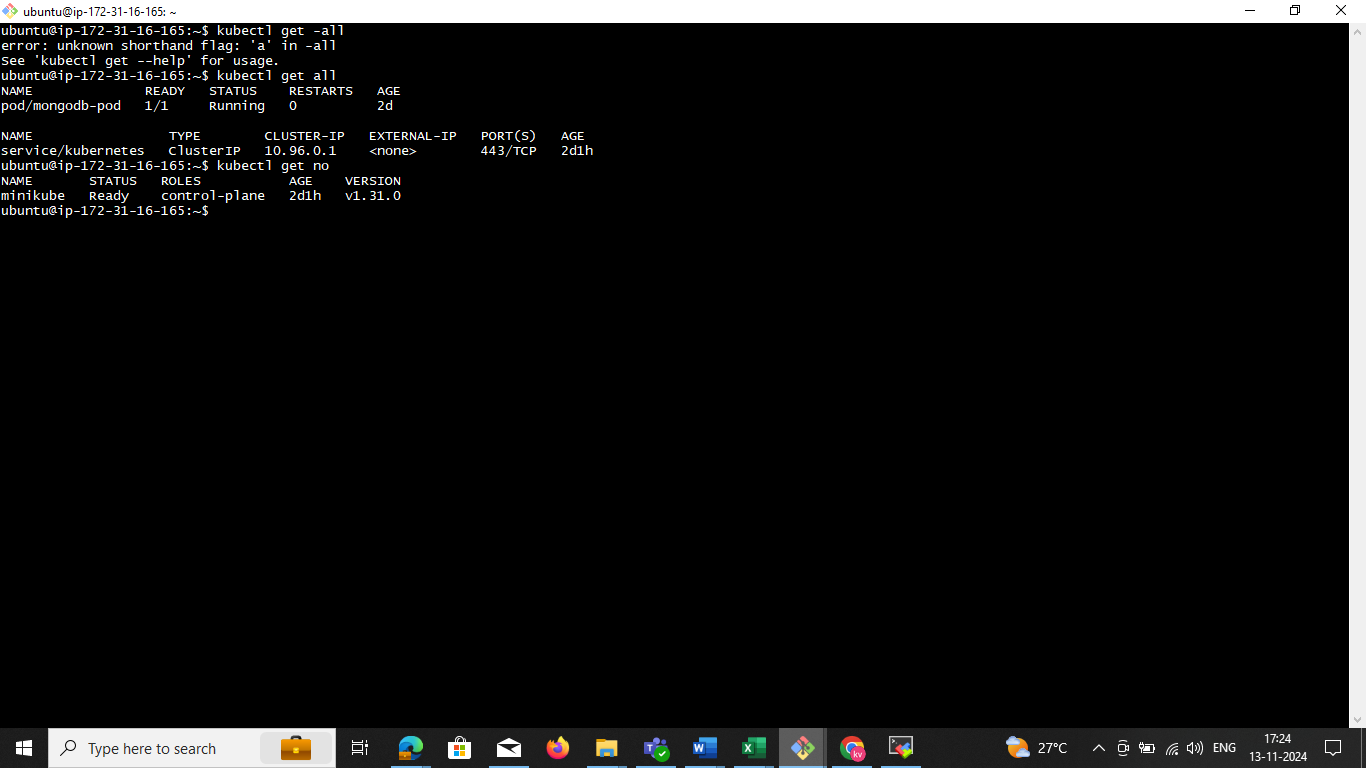
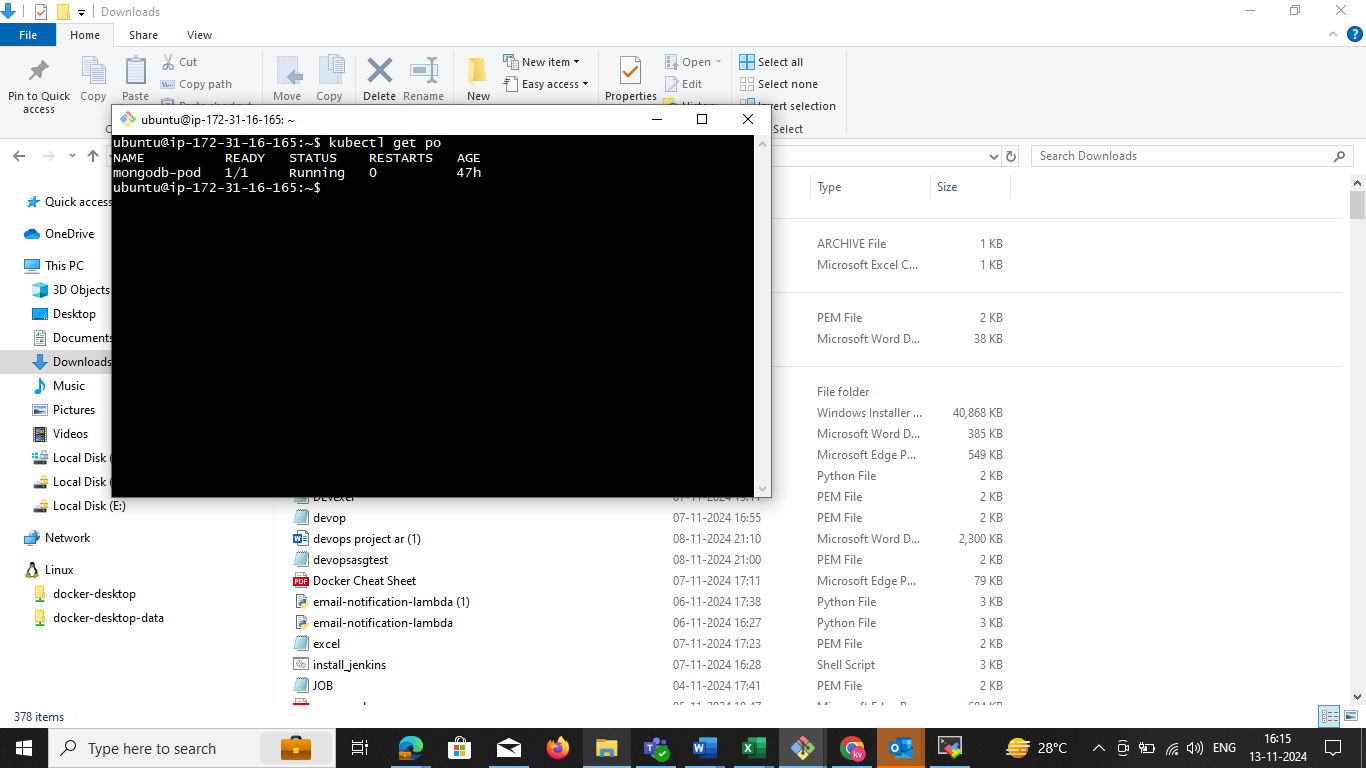
----------------------------------------**EKS mongodb pod backup** --------------------------------------------------

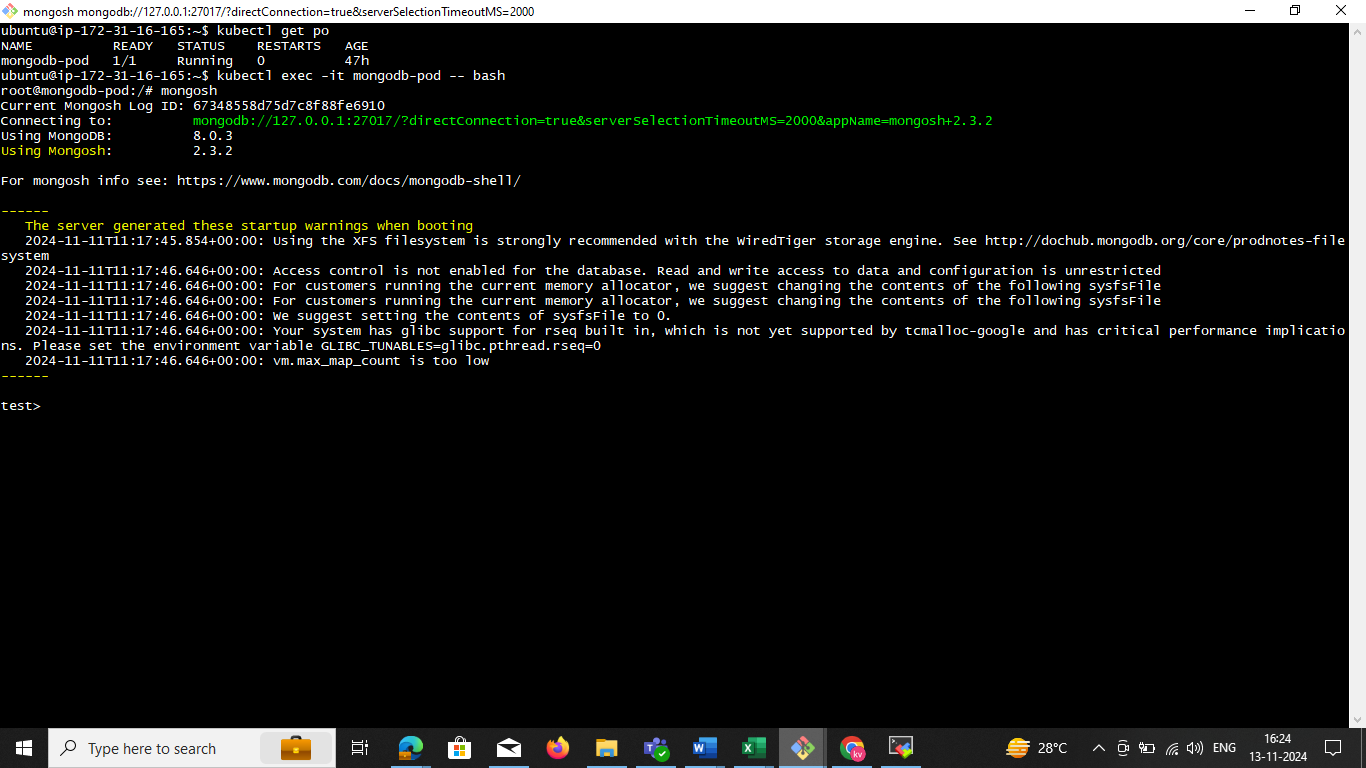
**Step1 :** got to pod login indise **kubectl exec -it mongodb-pod bash** (or)  
 **kubectl exec -it mongodb-pod -- bash** (or) **kubectl exec -it mongodb-pod -- /bin/bash** and login to shell mongodb **mongosh**

**In my case using minikube got the last page doc perquisites to setup it**

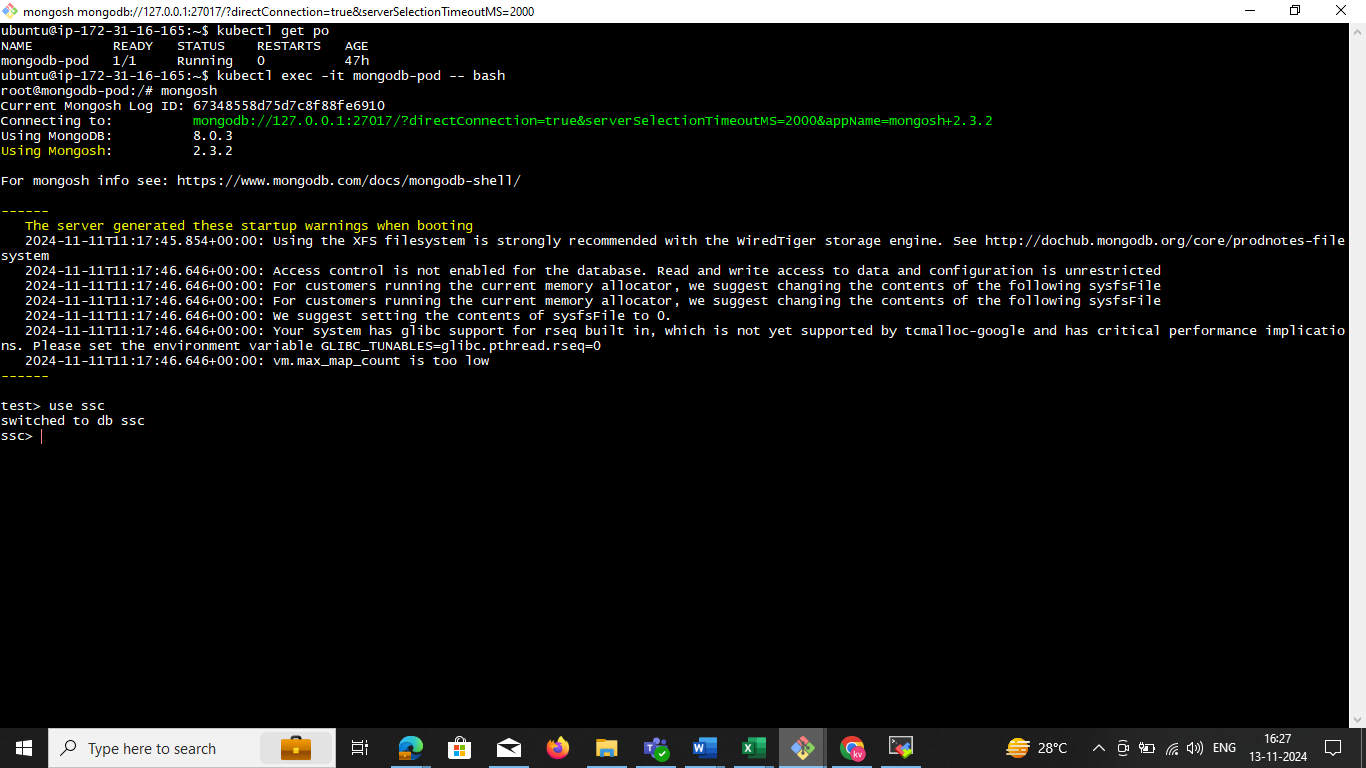


Now we need to enter in to the inside pod

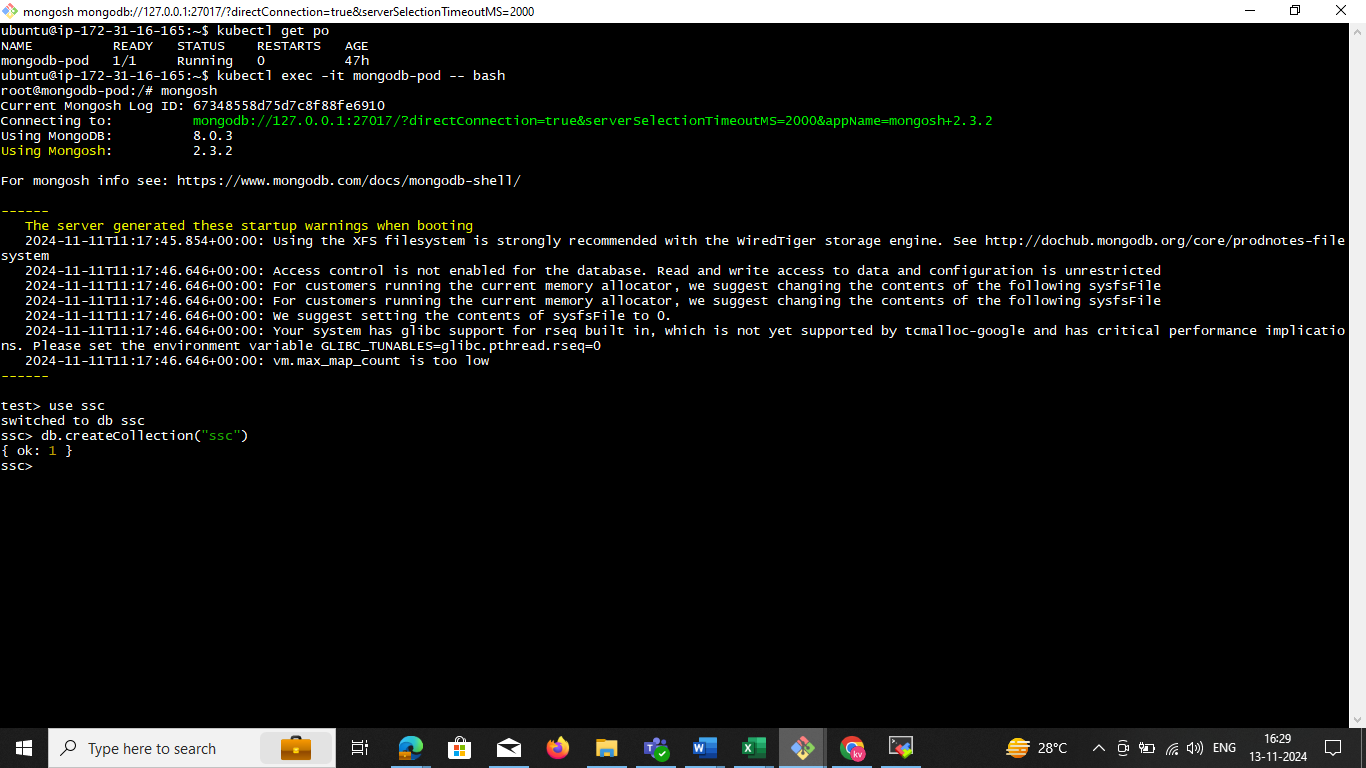
Fallow **step1** command now you enter inside of the pod then **mongosh**



**Step2:** create user 🡪 **use sss** now you can switch in to ssc



**Step3:** create the collection **db.createCollection("ssc")**



**Step4:** Insert Multiple Documents and collection

db.mycollection.insertMany([

{ name: "Bob", age: 30, city: "Chicago" },

{ name: "Charlie", age: 28, city: "San Francisco" }

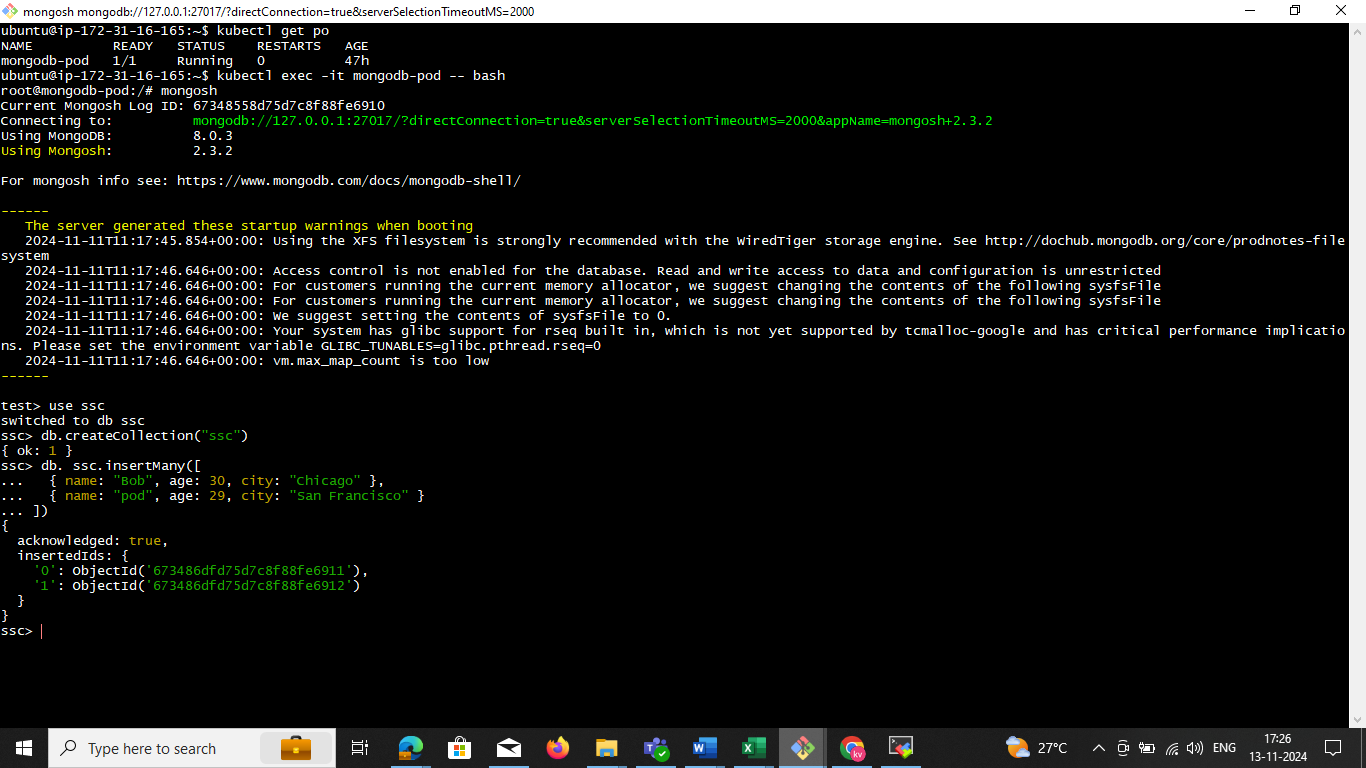
]) collection name

db. **ssc**.insertMany([

{ name: "Bob", age: 30, city: "Chicago" },

{ name: "pod", age: 29, city: "San Francisco" }

])



**Step5**: use this command single DB and Multiple db

kubectl exec -it mongodb-pod -- mongodump --db **mgdata** --out /home/ubuntu/**devopd**/

kubectl cp mongodb-pod:/home/ubuntu**/ags/** /home/ubuntu**/ags/**

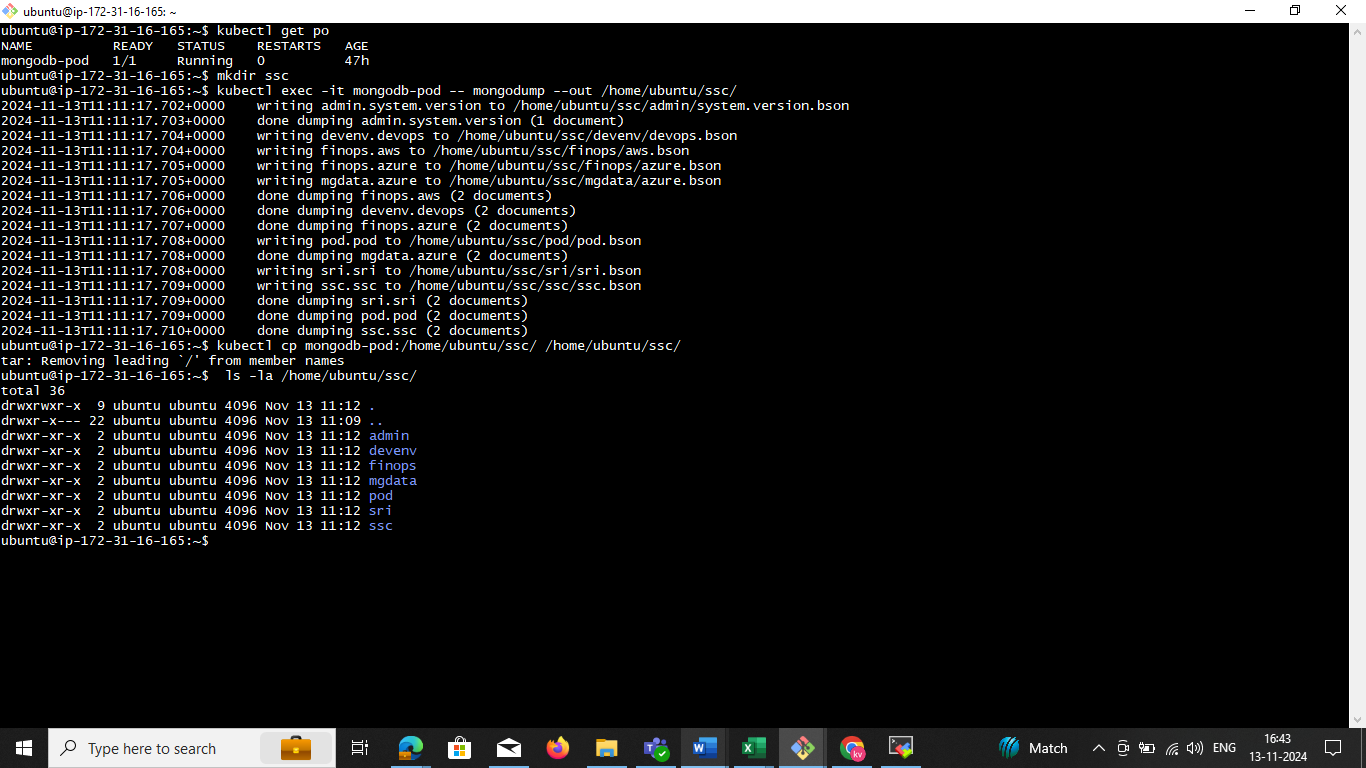
ls -la /home/ubuntu**/uat/**

**Multiple db**

kubectl exec -it **mongodb-pod** -- mongodump --out /home/ubuntu/**allbackup**/

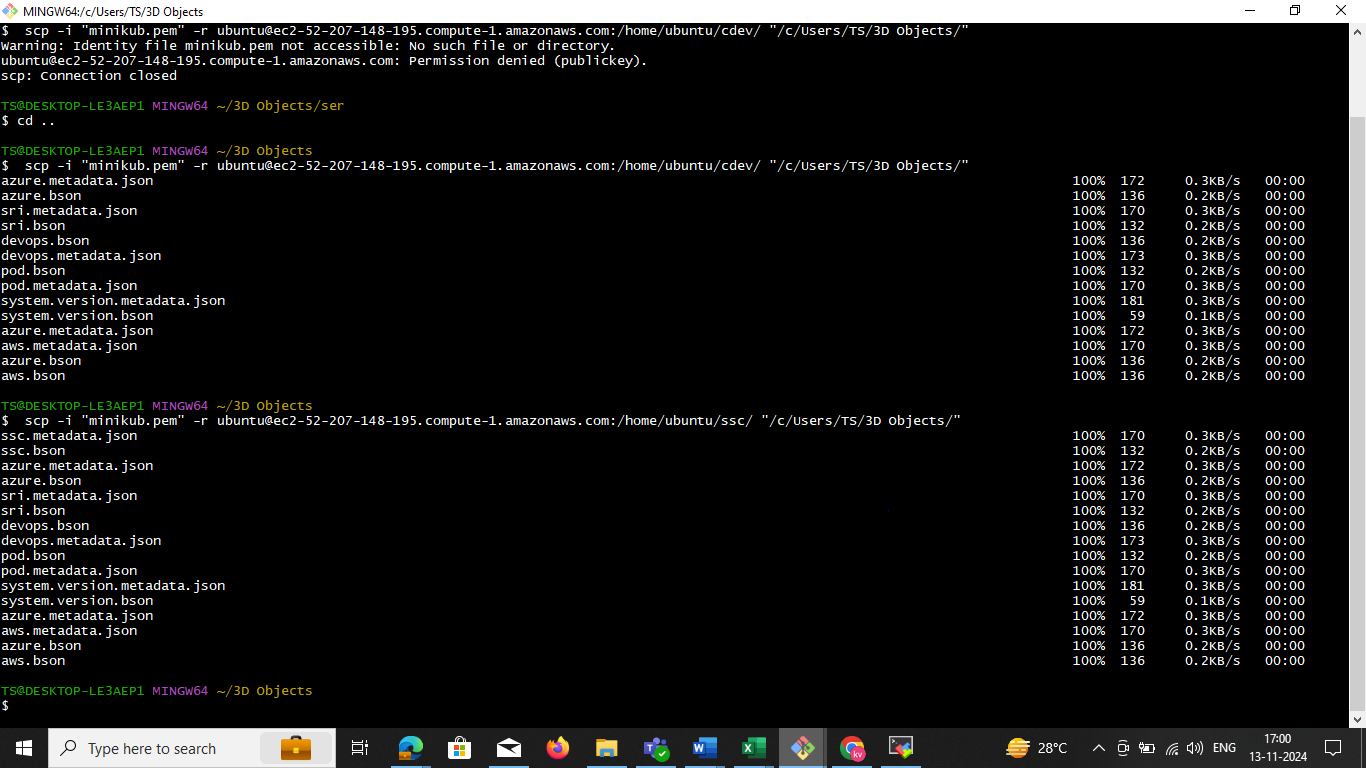
kubectl cp mongodb-pod:/home/ubuntu/**allbackup**/ /home/ubuntu/**allbackup**/

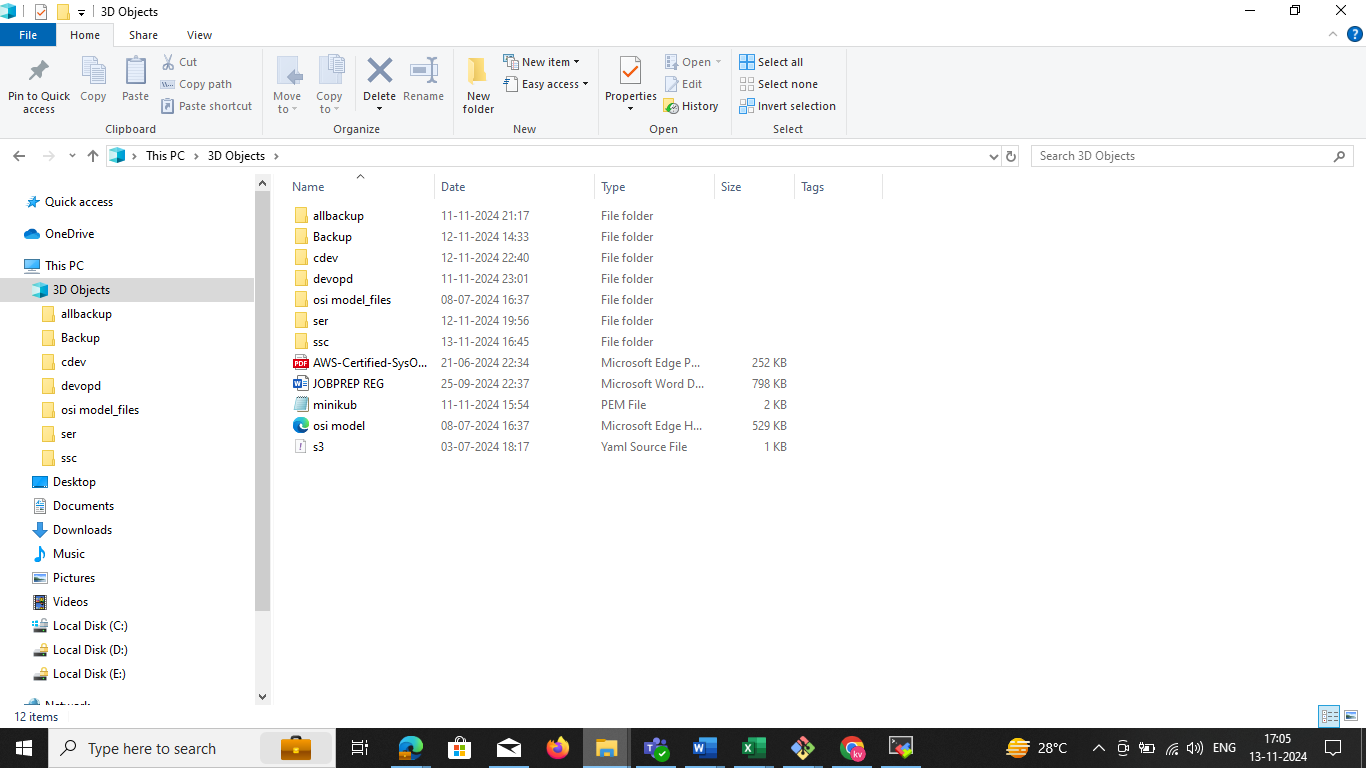
ls -la /home/ubuntu**/allbackup/**

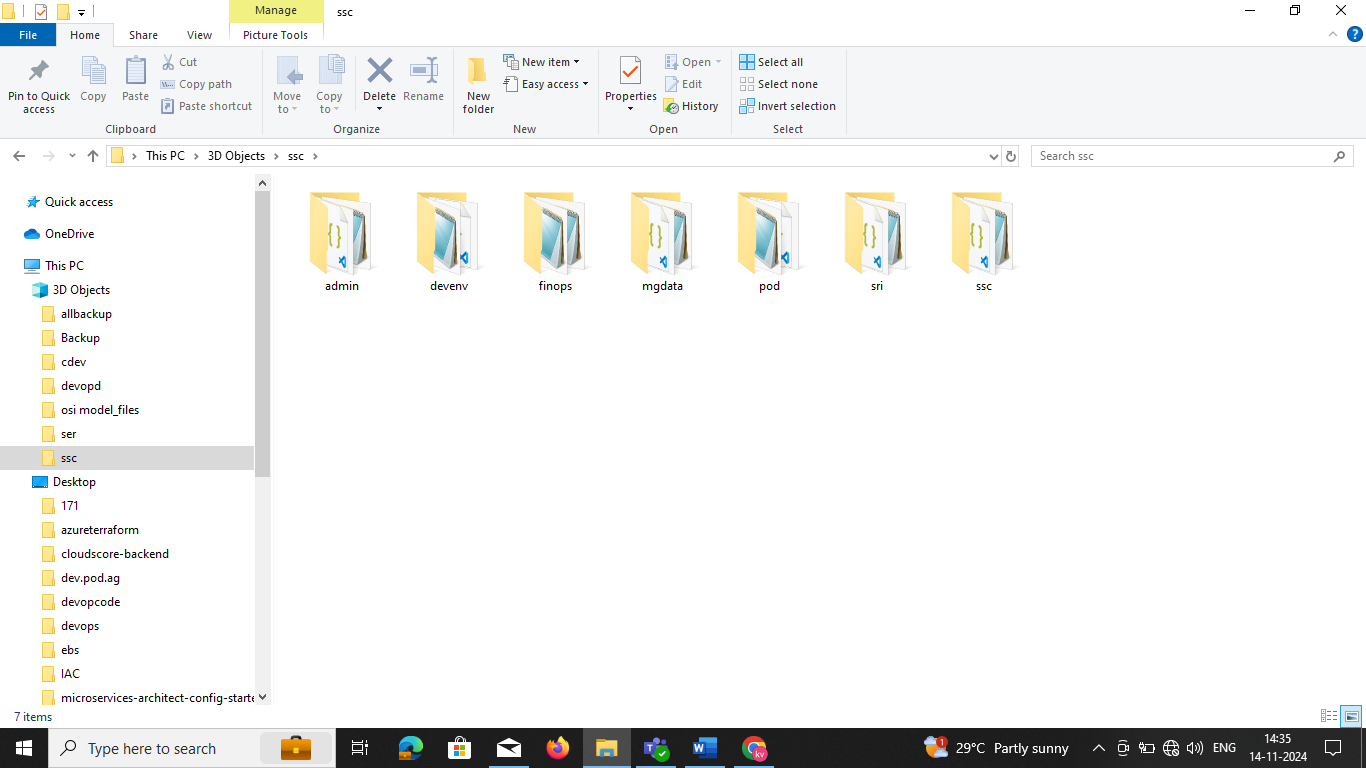


**step6 : scp**

scp -i "minikub.pem" -r ubuntu@ec2-52-207-148-195.compute-1.amazonaws.com:/home/ubuntu/ssc/ "/c/Users/TS/3D Objects/"



**step7:** nowwe need to check the local windows system



**----------------------------------------------prerequisites---------------------------------------------------------**

To install Minikube on Ubuntu, you need the following minimum requirements:

* **CPU**: At least two CPUs

* **Memory**: 2 GB of free memory

* **Disk space**: 20 GB of free disk space

* **Internet connection**: An internet connection

* **Container or virtual machine manager**: A container or virtual machine manager, such as Docker, QEMU, Hyperkit, Hyper-V, KVM, Parallels, Podman, VirtualBox, or VMware Fusion/Workstation

* **User**: A user with sudo privileges

Minikube is a tool that runs a single-node Kubernetes cluster inside a virtual machine (VM) on your laptop. It's useful for people who want to try Kubernetes or develop with it on a day-to-day basis

**Install steps in below minikube**

sudo apt-get update -y

sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common -y

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

sudo apt-get update -y

sudo apt-get install docker-ce docker-ce-cli containerd.io -y

curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt`/bin/linux/amd64/kubectl

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin/kubectl

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube

####

echo the script is now ready

echo manually run minikube start --vm-driver=docker --cni=calico to start minikube

sudo usermod -aG docker $USER

newgrp docker

minikube start --vm-driver=docker --cni=calico

**CREATE the pod**

apiVersion: v1

kind: Pod

metadata:

name: mongodb-pod

labels:

app: mongodb

spec:

containers:

- name: mongodb

image: mongo:latest

ports:

- containerPort: 27017

volumeMounts:

- name: mongodb-storage

mountPath: /data/db

volumes:

- name: mongodb-storage

emptyDir: {}

vi mongodb-pod.yaml

kubectl apply -f mongodb-pod.yaml

kubectl get pods

**Step 1: Back Up All Databases**

You can use mongodump to create a backup of all MongoDB databases. This command can be run from within the Kubernetes Pod.

1. **Create a Backup Directory**: First, create a directory on your local machine where you want to store the backup:

mkdir /path/to/backup

1. **Run mongodump Inside the MongoDB Pod**: Run the following command to create a backup of all databases:

kubectl exec -it mongodb-pod -- mongodump --out /tmp/backup/

This command runs mongodump inside the mongodb-pod and dumps all databases to /tmp/backup/ in the Pod.

1. **Copy the Backup from the Pod to Local Machine**: After running the mongodump, copy the backup from the Pod to your local machine:

kubectl cp mongodb-pod:/tmp/backup/ /path/to/backup/

This command copies the backup directory from the Pod to /path/to/backup/ on your local machine.

**Step 2: Restore the Backup to a MongoDB Pod**

To restore the backup, follow these steps:

1. **Copy the Backup to the Pod**: If you have the backup on your local machine and want to restore it in the Pod, copy the backup directory back into the Pod using kubectl cp:

kubectl cp /path/to/backup/ mongodb-pod:/tmp/restore/

This command copies the backup from /path/to/backup/ on your local machine to /tmp/restore/ in the Pod.

1. **Run mongorestore Inside the Pod**: After copying the backup to the Pod, you can restore all databases by running the following command inside the Pod:

kubectl exec -it mongodb-pod -- mongorestore --dir /tmp/restore/ --drop

* + The --dir /tmp/restore/ option specifies the location of the backup.
  + The --drop option ensures that existing collections are dropped before restoring the backup.

1. **Verify the Restoration**: After restoring, you can verify that the databases have been restored by running:

kubectl exec -it mongodb-pod -- mongo --eval "show dbs"

**pritical:**

1. **Back Up All Databases:**

kubectl exec -it mongodb-pod -- mongodump --out /tmp/backup/

kubectl cp mongodb-pod:/tmp/backup/ /path/to/backup/

EXg:

kubectl cp mongodb-pod:/tmp/uat/ /home/ubuntu/sd

ubuntu@ip-172-31-16-165:~$ kubectl cp mongodb-pod:/tmp/uat/ /home/ubuntu/sd/

tar: Removing leading `/' from member names

ubuntu@ip-172-31-16-165:~$

1. **Restore All Databases:**

kubectl cp /path/to/backup/ mongodb-pod:/tmp/restore/

kubectl exec -it mongodb-pod -- mongorestore --dir /tmp/restore/ --drop

kubectl exec -it mongodb-pod -- mongorestore --dir /tmp// --drop

kubectl exec -it mongodb-pod -- mongorestore --dir /tmp/uat/ --drop

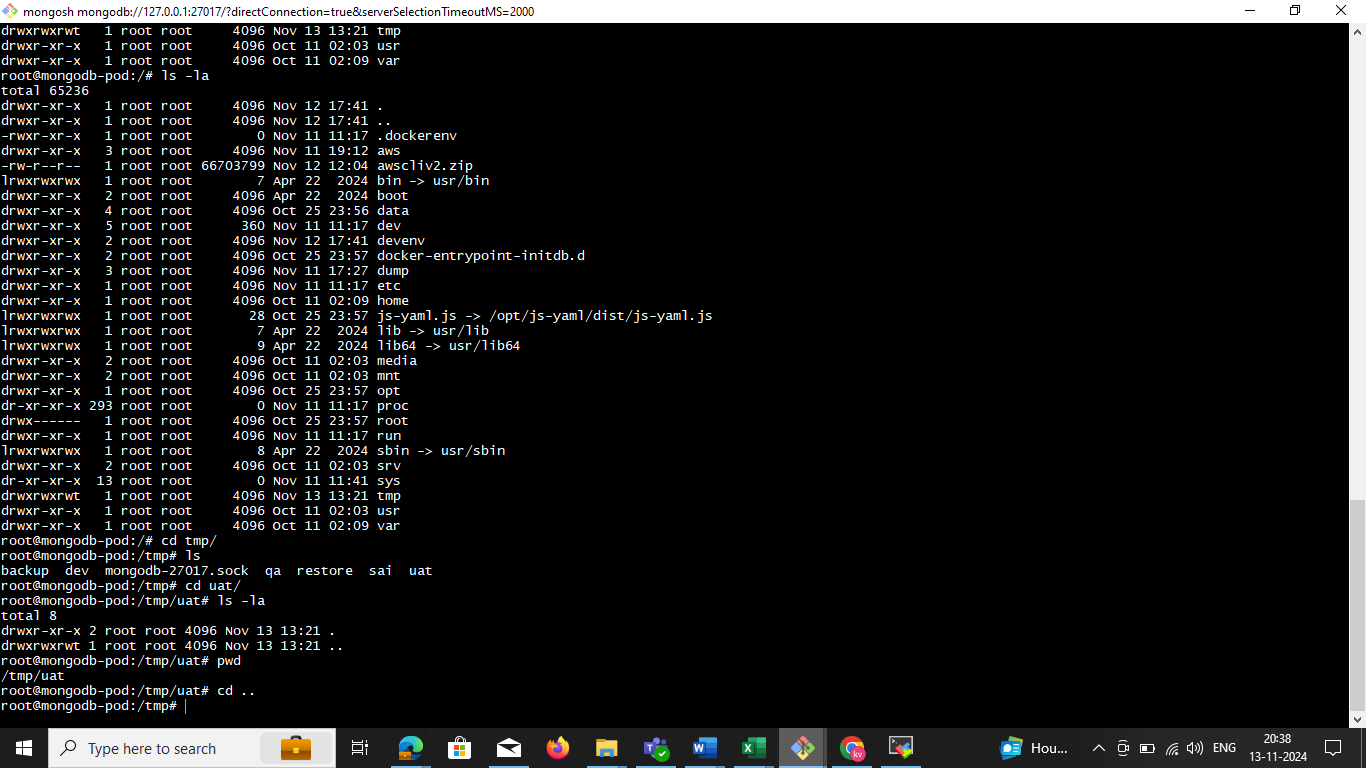
**Notes:**

* The mongodump command will back up **all databases**.
* The mongorestore command with the --drop option will overwrite existing databases.
* Ensure you have enough space on your local machine for the backup, as MongoDB backups can be large depending on the size of your databases.

drwxrwxrwt 1 root root 4096 Nov 13 13:21 ..

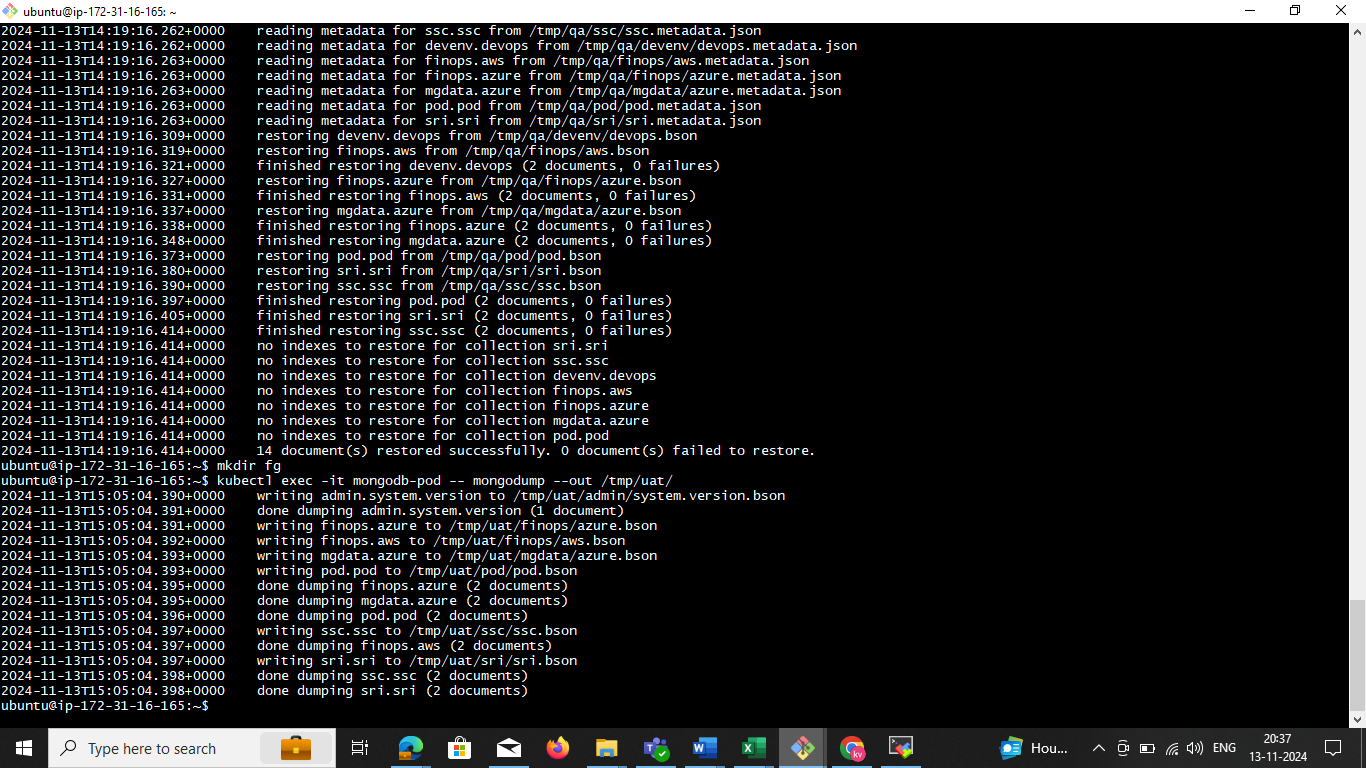
**root@mongodb-pod:/tmp/uat# pwd**

**/tmp/uat**



**Backup and restore**

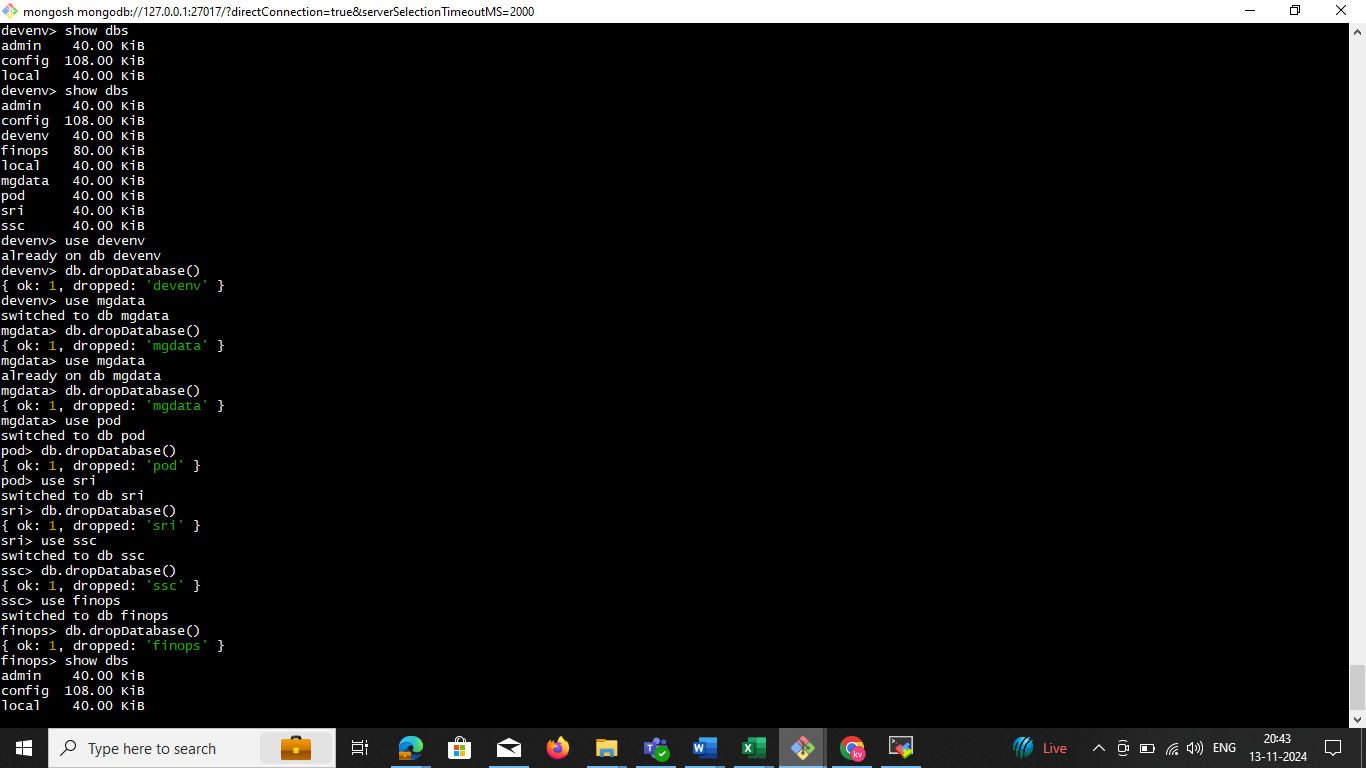
kubectl exec -it mongodb-pod -- mongodump --out /tmp/uat/



**Now we can delete the db. in the**

**use finops**

**db.dropDatabase()**

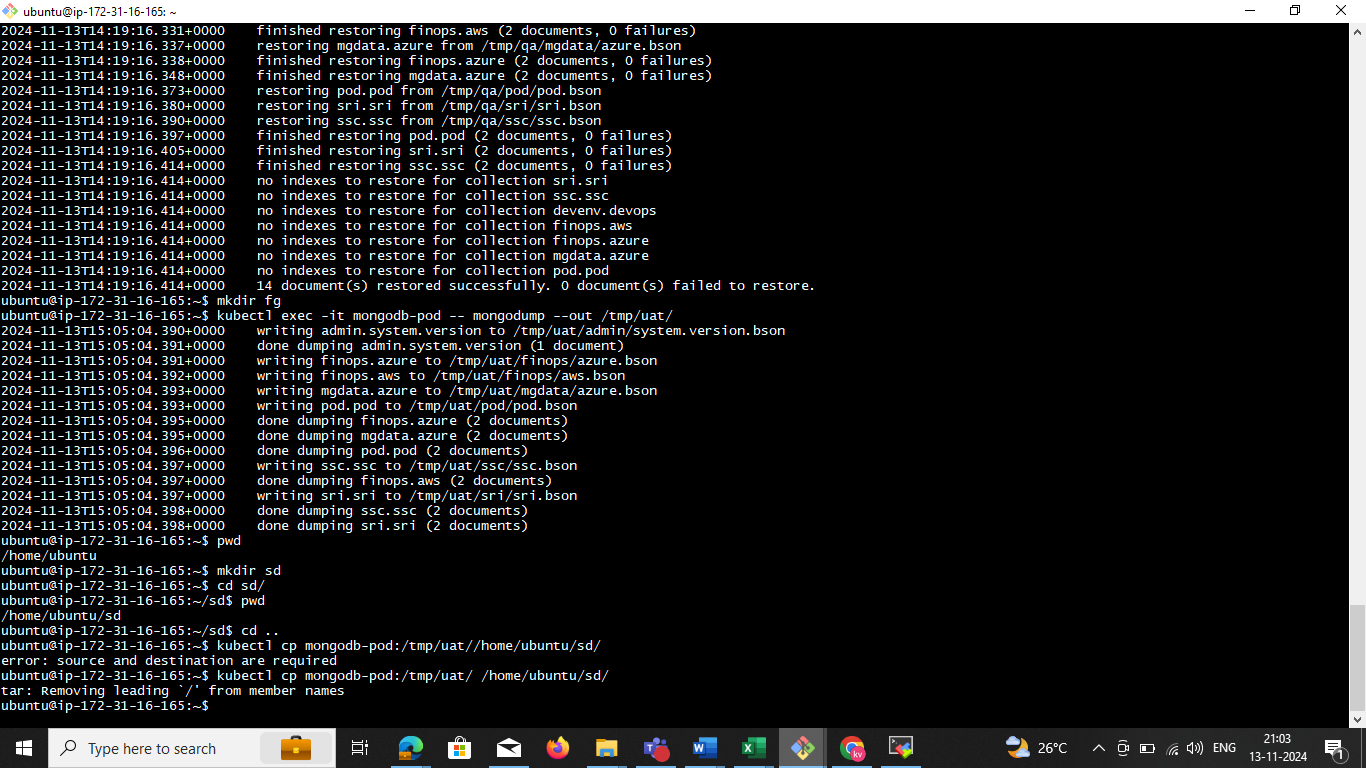


**EXg:**

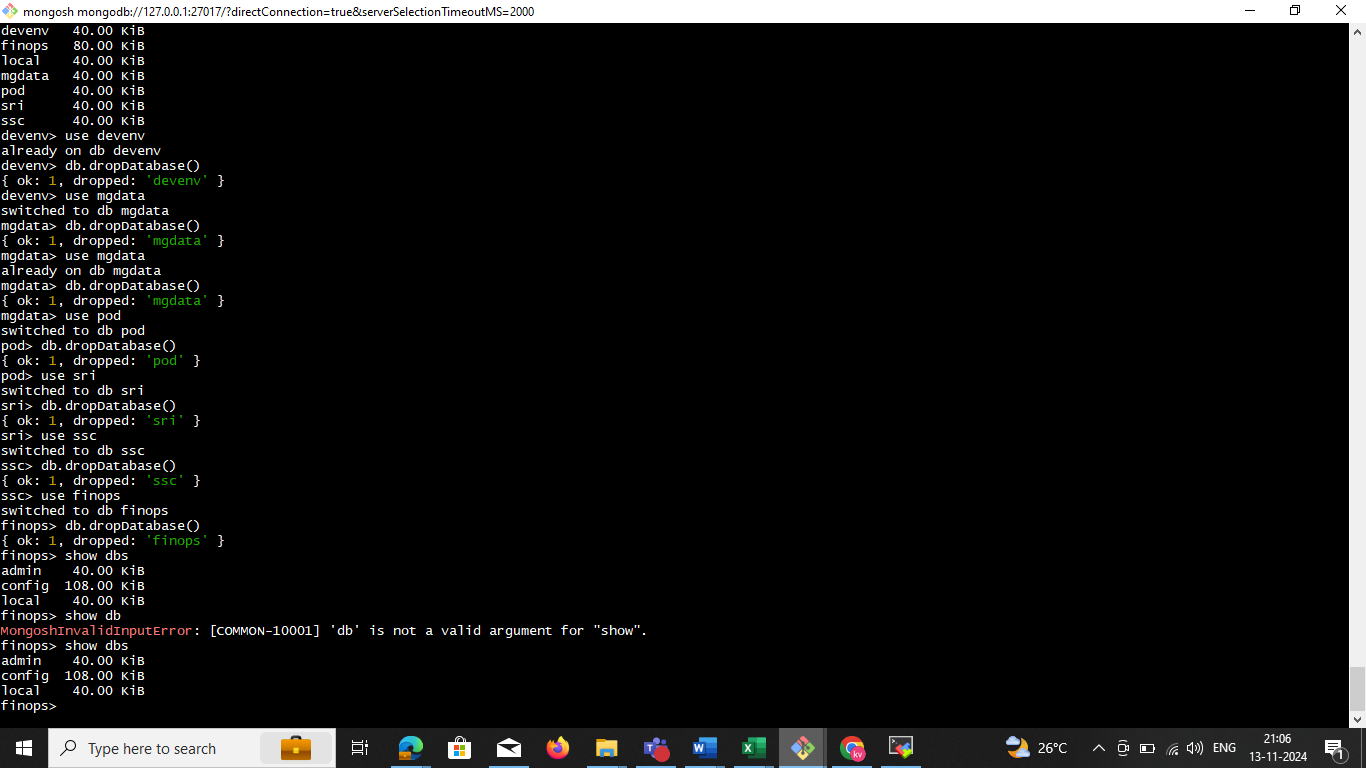
**kubectl cp mongodb-pod:/tmp/uat/ /home/ubuntu/sd**

ubuntu@ip-172-31-16-165:~$ **kubectl cp mongodb-pod:/tmp/uat/ /home/ubuntu/sd/**

tar: Removing leading `/' from member names

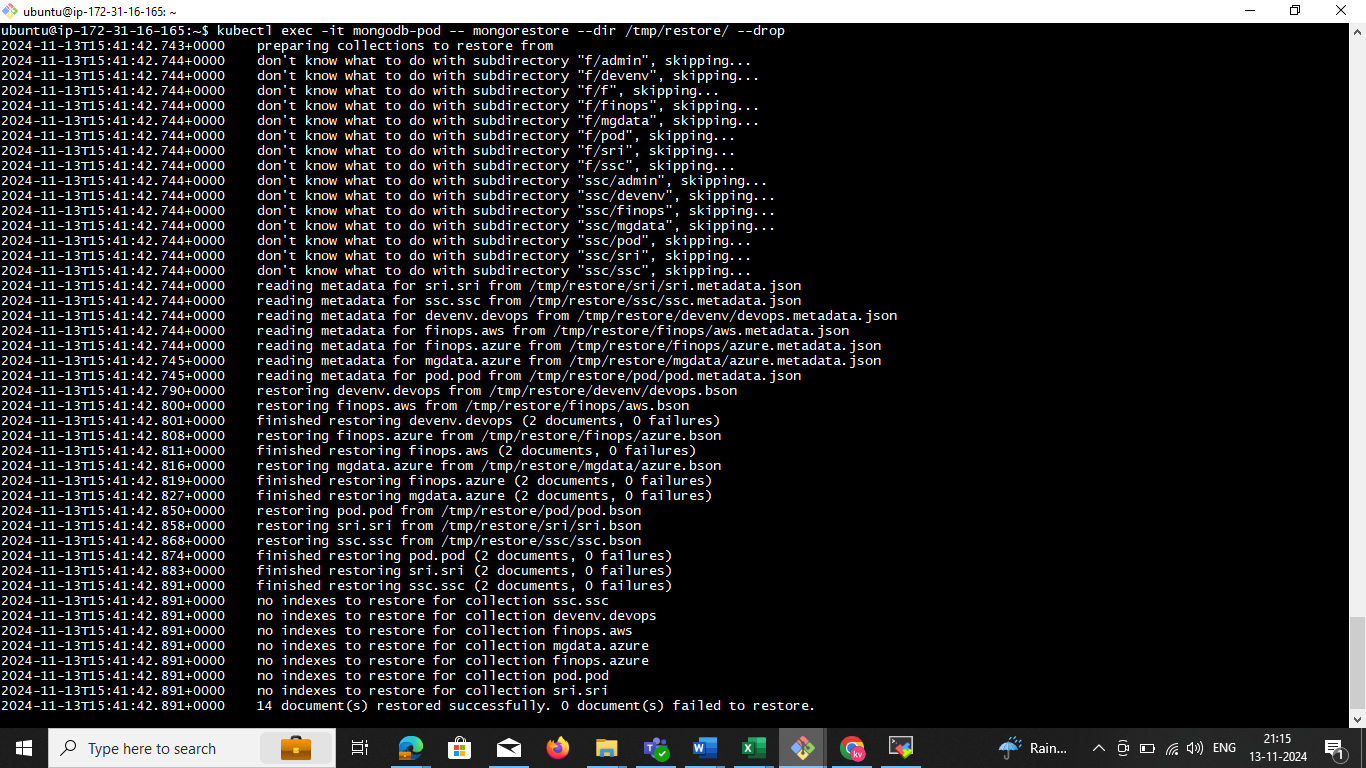


Login to **mongosh**

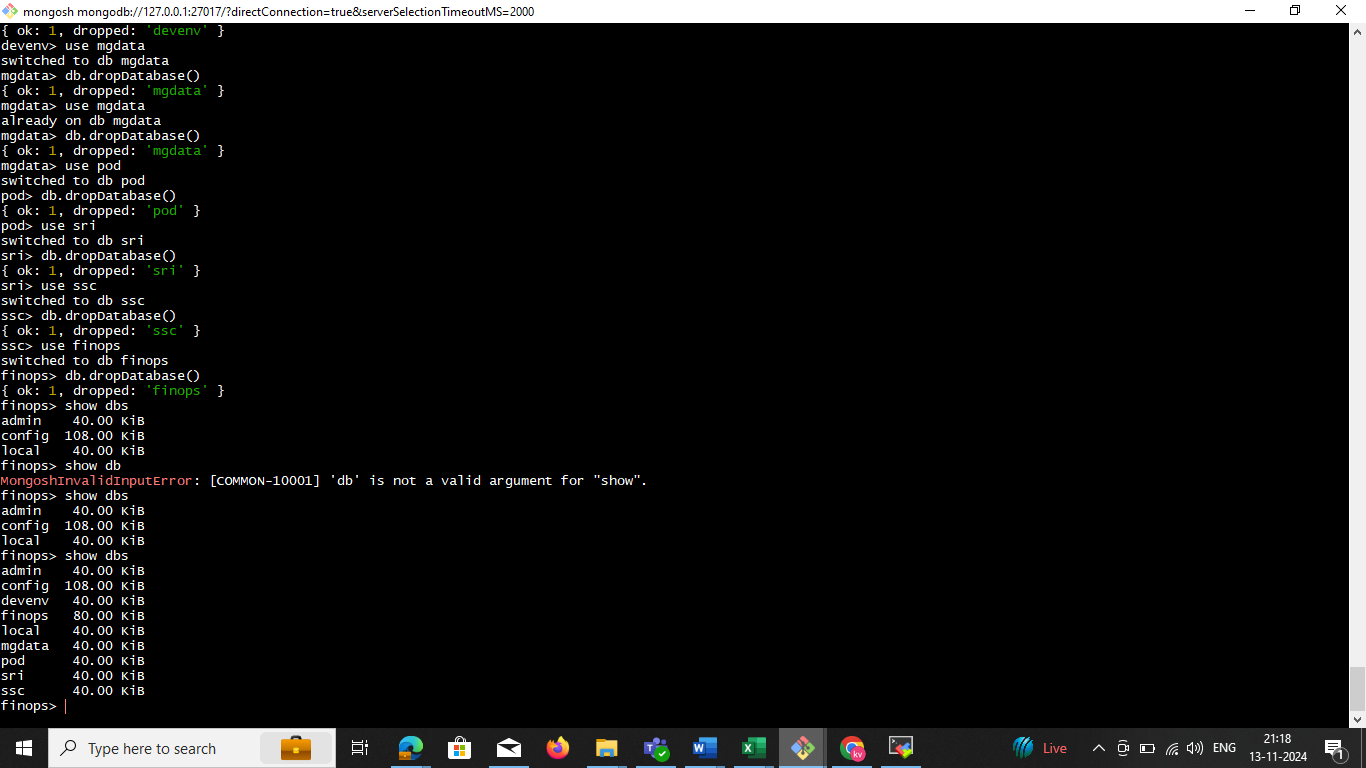


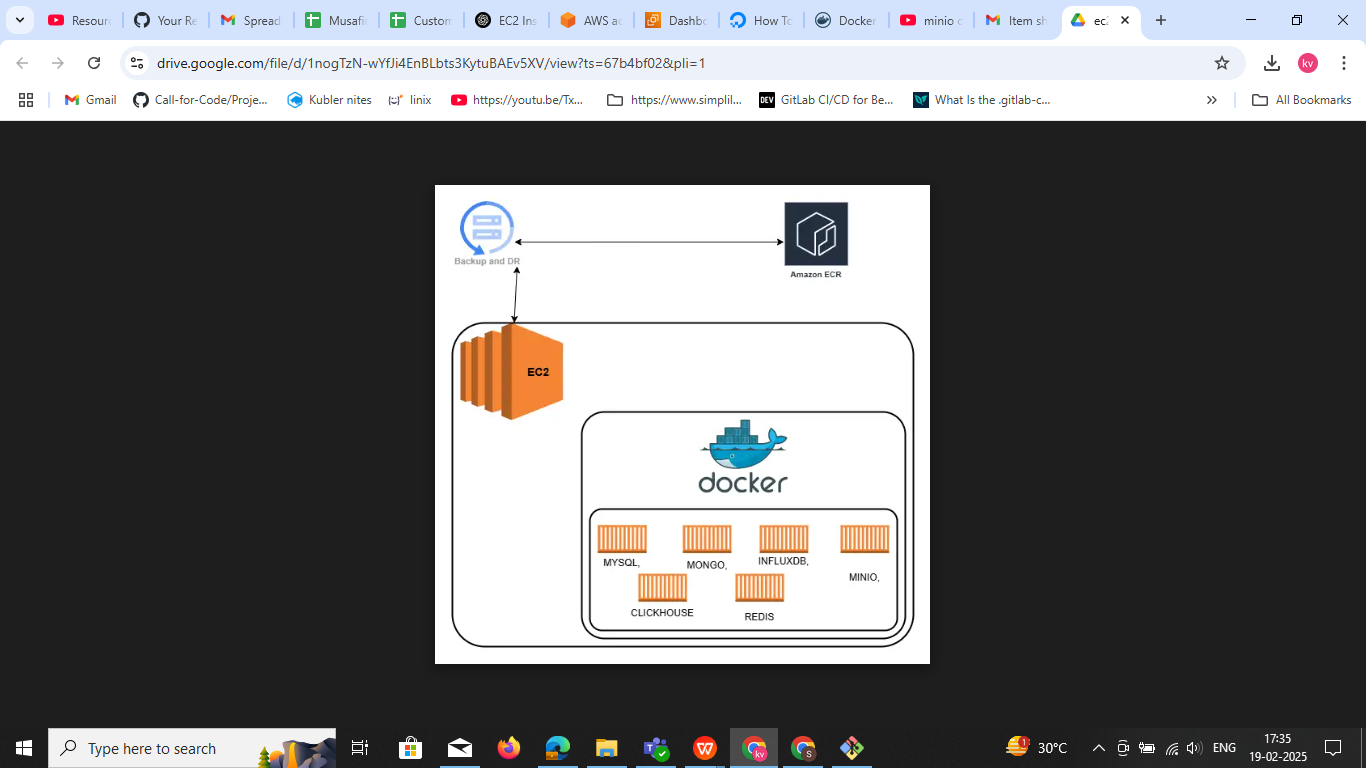
**Restore All Databases**

**kubectl exec -it mongodb-pod -- mongorestore --dir /tmp/restore/ --drop**



**now we need to check all db is the restore in mongo sh check it**





**Install the all Database set up**

**Pervisites t3x.large**

**30 gb storgare select**

**vi dockerdatabase-compose.yml**

Docker file

version: '3.8'

services:

mysql:

image: mysql:latest

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: root

MYSQL\_DATABASE: mydb

MYSQL\_USER: myuser

MYSQL\_PASSWORD: mypassword

ports:

- "3306:3306"

mongodb:

image: mongo:latest

restart: always

environment:

MONGO\_INITDB\_ROOT\_USERNAME: admin

MONGO\_INITDB\_ROOT\_PASSWORD: admin

ports:

- "27017:27017"

influxdb:

image: influxdb:latest

restart: always

environment:

INFLUXDB\_ADMIN\_USER: admin

INFLUXDB\_ADMIN\_PASSWORD: admin

ports:

- "8086:8086"

minio:

image: minio/minio

restart: always

environment:

MINIO\_ROOT\_USER: admin

MINIO\_ROOT\_PASSWORD: admin123

ports:

- "9000:9000"

- "9001:9001"

command: server /data --console-address ":9001"

clickhouse:

image: clickhouse/clickhouse-server:latest

restart: always

ports:

- "8123:8123"

- "9000:9000"

redis:

image: redis:latest

restart: always

ports:

- "6379:6379"

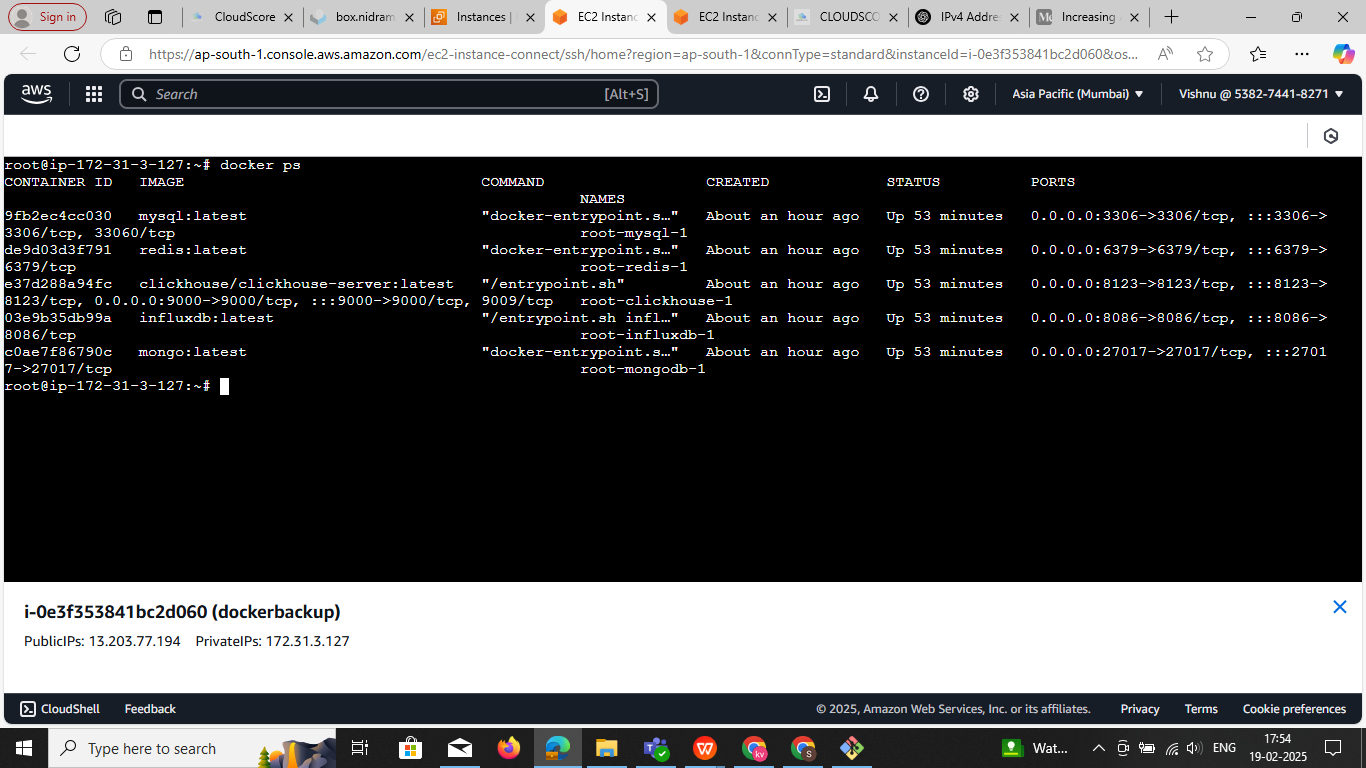
### ****Run Docker Compose****

Since your file is not named docker-compose.yml, you must specify it using the -f flag:

**docker-compose -f dockerdatabase-compose.yml up -d**

**Explanation:**

* -f dockerdatabase-compose.yml → Specifies the file to use.
* up → Starts the services.
* -d → Runs in **detached mode** (in the background).



**Login the mysql**

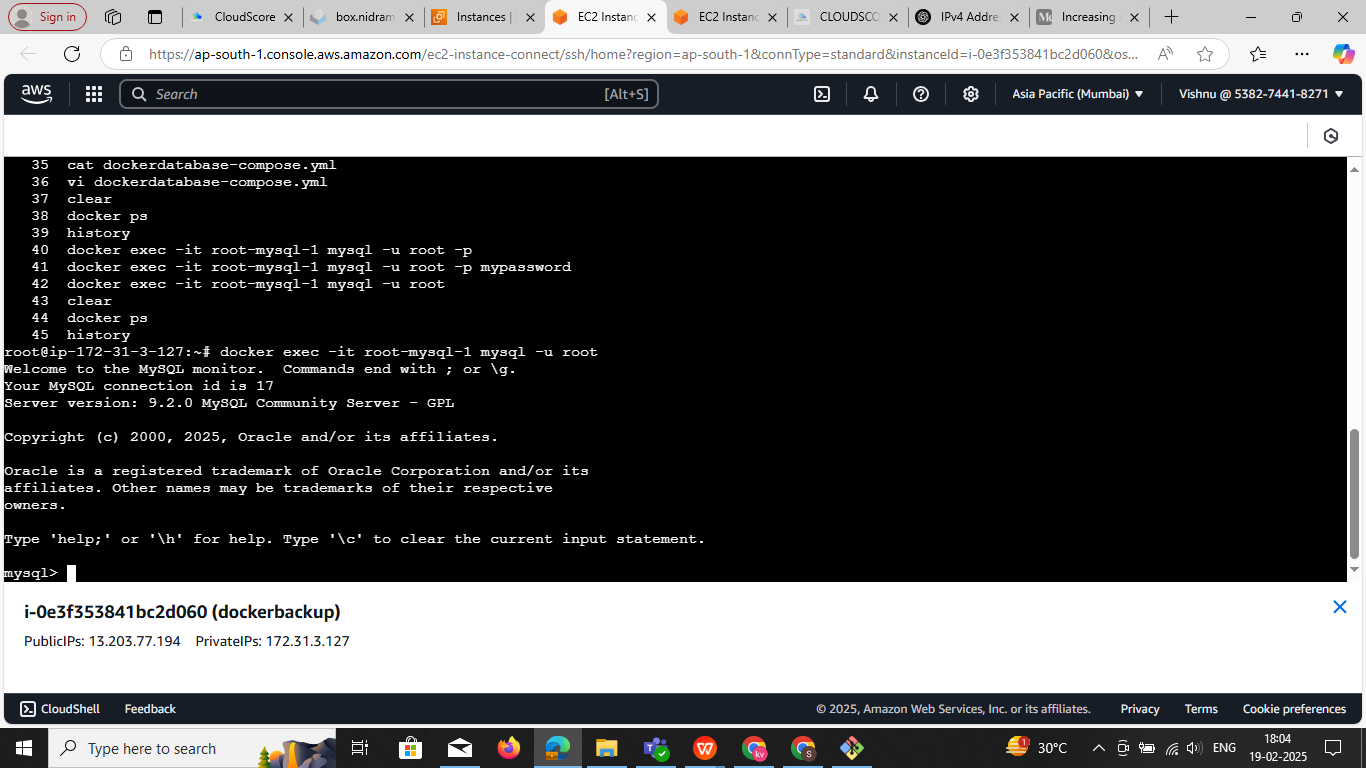
**docker exec -it root-mysql-1 mysql -u root -p**

**docker exec -it root-mysql-1 mysql -u root**

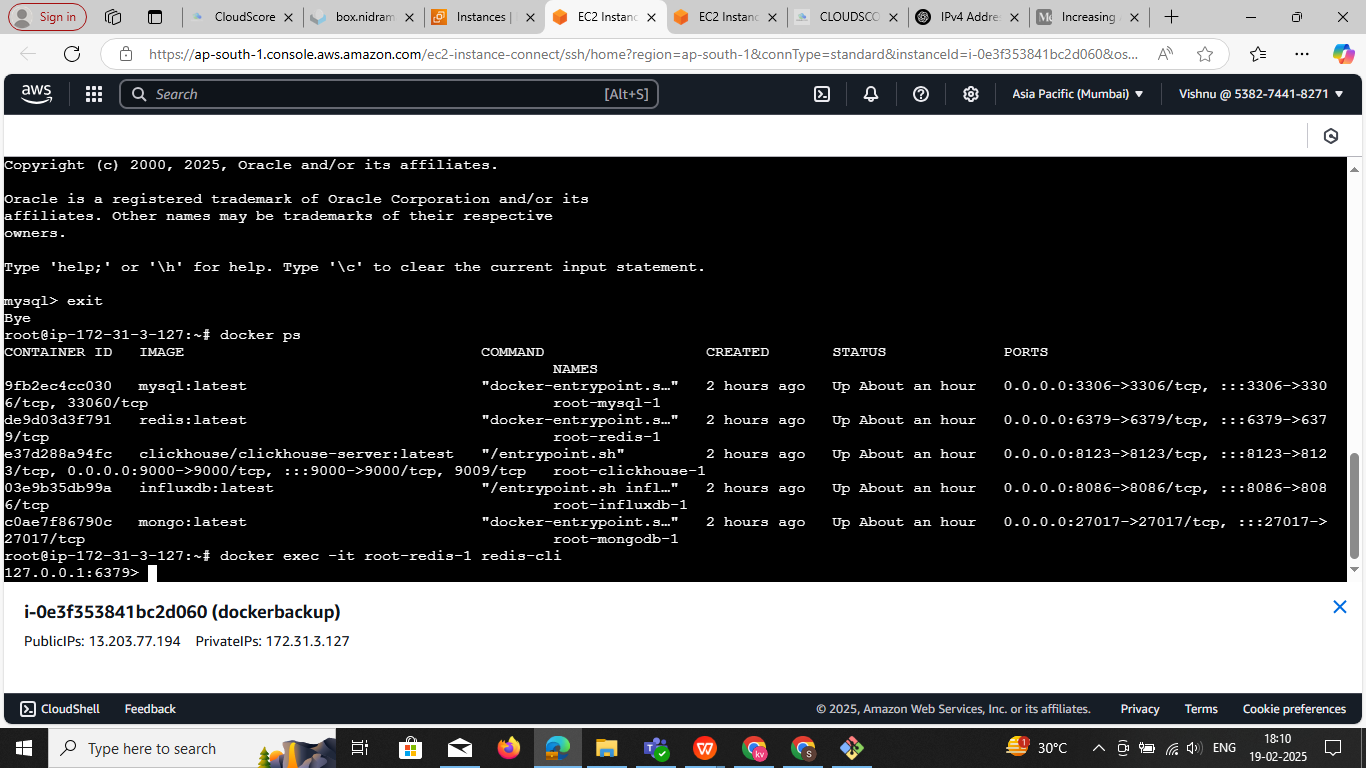
**docker exec -it d9d5ec4bdeeb mysql -u myuser -p mypassword**  if it not work

**docker exec -it d9d5ec4bdeeb mysql -u myuser -pmypassword used this command**

**Note:** try this 3 commands



**docker exec -it root-redis-1 redis-cli login to inside redis container**



**root@ip-172-31-3-127:~# docker exec -it root-clickhouse-1 clickhouse-client --host localhost**

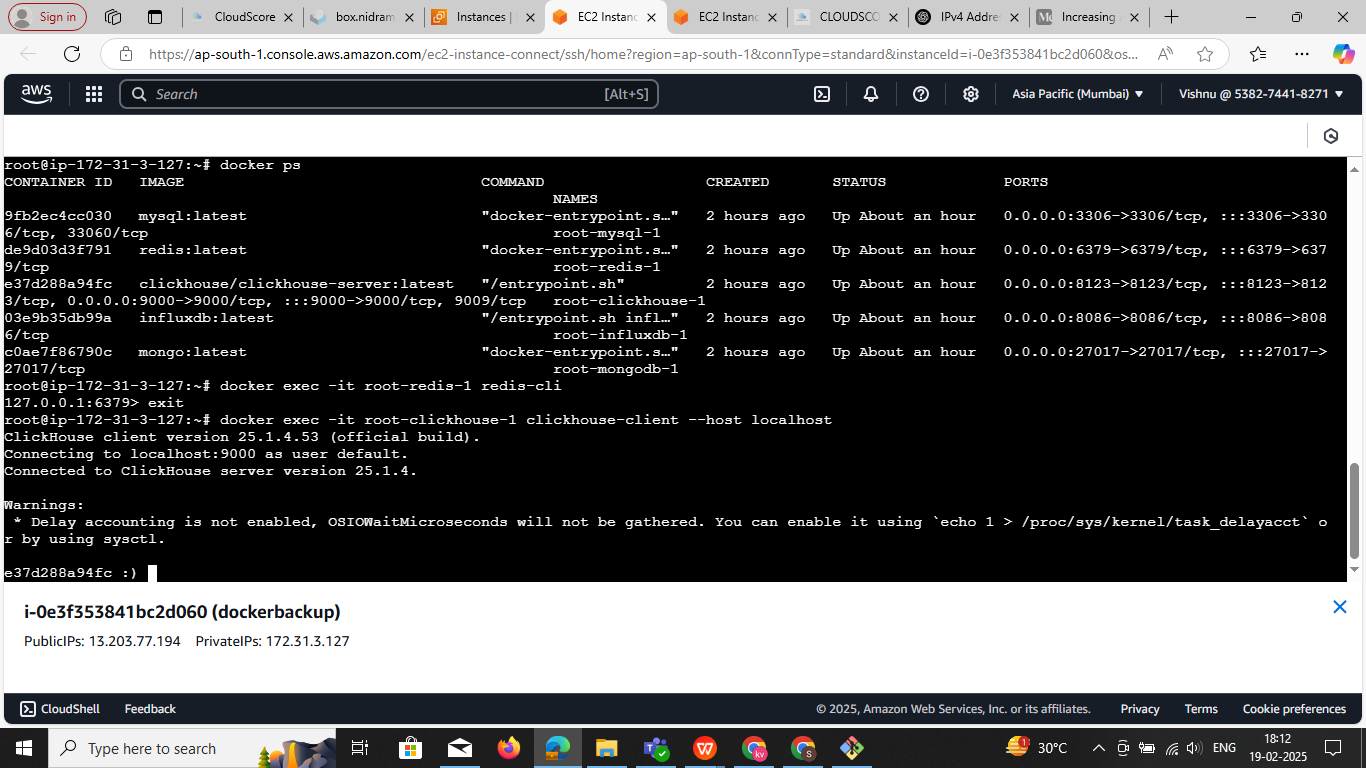
ClickHouse client version 25.1.4.53 (official build).

Connecting to localhost:9000 as user default.

Connected to ClickHouse server version 25.1.4.

Warnings:

\* Delay accounting is not enabled, OSIOWaitMicroseconds will not be gathered. You can enable it using `echo 1 > /proc/sys/kernel/task\_delayacct` or by using sysctl.



### ****Access the Container Shell****

Use the docker exec command to start an interactive shell inside the container:

docker exec -it root-influxdb-1 /bin/bash

To **remove all containers (running & stopped) and images** in a single command, run:

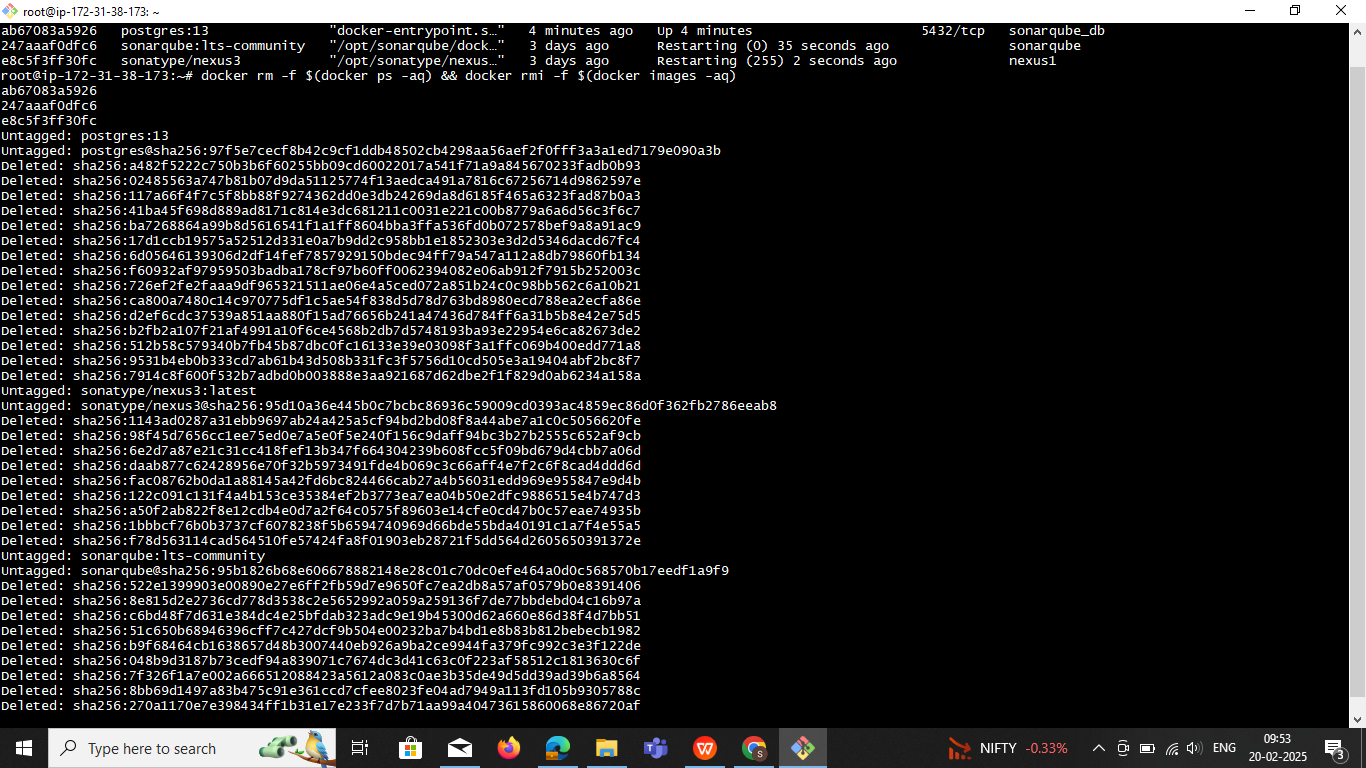
**docker rm -f $(docker ps -aq) && docker rmi -f $(docker images -aq)**

### Explanation:

* docker ps -aq → Lists all container IDs.
* docker rm -f $(docker ps -aq) → Force removes all containers.
* docker images -aq → Lists all image IDs.
* docker rmi -f $(docker images -aq) → Force removes all images.

⚠️ **Warning:** This will completely clean up **all containers and images** on your system.  
If you also want to remove unused volumes and networks, use:

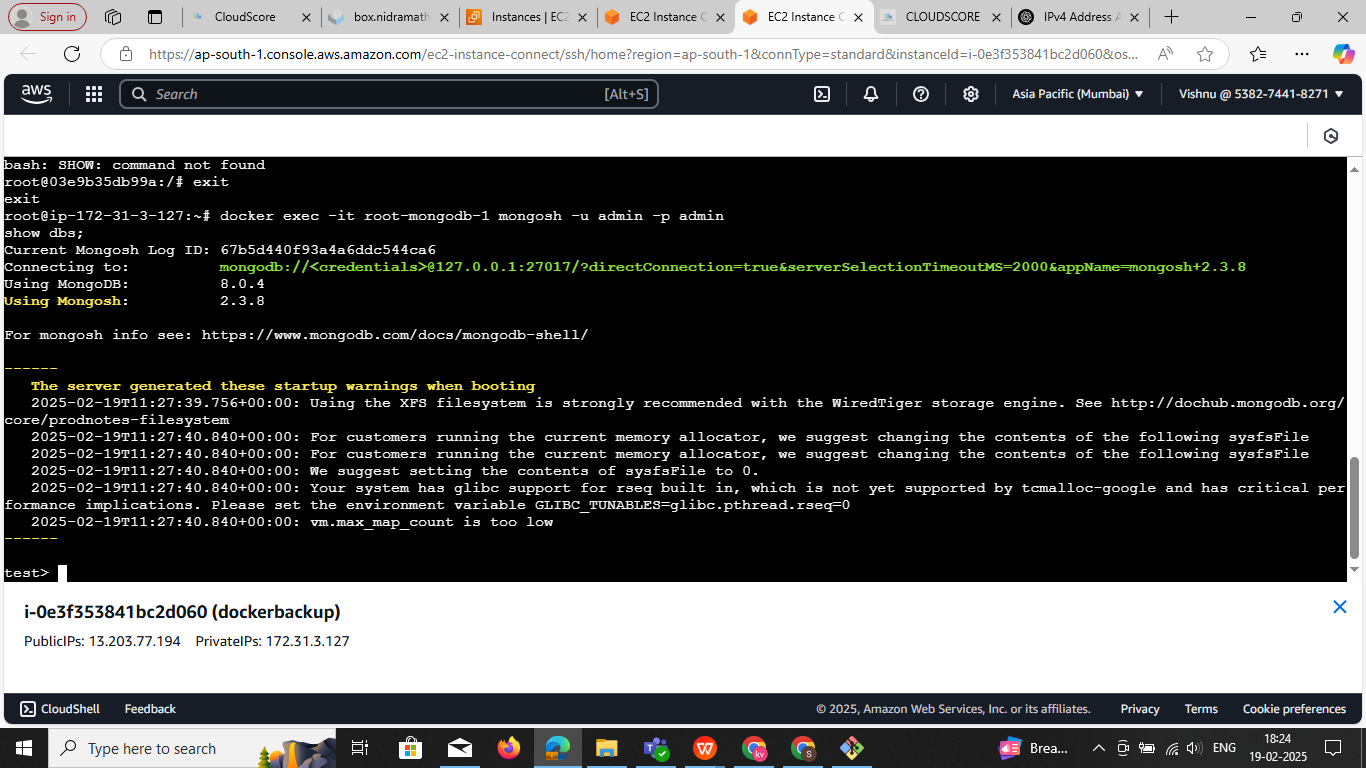
**docker system prune -a -f**



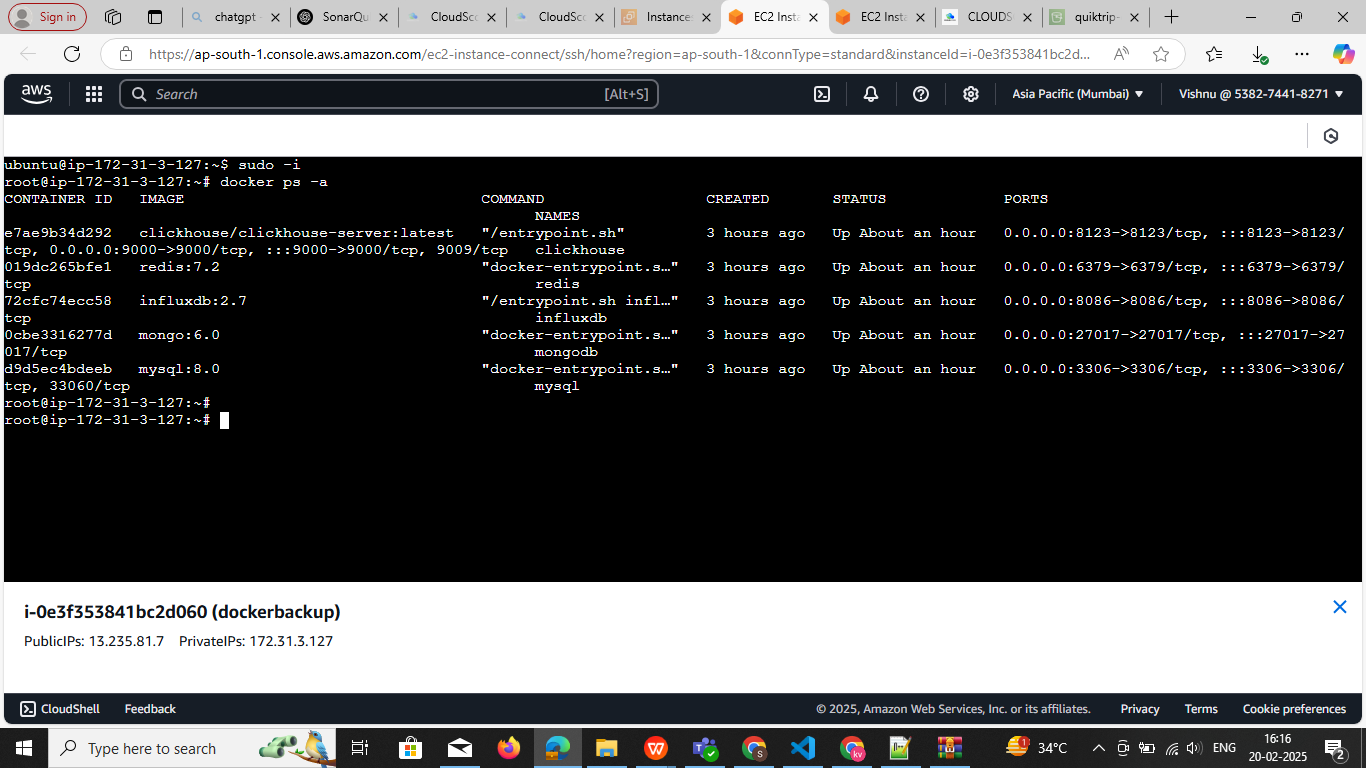
**Mongodb login**

**docker exec -it root-mongodb-1 mongosh -u admin -p admin**

**show dbs;**



**------------------------------------DataBase: clickhouse---------------------------**



**Command to login** :**docker exec -it clickhouse clickhouse-client**

**1. Ensure a Table Exists**

Before inserting data, create a table that matches your dataset:

CREATE TABLE cloud\_resources (

cloud\_resource\_id String,

resource\_name String,

resource\_id String,

cloud\_account\_id String,

cloud\_type String,

region String,

flavor String,

saving Float64,

average\_saving Float64

) ENGINE = MergeTree()

ORDER BY cloud\_resource\_id;

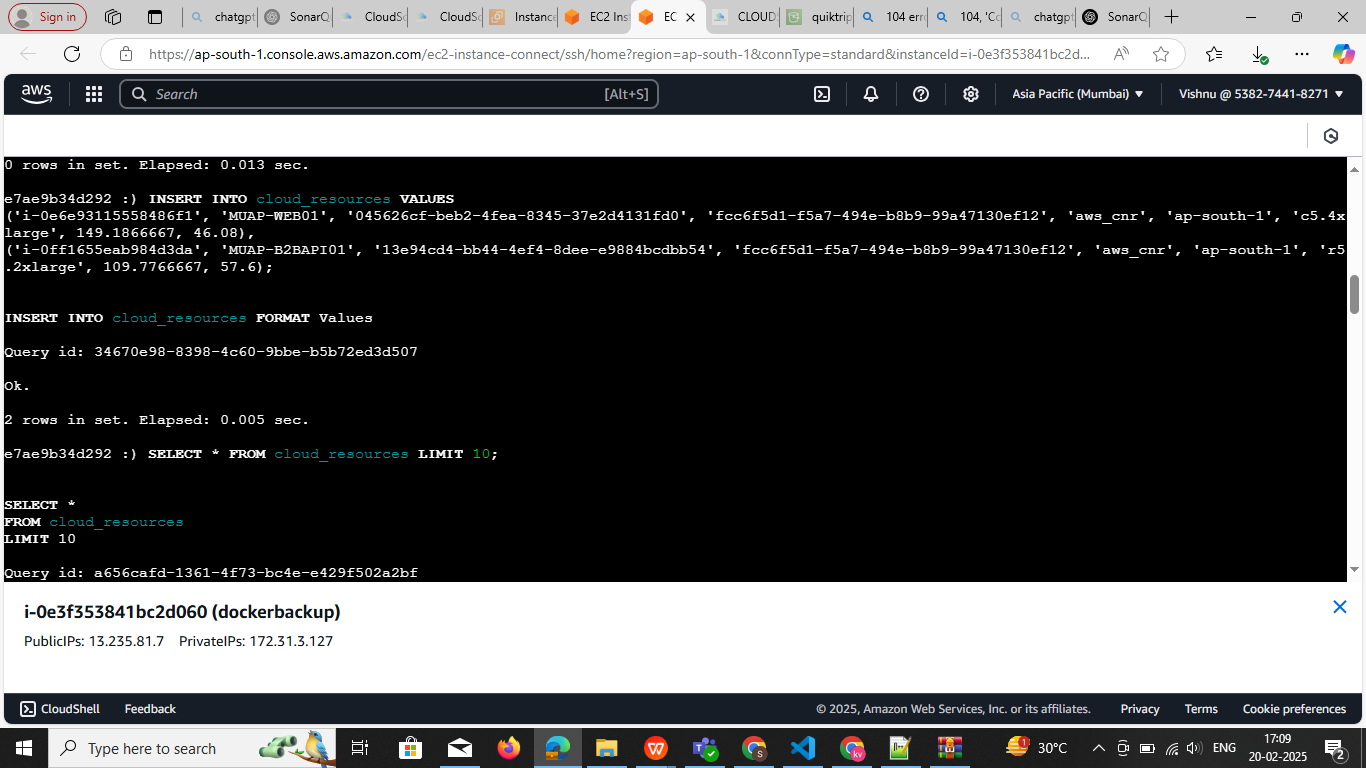
**2. Insert Data Manually**

You can insert records one by one:

**INSERT INTO cloud\_resources VALUES**

**('i-0e6e93115558486f1', 'MUAP-WEB01', '045626cf-beb2-4fea-8345-37e2d4131fd0', 'fcc6f5d1-f5a7-494e-b8b9-99a47130ef12', 'aws\_cnr', 'ap-south-1', 'c5.4xlarge', 149.1866667, 46.08),**

**('i-0ff1655eab984d3da', 'MUAP-B2BAPI01', '13e94cd4-bb44-4ef4-8dee-e9884bcdbb54', 'fcc6f5d1-f5a7-494e-b8b9-99a47130ef12', 'aws\_cnr', 'ap-south-1', 'r5.2xlarge', 109.7766667, 57.6);**



**3. Bulk Insert from a File (Recommended)**

**Step 1:** Save Data as a CSV File

Create a CSV file (data.csv) with this format:

cloud\_resource\_id,resource\_name,resource\_id,cloud\_account\_id,cloud\_type,region,flavor,saving,average\_saving

i-0e6e93115558486f1,MUAP-WEB01,045626cf-beb2-4fea-8345-37e2d4131fd0,fcc6f5d1-f5a7-494e-b8b9-99a47130ef12,aws\_cnr,ap-south-1,c5.4xlarge,149.1866667,46.08

i-0ff1655eab984d3da,MUAP-B2BAPI01,13e94cd4-bb44-4ef4-8dee-e9884bcdbb54,fcc6f5d1-f5a7-494e-b8b9-99a47130ef12,aws\_cnr,ap-south-1,r5.2xlarge,109.7766667,57.6

...

**Step 2: Copy the File into the ClickHouse Container**

If data.csv is on your local machine, copy it into the running container:

**docker cp data.csv clickhouse:/data.csv**

**Step 3: Load Data into ClickHouse**

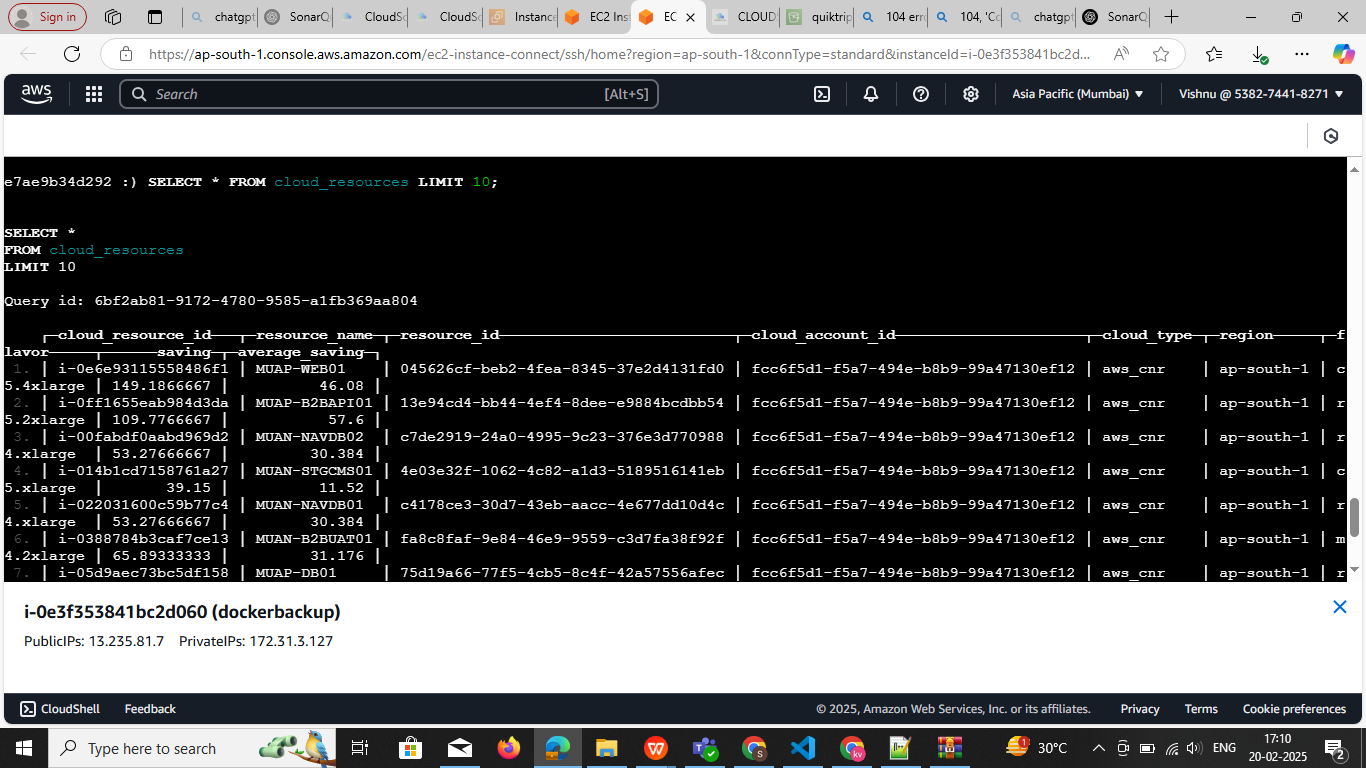
Now, run this inside the ClickHouse client:

INSERT INTO cloud\_resources FORMAT CSV

docker exec -i clickhouse clickhouse-client --query="INSERT INTO cloud\_resources FORMAT CSV" < data.csv

**4. Verify Data**

To confirm data was inserted correctly:

**SELECT \* FROM cloSELECT \* FROM cloud\_resources LIMIT 10;**

**ud\_resources LIMIT 10;**

**-------------------------------------Redis-ci-----------------------------------------**

## **Login to Redis Container** Run the following command to enter the Redis CLI:

## docker exec -it redis redis-cli and insert data

## ****Use a Redis Pipeline to Insert Multiple Records****

## Copy and paste the following into the Redis CLI to insert all data in **bulk**:

**MULTI**

HSET cloud\_resource:i-0e6e93115558486f1 resource\_name "MUAP-WEB01" resource\_id "045626cf-beb2-4fea-8345-37e2d4131fd0" cloud\_account\_id "fcc6f5d1-f5a7-494e-b8b9-99a47130ef12" cloud\_type "aws\_cnr" region "ap-south-1" flavor "c5.4xlarge" saving "149.1866667" average\_saving "46.08"

HSET cloud\_resource:i-0ff1655eab984d3da resource\_name "MUAP-B2BAPI01" resource\_id "13e94cd4-bb44-4ef4-8dee-e9884bcdbb54" cloud\_account\_id "fcc6f5d1-f5a7-494e-b8b9-99a47130ef12" cloud\_type "aws\_cnr" region "ap-south-1" flavor "r5.2xlarge" saving "109.7766667" average\_saving "57.6"

HSET cloud\_resource:i-05d9aec73bc5df158 resource\_name "MUAP-DB01" resource\_id "75d19a66-77f5-4cb5-8c4f-42a57556afec" cloud\_account\_id "fcc6f5d1-f5a7-494e-b8b9-99a47130ef12" cloud\_type "aws\_cnr" region "ap-south-1" flavor "r5.2xlarge" saving "100.1433333" average\_saving "57.6"

HSET cloud\_resource:i-06cbf4e6cc6969b78 resource\_name "MUAN-BWEB01" resource\_id "bfe6e76b-e1de-4e0a-9824-4f3da3f17ab0" cloud\_account\_id "fcc6f5d1-f5a7-494e-b8b9-99a47130ef12" cloud\_type "aws\_cnr" region "ap-south-1" flavor "t3.2xlarge" saving "91.782" average\_saving "28.728"

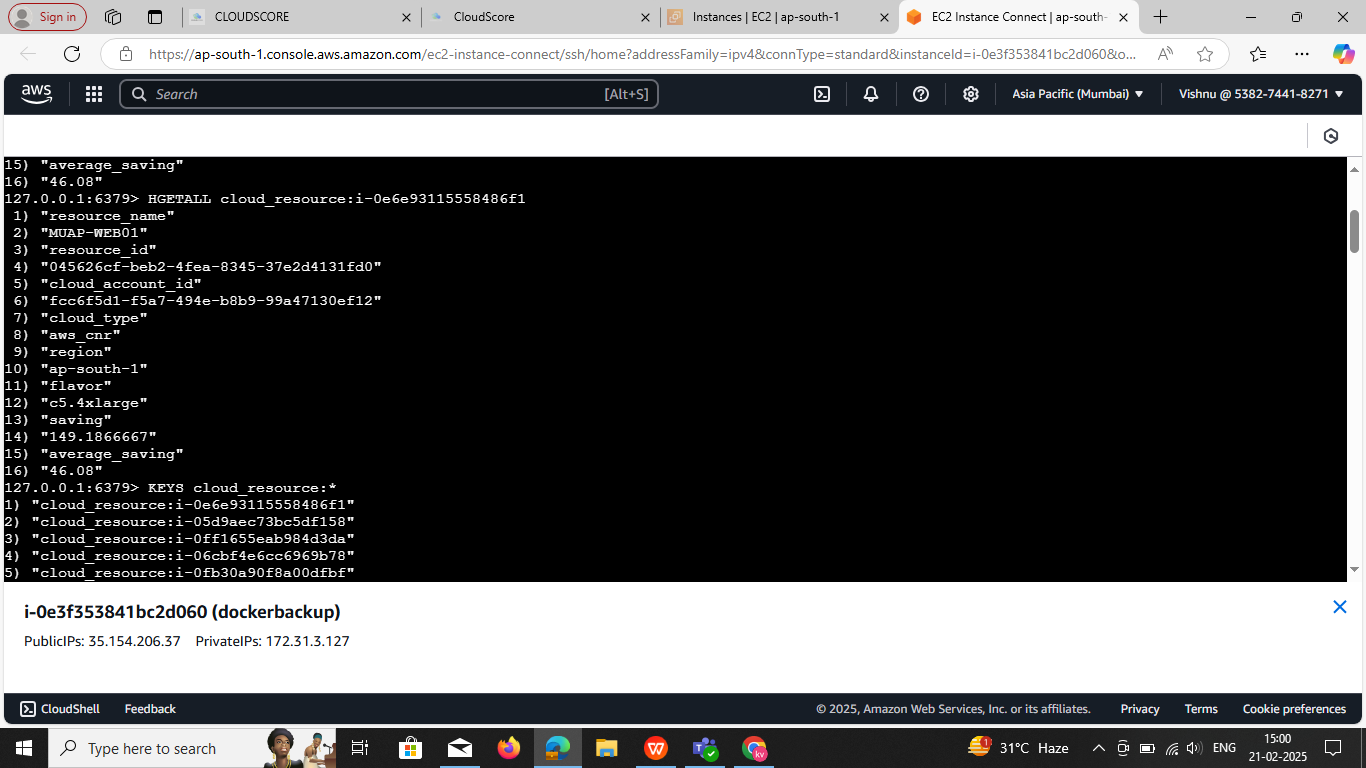
HSET cloud\_resource:i-0fb30a90f8a00dfbf resource\_name "MUAP-B2BDB01" resource\_id "443d9bc5-910e-4569-a9e3-158e31e54c8e" cloud\_account\_id "fcc6f5d1-f5a7-494e-b8b9-99a47130ef12" cloud\_type "aws\_cnr" region "ap-south-1" flavor "m4.xlarge" saving "88.71666667" average\_saving "15.552"

EXEC

### ****1. Get All Keys Matching**** cloud\_resource:\*

**KEYS cloud\_resource:\***

This will return all keys matching the pattern.

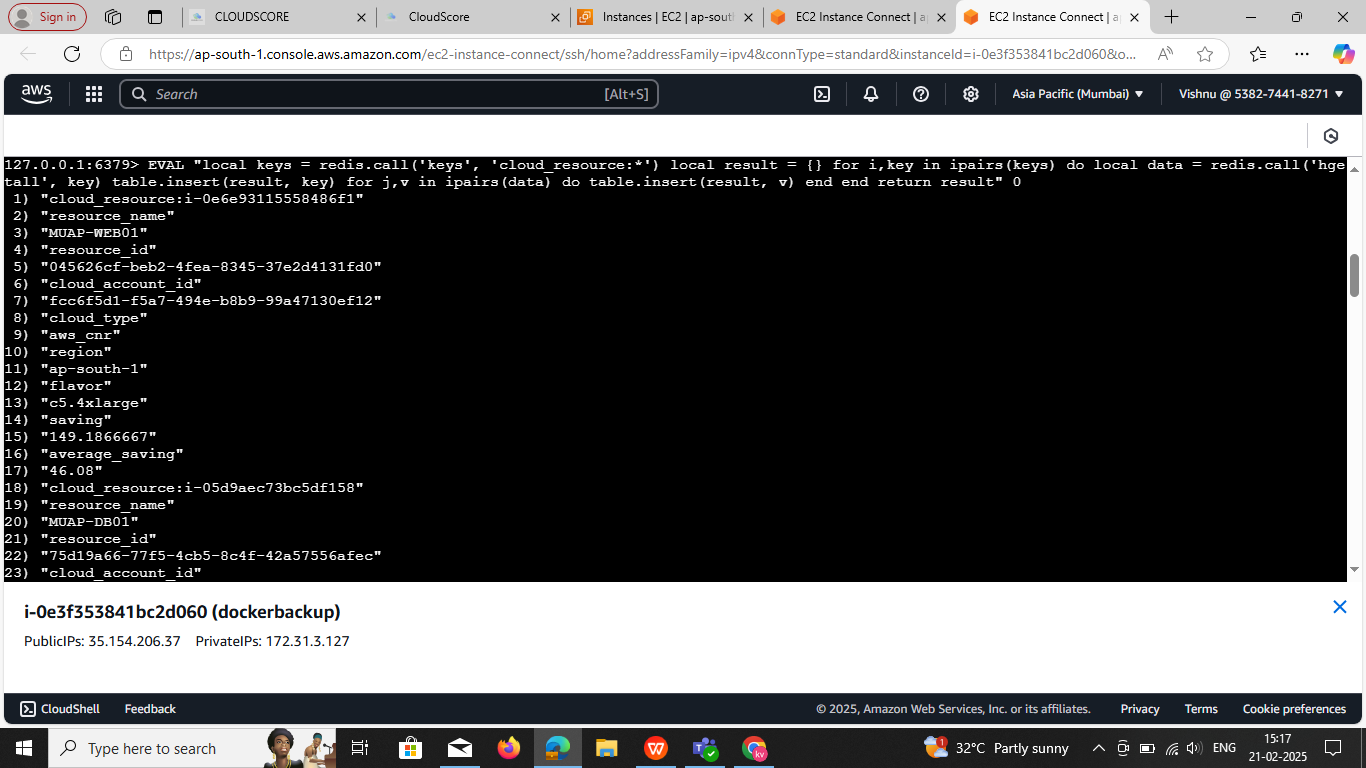


### ****Fetch All Data for Each Key in a Single Command****

### Use the EVAL command to loop over the keys and fetch all their fields and values in a single command:

**Command :**

EVAL "local keys = redis.call('keys', 'cloud\_resource:\*') local result = {} for i,key in ipairs(keys) do local data = redis.call('hgetall', key) table.insert(result, key) for j,v in ipairs(data) do table.insert(result, v) end end return result" 0



**Alternative: Use SCAN Instead of KEYS (Recommended for Large Datasets)**

EVAL "local cursor = '0' local result = {} repeat local scan = redis.call('SCAN', cursor, 'MATCH', 'cloud\_resource:\*', 'COUNT', 1000) cursor = scan[1] for \_, key in ipairs(scan[2]) do local data = redis.call('HGETALL', key) table.insert(result, key) for j,v in ipairs(data) do table.insert(result, v) end end until cursor == '0' return result" 0

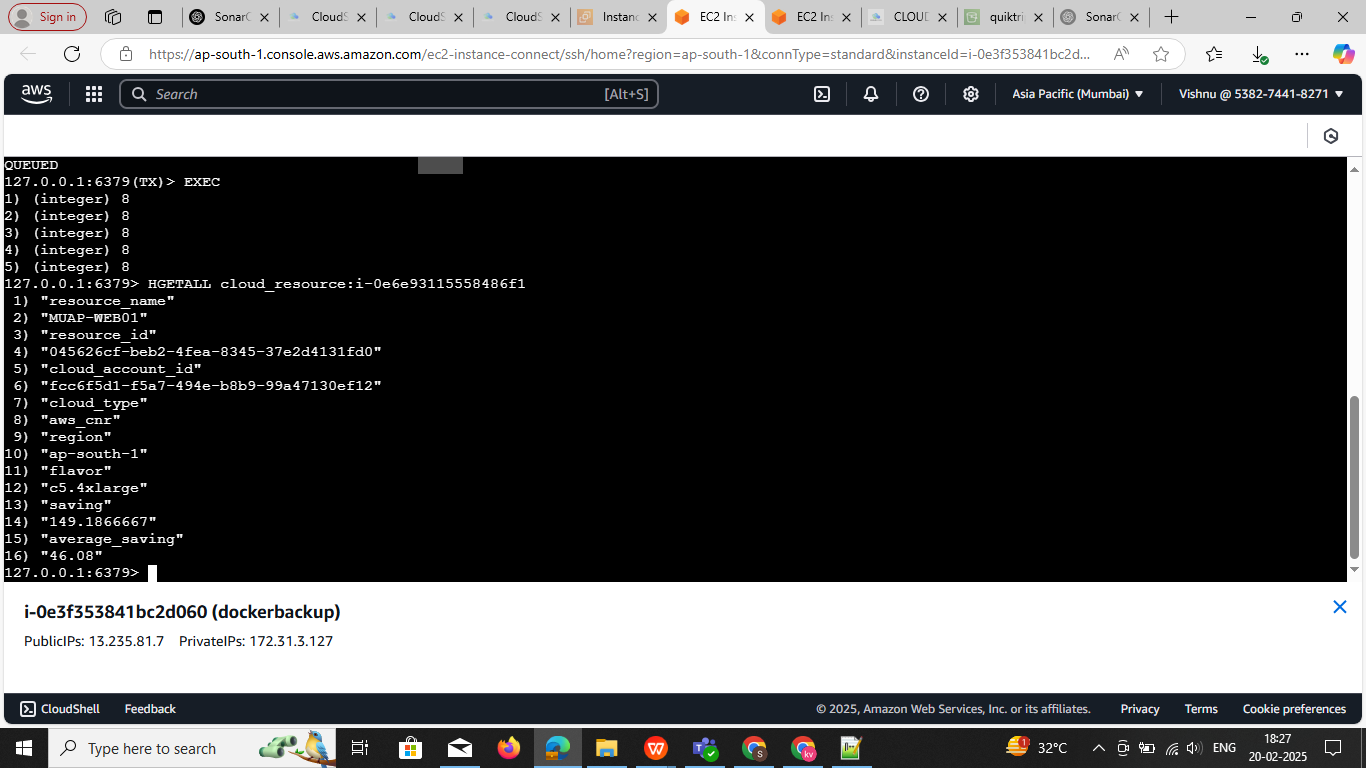
This method prevents performance issues if you have a large number of keys.**Expected Output** The response will include all keys and their corresponding hash field-value pairs.



## ****3 Verify Data in Redis****

To check if the data was inserted correctly, run:

HGETALL cloud\_resource:**i-0e6e93115558486f1 change ids single id**



## ****Verify Data in Redis****

To check if the data was inserted correctly, run:

**HGETALL cloud\_resource:i-0e6e93115558486f1**

✔️ **Output:**

1) "resource\_name"2) "MUAP-WEB01"3) "resource\_id"4) "045626cf-beb2-4fea-8345-37e2d4131fd0"5) "cloud\_account\_id"6) "fcc6f5d1-f5a7-494e-b8b9-99a47130ef12"7) "cloud\_type"8) "aws\_cnr"9) "region"10) "ap-south-1"11) "flavor"12) "c5.4xlarge"13) "saving"14) "149.1866667"15) "average\_saving"16) "46.08"

## ****Bonus:Import Data from a CSV File into Redis****

If you have a **CSV file** with your data, you can use a Python script to import it into Redis.

### ****1Create**** data.csv

cloud\_resource\_id,resource\_name,resource\_id,cloud\_account\_id,cloud\_type,region,flavor,saving,average\_saving

i-0e6e93115558486f1,MUAP-WEB01,045626cf-beb2-4fea-8345-37e2d4131fd0,fcc6f5d1-f5a7-494e-b8b9-99a47130ef12,aws\_cnr,ap-south-1,c5.4xlarge,149.1866667,46.08

i-0ff1655eab984d3da,MUAP-B2BAPI01,13e94cd4-bb44-4ef4-8dee-e9884bcdbb54,fcc6f5d1-f5a7-494e-b8b9-99a47130ef12,aws\_cnr,ap-south-1,r5.2xlarge,109.7766667,57.6

### ****2️⃣ Run a Python Script to Import****

python

CopyEdit

import csvimport redis

# Connect to Redis

r = redis.Redis(host='localhost', port=6379, decode\_responses=True)

# Open CSV filewith open('data.csv', 'r') as file:

reader = csv.DictReader(file)

for row in reader:

key = f"cloud\_resource:{row['cloud\_resource\_id']}"

r.hset(key, mapping=row)

print("Data imported successfully!")

**--------------------------------------------------**MongoDB**--------------------------------------------------------**

## ****Step 1: Log In to MongoDB****

If you are using **Docker**, first enter your MongoDB container:

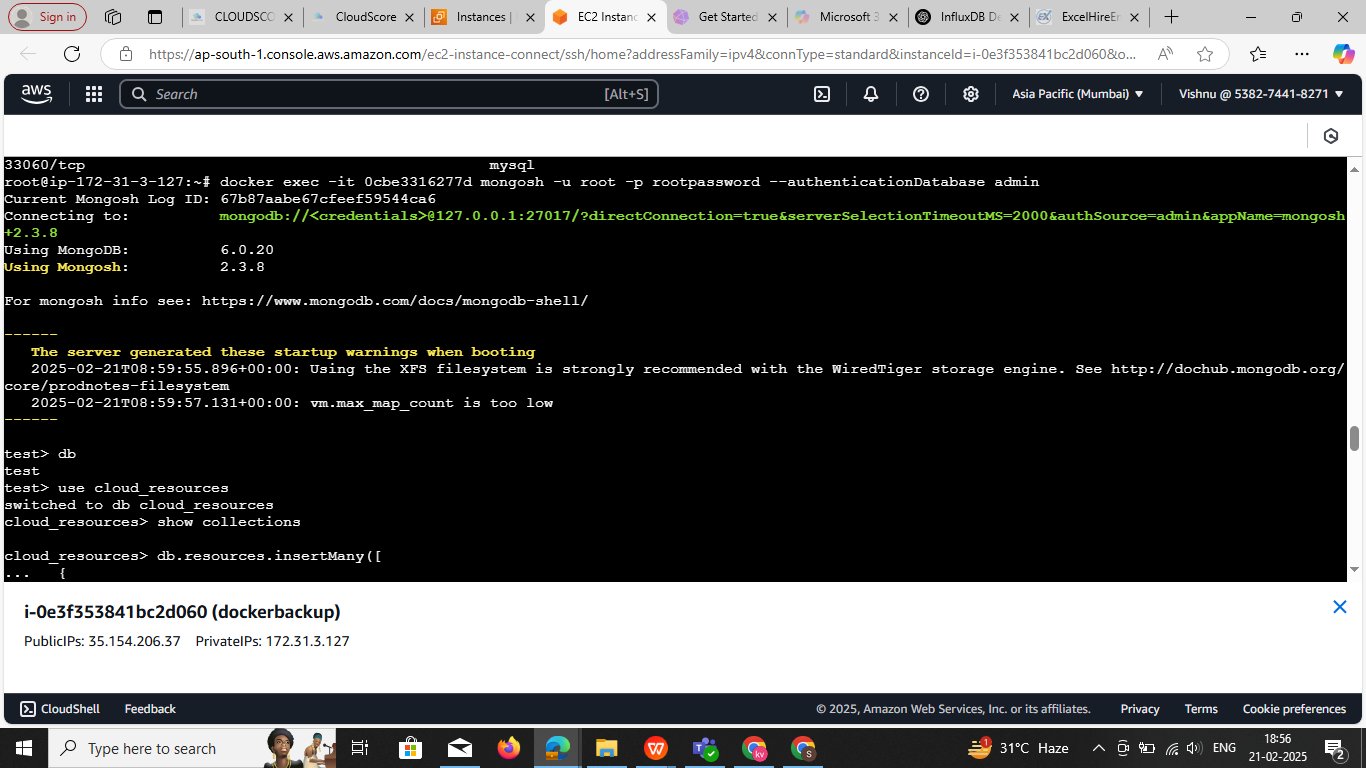
**docker exec -it <container\_id> mongosh -u root -p rootpassword --authenticationDatabase admin**

(Replace <container\_id> with your actual container ID from docker ps.)

**If MongoDB is running locally, just run:**

**mongosh -u root -p rootpassword --authenticationDatabase admin**

**Note : if using dockercompose in file password and user name is set**



### **Explanation:**

1. docker exec -it <container\_id>: Runs a command inside a running container interactively.
2. mongosh: Starts the MongoDB shell.
3. -u root: Specifies the username (root in this case).
4. -p rootpassword: Specifies the password (rootpassword here).
5. --authenticationDatabase admin: Specifies the database where authentication should occur (admin is the default for root users).

## ****Step 2: Select Your Database and create DB****

Once logged in, switch to your database. For example, if your database is cloud\_resources:

**Use<name> exp: use cloud\_resources**



## ****Step 3: Insert Data (Create a Collection)****

In MongoDB, collections are **automatically created** when you insert the first document.

For example, to create a collection named servers and insert data:

**db.servers.insertOne({**

**name: "Server1",**

**type: "AWS EC2",**

**region: "ap-south-1",**

**status: "running"**

**})**

**To insert multiple documents:**

db.servers.insertMany([

{ name: "Server2", type: "AWS EC2", region: "us-east-1", status: "stopped" },

{ name: "Server3", type: "Azure VM", region: "eu-west-1", status: "running" }

])

**Here is the **correct MongoDB insert Many format** for your cloud resources data:**

Upload multiple data in

db.resources.insertMany([

{

cloud\_resource\_id: "i-0ff1655eab984d3da",

resource\_name: "MUAP-B2BAPI01",

resource\_id: "13e94cd4-bb44-4ef4-8dee-e9884bcdbb54",

cloud\_account\_id: "fcc6f5d1-f5a7-494e-b8b9-99a47130ef12",

cloud\_type: "aws\_cnr",

region: "ap-south-1",

flavor: "r5.2xlarge",

saving: 109.7766667,

average\_saving: 57.6

},

{

cloud\_resource\_id: "i-05d9aec73bc5df158",

resource\_name: "MUAP-DB01",

resource\_id: "75d19a66-77f5-4cb5-8c4f-42a57556afec",

cloud\_account\_id: "fcc6f5d1-f5a7-494e-b8b9-99a47130ef12",

cloud\_type: "aws\_cnr",

region: "ap-south-1",

flavor: "r5.2xlarge",

saving: 100.1433333,

average\_saving: 57.6

},

{

cloud\_resource\_id: "i-06cbf4e6cc6969b78",

resource\_name: "MUAN-BWEB01",

resource\_id: "bfe6e76b-e1de-4e0a-9824-4f3da3f17ab0",

cloud\_account\_id: "fcc6f5d1-f5a7-494e-b8b9-99a47130ef12",

cloud\_type: "aws\_cnr",

region: "ap-south-1",

flavor: "t3.2xlarge",

saving: 91.782,

average\_saving: 28.728

}

])

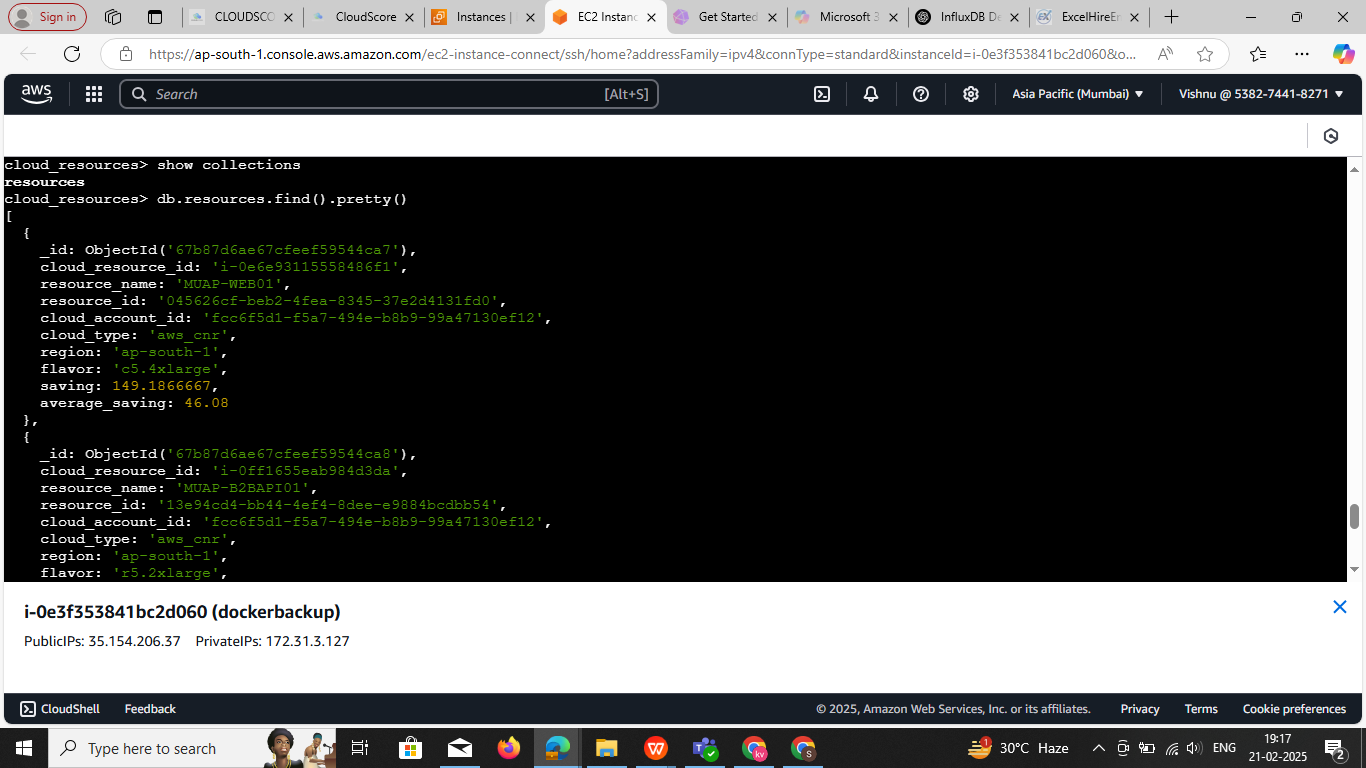
## ****Step 4: Verify Collections Exist****

Check if the collection was created:

**show collections**

To see the data inside your collection:

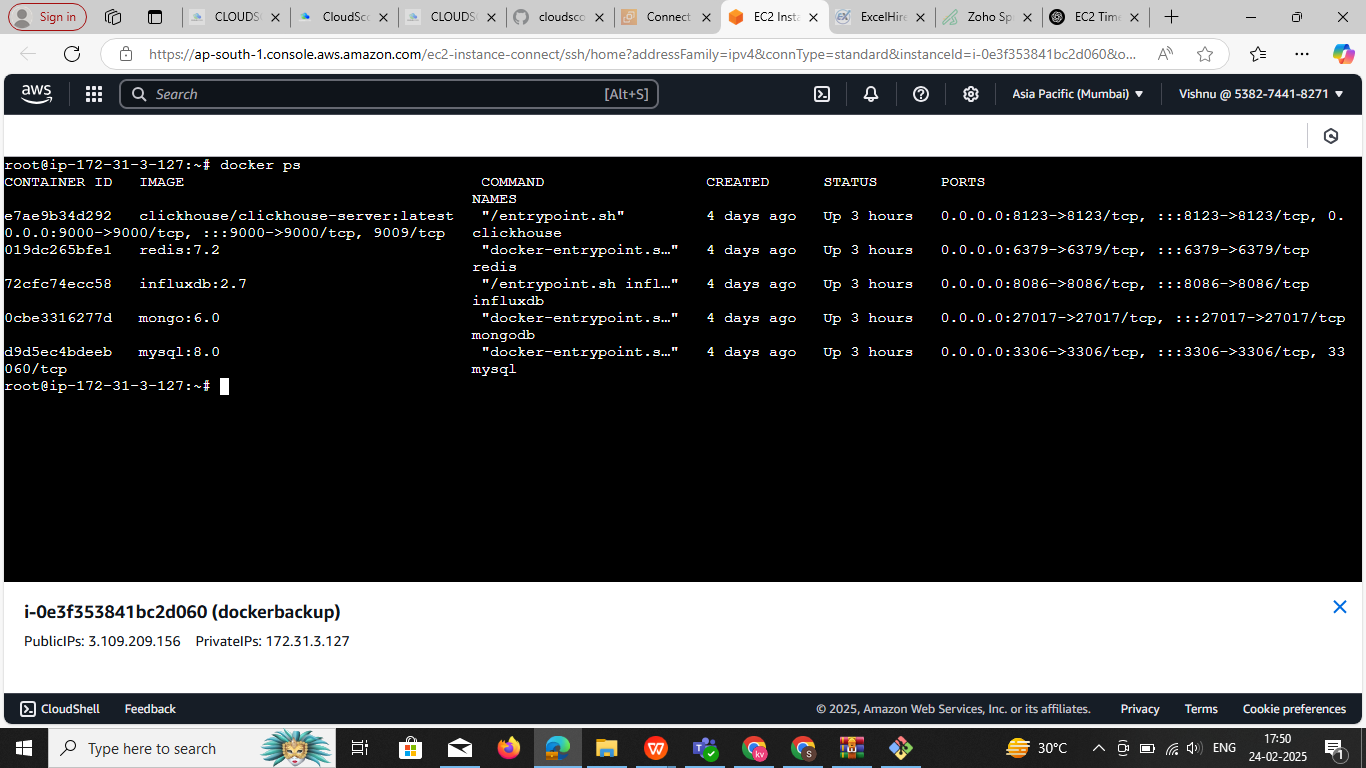
**db.servers.find().pretty()** change collection name



**----------------- -----------------MYSQL-----------------------------**

to check the list of container

**Step1 :** check continer and login the mysql



**Step2:**

**Login the mysql**

**docker exec -it root-mysql-1 mysql -u root -p docker exec -it root-mysql-1 mysql -u root**

· **Check container environment variables**: if your user name and pws not work go root password

docker inspect d9d5ec4bdeeb | grep MYSQL\_ROOT\_PASSWORD

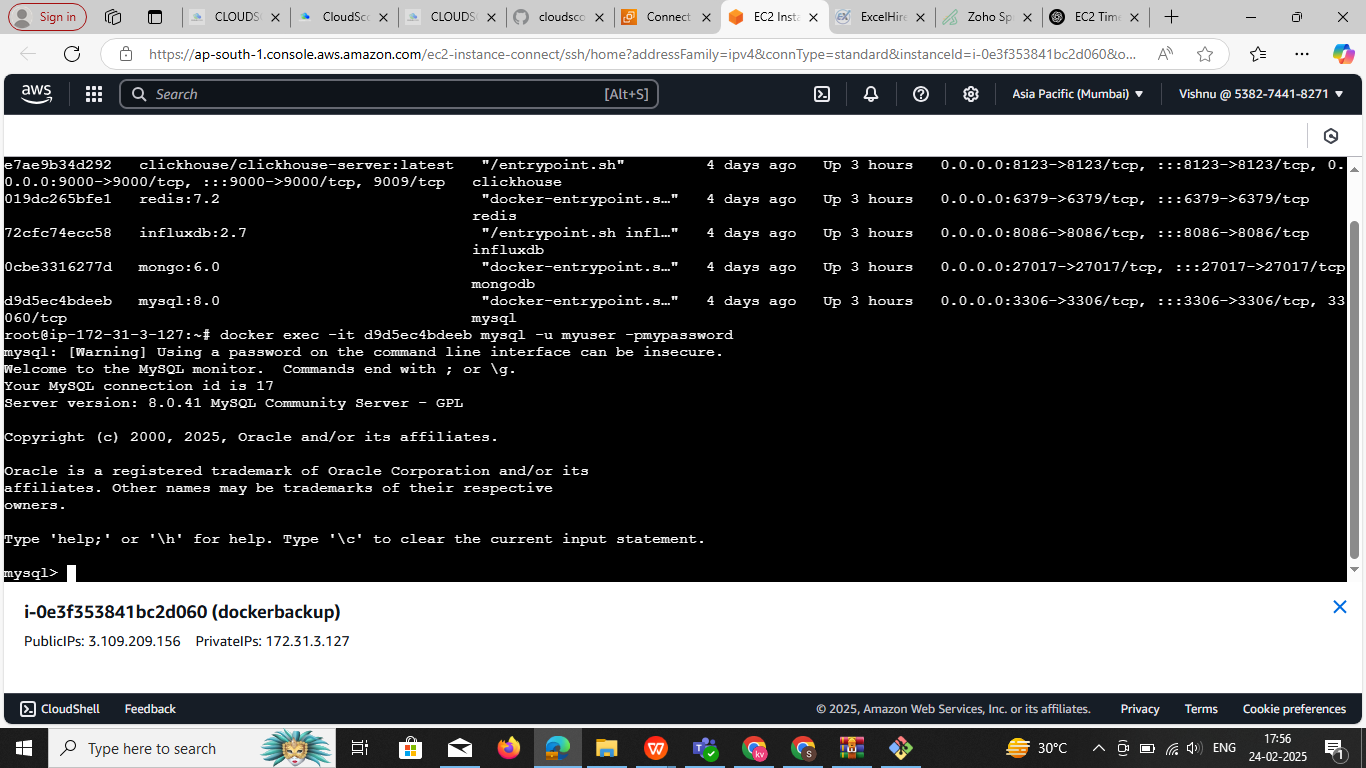
**Look at container logs**:

docker logs d9d5ec4bdeeb | grep -i password

**docker exec -it d9d5ec4bdeeb mysql -u myuser -p mypassword**  if it not work

**docker exec -it d9d5ec4bdeeb mysql -u myuser -pmypassword used this command**

**Note:** try this 3 commands if you using docker compose you have password and user names in file

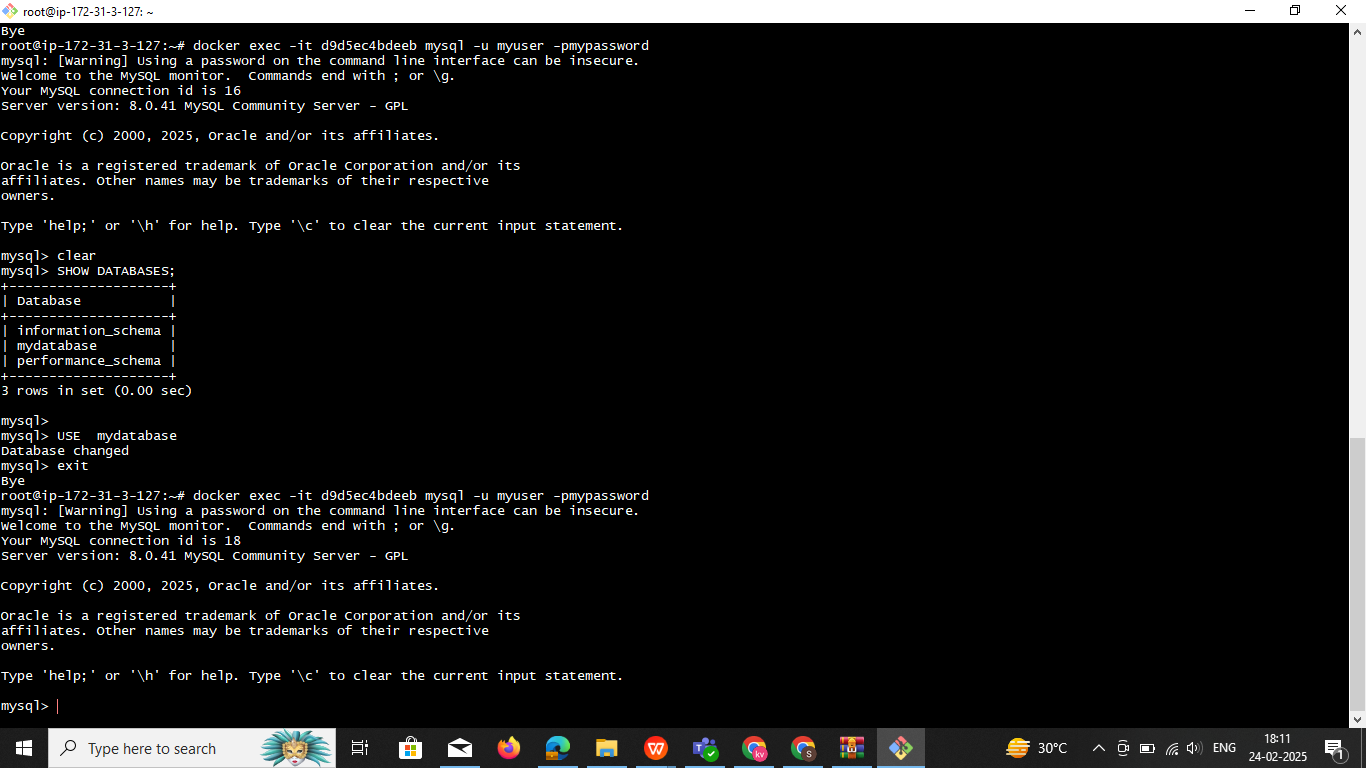


**3:Check Available Databases**

SHOW DATABASES;

You'll see a list of databases. If you created mydatabase (from your docker-compose.yml), switch to it:

USE mydatabase;



You can exit

### 1 ****If No Database Exists? Create One****

If you don’t have a database yet, create one:

Check the fist 1 SHOW DATABASES;

**CREATE DATABASE mydatabase;**

**USE mydatabase;**

Note :

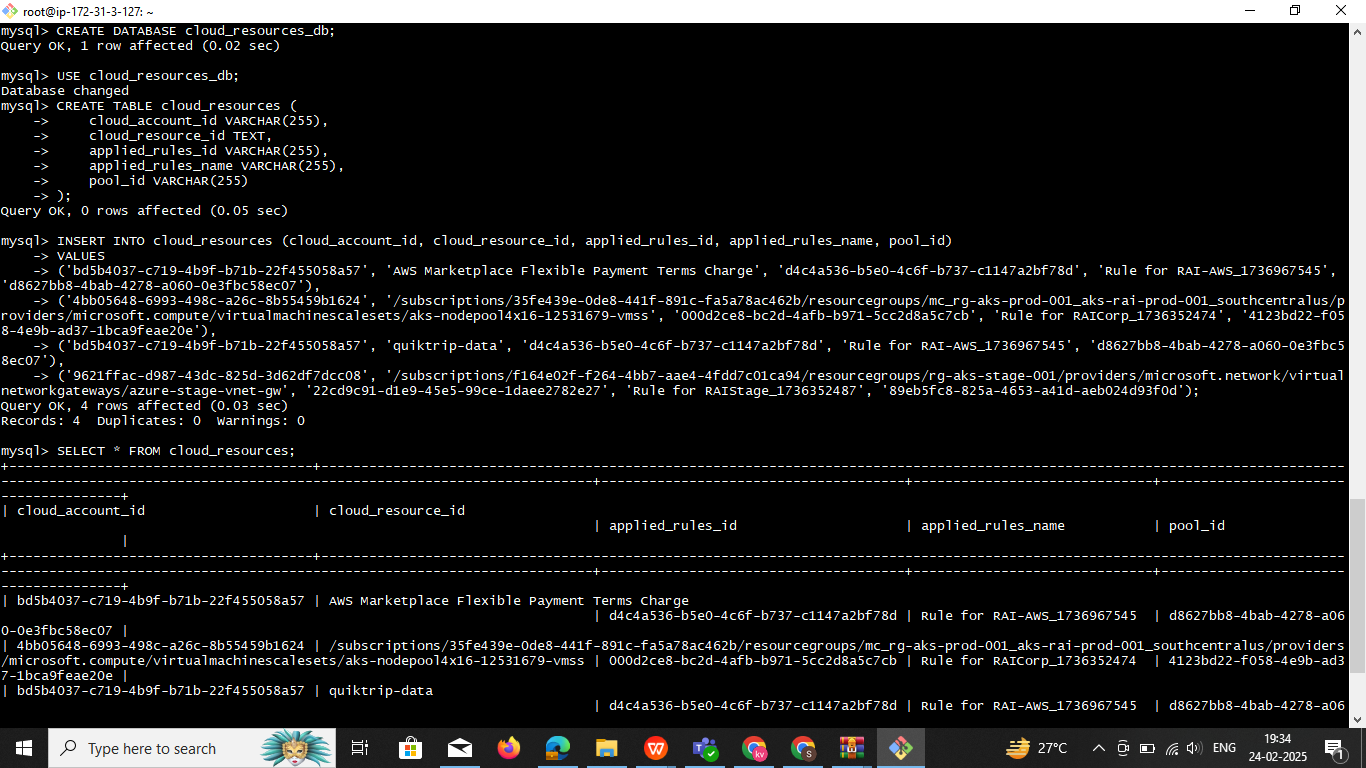
---------------------------------------------------\*\*\*\*\*set-by-seps\*\*\*\*\*---------------------------------------------------

### ****Step 1: Create a New Database****

Run the following SQL command inside MySQL:

CREATE DATABASE cloud\_resources\_db;

USE cloud\_resources\_db;



### ****Step 2: Create the Table****

Now, create a table called cloud\_resources:

CREATE TABLE cloud\_resources (

cloud\_account\_id VARCHAR(255),

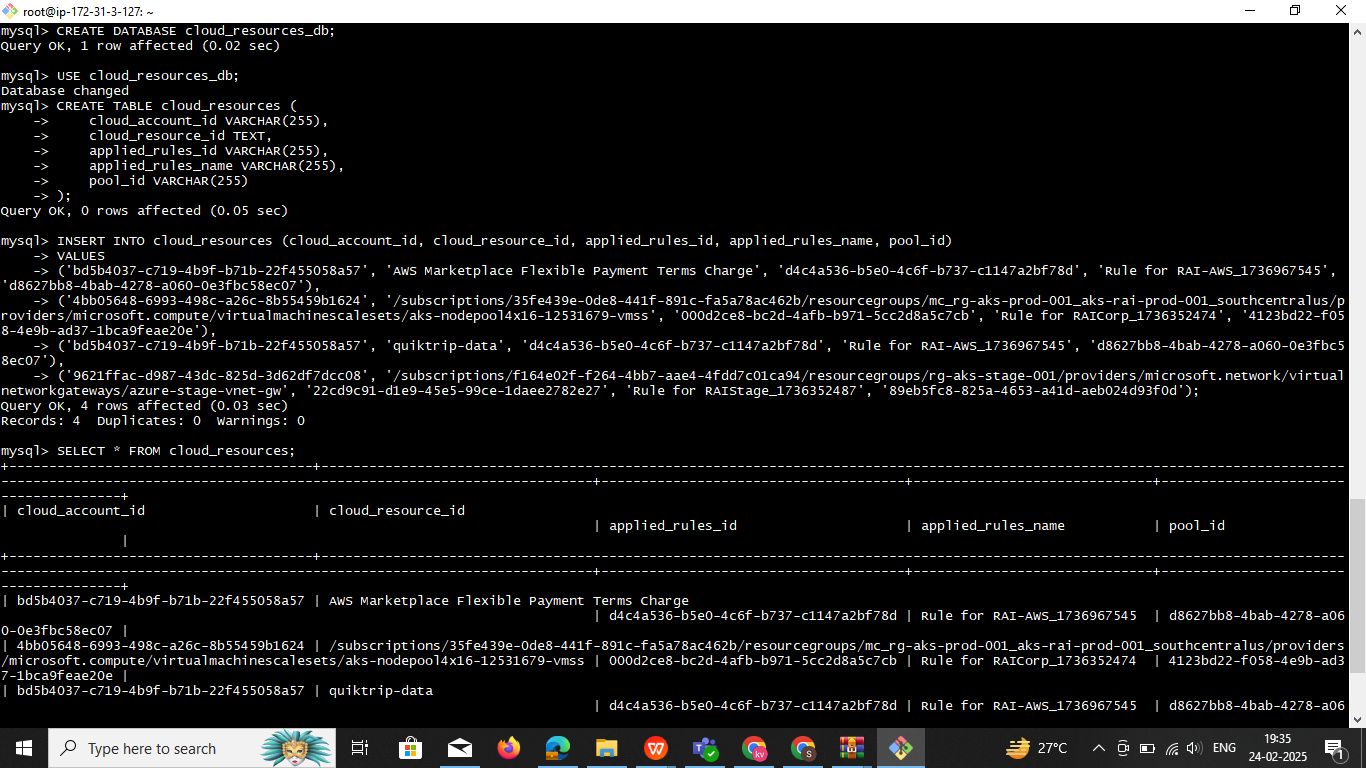
cloud\_resource\_id TEXT,

applied\_rules\_id VARCHAR(255),

applied\_rules\_name VARCHAR(255),

pool\_id VARCHAR(255)

);



### ****Step 3: Insert Multiple Records****

Now, insert multiple records at once:

INSERT INTO cloud\_resources (cloud\_account\_id, cloud\_resource\_id, applied\_rules\_id, applied\_rules\_name, pool\_id)

VALUES

('bd5b4037-c719-4b9f-b71b-22f455058a57', 'AWS Marketplace Flexible Payment Terms Charge', 'd4c4a536-b5e0-4c6f-b737-c1147a2bf78d', 'Rule for RAI-AWS\_1736967545', 'd8627bb8-4bab-4278-a060-0e3fbc58ec07'),

('4bb05648-6993-498c-a26c-8b55459b1624', '/subscriptions/35fe439e-0de8-441f-891c-fa5a78ac462b/resourcegroups/mc\_rg-aks-prod-001\_aks-rai-prod-001\_southcentralus/providers/microsoft.compute/virtualmachinescalesets/aks-nodepool4x16-12531679-vmss', '000d2ce8-bc2d-4afb-b971-5cc2d8a5c7cb', 'Rule for RAICorp\_1736352474', '4123bd22-f058-4e9b-ad37-1bca9feae20e'),

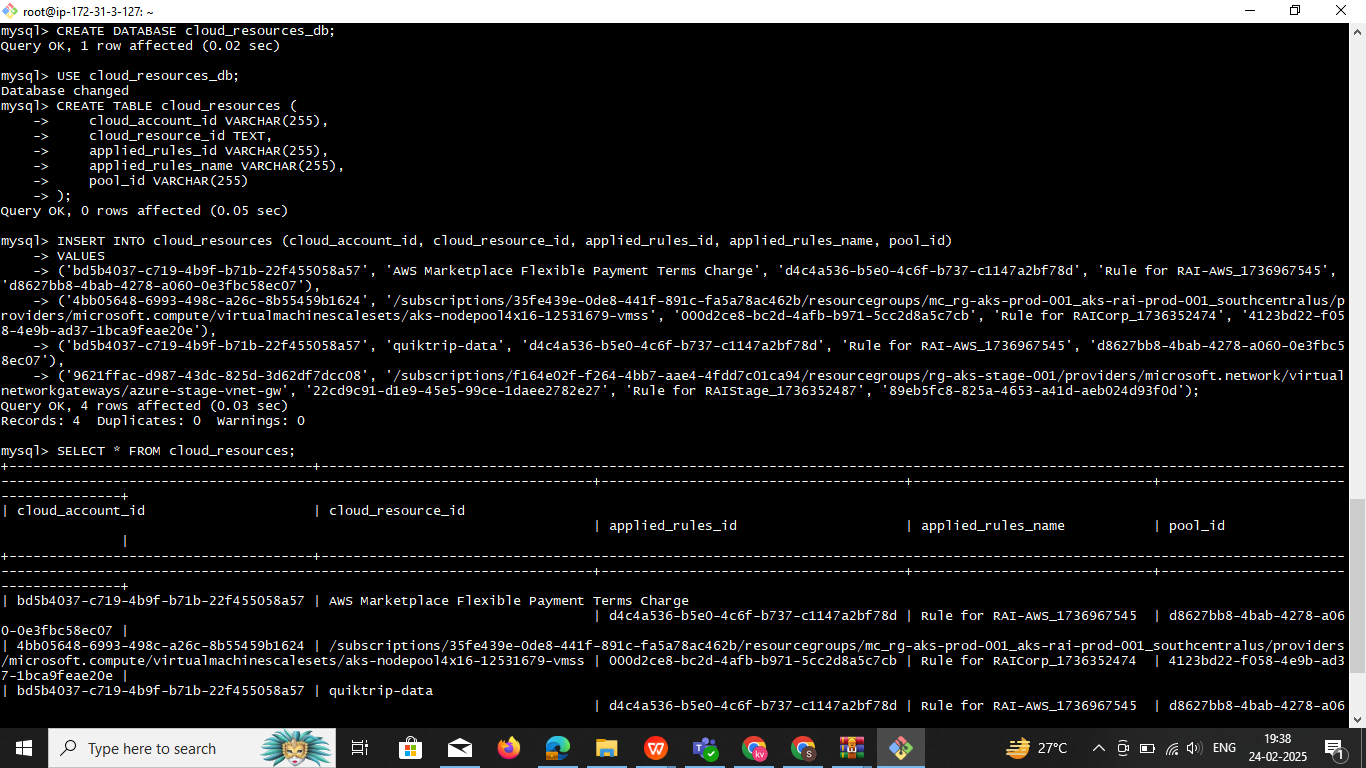
('bd5b4037-c719-4b9f-b71b-22f455058a57', 'quiktrip-data', 'd4c4a536-b5e0-4c6f-b737-c1147a2bf78d', 'Rule for RAI-AWS\_1736967545', 'd8627bb8-4bab-4278-a060-0e3fbc58ec07'),

('9621ffac-d987-43dc-825d-3d62df7dcc08', '/subscriptions/f164e02f-f264-4bb7-aae4-4fdd7c01ca94/resourcegroups/rg-aks-stage-001/providers/microsoft.network/virtualnetworkgateways/azure-stage-vnet-gw', '22cd9c91-d1e9-45e5-99ce-1daee2782e27', 'Rule for RAIStage\_1736352487', '89eb5fc8-825a-4653-a41d-aeb024d93f0d');

### ****Step 4: Verify the Data****

Check if the records were inserted successfully:

SELECT \* FROM cloud\_resources;



****Notes****

If you have a large dataset, you can use a CSV file to import the data.

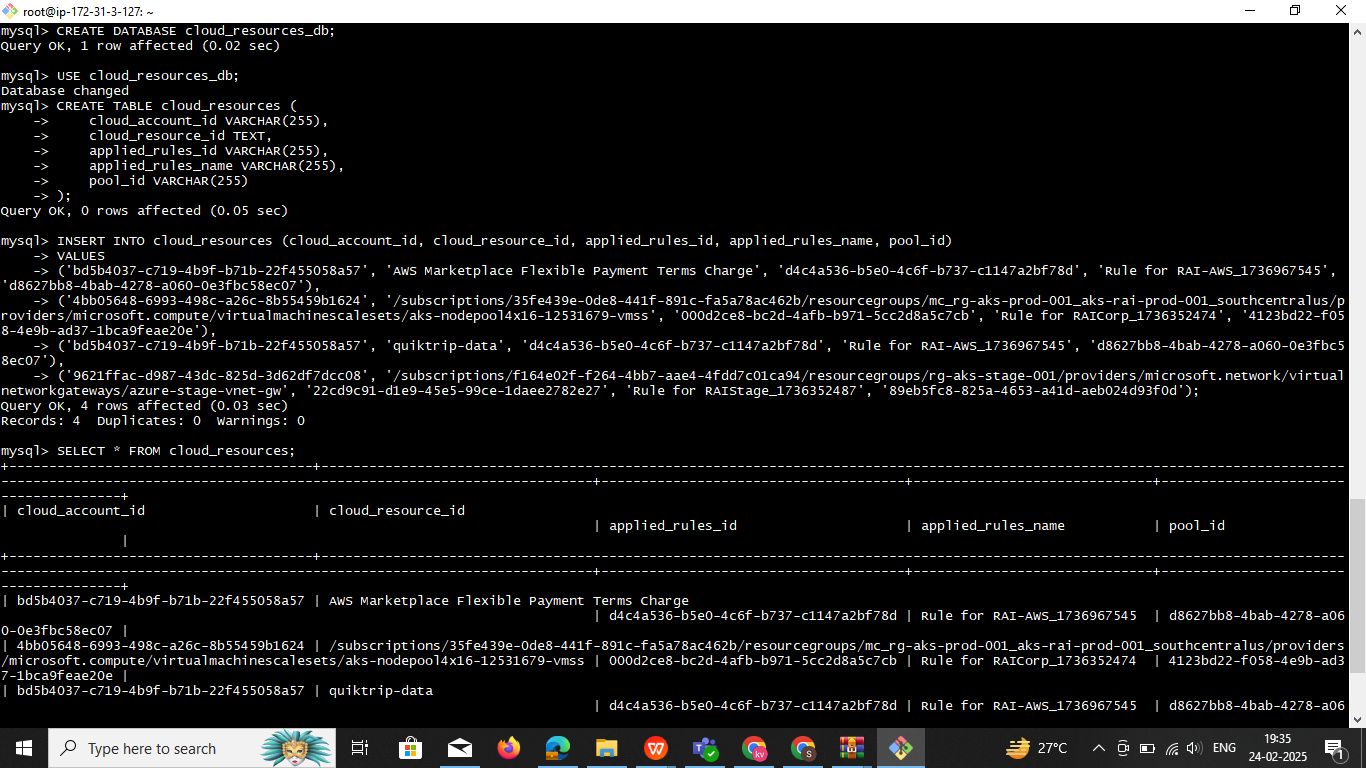
**If you need to update records, use:**

UPDATE cloud\_resources

SET applied\_rules\_name = 'New Rule Name'

WHERE cloud\_account\_id = 'bd5b4037-c719-4b9f-b71b-22f455058a57';

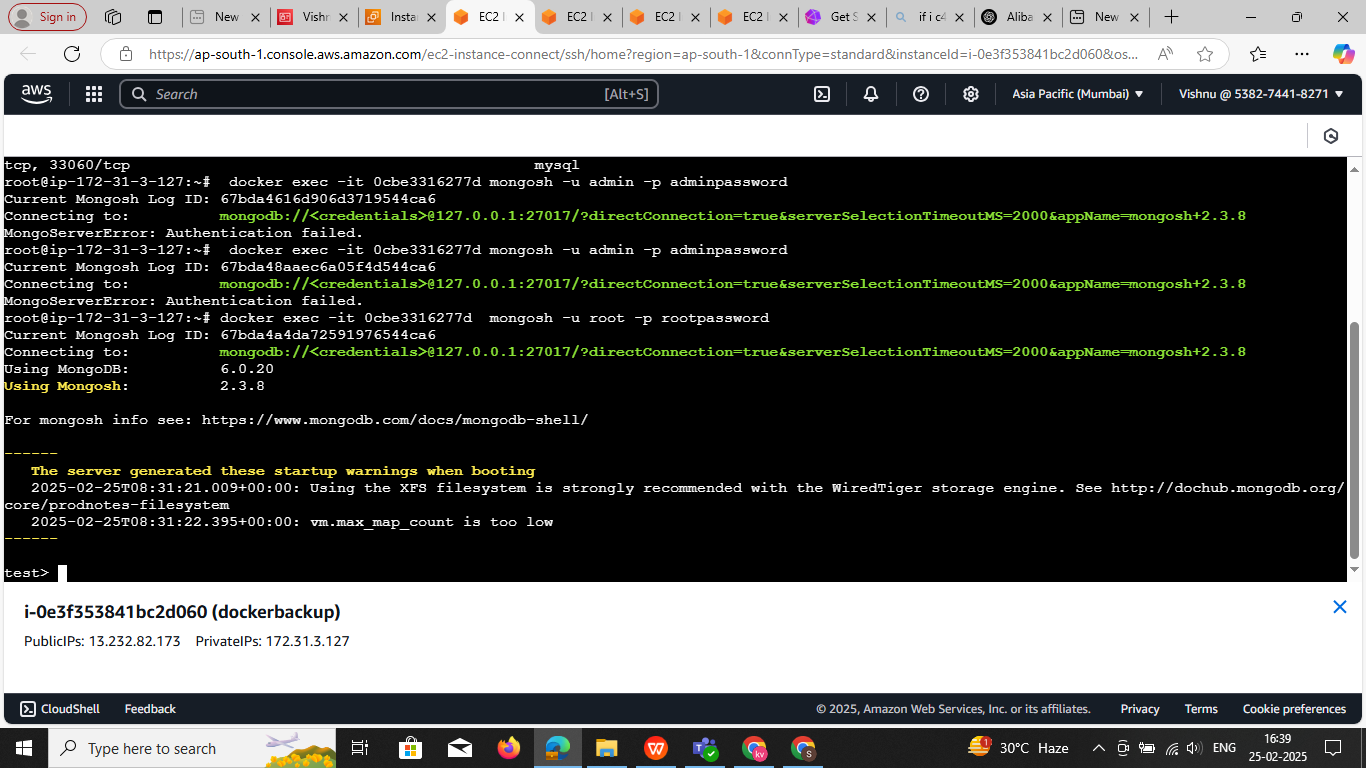
ALL commands in one image



-----------------------------------------**Backup MongoDB**------------------------------------

Step1: login to the mongo continer shell

Cmd : **docker exec -it 0cbe3316277d mongosh -u root -p rootpassword** need spaces and check



Note : if have all dbs and reco in create all redy go fallow this below commands if don’t know REF above page

## ****Backup MongoDB (Outside the Container)****

### ****Backup All Databases****

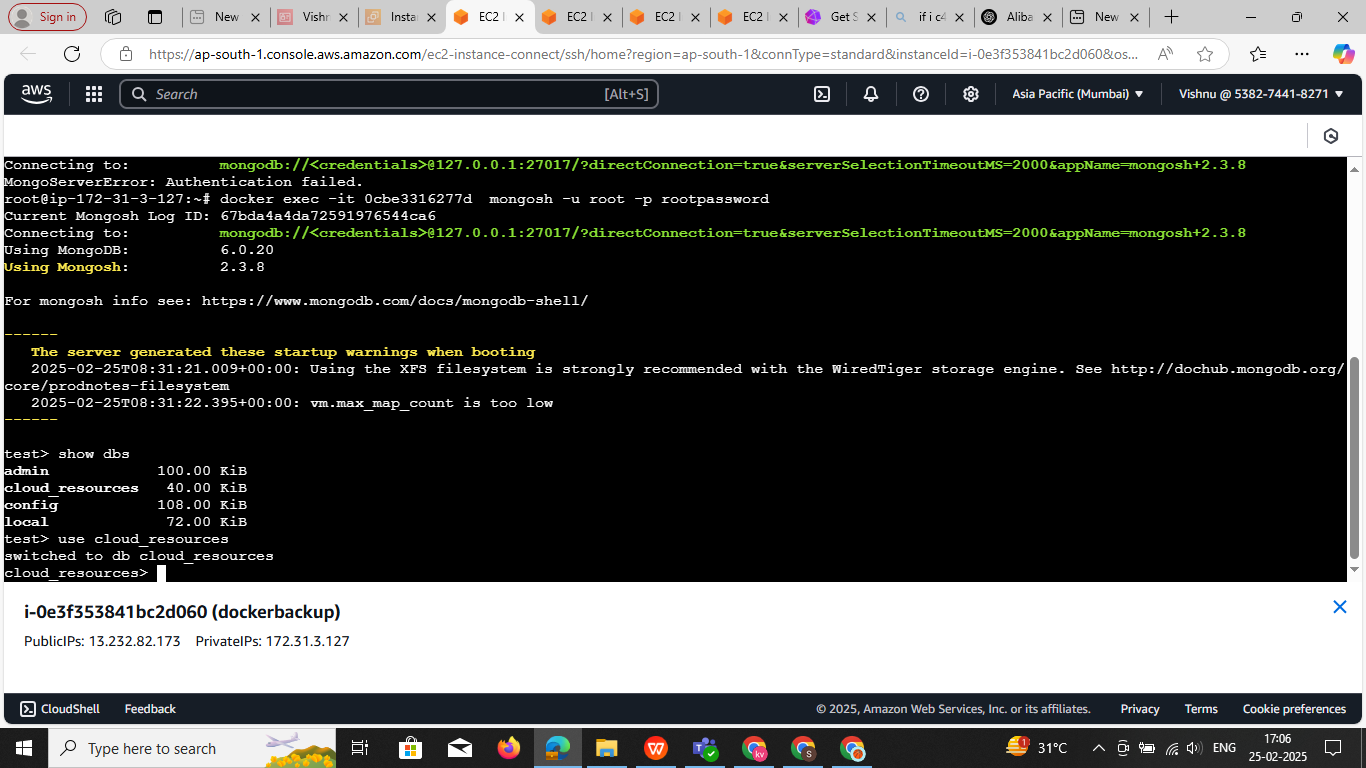
Fist create the folder in your working DIR in linux os

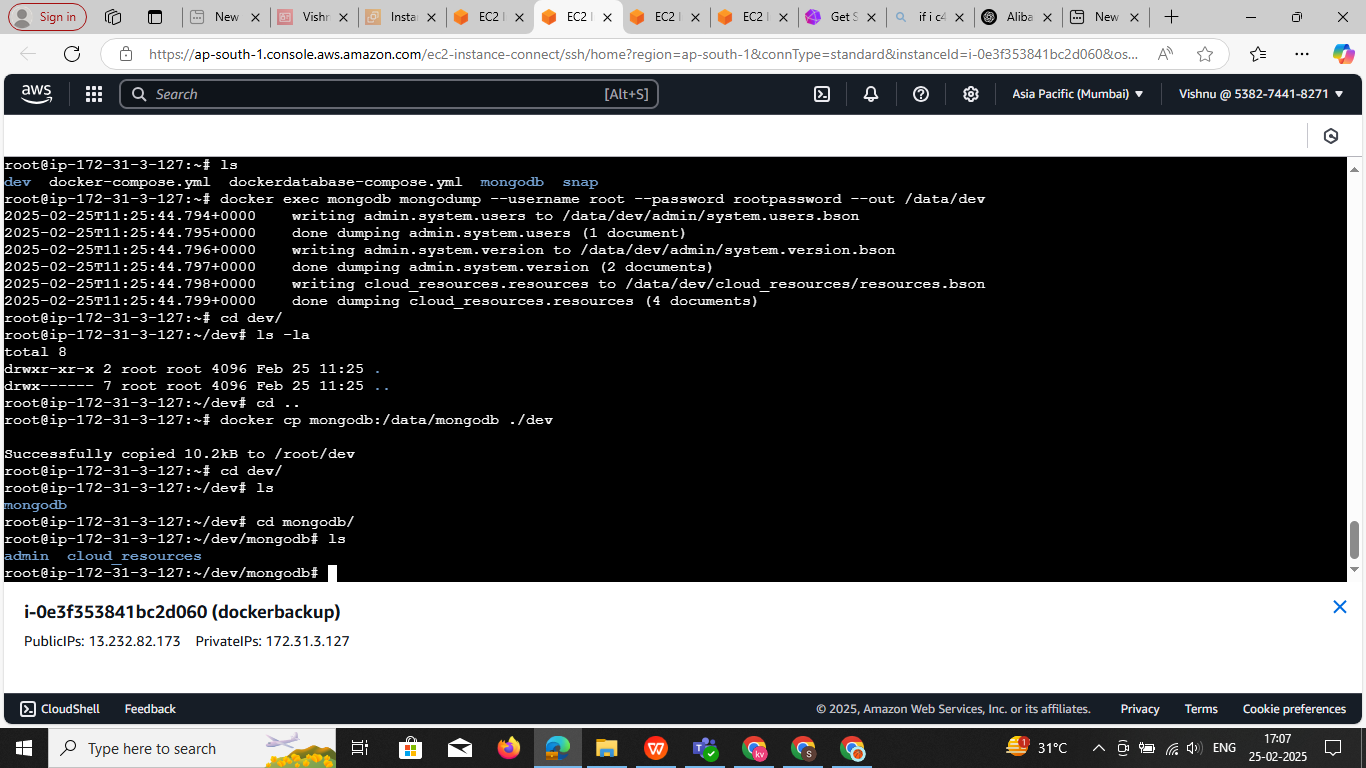
1docker exec mongodb mongodump --username root --password rootpassword --authenticationDatabase admin --out /data/**mongodb**

docker cp mongodb:/data/mongodb .**/mongodb**

**docker exec mongodb mongodump --username root --password rootpassword --out /data/dev**

**docker cp mongodb:/data/mongodb ./dev**



 **Ex1: All command in one screen**

The command you've written: docker cp mongodb:/data/mongodb ./mongodb

is copying data from the **/data/mongodb** directory inside the mongodb container to a local directory **./mongodb** on your host system.

If you want to **change the destination directory** on your local system or modify the source path inside the container, you can adjust the paths as follows:

### To change the ****destination directory****:

For example, if you want to copy the MongoDB data into a folder named backup instead of ./mongodb, modify the destination like this:

**docker cp mongodb:/data/mongodb ./backup**

This command will copy the MongoDB data from the /data/mongodb directory inside the container to the ./backup directory on your host system.

### To change the ****source directory**** inside the container:

If you want to copy data from a different directory inside the container, you can change the source part of the command. For example, if you want to copy from /var/lib/mongo instead of /data/mongodb, you can run:

docker cp mongodb:/var/lib/mongo ./mongodb

### Summary:

* The **source path** (mongodb:/data/mongodb) is the directory inside the container you want to copy from.
* The **destination path** (./mongodb) is the directory on your local system where the data will be copied.

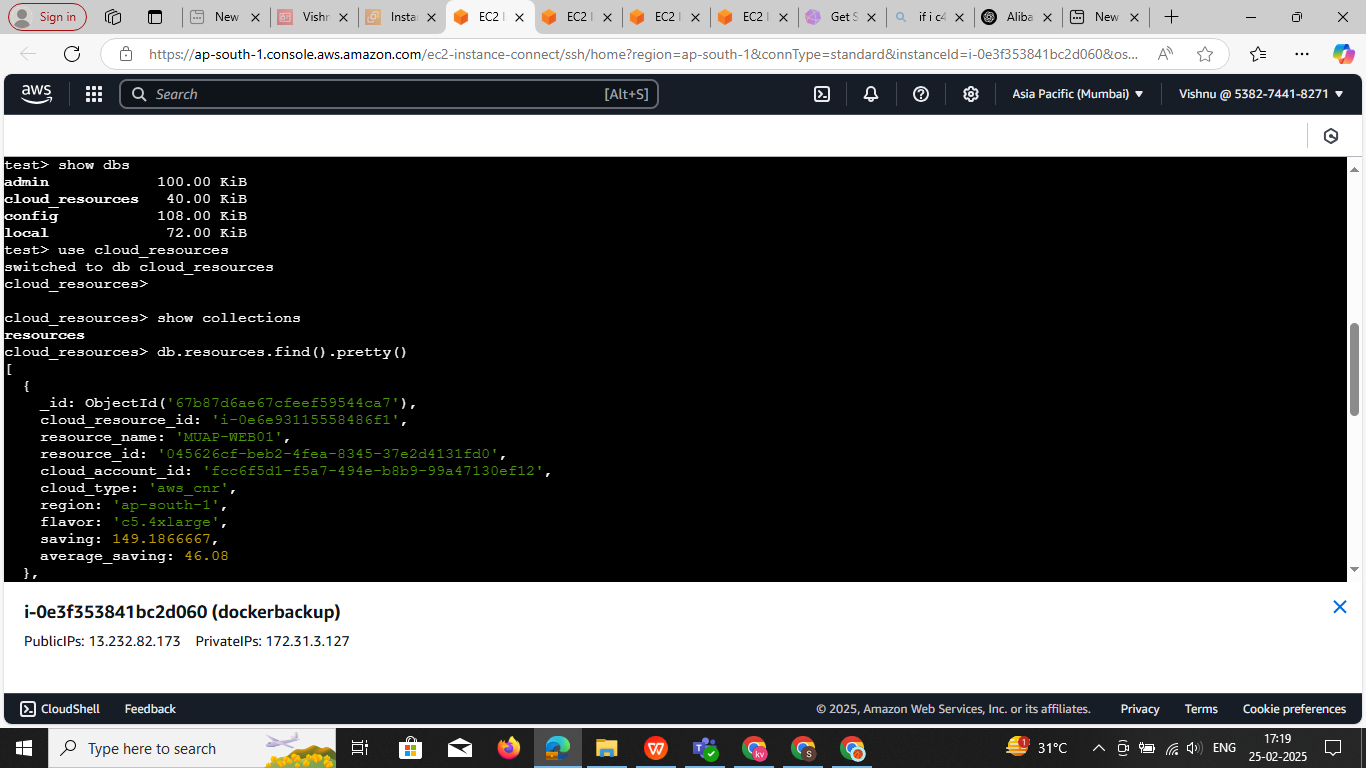
**Step2: validate the collection name are good**

Check if the collection was created:

**show collections**

To see the data inside your collection:

**db.servers.find().pretty()** change collection name



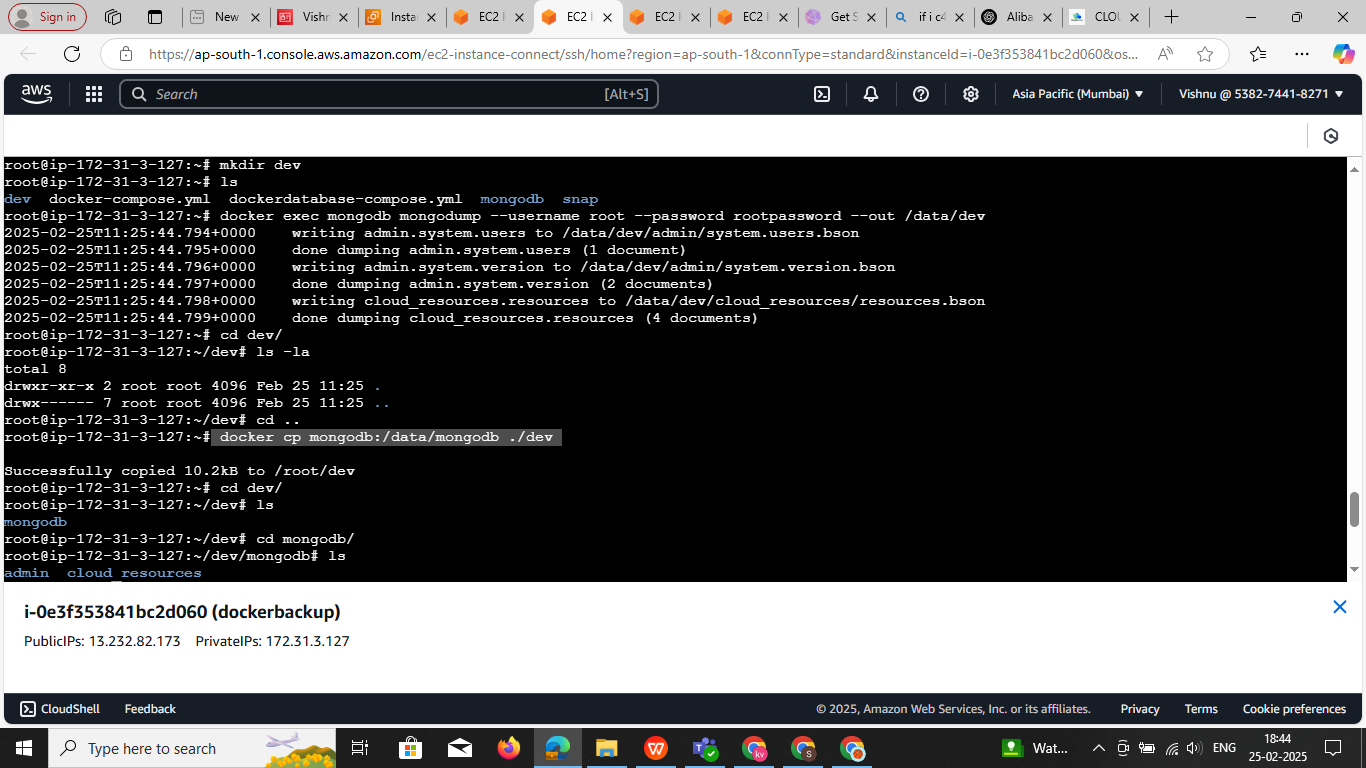
**Step3:now we dump the data out side dir**

Creatre the **dir** and now dump **specf dir** : dev check the below image

**docker exec mongodb mongodump --username root --password rootpassword --out /data/dev**

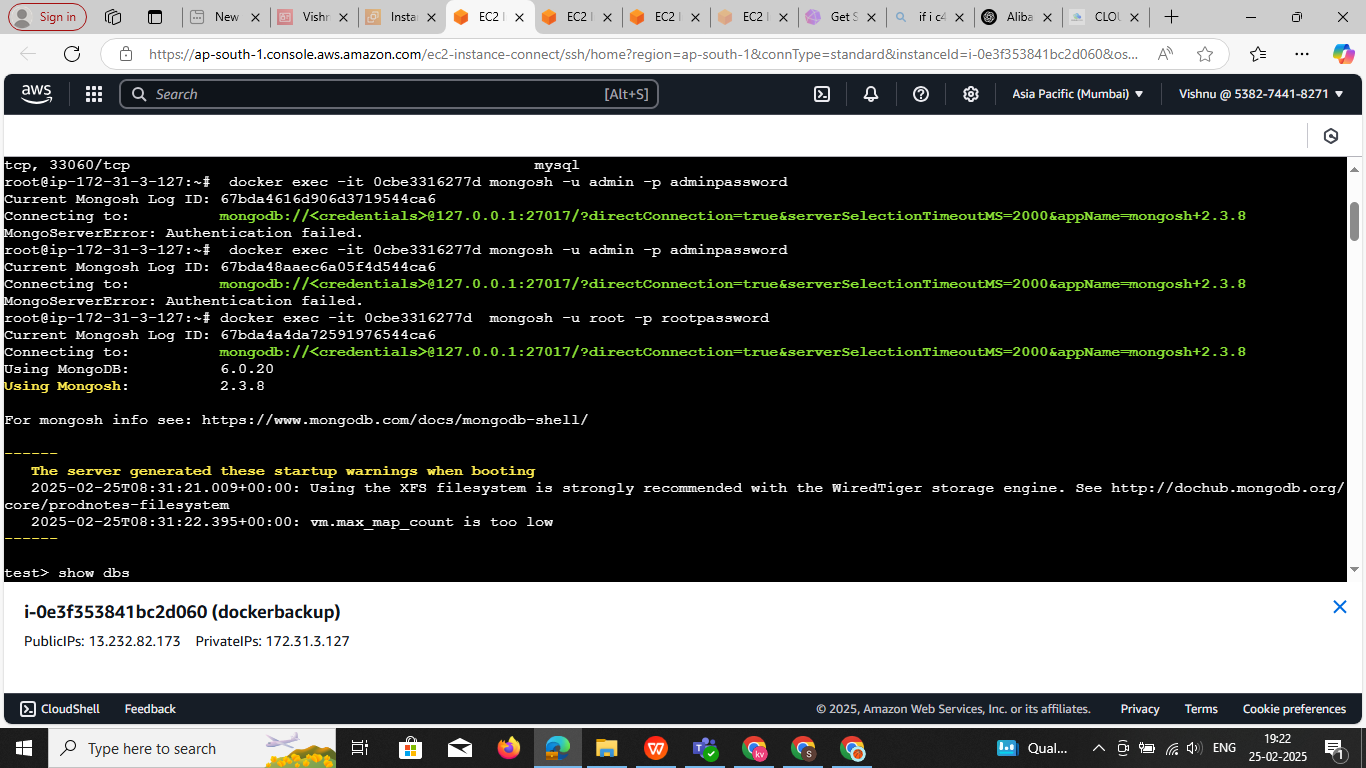
**docker cp mongodb:/data/mongodb ./dev**

**Note** It will take all data base form the mongo db to particulate DIR

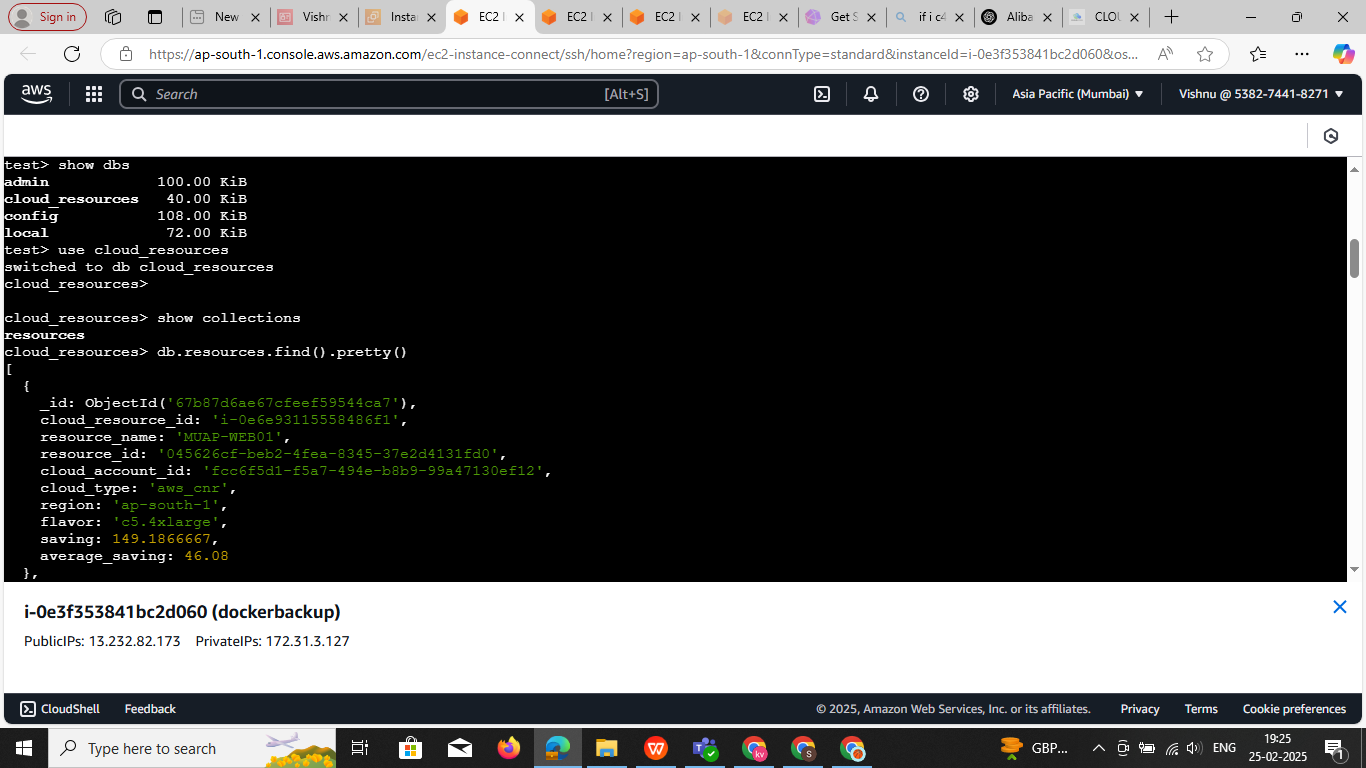


Step 4:

Now login to mongo sh



Check the DB and switch in to db switch the collection collection and

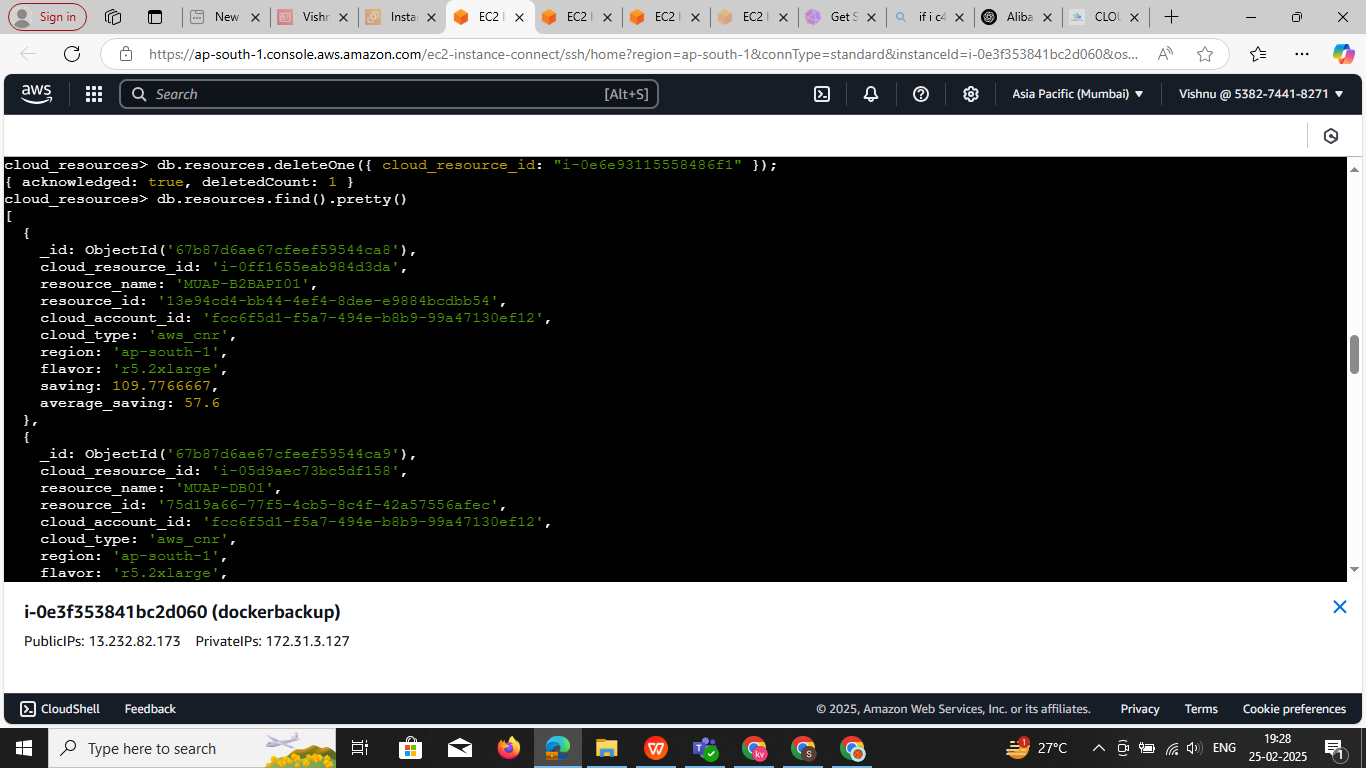


Commands : **show dbs**

**User cloud\_resources**

**show collections**

**Step 5:** now delete the one of the record

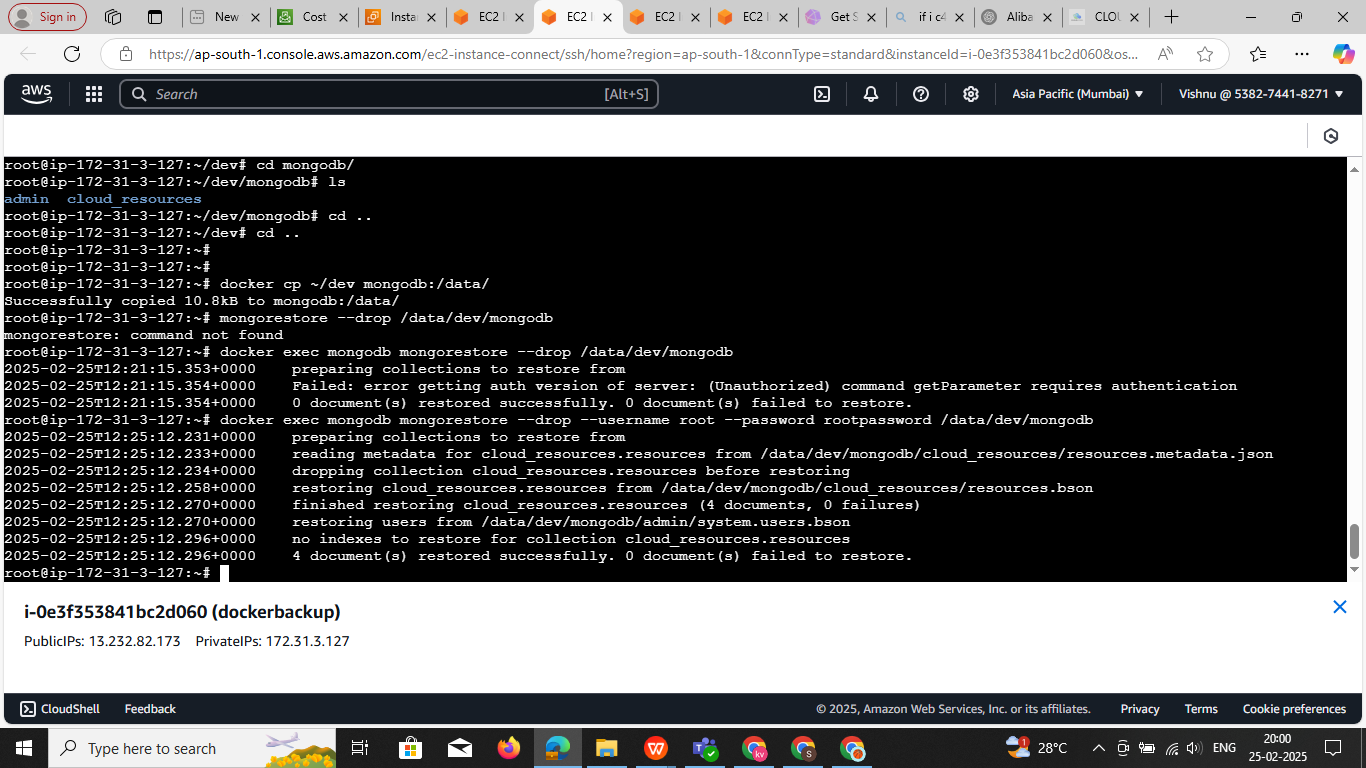


**Note :** I have 4 record now I delete one recod now I have 3 noe we need to restore the data and rec

**Step6: Now new need to restore the record commands**

docker cp ~/**dev** mongodb:/data/

docker exec mongodb mongorestore --drop /data/**dev**/mongodb

****

**Note :** now deleted data is in restore

cloud\_resources> show collections

resources

cloud\_resources> db.resources.find().pretty()

----------------------------------------------- -----------------------------------------------------------------------

**Login :** **docker exec -it 0cbe3316277d mongosh -u root -p rootpassword**

1:show dbs

2: use cloud\_resources ex: use <exp>

3: show collections create the collection

3.1

4:To see the data inside your collection: **db.servers.find().pretty()**

5 db.resources.deleteOne({ cloud\_resource\_id: "67c06ceba8f15650b851e955" })

5.db.servers.deleteOne({ \_id: ObjectId("67c06ceba8f15650b851e955") }) base on data stucher it will change

**docker exec mongodb mongodump --username root --password rootpassword --out /data/dev**

**docker cp mongodb:/data/mongodb ./dev**

docker cp ~/dev mongodb:/data/

docker exec mongodb mongorestore --drop /data/dev/mongodb

**Note : it is the dump the out side of continer in spec dir in this method all DB not dump**

docker exec mongodb mongodump --host=localhost --port=27017 --username=root --password=rootpassword \ --authenticationDatabase=admin --out=/data/mongodump/

docker exec mongodb mongorestore --username root --password rootpassword --authenticationDatabase admin --drop /data/devops/mongodb

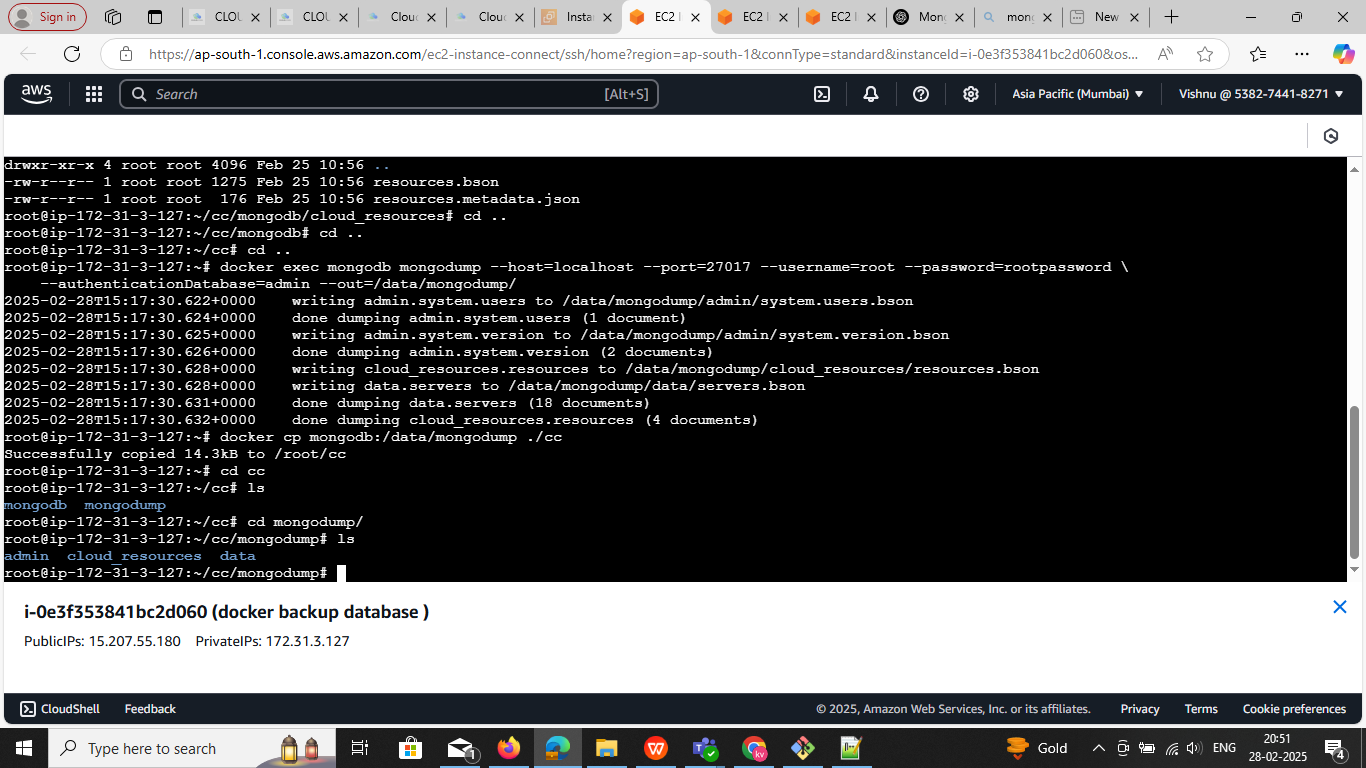
docker exec mongodb mongodump --host=localhost --port=27017 --username=root --password=rootpassword \ --authenticationDatabase=admin --out=/data/mongodump/

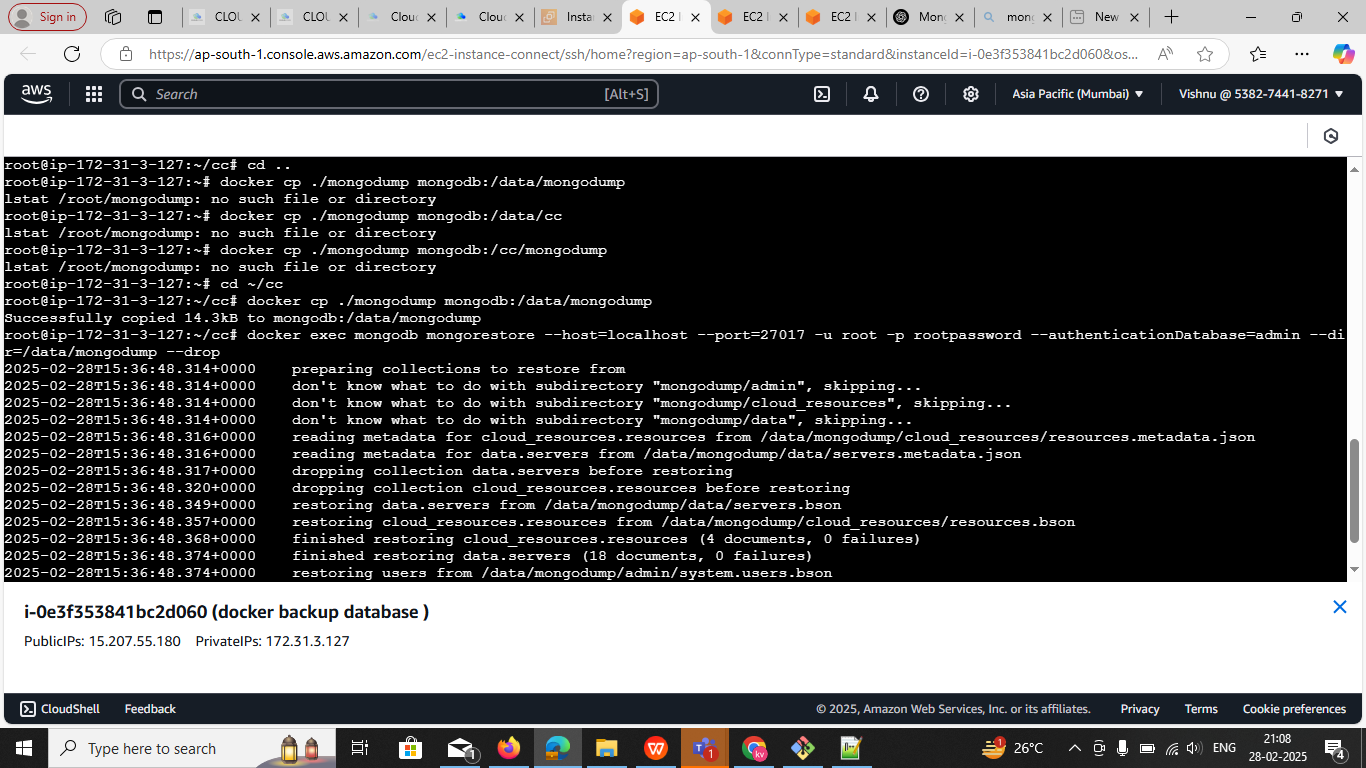
**Mongodump command** :

* docker exec mongodb mongodump --host=localhost --port=27017 -u root -p rootpassword --authenticationDatabase=admin --out=/data/mongodump/

Runs mongodump inside the mongodb container

* Authenticates with root/rootpassword
* Dumps all databases into /data/mongodump/ inside the containe
* docker cp mongodb:/data/mongodump ./**cc**



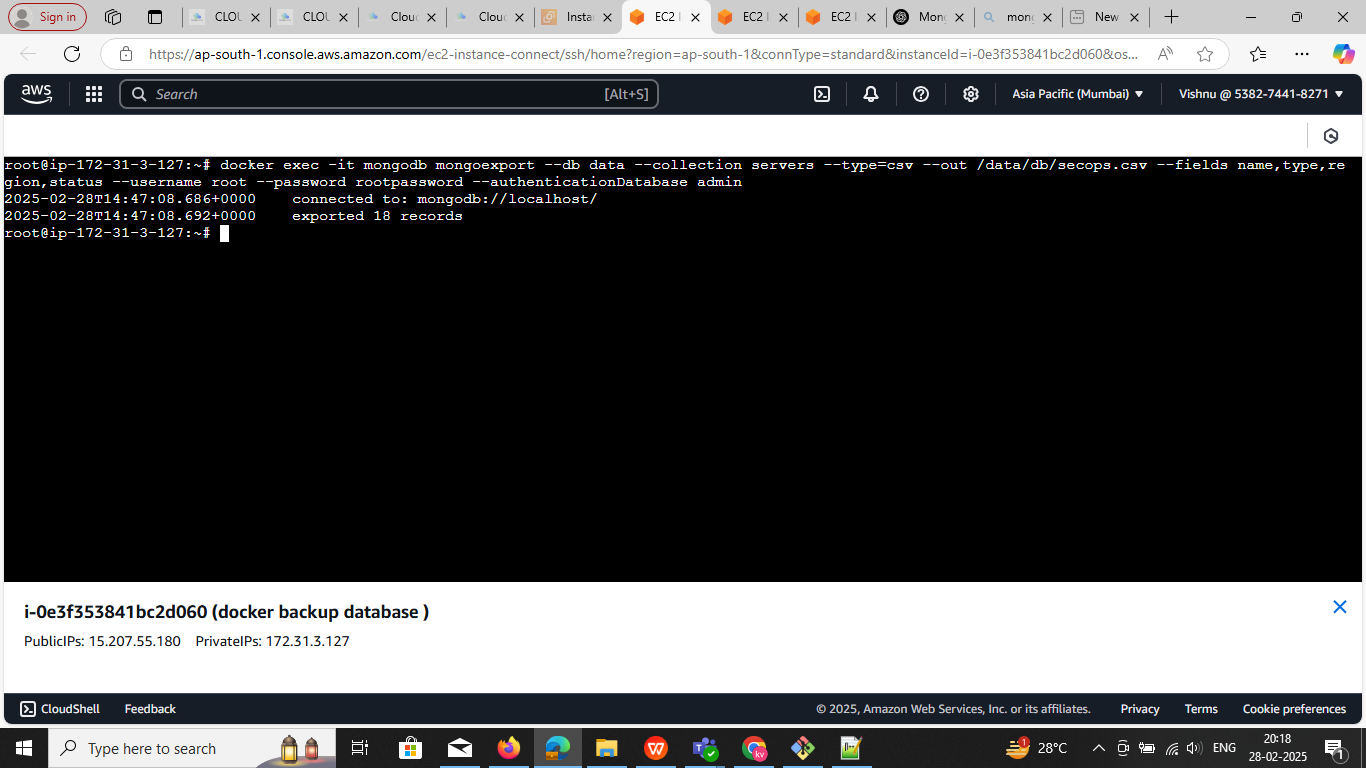


**Note : change the csv format for mongodb data base**

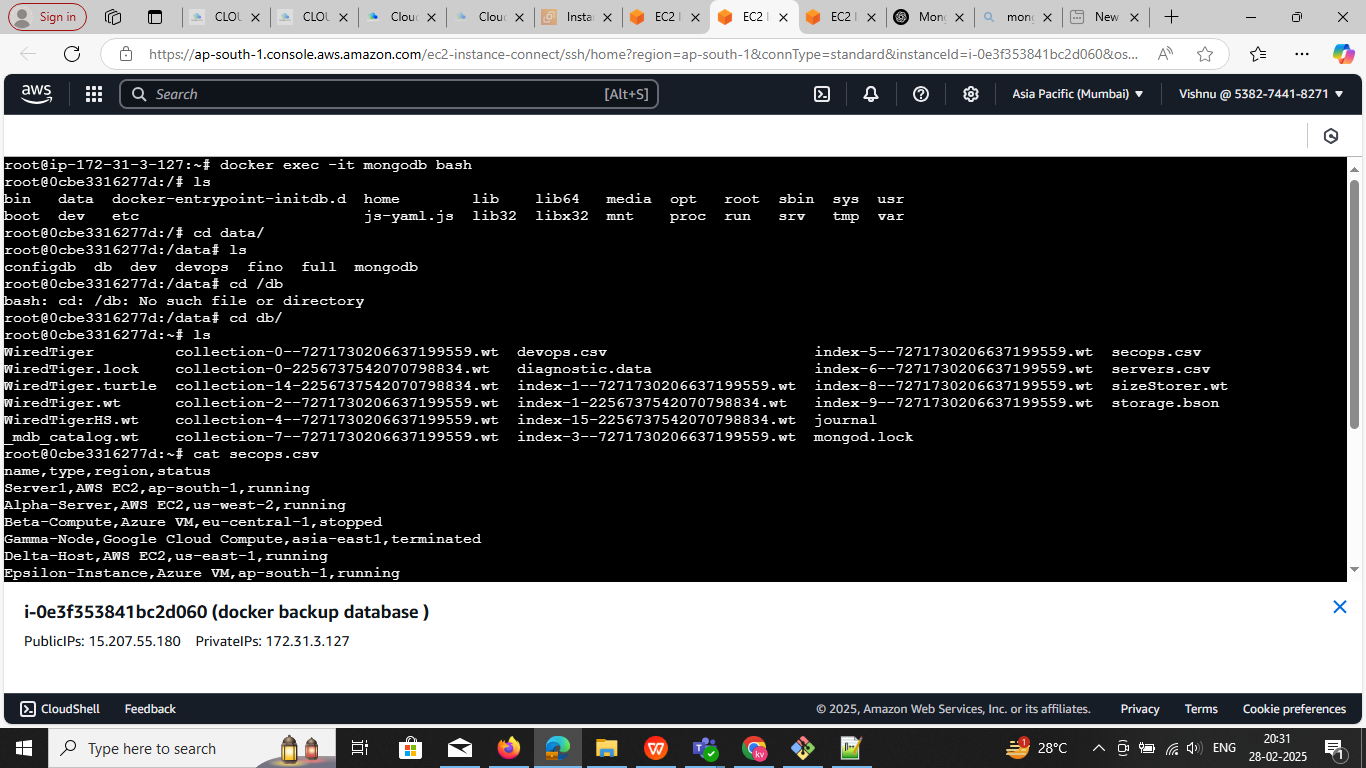
**Command : that change the all .csv format**

**docker exec -it mongodb mongoexport --db data --collection servers --type=csv --out /data/db/secops.csv --fields name,type,region,status --username root --password rootpassword --authenticationDatabase admin**

**docker exec -it mongodb mongoexport --db cloud\_resources --collection resources --type=csv --out /data/db/servers.csv --fields name,type,region,status --username root --password rootpassword --authenticationDatabase admin**



docker exec -it mongodb bash

root@0cbe3316277d:/data# cd db

#!/bin/bash

# Configuration

CONTAINER\_NAME="mongodb" # Replace with your actual container name

USERNAME="root"

PASSWORD="rootpassword"

# Get date and time dynamically

TODAY=$(date +"%d-%m-%Y") # Format: DD-MM-YYYY

TIMESTAMP=$(date +"%H%M%S") # Format: HHMMSS

# Backup directory

BACKUP\_DIR="/root/pod/$TODAY"

# Ensure backup directory exists

mkdir -p "$BACKUP\_DIR"

full\_backup() {

echo "Taking full backup..."

BACKUP\_PATH="$BACKUP\_DIR/full-$TIMESTAMP"

docker exec $CONTAINER\_NAME mongodump --username $USERNAME --password $PASSWORD --out /data/dev

docker cp $CONTAINER\_NAME:/data/dev $BACKUP\_PATH

echo "Full backup saved at: $BACKUP\_PATH"

}

incremental\_backup() {

echo "Taking incremental backup (oplog)..."

BACKUP\_PATH="$BACKUP\_DIR/oplog-$TIMESTAMP"

docker exec $CONTAINER\_NAME mongodump --username $USERNAME --password $PASSWORD -d local -c oplog.rs --out /data/oplog

docker cp $CONTAINER\_NAME:/data/oplog $BACKUP\_PATH

echo "Incremental backup (oplog) saved at: $BACKUP\_PATH"

}

restore\_backup() {

echo "Restoring full backup..."

read -p "Enter full backup path: " FULL\_PATH

read -p "Enter oplog backup path (optional, press Enter to skip): " OPLOG\_PATH

if [ ! -d "$FULL\_PATH" ]; then

echo "Error: Full backup path does not exist!"

exit 1

fi

docker cp "$FULL\_PATH" $CONTAINER\_NAME:/data/full

docker exec $CONTAINER\_NAME mongorestore --drop /data/full

if [ -n "$OPLOG\_PATH" ] && [ -d "$OPLOG\_PATH" ]; then

echo "Applying oplog (point-in-time recovery)..."

docker cp "$OPLOG\_PATH" $CONTAINER\_NAME:/data/oplog

docker exec $CONTAINER\_NAME mongorestore --oplogReplay /data/oplog

fi

echo "Restore completed!"

}

echo "MongoDB Backup & Restore Script"

echo "1. Full Backup"

echo "2. Incremental (Oplog) Backup"

echo "3. Restore Backup"

read -p "Enter your choice (1/2/3): " choice

case $choice in

1) full\_backup ;;

2) incremental\_backup ;;

3) restore\_backup ;;

\*) echo "Invalid choice! Exiting..." ;;

esac