**Aim 1. Create a web application to find multiplication of two numbers using Simple SOAP service.**

**🔷 Step 1: Create a New Web Application**

1. Open **NetBeans**.
2. Click **File → New Project**.
3. Select:
   * **Java Web → Web Application**
   * Click **Next**
4. Name your project: MultiplicationService
   * Click **Next**
5. Choose **GlassFish Server**
   * Java EE Version: Select Java EE 6 Web or higher
   * Click **Finish**

**🔷 Step 2: Create a Web Service**

1. Right-click the **project name (MultiplicationService)** → **New → Web Service**
2. Set:
   * **Web Service Name**: MultiplyNumbers
   * **Package**: com.service
   * Click **Finish**

**🔷 Step 3: Write the Web Service Logic**

NetBeans will open MultiplyNumbers.java.

Replace the code with this:

package com.service;

import javax.jws.WebService;

import javax.jws.WebMethod;

import javax.jws.WebParam;

@WebService(serviceName = "MultiplyNumbers")

public class MultiplyNumbers {

@WebMethod(operationName = "multiply")

public int multiply(@WebParam(name = "num1") int num1, @WebParam(name = "num2") int num2) {

return num1 \* num2;

}

}

**🔷 Step 4: Deploy the Project**

1. Right-click the **project** → **Run**
2. Wait for **GlassFish Server** to start and deploy your project.

Your browser will open automatically at:

http://localhost:8080/MultiplicationService/

**🔷 Step 5: Access the WSDL (Optional)**

Try opening:

http://localhost:8080/MultiplicationService/MultiplyNumbers?wsdl

✅ If you see XML WSDL output — your service is successfully deployed!

**🔷 Step 6: Test the Web Service**

1. In NetBeans, expand:
   * MultiplicationService
   * → Web Services
   * → MultiplyNumbers
2. Right-click MultiplyNumbers → Click **Test Web Service**
3. Your browser will open the test page:
4. http://localhost:8080/MultiplicationService/MultiplyNumbers?Tester
5. Enter values like: 5 and 6
6. Click multiply()  
   ✅ Output should be: 30

**Aim 2. Write a Program to implement web Service to Convert temperature from Fahrenheit to Celsius. Use following formula for conversion. Formula: C = (°F - 32) × 5/9. Deploy the above Web service properly and call it by designing appropriate interface in JAVA / .NET**

**🛠 Step-by-Step Instructions**

**✅ Step 1: Create a New Web Application Project**

1. Open **NetBeans**.
2. Click: File > New Project
3. Choose:
   * **Java with Ant > Web Application**
4. Click **Next**.
5. Project Name: TempConverterService
6. Server: **GlassFish Server** (or Payara)
7. Java EE Version: **Java EE 7 Web**
8. Click **Finish**

**✅ Step 2: Create the Web Service**

1. In the **Projects** pane, right-click on Source Packages
2. Select: New > Web Service
3. Name: TemperatureService
4. Package: com.temp (or any name you like)
5. Click **Finish**

**✅ Step 3: Add the Conversion Method**

Replace the default method with this code:

package com.temp;

import javax.jws.WebService;

import javax.jws.WebMethod;

@WebService(serviceName = "TemperatureService")

public class TemperatureService {

@WebMethod(operationName = "convertFahrenheitToCelsius")

public double convertFahrenheitToCelsius(double fahrenheit) {

double celsius = (fahrenheit - 32) \* 5.0 / 9.0;

return celsius;

}

}

✅ Save the file.

**✅ Step 4: Deploy the Project**

1. Right-click the project TempConverterService
2. Click Run

Your browser will open:

http://localhost:8080/TempConverterService/

But your actual service will be here (try it in browser): (Optional)

http://localhost:8080/TempConverterService/TemperatureServiceService?wsdl

If you see the WSDL, it’s working ✅

**✅ Step 5: Test the Web Service (Tester Page)**

1. In **NetBeans**, expand:
2. TempConverterService
3. └── Web Services
4. └── TemperatureService
5. Right-click TemperatureService → Click Test Web Service

This opens:

http://localhost:8080/TempConverterService/TemperatureServiceService?Tester

**✅ Step 6: Use the Web Tester**

You’ll see:

* Input box for Fahrenheit value
* A button with method name: convertFahrenheitToCelsius

**Try it:**

* Enter 100
* Click the button

✅ You’ll get:

* Celsius output: 37.77778
* SOAP request/response XML

**Aim 3. Create a Virtual machine using KVM and configure as given below: -**

**a) Check KVM launching.**

**b) Create Virtual machine and connect it to KVM.**

**c) Create Network bridge**

**✅ STEP 1: Check if your system supports KVM**

**🧪 Run:**

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

* If the result is 1 or more → ✅ your CPU supports virtualization.
* If 0 → ❌ Enable virtualization in your BIOS.

Also verify:

kvm-ok

If kvm-ok not found, install it:

sudo apt install cpu-checker

**✅ STEP 2: Install KVM and Tools**

sudo apt update

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

This installs:

* KVM
* Virtual machine manager (GUI)
* Network bridge tools

**✅ STEP 3: Add Your User to the libvirt Group**

sudo usermod -aG libvirt $USER

Then **log out and log back in** for changes to take effect.

**✅ STEP 4: Check KVM Status (Part a of Practical)**

sudo systemctl status libvirtd

* If it's running → ✅
* If not, start it:

sudo systemctl start libvirtd

sudo systemctl enable libvirtd

To confirm KVM is ready:

virsh list --all

If you see a table (even empty) → You're good.

✅ **✔️ KVM is launching properly**

**✅ STEP 5: Create a Virtual Machine (Part b)**

Use the GUI tool: virt-manager

1. Run it:
2. virt-manager
3. Click on **"Create a new virtual machine"**.
4. Choose:
   * Local install media (ISO file)
   * Or use existing OS image (like Ubuntu ISO)
5. Select the ISO file path (download Ubuntu ISO if needed).
6. Assign memory and CPUs (e.g., 2048 MB RAM, 2 CPUs).
7. Create a virtual disk (e.g., 10GB).
8. Choose the network type (use default NAT or later choose Bridge).
9. Click **Finish**.

🎉 ✅ Your virtual machine is created and connected to KVM.

**✅ STEP 6: Create a Network Bridge (Part c)**

Now, let’s make your VM act like it's part of your LAN.

**🔧 1. Edit netplan or interfaces file:**

Check your active interface name:

ip addr

Usually it's enp3s0, eth0, or something like that.

**📝 2. Create Netplan config for a bridge:**

Edit the file:

sudo nano /etc/netplan/01-network-manager-all.yaml

Example config:

network:

version: 2

renderer: NetworkManager

ethernets:

enp3s0: {}

bridges:

br0:

interfaces: [enp3s0]

dhcp4: true

(Replace enp3s0 with your actual Ethernet name)

**🔄 3. Apply changes:**

sudo netplan apply

Now br0 acts as your bridge network.

**✅ STEP 7: Attach VM to Bridge Network**

In virt-manager:

1. Open the VM settings.
2. Go to **"NIC"** or "Network" tab.
3. Change the source from NAT to Bridge → select br0.
4. Save and reboot VM.

🎉 ✅ Now your VM is using bridged networking.

**Aim 4. Create a web application to find largest of two numbers using Simple SOAP service.**

**✅ Step 1: Create a New Web Application Project**

1. Open **NetBeans**
2. Go to File > New Project
3. Select: Java with Ant → Web Application
4. Click **Next**
5. Set:
   * **Project Name:** LargestNumberService
   * **Server:** GlassFish Server (or Payara)
   * **Java EE Version:** Java EE 7 Web or higher
6. Click **Finish**

**✅ Step 2: Create the SOAP Web Service**

1. In the **Projects** pane, right-click on Source Packages
2. Choose: New > Web Service
3. Set:
   * **Web Service Name:** LargestNumber
   * **Package:** com.soap (or any package name you like)
4. Click **Finish**

**✅ Step 3: Write the Logic to Find the Largest Number**

Replace the default code with:

package com.soap;

import javax.jws.WebService;

import javax.jws.WebMethod;

@WebService(serviceName = "LargestNumber")

public class LargestNumber {

@WebMethod(operationName = "findLargest")

public int findLargest(int num1, int num2) {

return (num1 > num2) ? num1 : num2;

}

}

✅ Save the file.

**✅ Step 4: Deploy the Project**

1. Right-click the project LargestNumberService
2. Click Run

Wait for GlassFish to start and deploy the service.

Your browser will open:

http://localhost:8080/LargestNumberService/

Now try to visit the **WSDL**: (Optional)

http://localhost:8080/LargestNumberService/LargestNumberService?wsdl

If you see WSDL XML output → your SOAP web service is working ✅

**✅ Step 5: Test the Web Service**

1. In **NetBeans**, expand:
2. LargestNumberService
3. └── Web Services
4. └── LargestNumber
5. Right-click on LargestNumber → Click Test Web Service

This opens:

http://localhost:8080/LargestNumberService/LargestNumberService?Tester

**🔍 Test It:**

* Enter values: 45 and 67
* Click **findLargest()**
* Output: 67

🎉 You’ve successfully tested your SOAP web service!

**Aim 5. Create a web application to find cube of a number using Simple SOAP service.**

**🔧 Step-by-Step Guide**

**🛠 Step 1: Create a New Web Application**

1. Open **NetBeans IDE**
2. Go to: File → New Project
3. Choose: Java Web → Web Application → Click Next
4. Project Name: CubeService
5. Server: **GlassFish Server**  
   Java EE Version: **Java EE 7 or 8** → Click Next
6. Frameworks: **(leave blank)** → Click Finish

**📦 Step 2: Create the Web Service**

1. In the **Projects pane**, right-click CubeService → New → Web Service
2. Service Name: CubeCalculator
3. Package Name: service → Click Finish

It creates a file CubeCalculator.java with a default method.

**✏️ Step 3: Add the Cube Method**

Replace the default operation() method with the cube method:

@WebMethod(operationName = "findCube")

public int findCube(@WebParam(name = "number") int number) {

return number \* number \* number;

}

🧠 **Explanation:**

* @WebMethod – exposes this method in WSDL
* @WebParam – defines input parameter name

**🚀 Step 4: Deploy the Web Service**

1. Right-click the project → Click **Run**
2. NetBeans will:
   * Compile the project
   * Start GlassFish
   * Deploy your SOAP service
   * Open browser:  
     http://localhost:8080/CubeService/

✅ Deployment is successful if you see the home page.

**🧪 Step 5: Test the Web Service**

1. In NetBeans, go to:
2. CubeService
3. └── Web Services
4. └── CubeCalculator
5. Right-click on CubeCalculator → Click **Test Web Service**
6. It opens a Tester page:  
   http://localhost:8080/CubeService/CubeCalculatorService?Tester
7. Enter a number: 4
8. Click on **findCube()**

✅ Output: 64

**Aim 6. Create a web application to find addition of a three number using Simple SOAP service.**

**🛠 Step-by-Step Instructions**

**🧱 Step 1: Create a New Web Application**

1. Open **NetBeans IDE**
2. Go to File → New Project
3. Select: Java Web → Web Application → Click Next
4. Project Name: AdditionService
5. Choose Server: **GlassFish Server**
6. Java EE Version: **Java EE 7/8** → Click Next
7. Frameworks: leave blank → Click Finish

**🧩 Step 2: Create the Web Service**

1. In the Projects panel, right-click the AdditionService project
2. Select: New → Web Service
3. Service Name: AddThreeNumbers
4. Package Name: service → Click Finish

**✏️ Step 3: Add Web Method for Addition**

NetBeans will create a default method in AddThreeNumbers.java. Replace it with:

@WebMethod(operationName = "addThreeNumbers")

public int addThreeNumbers(

@WebParam(name = "num1") int num1,

@WebParam(name = "num2") int num2,

@WebParam(name = "num3") int num3) {

return num1 + num2 + num3;

}

✅ This method takes 3 integers and returns their sum.

**🚀 Step 4: Deploy the Web Service**

1. Right-click the project name → Click **Run**
2. NetBeans will start **GlassFish** and deploy the service.
3. A browser opens:  
   http://localhost:8080/AdditionService/

**🧪 Step 5: Test the Web Service**

1. In NetBeans:
2. AdditionService
3. └── Web Services
4. └── AddThreeNumbers
5. Right-click AddThreeNumbers → Click **Test Web Service**
6. A new tab opens:  
   http://localhost:8080/AdditionService/AddThreeNumbersService?Tester
7. Enter: 10, 20, 30
8. Click on **addThreeNumbers()**

✅ Output: 60 (correct addition result)

**Aim 7. Create a Simple REST Service to demonstrate CRUD operations with “Player” database. Implement the service with fields viz. Player\_Name, age, Country, Runs. Deploy the service and execute the operations.**

**🔧 Step 1: Check If Node.js is Installed**

1. Open **Terminal** (Linux/Mac) or **Command Prompt** (Windows).
2. Type the following command:
3. node -v
4. If a version number is shown (e.g., v20.0.0), Node.js is installed.
5. If not, download and install Node.js from the official website:

👉 <https://nodejs.org/>

**🧱 Step 2: Set Up JSON Server**

**✅ Install JSON Server Globally: (Recommended)**

In the terminal, type:

npm install -g json-server

If you prefer to install it locally, you can use npx later.

**📁 Step 3: Create the Database File**

1. Open your terminal in the folder where you want to create the file.
2. Create a new file named playerdb.json.

If you have Visual Studio Code installed, use:

code playerdb.json

**(Recommended),** just create a file manually in any editor like Notepad or VS Code. **(Create a Folder name CRUD open it in VS code – create playerdb.json file and paste the following code – save and close file must).**

1. Paste the following data into playerdb.json:

{

"players": [

{

"id": 1,

"playerName": "Virat",

"age": 36,

"country": "India",

"runs": 12000

}

]

}

1. Save and close the file.

**🚀 Step 4: Start the JSON Server**

**(Open CRUD Folder from the Desktop – right click anywhere – select “Open In Terminal” and paste the Globally command and enter – will get the REST API)**

**If installed globally, run:**

json-server --watch playerdb.json

**If installed locally, run:**

npx json-server playerdb.json

**🔗 Your REST API is now running at:**

👉 <http://localhost:3000/players>

**🔍 Step 5: Open Postman to Test the API**

**➤ Download Postman (if not already installed):**

👉 <https://www.postman.com/downloads/>

**After Login in Postman Software beside the “Overview” click Plus (+) sign there paste the URL and select the Methods as per the step 6. (For each method open new tab by clicking plus).**

**🧪 Step 6: Test CRUD Operations Using Postman**

**1️⃣ GET – View All Players**

* **Method**: GET
* **URL**: http://localhost:3000/players
* **Action**: Click **Send**
* **Result**: Shows the list of all players.

**2️⃣ POST – Add a New Player**

* **Method**: POST
* **URL**: http://localhost:3000/players
* **Action**:
  + Go to **Body > raw**
  + Select **JSON** format
  + Paste the following:

{

"id": "2",

"playerName": "Rohit",

"age": 37,

"country": "India",

"runs": 11000

}

* **Click**: Send

**3️⃣ PATCH – Update Only One Field (Partial Update)**

* **Method**: PATCH
* **URL**: http://localhost:3000/players/1 (or any player's ID)
* **Body**:

{

"runs": 15000

}

* **Click**: Send

**4️⃣ PUT – Replace the Entire Player Record**

* **Method**: PUT
* **URL**: http://localhost:3000/players/1
* **Body**:

{

"id": "1",

"playerName": "Dhoni",

"age": 43,

"country": "India",

"runs": 17000

}

* **Click**: Send

**5️⃣ DELETE – Remove a Player**

* **Method**: DELETE
* **URL**: http://localhost:3000/players/1
* **Click**: Send

**Aim 8. Implementation of Openstack with user and private network creation.**

**🔑 Step 1: Login to the Horizon Dashboard**

1. Open a browser and go to your OpenStack dashboard URL, typically:
2. http://<your-openstack-ip>/dashboard
3. Login as **admin** (username: admin, password: <your-password>).

**👤 Step 2: Create a New User**

1. From the **top menu**, click on **"Identity"** > **"Users"**.
2. Click on the **"+ Create User"** button.
3. Fill in the details:
   * **User Name**: testuser (or any name)
   * **Password**: yourpassword
   * **Confirm Password**
   * **Email**: testuser@example.com
   * **Primary Project**: You can create one or use admin
   * **Role**: Leave as default (\_member\_)
4. Click **"Create User"**.

✅ Done! User is created.

**🏢 Step 3: Create a New Project (Tenant)**

1. Go to **"Identity"** > **"Projects"**.
2. Click on **"+ Create Project"**.
3. Fill:
   * **Project Name**: testproject
   * **Description**: Project for user
   * **Domain**: Leave default
   * **Enabled**: ✅ Yes
4. Click **"Create Project"**.

✅ Now you have a project that you can assign the user to.

**👥 Step 4: Assign User to the Project**

1. Go back to **"Identity"** > **"Projects"**.
2. Find your project (testproject) and click **"Manage Members"**.
3. In the left list, find testuser, click the arrow to move to the right.
4. Assign a role like \_member\_ (or admin if needed).
5. Click **"Save"**.

✅ User testuser is now part of the testproject.

**🌐 Step 5: Create a Private Network**

Now switch to the **new user** you created, to create a private network.

1. Logout from admin and login as testuser.
2. Switch to the **project scope** (testproject).
3. Go to **"Network"** > **"Networks"**.
4. Click **"+ Create Network"**.

**Step 5.1: Network Tab**

* **Network Name**: private-net
* Leave **Admin State** checked.
* Click **"Next"**.

**Step 5.2: Subnet Tab**

* **Subnet Name**: private-subnet
* **Network Address**: 192.168.100.0/24 (example)
* Click **"Next"**.

**Step 5.3: Subnet Details Tab**

* Enable **DHCP**
* Set **Gateway IP**: 192.168.100.1 (optional)
* Click **"Create"**

✅ You now have a private network created for this project.

**🔄 Optional Step 6: Create a Router to Connect Private Network to External**

1. Go to **"Network"** > **"Routers"**.
2. Click **"+ Create Router"**
   * Name: router1
   * External Network: public (or whatever is available)
3. Click **"Create Router"**

**Attach Interface**

1. Click the newly created router > **"Interfaces"** tab > **"Add Interface"**.
2. Choose the **private subnet** you created earlier.
3. Click **"Add Interface"**

✅ This connects your private network to the outside world.

**🚀 Optional Step 7: Launch an Instance (VM)**

1. Go to **"Project"** > **"Compute"** > **"Instances"**
2. Click **"Launch Instance"**
3. Fill:
   * **Instance Name**: vm1
   * **Flavor**: choose m1.small or similar
   * **Source**: choose a bootable image like Ubuntu or Cirros
   * **Networks**: Select private-net
   * **Key Pair**: Create or select one
4. Click **"Launch"**

✅ Your VM will boot on the private network.

**Aim 9. Create a Simple REST Service to demonstrate CRUD operations with “Product” database. Implement the service with fields viz. Product\_Name, Quantity, Expiry\_date. Deploy the service and execute the operations.**

**🔧 Step 1: Check If Node.js is Installed**

1. Open **Terminal** (Linux) or **Command Prompt** (Windows).
2. Type the following command:
3. node -v
4. If a version is displayed (e.g., v20.0.0), you're good to go.
5. If not installed, download and install Node.js from:

👉 <https://nodejs.org/>

**🧱 Step 2: Install JSON Server**

**✅ Install it globally: (Recommended)**

npm install -g json-server

Or if you want to use it only once, use npx in Step 4.

**📁 Step 3: Create the Database File**

1. Open terminal and navigate to your project folder.
2. Create a file named productdb.json:
3. code productdb.json

**(Recommended),** open any editor (like Notepad, gedit, or VS Code) and create the file manually. **(Create a Folder name CRUD open it in VS code – create productdb.json file and paste the following code – save and close file must).**

1. Paste the following sample data:

{

"products": [

{

"id": 1,

"Product\_Name": "Milk",

"Quantity": 10,

"Expiry\_date": "2025-04-30"

}

]

}

1. Save the file.

**🚀 Step 4: Start JSON Server**

**(Open CRUD Folder from the Desktop – right click anywhere – select “Open In Terminal” and paste the Globally command and enter – will get the REST API)**

**If installed globally, use:**

json-server --watch productdb.json

**If not installed globally, use:**

npx json-server productdb.json

**🔗 API will be available at:**

👉 http://localhost:3000/products

**🔍 Step 5: Open Postman to Test the API**

**➤ Download Postman (if not installed):**

👉 <https://www.postman.com/downloads/>

**After Login in Postman Software beside the “Overview” click Plus (+) sign there paste the URL and select the Methods as per the step 6. (For each method open new tab by clicking plus).**

**🧪 Step 6: Perform CRUD Operations Using Postman**

**1️⃣ GET – View All Products**

* **Method**: GET
* **URL**: http://localhost:3000/products
* **Click**: **Send**
* **Result**: List of all products

**2️⃣ POST – Add a New Product**

* **Method**: POST
* **URL**: http://localhost:3000/products
* **Go to**: Body → raw → Select JSON
* **Example Body**:

{

"id": "2",

"Product\_Name": "Bread",

"Quantity": 5,

"Expiry\_date": "2025-04-30"

}

* **Click**: **Send**

**3️⃣ PATCH – Update a Field (e.g., Quantity)**

* **Method**: PATCH
* **URL**: http://localhost:3000/products/1 (use valid ID)
* **Body**:

{

"Quantity": 20

}

* **Click**: **Send**

**4️⃣ PUT – Replace the Whole Product Record**

* **Method**: PUT
* **URL**: http://localhost:3000/products/1
* **Body**:

{

"id": "1",

"Product\_Name": "Cheese",

"Quantity": 8,

"Expiry\_date": "2025-05-02"

}

* **Click**: **Send**

**5️⃣ DELETE – Remove a Product**

* **Method**: DELETE
* **URL**: http://localhost:3000/products/1
* **Click**: **Send**

**Aim 10. Create a Simple REST Service to demonstrate CRUD operations with “Student” database. Implement the service with fields viz. Student\_Name, age, Phone\_no, Subject. Deploy the service and execute the operations.**

**✅ Step-by-Step Instructions:**

**🛠 Step 1: Check if Node.js is installed**

1. Open **Terminal** (Linux/macOS) or **Command Prompt** (Windows).
2. Type the command:
3. node -v
4. If a version is shown, Node.js is installed.
5. If not, download and install it from: 👉 <https://nodejs.org/>

**🧰 Step 2: Install JSON Server**

1. **Type this command to install JSON Server globally: (Recommended)**

npm install -g json-server

If you don’t want global install, you can also use:

npx json-server studentdb.json

**📁 Step 3: Create the Database File**

1. Open Terminal or Command Prompt.
2. Run:
3. code studentdb.json

**(Recommended),** open any text editor (e.g. Notepad, VS Code) and create a file manually named studentdb.json. **(Create a Folder name CRUD open it in VS code – create citydb.json file and paste the following code – save and close file must).**

1. Paste the following sample JSON data:

{

"students": [

{

"id": 1,

"Student\_Name": "Logan Anderson",

"age": 20,

"Phone\_no": "9876543210",

"Subject": "Computer Science"

}

]

}

1. Save and close the file.

**▶️ Step 4: Start JSON Server**

**(Open CRUD Folder from the Desktop – right click anywhere – select “Open In Terminal” and paste the Globally command and enter – will get the REST API)**

1. Run the server with for globally:

json-server --watch studentdb.json

Or if you didn’t install it globally:

npx json-server studentdb.json

1. You will see:
2. \{^\_^}/ hi!
3. Loading studentdb.json
4. Resources: http://localhost:3000/students

Your REST API is now live at 👉 http://localhost:3000/students

**✅ Step 5: Perform CRUD Operations Using Postman**

📥 Download Postman if you don’t have it:  
👉 <https://www.postman.com/downloads/>

**After Login in Postman Software beside the “Overview” click Plus (+) sign there paste the URL and select the Methods as per the step 5. (For each method open new tab by clicking plus).**

**1️⃣ GET – View All Students**

* **URL**: http://localhost:3000/students
* **Method**: GET
* Click **Send**  
  ✅ You’ll get a list of all student records.

**2️⃣ POST – Add a New Student**

* **URL**: http://localhost:3000/students
* **Method**: POST
* Go to: **Body → raw → JSON**
* Paste:

{

"id": "2",

"Student\_Name": "Lily Kay",

"age": 21,

"Phone\_no": "9123456789",

"Subject": "Mathematics"

}

* Click **Send**  
  ✅ A new student will be added with a unique ID.

**3️⃣ PATCH – Update One Field of a Student**

* **URL**: http://localhost:3000/students/1 (replace 1 with correct ID)
* **Method**: PATCH
* Body → raw → JSON:

{

"age": 22

}

✅ This updates just the age field.

**4️⃣ PUT – Replace Entire Student Record**

* **URL**: http://localhost:3000/students/1
* **Method**: PUT
* Body → raw → JSON:

{

"id": "1",

"Student\_Name": "Tom Ray",

"age": 23,

"Phone\_no": "9876543210",

"Subject": "Physics"

}

✅ This replaces the full record.

**5️⃣ DELETE – Remove a Student**

* **URL**: http://localhost:3000/students/1
* **Method**: DELETE  
  ✅ This deletes the student with ID 1.

**Aim 11. Create a Simple REST Service to demonstrate CRUD operations with “Employee” database. Implement the service with fields viz. Employee\_Name, age, Phone\_no, Department. Deploy the service and execute the operations.**

**🛠 Step 1: Check If Node.js is Installed**

Open **Terminal (Ubuntu)** or **Command Prompt (Windows)** and type:

node -v

If Node.js is not installed, download and install from:  
👉 <https://nodejs.org/>

**🛠 Step 2: Set Up JSON Server**

**🔹 1. Install JSON Server globally: (Recommended)**

npm install -g json-server

If you don’t want to install globally, you can use npx in the next steps.

**🔹 2. Create a new file for database:**

code employeedb.json

**(Recommended),** Or use any text editor like Notepad, VS Code, or Nano. **(Create a Folder name CRUD open it in VS code – create citydb.json file and paste the following code – save and close file must).**

**🔹 3. Paste this sample employee data inside the file:**

{

"employees": [

{

"id": 1,

"employeeName": "Ben Lee",

"age": 28,

"phone\_no": "9876543210",

"department": "HR"

}

]

}

✅ id is required. JSON Server uses it as a unique key.

**🔹 4. Start the server:**

**(Open CRUD Folder from the Desktop – right click anywhere – select “Open In Terminal” and paste the Globally command and enter – will get the REST API)**

* 1. Start the server for globally:

json-server --watch employeedb.json

Or if not installed globally:

npx json-server employeedb.json

🌐 API will be available at:  
👉 [**http://localhost:3000/employees**](http://localhost:3000/employees)

**📮 Step 3: Test Your API with Postman**

👉 Download Postman if not installed:  
<https://www.postman.com/downloads/>

**After Login in Postman Software beside the “Overview” click Plus (+) sign there paste the URL and select the Methods as per the step 3. (For each method open new tab by clicking plus).**

**1️⃣ GET – View All Employees**

* **Method:** GET
* **URL:** http://localhost:3000/employees

**2️⃣ POST – Add a New Employee**

* **Method:** POST
* **URL:** http://localhost:3000/employees
* **Go to:** Body → raw → select JSON
* **Example:**

{

"employeeName": "Amy Fox",

"age": 30,

"phone\_no": "9123456789",

"department": "IT"

}

**3️⃣ PATCH – Update One Field**

* **Method:** PATCH
* **URL:** http://localhost:3000/employees/1
* **Body:**

{

"age": 29

}

**4️⃣ PUT – Replace Entire Record**

* **Method:** PUT
* **URL:** http://localhost:3000/employees/1
* **Body:**

{

"employeeName": "Ben Lee",

"age": 28,

"phone\_no": "9876543210",

"department": "Marketing"

}

**5️⃣ DELETE – Remove an Employee**

* **Method:** DELETE
* **URL:** http://localhost:3000/employees/1

**Aim 12. Write a Program to implement web Service to Convert Currency from Rupees to Dollar. Use following formula for conversion. Conversion Rate: 1 USD = 75 INR. Deploy the above Web service properly and call it by designing appropriate interface in .NET. / JAVA**

**✅ Step 1: Create a New Web Application Project**

1. Open **NetBeans IDE**
2. Click **File → New Project**
3. In the **New Project** window:
   * Category: **Java Web**
   * Project: **Web Application**
   * Click **Next**
4. In the next window:
   * Project Name: CurrencyConverterWebService
   * Click **Next**
5. Select Server: **GlassFish Server**
   * Java EE Version: **Java EE 7 Web**
   * Click **Finish**

**✅ Step 2: Create a Web Service**

1. In the **Projects** pane (left panel), right-click on the project CurrencyConverterWebService
2. Click **New → Web Service**
3. In the dialog box:
   * Web Service Name: CurrencyConverter
   * Package: server
   * Click **Finish**

**✅ Step 3: Add Web Method**

1. In the CurrencyConverter.java file, **right-click inside the class**
2. Click **Insert Code...**
3. Choose **Add Web Service Operation**
4. In the dialog box:
   * Name: InrtoDollar
   * Parameter:
     + Name: a
     + Type: double
   * Return Type: double
   * Click **OK**

📝 Now your method will look like this:

@WebMethod(operationName = "InrtoDollar")

public double InrtoDollar(double a) {

return a / 75;

}

**✅ Step 4: Deploy the Project**

1. Right-click the project name CurrencyConverterWebService
2. Click **Deploy**

Wait for the message: **"BUILD SUCCESSFUL"** in the output.

**✅ Step 5: Test the Web Service**

1. In the **Projects** pane, expand:
   * CurrencyConverterWebService
   * Web Services
   * Right-click on CurrencyConverter
2. Click **Test Web Service**

🧪 A browser window will open with a test form.

* Enter a value in the text box (e.g., 150)
* Click **InrtoDollar**
* It will return the result in **USD**

Example:

Input: 150

Output: 2.0 (150 / 75 = 2 USD)

**Aim 13. Create a Simple REST Service to demonstrate CRUD operations with “City” database. Implement the service with fields viz. City\_Name, District\_Name, Population. Deploy the service and execute the operations.**

**🔧 Step 1: Check If Node.js is Installed**

1. Open Terminal (Linux/Mac) or Command Prompt (Windows).
2. Type the following command:
3. node -v
4. If a version number is shown (e.g., v20.0.0), Node.js is installed.
5. If not, download and install Node.js from the official website:  
   👉 <https://nodejs.org/>

**🧱 Step 2: Set Up JSON Server**

✅ Install JSON Server Globally: **(Recommended)**  
In the terminal, type:

npm install -g json-server

If you prefer to install it locally, you can use npx later.

**📁 Step 3: Create the Database File**

1. Open your terminal in the folder where you want to create the file.
2. Create a new file named citydb.json.  
   If you have Visual Studio Code installed, use:
3. code citydb.json

**(Recommended)**, just create a file manually in any editor like Notepad or VS Code. **(Create a Folder name CRUD open it in VS code – create citydb.json file and paste the following code – save and close file must).**

1. Paste the following data into citydb.json:

{

"cities": [

{

"id": 1,

"cityName": "Mumbai",

"districtName": "Mumbai Suburban",

"population": 20411000

}

]

}

1. Save and close the file.

**🚀 Step 4: Start the JSON Server**

**(Open CRUD Folder from the Desktop – right click anywhere– select “Open In Terminal” and paste the Globally command and enter – will get the REST API)**

If installed globally, run:

json-server --watch citydb.json

If installed locally, run:

npx json-server citydb.json

🔗 Your REST API is now running at:  
👉 http://localhost:3000/cities

**🔍 Step 5: Open Postman to Test the API**

➤ Download Postman (if not already installed):  
👉 <https://www.postman.com/downloads/>

**After Login in Postman Software, beside the “Overview” click the Plus (+) sign. Paste the URL and select the Methods as per Step 6. (For each method, open a new tab by clicking the plus sign.)**

**🧪 Step 6: Test CRUD Operations Using Postman**

**1️⃣ GET – View All Cities**

* **Method:** GET
* **URL:**
* http://localhost:3000/cities
* **Action:** Click **Send**
* **Result:** Shows the list of all cities.

**2️⃣ POST – Add a New City**

* **Method:** POST
* **URL:**
* http://localhost:3000/cities
* **Action:**
  + Go to **Body** > **raw**
  + Select **JSON format**
  + Paste the following:

{

"id": "2",

"cityName": "Pune",

"districtName": "Pune",

"population": 6800000

}

* **Click:** Send

**3️⃣ PATCH – Update Only One Field (Partial Update)**

* **Method:** PATCH
* **URL:**
* http://localhost:3000/cities/1
* **Body:**

{

"population": 21000000

}

* **Click:** Send

**4️⃣ PUT – Replace the Entire City Record**

* **Method:** PUT
* **URL:**
* http://localhost:3000/cities/1
* **Body:**

{

"id": "1",

"cityName": "Nagpur",

"districtName": "Nagpur",

"population": 2900000

}

* **Click:** Send

**5️⃣ DELETE – Remove a City**

* **Method:** DELETE
* **URL:**
* http://localhost:3000/cities/1
* **Click:** Send

**Aim 14. Create a web application to find addition of a two number using Simple SOAP service.**

**🔹 Step 1: Open NetBeans IDE**

* Launch **NetBeans IDE**.
* Wait for it to fully load.

**🔹 Step 2: Create a Web Application Project**

1. Go to **File → New Project**.
2. In the left pane, select **Java Web**.
3. In the right pane, select **Web Application**.

Click **Next**.

1. Give your project a name, e.g., AdditionSOAPWebService.

Click **Next**.

1. Choose the server:
   * Select **GlassFish Server** or **Apache Tomcat** (GlassFish is better for SOAP).
   * Set **Java EE version to Java EE 5 or Java EE 6**.

Click **Next**.

1. Leave default settings or add a framework if needed.

Click **Finish**.

**🔹 Step 3: Create the SOAP Web Service**

1. In the **Projects** tab, expand your project:  
   AdditionSOAPWebService
2. Right-click on the **Source Packages** → **New → Web Service**.
3. Name the service: AdditionService.
4. Choose a package name like: com.soap.

Click **Finish**.

**🔹 Step 4: Add Web Method to Perform Addition**

1. Replace the auto-generated code in AdditionService.java with:

package com.soap;

import javax.jws.WebService;

import javax.jws.WebMethod;

@WebService(serviceName = "AdditionService")

public class AdditionService {

@WebMethod(operationName = "add")

public int add(int a, int b) {

return a + b;

}

}

1. Save the file (Ctrl + S).

**🔹 Step 5: Deploy the Web Service (Optional)**

1. Right-click on the **project name** (AdditionSOAPWebService) → **Run**.
2. The browser should open with the WSDL page:
3. http://localhost:8080/AdditionSOAPWebService/AdditionService?wsdl

📌 If it doesn't open, go to **Services tab** → **Servers** → right-click on your server → Start.

**🔹 Step 6: Test the Web Service**

1. In NetBeans, go to **Projects** tab.
2. Expand the project → **Web Services** → Right-click on AdditionService → **Test Web Service**.
3. A browser window will open:
   * Enter two numbers in the input fields.
   * Click on **add**.
   * The result will appear below.

🎉 You have successfully created and tested a SOAP web service for addition!

**Aim 15. Write a Program to implement web Service to identify whether number is prime or not. Deploy the above Web service properly and call it by designing appropriate interface in JAVA / .NET**

**🔹 Step 1: Open NetBeans**

* Open **NetBeans IDE**
* Wait for it to fully load

**🔹 Step 2: Create a New Web Application Project**

1. Go to **File → New Project**
2. Choose:
   * **Java Web** → **Web Application**

Click **Next**

1. Name the project: PrimeCheckService

Click **Next**

1. Choose the server: **GlassFish Server**

Java EE version: Java EE 5 or 6  
Click **Next**

1. Click **Finish**

**🔹 Step 3: Add Web Service**

1. In **Projects** tab, right-click on **Source Packages** → **New → Web Service**
2. Name: PrimeService
3. Package: com.prime

Click **Finish**

**🔹 Step 4: Add Prime Logic in Service**

Replace the default code in PrimeService.java with:

package com.prime;

import javax.jws.WebService;

import javax.jws.WebMethod;

@WebService(serviceName = "PrimeService")

public class PrimeService {

@WebMethod(operationName = "isPrime")

public boolean isPrime(int number) {

if (number <= 1) return false;

for (int i = 2; i <= Math.sqrt(number); i++) {

if (number % i == 0) return false;

}

return true;

}

}

💾 Press **Ctrl + S** to Save

**🔹 Step 5: Deploy and Test the Service**

1. Right-click the project name PrimeCheckService → **Run**
2. Wait for browser to open this WSDL URL:
3. http://localhost:8080/PrimeCheckService/PrimeService?wsdl
4. In **NetBeans Projects** tab:
   * Expand the project
   * Expand **Web Services**
   * Right-click PrimeService → **Test Web Service**
5. A page will open in your browser:
   * Enter a number and test
   * It will show **true** or **false**

✅ Web service is working!

**Aim 16. Implement FOSS-Cloud Functionality VSI (Virtual Server Infrastructure) –**

**a) Create a Virtual Machine using available OS and load.**

**b) Create a Storage pool by allocating memory.**

**✅ Step 1: Update Your Ubuntu System**

Open your terminal and run:

sudo apt update

sudo apt upgrade -y

**✅ Step 2: Install Required Packages for FOSS-Cloud**

FOSS-Cloud requires multiple components like Apache, MySQL, PHP, and KVM. Run this command:

sudo apt install apache2 mysql-server php libapache2-mod-php \

php-mysql php-cli php-curl php-mbstring php-xml php-zip \

qemu-kvm libvirt-daemon-system libvirt-clients bridge-utils virtinst \

genisoimage unzip wget -y

**✅ Step 3: Enable KVM (Virtualization)**

Check if your system supports virtualization:

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

If the output is **0**, virtualization is not supported/enabled in BIOS.

Add your user to the KVM group:

sudo usermod -aG libvirt $(whoami)

newgrp libvirt

**✅ Step 4: Download FOSS-Cloud**

FOSS-Cloud source is on SourceForge:

cd /tmp

wget https://sourceforge.net/projects/foss-cloud/files/latest/download -O fosscloud.zip

unzip fosscloud.zip -d fosscloud

**✅ Step 5: Install and Configure FOSS-Cloud**

Now go into the extracted directory and run the install script (if available). If not, follow manual steps:

1. Move FOSS-Cloud web files to Apache directory:

sudo cp -r fosscloud/\* /var/www/html/

1. Set permissions:

sudo chown -R www-data:www-data /var/www/html/

sudo chmod -R 755 /var/www/html/

1. Restart Apache:

sudo systemctl restart apache2

**✅ Step 6: Setup MySQL for FOSS-Cloud**

Log into MySQL:

sudo mysql

Inside MySQL:

CREATE DATABASE fosscloud;

CREATE USER 'foss'@'localhost' IDENTIFIED BY 'foss123';

GRANT ALL PRIVILEGES ON fosscloud.\* TO 'foss'@'localhost';

FLUSH PRIVILEGES;

EXIT;

**✅ Step 7: Access FOSS-Cloud Web Interface**

Now open your **browser** and go to:

http://localhost

You should see the FOSS-Cloud setup page. Fill in:

* **Database Name**: fosscloud
* **User**: foss
* **Password**: foss123

Click **Install or Continue Setup**

**🎯 Complete the VSI Practical Tasks**

**🔹 (a) Create a Virtual Machine**

1. On the dashboard, go to:  
   **"Virtualization → Virtual Server Infrastructure"**
2. Click **“+ New Virtual Machine”** or **“Create”**
3. Fill:
   * Name: UbuntuVM
   * CPU: 2
   * RAM: 2048
   * Network: Default or eth0
4. Select boot media: choose ISO image (upload first if not present)

To upload:

* + Go to **"Storage → ISO Upload"**
  + Browse and upload Ubuntu ISO

1. Save → Start VM → Access Console → Install OS as usual

**🔹 (b) Create a Storage Pool**

1. Go to: **"Storage → Storage Pools"**
2. Click **"Create New Pool"**
3. Fill:
   * Name: MyStorage
   * Type: local or LVM
   * Format: qcow2
   * Size: 50 GB
4. Save.

This pool can now be used when creating VMs.

**Aim 17. Create a web application to find square of a number using Simple SOAP service.**

**✅ STEP 1: Open NetBeans**

1. Open **NetBeans IDE**.
2. Go to **File → New Project**
3. Choose:
   * **Java Web → Web Application**
4. Click **Next**

**✅ STEP 2: Name and Set the Project**

1. Project Name: SquareServiceSOAP
2. Project Location: (choose any folder)
3. Click **Next**

**✅ STEP 3: Select Server and Framework**

1. Choose your **Apache Tomcat server** from the list.  
   (If not configured, click *Add* → Select Tomcat → Give path)
2. Java EE Version: **Java EE 6 Web** or **Java EE 7 Web**
3. Click **Finish**

**✅ STEP 4: Create the SOAP Web Service**

1. In the **Projects** tab → Right-click SquareServiceSOAP project
2. Click **New → Web Service**

**Details:**

* Web Service Name: SquareService
* Package: com.square

Click **Finish**

**✅ STEP 5: Add Method to Find Square**

Now NetBeans opens your SquareService.java. Add the following method inside the class:

@WebMethod(operationName = "findSquare")

public int findSquare(@WebParam(name = "number") int number) {

return number \* number;

}

📝 **Explanation:**

* @WebMethod → Exposes the method as SOAP callable
* @WebParam → Accepts input from client

**✅ STEP 6: Deploy the Service**

1. Right-click your project (SquareServiceSOAP)
2. Click **Deploy**

Wait for it to build and deploy to Tomcat.

✅ You’ll see a message: BUILD SUCCESSFUL

**✅ STEP 7: Test the Web Service**

1. Go to **Projects tab → Web Services → SquareService**
2. Right-click → **Test Web Service**

Your browser will open a page like:

http://localhost:8080/SquareServiceSOAP/SquareService?Tester

1. In the test interface:
   * Enter a number (e.g. 4)
   * Click **findSquare**
   * Result: 16

✅ Success! You've built and tested the SOAP service.

**Aim 18. 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 of activities and workflow coordination logic and worker programs to host them.**

**🛠️ Prerequisites (Do this before starting):**

1. ✅ AWS Account (<https://aws.amazon.com/>)
2. ✅ AWS credentials configured (accessKeyId and secretAccessKey)
3. ✅ Install **Java JDK 8+**
4. ✅ Install **NetBeans** or **Eclipse IDE**
5. ✅ Install Apache Maven
6. ✅ Create IAM User with access to **SWF**

**🔹 Step-by-Step Guide to Build Hello World Workflow Using AWS Flow Framework in Java**

**🧩 Step 1: Create a New Maven Project in NetBeans**

1. Open **NetBeans**
2. Go to **File → New Project**
3. Select **Maven → Java Application → Next**
4. Name your project (e.g., HelloWorldFlow)
5. Click **Finish**

**🧩 Step 2: Add AWS Flow Framework to your pom.xml**

Paste this dependency inside <dependencies>:

<dependency>

<groupId>com.amazonaws</groupId>

<artifactId>aws-java-sdk-swf-libraries</artifactId>

<version>1.11.1000</version> <!-- use latest if available -->

</dependency>

Also add this repository if not present:

<repositories>

<repository>

<id>aws-java-sdk</id>

<url>https://repo.maven.apache.org/maven2</url>

</repository>

</repositories>

Then **right-click project > Build with Dependencies** to download the AWS libraries.

**🧩 Step 3: Create Interfaces (Contracts)**

In src/main/java, create a new interface file:

**HelloWorldWorkflow.java**

import com.amazonaws.services.simpleworkflow.flow.annotations.Workflow;

import com.amazonaws.services.simpleworkflow.flow.annotations.WorkflowMethod;

@Workflow

public interface HelloWorldWorkflow {

@WorkflowMethod

void sayHello();

}

**HelloWorldActivities.java**

import com.amazonaws.services.simpleworkflow.flow.annotations.Activity;

import com.amazonaws.services.simpleworkflow.flow.annotations.ActivityMethod;

@Activity

public interface HelloWorldActivities {

@ActivityMethod

void printHello();

}

**🧩 Step 4: Create Implementation Classes**

**HelloWorldActivitiesImpl.java**

public class HelloWorldActivitiesImpl implements HelloWorldActivities {

@Override

public void printHello() {

System.out.println("Hello World from AWS Flow Framework!");

}

}

**HelloWorldWorkflowImpl.java**

import com.amazonaws.services.simpleworkflow.flow.core.Promise;

public class HelloWorldWorkflowImpl implements HelloWorldWorkflow {

private HelloWorldActivities activities = new HelloWorldActivitiesClientImpl();

@Override

public void sayHello() {

activities.printHello(); // Workflow calls the activity

}

}

**🧩 Step 5: Register Domain and Workflow Type in AWS Console**

1. Go to AWS Console → Search for **SWF** (Simple Workflow Service)
2. Click **Create domain** → Give name like HelloWorldDomain
3. Create a new **Workflow Type** with:
   * Name: HelloWorldWorkflow
   * Version: 1.0
   * Task list: HelloTaskList

**🧩 Step 6: Create the Worker (Java Class to Host Workflow)**

**HelloWorldWorker.java**

import com.amazonaws.ClientConfiguration;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;

import com.amazonaws.regions.Regions;

import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClient;

import com.amazonaws.services.simpleworkflow.flow.WorkflowWorker;

import com.amazonaws.services.simpleworkflow.flow.ActivityWorker;

public class HelloWorldWorker {

public static void main(String[] args) throws Exception {

AmazonSimpleWorkflowClient swf = new AmazonSimpleWorkflowClient(new ProfileCredentialsProvider());

swf.setRegion(com.amazonaws.regions.Region.getRegion(Regions.US\_EAST\_1)); // Use your region

String domain = "HelloWorldDomain";

String taskList = "HelloTaskList";

// Register Activity

ActivityWorker activityWorker = new ActivityWorker(swf, domain, taskList);

activityWorker.addActivitiesImplementation(new HelloWorldActivitiesImpl());

// Register Workflow

WorkflowWorker workflowWorker = new WorkflowWorker(swf, domain, taskList);

workflowWorker.addWorkflowImplementationType(HelloWorldWorkflowImpl.class);

activityWorker.start();

workflowWorker.start();

}

}

**🧩 Step 7: Start Execution from a Client**

**HelloWorldStarter.java**

import com.amazonaws.auth.profile.ProfileCredentialsProvider;

import com.amazonaws.regions.Regions;

import com.amazonaws.services.simpleworkflow.AmazonSimpleWorkflowClient;

import com.amazonaws.services.simpleworkflow.flow.WorkflowClientFactoryExternal;

public class HelloWorldStarter {

public static void main(String[] args) {

AmazonSimpleWorkflowClient swf = new AmazonSimpleWorkflowClient(new ProfileCredentialsProvider());

swf.setRegion(com.amazonaws.regions.Region.getRegion(Regions.US\_EAST\_1));

WorkflowClientFactoryExternal factory = new WorkflowClientFactoryExternalImpl(swf, "HelloWorldDomain");

HelloWorldWorkflow workflow = factory.getClient(HelloWorldWorkflow.class);

workflow.sayHello();

}

}

**🧩 Step 8: Run the Application**

1. Run HelloWorldWorker.java (keeps listening for workflow + activity tasks)
2. Then run HelloWorldStarter.java
3. Output should show in the console:

Hello World from AWS Flow Framework!

**Aim 19. Implement FOSS-Cloud Functionality VSI (Virtual Server Infrastructure) –**

**a) Create a Virtual Machine using available OS and load.**

**b) Create at least two users in it**

**✅ Step 1: Install Virtualization Tools (One-time Setup)**

Open your terminal and run:

sudo apt update

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

Check if your system supports virtualization:

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

If output is **1 or more**, you're good!

Enable and start libvirt:

sudo systemctl enable libvirtd

sudo systemctl start libvirtd

**✅ Step 2: Launch Virtual Machine Manager**

1. Press **Super (Windows key)** > search **Virtual Machine Manager**
2. Click to launch virt-manager
3. It will ask for password – enter your Ubuntu user password
4. You'll now see **QEMU/KVM** as your local hypervisor

**✅ Step 3: Create a Virtual Machine**

1. Click **"Create a new virtual machine"** (Top-left + icon)
2. Choose:
   * **Local install media (ISO image or CDROM)** → **Forward**
3. Browse your downloaded ISO (Ubuntu or any Linux distro)
   * Click **Browse → Browse Local → Select ISO**
   * Click **Forward**
4. Memory and CPU:
   * RAM: e.g. **2048 MB**
   * CPU: **1 or 2 cores** → **Forward**
5. Disk space:
   * Allocate: e.g. **15 GB or more** → **Forward**
6. Name the VM: e.g. MyUbuntuVM
   * Enable **"Customize configuration before install"** → **Finish**

In customization screen:

* Keep everything as default or add hardware if needed
* Click **Begin Installation**

**✅ Step 4: Install the OS in the VM**

Once the VM boots from the ISO:

1. Proceed with standard **Ubuntu Installation**
2. Create a **main user** (e.g. adminuser)
3. Wait for installation to complete
4. Once done, restart the VM
5. Remove installation media if prompted

**✅ Step 5: Create Two Users Inside the VM**

Now you’re inside the running Ubuntu virtual machine.

Open Terminal in the VM (inside):

sudo adduser user1

sudo adduser user2

Enter passwords and details as prompted.

To verify:

cat /etc/passwd | grep user

You should see user1, user2, and your initial user listed.

**✅ Step 6: Test User Login (Optional)**

You can **switch user** in the VM to test:

su - user1

whoami

It should print user1

Then:

exit

su - user2

whoami

Should print user2

**Aim 20. Create a Calculator application using Simple SOAP service. Implement basic arithmetic operations viz. multiplication and division. Deploy the service and check the execution with passing proper inputs.**

**🔹 Step 1: Open NetBeans**

1. Launch **NetBeans IDE**
2. Go to: File > New Project

**🔹 Step 2: Create Web Application Project**

1. Choose:
   * **Categories**: Java Web
   * **Projects**: Web Application
   * Click **Next**
2. **Project Name**: CalculatorSOAPService
   * Click **Next**
3. **Server**: Select GlassFish Server
   * **Java EE Version**: Java EE 6 Web or Java EE 7
   * Click **Finish**

**🔹 Step 3: Create Web Service**

1. In **Projects panel** (left):
   * Right-click on **CalculatorSOAPService** > New > Web Service
2. Name: CalculatorService
   * Package: com.calculator
   * Click **Finish**

You’ll see a file: CalculatorService.java

**🔹 Step 4: Add Methods (Multiplication and Division)**

Replace the default method in CalculatorService.java with:

package com.calculator;

import javax.jws.WebService;

import javax.jws.WebMethod;

@WebService(serviceName = "CalculatorService")

public class CalculatorService {

@WebMethod(operationName = "multiply")

public double multiply(double a, double b) {

return a \* b;

}

@WebMethod(operationName = "divide")

public double divide(double a, double b) {

if (b == 0) {

throw new IllegalArgumentException("Cannot divide by zero!");

}

return a / b;

}

}

**🔹 Step 5: Test the Web Service**

1. Expand **Source Packages ->** Right Click **CalculatorService**
2. Click: **Test Web Service**
3. Enter the values

**Aim 21. Write a Program to implement web Service to Calculate factorial of a given number. Deploy the above Web service properly and call it by designing appropriate interface in JAVA / .NET**

**🔹 Step 1: Open NetBeans and Create New Project**

1. Open NetBeans
2. Click **File > New Project**
3. Choose:
   * **Java with Ant > Web Application**
4. Click **Next**
5. Project Name: FactorialWebService
6. Select a location to save the project
7. Click **Next**

**🔹 Step 2: Set Server and Settings**

1. Choose **GlassFish Server** in the dropdown
2. Java EE Version: **Java EE 7 Web** (or 6 if that's what you have)
3. Click **Finish**

**🔹 Step 3: Create a Web Service**

1. In the **Projects pane**, right-click on **"Source Packages"**
2. Choose: **New > Web Service**
3. Name it: FactorialService
4. Package: com.ws.factorial (you can name anything)
5. Click **Finish**

**🔹 Step 4: Add the Factorial Code**

Replace the default method with your own factorial method like this:

package com.ws.factorial;

import javax.jws.WebService;

import javax.jws.WebMethod;

import javax.jws.WebParam;

@WebService(serviceName = "FactorialService")

public class FactorialService {

@WebMethod(operationName = "getFactorial")

public int getFactorial(@WebParam(name = "number") int number) {

int fact = 1;

for (int i = 1; i <= number; i++) {

fact \*= i;

}

return fact;

}

}

**🔹 Step 5: Deploy and Run**

1. **Right-click on the project** > **Clean and Build**
2. Then right-click again > **Run**
3. NetBeans will:
   * Start the **GlassFish server**
   * Deploy your web service
   * Open a browser showing:  
     http://localhost:8080/FactorialWebService/FactorialService?Tester

**✅ Step 6: Test the Web Service (No Client Needed)**

1. The browser will show a page: **"Test Web Service"**
2. Under getFactorial, enter a number like 5
3. Click **Call**
4. You’ll see the result: 120 (because 5! = 120)

That’s it — your factorial web service works 🎉

**Aim 22. Create a Virtual machine using KVM and configure as given below: - a) Check KVM launching. b) Create Virtual machine and connect it to KVM. c) Install TrueConf Server.**

**✅ Step 1: Verify System Supports Virtualization**

**➤ Open Terminal (Ctrl + Alt + T)**

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

* If result ≥ 1 → KVM is supported.

To be extra sure:

sudo kvm-ok

If kvm-ok not found:

sudo apt install cpu-checker

**✅ Step 2: Install KVM and Virtual Machine Tools**

**➤ Run in Terminal:**

sudo apt update

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

**✅ Step 3: Check KVM is Running**

sudo systemctl status libvirtd

If it's not active:

sudo systemctl start libvirtd

sudo systemctl enable libvirtd

**✅ Step 4: Launch KVM GUI (Virtual Machine Manager)**

**➤ In Terminal:**

virt-manager

* The GUI will open → “Virtual Machine Manager”
* This is where you'll **create** and **manage** VMs.

**✅ Step 5: Create a Virtual Machine (GUI Steps)**

1. **Open virt-manager**
2. Click **"Create a new virtual machine"** (top-left icon)

**🖥️ In the dialog:**

* **Step 1**: Choose *Local install media (ISO)* → Click **Forward**
* **Step 2**: Browse to your **.iso file** (e.g., Ubuntu ISO)
  + You can download from: <https://ubuntu.com/download/desktop>
* **Step 3**: Set RAM (e.g., 2048 MB) and CPU (2 CPUs) → Click **Forward**
* **Step 4**: Set disk space (e.g., 15 GB) → Click **Forward**
* **Step 5**: Name it TrueConfVM → Click **Finish**

VM will boot and installation begins (like regular Ubuntu install).

**✅ Step 6: Inside VM – Install Ubuntu**

Just go through the normal Ubuntu installation process inside the virtual machine:

* Choose language, keyboard
* Set username/password
* Wait for it to install

**✅ Step 7: Download & Install TrueConf Server (Inside VM)**

**➤ After booting into the installed Ubuntu in your VM:**

**Step 1: Download TrueConf Server**

Go to: <https://trueconf.com/download/>

Or directly in terminal:

wget https://download.trueconf.com/server/linux/trueconf-server\_5.3.4-16391\_amd64.deb

(*Note: Version may change; download latest*)

**Step 2: Install the .deb package:**

sudo dpkg -i trueconf-server\_\*.deb

If errors:

sudo apt --fix-broken install

**✅ Step 8: Access TrueConf Server**

1. Open browser inside the VM.
2. Go to: https://localhost:8888 or https://<VM-IP>:8888
3. Login and configure TrueConf server (set up domain, admin password, etc.)

**✅ Bonus: Create 2 Users in TrueConf**

* Login to TrueConf Server (inside browser in VM)
* Go to **Users > Add User**
* Add 2 usernames and passwords
* Now you have 2 TrueConf accounts for testing

**Aim 23. Develop application to consume Google’s search / Google’s Map RESTful Web service.**

**🔹 Step 1: Create a New Project**

1. Open **NetBeans** (or your IDE).
2. Go to **File > New Project**.
3. Select:
   * **Categories**: Java Web
   * **Projects**: Web Application
4. Click **Next**.
5. Enter **Project Name**: googleMapREST
6. Click **Finish**.

**🔹 Step 2: Create JSP File – input.jsp**

1. **Right-click** on your project googleMapREST
2. Go to: **New > JSP**
3. Enter **File Name**: input
4. Click **Finish**

**🔹 Step 3: Code for 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 action="index.jsp">

<pre>

Enter latitude:<input type="text" name="t1"/>

Enter longitude:<input type="text" name="t2"/>

<input type="submit" value="Show"/>

</pre>

</form>

</body>

</html>

**🔹 Step 4: Get Google Maps API Key**

1. Visit: 👉 [https://console.developers.google.com](https://console.developers.google.com/)  
   (Login with your Google account)
2. Click **“Create Project”**
3. Enter project name: DemoProject
4. Go to Library Section, In the search bar, search for: "Maps JavaScript API"
5. **Enable** the Maps JavaScript API
6. On the left side, go to: **Credentials**
7. Click: **Create Credentials > API Key**
8. Copy the API Key that is generated

**🔹 Step 5: Create JSP File – index.jsp**

1. **Right-click** on the project googleMapREST
2. Go to: **New > JSP**
3. Enter **File Name**: index
4. Click **Finish**

**🔹 Step 6: Code for index.jsp**

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<style>

#map {

height: 400px;

width: 100%;

}

</style>

</head>

<body>

<%

double lati = Double.parseDouble(request.getParameter("t1"));

double longi = Double.parseDouble(request.getParameter("t2"));

%>

<h3> Google Maps </h3>

<div id="map"></div>

<script lang="javascript">

function initMap() {

var info = { lat: <%=lati%>, lng: <%=longi%> };

var map = new google.maps.Map(document.getElementById('map'), {

zoom: 4,

center: info

});

var marker = new google.maps.Marker({

position: info,

map: map

});

}

</script>

<script async defer

src="https://maps.googleapis.com/maps/api/js?key=herecopyyourAPIkey&callback=initMap">

</script>

</body>

</html>

📌 **Important:** Replace herecopyyourAPIkey with your actual Google Maps API Key.

**🔹 Step 7: Deploy and Run the Project**

1. **Right-click** on the project → **Deploy**
2. **Right-click** on input.jsp → **Run File**
3. Your browser will open with the input form.

**🔹 Step 8: Test the Application**

* Enter latitude: 19.0269
* Enter longitude: 72.8553
* Click the **“Show”** button

🎉 **Result:** Google Map with the given coordinates will appear!

**Aim 24. Develop application to download image/video from server or upload image/video to server using MTOM techniques.**

**🛠 Step-by-Step Instructions**

**✅ Step 1: Create a New Project**

1. Open **Visual Studio 2022**
2. Click on **Create a new project**
3. Select **ASP.NET Web Application (.NET Framework)** or search and select it
4. Click **Next**
5. In the next window:
   * **Name:** DownloadImageFromWS
   * **Framework:** .NET Framework 4.7.2
6. Click **Create**
7. In the next window:
   * Select **Empty**
   * Click **Create**
8. A new project will now be created

**✅ Step 2: Add Web Service**

1. On the **right side** in **Solution Explorer**:
   * Right-click on DownloadImageFromWS
   * Click **Add > New Item**
   * Search **Web Service**
   * Select **Web Service (ASMX)** – C#
   * **Name:** DownloadImageWS.asmx
   * Click **Add**

**✅ Step 3: Add Code in DownloadImageWS.asmx**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Web;

using System.Web.Services;

namespace DownloadImageFromWS

{

/// <summary>

/// Summary description for DownloadImageWS

/// </summary>

[WebService(Namespace = "http://tempuri.org/")]

[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]

[System.ComponentModel.ToolboxItem(false)]

// To allow this Web Service to be called from script, using ASP.NET AJAX,

// uncomment the following line.

// [System.Web.Script.Services.ScriptService]

public class DownloadImageWS : System.Web.Services.WebService

{

[WebMethod]

public string HelloWorld()

{

return "Hello World";

}

[WebMethod, Description("Get Image Content")]

public byte[] GetImageFile(string fileName)

{

if (System.IO.File.Exists(Server.MapPath("~/Images/") + fileName))

{

return System.IO.File.ReadAllBytes(Server.MapPath("~/Images/") + fileName);

}

else

{

return new byte[] { 0 };

}

}

}

}

**✅ Step 4: Add a Generic Handler**

1. Again, on the **right side** in **Solution Explorer**:
   * Right-click on DownloadImageFromWS
   * Click **Add > New Item**
   * Search for **Handler**
   * Select **Generic Handler** – C#
   * **Name:** MyHandler.ashx
   * Click **Add**

**✅ Step 5: Modify Code in MyHandler.ashx**

Replace the existing method with the following code and keep rest as it is:

public void ProcessRequest(HttpContext context)

{

DownloadImageWS ws = new DownloadImageWS();

byte[] binImage = ws.GetImageFile(context.Request["fileName"]);

if (binImage.Length > 1) // Ensuring valid image data is returned

{

context.Response.ContentType = "image/jpeg";

context.Response.BinaryWrite(binImage);

context.Response.End(); // Stop further execution

}

else

{

context.Response.StatusCode = 404;

context.Response.Write("Image not found.");

}

}

**✅ Step 6: Add Web Form**

1. On the **right side** in **Solution Explorer**:
   * Right-click on DownloadImageFromWS
   * Click **Add > New Item**
   * Select **Web Form** – C#
   * **Name:** DownloadImage.aspx
   * Click **Add**

**✅ Step 7: Modify DownloadImage.aspx**

Replace the code with and keep rest as it is:

<form id="form1" runat="server">

<div>

<center>

</center>

</div>

</form>

**✅ Step 8: Add Controls from Toolbox**

1. In DownloadImage.aspx, at the **bottom-left**, click on **Design**
2. From the **top menu**, click **View > Toolbox**
3. Drag and drop the following controls into the design area:
   * **Label**: Leave it as it is
   * **TextBox**: In the TextBox, type "Enter the Name of Image to Download and Show"
   * **Button**: In the Button, type "Download Image and Show"
   * **Image**: Drag an Image control as well

**✅ Step 9: Add Images Folder**

1. In **Solution Explorer**:
   * Right-click on DownloadImageFromWS
   * Click **Add > New Folder**
   * Name the folder: images
2. Open your local file explorer
3. Copy any downloaded or screenshot image
4. Paste the image inside the images folder in **Visual Studio**

**✅ Step 10: Run and Test the Project**

1. Go back to DownloadImage.aspx
2. In **Design** view, right-click anywhere
3. Select **View in Browser**
4. In the opened browser:
   * Enter the image name (e.g., image1.jpg)
   * Click the **Download Image and Show** button
   * The image will be displayed on the screen

**Aim 25. Create a Calculator application using Simple SOAP service. Implement basic arithmetic operations viz. addition and subtraction. Deploy the service and check the execution with passing proper inputs.**

**📦 Step 1: Open NetBeans and Create a New Web Application**

1. Open **NetBeans IDE**.
2. Go to **File → New Project**.
3. Select **Java Web → Web Application** → Click **Next**.
4. Give your project a name, e.g., CalculatorSOAP.
5. Choose **GlassFish Server** and Java EE version (e.g., Java EE 7).
6. Click **Finish**.

**🔧 Step 2: Create the SOAP Web Service**

1. Right-click on the **Source Packages** folder in your project.
2. Choose **New → Web Service**.
3. Give the class a name like CalculatorService.
4. Select the package (e.g., com.soap.calculator) → Click **Finish**.

**✍️ Step 3: Implement Arithmetic Operations (Addition and Subtraction)**

Replace the default content in CalculatorService.java with:

package com.soap.calculator;

import javax.jws.WebService;

import javax.jws.WebMethod;

import javax.jws.WebParam;

@WebService(serviceName = "CalculatorService")

public class CalculatorService {

@WebMethod(operationName = "add")

public int add(@WebParam(name = "a") int a, @WebParam(name = "b") int b) {

return a + b;

}

@WebMethod(operationName = "subtract")

public int subtract(@WebParam(name = "a") int a, @WebParam(name = "b") int b) {

return a - b;

}

}

✅ This defines two web methods: add and subtract.

**🚀 Step 4: Deploy the SOAP Web Service**

1. Right-click your project → Click **Deploy**.
2. Ensure the **GlassFish server is running**.
3. Wait for the build to succeed and the service to deploy.

**🔗 Step 5: Test the Web Service**

1. After deployment, go to **Projects** tab.
2. Expand your project → expand Web Services.
3. Right-click CalculatorService → Choose **Test Web Service**.
4. A browser window opens with a test interface.
5. Try:
   * Enter a = 10 and b = 5 in the **add** method → Click **add** → Should return 15.
   * Try the **subtract** method similarly → Should return 5.

🎉 You’ve now tested your SOAP web service in the browser!