# Experiment 1: Web Services Lab

**Objective**

To understand and implement basic web services using REST and SOAP protocols.

**Requirements**

- Python Program 3.x

- Flask framework

- Postman (API testing tool)

**Theory**

Brief introduction to web services, REST principles, and SOAP protocol.

**Procedure**

1. **Environment Setup**

   # Create a virtual environment

   Python Program -m venv venv

   # Activate virtual environment

   # On Windows:

   venv\Scripts\activate

   # On Linux/Mac:

   source venv/bin/activate

   # Install required packages

   pip install flask flask-restful requests

2. **REST API Implementation**

   Python Program

   # app.py

   from flask import Flask, request, jsonify

   from flask\_restful import Api, Resource

   app = Flask(\_\_name\_\_)

   api = Api(app)

   # Sample data

   books = []

   class BookResource(Resource):

       def get(self):

           return jsonify(books)

       def post(self):

           book = request.get\_json()

           books.append(book)

           return jsonify({"message": "Book added", "book": book})

   api.add\_resource(BookResource, '/books')

   if \_\_name\_\_ == '\_\_main\_\_':

       app.run(debug=True)

3. **Testing REST API**

   # Start the server

   Python Program app.py

   # In another terminal, test using curl:

   curl -X POST -H "Content-Type: application/json" -d '{"title":"Cloud Computing","author":"John Doe"}' http://localhost:5000/books

   curl http://localhost:5000/books

4. **Documentation**

   - Create API documentation using Swagger/OpenAPI

   - Document endpoints, request/response formats

   - Include example requests and responses

**Common Issues and Solutions**

1. Port already in use

   # Check port usage

   netstat -ano | findstr :5000

   # Kill process if needed

   taskkill /PID <process\_id> /F

2. CORS issues

   Python Program

   # Install Flask-CORS

   pip install flask-cors

   # Add to app.py

   from flask\_cors import CORS

   CORS(app)

**Expected Output**

- Successful REST API responses

- SOAP service functionality

- API documentation

**Verification**

1. Test REST endpoints using Postman

2. Verify SOAP responses

3. Document the results

# Experiment 2: IPC and Messaging Lab

**Objective**

To implement Inter-Process Communication and messaging patterns.

**Requirements**

- Python Program 3.x

- RabbitMQ

- ZeroMQ

**Theory**

Introduction to IPC mechanisms and publish/subscribe patterns.

**Procedure**

1. **RabbitMQ Setup**

   # Install RabbitMQ Server

   # Windows: Download and install from rabbitmq.com

   # Linux:

   sudo apt-get install rabbitmq-server

   # Start RabbitMQ

   # Windows: Start from Services

   # Linux:

   sudo systemctl start rabbitmq-server

   # Install Python Program client

   pip install pika

2. **Publisher Implementation**

   Python Program

   # publisher.py

   import pika

   connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))

   channel = connection.channel()

   # Create queue

   channel.queue\_declare(queue='hello')

   # Publish message

   channel.basic\_publish(exchange='',

                        routing\_key='hello',

                        body='Hello World!')

   print(" [x] Sent 'Hello World!'")

   connection.close()

3. **Subscriber Implementation**

   Python Program

   # subscriber.py

   import pika

   def callback(ch, method, properties, body):

       print(f" [x] Received {body}")

   connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))

   channel = connection.channel()

   channel.queue\_declare(queue='hello')

   channel.basic\_consume(queue='hello',

                        auto\_ack=True,

                        on\_message\_callback=callback)

   print(' [\*] Waiting for messages. To exit press CTRL+C')

   channel.start\_consuming()

4. **Testing**

   # Terminal 1: Start subscriber

   Python Program subscriber.py

   # Terminal 2: Run publisher

   Python Program publisher.py

**Common Issues and Solutions**

1. RabbitMQ connection issues

   # Check service status

   rabbitmqctl status

   # Reset RabbitMQ

   rabbitmqctl stop\_app

   rabbitmqctl reset

   rabbitmqctl start\_app

**Expected Output**

- Successful message publishing

- Message reception confirmation

- Queue management

# Experiment 3: Virtualization Setup Lab

**Objective**

Setup virtual machines using VirtualBox/VMware Workstation.

**Requirements**

- Windows 8 or higher

- VirtualBox/VMware Workstation

- Linux ISO image

**Procedure**

1. **VirtualBox Installation**

   a. Download VirtualBox

      - Visit virtualbox.org

      - Download latest version for Windows

      - Download Extension Pack

   b. Installation

      1. Run VirtualBox installer

      2. Accept defaults

      3. Allow network adapters installation

      4. Complete installation

      5. Install Extension Pack

         - File -> Preferences -> Extensions

         - Add Extension Pack

2. **VM Creation**

   1. Click "New"

   2. Set VM Name and OS Type

   3. Allocate Memory (4GB recommended)

   4. Create Virtual Hard Disk

      - VDI format

      - Dynamically allocated

      - 20GB size

   5. Configure Network

      - Adapter 1: NAT

      - Adapter 2: Host-only

3. **OS Installation**

   1. Start VM

   2. Select ISO file

   3. Follow installation wizard

      - Language selection

      - Partition setup

      - User creation

   4. Install Guest Additions

      - Devices -> Insert Guest Additions CD

      - Run VBoxLinuxAdditions.run

**Common Issues and Solutions**

1. VT-x/AMD-v not enabled

   - Enter BIOS

   - Enable Virtualization Technology

2. Network issues

   # Check network adapters

   ipconfig /all

   # Reset network

   sudo systemctl restart NetworkManager

**Expected Output**

- VM boots properly

- Network connectivity

- Shared folders working

# Experiment 4: Virtual Machine Programming Lab

**Objective**

Setup development environment in VM and execute programs.

**Requirements**

- Virtual machine from Exp 3

- GCC compiler

- Text editor

**Procedure**

1. **Development Environment Setup**

   # Update package list

   sudo apt update

   # Install build tools

   sudo apt install build-essential gdb

   # Install text editor

   sudo apt install vim

2. **Create Test Programs**

   a. Basic I/O Program

   c

   // io.c

   #include <stdio.h>

   int main() {

       int num;

       printf("Enter a number: ");

       scanf("%d", &num);

       printf("You entered: %d\n", num);

       return 0;

   }

   b. File Operations

   c

   // file.c

   #include <stdio.h>

   int main() {

       FILE \*fp;

       fp = fopen("test.txt", "w");

       fprintf(fp, "Testing file operations\n");

       fclose(fp);

       return 0;

   }

3. **Compilation and Debugging**

   # Compile with debug info

   gcc -g io.c -o io

   # Run program

   ./io

   # Debug with GDB

   gdb ./io

**Common Issues and Solutions**

1. Compilation errors

   # Check compiler version

   gcc --version

   # Install specific version if needed

   sudo apt install gcc-<version>

**Expected Output**

- Successful compilation and execution

- Debugging functionality

# Experiment 5: AWS EC2 Setup Lab

**Objective**

Create and configure AWS EC2 instance with web server.

**Requirements**

- AWS account

- SSH client

- Web browser

**Procedure**

1. **AWS Account Setup**

   1. Visit aws.amazon.com

   2. Create free tier account

   3. Set up billing alerts

   4. Generate access keys

2. **EC2 Instance Launch**

   1. Open EC2 Dashboard

   2. Click "Launch Instance"

   3. Select Amazon Linux 2

   4. Choose t2.micro

   5. Configure Instance

      - Network: default VPC

      - Auto-assign Public IP: Enable

   6. Add Storage: 8GB

   7. Configure Security Group

   8. Create/Select Key Pair

3. **Web Server Configuration**

   # Connect to instance

   ssh -i key.pem ec2-user@<public-ip>

   # Update system

   sudo yum update -y

   # Install Apache

   sudo yum install httpd -y

   # Start service

   sudo systemctl start httpd

   sudo systemctl enable httpd

   # Create web page

   echo "<h1>Hello from AWS</h1>" | sudo tee /var/www/html/index.html

**Common Issues and Solutions**

1. SSH connection issues

   # Check key permissions

   chmod 400 key.pem

   # Verify security group rules

2. Web server issues

   # Check service status

   sudo systemctl status httpd

   # Check logs

   sudo tail -f /var/log/httpd/error\_log

**Expected Output**

- Web server accessible

- Security group working

- SSH access confirmed

# Experiment 6: OpenStack Setup Lab

**Objective**

Setup and configure OpenStack environment.

**Requirements**

- DevStack

- Ubuntu Server

- 8GB RAM minimum

**Procedure**

1. **DevStack Installation**

   sudo useradd -s /bin/ -d /opt/stack -m stack

   sudo chmod +x /opt/stack

   git clone https://opendev.org/openstack/devstack

2. **Configuration**

   - Edit local.conf

   - Setup network

   - Initialize services

3. **Instance Launch**

   - Create flavor

   - Upload image

   - Launch instance

**Expected Output**

- Successful OpenStack setup

- Instance creation

# Experiment 7: Google App Engine Lab

**Objectives**

- Set up Google Cloud SDK and App Engine environment

- Create and deploy a simple Python web application

- Understand App Engine's scaling and versioning

- Implement Google Cloud storage integration

**Requirements**

- Google Cloud account

- Python 3.7 or higher

- Google Cloud SDK

- Text editor or IDE

- Active billing account (free tier available)

**Theory**

Google App Engine (GAE) is a Platform as a Service (PaaS) offering that enables developers to build and deploy applications on Google's infrastructure. It automatically handles scaling, load balancing, and application instance management.

**Step-by-Step Process**

1. **Install Google Cloud SDK**

   # Download and install Google Cloud SDK

   # For Windows:

   # Visit https://cloud.google.com/sdk/docs/install

   # Verify installation

   gcloud --version

2. **Initialize Google Cloud SDK**

   # Initialize SDK and set project

   gcloud init

   # Authenticate your account

   gcloud auth login

3. **Create a New Project**

   # Create new project

   gcloud projects create [PROJECT\_ID] --name=[PROJECT\_NAME]

   # Set project as active

   gcloud config set project [PROJECT\_ID]

4. **Create Python Application**

   Python Program

   # main.py

   from flask import Flask

   app = Flask(\_\_name\_\_)

   @app.route('/')

   def hello():

       return 'Hello, Google App Engine!'

   if \_\_name\_\_ == '\_\_main\_\_':

       app.run(host='127.0.0.1', port=8080)

5. **Create App Configuration**

   yaml

   # app.yaml

   runtime: python39

   entrypoint: gunicorn -b :$PORT main:app

   instance\_class: F1

   automatic\_scaling:

     target\_cpu\_utilization: 0.65

     min\_instances: 1

     max\_instances: 10

6. **Set Up Dependencies**

   # Create requirements.txt

   echo "Flask==2.0.1\ngunicorn==20.1.0" > requirements.txt

   # Install dependencies

   pip install -r requirements.txt

7. **Test Locally**

   # Run development server

   python main.py

   # Access http://localhost:8080

8. **Deploy to App Engine**

   # Deploy application

   gcloud app deploy app.yaml

   # Stream logs

   gcloud app logs tail

9. **Manage Versions**

   # List versions

   gcloud app versions list

   # Switch traffic

   gcloud app services set-traffic [SERVICE] --splits [VERSION]=1

**Common Issues and Solutions**

1. **Deployment Failures**

   # Check deployment status

   gcloud app deploy --verbosity=debug

   # Verify billing is enabled

   gcloud billing projects describe [PROJECT\_ID]

2. **Permission Issues**

   # Verify permissions

   gcloud auth list

   # Set correct project

   gcloud config set project [PROJECT\_ID]

3. **Resource Limits**

   - Check quotas in Google Cloud Console

   - Request quota increase if needed

   - Monitor resource usage

**Verification Steps**

1. Access deployed application URL

2. Verify application scaling under load

3. Check logs for errors

4. Test version management

5. Monitor resource usage

**Expected Output**

- Successful deployment confirmation

- Application accessible via [PROJECT\_ID].appspot.com

- Logs showing request handling

- Automatic scaling in action

**Additional Tasks**

1. Implement Cloud Storage

2. Add Cloud SQL integration

3. Set up custom domain

4. Configure HTTPS

5. Implement monitoring

# Experiment 8: Docker Container Lab

**Objectives**

- Install and configure Docker environment

- Create and manage Docker containers

- Build custom Docker images

- Implement container networking

- Use Docker Compose for multi-container applications

**Requirements**

- Windows 10/11 Pro or Enterprise (for Docker Desktop)

- Docker Desktop or Docker Engine

- Text editor or IDE

- Internet connection

- Git (optional)

**Theory**

Docker is a platform for developing, shipping, and running applications in containers. Containers are lightweight, portable, and self-sufficient units that can run anywhere Docker is installed.

**Step-by-Step Process**

1.  **Install Docker**

   # For Windows:

   # Download Docker Desktop from https://www.docker.com/products/docker-desktop

   # Verify installation

   docker --version

   docker-compose --version

2.  **Basic Docker Commands**

   # List images

   docker images

   # List containers

   docker ps -a

   # Pull an image

   docker pull nginx:latest

3.  **Run Your First Container**

   # Run nginx container

   docker run -d -p 80:80 --name my-nginx nginx

   # View container logs

   docker logs my-nginx

   # Stop container

   docker stop my-nginx

4.  **Create Custom Dockerfile**

    dockerfile

   # Dockerfile

   FROM python:3.9-slim

   WORKDIR /app

   COPY requirements.txt .

   RUN pip install -r requirements.txt

   COPY . .

   EXPOSE 5000

   CMD ["python", "app.py"]

5.  **Build and Run Custom Image**

   # Build image

   docker build -t myapp:1.0 .

   # Run container

   docker run -d -p 5000:5000 --name myapp myapp:1.0

6.  **Docker Networking**

   # Create network

   docker network create mynetwork

   # List networks

   docker network ls

   # Run container in network

   docker run -d --network mynetwork --name db mysql:5.7

7.  **Docker Compose Setup**

    yaml

   # docker-compose.yml

   version: '3.8'

   services:

     web:

       build: .

       ports:

         - "5000:5000"

       depends\_on:

         - db

     db:

       image: mysql:5.7

       environment:

         MYSQL\_ROOT\_PASSWORD: example

       volumes:

         - db\_data:/var/lib/mysql

   volumes:

     db\_data:

8.  **Use Docker Compose**

   # Start services

   docker-compose up -d

   # View logs

   docker-compose logs

   # Stop services

   docker-compose down

9.  **Container Management**

   # Container stats

   docker stats

   # Execute command in container

   docker exec -it myapp

   # Copy files to/from container

   docker cp file.txt myapp:/app/

**Common Issues and Solutions**

1.  **Permission Issues**

   # Add user to docker group

   sudo usermod -aG docker $USER

   # Restart Docker service

   sudo service docker restart

2.  **Container Connectivity**

   # Check container network

   docker network inspect mynetwork

   # Restart container networking

   docker-compose restart

3.  **Resource Constraints**

   # Set container memory limit

   docker run -d -m 512m myapp:1.0

   # Set CPU limit

   docker run -d --cpus=".5" myapp:1.0

**Verification Steps**

1. Check container status

2. Verify application accessibility

3. Test container networking

4. Validate volume persistence

5. Monitor resource usage

**Expected Output**

- Running containers list

- Container logs

- Network connectivity between containers

- Persistent data in volumes

- Resource usage statistics

**Additional Tasks**

1. Implement container orchestration

2. Set up container monitoring

3. Configure container security

4. Implement CI/CD pipeline

5. Deploy to container registry

# Experiment 9: VM and Container File Transfer Lab

**Objective:** Learn and implement various methods of file transfer between virtual machines and containers, understanding the security implications and best practices for data movement in virtualized environments.

**Theory**

- Understanding different file transfer protocols (SCP, SFTP, rsync)

- Docker volume management and data persistence

- Network configurations for secure file transfer

- Container data management best practices

**Requirements**

- Two VMs/containers

- SSH/SCP tools

- Docker volumes

- rsync (optional)

- FileZilla or WinSCP (for GUI-based transfers)

**Detailed Procedure**

1.  **VM to VM Transfer Using SCP**

   # Basic file transfer

   scp file.txt user@vm2:/destination/

   # Transfer directory with compression

   scp -r -C /source/directory user@vm2:/destination/

   # Transfer with specific port

   scp -P 2222 file.txt user@vm2:/destination/

   # Transfer multiple files

   scp file1.txt file2.txt user@vm2:/destination/

2.  **Using rsync for Efficient Transfer**

   # Basic rsync

   rsync -av /source/directory/ user@vm2:/destination/

   # Resume interrupted transfer

   rsync -av --partial --progress /source/directory/ user@vm2:/destination/

   # Sync with deletion

   rsync -av --delete /source/directory/ user@vm2:/destination/

3.  **Container to Container Transfer**

   # Using docker cp

   docker cp file.txt container2:/path/

   # Copy from container to host

   docker cp container1:/path/file.txt ./

   # Copy entire directory

   docker cp container1:/app/. /host/path/

4.  **Using Docker Volumes**

   # Create a named volume

   docker volume create shared-data

   # Mount volume to containers

   docker run -v shared-data:/data container1

   docker run -v shared-data:/data container2

   # List volumes

   docker volume ls

   # Inspect volume

   docker volume inspect shared-data

5.  **Setting Up Shared Storage**

   # Create a bind mount

   docker run -v /host/path:/container/path container1

   # Use tmpfs mount

   docker run --tmpfs /app/temp container1

   # Create volume with specific driver

   docker volume create --driver local \

     --opt type=nfs \

     --opt o=addr=192.168.1.1,rw \

     --opt device=:/path/to/dir \

     nfs-volume

**Common Issues and Solutions**

1.  **Permission Issues**

   # Fix permissions on source

   chmod 644 file.txt

   # Fix ownership

   chown -R user:group /path/to/files

2.  **Network Connectivity**

   # Test connectivity

   ping vm2

   # Check SSH service

   systemctl status ssh

   # Configure firewall

   ufw allow ssh

3.  **Volume Mounting Issues**

   # Clean up unused volumes

   docker volume prune

   # Fix SELinux context

   chcon -Rt svirt\_sandbox\_file\_t /host/path

**Expected Output**

- Successful file transfers between VMs

- Successful container data sharing

- Proper volume management

- Secure and efficient data movement

**Verification Steps**

1. Check file integrity

   # Compare checksums

   md5sum file.txt

   sha256sum file.txt

2. Verify permissions

   ls -l /destination/file.txt

3. Test volume access

   docker exec container1 ls /data

# Experiment 10: TryStack OpenStack Lab

**Objective**

Learn to use TryStack (OpenStack's online demo environment) to understand OpenStack operations and management without local installation overhead.

**Theory**

- Understanding OpenStack components

- Cloud resource management

- Software-defined networking

- Virtual machine lifecycle management

**Requirements**

- TryStack account

- Web browser

- SSH client

- OpenStack CLI (optional)

**Detailed Procedure**

1.  **Account Setup and Access**

   # Install OpenStack CLI

   pip install python-openstackclient

   # Configure environment variables

   export OS\_AUTH\_URL=https://trystack.openstack.org:5000/v3

   export OS\_PROJECT\_NAME="demo"

   export OS\_USERNAME="username"

   export OS\_PASSWORD="password"

2.  **Network Configuration**

   # Create network

   openstack network create private-net

   # Create subnet

   openstack subnet create private-subnet \

     --network private-net \

     --subnet-range 192.168.1.0/24

   # Create router

   openstack router create main-router

   # Add subnet to router

   openstack router add subnet main-router private-subnet

3.  **Security Group Management**

   # Create security group

   openstack security group create web-server

   # Add rules

   openstack security group rule create \

     --protocol tcp \

     --dst-port 80 \

     --remote-ip 0.0.0.0/0 \

     web-server

4.  **Instance Management**

   # Create keypair

   openstack keypair create --public-key ~/.ssh/id\_rsa.pub mykey

   # Launch instance

   openstack server create \

     --image ubuntu-20.04 \

     --flavor m1.small \

     --network private-net \

     --security-group web-server \

     --key-name mykey \

     web-server

5.  **Volume Operations**

   # Create volume

   openstack volume create \

     --size 10 \

     data-volume

   # Attach volume

   openstack server add volume \

     web-server \

     data-volume

**Common Issues and Solutions**

1.  **Authentication Issues**

   # Verify credentials

   openstack token issue

   # Update expired credentials

   source openrc.sh

2.  **Network Connectivity**

   # Check network status

   openstack network list

   # Verify router status

   openstack router show main-router

**Expected Output**

- Functioning OpenStack environment

- Successfully deployed instances

- Proper network configuration

- Working storage volumes

# Experiment 11: Hadoop Setup Lab

**Objective**

Set up and configure a single-node Hadoop cluster and understand the basics of MapReduce programming through practical implementation.

**Theory**

- Hadoop architecture

- HDFS (Hadoop Distributed File System)

- MapReduce programming model

- YARN resource management

**Requirements**

- Java JDK 8 or later

- Hadoop distribution

- Linux system

- Minimum 8GB RAM

- 50GB free disk space

**Detailed Procedure**

1.  **Environment Setup**

   # Install Java

   sudo apt update

   sudo apt install openjdk-8-jdk

   # Set JAVA\_HOME

   echo 'export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64' >> ~/. rc

   source ~/. rc

   # Download and extract Hadoop

   wget https://downloads.apache.org/hadoop/common/hadoop-3.3.4/hadoop-3.3.4.tar.gz

   tar -xzf hadoop-3.3.4.tar.gz

   mv hadoop-3.3.4 /usr/local/hadoop

2.  **Hadoop Configuration**

    xml

   <!-- core-site.xml -->

   <configuration>

     <property>

       <name>fs.defaultFS</name>

       <value>hdfs://localhost:9000</value>

     </property>

   </configuration>

   <!-- hdfs-site.xml -->

   <configuration>

     <property>

       <name>dfs.replication</name>

       <value>1</value>

     </property>

   </configuration>

   <!-- mapred-site.xml -->

   <configuration>

     <property>

       <name>mapreduce.framework.name</name>

       <value>yarn</value>

     </property>

   </configuration>

3.  **HDFS Initialization**

   # Format namenode

   hdfs namenode -format

   # Start HDFS

   start-dfs.sh

   # Create HDFS directories

   hdfs dfs -mkdir /user

   hdfs dfs -mkdir /user/$USER

4.  **WordCount Implementation**

    java

   import org.apache.hadoop.conf.Configuration;

   import org.apache.hadoop.fs.Path;

   import org.apache.hadoop.io.IntWritable;

   import org.apache.hadoop.io.Text;

   import org.apache.hadoop.mapreduce.Job;

   import org.apache.hadoop.mapreduce.Mapper;

   import org.apache.hadoop.mapreduce.Reducer;

   public class WordCount {

       public static class TokenizerMapper

           extends Mapper<Object, Text, Text, IntWritable> {

           private final static IntWritable one = new IntWritable(1);

           private Text word = new Text();

           public void map(Object key, Text value, Context context) {

               StringTokenizer itr = new StringTokenizer(value.toString());

               while (itr.hasMoreTokens()) {

                   word.set(itr.nextToken());

                   context.write(word, one);

               }

           }

       }

       public static class IntSumReducer

           extends Reducer<Text,IntWritable,Text,IntWritable> {

           public void reduce(Text key, Iterable<IntWritable> values,

                            Context context) {

               int sum = 0;

               for (IntWritable val : values) {

                   sum += val.get();

               }

               context.write(key, new IntWritable(sum));

           }

       }

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

           Configuration conf = new Configuration();

           Job job = Job.getInstance(conf, "word count");

           job.setJarByClass(WordCount.class);

           job.setMapperClass(TokenizerMapper.class);

           job.setCombinerClass(IntSumReducer.class);

           job.setReducerClass(IntSumReducer.class);

           job.setOutputKeyClass(Text.class);

           job.setOutputValueClass(IntWritable.class);

           FileInputFormat.addInputPath(job, new Path(args[0]));

           FileOutputFormat.setOutputPath(job, new Path(args[1]));

           System.exit(job.waitForCompletion(true) ? 0 : 1);

       }

   }

5.  **Running MapReduce Job**

   # Compile

   hadoop com.sun.tools.javac.Main WordCount.java

   jar cf wc.jar WordCount\*.class

   # Create input

   echo "Hello World Bye World" > input.txt

   hdfs dfs -put input.txt /user/$USER/input

   # Run job

   hadoop jar wc.jar WordCount input output

   # View results

   hdfs dfs -cat /user/$USER/output/part-r-00000

**Common Issues and Solutions**

1.  **Java Version Issues**

   # Check Java version

   java -version

   # Update alternatives

   sudo update-alternatives --config java

2.  **Permission Problems**

   # Fix HDFS permissions

   hdfs dfs -chmod -R 755 /user/$USER

   # Fix local permissions

   chmod +x $HADOOP\_HOME/sbin/\*

**Expected Output**

- Functioning Hadoop cluster

- Successful MapReduce job execution

- Correct word count results

- Proper resource management

# Experiment 12: OpenFaaS Serverless Lab

**Objective**

Implement and deploy serverless functions using OpenFaaS, understanding the serverless architecture and its benefits.

**Theory**

- Serverless computing concepts

- Function as a Service (FaaS)

- Event-driven architecture

- Container orchestration

**Requirements**

- Docker

- Kubernetes/K3s

- OpenFaaS CLI

- kubectl

- helm (optional)

**Detailed Procedure**

1.  **OpenFaaS Setup**

   # Install OpenFaaS CLI

   curl -sL https://cli.openfaas.com | sudo sh

   # Deploy OpenFaaS

   git clone https://github.com/openfaas/faasd

   cd faasd

   # Install faasd

   ./hack/install.sh

   # Get the gateway URL

   echo http://127.0.0.1:8080

2.  **Function Development**

    python

   # handler.py

   def handle(req):

       """

       Handle a serverless request

       Args:

           req (str): request body

       """

       return f"Hello! You said: {req}"

   # requirements.txt

   requests==2.26.0

3.  **Function Deployment**

   # Create new function

   faas-cli new --lang python3 hello-function

   # Build function

   faas-cli build -f hello-function.yml

   # Deploy function

   faas-cli deploy -f hello-function.yml

4.  **Testing Functions**

   # Invoke function

   echo "OpenFaaS" | faas-cli invoke hello-function

   # Get function info

   faas-cli describe hello-function

   # List functions

   faas-cli list

5.  **Advanced Features**

    yaml

   # hello-function.yml

   version: 1.0

   provider:

     name: openfaas

     gateway: http://127.0.0.1:8080

   functions:

     hello-function:

       lang: python3

       handler: ./hello-function

       image: hello-function:latest

       environment:

         write\_debug: true

       labels:

         com.openfaas.scale.min: "1"

         com.openfaas.scale.max: "10"

       annotations:

         topic: "async-function"

**Common Issues and Solutions**

1.  **Gateway Connection Issues**

   # Check gateway status

   curl http://127.0.0.1:8080/system/info

   # Login to gateway

   echo $OPENFAAS\_PASSWORD | faas-cli login --password-stdin

2.  **Build Problems**

   # Clean build cache

   faas-cli build --clean

   # Check Docker daemon

   docker info

**Expected Output**

- Successfully deployed functions

- Proper function invocation

- Auto-scaling capabilities

- Event-driven execution

# Experiment 13: CloudSim Simulation Lab

**Objective**

Simulate and analyze cloud computing environments using CloudSim, implementing custom scheduling algorithms and understanding resource management.

**Theory**

- Cloud simulation concepts

- Resource provisioning

- Scheduling algorithms

- Performance metrics

**Requirements**

- Java JDK 8 or later

- CloudSim package

- Eclipse IDE

- Maven (optional)

**Detailed Procedure**

1.  **CloudSim Setup**

   # Clone CloudSim

   git clone https://github.com/Cloudslab/cloudsim.git

   # Import into Eclipse

   1. File -> Import -> Maven -> Existing Maven Project

   2. Select cloudsim directory

   3. Build project

2.  **Custom Scheduler Implementation**

    java

   import org.cloudbus.cloudsim.\*;

   public class CustomScheduler extends CloudletScheduler {

       private List<ResCloudlet> cloudletExecList;

       private List<ResCloudlet> cloudletWaitingList;

       public CustomScheduler() {

           cloudletExecList = new ArrayList<ResCloudlet>();

           cloudletWaitingList = new ArrayList<ResCloudlet>();

       }

       @Override

       public double updateVmProcessing(double currentTime, List<Double> mipsShare) {

           // Implementation of scheduling logic

           setCurrentMipsShare(mipsShare);

           double timeSpam = currentTime - getPreviousTime();

           // Update cloudlet execution time

           double capacity = 0.0;

           int cpus = 0;

           for (Double mips : mipsShare) {

               capacity += mips;

               if (mips > 0) {

                   cpus++;

               }

           }

           // Each cloudlet runs at the same capacity

           capacity /= cpus;

           // Update cloudlets

           for (ResCloudlet rcl : cloudletExecList) {

               rcl.updateCloudletFinishedSoFar(capacity \* timeSpam);

           }

           // Check finished cloudlets

           List<ResCloudlet> finishedCloudlets = new ArrayList<ResCloudlet>();

           for (ResCloudlet rcl : cloudletExecList) {

               if (rcl.getRemainingCloudletLength() == 0) {

                   finishedCloudlets.add(rcl);

               }

           }

           // Remove finished cloudlets

           cloudletExecList.removeAll(finishedCloudlets);

           // Add waiting cloudlets if possible

           if (!cloudletWaitingList.isEmpty()) {

               for (ResCloudlet rcl : cloudletWaitingList) {

                   if (cloudletExecList.size() < cpus) {

                       cloudletExecList.add(rcl);

                   }

               }

           }

           setPreviousTime(currentTime);

           return 0.0;

       }

   }

3.  **Simulation Setup**

    java

   import org.cloudbus.cloudsim.\*;

   import java.util.\*;

   public class CloudSimulation {

       private static List<Cloudlet> cloudletList;

       private static List<Vm> vmList;

       public static void main(String[] args) {

           // Initialize

           int num\_user = 1;

           Calendar calendar = Calendar.getInstance();

           boolean trace\_flag = false;

           CloudSim.init(num\_user, calendar, trace\_flag);

           // Create Datacenter

           Datacenter datacenter0 = createDatacenter("Datacenter\_0");

           // Create Broker

           DatacenterBroker broker = createBroker();

           int brokerId = broker.getId();

           // Create VMs

           vmList = new ArrayList<Vm>();

           int vmid = 0;

           int mips = 1000;

           long size = 10000;

           int ram = 512;

           long bw = 1000;

           int pesNumber = 1;

           String vmm = "Xen";

           Vm vm = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new CustomScheduler());

           vmList.add(vm);

           // Submit VM list to broker

           broker.submitVmList(vmList);

           // Create Cloudlets

           cloudletList = new ArrayList<Cloudlet>();

           int id = 0;

           long length = 400000;

           long fileSize = 300;

           long outputSize = 300;

           UtilizationModel utilizationModel = new UtilizationModelFull();

           Cloudlet cloudlet = new Cloudlet(id, length, pesNumber, fileSize, outputSize,

                   utilizationModel, utilizationModel, utilizationModel);

           cloudlet.setUserId(brokerId);

           cloudletList.add(cloudlet);

           // Submit cloudlets to broker

           broker.submitCloudletList(cloudletList);

           // Start simulation

           CloudSim.startSimulation();

           // Stop simulation

           CloudSim.stopSimulation();

           // Print results

           List<Cloudlet> newList = broker.getCloudletReceivedList();

           printCloudletList(newList);

       }

       private static void printCloudletList(List<Cloudlet> list) {

           int size = list.size();

           Cloudlet cloudlet;

           String indent = "    ";

           Log.printLine();

           Log.printLine("========== OUTPUT ==========");

           Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +

                   "Data center ID" + indent + "VM ID" + indent + "Time" + indent +

                   "Start Time" + indent + "Finish Time");

           DecimalFormat dft = new DecimalFormat(" .##");

           for (int i = 0; i < size; i++) {

               cloudlet = list.get(i);

               Log.print(indent + cloudlet.getCloudletId() + indent + indent);

               if (cloudlet.getStatus() == Cloudlet.SUCCESS) {

                   Log.print("SUCCESS");

                   Log.printLine(indent + indent + cloudlet.getResourceId() +

                           indent + indent + indent + cloudlet.getVmId() +

                           indent + indent + dft.format(cloudlet.getActualCPUTime()) +

                           indent + indent + dft.format(cloudlet.getExecStartTime()) +

                           indent + indent + dft.format(cloudlet.getFinishTime()));

               }

           }

       }

   }

4.  **Running and Analysis**

   # Compile

   javac -cp cloudsim-4.0.jar:commons-math3-3.6.1.jar CloudSimulation.java

   # Run

   java -cp .:cloudsim-4.0.jar:commons-math3-3.6.1.jar CloudSimulation

   # Analyze results

   - Check execution times

   - Verify resource utilization

   - Compare scheduling efficiency

**Common Issues and Solutions**

1.  **Compilation Issues**

   # Check classpath

   echo $CLASSPATH

   # Add dependencies

   export CLASSPATH=$CLASSPATH:path/to/cloudsim-4.0.jar

2.  **Memory Problems**

   # Increase heap size

   java -Xmx2g -cp ... CloudSimulation

**Expected Output**

- Detailed simulation results

- Resource utilization statistics

- Performance metrics

- Scheduling analysis

**Performance Metrics**

1.  **Resource Utilization**

   - CPU usage

   - Memory consumption

   - Bandwidth utilization

2.  **Time Metrics**

   - Task completion time

   - Resource allocation time

   - Queue waiting time

3.  **Cost Analysis**

   - Resource cost

   - Operation cost

   - Total simulation cost