

## Multi-Database Hosting on a Single Virtual Machine:

⌚ Objective: Build a secure, cost-effective, scalable system for hosting multiple client-specific databases (like PostgreSQL, MySQL, MongoDB etc.) using Docker on a single Ubuntu VM.

### 📁 Directory Structure:

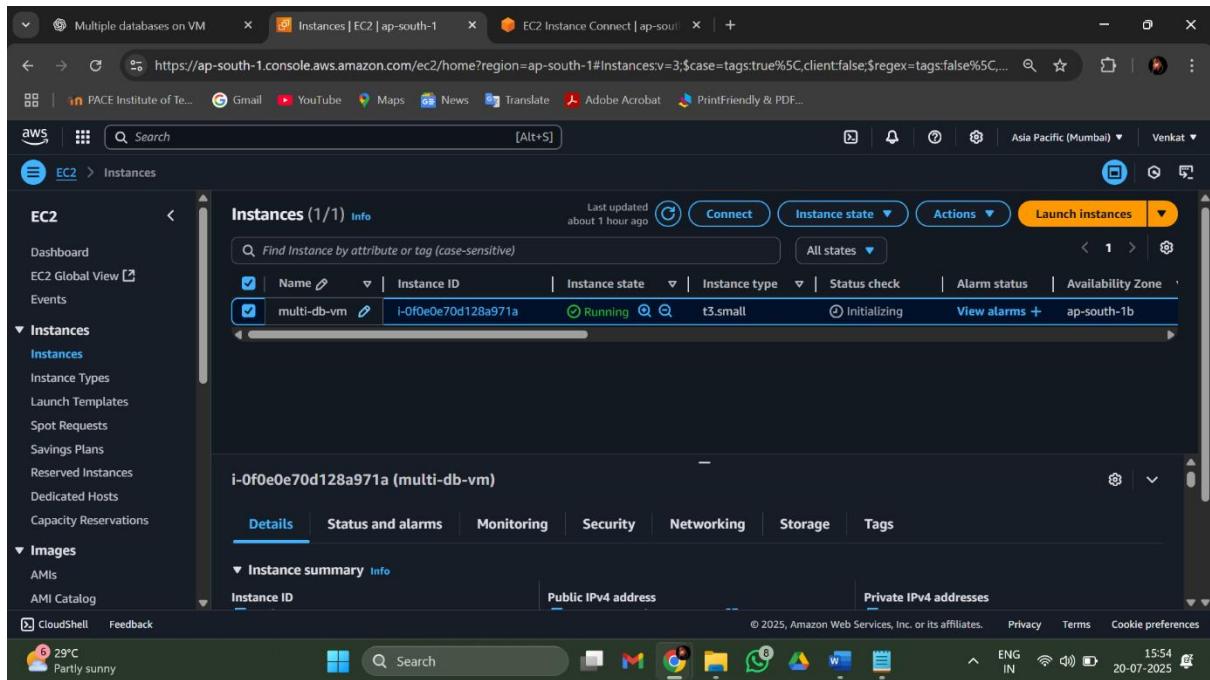
```
. └── multi-db-vm
    ├── backups
    │   ├── 20250720_0739
    │   │   ├── tcs.sql
    │   │   └── wipro.sql
    │   └── GenAILakes-postgres
    │       └── genailakesdb_20250720_105331.sql
    ├── clients
    │   ├── GenAILakes-postgres.yml
    │   ├── TCS-mongo.yml
    │   ├── TCS-mysql.yml
    │   ├── Wipro-postgres.yml
    │   ├── clients.json
    │   ├── cognizant-mongo.yml
    │   ├── infosys-db.yml
    │   ├── tcs-db.yml
    │   └── wipro-db.yml
    ├── docker-compose-template.yml
    └── scripts
        ├── add_new_client.sh
        ├── backup_all.sh
        ├── generate_clients_json.sh
        ├── list_clients.sh
        ├── remove_client.sh
        └── resize_volume.sh

```

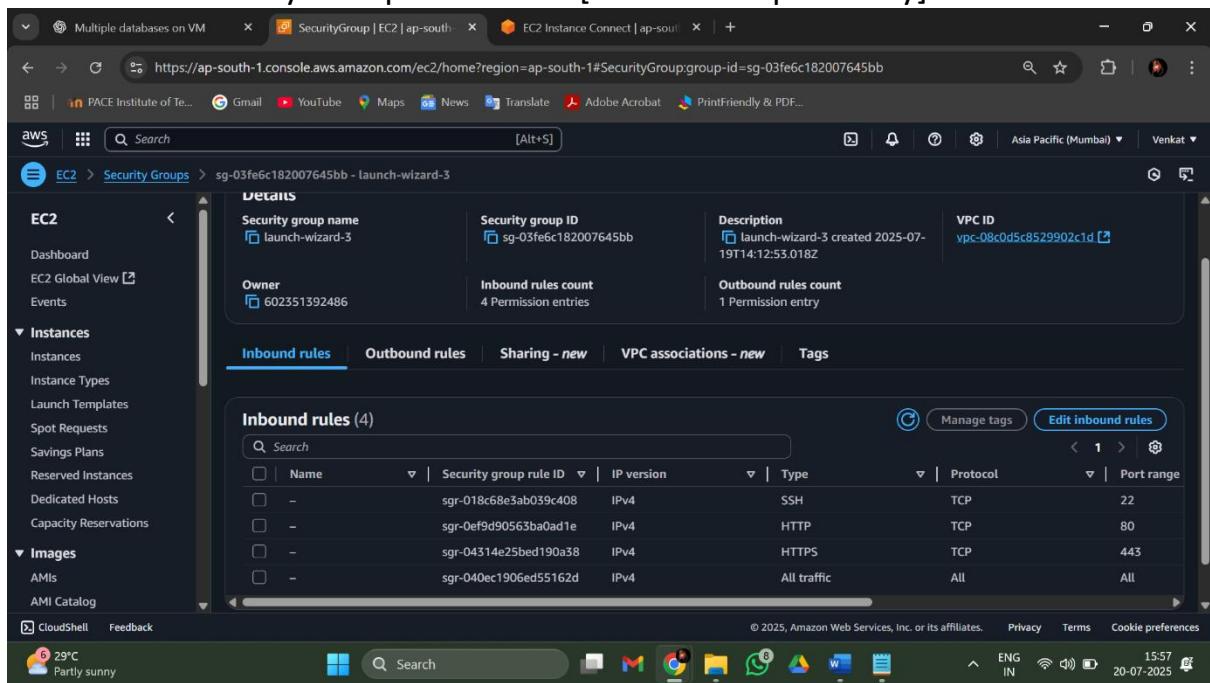
### ➔ Step-by-Step Build Process:

#### STEP 1: Install Prerequisites

- Ubuntu-based EC2 VM [[Launched t3.small](#)]
- Installed Docker and Docker Compose Packages.



→ Allowed Security Groups All Traffic [Practice Purpose only]:



→ # Install Docker

sudo apt update

sudo apt install -y docker.io docker-compose

sudo usermod -aG docker \$USER

newgrp docker

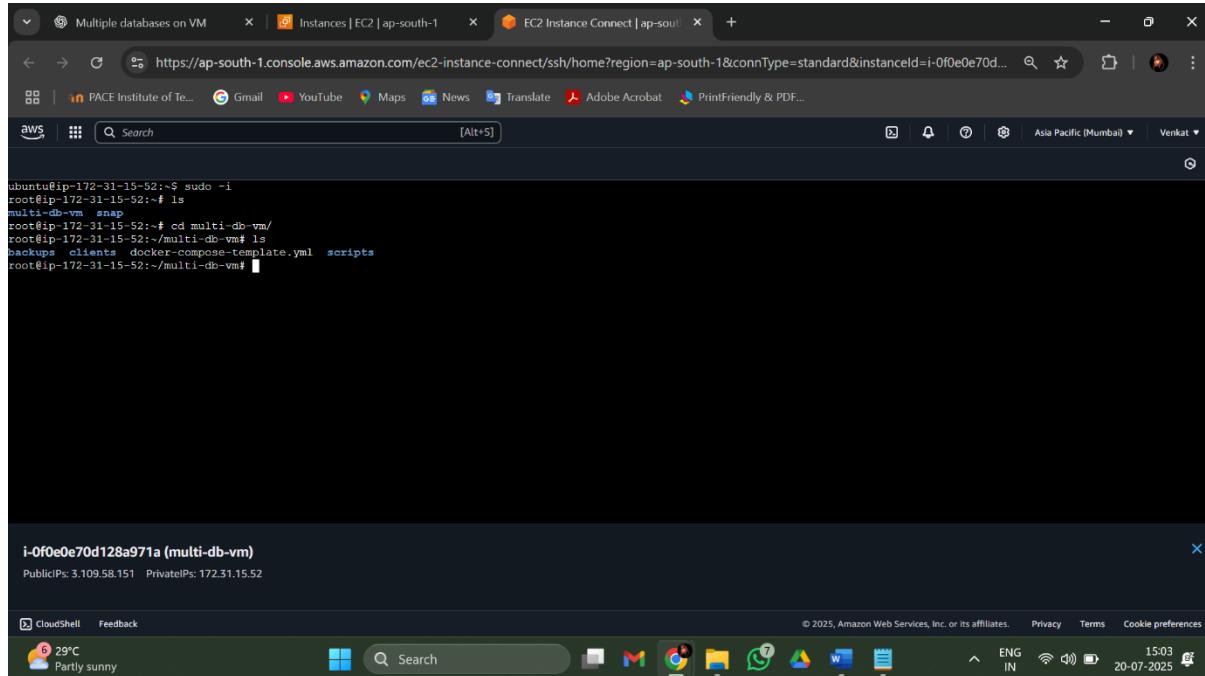
```
# Enable Docker
```

```
⇒ sudo systemctl enable docker
```

```
→ docker-compose up -d
```

```
→ docker ps -a [It shows all the Containers Up & Excited ]
```

```
→ mkdir -p ~/multi-db-vm/{scripts,clients,backups}  
cd ~/multi-db-vm
```



```
ubuntu@ip-172-31-15-52:~$ sudo -i  
root@ip-172-31-15-52:~# ls  
multi-db-vm snap  
root@ip-172-31-15-52:~# cd multi-db-vm/  
root@ip-172-31-15-52:~/multi-db-vm# ls  
backups clients docker-compose-template.yml scripts  
root@ip-172-31-15-52:~/multi-db-vm#
```

```
→ cd multi-db-vm
```

```
⇒ Vi docker-compose-template.yml
```

```
--  
version: '3.8'  
  
services:  
  ${CLIENT_NAME}-db:  
    image: ${DB_IMAGE}  
    container_name: ${CLIENT_NAME}-db  
    restart: always  
    environment:  
      ${PASSWORD_ENV_VAR}: ${ROOT_PASSWORD}  
    volumes:  
      - ${CLIENT_NAME}_db_data:${VOLUME_PATH}  
    ports:
```

```
- "${PORT}:${DB_PORT}"
networks:
- db_net
```

```
volumes:
${CLIENT_NAME}_db_data:
```

```
networks:
db_net:
driver: bridge
```

## STEP 2: vi add\_new\_client.sh

→ cd multi-db-vm/scripts/add\_new\_client.sh

```
#!/bin/bash
```

```
echo "🔧 Add New Client Database Setup"
```

```
# Prompt for client details
read -p "Enter client name: " client_name
read -p "Enter database type (postgres/mysql/mongo): " db_type
```

```
db_type=$(echo "$db_type" | tr '[:upper:]' '[:lower:]')
```

```
case "$db_type" in
postgres)
default_port=5432
;;
mysql)
default_port=3306
;;
mongo)
default_port=27017
;;
*)
echo "Unsupported database type: $db_type"
exit 1
;;
esac
```

```
# Read user-defined DB config
read -p "Enter database name: " db_name
read -p "Enter database user: " db_user
read -s -p "Enter database password: " db_password
echo
read -p "Enter exposed port on host (default $default_port): " port
port=${port:-$default_port}

# Compose values
container_name="${client_name}-${db_type}"
volume_name="${client_name}-${db_type}-volume"
yml_path=../clients/${container_name}.yml
network_name="clients_net"

# Create network if not exists
docker network inspect $network_name >/dev/null 2>&1 || docker network
create $network_name

# Create Docker Compose YAML
echo "Generating Compose file at: $yml_path"

cat > "$yml_path" <<EOF
version: '3.8'

services:
${container_name}:
  image: ${db_type}
  container_name: ${container_name}
  restart: unless-stopped
  environment:
EOF

# Set DB-specific environment
case "$db_type" in
postgres)
  cat >> "$yml_path" <<EOF
    POSTGRES_DB: ${db_name}
    POSTGRES_USER: ${db_user}
    POSTGRES_PASSWORD: ${db_password}
  EOF
```

```
    ports:
      - "${port}:5432"
EOF
;;
mysql)
cat >> "$yml_path" <<EOF
  MYSQL_DATABASE: ${db_name}
  MYSQL_USER: ${db_user}
  MYSQL_PASSWORD: ${db_password}
  MYSQL_ROOT_PASSWORD: ${db_password}
ports:
  - "${port}:3306"
EOF
;;
mongo)
cat >> "$yml_path" <<EOF
  MONGO_INITDB_DATABASE: ${db_name}
  MONGO_INITDB_ROOT_USERNAME: ${db_user}
  MONGO_INITDB_ROOT_PASSWORD: ${db_password}
ports:
  - "${port}:27017"
EOF
;;
esac
```

```
# Finish YAML
cat >> "$yml_path" <<EOF
  volumes:
    - ${volume_name}:/data/db
  networks:
    - ${network_name}
```

```
volumes:
  ${volume_name}:
```

```
networks:
  ${network_name}:
    external: true
EOF
```

```

# Launch the container
echo "Deploying $db_type container for $client_name..."
docker-compose -f "$yaml_path" up -d

# Update clients.json
CLIENTS_JSON_PATH=../clients/clients.json"

# Create JSON file if it doesn't exist
if [ ! -f "$CLIENTS_JSON_PATH" ]; then
  echo "[]" > "$CLIENTS_JSON_PATH"
fi

# Remove existing entry for same client
TMP_JSON=$(mktemp)
jq "del(.[] | select(.name == \"$client_name\"))" "$CLIENTS_JSON_PATH" >
"$TMP_JSON" && mv "$TMP_JSON" "$CLIENTS_JSON_PATH"

# Append new client data
jq \
  --arg name "$client_name" \
  --arg type "$db_type" \
  --argjson port "$port" \
  --arg db_name "$db_name" \
  --arg db_user "$db_user" \
  --arg db_password "$db_password" \
  '. += [{name: $name, type: $type, port: $port, db_name: $db_name, db_user: $db_user, db_password: $db_password}]' \
"$CLIENTS_JSON_PATH" > "$TMP_JSON" && mv "$TMP_JSON" "$CLIENTS_JSON_PATH"

echo "$client_name $db_type database added and running!"
echo "Tracked in: $CLIENTS_JSON_PATH"

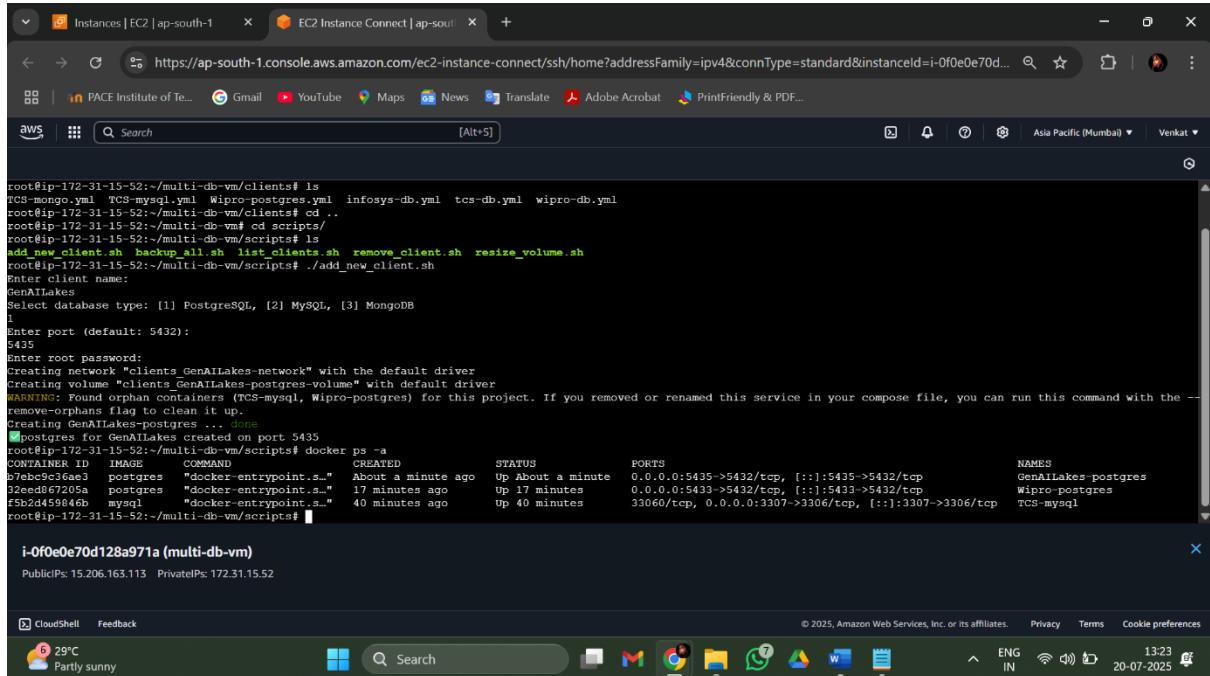
```

⇒ **chmod +x add\_new\_client.sh**  
 ⇒ **./add\_new\_client.sh**

#### → **add\_new\_client.sh** it Support:

- **Client name**
- **Database type** (postgres, mysql, mongo)

- Custom database name, username, and password
- Custom port
- Dynamically generates the correct Docker Compose file
- Tracks all clients in clients/clients.json



The screenshot shows the AWS CloudShell interface. The terminal window displays the following command sequence:

```

root@ip-172-31-15-52:~/multi-db-vm# ls
TCS-mongo.yml  TCS-mysql.yml  Wipro-postgres.yml  infosys-db.yml  tcs-db.yml  wipro-db.yml
root@ip-172-31-15-52:~/multi-db-vm/client$ cd ..
root@ip-172-31-15-52:~/multi-db-vm$ cd scripts/
root@ip-172-31-15-52:~/multi-db-vm/scripts$ ls
add_new_client.sh  backup_all.sh  list_clients.sh  remove_client.sh  resize_volume.sh
root@ip-172-31-15-52:~/multi-db-vm/scripts$ ./add_new_client.sh
Enter client name:
GenAllLakes
Select database type: [1] PostgreSQL, [2] MySQL, [3] MongoDB
1
Enter port (default: 5432):
5435
Enter root password:
Creating network "clients_GenAllLakes-network" with the default driver
Creating volume "clients_GenAllLakes-postgres-volume" with default driver
WARNING: Found orphan containers (TCS-mysql, Wipro-postgres) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphan flag to clean it up.
Creating GenAllLakes-postgres ...
  postgres
  postgres for GenAllLakes created on port 5435
root@ip-172-31-15-52:~/multi-db-vm/scripts$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
b7ebc9c36ae3        postgres            "docker-entrypoint.s..."   About a minute ago   Up About a minute   0.0.0.0:5435->5432/tcp, [::]:5435->5432/tcp
32eed867205a        postgres            "docker-entrypoint.s..."   17 minutes ago     Up 17 minutes      0.0.0.0:5433->5432/tcp, [::]:5433->5432/tcp
f5b2d459846b        mysql              "docker-entrypoint.s..."   40 minutes ago     Up 40 minutes      33060/tcp, 0.0.0.0:3307->3306/tcp, [::]:3307->3306/tcp
root@ip-172-31-15-52:~/multi-db-vm/scripts$ 

```

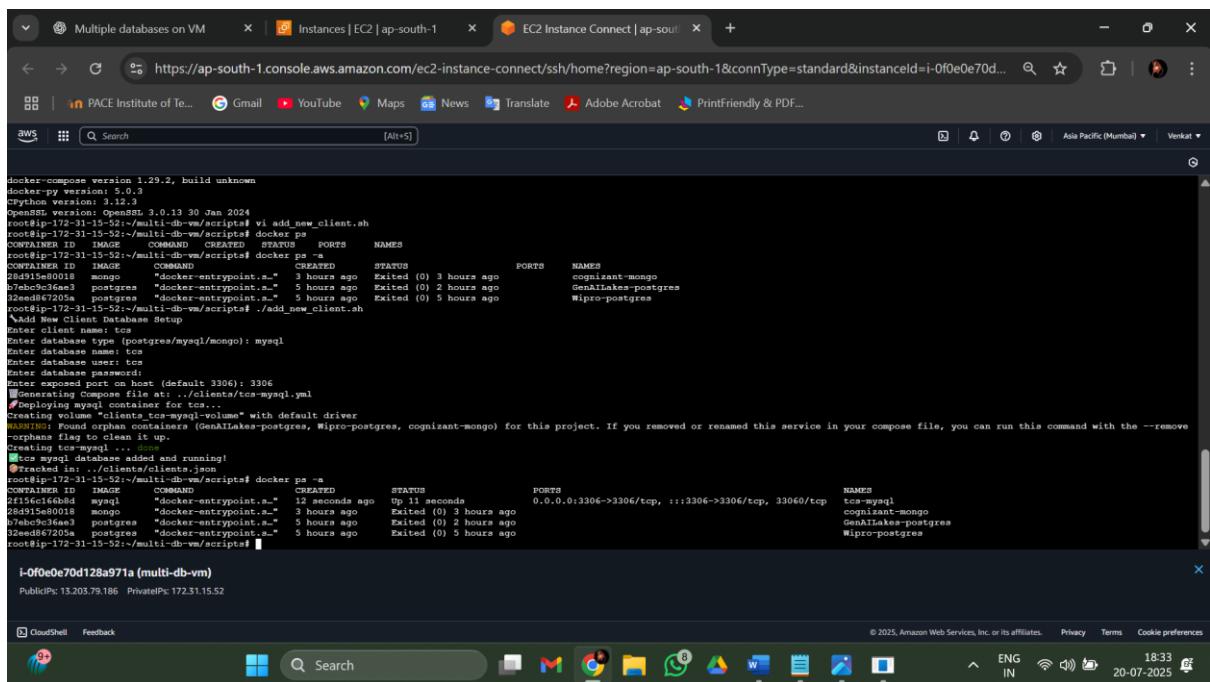
At the bottom of the terminal, it shows:

```

i-0f0e0e70d128a971a (multi-db-vm)
PublicIPs: 15.206.163.13 PrivateIPs: 172.31.15.52

```

The browser tab bar includes: Instances | EC2 | ap-south-1, EC2 Instance Connect | ap-south-1, and https://ap-south-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?addressFamily=ipv4&connType=standard&instanceId=i-0f0e0e70d... . The status bar at the bottom right shows: © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 13:23 20-07-2025.



The screenshot shows the AWS CloudShell interface. The terminal window displays the following command sequence:

```

docker-compose version 1.29.2, build unknown
docker-py version: 5.0.3
CPython version: 3.12.3
OpenSSL version: 1.1.13 20 Jan 2024
root@ip-172-31-15-52:~/multi-db-vm/scripts$ vi add_new_client.sh
root@ip-172-31-15-52:~/multi-db-vm/scripts$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
root@ip-172-31-15-52:~/multi-db-vm/scripts$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
root@ip-172-31-15-52:~/multi-db-vm/scripts$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
28d915e00018        mongo              "docker-entrypoint.s..."   3 hours ago       Exited (0) 3 hours ago   cognizant-mongo
b7ebc9c36ae3        postgres            "docker-entrypoint.s..."   5 hours ago       Exited (0) 2 hours ago   GenAllLakes-postgres
32eed867205a        postgres            "docker-entrypoint.s..."   5 hours ago       Exited (0) 5 hours ago   Wipro-postgres
root@ip-172-31-15-52:~/multi-db-vm/scripts$ ./add_new_client.sh
>Add New Client Database Setup
Enter client name: tcs
Enter database type: (postgres/mysql/mongo): mysql
Enter database name: tcs
Enter database user: tcs
Enter database password:
Enter exposed port on host (default: 3306): 3306
Deploying mysql container for tcs...
Creating volume "clients_tcs-mysql-volume" with default driver
WARNING: Found orphan containers (GenAllLakes-postgres, Wipro-postgres, cognizant-mongo) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphan flag to clean it up.
Create tcs database
  tcs
  tcs mysql database added and running!


At the bottom of the terminal, it shows:



```

i-0f0e0e70d128a971a (multi-db-vm)
PublicIPs: 13.203.79.186 PrivateIPs: 172.31.15.52

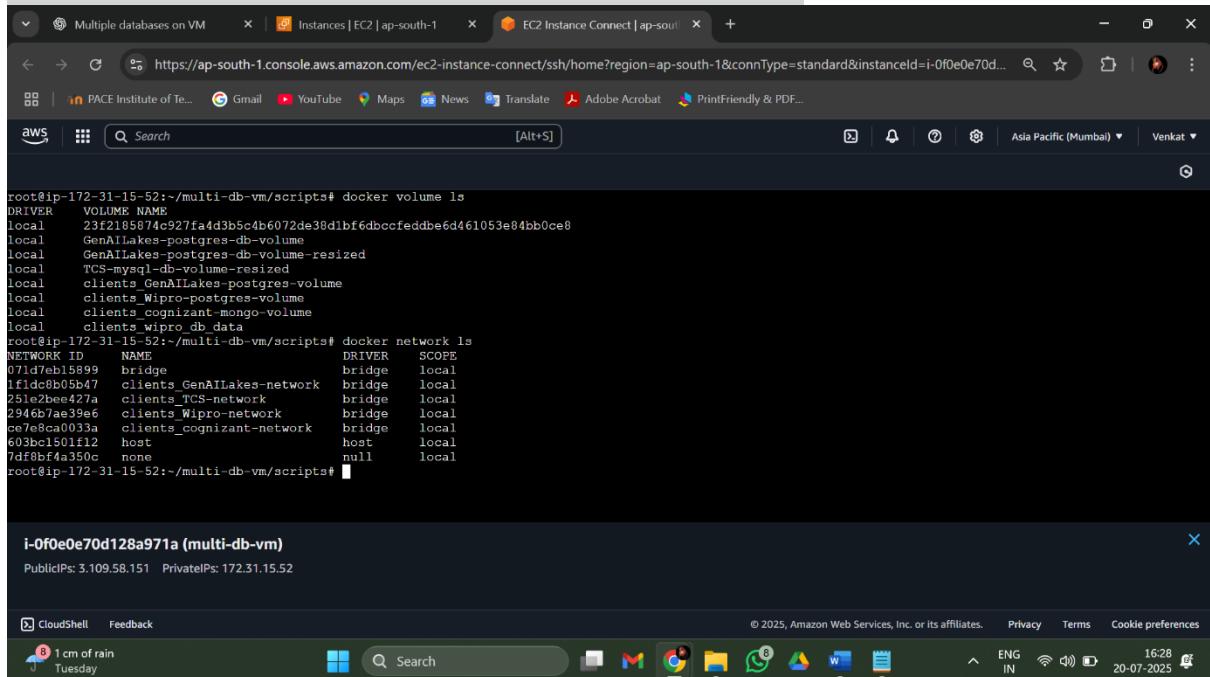
```



The browser tab bar includes: Instances | EC2 | ap-south-1, EC2 Instance Connect | ap-south-1, and https://ap-south-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?region=ap-south-1&connType=standard&instanceId=i-0f0e0e70d... . The status bar at the bottom right shows: © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 18:33 20-07-2025.


```

## Docker Volumes and Networks Created:



The screenshot shows a CloudShell terminal window with the following content:

```
root@ip-172-31-15-52:~/multi-db-vm/scripts# docker volume ls
DRIVER      VOLUME NAME
local       23f2185874c927fa4d3b5c4b6072de38d1bf6dbccfeddbe6d461053e84bb0ce8
local       GenAI Lakes-postgres-db-volume
local       GenAI Lakes-postgres-db-volume-resized
local       TCS-mysql-db-volume-resized
local       clients_GenAI Lakes-postgres-volume
local       clients_Wipro-postgres-volume
local       clients_cognizant-mongo-volume
local       clients_wipro_db_data
root@ip-172-31-15-52:~/multi-db-vm/scripts# docker network ls
NETWORK ID     NAME          DRIVER      SCOPE
071d7eb1599   bridge        bridge      local
ff1dc8b05b47  clients_GenAI Lakes-network  bridge      local
25le2bee427a  clients_TCS-network  bridge      local
2946b7ae39e6  clients_Wipro-network  bridge      local
ce7e8ca0033a  clients_cognizant-network  bridge      local
603bc1501f12  host          host       local
7df8bf4a350c  none          null       local
root@ip-172-31-15-52:~/multi-db-vm/scripts#
```

Below the terminal, a message box displays:

**i-0f0e0e70d128a971a (multi-db-vm)**  
PublicIPs: 3.109.58.151 PrivateIPs: 172.31.15.52

The CloudShell interface includes a toolbar with icons for CloudShell, Feedback, Search, and various services like Gmail, Google Sheets, and Google Slides. The status bar at the bottom shows the date (Tuesday), time (16:28), and location (ENG IN).

## Database Access Example:

```
sudo apt install yamllint
yamllint .../clients/tcs-db.yml
```

```
apt update
apt install docker-compose-plugin
```

## Test the Databases:

```
# Example: TCS wants MySQL on port 3307
./add_new_client.sh tcs mysql 3307
```

```
# Example: Wipro wants PostgreSQL on port 5433
./add_new_client.sh wipro postgres 5433
```

## Access Example Table

DB Type	Host	Port	Command Example
PostgreSQL	127.0.0.1	5433	psql ...
MySQL	127.0.0.1	3307	mysql ...
MongoDB	127.0.0.1	27018	mongo ...

→ docker ps -a

### Connect to the database to test:

```
mysql -h 127.0.0.1 -P 3307 -u root -p  
sudo apt install mysql-client -y
```

```
psql -h 127.0.0.1 -p 5433 -U postgres  
sudo apt install postgresql-client -y
```

```
mongo -h 127.0.0.1 -p 27018 -u root -p  
sudo apt install mongodb-client -y
```

### → Step 3: Resize a Volume:

Functionality:

- Creates a new volume
- Copies existing data
- Stops container
- Restarts container with new volume
- You can check in **df -h**

⇒ vi resize\_volume.sh

```
#!/bin/bash
```

```
read -p "Enter client name: " CLIENT_NAME  
read -p "Enter database type (postgres, mysql, mongo): " DB_TYPE
```

```
ORIGINAL_VOLUME="${CLIENT_NAME}-${DB_TYPE}-db-volume"  
RESIZED_VOLUME="${CLIENT_NAME}-${DB_TYPE}-db-volume-resized"  
CONTAINER_NAME="${CLIENT_NAME}-${DB_TYPE}"  
COMPOSE_FILE="..../clients/${CLIENT_NAME}-${DB_TYPE}.yml"
```

```
echo "📁 Original volume: $ORIGINAL_VOLUME"
echo "🆕 New volume: $RESIZED_VOLUME"
echo "⚙️ Container using volume: $CONTAINER_NAME"

read -p "Proceed with resizing volume for $CLIENT_NAME? (y/n): "
CONFIRM
if [ "$CONFIRM" != "y" ]; then
    echo "✗ Aborted."
    exit 1
fi

# Create new volume
docker volume create "$RESIZED_VOLUME"
echo "✓ New volume $RESIZED_VOLUME created."

# Find mount path from old volume
OLD_MOUNT=$(docker inspect --format '{{ range .Mounts }}{{ .Source }}{{ end }}' "$CONTAINER_NAME")

# Copy data
docker run --rm -v "$ORIGINAL_VOLUME":/from -v
"$RESIZED_VOLUME":/to alpine ash -c "cd /from && cp -av . /to"
echo "📂 Data copied from $ORIGINAL_VOLUME to $RESIZED_VOLUME."

# Stop and remove container
docker stop "$CONTAINER_NAME"
docker rm "$CONTAINER_NAME"
echo "🛑 Stopped and removed container $CONTAINER_NAME."

# Update compose file
if [ -f "$COMPOSE_FILE" ]; then
    sed -i "s/${ORIGINAL_VOLUME}/${RESIZED_VOLUME}/g"
"$COMPOSE_FILE"
    echo "📝 Updated $COMPOSE_FILE to use resized volume."
else
    echo "⚠️ $COMPOSE_FILE not found. Skipping update."
fi

# Restart container
```

```
docker compose -f "$COMPOSE_FILE" up -d
echo "✓ Container $CONTAINER_NAME restarted with resized volume."
```

⇒ chmod +x resize\_volume.sh  
⇒ ./resize\_volume.sh

## →Step 4: Take Database Backup:

Supports:

- pg\_dump for PostgreSQL
- mysqldump for MySQL
- mongodump MongoDB

Saved to: logs/backups/<client>/<timestamp>.gz

⇒ vi backup\_all.sh

```
#!/bin/bash
```

```
set -e
JSON_FILE="../clients/clients.json"
```

```
if [ ! -f "$JSON_FILE" ]; then
    echo "✗ clients.json not found. Please run add_new_client.sh first."
    exit 1
fi
```

```
echo "Enter client name to back up:"
read CLIENT_NAME
```

```
# Extract client data using jq
CLIENT_DATA=$(jq -r --arg name "$CLIENT_NAME" '.[] | select(.name ==
$name)' "$JSON_FILE")
```

```
if [ -z "$CLIENT_DATA" ]; then
    echo "✗ Client '$CLIENT_NAME' not found in clients.json."
    exit 1
fi
```

```
DB_TYPE=$(echo "$CLIENT_DATA" | jq -r '.type')
PORT=$(echo "$CLIENT_DATA" | jq -r '.port')
DB_NAME=$(echo "$CLIENT_DATA" | jq -r '.db_name')
```

```
DB_USER=$(echo "$CLIENT_DATA" | jq -r '.db_user')
DB_PASSWORD=$(echo "$CLIENT_DATA" | jq -r '.db_password')
TIMESTAMP=$(date +"%Y%m%d_%H%M%S")
BACKUP_DIR="../backups/${CLIENT_NAME}"
mkdir -p "$BACKUP_DIR"
```

```
echo "📦 Backing up $DB_TYPE database for $CLIENT_NAME..."
```

```
case "$DB_TYPE" in
  postgres)
    PGPASSWORD="$DB_PASSWORD" pg_dump -h 127.0.0.1 -p "$PORT" -U
"$DB_USER" "$DB_NAME" > "$BACKUP_DIR/${DB_NAME}_${TIMESTAMP}.sql"
    echo "✓ PostgreSQL backup saved to $BACKUP_DIR"
    ;;
  mysql)
    mysqldump -h 127.0.0.1 -P "$PORT" -u "$DB_USER" -p"$DB_PASSWORD"
"$DB_NAME" > "$BACKUP_DIR/${DB_NAME}_${TIMESTAMP}.sql"
    echo "✓ MySQL backup saved to $BACKUP_DIR"
    ;;
  mongodb)
    mongodump --host 127.0.0.1 --port "$PORT" --username "$DB_USER" --
password "$DB_PASSWORD" --db "$DB_NAME" --out
"$BACKUP_DIR/mongo_${TIMESTAMP}"
    echo "✓ MongoDB backup saved to $BACKUP_DIR"
    ;;
  *)
    echo "✗ Unsupported DB type: $DB_TYPE"
    exit 1
    ;;
esac
```

```
⇒ chmod +x backup_all.sh
⇒ ./backup_all.sh
```

**Backups will be organized like this:**

```
multi-db-vm/
├── backups/
│   ├── TCS-mysql/
│   │   └── tcs_db_20250720_145511.sql
│   ├── Wipro-postgres/
│   │   └── wipro_db_20250720_145800.sql
│   └── Cognizant-mongo/
        └── mongo_20250720_150012/
```

## → **STEP 5: Remove a Client**

Actions:

- Stops and removes container
- Deletes volume
- Deletes Compose file
- Removes network
- Updates clients.json

⇒ vi remove\_client.sh

```
#!/bin/bash
```

```
echo "Enter client name:"
read CLIENT_NAME
```

```
echo "Enter database type (postgres/mysql/mongo):"
read DB_TYPE
```

```
COMPOSE_FILE=..../clients/${CLIENT_NAME}-${DB_TYPE}.yml
VOLUME_NAME="${CLIENT_NAME}-${DB_TYPE}-volume"
NETWORK_NAME="${CLIENT_NAME}-network"
CONTAINER_NAME="${CLIENT_NAME}-${DB_TYPE}"
```

```
echo "Stopping and removing container: $CONTAINER_NAME"
docker rm -f $CONTAINER_NAME
```

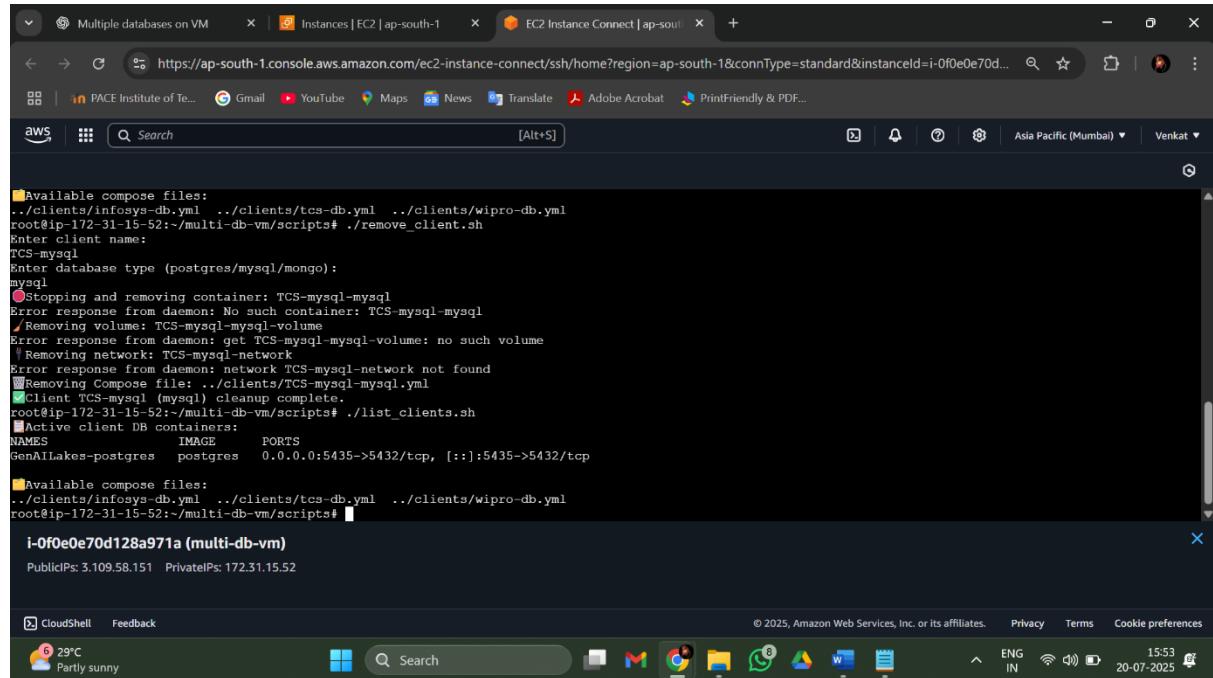
```
echo "✖ Removing volume: $VOLUME_NAME"
docker volume rm $VOLUME_NAME
```

```
echo "🔧 Removing network: $NETWORK_NAME"
docker network rm $NETWORK_NAME
```

```
echo "🌐 Removing Compose file: $COMPOSE_FILE"
rm -f "$COMPOSE_FILE"
```

```
echo "✓ Client $CLIENT_NAME ($DB_TYPE) cleanup complete."
```

- **chmod +x remove\_client.sh**
- **./remove\_client.sh**



```
Available compose files:
./clients/infosys-db.yml ..../clients/tcs-db.yml ..../clients/wipro-db.yml
root@ip-172-31-15-52:~/multi-db-vm/scripts# ./remove_client.sh
Enter client name:
TCS-mysql
Enter database type (postgres/mysql/mongo):
mysql
Stopping and removing container: TCS-mysql-mysql
Error response from daemon: No such container: TCS-mysql-mysql
✖ Removing volume: TCS-mysql-mysql-volume
Error response from daemon: get TCS-mysql-mysql-volume: no such volume
✖ Removing network: TCS-mysql-network
Error response from daemon: network TCS-mysql-network not found
✖ Removing Compose file: ..../clients/TCS-mysql-mysql.yml
✓ Client TCS-mysql (mysql) cleanup complete.
root@ip-172-31-15-52:~/multi-db-vm/scripts# ./list_clients.sh
Active client DB containers:
NAME      IMAGE      PORTS
GenAI Lakes-postgres  postgres  0.0.0.0:5435->5432/tcp, [::]:5435->5432/tcp

Available compose files:
./clients/infosys-db.yml ..../clients/tcs-db.yml ..../clients/wipro-db.yml
root@ip-172-31-15-52:~/multi-db-vm/scripts#
```

## → **STEP 6: Lists all the Clients:**

⇒ vi list\_clients.sh

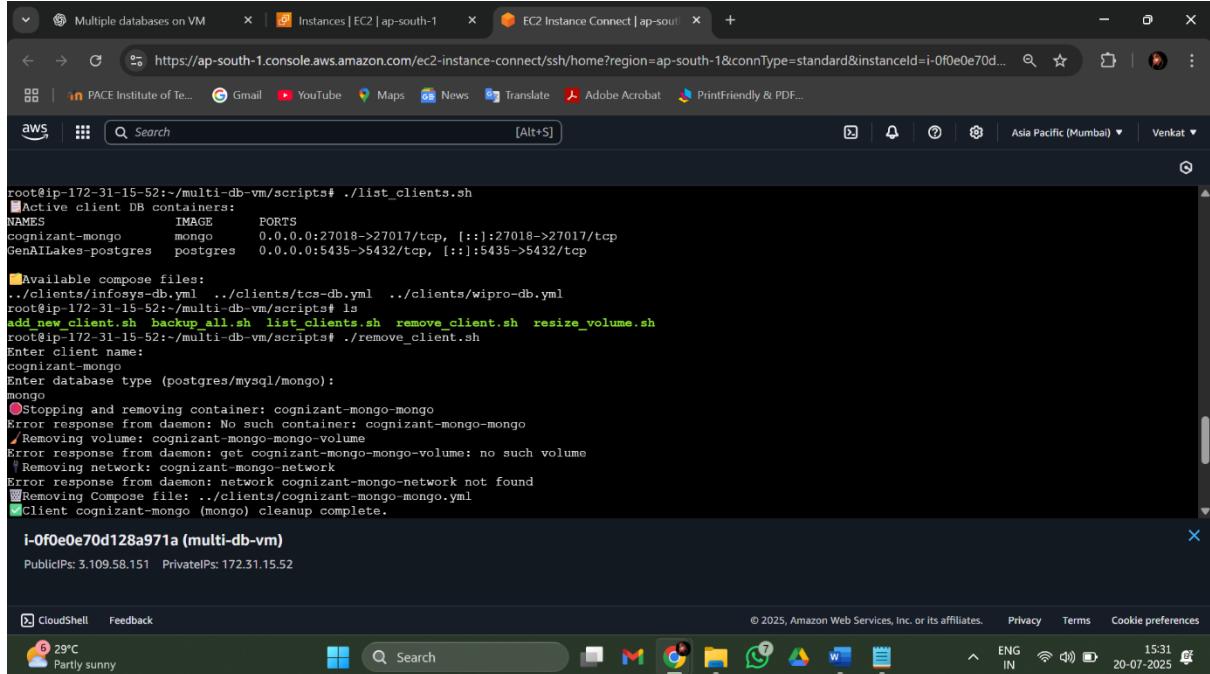
```
#!/bin/bash
```

```
echo "📋 Active client DB containers:"
docker ps --format "table {{.Names}}\t{{.Image}}\t{{.Ports}}"
```

```
echo -e "\n📋 Available compose files:"
```

```
ls ../*.yml 2>/dev/null || echo "No client YAMLs found."
```

⇒ chmod +x list\_clients.sh  
⇒ ./list\_clients.sh



```
root@ip-172-31-15-52:~/multi-db-vm/scripts# ./list_clients.sh
Active client DB containers:
NAME          IMAGE      PORTS
cognizant-mongo    mongo    0.0.0.0:27018->27017/tcp, [::]:27018->27017/tcp
GenAI Lakes-postgres postgres  0.0.0.0:5435->5432/tcp, [::]:5435->5432/tcp

Available compose files:
./clients/infosys-db.yml ./clients/tcs-db.yml ./clients/wipro-db.yml
root@ip-172-31-15-52:~/multi-db-vm/scripts# ls
add_new_client.sh backup_all.sh list_clients.sh remove_client.sh resize_volume.sh
root@ip-172-31-15-52:~/multi-db-vm/scripts# ./remove_client.sh
Enter client name:
cognizant-mongo
Enter database type (postgres/mysql/mongo):
mongo
Stopping and removing container: cognizant-mongo-mongo
Error response from daemon: No such container: cognizant-mongo-mongo
✖ Removing volume: cognizant-mongo-mongo-volume
Error response from daemon: get cognizant-mongo-mongo-volume: no such volume
✖ Removing network: cognizant-mongo-network
Error response from daemon: network cognizant-mongo-network not found
✖ Removing Compose file: ./clients/cognizant-mongo-mongo.yml
Client cognizant-mongo (mongo) cleanup complete.

i-0f0e0e70d128a971a (multi-db-vm)
PublicIPs: 3.109.58.151 PrivateIPs: 172.31.15.52

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences
CloudShell 29°C Partly sunny 20-07-2025 ENG IN 15:31
```

## → STEP 7: Generates Clients client.json file:

⇒ vi generate\_clients\_json.sh

```
#!/bin/bash
```

```
CLIENTS_DIR="../.clients"
OUTPUT_FILE="$CLIENTS_DIR/clients.json"
CLIENT_ENTRIES=()
```

```
echo "⌚ Scanning $CLIENTS_DIR for *.yml files..."
```

```
for file in "$CLIENTS_DIR"/*.yml; do
    filename=$(basename "$file" .yml)
    name="$filename"
```

```
# Auto-detect database type based on name
if [[ "$name" == *"postgres"* ]]; then
```

```
db_type="postgres"
port_guess=5432
user="postgres"
elif [[ "$name" == *"mysql"* ]]; then
    db_type="mysql"
    port_guess=3306
    user="root"
elif [[ "$name" == *"mongo"* ]]; then
    db_type="mongodb"
    port_guess=27017
    user="root"
else
    echo "⚠ Could not determine DB type for $name, skipping..."
    continue
fi
```

```
# Generate a guessed port to avoid conflicts (5432 + random)
port=$((port_guess + RANDOM % 100))
```

```
# Strip known vendor name to use for DB name (just a guess)
clean_name=$(echo "$name" | sed -E 's/[-_](postgres|mysql|mongo)//g' |
tr '[:upper:]' '[:lower:]')
```

```
# Build JSON entry
CLIENT_ENTRY=$(jq -n \
--arg name "$name" \
--arg type "$db_type" \
--argjson port "$port" \
--arg db_name "${clean_name}db" \
--arg db_user "$user" \
--arg db_password "changeme" \
'{name: $name, type: $type, port: $port, db_name: $db_name, db_user: $db_user, db_password: $db_password}')
)
```

```
CLIENT_ENTRIES+=("$CLIENT_ENTRY")
done
```

```
# Output to JSON array
jq -s '.' <<< "${CLIENT_ENTRIES[*]}" > "$OUTPUT_FILE"
```

```

echo "✓ Generated clients.json at $OUTPUT_FILE"
⇒ chmod +x generate_clients_json.sh
⇒ ./generate_clients_json.sh
⇒ cat ..../clients/clients.json | jq [You can View]

```

## Features:

- ✓ Multiple Database support
- ✓ Isolated container per client
- ✓ Port/volume/network automation
- ✓ Dynamic database name, user, and password input
- ✓ Dynamic port assignment
- ✓ Safe volume resizing
- ✓ Backup per engine
- ✓ Client removal automation
- ✓ Docker logs
- ✓ Auto-update of clients.json for tracking
- ✓ Can scale to 30+ clients per VM (based on specs)

Comparison Table				
Approach	Security	Isolation	Cost	Management
Single DB Engine, Multiple DBs	Moderate	Shared process	Low	Easy
Dockerized DBs	High	Full container isolation	Low	Moderate
Multi-Tenant Schema	Depends on implementation	Shared schema	Very Low	Complex

## Why this is BEST for You:

Requirement	Docker-based DB Isolation	🔗
💡 Cost-effective	One VM, multiple isolated DBs	
🔒 Secure per-client data	Each client runs in its own container (no data leakage)	
⚙️ Easy to automate	Can use <b>scripts + CI/CD (Jenkins, GitHub Actions)</b> to add DBs	
✳️ Tech flexibility	You can use <b>any DB type per client</b> (MySQL, PostgreSQL, MongoDB, etc.)	
📦 Scalable	New client = new container. Very fast setup	
📅 Backup & Monitoring ready	Easy to add auto backup + Grafana/Prometheus	
👨‍💻 DevOps-friendly	Perfect for DevOps engineers like you!	

## Security Tips

- Use UFW to block unwanted ports
- Avoid hardcoded passwords
- Use encrypted volumes
- Limit Docker network scope per client

## Ideal For:

- SaaS platforms
- Dev/test DB environments
- DB-as-a-Service (DBaaS)

## Suitable Use Cases

- ♦️ Startups & agencies hosting test/staging DBs for multiple clients.
- ♦️ Freelancers managing separate DBs for multiple projects.
- ♦️ Internal tools, POCs, small-scale multi-tenant applications.

## Not Suitable For

- ✗ Enterprises needing **high availability**, replication, and **auto-scaling**.
- ✗ Projects with **high compliance requirements** (like banking or healthcare).
- ✗ Apps needing **cloud-native DB features** (like AWS RDS snapshot, IAM auth, etc.)

