Practical: 1

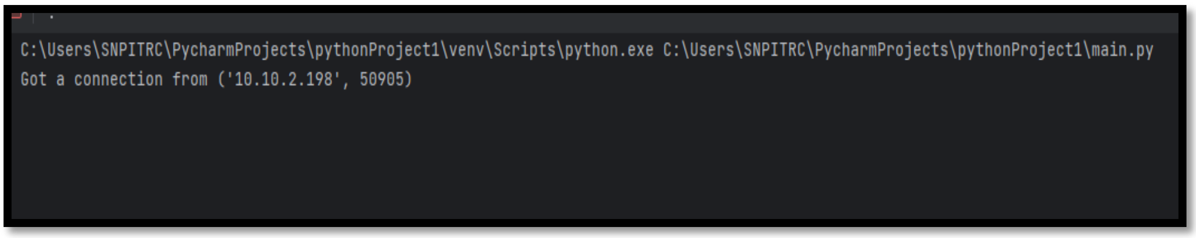
Aim: Write a Program to implement Concurrent Echo Client Server Application.

Code:

Echo Server:

```
# server.py

import socket

import time

# create a socket object

serversocket = socket.socket(

    socket.AF_INET, socket.SOCK_STREAM)

# get local machine name

host = socket.gethostname()

port = 9999

# bind to the port

serversocket.bind((host, port))

# queue up to 5 requests

serversocket.listen(5)

while True:

    # establish a connection

    clientsocket, addr = serversocket.accept()

    print("Got a connection from %s" % str(addr))

    currentTime = time.ctime(time.time()) + "\r\n"

    clientsocket.send(currentTime.encode('ascii'))

    clientsocket.close()
```

Output:



```
C:\Users\SNPITRC\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:\Users\SNPITRC\PycharmProjects\pythonProject1\main.py
Got a connection from ('10.10.2.198', 50905)
```

Echo Client:

import socket

# create a socket object

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# get local machine name

host = '10.10.2.200'

port = 9999

# connection to hostname on the port.

s.connect((host, port))

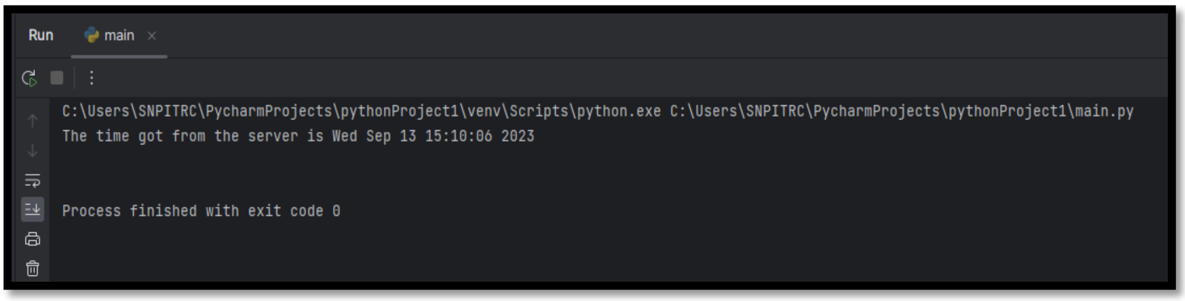
# Receive no more than 1024 bytes

tm = s.recv(1024)

s.close()

print("The time got from the server is %s" % tm.decode('ascii'))

Output:



```
C:\Users\SNPITRC\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:\Users\SNPITRC\PycharmProjects\pythonProject1\main.py
The time got from the server is Wed Sep 13 15:10:06 2023


Process finished with exit code 0
```

Practical: 2

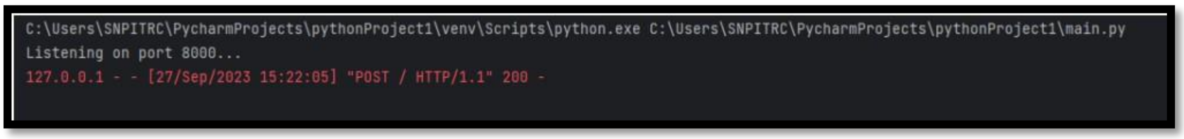
Aim: Write at least 2 Programs for Remote Procedure call.

Code:

Server Side:

```
from xmlrpc.server import SimpleXMLRPCServer

def add(x, y):

return x + y

server = SimpleXMLRPCServer(("127.0.0.1", 8000))

print("Listening on port 8000...")

server.register_function(add, "add")

server.serve_forever()
```

Output:

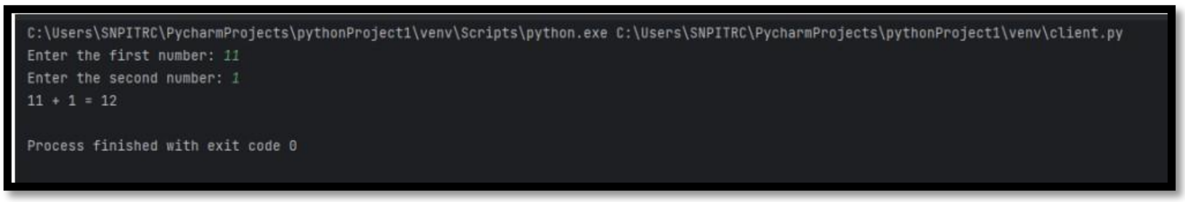


```
C:\Users\SNPITRC\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:\Users\SNPITRC\PycharmProjects\pythonProject1\main.py
Listening on port 8000...
127.0.0.1 - - [27/Sep/2023 15:22:05] "POST / HTTP/1.1" 200 -
```

Client Side:

```
import xmlrpc.client

proxy = xmlrpc.client.ServerProxy("http://localhost:8000/")

x = int(input("Enter the first number: "))

y = int(input("Enter the second number: "))

print(f"{x} + {y} = {str(proxy.add(x, y))}")
```

Output:



```
C:\Users\SNPITRC\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:\Users\SNPITRC\PycharmProjects\pythonProject1\venv\client.py
Enter the first number: 11
Enter the second number: 1
11 + 1 = 12

Process finished with exit code 0
```

Practical: 3

Aim: Write at least 2 Programs for Remote Method Invocation.

Code:

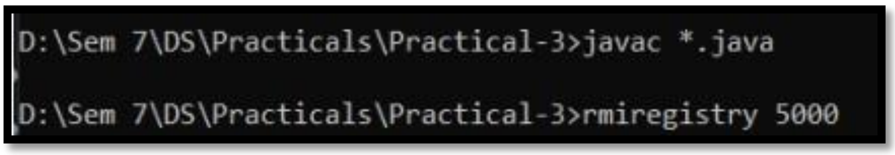
Adder.java

```
import java.rmi.*;
public interface Adder extends Remote
{
public int add(int x,int y)throws RemoteException;
}
```

AdderRemote.java

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder
{
AdderRemote()throws RemoteException
{
super();
}
public int add(int x,int y){return x+y;}
}
```

Output:

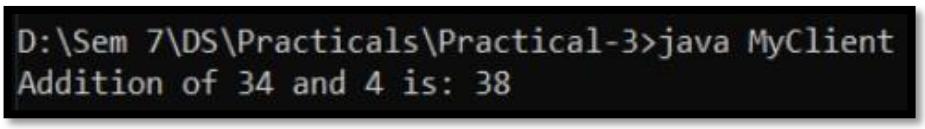


```
D:\Sem 7\DS\Practicals\Practical-3>javac *.java

D:\Sem 7\DS\Practicals\Practical-3>rmiregistry 5000
```

MyClient.java

```
import java.rmi.*;

public class MyClient

{

public static void main(String args[])

{

try{

Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/manan");

System.out.println("Addition of 34 and 4 is: "+stub.add(34,4));

}

catch(Exception e){}

}

}
```

Output:

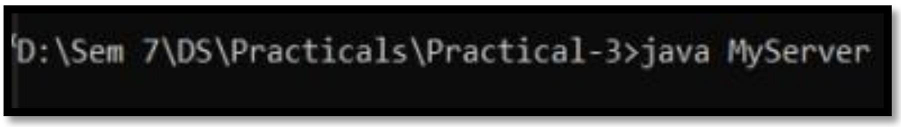


```
D:\Sem 7\DS\Practicals\Practical-3>java MyClient
Addition of 34 and 4 is: 38
```

MyServer.java

```
import java.rmi.*; import java.rmi.registry.*;

public class MyServer

{

public static void main(String args[])

{

try{

Adder stub=new AdderRemote();

Naming.rebind("rmi://localhost:5000/manan",stub);

}
catch(Exception e){System.out.println(e);}
}
}
```

Output:



```
D:\Sem 7\DS\Practicals\Practical-3>java MyServer
```

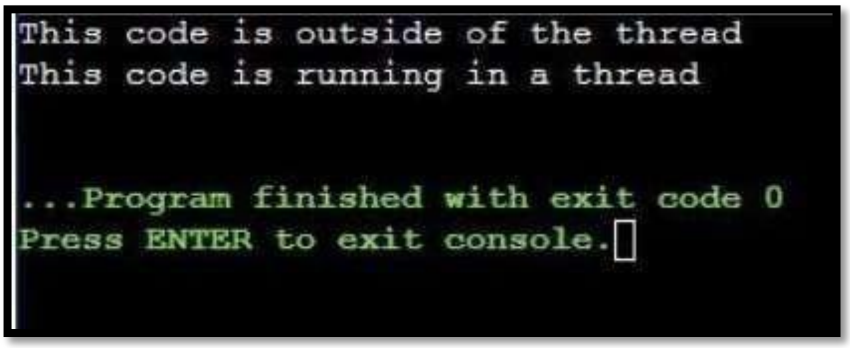
Practical: 4

AIM: Write the Programs for Thread Programming in JAVA.

Code:

```
public class Main extends Thread
{
public static void main(String[] args)
{
Main thread = new Main();
thread.start();
System.out.println("This code is outside of the thread");
}
public void run()
{
System.out.println("This code is running in a thread");
}
}
```

Output:



```
This code is outside of the thread
This code is running in a thread


...Program finished with exit code 0
Press ENTER to exit console.
```

Practical: 5

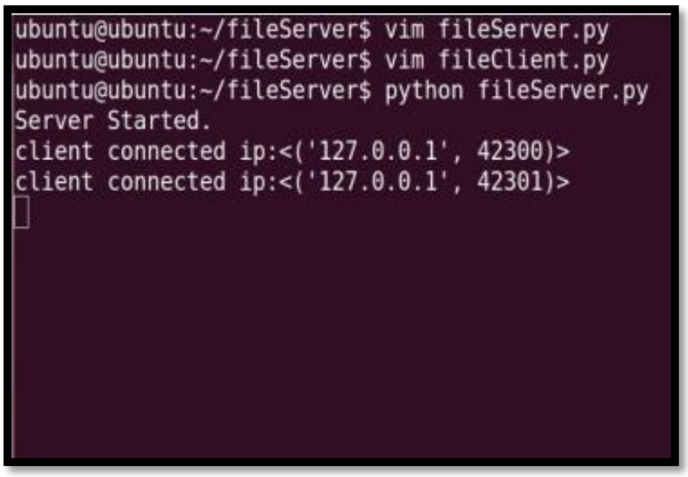
AIM: Implement Network File System (NFS).

Code:

File: FileServer.py

```
import socket
import threading
import os
def RetrFile(name, sock):
 filename = sock.recv(1024)
            if os.path.isfile(filename):
            sock.send("EXISTS " + str(os.path.getsize(filename)))
            userResponse = sock.recv(1024)
If userResponse[:2] == 'OK':
            with open(filename, 'rb') as f:
            bytesToSend = f.read(1024)
             sock.send(bytesToSend)
                        while bytesToSend != "":
            bytesToSend = f.read(1024)
            sock.send(bytesToSend)
            else:
            sock.send("ERR ")
            sock.close()
            def Main():
            host = '127.0.0.1'
            port = 5000
            s = socket.socket()
            s.bind((host,port))
            s.listen(5)
            print ("Server Started.")
            while True:
            c, addr = s.accept()
            print ("client connedted ip:<" + str(addr) + ">")
            t = threading.Thread(target=RetrFile, args=("RetrThread", c))
            t.start()
            s.close()
            if__name__== '_main_':
            Main()
```

Output:

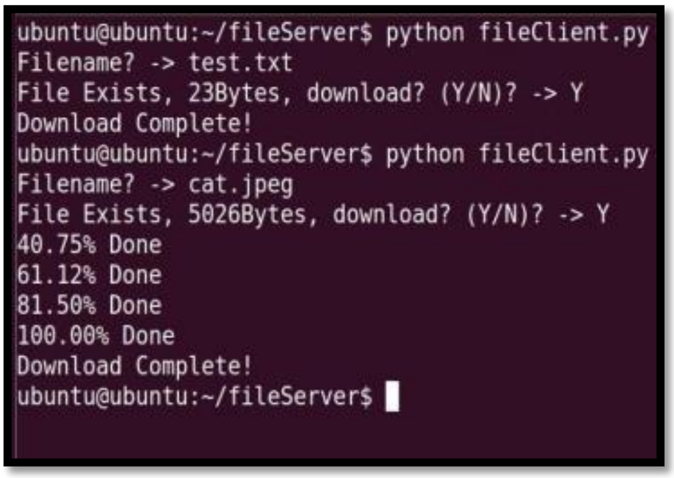


```
ubuntu@ubuntu:~/fileServer$ vim fileServer.py
ubuntu@ubuntu:~/fileServer$ vim fileClient.py
ubuntu@ubuntu:~/fileServer$ python fileServer.py
Server Started.
client connected ip:<('127.0.0.1', 42300)>
client connected ip:<('127.0.0.1', 42301)>
```

File: Fileclient.py

```
import socket
def Main():
             host = '127.0.0.1'
            port = 5000
            s = socket.socket()
             s.connect((host, port))
            filename = input("Filename? -> ")
            if filename != 'q':
             s.send(filename)
            data = s.recv(1024)
            if data[:6] == 'EXISTS':
            filesize = long(data[6:])
             message = input("File exists, " + str(filesize) +"Bytes, download? (Y/N)? -> ")
             if message == 'Y':
             s.send("OK")
            f = open('new_'+filename, 'wb')
            data = s.recv(1024)
            totalRecv = len(data)
             f.write(data)
            while totalRecv < filesize:
            data = s.recv(1024)
            totalRecv += len(data)
            f.write(data)
            print ("{0:.2f}".format((totalRecv/float(filesize))*100)+ "% Done")
            print ("Download Complete!")
            f.close()
            else:
            print ("File Does Not Exist!")
            s.close()
            if__name__== '_main_':
             Main()
```

Output:



```
ubuntu@ubuntu:~/fileServer$ python fileClient.py
Filename? -> test.txt
File Exists, 23Bytes, download? (Y/N)? -> Y
Download Complete!
ubuntu@ubuntu:~/fileServer$ python fileClient.py
Filename? -> cat.jpeg
File Exists, 5026Bytes, download? (Y/N)? -> Y
40.75% Done
61.12% Done
81.50% Done
100.00% Done
Download Complete!
ubuntu@ubuntu:~/fileServer$
```

Practical: 6

AIM: Creation of a BPEL (Business Process Execution Language) Module and a Composite Application.

What is BPEL?

The name says it all: Business Process Execution Language.

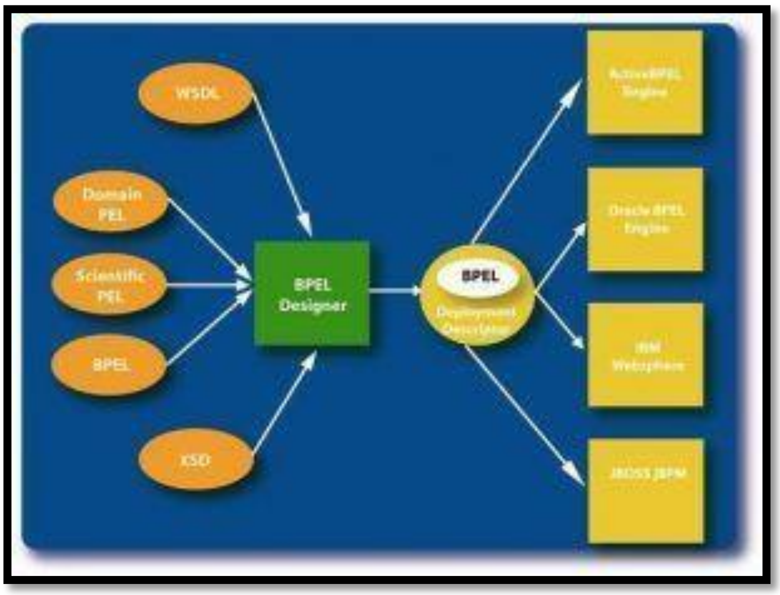
BPEL is a language that makes these business processes programmable and easy to implement. Being able to define your processes, orchestrate and choreograph processes, and integrate web services into the average business workflow makes integrating BPEL a jump forward for teams aiming to improve the automated aspect of their organization.

Components of BPEL:

The ease of integrating BPEL within your organization allows for enterprise-scale automation of services, including external process through Web services.

When originally designed, the team behind BPEL intend to fulfill 10 design goals:

- Define business processes through web services to help them interact with external entities
- Use an XML-based language to fulfill all business needs
- Build web services orchestration capabilities into the language as abstract and executable business processes
- Regimes that are both hierarchal and graph-like to reduce fragmentation
- Data manipulation and analysis functionality is built into the language
- Supports an identification mechanism
- Creation and termination of processes in the basic lifecycle mechanism
- Define a transaction model that includes techniques such as compensation actions and scoping
- Web services is the underlying model
- All functionality builds on and adds to the web services standard



Because of these 10 aims, BPEL (particularly the 2.0 version) was designed to be an orchestration language—not a choreography language like Xlang and the initial BPEL4WS creation.
The additional benefits that BPEL brings to a company through these 10 aims include:

- Large scale automation
- Orchestration of automated processes
- Easy to integrate web services

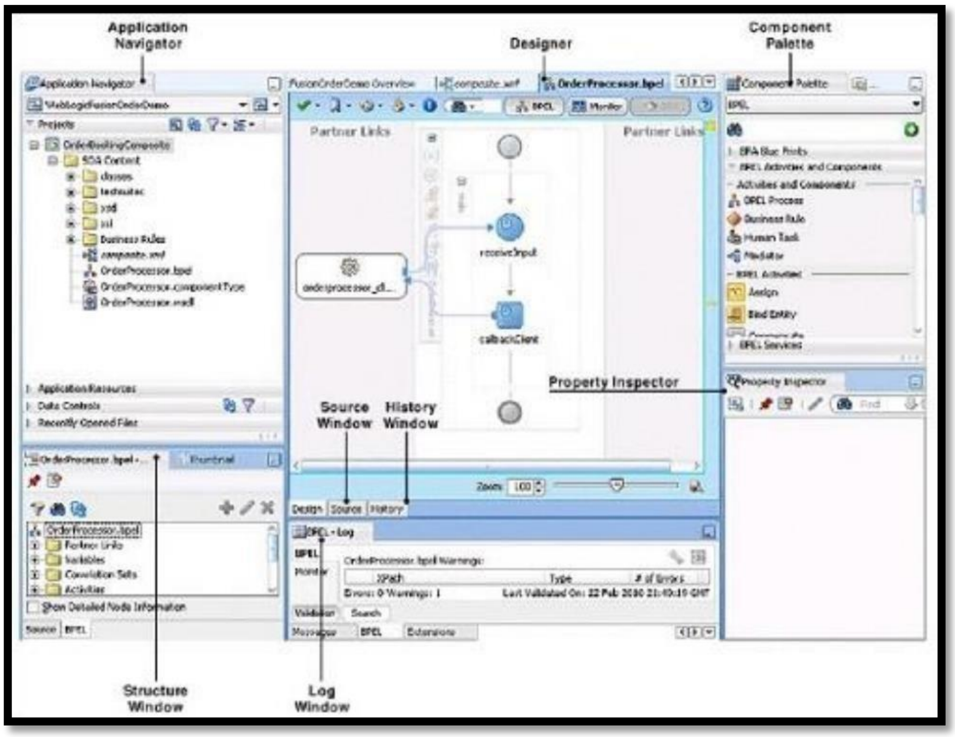
Adding a BPEL Process Service Component:

- Follow these steps to add a BPEL Process Service Component −
- From the Application Navigator, select File > New > Applications > SOA Application.
- This starts the Create SOA Application wizard.
- In the Application Name dialog, enter an application name in the Application Name field.
- In the Directory field, enter a directory path in which to create the SOA composite application and project.
- Click Next.
- In the Project Name dialog, enter a name in the Project Name field.
- Click Next.
- In the Project SOA Settings dialog, select Composite with the BPEL Process.
- Click Finish.

Files in the BPEL Composite:

- The BPEL composite contains the following files −
- **composite.xml** − This file describes the entire composite assembly of services, service
- components, references, and wires.
- **.bpel** − This file contains the set of activities added to the process.

- **.componentType** − This file describes the services and references for the BPEL
- process service component.
- **.wsdl** − This file defines the input and output messages for this BPEL process flow, the
- supported client interface and operations, and other features.



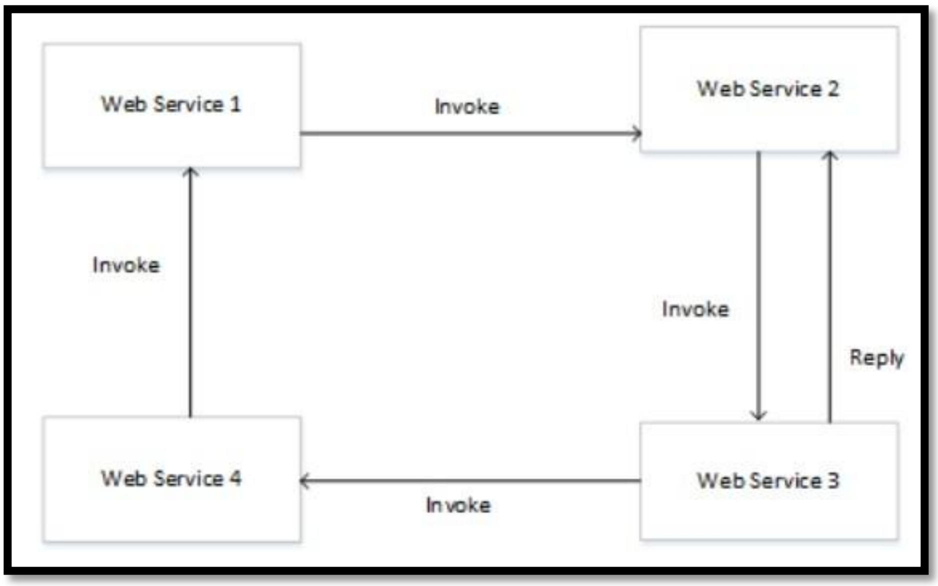
Choreography:

• Does not rely on a central coordinator.

• Each Web service involved in the choreography knows exactly when to execute its operations and with whom to interact.

• Each Web service involved in the choreography knows exactly when to execute its operations and with whom to interact.

• All participants in the choreography need to be aware of the business process, operations to execute, messages to exchange, and the timing of message exchanges.


Web Service 1
Invoke
Web Service 2
Invoke
Invoke
Reply
Web Service 4
Invoke
Web Service 3

Practical: 7

AIM: Implement CORBA file.

A simple CORBA implementation using Java:

1. Write the IDL file:

The IDL file defines the interface that will be used by the client and server for communicating and passing objects.
When the IDL file gets compiled, it will produce a number of files, known as the stub and skeleton:

- The stub is used by the client to communicate with the server
- The skeleton is used by the server to communicate with the client
- The stub and skeleton communicate with the ORB server to facilitate the remote procedure call

The module in the IDL file will correspond to the package and directory in which the Java code will be generated

Echo.idl

```
module EchoApp {

   interface Echo {

      string echoString();

   };

};
```

2. Generate the stub and skeleton code:

There is an idlj program that comes with the JDK for generating the stub and skeleton code in Java

```
idlj –fall sum.idl
```

The following files are generated by the idlj program:

- _EchoStub.java
- Echo.java
- EchoHelper.java
- EchoHolder.java
- EchoOperations.java
- EchoPOA.java

3. Write the server code:

The server program will inherit from the EchoPOA class that was generated as part of the idlj program.
The EchoPOA class implements the EchoOperations interface

- This interface contains the methods we defined in the Echo.idl file, but standardized to Java.

We create an EchoServer.java class that extends the abstract EchoPoa class and then implement the methods contained in it

We create a main method in Server.java to communicate with the object request broker (ORB), registering the server with the ORB so clients are able to find it.

4. Write the client code:

The client program Client.java needs to get a reference to the ORB then resolve the name of the server object it would like to invoke.

- This is ECHO-SERVER in this case

After getting an instance of a Server object from the server, it can invoke methods on it just like a method within its own JVM.

5. Compile the code:

1. Compile the stub and skeleton from the directory that contains the IDL file.

   Windows

   ```
   javac EchoApp\*.java
   ```

   Linux

   ```
   javac EchoApp/*.java
   ```

2. Generate a JAR file from the compiled stub and skeleton.

   Windows

   ```
   jar cvf echoapp.jar EchoApp\*.class
   ```

   Linux

   ```
   jar cvf echoapp.jar EchoApp/*.class
   ```

3. Compile the server and client classes

   Windows

   ```
   javac -classpath .;echoapp.jar Server.java EchoServer.java Client.java
   ```

Linux

```
javac -classpath .:echoapp.jar Server.java EchoServer.java Client.java
```

6. Running the application:

1. Start the ORB server

   ```
   orbd -ORBInitialPort 1050 -ORBInitialHost localhost
   ```

2. Start the server application

   ```
   java Server -ORBInitialPort 1050 -ORBInitialHost localhost
   ```

3. Start the client application

   ```
   java Client -ORBInitialPort 1050 -ORBInitialHost localhost
   ```

If everything was compiled correctly the output should be:

```
Hello World!!!!!!!
```

Code:

Counter.idl

```
interface Counter{
    readonly attribute long value;
    void inc();
    void dec();
};
```

CounterImpl.java

```
public class CounterImpl extends CounterPOA {
        private int count;
        public CounterImpl() {
                count =
                    0;
        }
        public void inc(){
                count++;
        }
        public void dec(){
                count --;
        }
        public int value(){
                return count;
        }
 }
```

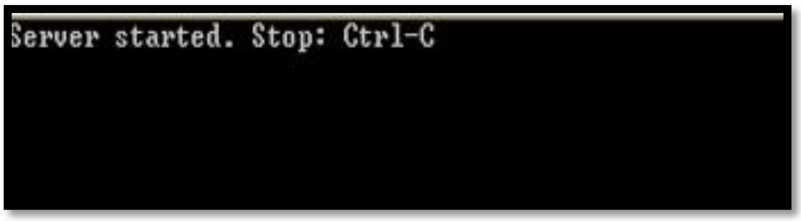
Server.java

```
import java.io.*;
import  java.util.Properties;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import static java.lang.System.*;

public class Server{
public static void main(String[] args){
        try
         {
                Properties props = getProperties();
                ORB orb = ORB.init(args, props);
                org.omg.CORBA.Object obj = null;
                POA rootPOA = null;
                try{
                        obj = orb.resolve_initial_references("RootPOA");
                        rootPOA = POAHelper.narrow(obj);
                }
                catch (org.omg.CORBA.ORBPackage.InvalidName e) { }
                CounterImpl c_impl = new CounterImpl();
                Counter c = c_impl._this(orb);

                try{
                FileOutputStream file = new FileOutputStream("Counter.ref");
                PrintWriter writer = new PrintWriter(file);
                String ref = orb.object_to_string(c);
                writer.println(ref);
                writer.flush();
                file.close();
                out.println("Server started." +  " Stop: Ctrl-C");
                }
                catch (IOException ex){
                        out.println("File error: " + ex.getMessage());
                        exit(2);
                }

                rootPOA.the_POAManager().activate();
                orb.run();
        }
        catch(Exception ex) {
                out.println("Exception: " + ex.getMessage());
                exit(1);
         }
}
}
```

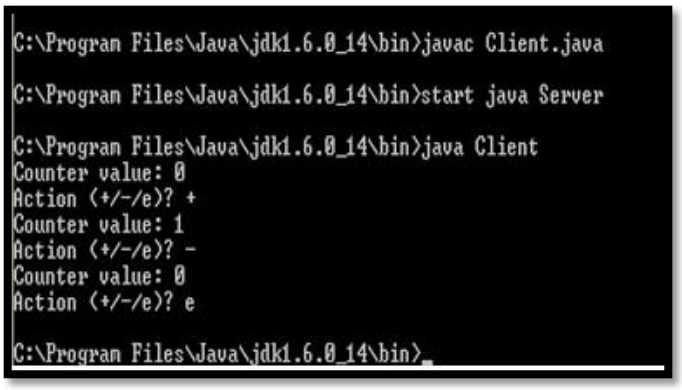
Client.java

```
import java.io.*;
import java.util.*;
import org.omg.CORBA.*;
import static java.lang.System.*;

public class Client{
        public static void main(String[] args) {
                try
                 {
                        Properties props = getProperties();
                        ORB orb = ORB.init(args, props);
                        String ref = null;
                        org.omg.CORBA.Object obj = null;
                        try {
                        Scanner reader =  new Scanner(new File("Counter.ref"));
                        ref = reader.nextLine();
                        }
                        catch (IOException ex) {
                                 out.println("File error: " + ex.getMessage());
                                exit(2
                                  );
                        }
                        obj = orb.string_to_object(ref);
                        if (obj == null) {
                                out.println("Invalid IOR");
                                exit(4);
                        }
                        Counter c = null;
                        try {
                                c = CounterHelper.narrow(obj);
                        }
                        catch (BAD_PARAM ex) {
                                out.println("Narrowing failed");
                                exit(3);
                        }

                        int inp = -1;
                        do {
                        out.print("Counter value: " + c.value() + "\nAction (+/-/e)? ");
                        out.flush();
                                do
                                 {
                                try
                                 {
                                        inp = in.read();
                                        }
                                        catch (IOException ioe) {
                                        }
                                        } while (inp != '+' && inp != '-' && inp != 'e');
                                    if (inp ==
                                        '+')
                                            c.inc(
                                              );
                                    else if (inp == '-
                                            ')
                                            c.dec(
                                              );
                        } while (inp != 'e');
                }
                catch (Exception ex) {
                        out.println("Exception: " + ex.getMessage());
                        exit(1);
                }
        }
}
```

Output:



```
Server started. Stop: Ctrl-C
```



```
C:\Program Files\Java\jdk1.6.0_14\bin>javac Client.java

C:\Program Files\Java\jdk1.6.0_14\bin>start java Server

C:\Program Files\Java\jdk1.6.0_14\bin>java Client
Counter value: 0
Action (+/-/e)? +
Counter value: 1
Action (+/-/e)? -
Counter value: 0
Action (+/-/e)? e

C:\Program Files\Java\jdk1.6.0_14\bin>_
```

Practical: 8

AIM: Study of Web Service Programming.

What are Web Services?

The Internet is the worldwide connectivity of hundreds of thousands of computers of various types that belong to multiple networks. On the World Wide Web, a web service is a standardized method for propagating messages between client and server applications. A web service is a software module that is intended to carry out a specific set of functions. Web services in cloud computing can be found and invoked over the network.

The web service would be able to deliver functionality to the client that invoked the web service.

A web service is a set of open protocols and standards that allow data to be exchanged between different applications or systems. Web services can be used by software programs written in a variety of programming languages and running on a variety of platforms to exchange data via computer networks such as the Internet in a similar way to inter-process communication on a single computer.

Any software, application, or cloud technology that uses standardized web protocols (HTTP or HTTPS) to connect, interoperate, and exchange data messages – commonly XML (Extensible Markup Language) – across the internet is considered a web service.

Web services have the advantage of allowing programs developed in different languages to connect with one another by exchanging data over a web service between clients and servers. A client invokes a web service by submitting an XML request, which the service responds with an XML response**.**

Functions of Web Services:

- It’s possible to access it via the internet or intranet networks.
- XML messaging protocol that is standardized.
- Operating system or programming language independent.
- Using the XML standard, it is self-describing.
- A simple location approach can be used to locate it.

Components of Web Service:

XML and HTTP is the most fundamental web services platform. The following components are used by all typical web services:

SOAP (Simple Object Access Protocol):

SOAP stands for –Simple Object Access Protocol.‖ It is a transport-independent messaging protocol. SOAP is built on sending XML data in the form of SOAP Messages. A document known as an XML document is attached to each message. Only the structure of the XML document, not the content, follows a pattern. The best thing about Web services and SOAP is that everything is sent through HTTP, the standard web protocol.

A root element known as the element is required in every SOAP document. In an XML document, the root element is the first element. The –envelope‖ is separated into two halves. The header comes first, followed by the body. The routing data, or information that directs the XML document to which client it should be sent to, is contained in the header. The real message will be in the body.

UDDI (Universal Description, Discovery, and Integration):

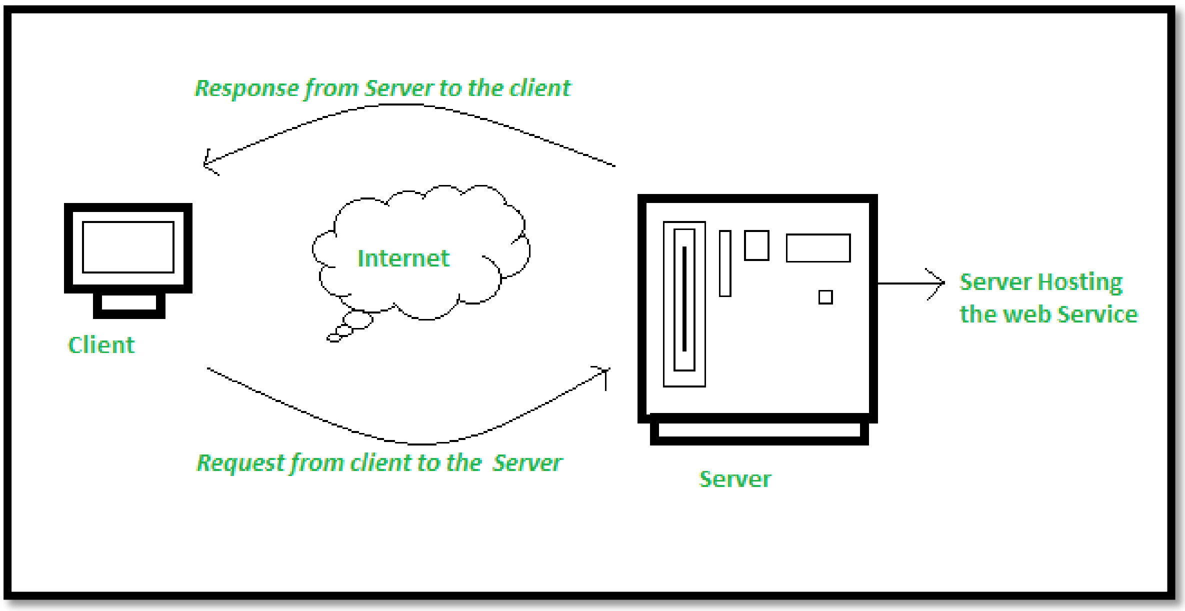
UDDI is a standard for specifying, publishing and discovering a service provider's online services. It provides a specification that aids in the hosting of data via web services. UDDI provides a repository where WSDL files can be hosted so that a client application can discover a WSDL file to learn about the various actions that a web service offers. As a result, the client application will have full access to the UDDI, which serves as a database for all WSDL files.
The UDDI registry will hold the required information for the online service, just like a telephone directory has the name, address, and phone number of a certain individual. So that a client application may figure out where it is.

WSDL (Web Services Description Language):

If a web service can't be found, it can't be used. The client invoking the web service should be aware of the location of the web service. Second, the client application must understand what the web service does in order to invoke the correct web service. The WSDL, or Web services description language, is used to accomplish this. The WSDL file is another XML-based file that explains what the web service does to the client application. The client application will be able to understand where the web service is located and how to use it by using the WSDL document.

How Does Web Service Work?

The diagram depicts a very simplified version of how a web service would function. The client would use requests to send a sequence of web service calls to a server that would host the actual web service.


Response from Server to the client
Internet
Client
Server Hosting the web Service
Request from client to the Server
Server

Remote procedure calls are what are used to make these requests. Calls to methods hosted by the relevant web service are known as Remote Procedure Calls (RPC). Example: Flipkart offers a web service that displays prices for items offered on Flipkart.com. The front end or presentation layer can be written in .Net or Java, but the web service can be communicated using either programming language.

The data that is exchanged between the client and the server, which is XML, is the most important part of a web service design. XML (Extensible markup language) is a simple intermediate language that is understood by various programming languages. It is a counterpart to HTML. As a result, when programs communicate with one another, they do so using XML. This creates a common platform for applications written in different programming languages to communicate with one another.

For transmitting XML data between applications, web services employ SOAP (Simple Object Access Protocol). The data is sent using standard HTTP. A SOAP message is data that is sent from the web service to the application. An XML document is all that is contained in a SOAP message. The client application that calls the web service can be created in any programming language because the content is written in XML.

Features/Characteristics Of Web Service:

Web services have the following features:

(a) XML Based: The information representation and record transportation layers of a web service employ XML. There is no need for networking, operating system, or platform binding when using XML. At the middle level, web offering-based applications are highly interoperable.

(b) Loosely Coupled: A customer of an internet service provider isn't necessarily directly linked to that service provider. The user interface for a web service provider can change over time without impacting the user's ability to interact with the service provider. A strongly coupled system means that the patron's and server's decisions are inextricably linked, indicating that if one interface changes, the other should be updated as well. A loosely connected architecture makes software systems more manageable and allows for easier integration between different structures.

(c) Capability to be Synchronous or Asynchronous: Synchronicity refers to the client's connection to the function's execution. The client is blocked and the client has to wait for the service to complete its operation, before continuing in synchronous invocations. Asynchronous operations allow a client to invoke a task and then continue with other tasks.
Asynchronous clients get their results later, but synchronous clients get their effect immediately when the service is completed. The ability to enable loosely linked systems requires asynchronous capabilities.

(d) Coarse-Grained: Object-oriented systems, such as Java, make their services available through individual methods. At the corporate level, a character technique is far too fine an operation to be useful. Building a Java application from the ground, necessitates the development of several fine-grained strategies, which are then combined into a rough-grained provider that is consumed by either a buyer or a service. Corporations should be coarse-grained, as should the interfaces they expose. Web services generation is an easy approach to define coarse-grained services that have access to enough commercial enterprise logic.

(e) Supports Remote Procedural Call: Consumers can use an XML-based protocol to call procedures, functions, and methods on remote objects utilizing web services. A web service must support the input and output framework exposed by remote systems. Enterprise-wide component development Over the last few years, JavaBeans (EJBs) and.NET Components have become more prevalent in architectural and enterprise deployments. A number of RPC techniques are used to allocate and access both technologies.
A web function can support RPC by offering its own services, similar to those of a traditional role, or by translating incoming invocations into an EJB or.NET component invocation.

(f) Supports Document Exchanges: One of XML's most appealing features is its simple approach to communicating with data and complex entities. These records can be as simple as talking to a current address or as complex as talking to an entire book or a Request for Quotation. Web administrations facilitate the simple exchange of archives, which aids incorporate reconciliation.
The web benefit design can be seen in two ways: **(i)** The first step is to examine each web benefit on-screen character in detail. **(ii)** The second is to take a look at the rapidly growing web benefit convention stack.

Advantages Of Web Service:

Using web services has the following advantages:

(a) Business Functions can be exposed over the Internet: A web service is a controlled code component that delivers functionality to client applications or end-users. This capability can be accessed over the HTTP protocol, which means it can be accessed from anywhere on the internet. Because all apps are now accessible via the internet, Web services have become increasingly valuable. Because all apps are now accessible via the internet, Web services have become increasingly valuable. That is to say, the web service can be located anywhere on the internet and provide the required functionality.

(b) Interoperability: Web administrations allow diverse apps to communicate with one another and exchange information and services. Different apps can also make use of web services. A .NET application, for example, can communicate with Java web administrations and vice versa. To make the application stage and innovation self-contained, web administrations are used.

(c) Communication with Low Cost: Because web services employ the SOAP over HTTP protocol, you can use your existing low-cost internet connection to implement them. Web services can be developed using additional dependable transport protocols, such as FTP, in addition to SOAP over HTTP.

(d) A Standard Protocol that Everyone Understands: Web services communicate via a defined industry protocol. In the web services protocol stack, all four layers (Service Transport, XML Messaging, Service Description, and Service Discovery) use well-defined protocols.

(e) Reusability: A single web service can be used simultaneously by several client applications.

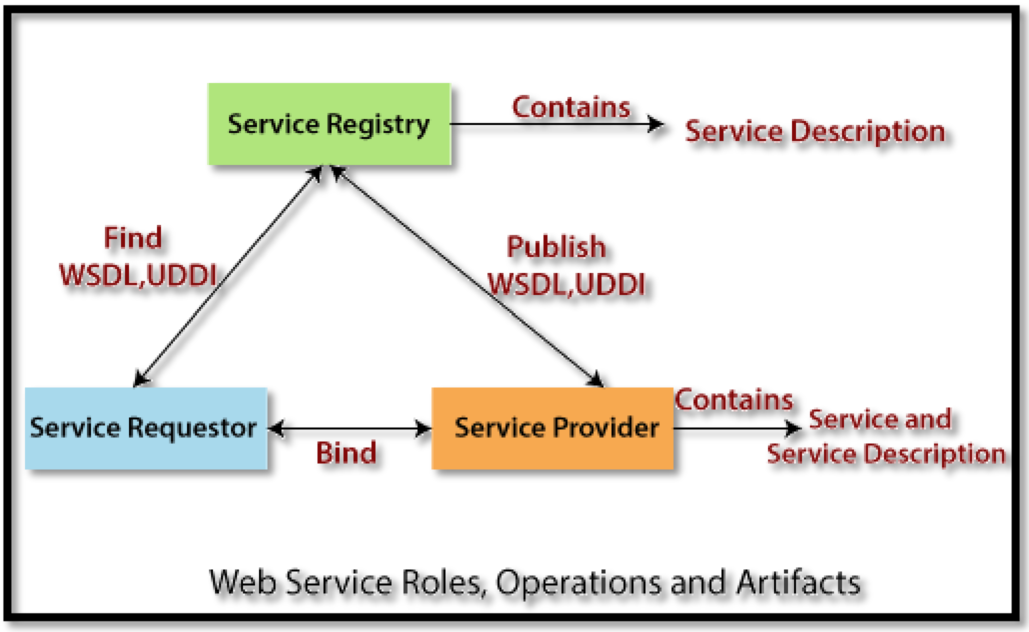
Architecture of Web Services:

The Web Services architecture describes how to instantiate the elements and implement the operations in an interoperable manner.

The architecture of web service interacts among three roles: **service provider, service requester,** and **service registry**. The interaction involves the three operations: **publish, find,** and **bind**. These operations and roles act upon the **web services artifacts**. The web service artifacts are the web service software module and its description.

The service provider hosts a network-associable module (web service). It defines a service description for the web service and publishes it to a service requestor or service registry. These service requestor uses a find operation to retrieve the service description locally or from the service registry. It uses the service description to bind with the service provider and invoke with the web service implementation.

The following figure illustrates the operations, roles, and their interaction.



Roles in a Web Service Architecture:

There are three roles in web service architecture:

- Service Provider
- Service Requestor
- Service Registry

Service Provider

From an architectural perspective, it is the platform that hosts the services.

Service Requestor

Service requestor is the application that is looking for and invoking or initiating an interaction with a service. The browser plays the requester role, driven by a consumer or a program without a user interface.

Service Registry

Service requestors find service and obtain binding information for services during development.

Operations in a Web Service Architecture:

Three behaviors that take place in the microservices:

- Publication of service descriptions **(Publish)**
- Finding of services descriptions **(Find)**
- Invoking of service based on service descriptions **(Bind)**

Publish: In the publish operation, a service description must be published so that a service requester can find the service.

Find: In the find operation, the service requestor retrieves the service description directly. It can be involved in two different lifecycle phases for the service requestor:

- At design, time to retrieve the service's interface description for program development.
- And, at the runtime to retrieve the service's binding and location description for invocation.

Bind: In the bind operation, the service requestor invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact, and invoke the service.

Practical: 9

AIM: Study of open source key management tool.

Theory:

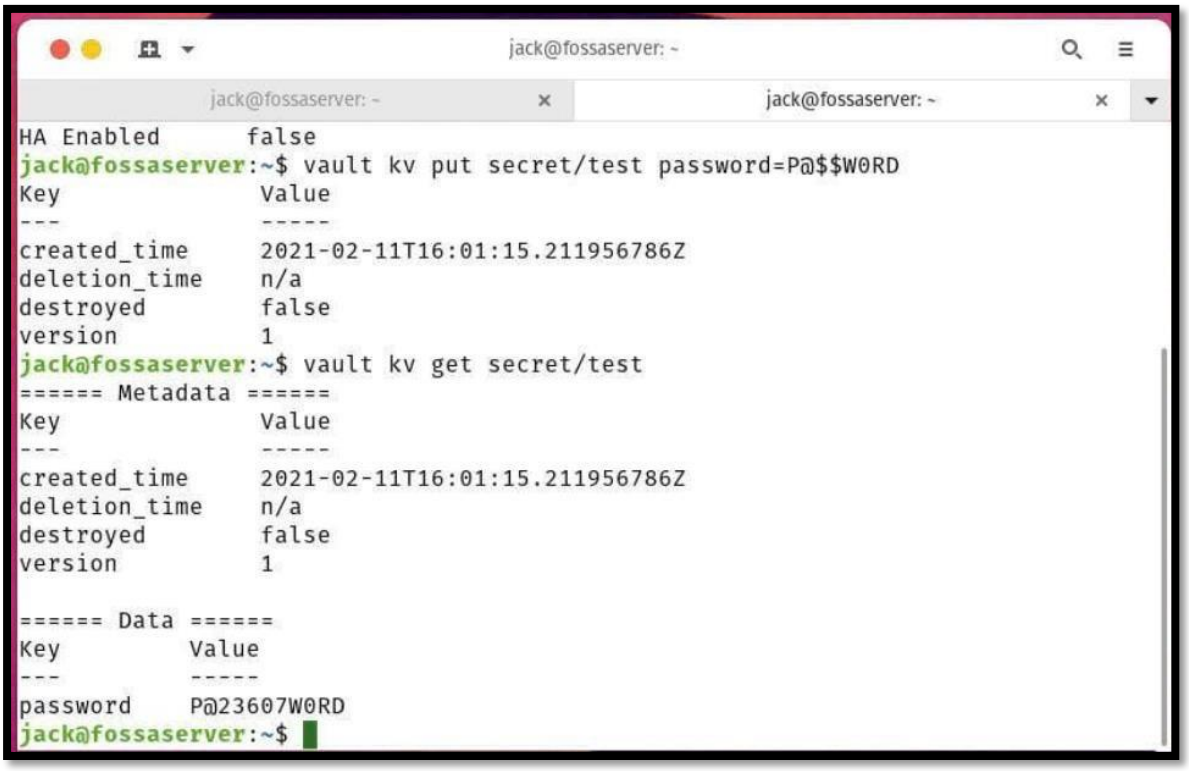
If you use SSH or services that require encryption keys, it can be challenging to safely store that data to allow you secure access to your accounts. Here are some services to help you keep track of them. If you use SSH or services that require encryption keys, it can be challenging to safely store that data to allow you secure access to your accounts. Here are some services to help you keep track of them.

Store those keys haphazardly and they could fall into the wrong hands. Or, you simply might lose track of what key goes to what service (at which point, you might as well have lost the key). What if you're a developer and you need some sort of vault to hold encryption key secrets that can then be linked to deployed services? What do you do?

You might consider an encryption key manager. These are different from password managers, because, in some cases, they actually function in the background to interact with various applications and services that depend on those keys. Of course, if you only need the means to securely store those keys so that you can manually retrieve them later, you could opt to make use of a simple password manager.

Tools and services to see which one might be the perfect match for your needs:

HashiCorp Vault:

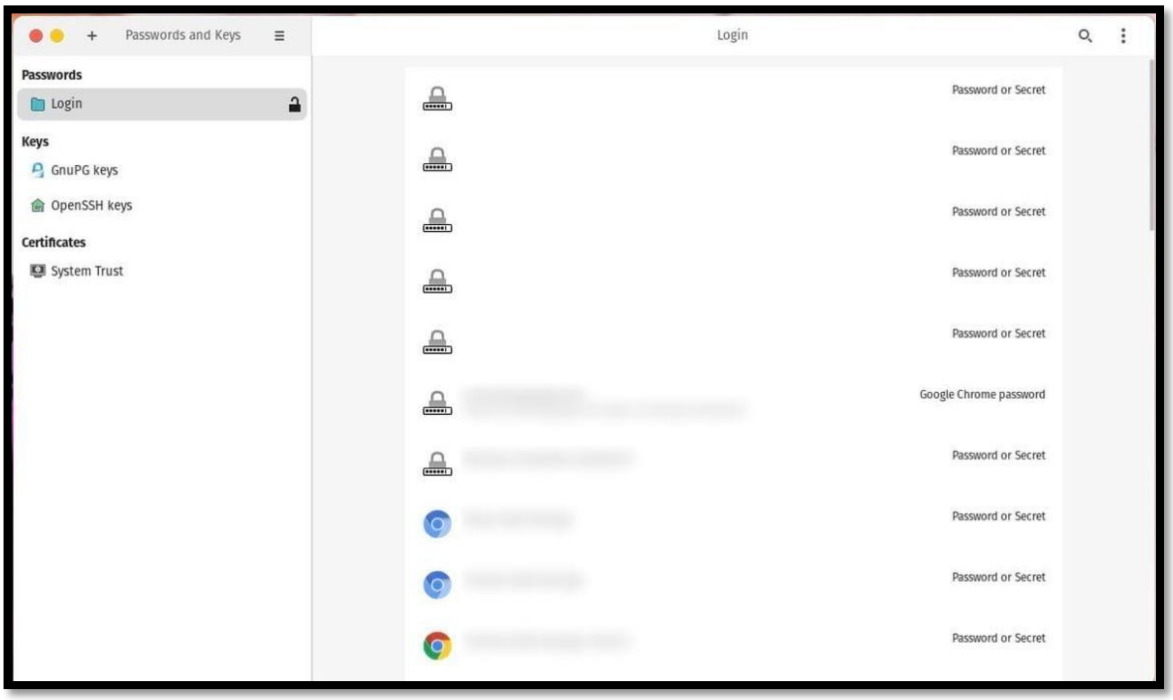


```
HA Enabled      false
jack@fossaserver:~$ vault kv put secret/test password=P@$$W0RD
Key              Value
---              -----
created_time     2021-02-11T16:01:15.211956786Z
deletion_time    n/a
destroyed        false
version          1
jack@fossaserver:~$ vault kv get secret/test
====== Metadata ======
Key              Value
---              -----
created_time     2021-02-11T16:01:15.211956786Z
deletion_time    n/a
destroyed        false
version          1

====== Data ======
Key         Value
---         -----
password    P@23607W0RD
jack@fossaserver:~$
```

HashiCorp Vault is a powerful tool for storing credentials, passwords, and various types of secrets (including encryption keys) that you can then safely use in your container deployments. If you're serious about the security of your containers, HashiCorp Vault should most certainly be on your radar. With HashiCorp Vault you can create and secure access tokens, passwords, certificates and encryption keys in such a way as to help you find the necessary balance between locked-down security and useability.

With HashiCorp Vault your developers will also save time because they won't have to struggle to find a reliable way to manage the secrets they use in their deployments and connecting services to third-party APIs. HashiCorp Vault helps you increase security across clouds and apps, across your entire IT landscape, with hundreds of integrations. With the ability to generate 10,000+ unique tokens daily, your teams can also use HashiCorp Vault to make automation a reality. HashiCorp Vault can be used for free (with the open-source, self-managed version), or you can opt for the managed Cloud plan (starting at 3 cents an hour) or the Enterprise plan (contact sales for information).

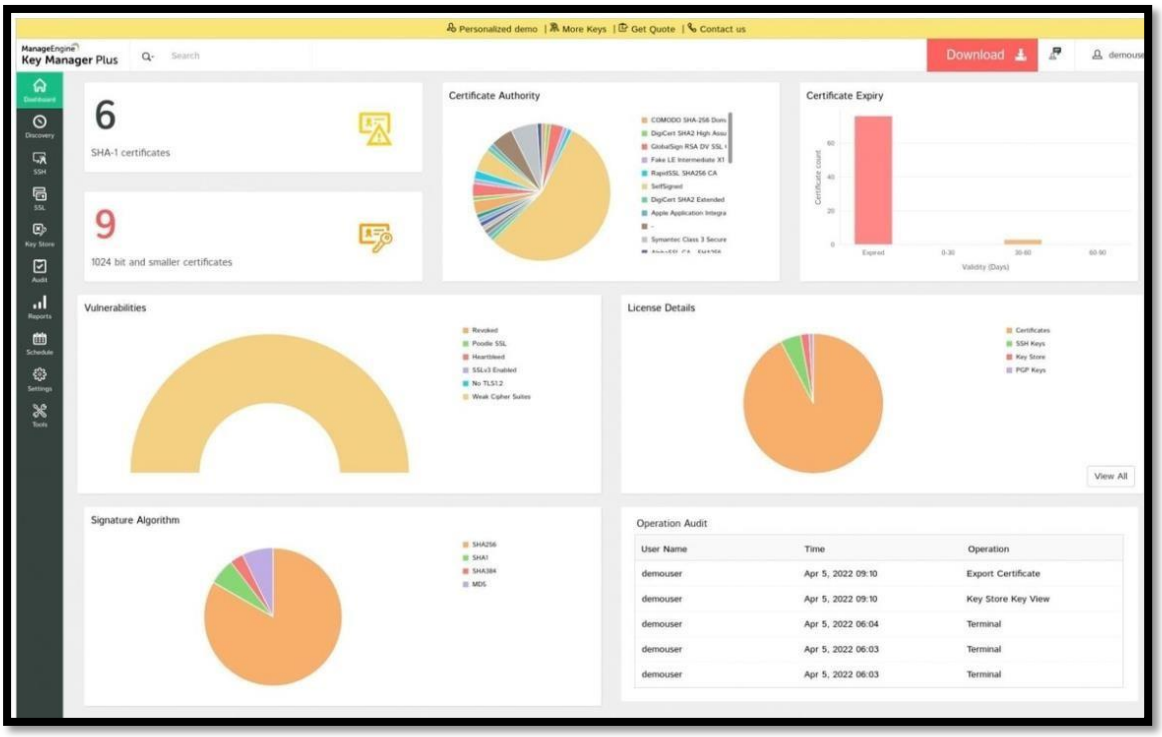
Seahorse:



Seahorse is an open-source tool, found on many a Linux Distribution, that makes creating, storing and managing encryption keys as user-friendly as it comes. Seahorse can work with SSH keys, GPG keys, passwords, and certificates … all from within a GUI that makes every step of the process simple. Store multiple keys (of each type), sign them, and even sync your keys with remote keyservers.

The one caveat to using Seahorse is that you have to be careful to ensure the keyring is locked when the tool is not in use (otherwise anyone can view your stored passwords). Seahorse also allows you to import keys from a file and export keys to a file. Seahorse is free to use and is found pre-installed on many Linux distributions. Seahorse is not available for either macOS or Windows.

ManageEngine Key Manager Plus:

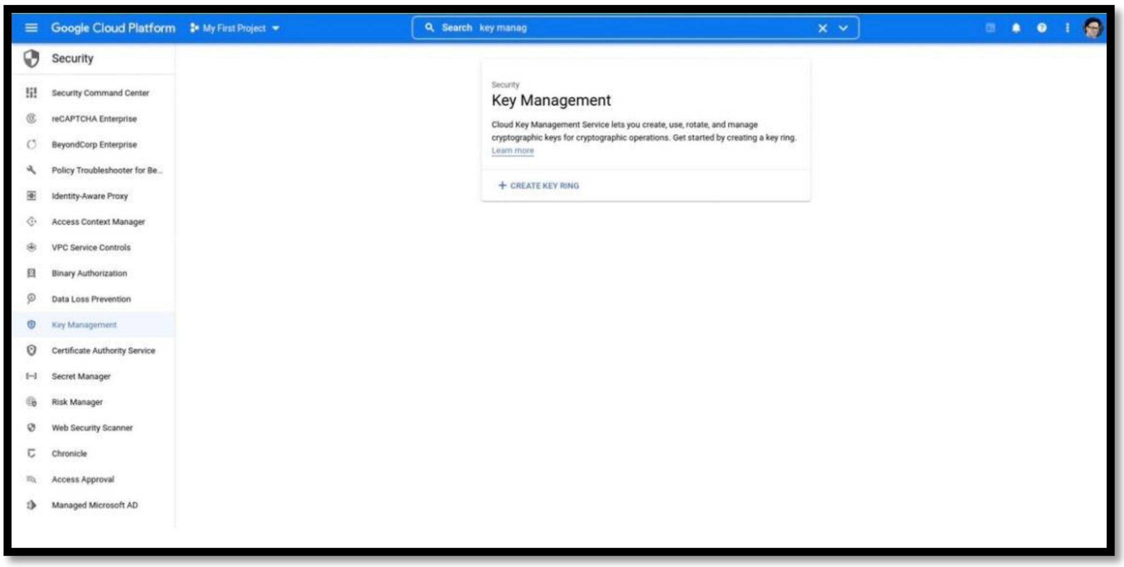


User Name	Time	Operation
demouser	Apr 5, 2022 09:10	Export Certificate
demouser	Apr 5, 2022 09:10	Key Store Key View
demouser	Apr 5, 2022 06:04	Terminal
demouser	Apr 5, 2022 06:03	Terminal
demouser	Apr 5, 2022 06:03	Terminal

If you’re looking for a web-based SSH and SSL certificate management solution, ManageEngine Key Manager Plus could very well solve this oft-convoluted problem. This platform makes it easy to consolidate, control, manage, monitor and audit your SSH keys and SSL certificates. If your business depends on a large number of SSL keys, across an entire IT landscape of servers, you owe it to your administrators to empower them with the tools to make the management of those keys effortless.

ManageEngine Key Manager Plus can be installed on a local server or you can opt for a hosted plan. Either way, you’ll get real-time dashboards to keep tabs on your keys, reports, schedules and even auditing tools. ManageEngine Key Manager Plus can be used for free as a trial, but you’ll soon have to pay up for a license, so you’ll have to contact the company to receive a quote.

Google Cloud Key Management:

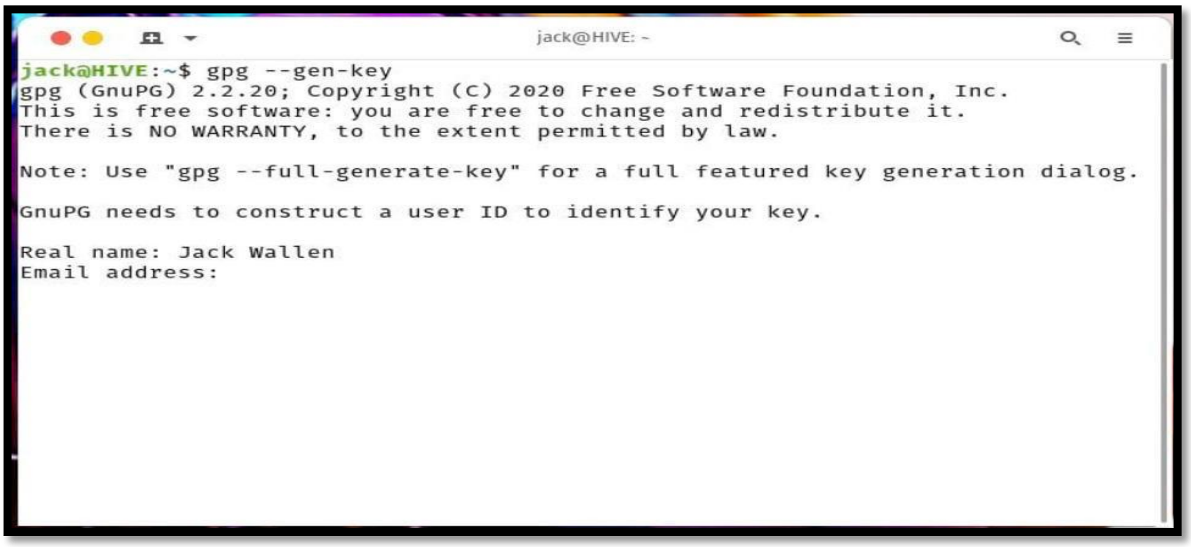


Cloud Key Management Service lets you create, use, rotate, and manage cryptographic keys for cryptographic operations. Get started by creating a key ring.

With Google Cloud Key Management you can enjoy scalable, centralized, cloud key management that can deliver compliance and privacy, and help bolster the security of your company. This service allows you to use Hardware Security Module (HSMs), and approve/deny any request for your encryption keys based on-premises justifications.

With Google Cloud Key Management you can also use your own managed keys to control the encryption of data across all Google Cloud products. The Google platform allows you to generate, use, rotate and destroy AES256, RSA 2048, RSA 3072, RSA 4096, EC P256, and EC P384 cryptographic keys, so it could easily serve the majority of your encryption key management. The price for Google Cloud Key Management is $3/active key.

GnuPG:



```
jack@HIVE:~$ gpg --gen-key
gpg (GnuPG) 2.2.20; Copyright (C) 2020 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: Jack Wallen
Email address:
```

If you’re looking for a local, command-line only tool to manage your encryption keys, GnuPG is the de facto standard. With this tool, you can manage key pairs with ease (adding, signing, deleting, revoking and editing). GnuPG is a free implementation of the OpenPGP standard (as defined by RFC4880) and can work with files and even integrates into many email clients for the encryption of your communications.

GnuPG comes pre-installed on most Linux distributions and is also available for macOS and Windows (via Gpg4win). GnuPG has been around since 1997, so its reputation for being one of the more trusted implementations of PGP is well earned.

Practical: 10

AIM: To study of apache web server.

Theory:

Apache HTTP Server is a free and open-source web server that delivers web content through the internet. It is commonly referred to as Apache and after development, it quickly became the most popular HTTP client on the web.

Apache is just one component that is needed in a web application stack to deliver web content. One of the most common web application stacks involves LAMP, or Linux, Apache, MySQL, and PHP.

Apache is considered open source software, which means the original source code is freely available for viewing and collaboration. Being open source has made Apache very popular with developers who have built and configured their own modules to apply specific functionality and improve on its core features. Apache has been around since 1995 and is responsible as a core technology that helped spur the initial growth of the internet in its infancy.

Features of Apache Web Server:

- Handling of static files
- Loadable dynamic modules
- Auto-indexing
- .htaccess
- Compatible with IPv6
- Supports HTTP/2
- FTP connections
- Gzip compression and decompression
- Bandwidth throttling
- Perl, PHP, Lua scripts
- Load balancing
- Session tracking
- URL rewriting
- Geolocation based on IP address

- Apache functions as a way to communicate over networks from client to server using the TCP/IP protocol. Apache can be used for a wide variety of protocols, but the most common is HTTP/S. HTTP/S or Hyper Text Transfer Protocol (S stands for Secure) is one of the main protocols on the web, and the one protocol Apache is most known for.

- HTTP/S is used to define how messages are formatted and transmitted across the web, with instructions for browsers and servers on how to respond .to various requests and commands. Hypertext Transfer Protocol Secure is usually through port 443 with the unsecured protocol being through port 80.

Alternatives for Apache HTTP Server:

While Apache web servers are very popular, they're not the only web servers on the market. Below are a number of alternatives for Apache HTTP servers.

- Nginx
- Apache Tomcat
- Node.js
- Lighttpd
- Cherokee
- Microsoft IIS
- Appweb
- Hiawatha