection, use the Create operation.

Read: Data from a collection is retrieved using the Read operation.

Update: The Update operation is used to edit existing documents in a collection.

Delete: A collection of documents can be deleted using the Delete procedure.

These four basic procedures can be used to accomplish a wide range of database operations in MongoDB. These activities, for instance, can be used to add new records, get data, edit records, and remove records.

Performing CRUD Operations in MongoDB

Before we start exploring the CRUD methods. Let's first set up the db and collection which we are going to use.

Creating a New Database:

To create a new database, you can simply run any command against a non-existing database, and MongoDB will automatically create it for you.

Example:

Let us create a database name userdb.

```
use userdb;
```

Output:

```
switched to db userdb
```

Creating a New Collection:

To create a new collection, you can use the "createCollection" method.

Syntax:

```
db.createCollection(name, options)
```

Example:

```
db.createCollection("users")
```

Output:

```
{ "ok" : 1 }
```

Create Operation

There are two ways to create new documents to a collection in MongoDB:

1. insertOne():

Adding a single document to a collection is done using this method. A document to be added is the only argument it accepts, and it returns a result object with details about the insertion.

Syntax:

```
db.collectionName.insertOne()
```

Example:

```
db.users.insertOne({  
  name: "Angela",  
  age: 27,  
});
```

Output:

```
{  
  "acknowledged": true,  
  "insertedId" : ObjectId("64255335abfa5ebad9b5e2f7")  
}
```

2. insertMany()

This method is used to insert multiple documents into a collection at once. It takes an array of documents as its argument and returns a result object that contains information about the insertion.

Syntax:

```
db.collectionName.insertMany();
```

Example:

```
db.users.insertMany([
  {
    name: "Angela",
    age: 27,
  },
  {
    name: "Dwight",
    age: 30,
  },
  {
    name: "Jim",
    age: 29,
  }
]);
```

Output:

```
{
  "acknowledged": true,
  "insertedIds" : [
    ObjectId("6425546aabfa5ebad9b5e2f8"),
    ObjectId("6425546aabfa5ebad9b5e2f9"),
    ObjectId("6425546aabfa5ebad9b5e2fa")
  ]
}
```

Both the `insertOne()` and `insertMany()` methods return a result object that contains information about the insertion operation. The result object includes the number of documents inserted, the unique `_id` field value of each inserted document, and any write errors that occurred during the operation.

Read Operations

In MongoDB, read operations are used to retrieve data from the database.

1. find()

The `find()` method is used to retrieve data from a collection. It returns a cursor that can be iterated to access all the documents that match the specified query criteria.

Syntax:

```
db.collectionName.find(query, projection)
```

Query - It specifies the selection criteria for the documents to be retrieved. It is an object that contains one or more key-value pairs, where each key represents a field in the document and the value represents the value to match for that field.

Projection - It specifies which fields to include or exclude in the result set. It is an object that contains one or more key-value pairs, where each key represents a field in the document and the value represents whether to include (1) or exclude (0) the field in the result set.

Note: Both query and projection are optional.

Example:

```
db.users.find()
```

Output:

```
{ "_id" : ObjectId("64255335abfa5ebad9b5e2f7"), "name" : "Angela", "age" : 27 }  
{ "_id" : ObjectId("6425546aabfa5ebad9b5e2f8"), "name" : "Angela", "age" : 27 }  
{ "_id" : ObjectId("6425546aabfa5ebad9b5e2f9"), "name" : "Dwight", "age" : 30 }  
{ "_id" : ObjectId("6425546aabfa5ebad9b5e2fa"), "name" : "Jim", "age" : 29 }
```

The above example will retrieve all documents from the collection without applying any filters or projecting any specific fields.

Example:

```
db.users.find({ age: { $gt: 29 } }, { name: 1, age: 1 })
```

Output:

```
{ "_id" : ObjectId("6425546aabfa5ebad9b5e2f9"), "name" : "Dwight", "age" : 30 }
```

This command will return all the documents in the "users" collection where the age is greater than 29, and only return the "name" and "age" fields.

2. findOne()

The findOne() method returns a single document object, or null if no document is found. You can pass a query object to this method to filter the results.

Syntax:

```
db.collectionName.findOne()
```

Example:

```
db.users.findOne({ name: "Jim" })
```

Output:

```
{ "_id" : ObjectId("6425546aabfa5ebad9b5e2fa"), "name" : "Jim", "age" : 29 }
```

This returns the first document in the user's collection where the name field is "Jim".

Update Operations

In MongoDB, the "update" operation is used to modify existing documents in a collection.

Methods:

There are several ways to perform an update operation, including the following:

1. updateOne()

The updateOne() method is used to update a single document that matches a specified filter.

Syntax:

```
db.collectionName.updateOne(filter, update, options)
```

Options include the following parameters:

- **Upsert** is an optional boolean that specifies whether to insert a new document if no document matches the filter. If upsert is set to true and no document matches the filter, a new document will be inserted. The default value of upsert is false.
- **WriteConcern** is an optional document that specifies the level of acknowledgement requested from MongoDB for write operations. If not specified, the default write concern will be used.

Example:

```
db.users.updateOne({ name: "Angela" }, { $set: { email: "angela@gmail.com" } })
```

Output

```
{ "_id" : ObjectId("64255335abfa5ebad9b5e2f7"), "name" : "Angela", "age" : 27, "email" : "angela@gmail.com" }
```

In the example, we have used the \$set operation.

Following are a few of the many available operations:

- **\$set**: Sets the value of a field in a document. If the field does not exist, the set will create it.
- **\$unset**: Removes a field from a document.
- **\$inc**: Increments the value of a field in a document by a specified amount.
- **\$push**: Adds an element to the end of an array field in a document. If the field does not exist, push will create it as an array with the specified element.
- **\$pull**: Removes all occurrences of a specified value from an array field in a document.

This command will update the email of the document in the "users" collection where the name is "Angela" to angela@gmail.com.

2. updateMany

The `updateMany()` method is used to update multiple documents that match a specified filter.

Syntax:

```
db.collectionName.updateMany(filter, update, options)
```

Example:

```
db.users.updateMany({ age: { $lt: 30 } }, { $set: { status: "active" } })
```

Output:

```
{ "_id" : ObjectId("64255335abfa5ebad9b5e2f7"), "name" : "Angela", "age" : 27, "email" :  
"angela@gmail.com", "status" : "active" }  
{ "_id" : ObjectId("6425546aabfa5ebad9b5e2f8"), "name" : "Angela", "age" : 27, "status" : "active" }  
{ "_id" : ObjectId("6425546aabfa5ebad9b5e2f9"), "name" : "Dwight", "age" : 30 }  
{ "_id" : ObjectId("6425546aabfa5ebad9b5e2fa"), "name" : "Jim", "age" : 29, "status" : "active" }
```

This command will update the status of all documents in the "users" collection where the age is less than 30 to "active".

Delete Operations

In MongoDB, the "delete" operation is used to remove documents from a collection.

There are several ways to perform a delete operation, including the following:

1. deleteOne()

The `deleteOne()` method is used to remove a single document that matches a specified filter.

Syntax:

```
db.collectionName.deleteOne(filter, options)
```

filter: Specifies deletion criteria using query operators. Specify an empty document `{ }` to delete the first document returned in the collection.

Options:

- **WriteConcern (Optional):** A document expressing the write concern. Omit to use the default write concern.
- **Collation (Optional):** Specifies the collation to use for the operation. Collation allows users to specify language-specific rules for string comparison, such as rules for letter case and accent marks.
- **Hint (Optional):** A document or string that specifies the index to use to support the query predicate.

Example:

```
db.users.deleteOne({ name: "Angela" })
```

Output:

```
{ "_id" : ObjectId("6425546aabfa5ebad9b5e2f8"), "name" : "Angela", "age" : 27, "status" : "active" }  
{ "_id" : ObjectId("6425546aabfa5ebad9b5e2f9"), "name" : "Dwight", "age" : 30 }  
{ "_id" : ObjectId("6425546aabfa5ebad9b5e2fa"), "name" : "Jim", "age" : 29, "status" : "active" }
```

This command will remove the first document in the "users" collection where the name is "Angela".

2. deleteMany()

The deleteMany() method is used to remove multiple documents that match a specified filter.

Syntax:

```
db.collectionName.deleteMany(filter, options)
```

Example:

```
db.users.deleteMany({ age: { $lt: 30 } })
```

Output:

```
{ "_id" : ObjectId("6425546aabfa5ebad9b5e2f9"), "name" : "Dwight", "age" : 30 }
```

This command will remove all documents in the "users" collection where the age is less than 30.

3. drop()

The drop() method is used to remove an entire collection.

Syntax:

```
db.collectionName.drop()
```

Example:

```
db.users.drop()
```

Output:

```
true
```

This command will remove the users collection.

Note: This operation is irreversible, and all data in the collection will be permanently deleted.

Libraries using for operation-

- Creating new documents can be done using insertOne() or insertMany().
- Reading documents can be done using find() or findOne().
- Updating documents can be done using updateOne() or updateMany().
- Deleting documents can be done using deleteOne() or deleteMany().

Result:

Commands run successfully in MongoDB Database.

Experiment 13: Write a simple code for Data modeling in MongoDB .

Aim: Using MongoDB coding , assume we are getting the details of employees in three different documents namely, Personal_details, Contact and, Address, you can embed all the three documents in a single one as shown below .

Procedure/Algorithm:

1. Start
2. Identify your application's workload.
3. Map relationships between objects in your collections.
4. Apply design patterns.
5. End

Program:

MongoDB provides two types of data models: — Embedded data model and Normalized data model. Based on the requirement, you can use either of the models while preparing your document.

Embedded Data Model

In this model, you can have (embed) all the related data in a single document, it is also known as de-normalized data model.

For example, assume we are getting the details of employees in three different documents namely, Personal_details, Contact and, Address, you can embed all the three documents in a single one as shown below –

```
{
  _id: ,
  Emp_ID: "10025AE336"
  Personal_details: {
    "firstName": "Romin",
    "lastName": "Irani",
    "preferredFullName": "Romin Irani",
  },
  Contact: {
    phoneNumber: "408-1234567",
    emailAddress: "romin.k.irani@gmail.com"
  },
  Address: {
    city: "Hyderabad",
    Area: "Madapur",
    State: "Telangana"
  }
}
```

Normalized Data Model

In this model, you can refer the sub documents in the original document, using references. For example, you can re-write the above document in the normalized model as:

Employee:

```
{
  "userId": "rirani",
  "jobTitleName": "Developer",
}
```

Personal_details:

```
{
  "userId": "rirani",
  "jobTitleName": "Developer",
  "firstName": "Romin",
  "lastName": "Irani",
  "preferredFullName": "Romin Irani"
}
```

Contact:

```
{
  "userId": "rirani",
  "jobTitleName": "Developer",
  "phoneNumber": "408-1234567",
  "emailAddress": "romin.k.irani@gmail.com"
}
```

Address:

```
{
  "userId": "rirani",
  "jobTitleName": "Developer",
  "city": "Hyderabad",
  "Area": "Madapur",
  "State": "Telangana"
}
```

Input:

Embedded Data to inserted into MongoDB

Output:

Normalized data to be resulted as mentioned above.

Result: Normalized command successfully.

Experiment 14 : Write steps for Creation of Queries in MongoDB

Aim: Program to create database in MongoDB

Procedure/ Algorithm/

How and when to create database

If there is no existing database, the following command is used to create a new database.

Syntax:

1. use DATABASE_NAME

If the database already exists, it will return the existing database.

Let's take an example to demonstrate how a database is created in MongoDB. In the following example, we are going to create a database "srmdb".

Program:

1. >use srmdb
Switched to db srmdb

To **check the currently selected database**, use the command db:

1. >db srmdb

To **check the database list**, use the command show dbs:

1. >show dbs
local 0.078GB

Here, your created database "srmdb" is not present in the list, **insert at least one document** into it to display database:

Input:

1. >db.movie.**insert**({ "name": "srmdb" })
WriteResult({ "nInserted": 1 })

Output:

1. >show dbs
srmdb 0.078GB
local 0.078GB

Result: Database Created by MongoDB Successfully.

Experiment 15 : Using Creation of Queries in MongoDB , write a program in mongodb to create a Collection:

Aim: Using MongoDB , write a program to create a Collection.

Procedure:

Step 1. Create Database if required

Step 2. Use Database using command **>use DbNew**

Step 3. Create Collection

Step 4. Check Collection , that command run successfully or not.

Program:

Syntax to create collection:

db.createCollection(name, options)

Let's take an **example to create collection**. In this example, we are going to create a collection name SRMDATA.

Input:

1. >use test

```
switched to db test
```

1. >db.createCollection("SRMDATA")

Output

```
{ "ok" : 1 }
```

To **check the created collection**, use the command "show collections".

1. >show collections

```
SRMDATA
```

Result:

A command to create a collection of name: SRMDATA has been executed successfully.

Experiment 16: Using Creation of Queries in MongoDB , write a program in mongodb to insert a document based data.

Aim: Using MongoDB , write a program to insert document.

Procedure:

In MongoDB, the **db.collection.insert()** method is used to add or insert new documents into a collection in your database.

There are also two methods "**db.collection.update()**" method and "**db.collection.save()**" method used for the same purpose. These methods add new documents through an operation called *upsert*.

Syntax

```
>>db.COLLECTION_NAME.insert(document)
```

Program:

```
db.srmdata.insert(  
  {  
    course: "BCA",  
    details: {  
      duration: "3 Year",  
      Trainer: "SRM Institute"  
    },  
    Batch: [ { size: "Small", qty: 15 }, { size: "Medium", qty: 25 } ],  
    category: "Computer Applications"  
  }  
)
```

Output:

```
WriteResult({ "nInserted" : 1 })
```

Check Inserted Data

Check the inserted documents

If the insertion is successful, you can view the inserted document by the following query.

Input Command

```
>>db.srmdata.find()
```

You will get the inserted document in return.

Output:

```
{ "_id" : ObjectId("56482d3e27e53d2dbc93cef8"), "course" : "BCA", "details" :  
  { "duration" : "3 Years", "Trainer" : "SRM Institute" }, "Batch" :  
  [ { "size" : "Small", "qty" : 15 }, { "size" : "Medium", "qty" : 25 } ],  
  "category" : "Computer Applications" }
```

Result: *Using MongoDB , Record inserted successfully.*

Experiment 17 : Using Creation of Queries in MongoDB , write a program in mongodb to update a document .

Aim: Using MongoDB , write a program to update document.

Procedure:

1. Insert any document using MongoDB
2. Update document using command
3. Check record updated or not.

Program:

Update the existing course "java" into "android":

```
>db.javatpoint.update({'category':'Computer Application'},{$set:{'course':'BCA '}})
```

Check the updated document in the collection:

```
>db.javatpoint.find()
```

Output:

```
{ "_id" : ObjectId("56482d3e27e53d2dbc93cef8"), "course" : "BCA", "details" :  
{ "duration" : "3 Years", "Trainer" : "SRM Institute" }, "Batch" :  
[ {"size" : "Small", "qty" : 15 }, { "size" : "Medium", "qty" : 25 } ],  
"category" : "BCA" }
```

Result: Command for updating a document executed successfully.

Experiment 18. Write a program to sort a single field in Mongodb

Aim: Write a program in MongoDB to sort a single field .

Procedure/ Algorithm:

1. Insert a document.
2. Sort a document according to field by using `sort()` method.
3. Print sorting document.

Note: The `sort()` method allows you to sort the matching documents by one or more fields (field1, field2, ...) in ascending or descending order.

Program:

Program to insert a document

Insert document with name : student

```
db.students.insertMany([
  { id: 1, name: 'Ryan', gender: 'M' },
  { id: 2, name: 'Jyoti', gender: 'F' },
  { id: 3, name: 'Aditya,', gender: 'M' },
  { id: 4, name: 'Ruby,', gender: 'F' }
]);
```

Input:

// here 1 is using for ascending order

```
db.students.find().sort({"name":1})
```

for Descending order use(-1)

```
db.students.find().sort({"name":-1})
```

output:

```
mycompiler_mongodb> ... .. {
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("6598295069d227c362f1c1f4"),
    '1': ObjectId("6598295069d227c362f1c1f5"),
    '2': ObjectId("6598295069d227c362f1c1f6"),
    '3': ObjectId("6598295069d227c362f1c1f7")
  }
}
mycompiler_mongodb> [
  {
    _id: ObjectId("6598295069d227c362f1c1f6"),
    id: 3,
    name: 'Aditya,',
    gender: 'M'
  },
  {
    _id: ObjectId("6598295069d227c362f1c1f7"),
    id: 4,
    name: 'Ruby,',
    gender: 'F'
  }
]
```

```
{
  _id: ObjectId("6598295069d227c362f1c1f5"),
  id: 2,
  name: 'Jyoti',
  gender: 'F'
},
{
  _id: ObjectId("6598295069d227c362f1c1f7"),
  id: 4,
  name: 'Ruby,',
  gender: 'F'
},
{
  _id: ObjectId("6598295069d227c362f1c1f4"),
  id: 1,
  name: 'Ryan',
  gender: 'M'
}
]
mycompiler_mongodb>
```

[Execution complete with exit code 0]

Result: by using MongoDB , a single field has been sorted ascending order. If we required it in descending order then we have to use -1 in place of 1.