



**SHRI G.P.M. DEGREE COLLEGE OF  
SCIENCE & COMM**



**SHRI G.P.M. DEGREE COLLEGE OF  
SCIENCE & COMMERCE.**  
(COMMITTED TO EXCELLENCE IN EDUCATION)

## CERTIFICATE

This is to certify Mr/Ms \_\_\_\_\_

In the subject of \_\_\_\_\_ As prescribed by the  
UNIVERSITY OF MUMBAI under my supervision during the Academic year 2024-2025

\_\_\_\_\_

Prof.Incharge

\_\_\_\_\_

Principal

\_\_\_\_\_

External Examiner

\_\_\_\_\_

Course Co-ordinator

Professor Name : Prof. Vivek Maurya	Class : TY.BSC-CS Semester : Sem VI (2024-2025)
Course code : USCSP602	Subject : Cloud Computing and web services

Sr. No.	Date	Index	Page No.	Sign
1		Define a simple services like Converting Rs into Doller and Call it from different platform like JAVA and .NET.		
2		Create a simple SOAP services.		
3		Create a simple REST services.		
4		Develop application to consume Google's search / Google's Map RESTful Web service.		
5		Installation and Configuration of virtualization using KVM.		
6		Develop application to download image / video from server or upload image / video to server using MTOM techniques		
7		Implement FOSS-Cloud Functionality VSI(Virtual Server Infrastructure) Infrastructure as a service (IaaS), Storage.		
8		Implement FOSS-Cloud Functionality - VSI Platform as a Service (PaaS),		
9		Using AWS Flow Framework develop application that includes a simple workflow. Workflow calls an activity to print hello world to the console. It must define the basic usage of AWS Flow Framework, including defining contracts, implementation activities and workdlow coordination logic and worker programs to host them.		

10		Implementation of Openstack with user and private network creation		
----	--	--	--	--



## Practical 1

### Define a simple Services like Converting Rs into Doller and Call it from different Platform like JAVA and .NET.

**Aim:** To create a simple service for converting Indian Rupees (INR) into US Dollars (USD), you can develop a RESTful API using Python and Flask. The API will accept an amount in INR via a GET request and return the equivalent value in USD based on a fixed conversion rate. For example, 1 INR = 0.012 USD. Once the service is up and running, you can consume this API from different platforms such as Java and .NET. In Java, you can use `URLConnection` or a library like `OkHttp` to send HTTP requests to the Flask service and handle the JSON response. Similarly, in .NET, `HttpClient` can be used to send a GET request to the API and parse the JSON response. Both Java and .NET applications can retrieve and display the converted value in USD by simply invoking the Flask service with the desired INR amount as a query parameter. This approach demonstrates how to expose a simple REST API for currency conversion and how to consume it across multiple platforms, ensuring cross-platform communication.

#### Tools & Technologies used:

1. **Python** (Flask) – To build the API service.
2. **Java** – For consuming the API via HTTP requests.
3. **.NET (C#)** – For consuming the API via HTTP requests.
4. **Postman** (Optional) – For testing the API endpoints manually.

#### Theory:

##### 1. What is a REST API?

A REST (Representational State Transfer) API is an architectural style for designing networked applications. It uses HTTP requests to perform CRUD (Create, Read, Update, Delete) operations on resources, typically represented as JSON.

##### 2. Why use REST APIs for Interoperability?

REST APIs are language agnostic, meaning that they can be consumed by any application that can send HTTP requests and parse JSON responses. This allows communication between different platforms (Java, .NET, Python, etc.) regardless of their underlying programming languages.

##### 3. Currency Conversion API:

The API will accept an amount in INR and convert it to USD using a fixed conversion rate (e.g., 1 INR = 0.012 USD). This conversion rate can be updated in a real-world scenario by integrating with a live currency conversion service.



## SHRI G.P.M. DEGREE COLLEGE

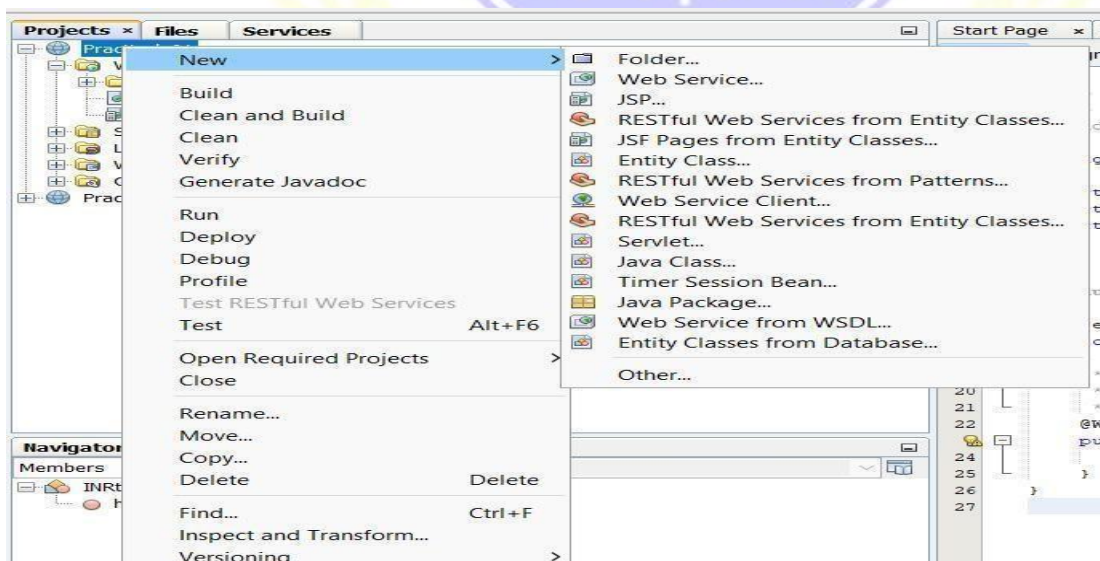
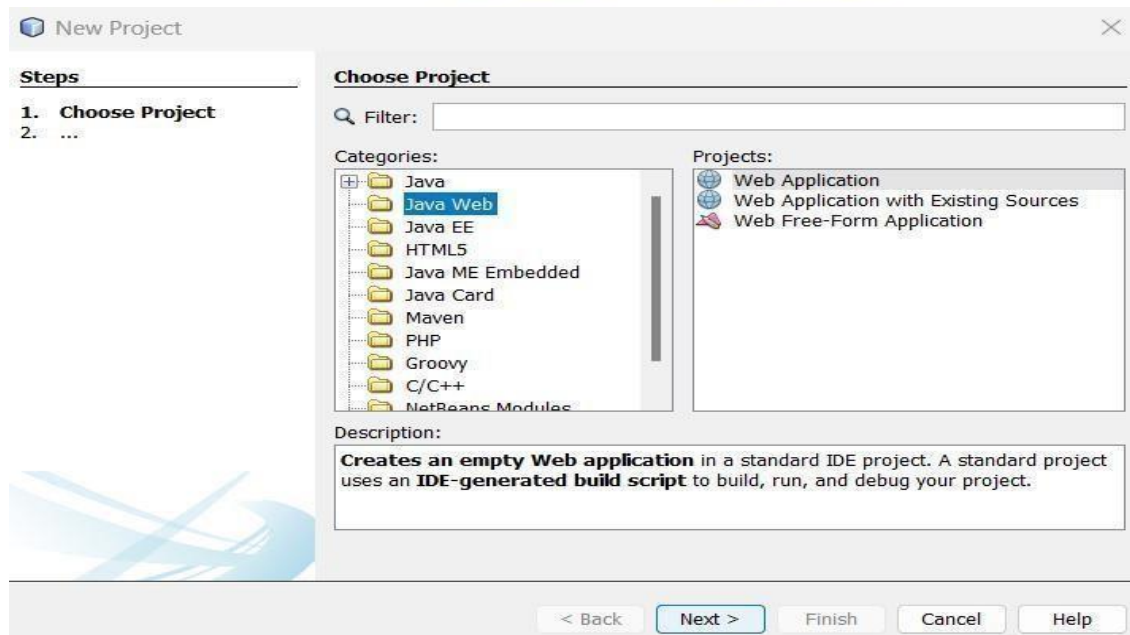
Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Code (Steps):

**Step 1:** Start NetBeans8.0.2

**Step 2:** Go to file ->new project ->java web(select) ->web application





## SHRI G.P.M. DEGREE COLLEGE

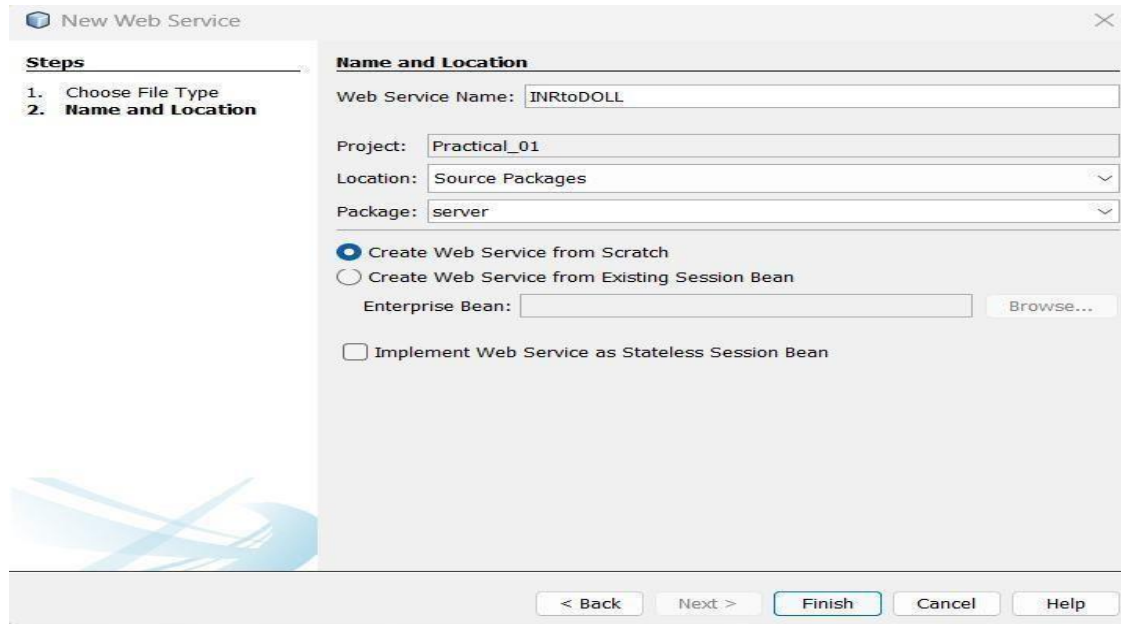
Department of Computer

Vision.. Innovation.. Solution.. Presentation

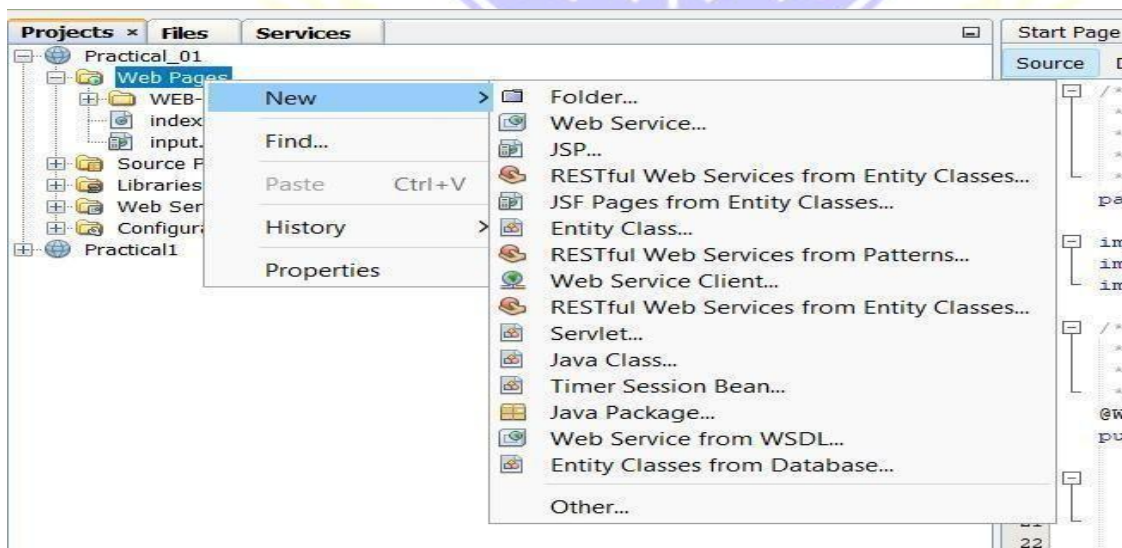
**Step 3:** After that Next and give the project name and finish.

**Step 4:** Right click on the project name page->new->webservic

Give the **web service name** and **package name** -> finish



**Step 5:** Right click on the web pages ->new ->jsp



Give the file name and finish it.





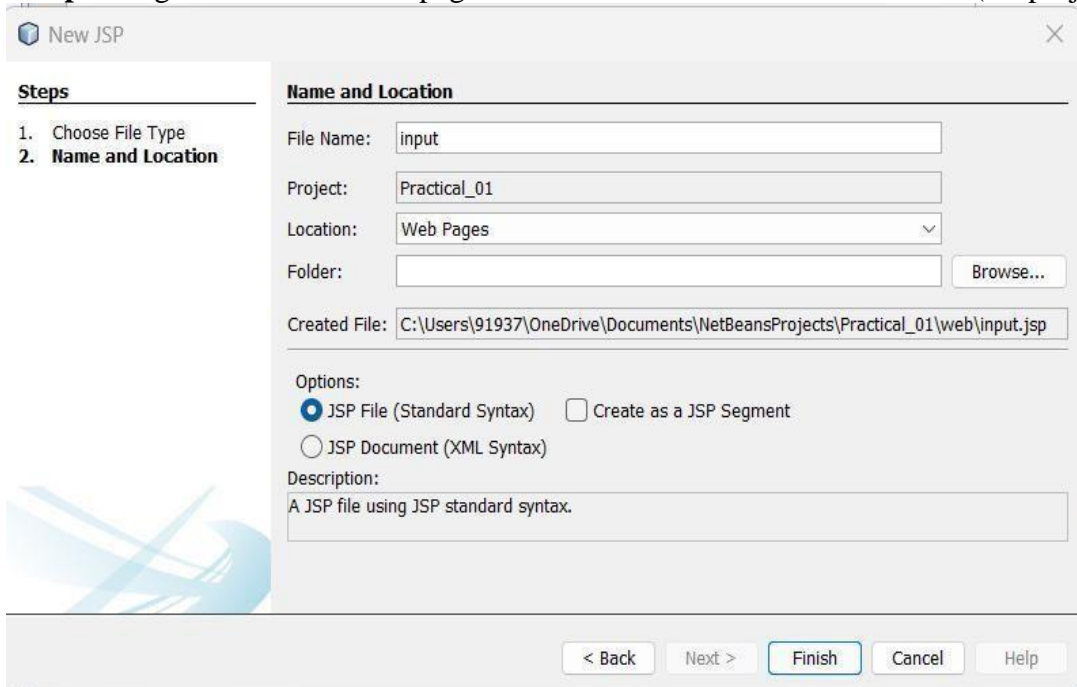
## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

**Step 6:** Repeat step 5 and create output.jsp.

**Step 7:** Right click on the web page ->new -> web service client->browse (for project name)



**New JSP**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

File Name:

Project:

Location:

Folder:

Created File: C:\Users\91937\OneDrive\Documents\NetBeansProjects\Practical\_01\web\input.jsp

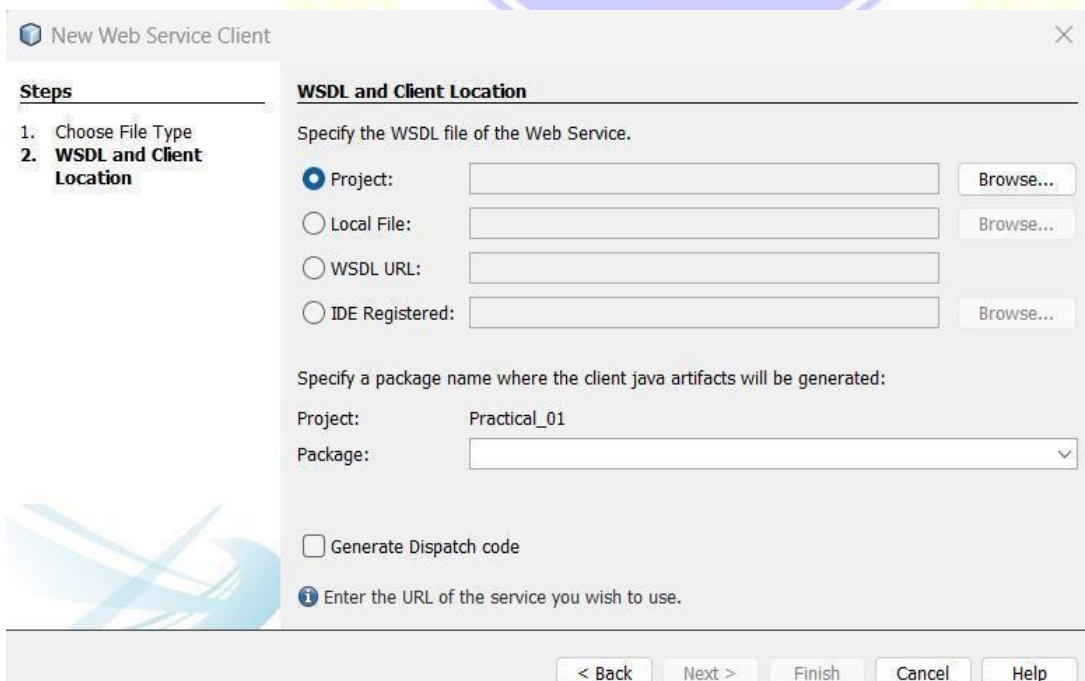
**Options:**

☒ JSP File (Standard Syntax) ☐ Create as a JSP Segment

☐ JSP Document (XML Syntax)

**Description:**

A JSP file using JSP standard syntax.



**New Web Service Client**

**Steps**

1. Choose File Type
2. **WSDL and Client Location**

**WSDL and Client Location**

Specify the WSDL file of the Web Service.

☒ Project:

☐ Local File:

☐ WSDL URL:

☐ IDE Registered:

Specify a package name where the client java artifacts will be generated:

Project:

Package:

☐ Generate Dispatch code

Enter the URL of the service you wish to use.

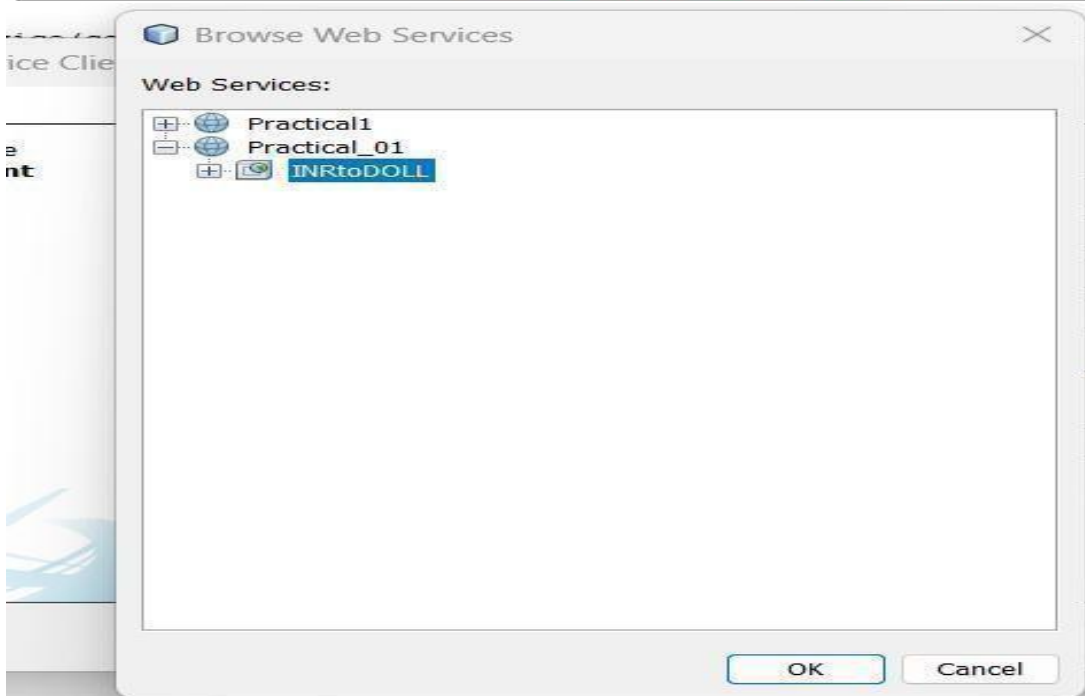




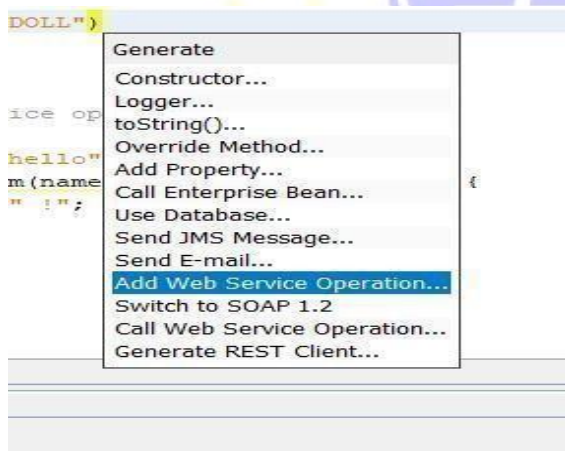
## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation



**Step 8:** Go to java file and right click on the screen ->insert code ->add web service operation.





**Step 9:** Give the operation name->click on add button ->give the name of variable and set the data type -> ok

Add Operation

Name: INRtodoll\_converter

Return Type: java.lang.String

Parameters

Name	Type	Final
INR	double	<input type="checkbox"/>

OK Cancel

```
10 import javax.jws.WebParam;
11
12 /**
13  *
14  * @author 91937
15  */
16 @WebService(serviceName = "INRtoDOLL")
17 public class INRtoDOLL {
18
19     /**
20      * This is a sample web service operation
21      */
22
23     /**
24      * Web service operation
25      */
26     @WebMethod(operationName = "INRtodoll_converter")
27     public String INRtodoll_converter(@WebParam(name = "INR") double INR) {
28         //TODO write your implementation code here:
29         return INR+" in the doller:"+(INR/83.11);
30     }
31 }
32
33
```

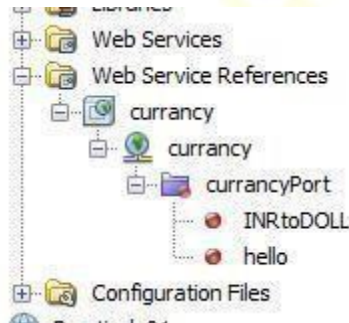


### Client as Java:

#### Step 10: Go to input.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>JSP Page</title>
  </head>
  <body>
    <form name="" action="output.jsp">
      <pre>
        Enter currency in rupee<input type="text" name="t1">
        <input type="submit"> <input type="reset">
      </pre>
    </form>
  </body>
</html>
```

#### Step 11: Open output.jsp. on the left side web service reference and select nestedly till the INR to DOLL pops



Drag and Drop the INR to DOLL

```
<%-- start web service invocation --%><hr/>
<%
try {
    client.Currency_Service service = new client.Currency_Service();
    client.Currency port = service.getCurrencyPort();
    // TODO initialize WS operation arguments here
    double inr = Double.parseDouble(request.getParameter("t1"));
    // TODO process result here
    java.lang.String result = port.inRtoDOLL(inr);
    out.println(result);
} catch (Exception ex) {
    // TODO handle custom exceptions here
}
%>
<%-- end web service invocation --%><hr/>
```

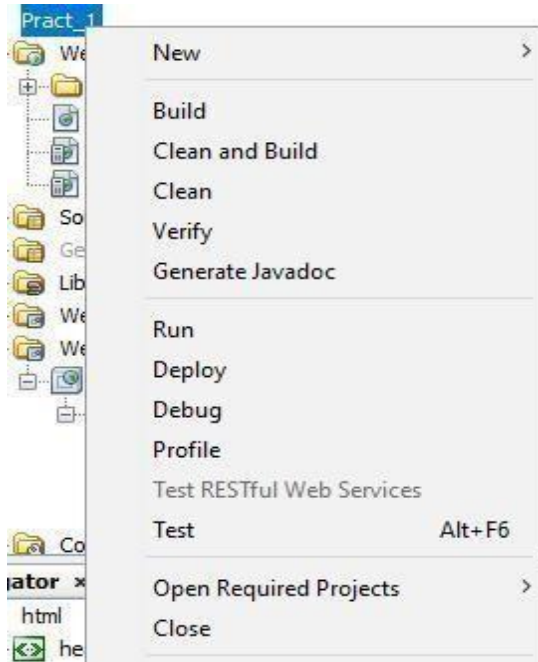


## SHRI G.P.M. DEGREE COLLEGE

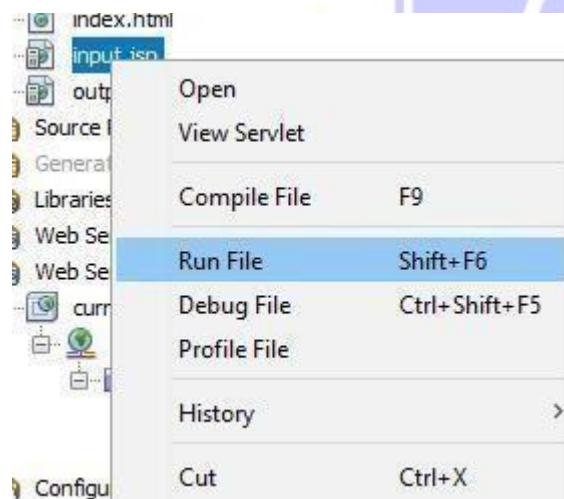
Department of Computer

Vision.. Innovation.. Solution.. Presentation

**Step 12:** Right click on the project name and deploy it.



**Step 13:** Right click on the input.jsp and select



### Output:

OUTPUT IS HERE:

Enter currancy in rupee

Learning Outcomes:

8000.0rupee equal to96.25797136325352DOLLER





## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### **Result and Discussion:**

The result involves creating a simple service, such as a currency converter from INR to USD, which can be accessed through APIs. The service can be called from platforms like Java and .NET by using HTTP requests to interact with the service, enabling cross-platform integration.

### **Learning Outcome:**

The learning outcome is the ability to design and implement a simple web service (e.g., currency conversion), and integrate it into applications built on different platforms (Java, .NET) using API calls, fostering cross-platform interoperability and enhancing skills in web services and HTTP communication.

### **Course Outcome:**

The course enables students to design and integrate simple web services (like currency converters) across different platforms (Java, .NET), focusing on web service development, API usage, and cross-platform communication.

### **Viva Questions:**

1. What is the purpose of using a RESTful API in this currency conversion service?
2. How does Flask handle invalid INR input in the API?
3. Explain how Java consumes the Flask API using HttpURLConnection.
4. How can you integrate a web service into a Java or .NET application?
5. What is the purpose of using APIs for cross-platform communication in web services?

### **For Faculty use:**

Correction Parameters	Formative Assessment[40 %]	Timely Completion of Practical[40%]	Attendance Learning Attitude[20%]



## Practical 2

### Create a simple SOAP services.

**Aim:** To create a Simple Web Service that converts the temperature from Celsius to Fahrenheit and vice versa. The aim is to create a simple SOAP-based web service that demonstrates how to exchange messages between a client and a server over the web. SOAP (Simple Object Access Protocol) is a protocol for exchanging structured information in the implementation of web services, and it uses XML as its message format.

#### Tools & Technologies used:

- Java (for developing the SOAP web service)
- Apache CXF or JAX-WS (Java API for XML Web Services)
- WSDL (Web Services Description Language) - for defining the service contract
- Apache Tomcat (for deploying the web service)
- Eclipse (IDE for development)

#### Theory:

##### 1. What is SOAP?

SOAP is a protocol specification for exchanging structured information in the implementation of web services. It relies on XML to encode the request and response messages, making it platform and language-independent.

##### 2. Components of SOAP Message:

Envelope: Contains the header and body, which hold the actual message content.

Header: Contains optional metadata, such as authentication information.

Body: Contains the actual message being passed between client and server.

##### 3. WSDL (Web Services Description Language):

WSDL is an XML-based language used for describing web services. It provides a standard way to describe the input and output of a service, as well as how to access it.



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Code (in Steps):

#### Step 1: Create the Web Service Interface:

This interface defines the service contract (the method the SOAP service will expose).

```
import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService
public interface GreetingService { @WebMethod
    String greet(String name);
}
```

#### Step 2: Implement the Web Service:

This class implements the GreetingServiceinterface and provides the business logic for the greet method.

```
import javax.jws.WebService;

@WebService(endpointInterface = "GreetingService")
public class GreetingServiceImpl implements GreetingService {

    @Override
    public String greet(String name) { return "Hello, "
        + name + "!";
    }
}
```

#### Step 3: Publish the Web Service:

The GreetingServicePublisherclass will expose the service at a specific URL using Endpoint.publish().

```
import javax.xml.ws.Endpoint;

public class GreetingServicePublisher { public static
    void main(String[] args) {
        // Publish the web service at the specified URL Endpoint.publish("http://localhost:8080/GreetingService", new
        GreetingServiceImpl());
    }
}
```





## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

```
        System.out.println("Service is running at http://localhost:8080/GreetingService");  
    }  
}
```

When you run the GreetingServicePublisherclass, it will publish the SOAP service at the URL <http://localhost:8080/GreetingService>.

### Step 4: Test the SOAP Web Service Using SOAP UI:

1. SOAP UI is a tool to test SOAP web services.
2. Open SOAP UI, create a new SOAP project.
3. In the project, paste the following WSDL URL:  
<http://localhost:8080/GreetingService?wsdl>
4. SOAP UI will generate the request and response for the greetoperation.
5. Test the greetoperation by passing a name parameter (e.g., "John").

### Step 5: Create a Client to Call the Web Service:

The following code demonstrates how to call the SOAP web service using Java.

```
import javax.xml.namespace.QName;  
import javax.xml.ws.Service; import  
java.net.URL;  
  
public class GreetingServiceClient {  
    public static void main(String[] args) throws Exception {  
        // WSDL URL  
        URL url = new URL("http://localhost:8080/GreetingService?wsdl");  
  
        // QName: namespace URI and the local part (service name)  
        QName qname = new QName("http://impl/", "GreetingServiceImplService");  
  
        // Create the service object  
        Service service = Service.create(url, qname);  
  
        // Get the port (proxy) for calling the web service methods GreetingService greetingService =  
        service.getPort(GreetingService.class);
```



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

```
// Call the greet method and print the result String result =  
greetingService.greet("World");  
System.out.println(result); // Expected output: Hello, World!  
}  
}
```

### Step 6: Maven Dependency (If Using Maven):

If you're using Maven, make sure to add the following dependencies for JAX-WS.

```
<dependencies>  
<dependency>  
<groupId>javax.jws</groupId>  
<artifactId>javax-jws-api</artifactId>  
<version>1.1</version>  
</dependency>  
<dependency>  
<groupId>javax.xml.ws</groupId>  
<artifactId>jaxws-api</artifactId>  
<version>2.3.1</version>  
</dependency>  
</dependencies>
```

### Output:

#### 1. Server Console (GreetingServicePublisher)

When you run the GreetingServicePublisher class, it will print the following message indicating that the service is up and running:

Service is running at http://localhost:8080/GreetingService

#### 2. Client Console (GreetingServiceClient)

When you run the GreetingServiceClient class, it will print the greeting message returned by the SOAP service:

Hello, World!



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### **Result and Discussion:**

A simple SOAP service in Java is created using the @WebService annotation, which generates the WSDL automatically. The service logic is then implemented and deployed on a web server like Apache Tomcat.

### **Learning Outcome:**

The learning outcome is gaining the ability to define a SOAP web service, generate its WSDL, implement business logic, and deploy it on a web server.

### **Course Outcome:**

The course outcome is to equip students with the skills to create, implement, and deploy SOAP web services using Java, including generating WSDL and integrating the services with web servers.

### **Viva Questions:**

1. What is SOAP, and how does it differ from REST?
2. What role does WSDL play in a SOAP web service?
3. What is the purpose of the @WebService annotation in Java?
4. What is the role of JAX-WS in creating SOAP web services in Java?
5. How does Java generate the WSDL file for a SOAP service?

### **For Faculty use:**

Correction Parameters	Formative Assessment[40 %]	Timely Completion of Practical[40%]	Attendance Learning Attitude[20%]



## Practical 3

### Create a simple REST services.

**Aim:** To create and deploy a simple RESTful web service that returns a greeting message. Demonstrate CRUD operations with suitable database using RESTful Web service.

#### Tools & Technologies used:

- Java (for developing the web service)
- JAX-RS (Java API for RESTful Web Services) for creating the REST API
- Jersey (reference implementation of JAX-RS)
- Apache Tomcat (for deploying the web service)
- Maven (for dependency management)
- Postman (for testing the REST API).

#### Theory:

REST (Representational State Transfer) is an architectural style for designing networked applications. It uses HTTP and follows the principles of stateless communication, meaning each request from a client contains all the information needed to process the request.

RESTful web services return responses in formats like JSON or XML. The most common HTTP methods used are:

- GET: To fetch data.
- POST: To submit data.
- PUT: To update data.
- DELETE: To delete data.

#### How Does It Work?

- REST is based on stateless communication between the client and server.
- Each RESTful service has a resource, which can be accessed using a URI (Uniform Resource Identifier).
- Data is typically transferred in JSON format, which is lightweight and human-readable.





## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Code:

#### Step 1: Install Flask

First, ensure that Python is installed on your system. If it's not installed, download and install Python from the official website: <https://www.python.org/downloads/>.

Once you have Python installed, you can use pip (Python's package manager) to install Flask. In your terminal, run the following command:

```
pip install Flask
```

#### Step 2: Create a Flask

1. Create a new file called app.py in your preferred directory.
2. Write the Code for the REST Service

In app.py, write the following Python code:

```
from flask import Flask

# Initialize the Flask application app =
Flask(__name__)

# Define a route for the /greet endpoint with GET method
@app.route('/greet', methods=['GET'])
def greet():
    return "Hello, World from Flask REST!"

# Run the Flask app
if __name__ == '__main__':
    app.run(debug=True)
```

Explanation of the Code:

- Flask(\_\_name\_\_): This initializes a Flask web application.
- @app.route('/greet', methods=['GET']): This decorator defines a route for the endpoint /greet that responds to HTTP GET requests.
- greet(): This function will return a simple greeting when the /greet endpoint is accessed.
- app.run(debug=True): This starts the Flask development server in debug mode, so you can see changes immediately and get detailed error messages if something goes wrong.



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Step 3: Run the Flask Application

Once you've written the code in app.py, open a terminal and navigate to the directory where app.py is located. Then, run the following command:

```
python app.py
```

**You should see output like this:**

- \* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
- \* Restarting with stat

This means your Flask application is running locally at <http://127.0.0.1:5000/>.

### Step 4: Test the REST Service

#### Option 1: Using a Web Browser

Open your web browser and visit the following URL:

<http://127.0.0.1:5000/greet>

**You should see the output:**

Hello, World from Flask REST!

#### Option 2: Using Postman

Open Postman.

Set the HTTP method to GET.

Enter the following URL in the Postman URL bar: <http://127.0.0.1:5000/greet>. Click the Send button.

You will get the response:

Hello, World from Flask REST!



## **SHRI G.P.M. DEGREE COLLEGE**

**Department of Computer**

Vision.. Innovation.. Solution.. Presentation

### **Option 3: Using CURL**

You can also use the curl command from the terminal:

```
curl http://127.0.0.1:5000/greet
```

The expected output will be:

Hello, World from Flask REST!

### **Output:**

There's what you should expect to see when testing the REST service. Browser Output:

URL: http://127.0.0.1:5000/greet

Postman Output:

Method: GET

URL: http://127.0.0.1:5000/greet

CURL Output:

```
curl http://127.0.0.1:5000/greet
```

Expected output:

Hello, World from Flask REST!







## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### **Result and Discussion:**

To create a simple REST service in Java, use JAX-RS annotations like @Path to define endpoints. Implement service methods with annotations like @GET, @POST, etc. Finally, deploy the service on a web server to handle HTTP requests.

### **Learning Outcome:**

The learning outcome is to understand how to create and expose RESTful web services in Java using JAX-RS annotations, implement various HTTP methods, and deploy the service on a web server to handle requests.

### **Course Outcome:**

The course outcome is to enable students to design, implement, and deploy RESTful web services in Java using JAX-RS, and to handle HTTP requests efficiently on a web server.

### **Viva Questions:**

1. What is Flask and how does it help in building RESTful APIs?
2. How do you create a route in Flask to handle HTTP requests?
3. What is the purpose of the jsonify function in Flask?
4. What is the purpose of the @Path annotation in a REST service in Java?
5. How does JAX-RS handle different HTTP methods like GET, POST, and PUT in a REST service?

### **For Faculty use:**

Correction Parameters	Formative Assessment[40%]	Timely Completion of Practical[40%]	Attendance Learning Attitude[20%]



## Practical 4

**Develop application to consume Google's search / Google's Map RESTful Web service.**

**Aim:** To develop an application that consumes Google's Search API or Google Maps **API** to display location data (such as address or place details).

### Tools & Technologies used:

- Python: Programming language for creating the application.
- Flask: A Python web framework to handle web requests and responses.
- Google Maps API: To retrieve data like places, addresses, and maps from Google.
- Requests: Python library to send HTTP requests to the API.
- JSON: For parsing the response data.

### Theory:

Google Maps provides various APIs that allow you to embed location-based services into your application. These services include:

- Geocoding: Convert addresses into geographic coordinates.
- Places API: Search for places (businesses, landmarks) based on location.
- Directions API: Get directions from one place to another.

### How to use Google APIs?

To use Google APIs, you need to obtain an API key from the Google Cloud Platform, which provides access to various services like the Maps API and Search API.

1. Go to the Google Cloud Console: <https://console.cloud.google.com/>
2. Create a new project or use an existing one.
3. Enable the desired APIs (e.g., Google Maps, Places API).
4. Create an API key.



## Code:

### Step 1: Install Dependencies

Before starting, install the required dependencies using pip:

```
pip install Flask requests
```

### Step 2: Create the Python Application (app.py)

Below is the Python code for a simple Flask application that uses Google's **Places API** to search for places.

```
from flask import Flask, jsonify,
request import requests

app = Flask(__name__)

# Google API Key (you need to replace this with your own key) API_KEY =
"YOUR_GOOGLE_API_KEY"

# Google Maps API URL
URL = "https://maps.googleapis.com/maps/api/place/textsearch/json"

@app.route('/search', methods=['GET']) def
search_places():
    query = request.args.get('query', 'restaurants in New York') # Default query if not provided
    location = request.args.get('location', '40.712776,-74.005974') # Default location (NYC)
    radius = request.args.get('radius', '5000') # Default radius (in meters)

# Construct the API request URL params =
{
'query': query, 'location': location,
'radius': radius, 'key': API_KEY
}

# Make the request to Google Maps API response =
requests.get(URL, params=params)

if response.status_code == 200:
```



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

```
places = response.json() # Parse JSON response results =
places.get('results', []) places_data = []

for place in results: places_data.append({
    'name': place['name'],
    'address': place.get('formatted_address', 'N/A'), 'rating': place.get('rating', 'N/A')
})

    return jsonify(places_data) else:
return jsonify({"error": "Unable to fetch data from Google API"}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

**Explanation of the Code:**

- requests.get(): This sends an HTTP GET request to the Google Maps API.
- params: Contains the query, location, radius, and API key that will be used in the API request.
- response.json(): Parses the JSON response returned by the Google Maps API.
- Flask route /search: The route handles GET requests and expects parameters like query(search term), location(latitude,longitude), and radius(search area radius in meters).
- JSON Response: The application returns the results in JSON format, including the name, address, and rating of each place.

### Testing the Application

#### Step 1: Run the Flask

To run the application, use the following command in your terminal:

```
python app.py
```

By default, the application will run at <http://127.0.0.1:5000/>.

#### Step 2: Testing with Browser/Postman

Test URL (Browser/Postman):

```
http://127.0.0.1:5000/search?query=restaurants&location=40.712776,-74.005974&radius=5000
```

This request will search for restaurants within a 5 km radius of New York City (latitude 40.712776, longitude -74.005974).





# SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

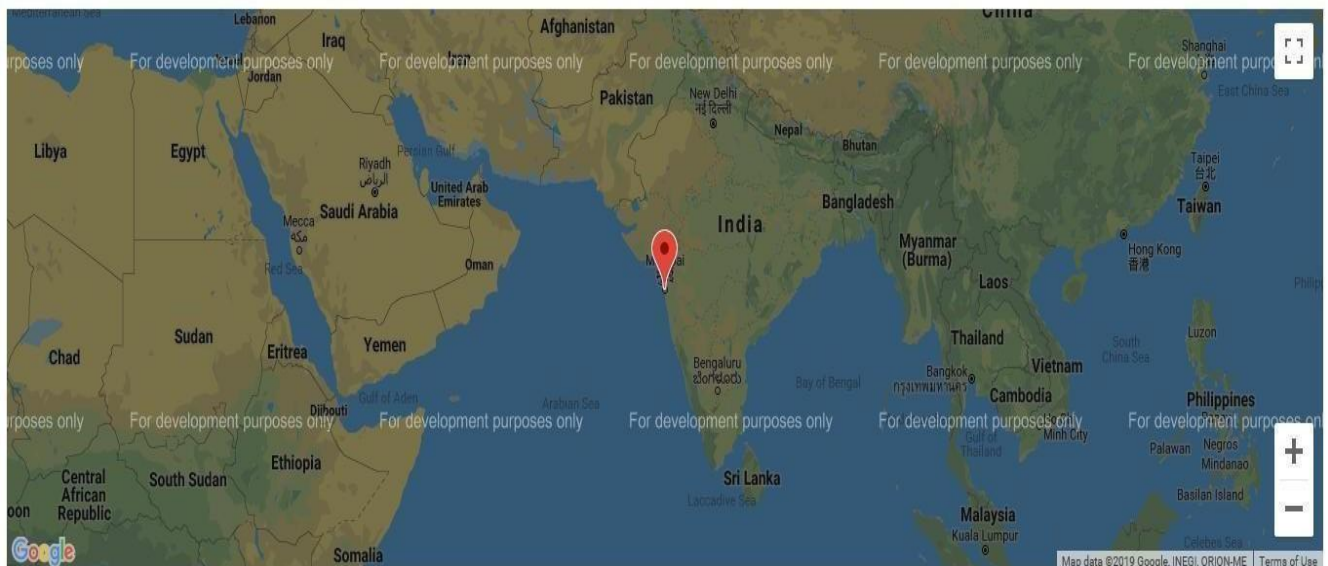
```
[  
{  
  
    "name": "Joe's Pizza",  
    "address": "7 Carmine St, New York, NY 10014, USA", "rating":  
    "4.5"  
},  
  
    "name": "Pizza City",  
    "address": "123 8th Ave, New York, NY 10010, USA", "rating":  
    "4.0"  
}  
]
```

This output contains the **name**, **address**, and **rating** of nearby restaurants.

Enter latitude:   
Enter longitude:



## Google Maps





## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### **Result and Discussion:**

The application uses HTTP requests to consume Google's Search or Maps RESTful APIs. It processes the response data (usually in JSON or XML). API keys are required for secure access to the Google services.

### **Learning Outcome:**

The learning outcome is to understand how to integrate and consume external RESTful APIs, such as Google's Search or Maps, using HTTP requests in a Java application and handle the response data.

### **Course Outcome:**

The course outcome is to equip students with the skills to develop applications that interact with external RESTful APIs, such as Google's Search or Maps, by making HTTP requests, processing responses, and using authentication methods like API keys.

### **Viva Questions:**

1. How can you extract and display data from a JSON response in Python?
2. What is the role of the API key when using Google APIs?
3. How do you consume a RESTful API in Python?
4. How do you make an HTTP request to consume a RESTful API in Java?
5. What is the purpose of an API key when consuming Google's RESTful services?

### **For Faculty use:**

Correction Parameters	Formative Assessment[40%]	Timely Completion of Practical[40%]	Attendance Learning Attitude[20%]



## Practical 5

### Installation and Configuration of virtualization using KVM.

**Aim:** The aim of this experiment is to understand and set up virtualization using KVM (Kernel-based Virtual Machine) on a Linux system. This will involve installing the necessary software, configuring the KVM hypervisor, and creating a virtual machine (VM).

#### Tools & Technologies used:

- KVM (Kernel-based Virtual Machine): A virtualization module in the Linux kernel that allows the user to run virtual machines.
- QEMU: A generic and open-source machine emulator and virtualizer that works with KVM to run virtual machines.
- Libvirt: A toolkit to manage virtual machines and virtualization technologies.
- Virt-Manager: A graphical user interface (GUI) to manage virtual machines.
- Linux-based OS (Ubuntu/Debian): The operating system used for this experiment.

#### Theory:

KVM (Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware. It uses hardware extensions (Intel VT or AMD-V) to provide virtualization support and enables running multiple virtual machines on a single physical host. KVM is built into the Linux kernel, and as a result, it is integrated into the Linux operating system.

#### KVM Components:

- QEMU: A machine emulator that provides hardware virtualization and is used in combination with KVM.
- Libvirt: A toolkit to manage virtualization technologies such as KVM. It provides a command-line and GUI interface for creating and managing virtual machines.
- Virt-Manager: A GUI tool to manage virtual machines on top of KVM.





## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Code:

Steps to Install and Configure KVM on Ubuntu (or any Debian-based system)

#### Step 1: Prerequisites

Ensure your system supports virtualization:

- Intel Processor: `grep --color vmx /proc/cpuinfo`
- AMD Processor: `grep --color svm /proc/cpuinfo`

If virtualization is supported, proceed with the installation.

#### Step 2: Installing KVM and Necessary Tools

Run the following commands to install KVM, QEMU, libvirt, and virt-manager:

```
sudo apt update  
sudo apt install qemu-kvm libvirt-daemon-system libvirt-clients bridge-utils virt-manager
```

- `qemu-kvm`: KVM binary package.
- `libvirt-daemon-system`: For managing virtual machines via libvirt.
- `libvirt-clients`: Command-line tools for interacting with virtual machines.
- `bridge-utils`: Tools for configuring network bridges.
- `virt-manager`: Graphical tool for managing virtual machines.

#### Step 3: Verify KVM Installation

After installation, verify that KVM is installed and functional:

```
sudo kvm-ok
```

This should display a message like KVM acceleration can be used.



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Step 4: Add User to libvirt and KVM Groups

To allow a non-root user to manage KVM, add them to the libvirt and kvm groups:

```
sudo usermod -aG libvirt,kvm $(whoami)
```

Log out and log back in to apply the group changes.

### Step 5: Start and Enable libvirt

Start and enable the libvirt service:

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

### Step 6: Configure Network for Virtual Machines

Create a network bridge to allow virtual machines to access the external network. This can be done by modifying the network configuration in /etc/network/interfaces (for older versions) or using NetworkManager (on modern systems).

Example bridge configuration:

```
auto  
br0  
iface br0 inet dhcp    bridge_ports  
eth0
```

Restart networking or reboot your system to apply the changes.

### Step 7: Creating a Virtual Machine Using Virt-Manager

Now, you can use virt-manager to create virtual machines. Open virt-manager:

```
virt-manager
```

In the GUI:

- Click "Create a new virtual machine."
- Choose the installation media (ISO file) and follow the prompts to configure CPU, memory, disk, and networking for your VM.



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Step 8: Creating a Virtual Machine Using Command Line

You can also create a virtual machine using the virt-install command. Example:

```
virt-install \
--name myvm \
--ram 2048 \
--vcpus 2 \
--disk path=/var/lib/libvirt/images/myvm.qcow2,size=20 \
--cdrom /path/to/your.iso \
--network bridge=br0 \
--os-type linux \
--os-variant ubuntu20.04 \
--graphics vnc
```

This command installs a virtual machine with 2GB RAM, 2 virtual CPUs, and 20GB of disk space.

### Output:

#### 1. Running the KVM Virtual Machine

After setting up your virtual machine, you can start it:

```
sudo virsh start myvm
```

To check the list of running VMs:

```
sudo virsh list --all
```

#### 2. Monitoring the VM

You can also connect to the console of the VM:

```
sudo virsh console myvm
```

If you're using virt-manager, you can simply open the VM window to interact with it.



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### **Result and Discussion:**

KVM virtualization on Linux allows efficient VM management through tools like virt-manager or virsh, requiring CPU virtualization support and package installation for proper setup.

### **Learning Outcome:**

The learning outcome of KVM virtualization installation and configuration includes gaining the ability to set up and manage virtual machines on Linux, understanding the necessary system requirements, and effectively utilizing virtualization tools to create and control VMs.

### **Course Outcome:**

The course outcome is to equip students with skills to install, configure, and manage virtual machines on Linux using KVM, focusing on system virtualization and resource optimization.

### **Viva Questions:**

1. What is KVM and how does it work?
2. What is the role of virt-manager in KVM virtualization?
3. What is the purpose of network bridging in KVM?
4. What is KVM, and how does it enable virtualization on Linux?
5. Can you explain the basic steps involved in setting up a virtual machine using KVM?

### **For Faculty use:**

Correction Parameters	Formative Assessment[40%]	Timely Completion of Practical[40%]	Attendance Learning Attitude[20%]





## Practical 6

**Develop application to download image / video from server or upload image / video to server using MTOM techniques.**

**Aim:** To develop a web service-based application that enables downloading and uploading of images or videos from/to a server using MTOM (Message Transmission Optimization Mechanism) techniques. MTOM is used to efficiently send binary data (like images or videos) over SOAP-based web services by encoding the data as attachments.

### Tools & Technologies Used:

- Java - Programming language to implement the web service client and server.
- Apache CXF or JAX-WS - For creating SOAP-based web services in Java.
- MTOM (Message Transmission Optimization Mechanism) - SOAP optimization technique for sending large binary data like images and videos.
- SOAP - Protocol for exchanging structured information in the implementation of web services.
- Eclipse IDE or IntelliJ IDEA - For Java application development.
- Apache Tomcat or any compatible servlet container - For hosting the web service.

### Theory:

MTOM (Message Transmission Optimization Mechanism) is a mechanism defined by the WS-I (Web Services Interoperability Organization) that enables the efficient transmission of large binary data, such as images, videos, and documents, over a SOAP-based web service. It works by sending the binary data as an attachment instead of embedding it directly in the SOAP message. This reduces message size and improves performance.

### Benefits:

- Optimizes the transmission of binary data.
- Reduces the size of the SOAP envelope.
- Allows large file uploads/downloads without disrupting SOAP message integrity.

### How does MTOM work?

**Server Side (Web Service):** The binary data (like an image or video) is converted to a DataHandler object and transmitted as a MIME attachment in the SOAP re

**Client Side:** The client receives the binary data as a MIME attachment and decodes it.



## Code:

Steps to Implement MTOM for File Upload/Download

### Step 1: Set up the Server (Web Service)

The server will expose a SOAP-based web service using JAX-WS with MTOM enabled for handling large binary files.

Server-side Code (Web Service)

```
import javax.jws.WebService;
import javax.activation.DataHandler;
import javax.jws.WebMethod;
import javax.xml.ws.soap.MTOM;

@WebService
@MTOM

public class
FileService {

    // Method to upload image/video (received as a DataHandler)
    @WebMethod
    public String uploadFile(DataHandler data) {
        try {
            // Save the uploaded file
            File file = new File("uploaded_" + data.getName());
            InputStream is = data.getInputStream();
            Files.copy(is, file.toPath(), StandardCopyOption.REPLACE_EXISTING);
            return "File uploaded successfully: " + file.getName();
        } catch (Exception e) { e.printStackTrace();
            return "Error uploading file.";
        }
    }
}
```



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

// Method to download image/video (send as a DataHandler)

@WebMethod

```
public DataHandler downloadFile(String fileName) {  
    try {  
        File file = new File(fileName); if  
(file.exists()) {  
            DataSource source = new FileDataSource(file);  
            return new DataHandler(source);  
        } else {  
            throw new FileNotFoundException("File not found.");  
        }  
    } catch (Exception e) { e.printStackTrace();  
        return null;  
    }  
}
```

@WebService: Marks the class as a web service.

@MTOM: Enables MTOM for optimizing the transmission of binary data.

uploadFile(): Method to handle file upload.

downloadFile(): Method to download the file from the server.





### Step 2: Set up the Client (Web Service Client)

The client will call the web service to either upload or download an image/video using MTOM. Client-side Code (SOAP Client)

```
import javax.xml.ws.Service;
import javax.activation.DataHandler;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;

public class FileClient {
    public static void main(String[] args) {
        try {
            // Create a service URL and QName
            URL url = new URL("http://localhost:8080/yourApp/FileService?wsdl");
            QName qname = new QName("http://service/", "FileService");

            // Create service and proxy
            Service service = Service.create(url, qname);
            FileService port = service.getPort(FileService.class);

            // Set MTOM for the client side
            BindingProvider bindingProvider = (BindingProvider) port;
            bindingProvider.getRequestContext().put(
                "javax.xml.ws.soap.MTOMFeature", new MTOMFeature(true)
            );
            // Upload File
            File file = new File("path_to_image_or_video");
            DataHandler handler = new DataHandler(new FileDataSource(file));
            System.out.println(port.uploadFile(handler));
```



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

// Download File

```
DataHandler downloadedFile = port.downloadFile("uploaded_image.jpg");
```

```
Files.copy(downloadedFile.getInputStream(),  
Paths.get("downloaded_image.jpg"));
```

```
System.out.println("File downloaded successfully!");
```

```
} catch (Exception e) { e.printStackTrace();
```

```
}
```

```
}
```

```
}
```

uploadFile(): Sends the file to the server.

downloadFile(): Downloads the file from the server and saves it locally.

### Step 3: Run the Application

Start the Server: Deploy the web service to a servlet container (like Tomcat).

Run the Client: Execute the client application to upload/download files.

निर्मलस्नेह उत्तम सेवाधम



## **SHRI G.P.M. DEGREE COLLEGE**

**Department of Computer**

Vision.. Innovation.. Solution.. Presentation

---

### **Output :**

Below is the expected output when the client uploads and downloads files using MTOM.

#### **Upload File Output:**

File uploaded successfully: uploaded\_image.jpg

Download File

Output:

File downloaded successfully!





## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### **Result and Discussion:**

MTOM-based application development optimizes the transfer of images and videos between client and server, improving performance by reducing message size and enhancing speed.

### **Learning Outcomes:**

The learning outcome is the ability to develop applications that efficiently transfer large binary data, such as images and videos, using MTOM techniques, enhancing performance in client-server communication.

### **Course Outcomes:**

The course outcome is to enable students to design and implement applications that efficiently upload and download large media files, such as images and videos, using MTOM techniques for optimized data transmission.

### **Viva Questions:**

1. What is MTOM and how does it optimize the transmission of binary data in SOAP?
2. How does the @MTOMAnnotation work in JAX-WS?
3. What is the role of the DataHandler class in handling binary data for file uploads and downloads?
4. What is MTOM, and how does it improve the transfer of large media files?
5. Can you explain the steps involved in uploading or downloading an image/video using MTOM in a client-server application?

### **For Faculty use:**

Correction Parameters	Formative Assessment[40%]	Timely Completion of Practical[40%]	Attendance Learning Attitude[20%]





## Practical 7

### Implement FOSS-Cloud Functionality VSI (Virtual Server Infrastructure)

#### Infrastructure as a service (IaaS), Storage.

**Aim:** To implement a Virtual Server Infrastructure (VSI) as part of a FOSS (Free and Open Source Software) Cloud platform, enabling Infrastructure as a Service (IaaS) and providing storage services. The goal is to create an environment where users can provision virtual servers, manage storage, and deploy virtualized workloads using open-source tools and technologies.

#### Tools & Technologies Used:

- OpenStack: Open-source cloud computing platform for managing and automating IaaS services like compute, networking, and storage.
- KVM (Kernel-based Virtual Machine): Hypervisor used to create and manage virtual machines on the host system.
- QEMU: Emulator and virtualizer that works with KVM to manage VMs.
- Ceph: Open-source software for creating scalable, distributed storage.
- Libvirt: Toolkit for managing OpenStack in IaaS:
- virtualized platforms, integrated with OpenStack.
- Nova (OpenStack Component): Manages the lifecycle of virtual machines.
- Neutron (OpenStack Component): Manages networking services for virtual machines.
- Horizon (OpenStack Dashboard): Web-based interface for managing OpenStack services (IaaS, networking, and storage).
- Linux (Ubuntu/CentOS): Operating systems used for the cloud infrastructure.



## Theory:

IaaS is a cloud computing model where virtualized computing resources (like virtual machines, networking, and storage) are provided over the internet. It allows organizations to rent computing resources on-demand instead of maintaining physical infrastructure.

In the case of Virtual Server Infrastructure (VSI), IaaS provides users with virtual machines (VMs) that can be provisioned, configured, and managed in a cloud environment, along with associated services like networking and storage.

Key Components of VSI in IaaS:

- **Compute (Virtual Machines):** Virtual servers that run applications and workloads.
- **Storage:** Persistent storage options (such as block storage and object storage) for VMs and data.
- **Networking:** Virtual networks for communication between VMs and the outside world.
- **Hypervisor (KVM):** Virtualizes hardware resources to run multiple VMs on a physical server.

OpenStack is a suite of open-source software that enables the creation and management of a private or public cloud infrastructure. Key components of OpenStack for IaaS:

- **Nova:** Manages compute resources, including creating, scheduling, and terminating VMs
- **Neutron:** Provides networking services, like creating virtual networks, routers and firewalls.
- **Cinder:** Provides block storage services for virtual machines.
- **Glance:** Manages VM images.
- **Ceph:** Scalable distributed storage for object and block storage.

Storage in OpenStack:

- **Block Storage:** Provides persistent storage volumes that can be attached to virtual machines.
- **Object Storage:** Offers scalable storage for large amounts of unstructured data (e.g., images, videos, backups).
- **Ceph Storage:** Provides a distributed object store, block storage, and file system capabilities.



## **Code:**

Steps to Implement VSI Using OpenStack (IaaS and Storage)

### **Step 1: Setting Up OpenStack (VSI, IaaS)**

OpenStack is typically installed using devstack, packstack, or kolla-ansible for a simpler setup or packstack/kola for production-grade setups. Here we assume the use of devstack for a basic setup.

Install OpenStack using Devstack:

```
sudo apt update  
sudo apt install git  
git clone https://github.com/openstack/devstack
```

```
cd devstack
```

```
./stack.sh
```

This will install all the necessary components like Nova, Neutron, Cinder, and Glance to provision virtual machines and storage.

### **Step 2: Setting Up Ceph Storage for OpenStack**

Ceph can be used to provide highly available and scalable storage.

Install Ceph:

```
sudo apt install ceph ceph-deploy
```

Set Up Ceph Cluster: Follow the official Ceph documentation to create and configure a Ceph cluster for use with OpenStack.

Integrate Ceph with OpenStack: Modify the OpenStack configuration files to use Ceph for storage backend (for Cinder and Glance).



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Step 3: Provisioning Virtual Machines (VSI) in OpenStack

Once OpenStack is set up, you can create virtual machines.

Create a VM using Horizon (OpenStack Dashboard):

Access the Horizon dashboard at <http://<controller-ip>/dashboard>.

Under Project > Compute > Instances, click Launch Instance.

Choose an image (Ubuntu, CentOS, etc.), select the flavor (size of the VM), and configure networking.

Click Launch to create the virtual machine.

Create a VM using

```
openstack server create --flavor m1.small --image <image-id> --network <network-id>
<vm-name>
```

### Step 4: Managing Block Storage (Cinder)

You can create and manage persistent storage volumes for your virtual machines.

Create a Storage Volume:

```
openstack volume create --size 10 <volume-name>
```

Attach the Volume to a VM:

```
openstack server add volume <vm-name> <volume-id>
```

Detach the Volume:

```
openstack server remove volume <vm-name> <volume-id>
```

### Step 5: Testing the Cloud Environment

Test the provisioning, storage, and management of virtual machines by:

- Starting and Stopping VMs: Use the OpenStack CLI or Horizon to manage VM lifecycle.
- Monitoring Resource Usage: Use Horizon to monitor CPU, RAM, and storage utilization.
- Testing Storage Operations: Attach and detach volumes to VMs and ensure the data is persistent.





## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Output:

Create a Virtual Machine (VSI) with CLI:

```
openstack server create --flavor m1.small --image ubuntu-20.04 --network private- network my-vm-1
```

Field	Value
OS-EXT-STS:power_state	Running
OS-EXT-STS:task_state	None
OS-EXT-STS:vm_state	Active
name	my-vm-1
status	ACTIVE

Create a Storage Volume:

```
openstack volume create --size 10 my-volume
```

Field	Value
id	8f819da7-54b9-4b55-8467-123f07e957f5
status	available
size	10



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### **Result and Discussion:**

Implementing FOSS-Cloud for VSI in an IaaS model enables scalable, on-demand virtual server and storage resources, focusing on efficient resource allocation and secure access.

### **Learning Outcome:**

The learning outcome is the ability to implement and manage a scalable IaaS cloud infrastructure, providing virtual servers and storage, while ensuring efficient resource management and secure access.

### **Course Outcome:**

The course outcome is to equip students with the skills to implement and manage FOSS-Cloud functionality for VSI in an IaaS model, enabling them to deploy scalable virtual server and storage resources in a cloud environment.

### **Viva Questions:**

1. What is the role of OpenStack in implementing IaaS (Infrastructure as a Service)?
2. How does Ceph storage integrate with OpenStack for managing block storage?
3. What is the purpose of Nova and Neutron components in OpenStack?
4. What is VSI in the context of IaaS, and how does it function in a cloud environment?
5. How does FOSS-Cloud contribute to the implementation of IaaS for virtual servers and storage?

### **For Faculty use:**

Correction Parameters	Formative Assessment[40%]	Timely Completion of Practical[40%]	Attendance Learning Attitude[20%]



## Practical 8

### Implement FOSS-Cloud Functionality - VSI Platform as a Service (PaaS)

**Aim:** To implement a Platform as a Service (PaaS) cloud functionality using FOSS (Free and Open Source Software) in a Virtual Server Infrastructure (VSI). This platform allows developers to deploy and manage applications without worrying about the underlying infrastructure. The goal is to enable users to deploy applications seamlessly in a cloud environment, offering tools for development, hosting, and scaling.

#### Tools & Technologies Used:

- OpenStack: Open-source cloud computing platform for managing compute, storage, and networking resources.
- Cloud Foundry: Open-source PaaS offering a developer-friendly environment to build, deploy, and run applications.
- Kubernetes: Open-source container orchestration platform that automates the deployment and scaling of containerized applications.
- Docker: Containerization tool that packages applications and their dependencies into containers.
- Nginx: Web server for managing requests and routing traffic to applications.
- MySQL/PostgreSQL: Relational database management systems for application storage.
- Apache Tomcat/Node.js: Web application servers for deploying Java or Node.js-based applications.

#### Theory:

PaaS is a cloud computing model where the cloud provider supplies and manages everything required to host and run applications, including hardware, operating systems, storage, networking, and middleware. PaaS allows developers to focus solely on building their applications without worrying about the underlying infrastructure.

#### Benefits of PaaS:

- Ease of Deployment: Applications can be deployed quickly with minimal configuration.
- Scalability: PaaS platforms automatically handle scaling and resource allocation based on demand.
- Built-in Tools: Tools for version control, monitoring, logging, and more are integrated into the platform.
- Cost-Effective: Developers pay only for the resources they use.

#### Key Components of a PaaS Platform:

- Compute Resources: Virtual machines or containers that run the applications.
- Middleware: Frameworks and tools required to run applications (e.g., databases, web servers).
- Databases: Managed relational or NoSQL databases for storing application data.
- Networking: Management of application networking, including load balancing and traffic routing.
- Storage: Persistent storage for application data and logs.
- Developer Tools: CLI, IDE integration, and version control tools for deployment.



## Code:

Steps to Implement a PaaS Platform on VSI using FOSS Tools

### Step 1: Setting Up OpenStack for Cloud Infrastructure

OpenStack will be used to provision and manage compute resources for deploying the PaaS environment. This includes setting up virtual machines (VMs) to run Docker containers and Kubernetes clusters.

Install OpenStack (DevStack):

Follow the steps to install OpenStack using DevStack (simple setup for testing purposes):

```
git clone https://github.com/openstack/devstack.git
```

```
cd devstack
```

```
./stack.sh
```

Set Up Compute Resources:

Use OpenStack's Nova component to provision VMs or bare metal servers that will host the PaaS platform.

### Step 2: Set Up Kubernetes for Container Orchestration

Kubernetes will be used to manage containerized applications in the PaaS platform.

Install Kubernetes on OpenStack VMs: Deploy Kubernetes on your OpenStack VMs to manage Docker containers:

```
sudo apt-get update && sudo apt-get install -y kubeadm kubelet kubectl sudo
```

```
kubeadm init
```

Set Up a Kubernetes Cluster: Initialize your Kubernetes master node and join worker nodes to form the cluster.





## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Step 3: Set Up Docker Containers

Docker will be used to containerize applications that will be deployed to the PaaS environment.

Install Docker: Install Docker on each VM or container node:

```
sudo apt-get install docker.io
```

Run Sample Application: For a simple example, we will deploy a Python Flask app inside a Docker container. Create a Dockerfile:

```
FROM python:3.8-slim
```

```
WORKDIR /app
```

```
COPY ./app
```

```
RUN pip install -r requirements.txt CMD
```

```
["python", "app.py"]
```

Build and run the container:

```
docker build -t flask-app .
```

```
docker run -d -p 5000:5000 flask-app
```

### Step 4: Set Up Cloud Foundry (Optional)

Cloud Foundry is another option for creating a PaaS environment that automates app deployment.

Install Cloud Foundry CLI: Install the Cloud Foundry CLI to interact with your PaaS environment.

```
sudo apt install cf-cli
```

Deploy an Application to Cloud Foundry: Push a sample Node.js app to Cloud

```
cf push my-app
```



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Step 5: Set Up Managed Databases

Provide managed databases (e.g., MySQL, PostgreSQL) for application data storage.

Install MySQL/PostgreSQL on Kubernetes: Use Kubernetes to deploy a MySQL container.

Example for deploying MySQL:

```
kubectrl run mysql --image=mysql:5.7 --env="MYSQL_ROOT_PASSWORD=root"
```

```
kubectrl expose pod mysql --port=3306
```

### Step 6: Set Up Networking (Ingress and Load Balancing)

Use Nginx or Kubernetes Ingress Controller to manage application traffic and expose apps to the internet.

Install Nginx for Load Balancing:

Install and configure Nginx as a reverse proxy for your app: bash

```
sudo apt install nginx
```

Configure Load Balancer:

Set up Nginx or a Kubernetes ingress controller to route traffic between multiple instances of your app.



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Output:

Deploy Flask Application Using Docker:

Here is a simple Python Flask app (app.py): python

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello():
    return "Hello, PaaS
World!" if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Dockerfile for Building the Container:

```
FROM python:3.8-slim
WORKDIR /app
COPY . /app
```

RUN pip install -r requirements.txt CMD  
["python", "app.py"]

Build and Run the Application:

Build and run the container:

docker build -t flask-app .

docker run -d -p 5000:5000 flask-app

Access the Application:

After running the container, access the app by navigating to <http://<VM-IP>:5000>.

### Output:

"Hello, PaaS  
World!".



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### **Result and Discussion:**

Implementing FOSS-Cloud for VSI in a PaaS model offers a scalable, cost-effective platform for developers to build and deploy applications without managing underlying infrastructure.

### **Learning Outcome:**

The learning outcome is the ability to design and deploy applications on a PaaS using FOSS-Cloud, focusing on platform scalability, resource management, and efficient application development.

### **Course Outcomes:**

The course outcome is to enable students to implement and manage VSI-based PaaS solutions using FOSS-Cloud, empowering them to build, deploy, and scale applications efficiently in a cloud environment.

### **Viva Questions:**

1. What is the role of Kubernetes in a PaaS environment?
2. How does Docker help in deploying applications in a PaaS setup?
3. What are the benefits of using Cloud Foundry for PaaS compared to other platforms?
4. What is the role of PaaS in cloud computing, and how does it differ from IaaS?
5. How does FOSS-Cloud support the deployment and management of applications in a PaaS environment?

### **For Faculty use:**

Correction Parameters	Formative Assessment[40%]	Timely Completion of Practical[40%]	Attendance Learning Attitude[20%]





## Practical 9

### Using AWS Flow Framework develop application that includes a simple workflow.

Workflow calls an activity to print hello world to the console. It must define the basic usage of AWS Flow Framework including defining contracts, implementation activities and workflow coordination logic and worker programs to host them.

**Aim:** To develop an application using the AWS Flow Framework that defines a simple workflow calling an activity to print "Hello World" to the console. This application will demonstrate the basic usage of the AWS Flow Framework, including defining contracts, implementing activities, workflow coordination logic, and the worker program that hosts the activities.

#### Tools & Technologies Used:

- AWS Flow Framework: A framework to build workflows with Amazon Simple Workflow Service (SWF).
- Amazon SWF: Managed service for coordinating tasks and workflows.
- Java: Programming language used for implementing the AWS Flow Framework logic.
- AWS SDK: Software development kit for accessing AWS services, including SWF.
- Maven: Build tool for compiling and managing dependencies.
- AWS CLI: Command Line Interface for interacting with AWS services.
- Eclipse or IntelliJ IDEA: IDE for Java development.

#### Theory:

The AWS Flow Framework is a higher-level abstraction built on top of Amazon Simple Workflow Service (SWF). It helps developers build scalable and reliable applications by allowing them to define workflows, activities, and coordination logic using workflows as the control flow. Activities are implemented as simple Java methods and are orchestrated within a workflow.

#### Key Concepts:

- Workflow: A series of activities that are orchestrated together. It represents the overall process.
- Worker: A program that executes activities. It polls for tasks from the SWF service and executes them.
- Task List: A list of tasks that workers process asynchronously.
- Decider: Coordinates workflow execution and manages task scheduling.
- Activity Worker: Runs in the background and executes activity methods defined in the workflow.



## Code: Steps to Develop the Simple Workflow Application

### Step 1: Set Up Maven Project for AWS Flow Framework

To begin, create a Maven project and add dependencies to your pom.xml.

```
<dependencies>
<dependency>
<groupId>com.amazonaws</groupId>
<artifactId>aws-java-sdk-swf</artifactId>
<version>1.12.78</version>
</dependency>
<dependency>
<groupId>com.amazonaws</groupId>
<artifactId>aws-flow-framework</artifactId>
<version>1.0.1</version>
</dependency>
</dependencies>
```

### Step 2: Define the Workflow Contract

The workflow contract defines the methods that the workflow will call. In our case, the workflow will call an activity to print "Hello World" to the console.

```
package com.example.workflow;
import com.amazonaws.flow.workflow.Task;
import com.amazonaws.flow.workflow.Workflow;
import com.amazonaws.flow.workflow.WorkflowType;
@Workflow
public interface HelloWorldWorkflow {
@Task
void helloWorldActivity();
}
```

In this contract, the helloWorldActivity() method is a placeholder for the activity that prints "Hello World".



### Step 3: Implement the Activity

The activity will contain the logic to print "Hello World" to the console

```
package com.example.activity;
import com.amazonaws.flow.activity.Activity;
import com.amazonaws.flow.activity.ActivityImplementation;
@ActivityImplementation
public class HelloWorldActivityImpl implements HelloWorldActivity {
    @Override
    public void helloWorld() {
        System.out.println("Hello World from AWS Flow Framework!");
    }
}
```

Here, we define the HelloWorldActivity interface and its implementation, which simply prints "Hello World".

### Step 4: Define the Worker Program

The worker is responsible for executing the activities defined in the workflow.

```
package com.example.worker;
import com.amazonaws.flow.worker.Worker;
import com.amazonaws.flow.worker.WorkerOptions;
import com.amazonaws.flow.worker.activity.ActivityWorker;
import com.amazonaws.flow.workflow.WorkflowWorker;
```



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

```
import com.example.activity.HelloWorldActivityImpl; import
com.example.workflow.HelloWorldWorkflow;

public class HelloWorldWorker {

    public static void main(String[] args) {
        // Define task list
        String taskList = "HelloWorldTaskList";

        // Create activity worker
        ActivityWorker activityWorker = new ActivityWorker(taskList, HelloWorldActivityImpl.class);

        WorkerOptions workerOptions = new
        WorkerOptions().withDomain("HelloWorldDomain").withTaskList(taskList);
        // Create workflow worker

        WorkflowWorker workflowWorker = new WorkflowWorker(taskList,
        HelloWorldWorkflow.class);

        workflowWorker.addActivityImplementation(new HelloWorldActivityImpl());
        // Start the worker

        Worker worker = new Worker(workerOptions); worker.start();
    }
}
```

Here, the worker is created to poll tasks from the HelloWorldTaskList, execute the HelloWorldActivity, and manage the workflow execution.

### Step 5: Define the Decider for Workflow Coordination

The decider will coordinate the execution of the workflow. It will invoke the activity defined in the workflow.

```
package com.example.decision;

import com.amazonaws.flow.workflow.Workflow;
import com.amazonaws.flow.workflow.WorkflowType;
import com.amazonaws.flow.workflow.WorkflowWorker;
```





```
public class HelloWorldWorkflowDecider {
    public static void main(String[] args) {
// Define task list and workflow
String taskList = "HelloWorldTaskList";
WorkflowType workflowType = new WorkflowType(HelloWorldWorkflow.class);
// Create and start the workflow worker
WorkflowWorker workflowWorker = new WorkflowWorker(taskList, workflowType);
workflowWorker.start();
}
}
```

The decider is responsible for making decisions regarding the sequence of activity executions in the workflow.

#### **Step 6: Running the Application**

Once the project is set up, you can start the workflow by executing the HelloWorldWorker class. It will start polling for tasks from the specified task list and will execute the helloWorldActivity() method, which will print "Hello World" to the console.

#### **Output:**

Once you run the worker, the console output will be:  
Hello World from AWS Flow Framework!

This output is generated by the HelloWorldActivityImplclass when it executes the helloWorld()  
\$java -cp target/aws-flow-framework-example-1.0.jar com.example.worker.HelloWorldWorker

Hello World from AWS Flow Framework!



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Result and Discussion:

Using the AWS Flow Framework allows the creation of scalable workflows with minimal coding, focusing on task orchestration, fault tolerance, and performance optimization.

### Learning Outcomes:

The learning outcome is the ability to design and develop scalable applications using the AWS Flow Framework, focusing on creating efficient workflows and optimizing task orchestration.

### Course Outcomes:

The course outcome is to equip students with the skills to develop applications using the AWS Flow Framework, enabling them to design, implement, and optimize workflows for scalable and efficient task management.

### Viva Questions:

1. What is the role of a decider in the AWS Flow Framework?
2. How does the worker program interact with AWS SWF in the context of workflow execution?
3. What is the purpose of defining an activity contract in the AWS Flow Framework?
4. What is the AWS Flow Framework, and how does it help in creating workflows?
5. Can you explain the key components involved in developing a simple workflow using the AWS Flow Framework?

### For Faculty use:

Correction Parameters	Formative Assessment[40%]	Timely Completion of Practical[40%]	Attendance Learning Attitude[20%]



## Practical 10

### Implementation of Openstack with user and private network creation.

**Aim:** To implement OpenStack cloud infrastructure with the ability to create a user and set up a private network. This setup allows users to manage cloud resources like virtual machines (VMs), networks, and storage securely and isolated from the public network.

#### Tools & Technologies Used:

- OpenStack: An open-source cloud computing platform for creating and managing public and private cloud infrastructures.
- OpenStack CLI: Command-line interface to interact with OpenStack services.
- Keystone: Identity service in OpenStack used for authentication and authorization of users.
- Neutron: Networking component of OpenStack used to manage networks and subnets.
- Nova: Compute service that manages virtual machine instances.
- Linux (Ubuntu/CentOS): Operating systems used for setting up OpenStack.

#### Theory:

OpenStack is an open-source cloud computing platform that provides infrastructure as a service (IaaS) to manage and automate pools of compute, storage, and networking resources. It allows users to run virtual machines, create private networks, and deploy various cloud services.

#### Key OpenStack Components:

- Keystone: Identity and authentication service.
- Nova: Compute service that manages virtual machines.
- Neutron: Networking service for creating networks, routers, subnets, and managing network isolation.
- Cinder: Block storage service.
- Glance: Image service for managing virtual machine images.
- Horizon: Web dashboard to interact with OpenStack services.



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

In OpenStack, Keystone is used to create users and assign roles. A user can be assigned to a project (tenant) and given permissions based on their role. The common roles include admin, member, and reader.

Neutron is responsible for creating private networks, subnets, and routers in OpenStack. Private networks are isolated from the public network, providing security for VMs running within them. Neutron enables flexible and scalable networking for virtual machines (VMs) in the cloud.

### Steps:

#### Step 1: Install OpenStack

You can install OpenStack using DevStack for testing purposes or using Packstack for a more production-ready setup. Here we assume a DevStack setup for simplicity.

# Clone DevStack repository

```
git clone https://github.com/openstack/devstack.git
```

```
cd devstack
```

# Run DevStack installation

```
./stack.sh
```

This will install all the necessary services for OpenStack, including Keystone, Nova, Neutron, and more.

#### Step 2: Create a User with Keystone

Once OpenStack is installed, the first step is to create a user.

Log in to OpenStack as the adminuser:

```
source /opt/stack/devstack/openrc admin admin
```

Create a new user:

```
openstack user create --domain default --password-prompt new_user
```

This command will prompt for a password and create a user named new\_user under the default domain.

Assign a role to the user:

```
openstack role add --project demo --user new_user member
```

This assigns the member role to the user new\_user in the demo project.





## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Step 3: Create a Private Network Using Neutron

Create a private network:

```
openstack network create private-network
```

This command creates a private network called private-network.

Create a subnet for the private network:

```
openstack subnet create --network private-network --subnet-range 192.168.1.0/24 private-subnet
```

This creates a subnet within the private-network with the IP range 192.168.1.0/24.

Create a network for the private network:

```
openstack router create private-router
```

This creates a router that will connect the private network to the external public network.

Set the router gateway to the external network:

```
openstack router set --external-gateway public-network private-router
```

This command sets the gateway for the private-router to the public-network (assumed to be configured).

Connect the private network to the router:

```
openstack router add subnet private-router private-subnet
```

This connects the private subnet to the router, enabling communication between the private network and the external network.

### Step 4: Verify the Network Setup

List networks:

```
openstack network list
```

This will show the created networks, including private-network.

List subnets:

```
openstack subnet list
```

This will show the subnets created, including private-subnet.

List routers:

```
openstack router list
```

This will show the router named private-router.



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Step 5: Create and Launch a Virtual Machine in the Private Network

Create a VM instance using the private network:

```
openstack server create --flavor m1.small --image ubuntu-20.04 --network private-network --key-name mykey private-vm
```

This command creates a new VM named private-vm in the private-network using the ubuntu-20.04 image and m1.small flavor.

Verify the VM status:

```
openstack server list
```

This will list all the servers, including the private-vm.

### Code with Output Example:

#### Create a user:

```
$ openstack user create --domain default --password-prompt new_user User
```

Password: \*\*\*\*\*

Confirm Password: \*\*\*\*\*

#### Create a private network:

```
$ openstack network create private-network
```

Field	Value
admin_state_up	True
id	a12345b678c90d1e2f34g5h678i9j01
name	private-network
status	ACTIVE

#### Create a subnet:

```
$ openstack subnet create --network private-network --subnet-range 192.168.1.0/24 private-subnet
```

Field	Value
cidr	192.168.1.0/24
id	12345b678c90d1e2f34g5h678i9j01
ip_version	4
name	private-subnet
network_id	a12345b678c90d1e2f34g5h678i9j01



## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### Create a VM:

```
$ openstack server create --flavor m1.small --image ubuntu-20.04 --network private- network --key-name mykey private-vm
```

Field	Value
OS-EXT-STS:power_state	Running
name	private-vm
status	ACTIVE





## SHRI G.P.M. DEGREE COLLEGE

Department of Computer

Vision.. Innovation.. Solution.. Presentation

### **Result and Discussion:**

Implementing OpenStack with user and private network creation enables secure, isolated environments for users to manage virtual machines and networks, with a focus on configuration, security, and resource optimization.

### **Learning Outcome:**

The learning outcome is the ability to configure and manage OpenStack environments, including the creation of user and private networks, ensuring secure and efficient resource allocation in a cloud infrastructure.

### **Course Outcome:**

The course outcome is to enable students to implement and manage OpenStack, including the creation of user and private networks, and to optimize resource allocation for secure and scalable cloud infrastructure.

### **Viva Questions:**

- 1.What is the purpose of creating a private network in OpenStack?
- 2.How does OpenStack manage user roles and permissions through Keystone?
- 3.What are the steps involved in launching a virtual machine in OpenStack's private network?
- 4.What is OpenStack, and how does it facilitate cloud infrastructure management?
- 5.How do you create a private network for users in OpenStack, and what are its benefits?

### **For Faculty use:**

Correction Parameters	Formative Assessment[40%]	Timely Completion of Practical[40%]	Attendance Learning Attitude[20%]