

Here is a comprehensive **POE Preparation Guide** containing all the tasks, code, and configuration steps we discussed. You can copy this content into a Word document or Google Doc and save it as a PDF.

Cloud Computing & Lab POE Preparation Guide

Course: UCSC0704 Cloud Computing Lab

Exam Type: Practical Oral Exam (POE)

1. OpenMPI: Master-Slave Communication (on 2 PCs)

Goal: Perform a task (like Matrix Multiplication) distributed across two machines (Master and Slave).

A. Pre-requisites (Network Setup)

Crucial for the exam. Do this before running code.

1. Install MPI (Both PCs):

```
Bash
sudo apt update
sudo apt install openmpi-bin libopenmpi-dev
```

2. SSH Configuration (Password-less Login):

- **On Master:** Generate key: `ssh-keygen -t rsa` (Press Enter for all prompts).
- **On Master:** Copy key to Slave: `ssh-copy-id user@<SLAVE_IP>`
- **Test:** `ssh user@<SLAVE_IP>` (Should log in without password).

B. The Code (mpi_matrix.c)

Save this file on the Master PC.

C

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

#define N 4 // Matrix Size

int main(int argc, char *argv[]) {
    int rank, size;
    int A[N][N], B[N][N], C[N][N]; // A, B = Input, C = Result
    int i, j, k;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // 1. Master initializes data
    if (rank == 0) {
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++) {
                A[i][j] = 1; // Fill with 1s
                B[i][j] = 1; // Fill with 1s
            }
        }
    }

    // 2. Broadcast Matrix B to all (everyone needs full B)
    MPI_Bcast(B, N*N, MPI_INT, 0, MPI_COMM_WORLD);

    // 3. Scatter Matrix A (Divide rows among workers)
    int rows_per_process = N / size;
    int sub_A[rows_per_process][N];
    MPI_Scatter(A, rows_per_process * N, MPI_INT, sub_A, rows_per_process * N, MPI_INT, 0,
MPI_COMM_WORLD);
```

```

// 4. Compute
int sub_C[rows_per_process][N];
for (i = 0; i < rows_per_process; i++) {
    for (j = 0; j < N; j++) {
        sub_C[i][j] = 0;
        for (k = 0; k < N; k++) {
            sub_C[i][j] += sub_A[i][k] * B[k][j];
        }
    }
}

// 5. Gather results back to Master
MPI_Gather(sub_C, rows_per_process * N, MPI_INT, C, rows_per_process * N, MPI_INT, 0,
MPI_COMM_WORLD);

// 6. Master prints result
if (rank == 0) {
    printf("Result Matrix:\n");
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) printf("%d ", C[i][j]);
        printf("\n");
    }
}

MPI_Finalize();
return 0;
}

```

C. Execution Steps

1. Compile:

```

Bash
mpicc mpi_matrix.c -o matrix_app

```

2. Create Hostfile (my_hosts):

```

Plaintext
localhost slots=2
192.168.1.XX slots=2

```

3. **Run:**

Bash

```
mpirun -np 4 --hostfile my_hosts ./matrix_app
```

2. DNS Configuration (Local)

Goal: Host a website on Machine A and access it via a domain name (www.mysite.com) on Machine B.

Step 1: Machine A (The Server)

1. Install Apache: `sudo apt install apache2`
2. Start Service: `sudo systemctl start apache2`
3. Find IP Address: Run `ifconfig` (Assume IP is 192.168.1.50).

Step 2: Machine B (The Client)

1. Open hosts file:

Bash

```
sudo nano /etc/hosts
```

2. Add the mapping at the bottom:

Plaintext

```
192.168.1.50 www.mysite.com
```

3. Save (Ctrl+O) and Exit (Ctrl+X).
 4. **Test:** Open browser and type `www.mysite.com`.
-

3. AWS EC2 Web App Deployment

Goal: Launch a cloud VM and host a simple HTML page.

Step 1: Launch Instance (AWS Console)

1. **OS:** Select **Ubuntu 22.04 LTS**.
2. **Key Pair:** Create new key pair (my-key.pem). **Download it immediately.**
3. **Security Group:** Add Rule -> Type: **HTTP**, Port: **80**, Source: **Anywhere (0.0.0.0/0)**.

Step 2: Connect & Install (Terminal)

1. Change key permissions:
Bash
`chmod 400 my-key.pem`
 2. SSH into instance:
Bash
`ssh -i "my-key.pem" ubuntu@<PUBLIC-IP>`
 3. Install Web Server:
Bash
`sudo apt update`
`sudo apt install apache2 -y`
 4. Deploy App:
Bash
`echo "<h1>Welcome to Cloud Lab</h1>" | sudo tee /var/www/html/index.html`
 5. **Verify:** Open <PUBLIC-IP> in your browser.
-

4. Docker App Deployment (Single Container)

Goal: Run a simple Python app inside a container.

Step 1: Create Dockerfile

Dockerfile

```
FROM python:3.9-slim
CMD ["echo", "Hello from Docker!"]
```

Step 2: Build & Run

1. Build Image:

```
Bash
docker build -t my-hello-app .
```

2. Run Container:

```
Bash
docker run my-hello-app
```

5. Docker Compose (Web App + Database)

Goal: Run a Python Flask app connected to a Redis database using orchestration.

File 1: app.py

Python

```

from flask import Flask
from redis import Redis

app = Flask(__name__)
# 'redis' is the hostname defined in docker-compose.yml
redis = Redis(host='redis', port=6379)

@app.route('/')
def hello():
    count = redis.incr('hits')
    return f'Hello! You have visited {count} times.\n'

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)

```

File 2: Dockerfile

Dockerfile

```

FROM python:3.9-slim
WORKDIR /code
RUN pip install flask redis
COPY . .
CMD ["python", "app.py"]

```

File 3: docker-compose.yml

YAML

```

version: "3.8"

```

```
services:
  web:
    build: .
    ports:
      - "5000:5000"
  redis:
    image: "redis:alpine"
```

Execution

1. Run: docker-compose up
 2. Test: Open `http://localhost:5000` in a browser. Refresh to see the counter increase.
-

6. Eavesdropping Attack (Network Security)

Goal: Capture unencrypted passwords using Wireshark.

1. **Tool:** Open **Wireshark** (or use tcpdump in CLI).
 2. **Capture:** Select interface (e.g., eth0 or wlan0) and click Start.
 3. **Filter:** Type `http.request.method == "POST"` in the top bar.
 4. **Action:**
 - Have the "victim" log into a test HTTP website.
 - Find the POST packet in Wireshark.
 - Right-click -> **Follow** -> **TCP Stream**.
 - Look for `username=...` and `password=...` in red/blue text.
-

7. AWS to GCP Migration (Steps Only)

Goal: Move a VM from Amazon to Google Cloud.

1. **Snapshot:** Create a snapshot of the EC2 instance root volume in AWS.
2. **Export:** Use AWS CLI to export snapshot to S3 bucket as .vmdk or .raw.
 - `aws ec2 export-image ...`
3. **Transfer:** Move the image file from S3 to a **Google Cloud Storage (GCS)** bucket.
4. **Import:** In GCP Console -> **Compute Engine** -> **Images** -> **Create Image**.

- Select "Source: Cloud Storage file".
5. **Launch:** Create a new GCP VM instance using this custom image.