

CC lab

🕒 Created	@February 27, 2026 7:33 PM
-----------	----------------------------

Lab 1

◆ PART 1: CREATE EC2 INSTANCE (LINUX – UBUNTU)

STEP 1: Launch Instance

1. Login to **AWS Management Console**
2. Go to **EC2 → Instances**
3. Click **Launch instances**

STEP 2: Choose AMI

- Select **Ubuntu Server 20.04 / 22.04 LTS**
- AMI = **Operating system + preinstalled packages**

STEP 3: Choose Instance Type

- Select **t2.micro** (Free Tier)

STEP 4: Key Pair (MANDATORY)

- Choose **Create new key pair**
- Type: **RSA**
- Format: **.pem**
- Download and **DO NOT DELETE**

👉 Why mandatory?

- Linux EC2 has **no password**
- SSH authentication works **only using key pair**
- ❌ Without key pair → **no login possible**

STEP 5: Configure Security Group

Add **Inbound Rule**:

Type	Port	Source
SSH	22	My IP

⚠ If SSH (22) is blocked → **connection fails**

STEP 6: Launch Instance

- Click **Launch Instance**
- Wait till **Instance State = Running**

♦ PART 2: CONNECT LINUX EC2 USING PuTTY (.ppk)

PREREQUISITES

- ✓ EC2 running
- ✓ Public IPv4 / DNS
- ✓ `.ppk` file
- ✓ PuTTY installed

STEP 1: Open PuTTY

- Launch **PuTTY**

STEP 2: Enter Host Name

Format:

```
ubuntu@<PUBLIC-IP>
```

Example:

```
ubuntu@3.110.xxx.xxx
```

📌 **Username depends on AMI**

- Ubuntu → `ubuntu`

- Amazon Linux → `ec2-user`

STEP 3: Attach Private Key

Go to:

Connection → SSH → Auth → Credentials

- Click **Browse**
- Select `.ppk` file

STEP 4: Connect

- Click **Open**
- First time → Click **Yes**

🎉 You are logged in automatically

VERIFY

`whoami`

Output:

`ubuntu`

◆ PART 3: USING .PEM KEY (Convert to .ppk)

WHY CONVERSION?

- PuTTY **cannot read .pem**
- PuTTY **only accepts .ppk**

STEP 1: Open PuTTYgen

1. Click **Load**
2. Select `.pem`

(change filter → *All Files*)

3. Click **Open**

STEP 2: Save as .ppk

- Click **Save private key**
- Ignore warning
- Save as `keyname.ppk`

✓ Conversion complete

➡ Now follow **PuTTY steps above**

◆ PART 4: CONNECT USING AWS CLOUDSHELL

WHEN TO USE?

- ✓ Corporate laptop restrictions
 - ✓ No local SSH tools
 - ✓ Browser-only access
-

STEP 1: Open CloudShell

- AWS Console → Click **CloudShell icon**

STEP 2: Copy SSH Command

- EC2 → Instance → **Connect**
- Choose **SSH client**
- Copy:

```
ssh -i "key.pem" ubuntu@<public-ip>
```

✗ Will fail initially (key not present)

STEP 3: Upload Key to CloudShell

- Click **Actions (+)** → Upload file
- Upload `.pem`

STEP 4: Run SSH Again

- Paste command
- Press Enter

✓ Connected successfully

◆ PART 5: CREATE WINDOWS EC2 + CONNECT VIA RDP

4

STEP 1: Launch Windows Instance

1. EC2 → Launch instance
2. Select **Windows Server 2019 / 2022**
3. Choose **t2.micro**
4. Create / select key pair (`.pem`)
5. Security Group → Add rule:

Type	Port	Source
RDP	3389	My IP

⚠ **Never use 0.0.0.0/0 in production**

STEP 2: Get Public IP

- EC2 → Instances → Copy **Public IPv4**
-

STEP 3: Get Administrator Password

1. Instance → **Connect**
 2. Choose **RDP client**
 3. Click **Get Password**
 4. Upload `.pem`
 5. Click **Decrypt Password**
 6. Copy password
-

STEP 4: Connect via RDP

1. Open **Remote Desktop (mstsc)**
2. Enter **Public IP**
3. Username: `Administrator`
4. Password: *(decrypted)*

 Windows desktop opens

◆ PART 6: COMMON FAILURE CHECKLIST

Issue	Check
SSH timeout	Port 22 allowed? Correct IP?
Permission denied	Wrong username or key
RDP fails	Port 3389 blocked
CloudShell SSH fails	Key not uploaded
PuTTY login fails	.ppk not attached

Lab 2

1(a) Create and Attach an EBS Volume to an EC2 Instance

STEP 1: Launch EC2 Instance (if not already created)

1. Login to **AWS Management Console**
2. Go to **Amazon EC2**
3. Click **Launch instance**
4. Choose:
 - AML: **Amazon Linux 2**
 - Instance Type: **t2.micro**
 - Key pair: existing / create new

- Security group: allow **SSH (22)**

5. Click **Launch instance**

✓ Instance state → **Running**

STEP 2: Create a New EBS Volume

1. EC2 Dashboard → **Elastic Block Store** → **Volumes**

2. Click **Create volume**

3. Choose:

- Volume type: **gp3**
- Size: **10–20 GB**
- Availability Zone: **Same AZ as EC2**

4. Click **Create volume**

📌 **IMPORTANT:**

EBS **CANNOT** be attached across AZs.

STEP 3: Attach the EBS Volume

1. Select the volume

2. **Actions** → **Attach volume**

3. Select:

- Instance: your EC2
- Device name: `/dev/xvdf`

4. Click **Attach**

✓ Volume state → **In-use**

STEP 4: Verify Attachment Inside EC2

Connect via SSH:

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

Check disks:

```
lsblk
```

You will see:

- `/dev/nvme0n1` → root
- `/dev/nvme1n1` → new EBS volume

TASK-1(b)

Scale EC2 Instance (Increase CPU & RAM)

STEP 1: Stop Instance

1. EC2 → Instances
2. Select instance
3. **Instance state** → **Stop**

 Required to change instance type.

STEP 2: Change Instance Type

1. Select stopped instance
2. **Actions** → **Instance settings** → **Change instance type**
3. Choose higher type:
 - Example: `t2.micro` → `t2.small`
4. Click **Apply**

STEP 3: Start Instance

1. **Instance state** → **Start**
- ✓ CPU & RAM increased
 - ✓ EBS volume remains attached

WEEK-2 TASK-2

Permanently Mount EBS Volume (Data Persistence)

STEP 1: Identify Disk

```
lsblk
```

New disk: `/dev/nvme1n1`

STEP 2: Create Partition

```
sudo fdisk /dev/nvme1n1
```

Inside fdisk:

```
n → p → Enter → Enter → Enter → w
```

Notify kernel:

```
sudo partprobe
```

STEP 3: Format the Volume

```
sudo mkfs.xfs /dev/nvme1n1p1
```

STEP 4: Create Mount Directory

```
sudo mkdir /mnt/archana
```

STEP 5: Mount the Volume

```
sudo mount /dev/nvme1n1p1 /mnt/archana  
df-h
```

STEP 6: Make Mount Persistent (fstab)

Get UUID:

```
sudo blkid /dev/nvme1n1p1
```

Edit fstab:

```
sudo nano /etc/fstab
```

Add:

```
UUID=xxxx-xxxx /mnt/archana xfs defaults,nofail 0  
0
```

Test:

```
sudo mount -a
```

Reboot:

```
sudo reboot
```

✓ Volume auto-mounts after reboot

✓ Data persists

✓ WEEK-2 TASK-3

Snapshot → Copy → Restore in Another Region

STEP 1: Create Snapshot

1. EC2 → **Volumes**
 2. Select EBS volume
 3. **Actions** → **Create snapshot**
 4. Add name & description
 5. Click **Create snapshot**
-

STEP 2: Copy Snapshot to Another Region

1. EC2 → **Snapshots**
 2. Select snapshot
 3. **Actions** → **Copy snapshot**
 4. Destination region (example: Mumbai → N. Virginia)
 5. Click **Copy snapshot**
-

STEP 3: Create Volume from Snapshot (New Region)

1. Switch AWS Region
 2. EC2 → Snapshots
 3. Select copied snapshot
 4. **Actions** → **Create volume**
 5. Choose:
 - AZ of target EC2
 - Volume type: gp3
 6. Click **Create**
-

STEP 4: Attach Volume to EC2 (New Region)

1. EC2 → Volumes
2. Select new volume
3. **Actions** → **Attach volume**
4. Choose EC2 instance

5. Device: `/dev/xvdf`

✓ Data restored successfully

Lab 3

✓ STEP 1: Launch Two EC2 Instances (Amazon Linux)

1. Open EC2 Console

- Login to **AWS Management Console**
- Search → **EC2**
- Click **Launch instance**

2. Configure First Instance (EFS-1)

Field	Value
Name	<code>EFS-1</code>
AMI	Amazon Linux 2
Instance Type	t2.micro
Key Pair	<code>EFS.pem</code>
VPC	Default
Subnet	Subnet-1a
Auto-assign Public IP	Enable

3. Create Security Group (VERY IMPORTANT)

Create **new Security Group**

Rule	Value
Type	NFS
Protocol	TCP
Port	2049

Rule	Value
Source	Anywhere (0.0.0.0/0)

👉 This allows EFS to communicate with EC2.

Click **Launch Instance**

4. Launch Second Instance (EFS-2)

Repeat **same steps**, change only:

Field	Value
Name	EFS-2
Subnet	Subnet-1b

✓ Now you have **2 EC2 instances in different subnets**

✓ STEP 2: Create Amazon EFS File System

1. Open EFS Console

- AWS Console → Search **EFS**
 - Click **Create file system**
-

2. Basic Settings

Field	Value
Name	Optional
VPC	Default
Enable Region	✓ Enabled

Click **Next**

3. Network Settings

- **Delete all default security groups**
- Select **the same NFS security group** used by EC2

👉 This is **mandatory** for EFS to mount.

Click **Next** → **Create file system**

✓ EFS is now created with **mount targets in all AZs**

✓ STEP 3: Connect to EC2 Instances (PowerShell)

1. Open Two PowerShell Windows

- PowerShell-1 → EFS-1
 - PowerShell-2 → EFS-2
-

2. SSH into Each Instance

```
ssh -i EFS.pem ec2-user@<Public-IP>
```

(Replace `<Public-IP>` with your instance IP)

3. Switch to Root User

```
sudo su
```

4. Create Mount Directory

```
mkdir efs
```

5. Install EFS Utilities

```
yum install -y amazon-efs-utils
```

6. Verify Installation

```
efs-utils --version
```

7. Repeat All Above Commands on EFS-2

✓ Both EC2 instances are now **ready to mount EFS**

✓ STEP 4: Attach (Mount) EFS to EC2 Instances

1. Go to EFS Console

- Select your **EFS file system**
 - Click **Attach**
-

2. Choose Mount Method

- Select **Mount via DNS**
- Copy the command (example):

```
mount-t efs fs-xxxx:/ efs
```

3. Run Command on Both EC2 Instances

Paste the command in:

- PowerShell-1 (EFS-1)
- PowerShell-2 (EFS-2)

✓ EFS is now mounted to `/efs`

✓ STEP 5: Verify EFS Working (MOST IMPORTANT)

On EFS-1

```
cd efs
touch testfile.txt
```

```
ls
```

On EFS-2

```
cd efs  
ls
```

🎉 You will see `testfile.txt` on both instances

Lab 4

✓ PART 1: Creating an S3 Bucket, Uploading an Object, and Enabling Public Access (ACL Method)

◆ Step 1: Open Amazon S3 Service

1. Login to **AWS Management Console**
2. In the search bar, type **S3**
3. Click on **Amazon S3**

📌 You will now see the **S3 Buckets dashboard**

◆ Step 2: Create a New S3 Bucket

1. Click **Create bucket**
2. Enter:
 - **Bucket name:** `my-public-bucket-123`
(Must be globally unique)
 - **Region:** Choose nearest region (e.g., Mumbai)

📌 Bucket names:

- No spaces
- Only lowercase letters, numbers, hyphens

- Must be unique worldwide
-

◆ Step 3: Disable Block Public Access (IMPORTANT)

1. Scroll to **Block Public Access settings for this bucket**
2. ✗ Uncheck **Block all public access**
3. AWS will show a warning
4. ✓ Check **"I acknowledge that this bucket will become public"**
5. Click **Create bucket**

✓ Bucket is created and can now allow public access

⚠ **Without this step, public access will NEVER work**

◆ Step 4: Enable Public Access at Bucket Level (ACL)

1. Click on the bucket name
2. Go to **Permissions** tab
3. Scroll to **Access Control List (ACL)**
4. Click **Edit**
5. Under **Public access:**
 - Everyone (public access) → ✓ **Read**
6. Click **Save changes**

📌 This allows **anyone on the internet** to read objects in the bucket

◆ Step 5: Upload an Object

1. Go to **Objects** tab
2. Click **Upload**
3. Click **Add files**
4. Select a file (example: `image.jpg`)
5. Click **Upload**

✓ File uploaded successfully

◆ Step 6: Enable Object-Level Public Access (ACL)

1. Click on the uploaded object
2. Go to **Permissions** tab
3. Scroll to **Access Control List (ACL)**
4. Click **Edit**
5. Under **Public access**:
 - Everyone (public access) → ✓ **Read**
6. Click **Save changes**

📌 Object is now publicly readable

◆ Step 7: Verify Public Access

1. Copy **Object URL**
2. Open it in **Incognito mode**
3. Paste and press Enter

✓ File opens without login

🎉 Public access confirmed

✓ PART 2: S3 Versioning & Cross-Region Replication (CRR)

◆ Steps to Enable Versioning

1. Open **S3**
2. Select your bucket
3. Go to **Properties**
4. Scroll to **Bucket Versioning**
5. Click **Edit**
6. Select **Enable**
7. Click **Save changes**

✓ Versioning enabled

◆ Output Scenario (What You Observe)

✓ Step A: Upload Object

- Upload `report.pdf`

📌 Output:

- Version ID is automatically assigned
-

✓ Step B: Upload Same File Again

- Upload `report.pdf` again

📌 Output:

- Old version is preserved
 - New version becomes **current version**
-

✓ Step C: Delete Object

- Click **Delete**

📌 Output:

- Object disappears
- **Delete marker** is created
- Old versions still exist and can be restored

✓ Data protection achieved

◆ 2 Cross-Region Replication (CRR)

🎯 Objective

Automatically copy objects from one region to another region.

◆ Prerequisites

✓ Versioning enabled on **both buckets**

✓ Buckets in **different regions**

✓ AWS Academy auto-creates permissions

◆ Step 1: Create Destination Bucket

1. Go to **S3** → **Create bucket**
 2. Bucket name: `dest-crr-bucket`
 3. Select **different region**
 4. Create bucket
 5. Enable **Versioning**
-

◆ Step 2: Create Replication Rule

1. Open **Source bucket**
 2. Go to **Management**
 3. Click **Replication rules**
 4. Click **Create replication rule**
-

◆ Step 3: Configure Rule

- Rule name: `academy-crr-rule`
 - Status: **Enabled**
 - Scope: **Apply to all objects**
-

◆ Step 4: Choose Destination

1. Select **Another bucket**
2. Choose destination bucket
3. Select **Use AWS-managed role**
4. Click **Save**

✓ CRR activated

◆ Output Verification

- Upload `image.png` to source bucket

- Open destination bucket
 - ✓ Object appears automatically
 - ✓ Version ID exists
 - 🎉 Replication successful
-

✓ PART 3: Static Website Hosting Using S3 (ACL Method)

🎯 Objective

To host a static website using **index.html** and **error.html**

◆ Step 1: Create Bucket

- Bucket name: `my-static-site-bucket-123`
 - Disable **Block Public Access**
 - Create bucket
-

◆ Step 2: Upload Website Files

Upload:

- `index.html`
 - `error.html`
-

◆ Step 3: Enable Static Website Hosting

1. Go to **Properties**
2. Scroll to **Static website hosting**
3. Click **Edit**
4. Enable
5. Enter:
 - Index document: `index.html`
 - Error document: `error.html`
6. Save changes

◆ Step 4: Enable Bucket-Level ACL

- Permissions → ACL → Everyone → ✓ Read

◆ Step 5: Enable Object-Level ACL

Repeat for **both files**:

- Permissions → ACL → Everyone → ✓ Read

◆ Step 6: Access Website

1. Copy **Bucket website endpoint**
2. Open in browser

✓ Website loads successfully

◆ Step 7: Test Error Page

- Open invalid URL

✓ `error.html` displayed

Lab 5

◆ STEP 1: Create a VPC (Your Own Private Network)

What is a VPC?

A **VPC (Virtual Private Cloud)** is like your **own private network** inside AWS.

Steps (DO THIS EXACTLY):

1. Login to **AWS Management Console**
2. Search **VPC** in the top search bar → open **VPC**
3. Click **Create VPC**
4. Select **VPC only**
5. Fill details:

- **Name tag** → Custom-VPC
- **IPv4 CIDR block** → 10.0.0.0/16

6. Leave everything else default

7. Click **Create VPC**

✓ **Checkpoint:**

You should see **Custom-VPC** listed with CIDR 10.0.0.0/16.

◆ STEP 2: Create Subnets (Public & Private)

Why Subnets?

- **Public Subnet** → Internet access
- **Private Subnet** → No direct internet

We'll create **2 subnets in SAME AZ**.

● Create Public Subnet

1. Go to **VPC** → **Subnets**
 2. Click **Create subnet**
 3. Select:
 - **VPC** → Custom-VPC
 - **Subnet name** → Public-Subnet
 - **Availability Zone** → e.g. ap-south-1a
 - **IPv4 CIDR** → 10.0.1.0/24
 4. Click **Create subnet**
-

● Create Private Subnet

1. Click **Create subnet** again
2. Select:
 - **VPC** → Custom-VPC
 - **Subnet name** → Private-Subnet

- **Availability Zone** → **same AZ** (`ap-south-1a`)
- **IPv4 CIDR** → `10.0.2.0/24`

3. Click **Create subnet**

✅ **Checkpoint:**

You must see **2 subnets**:

- **Public** → `10.0.1.0/24`
- **Private** → `10.0.2.0/24`

◆ **STEP 3: Create & Attach Internet Gateway (IGW)**

Why IGW?

Internet Gateway allows **public subnet** → **internet**.

Steps:

1. Go to **VPC** → **Internet Gateways**
2. Click **Create internet gateway**
3. Name → `Custom-IGW`
4. Click **Create**
5. Select the IGW
6. Click **Actions** → **Attach to VPC**
7. Select **Custom-VPC**
8. Click **Attach**

✅ **Checkpoint:**

IGW state should be **Attached**.

◆ **STEP 4: Create Route Tables**

● **Public Route Table**

1. Go to **VPC** → **Route Tables**
2. Click **Create route table**

3. Name → `Public-RT`
4. VPC → `Custom-VPC`
5. Click **Create**

Add Internet Route:

1. Select `Public-RT`
2. Go to **Routes** → **Edit routes**
3. Add route:
 - Destination → `0.0.0.0/0`
 - Target → **Internet Gateway (Custom-IGW)**
4. Save routes

Associate Public Subnet:

1. Go to **Subnet associations**
2. Click **Edit subnet associations**
3. Select **Public-Subnet**
4. Save

Private Route Table

1. Create route table:
 - Name → `Private-RT`
 - VPC → `Custom-VPC`
2. Do **NOT** add IGW route
3. Associate **Private-Subnet**

Checkpoint:

- Public subnet → has `0.0.0.0/0 → IGW`
- Private subnet → only local route

◆ **STEP 5: Enable Auto-Assign Public IP (IMPORTANT)**

1. Go to **VPC** → **Subnets**
2. Select **Public-Subnet**
3. Click **Actions** → **Edit subnet settings**
4. Enable ☒ **Auto-assign public IPv4 address**
5. Save

! If you skip this → EC2 won't get public IP

◆ STEP 6: Create Security Group

1. Go to **EC2** → **Security Groups**
2. Click **Create security group**
3. Name → **Web-SG**
4. VPC → **Custom-VPC**

Inbound Rules:

Type	Port	Source
SSH	22	My IP
HTTP	80	0.0.0.0/0
HTTPS	443	0.0.0.0/0

1. Save security group
-

◆ STEP 7: Launch EC2 Instances

● Public EC2

1. EC2 → Launch instance
2. AMI → Amazon Linux
3. Network → **Custom-VPC**
4. Subnet → **Public-Subnet**
5. Auto-assign IP → **Enable**
6. Security group → **Web-SG**

7. Launch

Private EC2

Same steps but:

- Subnet → **Private-Subnet**
- Auto-assign IP → **Disable**

◆ STEP 8: Test & Verify

Test	Expected
SSH to Public EC2	✓ Works
Internet from Public EC2	✓ Yes
SSH to Private EC2 directly	✗ No
Private EC2 via Bastion/NAT	✓ Yes

Lab 6

VPC USING BASTION SERVER — STEP BY STEP

(Platform: **Amazon Web Services**)

STEP 1: Create VPC

1. Open **AWS Console**
2. Go to **VPC** → **Create VPC**
3. Choose **VPC only**
4. Enter:
 - **Name:** **Bastion-VPC**
 - **IPv4 CIDR:** **10.0.0.0/16**
5. Click **Create VPC**

✓ VPC is created

STEP 2: Create Subnets

Public Subnet (Bastion)

1. VPC → **Subnets** → **Create subnet**
2. Select VPC: `Bastion-VPC`
3. Subnet name: `Public-Subnet`
4. CIDR: `10.0.1.0/24`
5. AZ: Any
6. Create subnet

Private Subnet (Database)

1. Create another subnet
2. Subnet name: `Private-Subnet`
3. CIDR: `10.0.2.0/24`
4. Create subnet

✓ Two subnets created

STEP 3: Create Internet Gateway

1. VPC → **Internet Gateways** → **Create**
2. Name: `Bastion-IGW`
3. Create
4. **Attach IGW** to `Bastion-VPC`

✓ Internet access enabled for public subnet

STEP 4: Configure Route Tables

Public Route Table

1. VPC → **Route Tables** → **Create**
2. Name: `Public-RT`
3. VPC: `Bastion-VPC`

4. Routes:

- `0.0.0.0/0` → **Internet Gateway**

5. **Associate** with `Public-Subnet`

Private Route Table

1. Create `Private-RT`
2. No internet route
3. Associate with `Private-Subnet`

✓ Only public subnet has internet

STEP 5: Create Security Groups (VERY IMPORTANT)

A. Bastion Security Group

1. EC2 → Security Groups → Create
 2. Name: `Bastion-SG`
 3. Inbound rule:
 - **SSH (22)** → **My IP**
 4. Outbound: Allow all
-

B. Database Security Group

1. Create `DB-SG`
2. Inbound rules:
 - **MySQL (3306)** → `10.0.1.0/24`
 - **SSH (22)** → **Private IP of Bastion**
3. ✗ No public IP access

✓ DB is fully protected

STEP 6: Create Key Pairs

1. EC2 → **Key Pairs**
2. Create:

- `bastion.pem`
- `dbserver.pem`

3. Download and store safely

STEP 7: Launch Bastion Server (Public Subnet)

1. EC2 → Launch Instance
2. OS: **Ubuntu**
3. Subnet: `Public-Subnet`
4. Auto-assign Public IP: **Enabled**
5. Security Group: `Bastion-SG`
6. Key pair: `bastion.pem`
7. Launch

✓ Bastion server is public

STEP 8: Launch Database Server (Private Subnet)

1. Launch another EC2
2. OS: **Ubuntu**
3. Subnet: `Private-Subnet`
4. Auto-assign Public IP: ✗ Disabled
5. Security Group: `DB-SG`
6. Key pair: `dbserver.pem`
7. Launch

✓ DB server is private (no public IP)

STEP 9: Copy DB Key to Bastion Server

(DB server cannot be accessed directly)

From Local System (PowerShell / Terminal):

```
scp -i bastion.pem dbserver.pem ubuntu@<BASTION_PUBLIC_IP>:/home/ubuntu/
```

✓ Key securely copied

STEP 10: Connect to Bastion Server

```
ssh -i bastion.pem ubuntu@<BASTION_PUBLIC_IP>
```

Verify:

```
ls
```

Output:

```
bastion.pem  
dbserver.pem
```

STEP 11: Set Permissions (IMPORTANT – COMMON PROBLEM)

If SSH fails later, run:

```
chmod 400 dbserver.pem
```

✓ Fixes permission issue

STEP 12: Connect from Bastion to DB Server

```
ssh -i dbserver.pem ubuntu@10.0.2.x
```

✓ Successfully connected to **Private DB Server**

STEP 13: Install MySQL (Problem Area You Mentioned)

Update Packages

```
sudo apt update
```

Install MySQL

```
sudo apt install mysql-server-y
```

Start & Enable MySQL

```
sudo systemctl start mysql  
sudo systemctl enable mysql
```

Check Status

```
sudo systemctl status mysql
```

✓ MySQL running successfully

SECURITY OUTCOME (FINAL CONFIRMATION)

- ✓ Database server **NOT** publicly accessible
- ✓ SSH to DB **ONLY** via Bastion
- ✓ MySQL accessible **only** from Bastion subnet
- ✓ External users **cannot** reach DB