**SRM Institute of Science and Technology**

**Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204**

**Department of Computer Applications**

**Circular – 2023-24**

**B.Sc. CS 5th Sem**

# Cloud Computing (UCS23D02J) Lab Manual

This manual provides a structured guide for practical experiments in Cloud Computing, covering various aspects from virtual machine management to application deployment and cloud service integration. Each lab is designed to provide hands-on experience with fundamental cloud concepts and technologies.

## Lab 1: Create a Virtual Machine

- **Title:** Creating a Virtual Machine on a Cloud Platform
- **Aim:** To understand the process of provisioning and configuring a virtual machine instance on a cloud provider's infrastructure.
- **Procedure:**
  1. **Choose a Cloud Provider:** Select a cloud provider (e.g., AWS, Google Cloud, Azure). For this lab, we'll assume a generic cloud console.
  2. **Navigate to VM Creation:** Log in to the cloud provider's console and navigate to the Virtual Machines or Compute Engine section.
  3. **Start Instance Creation:** Click on "Create Instance," "Launch Instance," or similar.
  4. **Select Machine Image (OS):** Choose an operating system image (e.g., Ubuntu, CentOS, Windows Server).
  5. **Choose Instance Type:** Select an instance type that defines the CPU, memory, and network performance (e.g., `t2.micro` on AWS, `e2-micro` on Google Cloud).
  6. **Configure Network Settings:**
     - Select a Virtual Private Cloud (VPC) or network.
     - Configure subnet and assign a public IP address if external access is required.
     - Set up security groups or firewall rules to allow necessary inbound traffic (e.g., SSH for Linux, RDP for Windows, HTTP/HTTPS for web servers).
  7. **Add Storage:** Specify the size and type of the boot disk and add any additional data disks if needed.
  8. **Review and Launch:** Review all the configurations. If everything is correct, launch the instance.
  9. **Connect to VM:** Once the VM is running, connect to it using SSH (for Linux) or RDP (for Windows) with the generated key pair or credentials.
- **Source Code:** N/A (This lab primarily involves graphical user interface (GUI) interactions or command-line interface (CLI) commands for provisioning).
- **Input:**
  1. Cloud Provider Account Credentials
  2. Desired OS Image (e.g., Ubuntu 20.04 LTS)
  3. Instance Type (e.g., 1 vCPU, 2GB RAM)
  4. Disk Size (e.g., 20 GB)
  5. SSH Key Pair (for Linux) or Administrator Password (for Windows)
  6. Inbound Firewall Rules (e.g., Port 22 for SSH, Port 80 for HTTP)

- **Expected Output:**
    1. A running virtual machine instance accessible via SSH/RDP.
    2. The instance's public IP address and DNS name.
    3. Ability to log in and interact with the VM's operating system.

## Lab 2: Installation of Platforms

- **Title:** Installation of Cloud Development Platforms and SDKs
- **Aim:** To set up the necessary software development kits (SDKs) and command-line interfaces (CLIs) for interacting with cloud services from a local machine.
- **Procedure:**
    1. **Identify Required SDKs/CLIs:** Determine which cloud provider's SDK/CLI is needed (e.g., Google Cloud SDK, AWS CLI, Azure CLI).
    2. **Download Installer:** Visit the official documentation page of the chosen cloud provider and download the appropriate installer for your operating system (Windows, macOS, Linux).
    3. **Run Installation:**
        - **Windows:** Execute the downloaded `.exe` installer and follow the on-screen prompts.
        - **macOS/Linux:** Use package managers (e.g., `brew` for macOS, `apt` or `yum` for Linux) or run the provided installation scripts.
    4. **Initialize/Configure CLI:** After installation, open a new terminal or command prompt and initialize the CLI. This usually involves running a command like `gcloud init` (Google Cloud), `aws configure` (AWS), or `az login` (Azure) to authenticate your local environment with your cloud account.
    5. **Verify Installation:** Run a simple command to verify the installation and authentication (e.g., `gcloud compute instances list`, `aws s3 ls`, `az account show`).
- **Source Code:** N/A (This lab involves system-level installations and configurations).
- **Input:**
    1. Operating System Type (e.g., Windows 10, macOS, Ubuntu)
    2. Cloud Provider Credentials (for CLI configuration)
- **Expected Output:**
    1. Successfully installed cloud SDK/CLI on the local machine.
    2. Ability to authenticate with the cloud provider from the command line.
    3. Successful execution of basic cloud resource listing commands.

## Lab 3: Deploying Existing Apps

- **Title:** Deploying an Existing Application to a Cloud Platform
- **Aim:** To learn how to package and deploy a pre-built application to a cloud-based application hosting service.
- **Procedure:**
    1. **Choose** a **Cloud Hosting Service:** Select a suitable cloud service for application deployment (e.g., Google App Engine, AWS Elastic Beanstalk, Azure App Service, Heroku).
    2. **Prepare Application:** Ensure the existing application is ready for deployment. This might involve:
        - Adding a `Procfile` (Heroku) or `app.yaml` (Google App Engine) for configuration.
        - Ensuring all dependencies are listed (e.g., `requirements.txt` for Python, `package.json` for Node.js).
        - Configuring database connections or environment variables.
    3. **Authenticate CLI:** Log in to your cloud provider account via the command-line interface (as done in Lab 2).
    4. **Navigate to Application Directory:** Change your current directory in the terminal to the root of your application's source code.
    5. **Execute Deployment Command:** Use the cloud provider's specific deployment command:
        - **Google App Engine:** `gcloud app deploy`
        - **AWS Elastic Beanstalk:** `eb deploy`
        - **Azure App Service:** `az webapp up` (for quick deployment) or `git` push azure `master` (for Git deployment).
        - **Heroku:** `git push heroku master`
    6. **Monitor Deployment:** Observe the output in the terminal for deployment progress and any errors.
    7. **Access Deployed App:** Once deployment is complete, the cloud service will provide a URL to access your application.
- **Source Code:** N/A (This lab uses an *existing* application's source code. The deployment process is tool-driven).
- **Input:**
    1. An existing web application (e.g., a simple Python Flask app, Node.js Express app, Java Spring Boot app).
    2. Configuration files specific to the deployment platform (e.g., `app.yaml`, `Procfile`).
- **Expected Output:**
    1. The application successfully deployed to the cloud.
    2. A publicly accessible URL for the deployed application.
    3. The application functioning as expected when accessed via its URL.

## Lab 4: Create a Dropbox using Google APIs

- **Title:** Implementing a Simple File Storage System (Dropbox-like) using Google Drive API
- **Aim:** To learn how to interact with Google Drive programmatically to upload, list, and manage files, simulating basic Dropbox functionality.
- **Procedure:**
  1. **Google Cloud Project Setup:**
     - Create a new Google Cloud Project.
     - Enable the Google Drive API for your project.
     - Create OAuth 2.0 Client IDs (Desktop app or Web application) and download the `credentials.json` file.
  2. **Install Google Client Library:** Install the Google API Client Library for your chosen language (e.g., `google-api-python-client` for Python).
  3. **Authentication Flow:** Implement the OAuth 2.0 authentication flow to get user consent and obtain access tokens. Store these tokens securely for future use.
  4. **Upload File Function:** Write a function to upload a file to Google Drive. This involves:
     - Specifying the file path.
     - Creating a `File` metadata object (name, MIME type).
     - Using the `files().create()` method of the Drive API.
  5. **List Files Function:** Write a function to list files in Google Drive. This involves using the `files().list()` method.
  6. **Download File Function:** Write a function to download a file from Google Drive, given its file ID. This involves using the `files().get()` method with `alt='media'`.
  7. **Integrate Functions:** Create a simple command-line interface or a basic web interface to trigger these functions.
- **Source Code (Python Example - Snippets for core operations):**

```
# Ensure you have installed: pip install google-api-python-client google-auth-oauthlib google-auth-httplib2

from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from googleapiclient.discovery import build
from googleapiclient.http import MediaFileUpload
import os

# If modifying these scopes, delete the file token.json.
SCOPES = ['https://www.googleapis.com/auth/drive.file'] # Access to files created or opened by the app

def authenticate_google_drive():
    creds = None
    # The file token.json stores the user's access and refresh tokens, and is
    # created automatically when the authorization flow completes for the first
    # time.
    if os.path.exists('token.json'):
        creds = Credentials.from_authorized_user_file('token.json', SCOPES)
    # If there are no (valid) credentials available, let the user log in.
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh_token(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES) # Path to your downloaded credentials.json
            creds = flow.run_local_server(port=0)
        # Save the credentials for the next run
        with open('token.json', 'w') as token:
            token.write(creds.to_json())
    return build('drive', 'v3', credentials=creds)

def upload_file(service, file_path, folder_id=None):
    file_name = os.path.basename(file_path)
    file_metadata = {'name': file_name}
    if folder_id:
        file_metadata['parents'] = [folder_id]

    media = MediaFileUpload(file_path, resumable=True)
    file = service.files().create(body=file_metadata, media_body=media, fields='id').execute()
    print(f"File ID: {file.get('id')} uploaded.")
    return file.get('id')

def list_files(service):
    results = service.files().list(
        pageSize=10, fields="nextPageToken, files(id, name, mimeType)").execute()
    items = results.get('files', [])
    if not items:
        print('No files found.')
    else:
        print('Files:')
        for item in items:
            print(f"{item['name']} ({item['id']}) - {item['mimeType']}")

def download_file(service, file_id, destination_path):
    request = service.files().get_media(fileId=file_id)
    with open(destination_path, 'wb') as fh:
        downloader = MediaIoBaseDownload(fh, request)
        done = False
        while done is False:
            status, done = downloader.next_chunk()
            print(f"Download {int(status.progress() * 100)}%.")
    print(f"File downloaded to {destination_path}")

# Example Usage:
if __name__ == '__main__':
    service = authenticate_google_drive()
    # To upload a file:
    # uploaded_file_id = upload_file(service, 'path/to/your/local/file.txt')
    # To list files:
    # list_files(service)
    # To download a file:
    # from googleapiclient.http import MediaIoBaseDownload # Import this for download_file
```

-
  ```
      # download_file(service, 'YOUR_FILE_ID_HERE', 'path/to/save/downloaded_file.txt')
  ```

- **Input:**
  1. `credentials.json` file from Google Cloud Console.
  2. Local file path for upload (e.g., `my_document.txt`).
  3. Google Drive File ID for download (e.g., `1aB2c3D4e5F6g7H8i9J0kL`).
  4. Destination path for downloaded files (e.g., `downloaded_file.txt`).
- **Expected Output:**
  1. Successful authentication and generation of `token.json`.
  2. Confirmation message for file upload, including the new file ID.
  3. A list of files present in your Google Drive, showing their names, IDs, and MIME types.
  4. Confirmation message for file download and the downloaded file appearing at the specified local path.

## Lab 5: Transfer Data using Google APPS

- **Title:** Transferring Data Between Google Applications using Google APIs
- **Aim:** To demonstrate how to programmatically transfer or copy data between different Google applications (e.g., copying a file from Google Drive to another Drive folder, or exporting data from Sheets).
- **Procedure:**
    1. **Google Cloud Project Setup:** (Similar to Lab 4)
        - Ensure the relevant Google APIs are enabled (e.g., Drive API, Sheets API).
        - Obtain OAuth 2.0 Client IDs and `credentials.json`.
    2. **Authentication:** Implement the OAuth 2.0 authentication flow for the necessary scopes.
    3. **Identify Source and Destination:** Determine the source of the data (e.g., a file in Drive, a spreadsheet in Sheets) and the destination.
    4. **Implement Data Transfer Logic:**
        - **Drive to Drive Copy:** Use the `files().copy()` method of the Google Drive API to duplicate a file.
        - **Sheets Data Export/Import:** Use the Google Sheets API to read data from one spreadsheet and write it to another, or to export data to a different format.
        - **Drive** to Cloud **Storage (Advanced):** For larger-scale transfers, consider using Google Cloud Storage and its transfer services, though this might involve more complex setup.
    5. **Error Handling:** Implement robust error handling for API calls.
- **Source Code (Python Example - Drive to Drive Copy):**

```
# Building upon the authentication from Lab 4

from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from googleapiclient.discovery import build
import os

SCOPES = ['https://www.googleapis.com/auth/drive'] # Full Drive access for copying

def authenticate_google_drive_full():
    creds = None
    if os.path.exists('token_full_drive.json'): # Use a different token file for broader scope
        creds = Credentials.from_authorized_user_file('token_full_drive.json', SCOPES)
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh_token(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)
        with open('token_full_drive.json', 'w') as token:
            token.write(creds.to_json())
    return build('drive', 'v3', credentials=creds)

def copy_file_in_drive(service, file_id, new_name, destination_folder_id=None):
    copied_file_metadata = {'name': new_name}
    if destination_folder_id:
        copied_file_metadata['parents'] = [destination_folder_id]

    try:
        copied_file = service.files().copy(
            fileId=file_id,
            body=copied_file_metadata,
            fields='id,name'
        ).execute()
        print(f"File '{copied_file.get('name')}' copied with ID: {copied_file.get('id')}")
        return copied_file.get('id')
    except Exception as e:
        print(f"An error occurred: {e}")
        return None

# Example Usage:
if __name__ == '__main__':
    service = authenticate_google_drive_full()
    # Replace 'SOURCE_FILE_ID' with the ID of the file you want to copy
    # Replace 'NEW_FILE_NAME' with the desired name for the copied file
    # Replace 'DESTINATION_FOLDER_ID' with the ID of the folder where you want to copy (optional)
    # copied_id = copy_file_in_drive(service, 'SOURCE_FILE_ID', 'NEW_FILE_NAME', 'DESTINATION_FOLDER_ID')
```

- **Input:**
  1. Source Google Drive File ID (e.g., `1aB2c3D4e5F6g7H8i9J0kL`).
  2. New name for the copied file (e.g., `my_document_copy.txt`).
  3. (Optional) Destination Google Drive Folder ID.
- **Expected Output:**
  1. Confirmation message that the file has been copied, along with the ID of the new copied file.
  2. The copied file appearing in the specified destination folder in Google Drive.

## Lab 6: Upload and Download using Google APPS

- **Title:** Advanced File Upload and Download Operations using Google Drive API
- **Aim:** To gain a deeper understanding of file upload and download mechanisms with Google Drive, including handling different file types and potential resumeable uploads.
- **Procedure:**
    1. **Google Cloud Project Setup & Authentication:** (Same as Lab 4)
        - Ensure Google Drive API is enabled.
        - Authenticate using OAuth 2.0.
    2. **Robust Upload Function:**
        - Implement a function to upload files, handling various MIME types.
        - Consider implementing resumable uploads for large files (though the `MediaFileUpload` class in Python client handles this automatically).
        - Add error handling for network issues or API limits.
    3. **Robust Download Function:**
        - Implement a function to download files, ensuring correct handling of binary data.
        - Allow specifying the destination path and filename.
        - Add error handling.
    4. **Folder Creation (Optional but useful):** Add a function to create new folders in Google Drive, which can be used as destinations for uploads.
    5. **User Interface:** Develop a simple command-line script or a basic web interface to allow users to specify files for upload/download and view progress.
- **Source Code (Python Example - Building on Lab 4):**

```
# Building upon the authentication and core functions from Lab 4

from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from googleapiclient.discovery import build
from googleapiclient.http import MediaFileUpload, MediaIoBaseDownload
import os
import io

SCOPES = ['https://www.googleapis.com/auth/drive'] # Full Drive access for comprehensive operations

def authenticate_google_drive_full():
    creds = None
    if os.path.exists('token_full_drive.json'):
        creds = Credentials.from_authorized_user_file('token_full_drive.json', SCOPES)
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh_token(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)
        with open('token_full_drive.json', 'w') as token:
            token.write(creds.to_json())
    return build('drive', 'v3', credentials=creds)

def create_folder(service, folder_name, parent_folder_id=None):
    file_metadata = {
        'name': folder_name,
        'mimeType': 'application/vnd.google-apps.folder'
    }
    if parent_folder_id:
        file_metadata['parents'] = [parent_folder_id]
    file = service.files().create(body=file_metadata, fields='id').execute()
    print(f"Folder ID: {file.get('id')} created.")
    return file.get('id')

# upload_file function (from Lab 4) can be reused here.
# download_file function (from Lab 4) can be reused here.

# Example Usage:
if __name__ == '__main__':
    service = authenticate_google_drive_full()

    # 1. Create a folder
    # new_folder_id = create_folder(service, 'MyCloudLabFiles')
    # print(f"New folder created with ID: {new_folder_id}")

    # 2. Upload a file to the new folder
    # local_file_to_upload = 'path/to/your/document.pdf'
    # uploaded_file_id = upload_file(service, local_file_to_upload, folder_id=new_folder_id)
    # print(f"Uploaded file ID: {uploaded_file_id}")

    # 3. List files in the new folder (optional, for verification)
    # print("\nFiles in 'MyCloudLabFiles' folder:")
    # results = service.files().list(
    #     q=f"'{new_folder_id}' in parents",
    #     pageSize=10, fields="nextPageToken, files(id, name, mimeType)").execute()
    # items = results.get('files', [])
    # if not items:
    #     print('No files found in this folder.')
    # else:
    #     for item in items:
    #         print(f"{item['name']} ({item['id']})")

    # 4. Download the uploaded file
    # downloaded_path = 'path/to/save/downloaded_document.pdf'
    # download_file(service, uploaded_file_id, downloaded_path)
```

- **Input:**
    1. Local file path for upload (e.g., `my_image.jpg`, `my_spreadsheet.xlsx`).
    2. Google Drive File ID for download.
    3. Destination local file path for download.
    4. (Optional) Folder name or ID for creating new folders or specifying upload destinations.
- **Expected Output:**
    1. Confirmation of successful file uploads with their respective IDs.
    2. Confirmation of successful file downloads, with the file appearing at the specified local path.
    3. Correct handling of various file types (e.g., images, documents, videos).
    4. (If implemented) Progress indicators for large file transfers.

## Lab 7: Encryption and Decryption of Text

- **Title:** Implementing Text Encryption and Decryption
- **Aim:** To understand and implement basic symmetric-key encryption and decryption algorithms for securing textual data.
- **Procedure:**
    1. **Choose an Algorithm:** Select a symmetric encryption algorithm (e.g., AES - Advanced Encryption Standard is commonly used and recommended for practical applications). For simplicity, a basic XOR cipher or Caesar cipher can be used for conceptual understanding, but AES is more robust.
    2. **Select a Library:** Use a cryptographic library available in your chosen programming language (e.g., `cryptography` in Python, `javax.crypto` in Java).
    3. **Key Generation:** Implement a method to generate a secure encryption key. For AES, this typically involves generating a random byte string of a specific length (e.g., 16, 24, or 32 bytes for AES-128, AES-192, AES-256).
    4. **Encryption Function:**
        - Take plaintext and a key as input.
        - Pad the plaintext if necessary (block ciphers require data to be a multiple of the block size).
        - Encrypt the padded plaintext using the chosen algorithm and key.
        - Return the ciphertext (often base64 encoded for text representation) and the Initialization Vector (IV) if using a mode like CBC.
    5. **Decryption Function:**
        - Take ciphertext (and IV) and the key as input.
        - Decrypt the ciphertext using the same algorithm and key.
        - Remove any padding.
        - Return the original plaintext.
    6. **User Interface:** Create a simple interface (command-line or GUI) to input text, generate a key, encrypt, and then decrypt the result.
- **Source Code (Python Example using `cryptography` library for AES):**

```
# Ensure you have installed: pip install cryptography

from cryptography.fernet import Fernet
import base64
import os
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.backends import default_backend

def generate_key():
    """Generates a new Fernet key."""
    return Fernet.generate_key()

def derive_key_from_password(password: str, salt: bytes = None) ->
bytes:
    """Derives a Fernet key from a password using PBKDF2HMAC."""
    if salt is None:
        salt = os.urandom(16) # Generate a new salt if not provided
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000,
        backend=default_backend()
    )
    key = base64.urlsafe_b64encode(kdf.derive(password.encode()))
    return key, salt

def encrypt_text(plaintext: str, key: bytes) -> bytes:
    """Encrypts plaintext using the given Fernet key."""
    f = Fernet(key)
    encrypted_text = f.encrypt(plaintext.encode())
    return encrypted_text

def decrypt_text(encrypted_text: bytes, key: bytes) -> str:
    """Decrypts encrypted text using the given Fernet key."""
    f = Fernet(key)
    decrypted_text = f.decrypt(encrypted_text).decode()
    return decrypted_text

# Example Usage:
if __name__ == '__main__':
    # Option 1: Generate a random key
    # key = generate_key()
    # print(f"Generated Key: {key.decode()}")

    # Option 2: Derive key from a password (more practical for user input)
    user_password = "mysecretpassword123"
    # In a real app, store salt with the encrypted data or derive it deterministically
    # For demonstration, we'll generate it here.
    key_from_password, salt = derive_key_from_password(user_password)
    print(f"Derived Key (from password): {key_from_password.decode()}")
    print(f"Salt used: {salt.hex()}") # Store this salt to decrypt later

    original_text = "This is a secret message that needs to be encrypted."
    print(f"\nOriginal Text: {original_text}")

    # Encrypt
    encrypted_message = encrypt_text(original_text, key_from_password)
    print(f"Encrypted Message: {encrypted_message.decode()}")

    # Decrypt
    # When decrypting, you'd need the same password and the *same* salt
    retrieved_key, _ = derive_key_from_password(user_password, salt) # Use the same salt
    decrypted_message = decrypt_text(encrypted_message, retrieved_key)
    print(f"Decrypted Message: {decrypted_message}")

    assert original_text == decrypted_message
    print("\nEncryption and Decryption successful!")
```

- **Input:**
    1. Plaintext string to be encrypted (e.g., "Hello, Cloud Security!").
    2. A secret key (generated by the program or derived from a password).
- **Expected Output:**
    1. The generated encryption key (if applicable).
    2. The base64-encoded ciphertext.
    3. The successfully decrypted text, which should match the original plaintext.

## Lab 8: Simple Experiments in CloudSim

- **Title:** Basic Cloud Simulation with CloudSim
- **Aim:** To introduce the CloudSim toolkit and perform a simple simulation of a cloud computing environment, including datacenters, hosts, virtual machines (VMs), and cloudlets (tasks).
- **Procedure:**
    1. **CloudSim Setup:**
        - Download the CloudSim library (Java JARs).
        - Set up a Java development environment (e.g., Eclipse, IntelliJ IDEA) and create a new Java project.
        - Add the CloudSim JARs to your project's build path.
    2. **Initialize CloudSim:** Start the CloudSim simulation engine.
    3. **Create Datacenter:** Define a datacenter with specific characteristics (e.g., architecture, OS, VMM, cost per processing unit, memory, storage, bandwidth).
    4. **Create Hosts:** Within the datacenter, define one or more physical hosts, specifying their processing capabilities (MIPS - Millions of Instructions Per Second), RAM, storage, and bandwidth.
    5. **Create Virtual Machines (VMs):** Define several VMs, each with its own MIPS, RAM, storage, and image size. Allocate these VMs to the hosts.
    6. **Create Cloudlets (Tasks):** Define cloudlets, which represent computational tasks. Each cloudlet has a length (in Million Instructions - MI) and file sizes.
    7. **Broker Creation:** Create a DatacenterBroker, which acts on behalf of users to submit cloudlets to datacenters and manage their execution on VMs.
    8. **Submit Cloudlets:** The broker submits the defined cloudlets to the VMs.
    9. **Start Simulation:** Run the CloudSim simulation.
    10. **Print Results:** After the simulation finishes, retrieve and print the execution results, such as cloudlet status, execution time, and host utilization.
- **Source Code (Java Example - Basic CloudSim Setup):**

```
// Ensure CloudSim JARs are in your project's build path.
// Example requires CloudSim 3.0 or later.

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerSpaceShared;
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

public class SimpleCloudSimExperiment {

```

```
•       private static List<Cloudlet> cloudletList;
•       private static List<Vm> vmList;
•
•       public static void main(String[] args) {
•           Log.printLine("Starting Simple CloudSim Experiment...");
•
•           try {
•               // 1. Initialize CloudSim
•               int num_user = 1; // number of cloud users
•               Calendar calendar = Calendar.getInstance();
•               boolean trace_flag = false; // mean trace events
•
•               CloudSim.init(num_user, calendar, trace_flag);
•
•               // 2. Create Datacenter
•               Datacenter datacenter0 = createDatacenter("Datacenter_0");
•
•               // 3. Create Broker
•               DatacenterBroker broker = createBroker();
•               int brokerId = broker.getId();
•
•               // 4. Create VMs
•               vmList = new ArrayList<Vm>();
•               int vmid = 0;
•               long mips = 1000; // MIPS for each PE
•               int pesNumber = 1; // number of PEs (cores)
•               long ram = 512; // VM memory (MB)
•               long bw = 1000; // VM bandwidth (Mb/s)
•               long size = 10000; // VM image size (MB)
•               String vmm = "Xen"; // VMM name
•
•               // Create 2 VMs
•               vmList.add(new Vm(vmid++, brokerId, mips, pesNumber, ram,
  bw, size, vmm, new CloudletSchedulerSpaceShared()));
•               vmList.add(new Vm(vmid++, brokerId, mips, pesNumber, ram,
  bw, size, vmm, new CloudletSchedulerSpaceShared()));
•
•               broker.submitVmList(vmList);
•
•               // 5. Create Cloudlets
•               cloudletList = new ArrayList<Cloudlet>();
•               int id = 0;
•               long length = 40000; // Cloudlet length (MI)
•               long fileSize = 300; // Input file size (bytes)
•               long outputSize = 300; // Output file size (bytes)
•               int pes = 1; // Number of PEs required by this Cloudlet
•
•               // Create 4 Cloudlets
•               cloudletList.add(new Cloudlet(id++, length, pes, fileSize,
  outputSize, new DatacenterCharacteristics("x86", 3.0, 100.0, 10.0,
  5.0), new CloudletSchedulerSpaceShared()));
•               cloudletList.add(new Cloudlet(id++, length, pes, fileSize,
  outputSize, new DatacenterCharacteristics("x86", 3.0, 100.0, 10.0,
  5.0), new CloudletSchedulerSpaceShared()));
•               cloudletList.add(new Cloudlet(id++, length, pes, fileSize,
  outputSize, new DatacenterCharacteristics("x86", 3.0, 100.0, 10.0,
  5.0), new CloudletSchedulerSpaceShared()));
```

```
            cloudletList.add(new Cloudlet(id++, length, pes, fileSize, outputSize, new DatacenterCharacteristics("x86", 3.0, 100.0, 10.0, 5.0), new CloudletSchedulerSpaceShared()));

            broker.submitCloudletList(cloudletList);

            // 6. Start the simulation
            CloudSim.startSimulation();

            // 7. Print results
            List<Cloudlet> newList = broker.getCloudletReceivedList();
            CloudSim.stopSimulation();
            printCloudletResults(newList);

            Log.printLine("CloudSim Experiment finished!");

        } catch (Exception e) {
            e.printStackTrace();
            Log.printLine("Unwanted errors happen");
        }
    }

    private static Datacenter createDatacenter(String name) {
        List<Host> hostList = new ArrayList<Host>();
        List<Pe> peList = new ArrayList<Pe>();

        int mips = 10000; // MIPS for each PE
        peList.add(new Pe(0, new PeProvisionerSimple(mips))); // 1 PE

        int hostId = 0;
        int ram = 2048; // host memory (MB)
        long storage = 1000000; // host storage (MB)
        int bw = 10000; // host bandwidth (Mb/s)

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList,
                        new VmAllocationPolicySimple(peList)
                )
        ); // This host has 1 PE

        String arch = "x86"; // system architecture
        String os = "Linux"; // operating system
        String vmm = "Xen";
        double time_zone = 10.0; // time zone this resource is in
        double cost = 3.0; // the cost of using processing in this resource
        double costPerMem = 0.05; // the cost of using memory in this resource
        double costPerStorage = 0.001; // the cost of using storage in this resource
        double costPerBw = 0.0; // the cost of using bandwidth in this resource

        LinkedList<Storage> storageList = new LinkedList<Storage>(); // We are not creating any storage devices for this example

        DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
                arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);

        Datacenter datacenter = null;
        try {
            datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList, 0);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return datacenter;
    }

    private static DatacenterBroker createBroker() {
        DatacenterBroker broker = null;
        try {
            broker = new DatacenterBroker("Broker");
        } catch (Exception e) {
            e.printStackTrace();
        }
        return broker;
    }

    private static void printCloudletResults(List<Cloudlet> list) {
        int numCloudlet = list.size();
        String indent = "    ";
        Log.printLine();
        Log.printLine("========== OUTPUT ==========");
        Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +
                "Datacenter ID" + indent + "VM ID" + indent + "Time" + indent + "Start Time" + indent + "Finish Time");

        DecimalFormat dft = new DecimalFormat("###.##");
        for (int i = 0; i < numCloudlet; i++) {
            Cloudlet cloudlet = list.get(i);
            Log.print(indent + cloudlet.getCloudletId() + indent + indent);

            if (cloudlet.getCloudletStatus() == 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()));
            }
        }
    }
}
```

- **Input:**
  1. CloudSim library JARs.
  2. Java Development Kit (JDK).
  3. Configuration parameters within the Java code (e.g., number of hosts, VMs, cloudlets, their MIPS, RAM, etc.).
- **Expected Output:**
  1. Console output showing the simulation progress.
  2. A table summarizing the results for each cloudlet, including its ID, status (SUCCESS), the datacenter and VM ID it ran on, actual CPU time, start time, and finish time.

## Lab 9: Simple Experiments in CloudSim (Continued)

- **Title:** Advanced Cloud Simulation Experiments with CloudSim
- **Aim:** To explore more complex scenarios in CloudSim, such as different VM allocation policies, cloudlet scheduling algorithms, or heterogeneous host configurations.
- **Procedure:**
  1. **CloudSim Setup:** (Same as Lab 8)
  2. **Modify Datacenter/Host Characteristics:**
     - Introduce hosts with varying MIPS, RAM, or bandwidth to simulate a heterogeneous environment.
     - Create multiple datacenters.
  3. **Experiment with VM Allocation Policies:**
     - Instead of `VmAllocationPolicySimple`, try `VmAllocationPolicyFirstFit` or implement a custom allocation policy.
  4. **Experiment with Cloudlet Scheduling:**
     - Change `CloudletSchedulerSpaceShared` to `CloudletSchedulerTimeShared` for VMs.
     - Observe the impact on cloudlet execution times.
  5. **Dynamic VM Creation/Migration (Advanced):** Implement logic for creating VMs dynamically or migrating them during the simulation (requires more advanced CloudSim features).
  6. **Analyze Results:** Collect and analyze more detailed metrics, such as datacenter power consumption, network traffic, or specific VM utilization patterns.
- **Source Code (Java Example - Modifying VM Allocation Policy and adding another Host):**

```
// Building upon SimpleCloudSimExperiment from Lab 8
// Changes highlighted below

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerSpaceShared;
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple; // Still using simple for demonstration, but you can change this
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

public class AdvancedCloudSimExperiment {

    private static List<Cloudlet> cloudletList;
    private static List<Vm> vmList;

    public static void main(String[] args) {
        Log.printLine("Starting Advanced CloudSim Experiment...");

        try {
            int num_user = 1;
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false;

            CloudSim.init(num_user, calendar, trace_flag);

            // Create Datacenter with multiple hosts (heterogeneous example)
            Datacenter datacenter0 = createHeterogeneousDatacenter("Datacenter_0");

            DatacenterBroker broker = createBroker();
            int brokerId = broker.getId();

            vmList = new ArrayList<Vm>();
            int vmid = 0;
            long mips = 1000;
            int pesNumber = 1;
            long ram = 512;
            long bw = 1000;
            long size = 10000;
            String vmm = "Xen";

            // Create 3 VMs
            vmList.add(new Vm(vmid++, brokerId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerSpaceShared()));
            vmList.add(new Vm(vmid++, brokerId, mips * 2, pesNumber, ram * 2, bw * 2, size, vmm, new CloudletSchedulerSpaceShared())); // A more powerful VM
            vmList.add(new Vm(vmid++, brokerId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerSpaceShared()));

            broker.submitVmList(vmList);

            cloudletList = new ArrayList<Cloudlet>();
            int id = 0;
            long length = 40000;
            long fileSize = 300;
            long outputSize = 300;
            int pes = 1;

            // Create 6 Cloudlets
            cloudletList.add(new Cloudlet(id++, length, pes, fileSize, outputSize, new DatacenterCharacteristics("x86", 3.0, 100.0, 10.0, 5.0), new CloudletSchedulerSpaceShared()));
            cloudletList.add(new Cloudlet(id++, length, pes, fileSize, outputSize, new DatacenterCharacteristics("x86", 3.0, 100.0, 10.0, 5.0), new CloudletSchedulerSpaceShared()));
            cloudletList.add(new Cloudlet(id++, length * 2, pes, fileSize, outputSize, new DatacenterCharacteristics("x86", 3.0, 100.0, 10.0, 5.0), new CloudletSchedulerSpaceShared())); // Longer cloudlet
            cloudletList.add(new Cloudlet(id++, length, pes, fileSize, outputSize, new DatacenterCharacteristics("x86", 3.0, 100.0, 10.0, 5.0), new CloudletSchedulerSpaceShared()));
            cloudletList.add(new Cloudlet(id++, length, pes, fileSize, outputSize, new DatacenterCharacteristics("x86", 3.0, 100.0, 10.0, 5.0), new CloudletSchedulerSpaceShared()));
            cloudletList.add(new Cloudlet(id++, length * 3, pes, fileSize, outputSize, new DatacenterCharacteristics("x86", 3.0, 100.0, 10.0, 5.0), new CloudletSchedulerSpaceShared())); // Even longer cloudlet


            broker.submitCloudletList(cloudletList);

            CloudSim.startSimulation();

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

            Log.printLine("CloudSim Advanced Experiment finished!");

        } catch (Exception e) {
            e.printStackTrace();
            Log.printLine("Unwanted errors happen");
        }
    }

    private static Datacenter createHeterogeneousDatacenter(String name) {
        List<Host> hostList = new ArrayList<Host>();

        // Host 1: Standard
        List<Pe> peList1 = new ArrayList<Pe>();
        peList1.add(new Pe(0, new PeProvisionerSimple(10000))); // 10000 MIPS
        hostList.add(
                new Host(
                        0,
                        new RamProvisionerSimple(2048),
                        new BwProvisionerSimple(10000),
                        1000000,
                        peList1,
                        new VmAllocationPolicySimple(peList1)
                )
        );

        // Host 2: More powerful
        List<Pe> peList2 = new ArrayList<Pe>();
        peList2.add(new Pe(0, new PeProvisionerSimple(20000))); // 20000 MIPS
        peList2.add(new Pe(1, new PeProvisionerSimple(20000))); // 2 PEs
        hostList.add(
                new Host(
                        1,
                        new RamProvisionerSimple(4096),
                        new BwProvisionerSimple(20000),
                        2000000,
                        peList2,
                        new VmAllocationPolicySimple(peList2)
                )
        );

        String arch = "x86";
        String os = "Linux";
        String vmm = "Xen";
        double time_zone = 10.0;
        double cost = 3.0;
        double costPerMem = 0.05;
        double costPerStorage = 0.001;
        double costPerBw = 0.0;

        LinkedList<Storage> storageList = new LinkedList<Storage>();

        DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
                arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);

        Datacenter datacenter = null;
        try {
            datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList
```