Dr. Shubhangi Kharche

AWS Project 3

Scenario:

Testing/development of a cloud-native application on the cloud might not be feasible due to cost constraints. Therefore, when working on an application using DynamoDB as the back-end, we have to deploy it on-prem first for debugging purposes.

This project deploys DynamoDB using an IaaS deployment on EC2 instances. This easily be adapted to an on-premise installation for testing and debugging purposes.

Objectives	
1	Install DynamoDB on an EC2 instance as an IaaS installation
2	Connect to the IaaS DynamoDB installation using the given Python Script to perform CRUD operations
3	Run a Python Program to create a DynamoDB table in your AWS account with secondary indexes.

Step 1: EC2 instance configuration

Step number	a
Step name	DynamoDB IaaS Setup
Instructions	1)Create an Amazon Linux 2 EC2 instance with ports open for port numbers 22 and 8000 2) Attach the built-in IAM role LabInstanceProfile to the instance 3) SSH into the instance using your preferred SSH client 4) Run the below commands into the terminal once the SSH connection is successful ● *sudo yum update* ● *sudo yum install python-pip -y* ● *sudo yum install awscli -y* ● *pip install boto3* ● *sudo yum install java-1.8.0-openjdk* ● *mkdir dynamodb-local* ● *cd dynamodb-local* ● *wget http://dynamodb-local.s3-website-us-west-2.amazonaws.com/dynamodb_local_latest.tar.gz* ● *tar xzf dynamodb_local_latest.tar.gz* ● *java -Djava.library.path=./DynamoDBLocal_lib/ -jar DynamoDBLocal.jar &* 7) Keep this terminal window open for the rest of the exercise. If the program crashes or you get logged out of the instance, restart the program. Use a new terminal window for the rest of this exercise.
Expected screenshots	1) Created EC2 instance 2) Successful execution of the java command

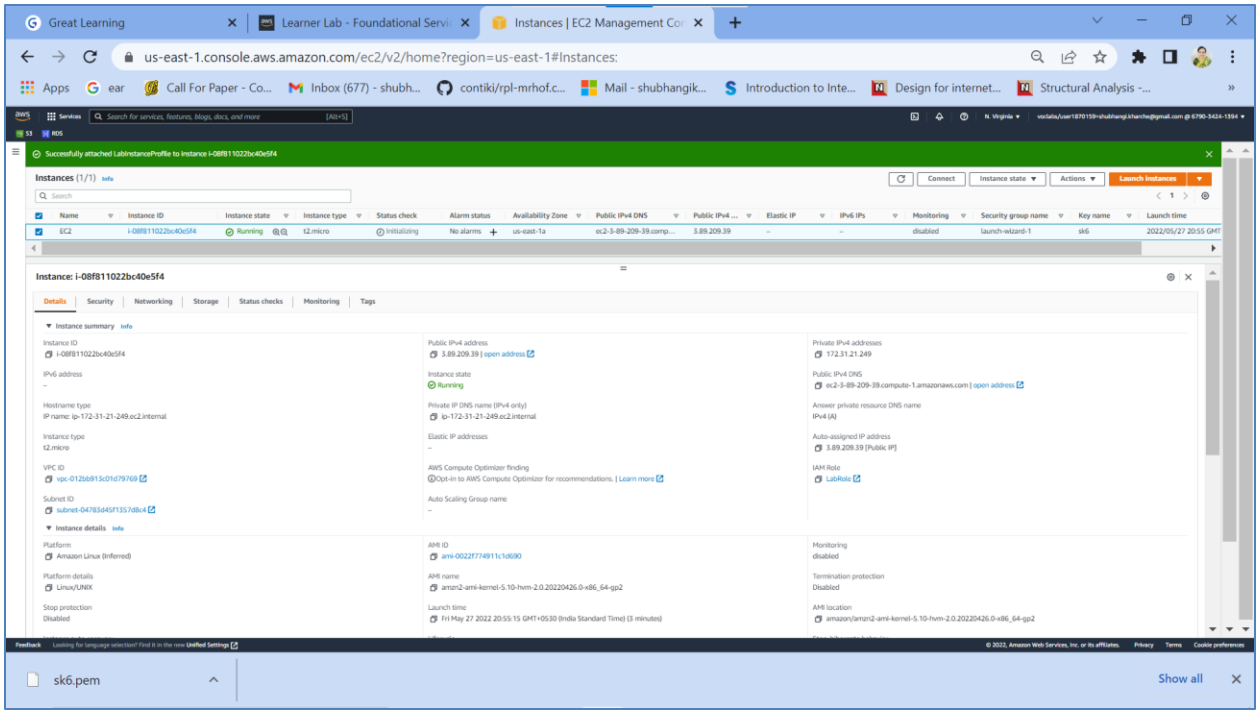

Fig.1 : Created EC2 instance

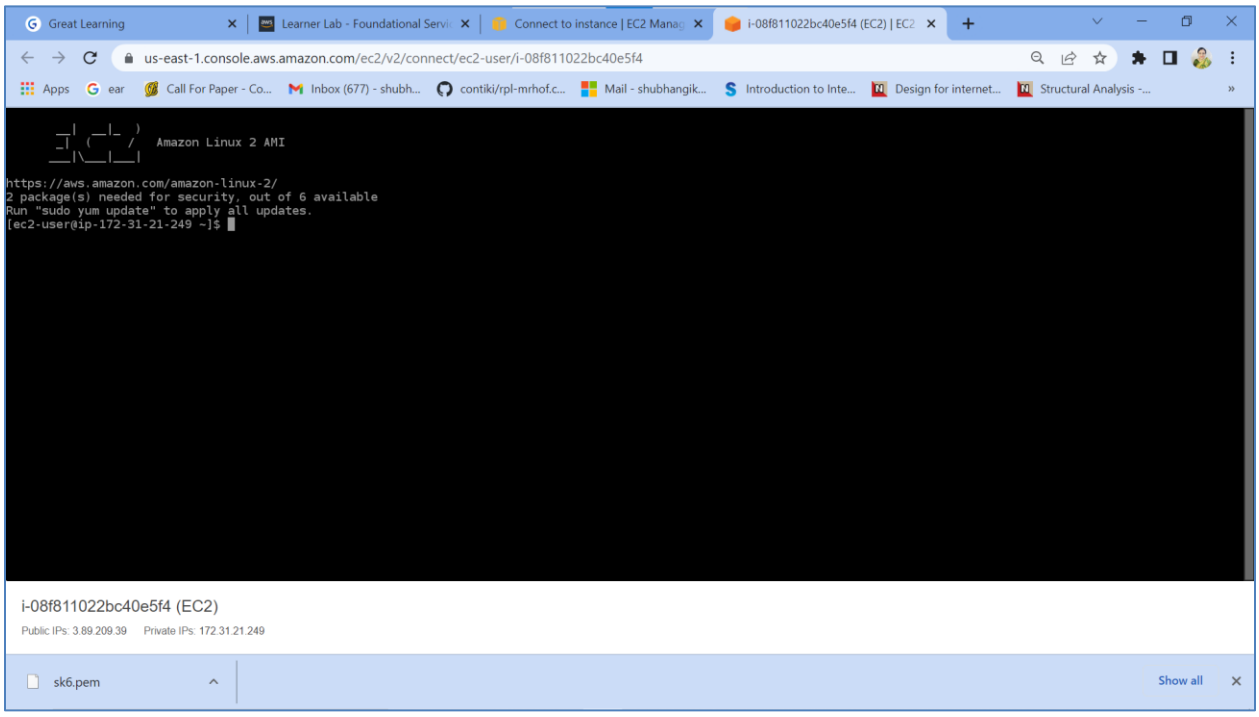

Fig.1 a: Successful login to EC2 instance (optional screenshot)

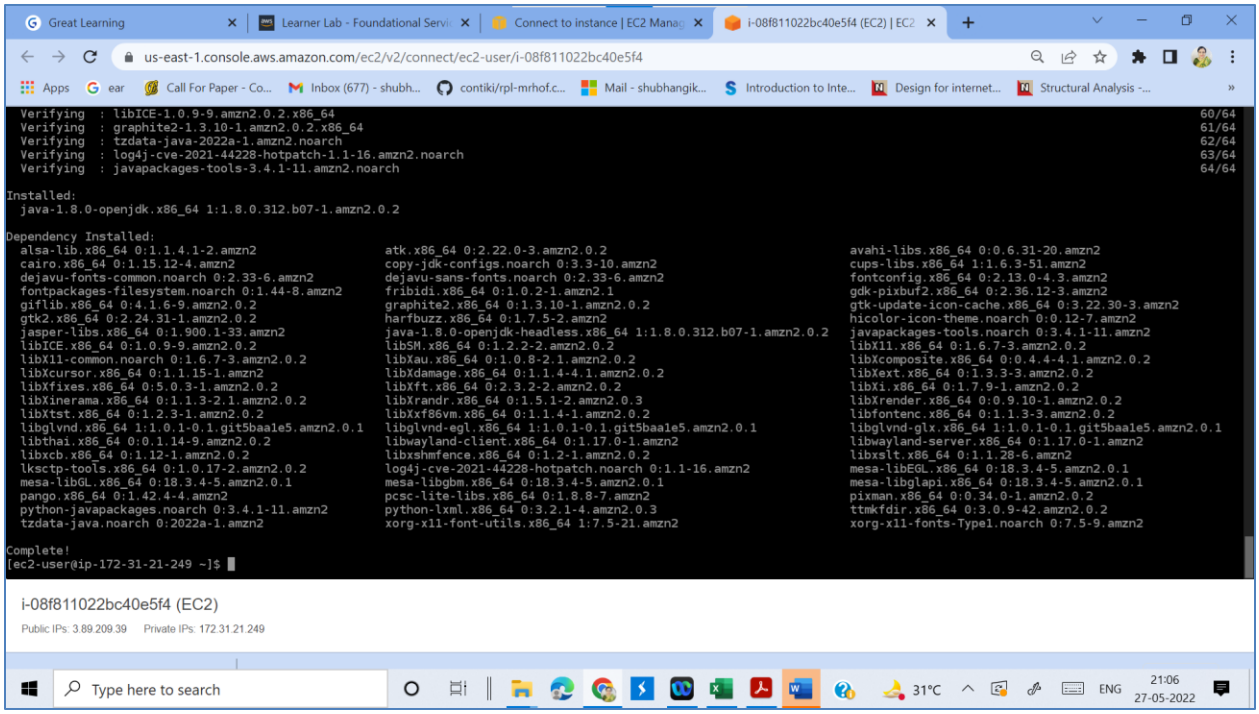

Fig.2 a : Successful installation of Java (optional screenshot)

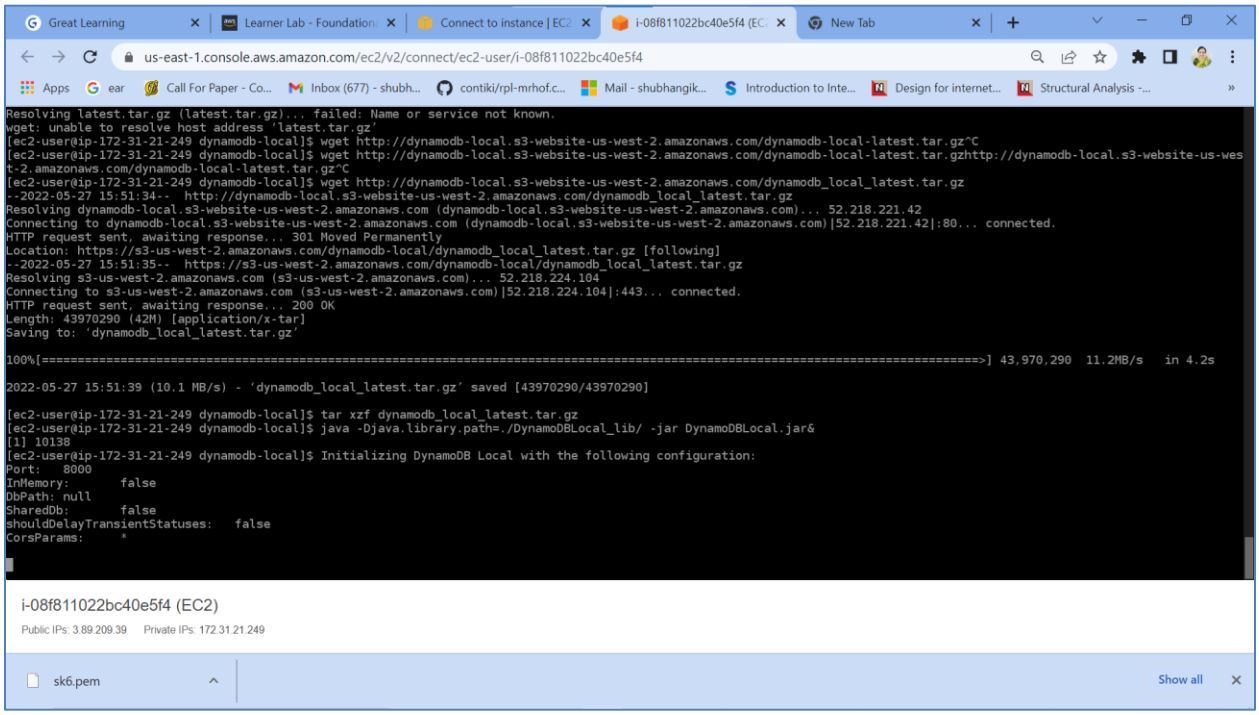

Fig.2 b : Successful execution of the java command (after getting dynamodb_local_latest.tar.gz file and un-taring it)

Step 2: CRUD operations on local DynamoDB deployment

Step number	b
Step name	Deployment of Python Script
Instructions	1) Download the script dynamo-ops2.py 2) Use your preferred SCP client to copy it to the folder /home/ec2-user in the instance created above 3) SSH into the instance again 4) Run the following command in the terminal to execute the python script *python dynamo-ops2.py* 5) Run the below commands to verify that the table has been created ● *aws configure* Skip the access key and secret access key fields. Enter the region as **us-east-1** and format as **json** ● *aws dynamodb list-tables --endpoint-url http://localhost:8000*
Expected screenshots	1) Successful execution of the Python command 2) Execution of aws configure 3) Verification of table creation

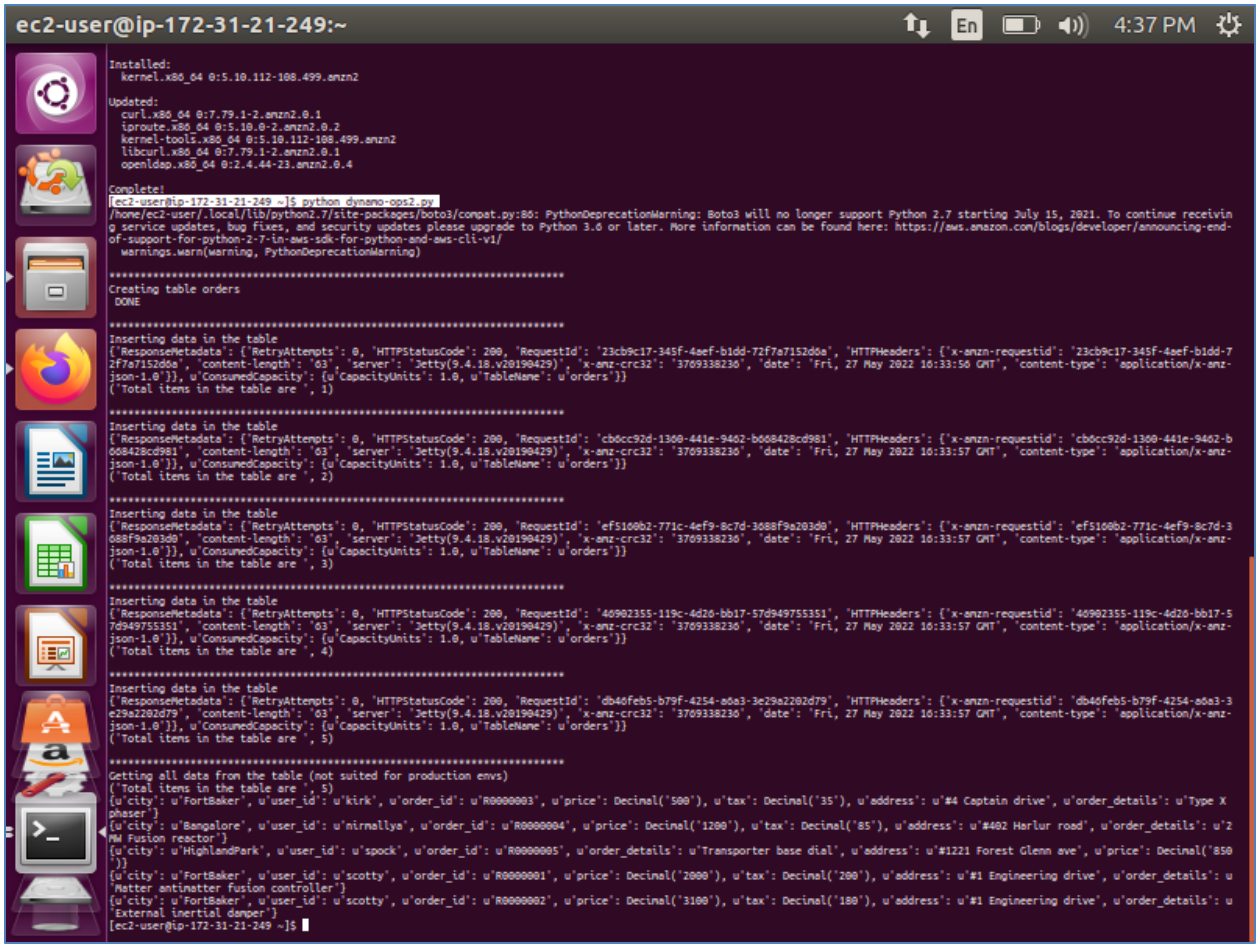

Fig.3 : Successful execution of the python command

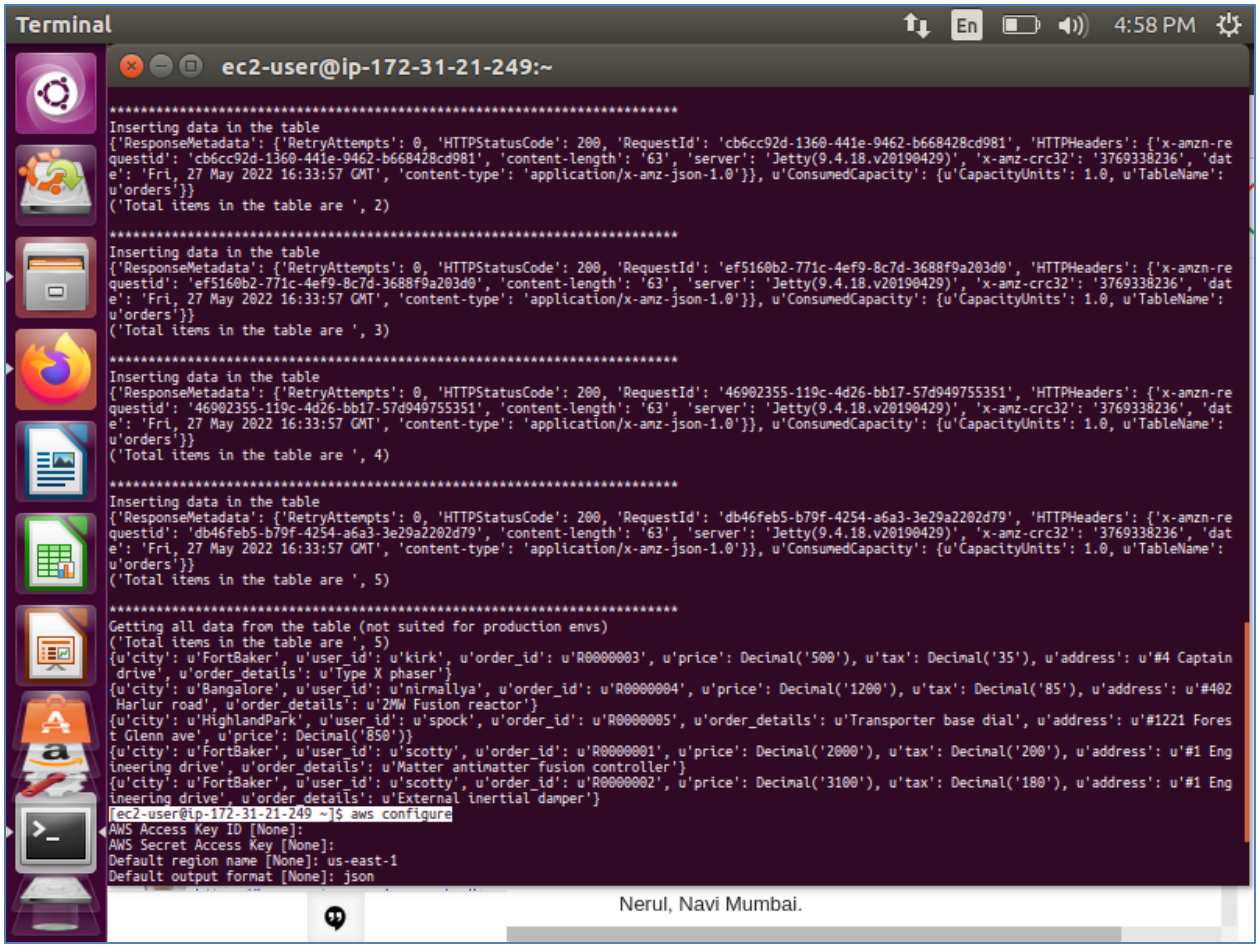

Fig.4 : Execution of aws configure

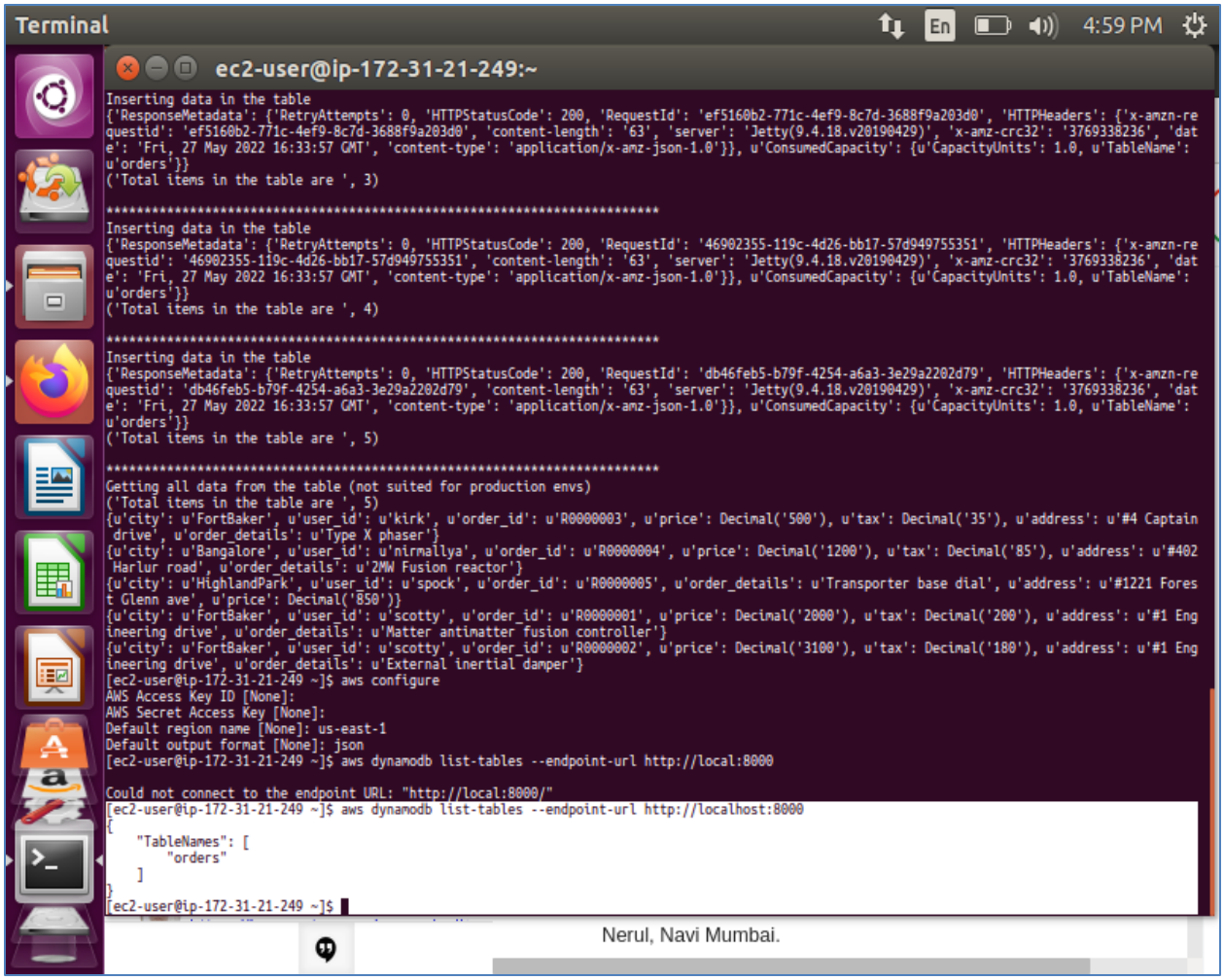

Fig.5: Verification of table creation

Step 3 : Creation of DynamoDB table with secondary index.

Step number	c
Step name	Deployment of Python Script
Instructions	1) Create another EC2 instance and configure it as shown in Step 1. Also, configure the AWS CLI as shown in Step 2. 2)Download the provided script dynamo-advanced.py. 2) Use your preferred SCP client to copy the given script dynamo-advanced.py to the EC2 instance created above into the location /home/ec2-user 2) SSH into the EC2 instance again 3) Run the provided Python program using the command below python dynamo-advanced.py 4) Verify the table creation is successful using the following command *aws dynamodb list-tables --endpoint-url http://localhost:8000* 5) Show the properties of the table using the command below *aws dynamodb describe-table --table-name orders --endpoint-url http://localhost:8000*
Expected screenshots	1) Successful execution of the Python program 2) Verification of table creation 3) Verification of table properties

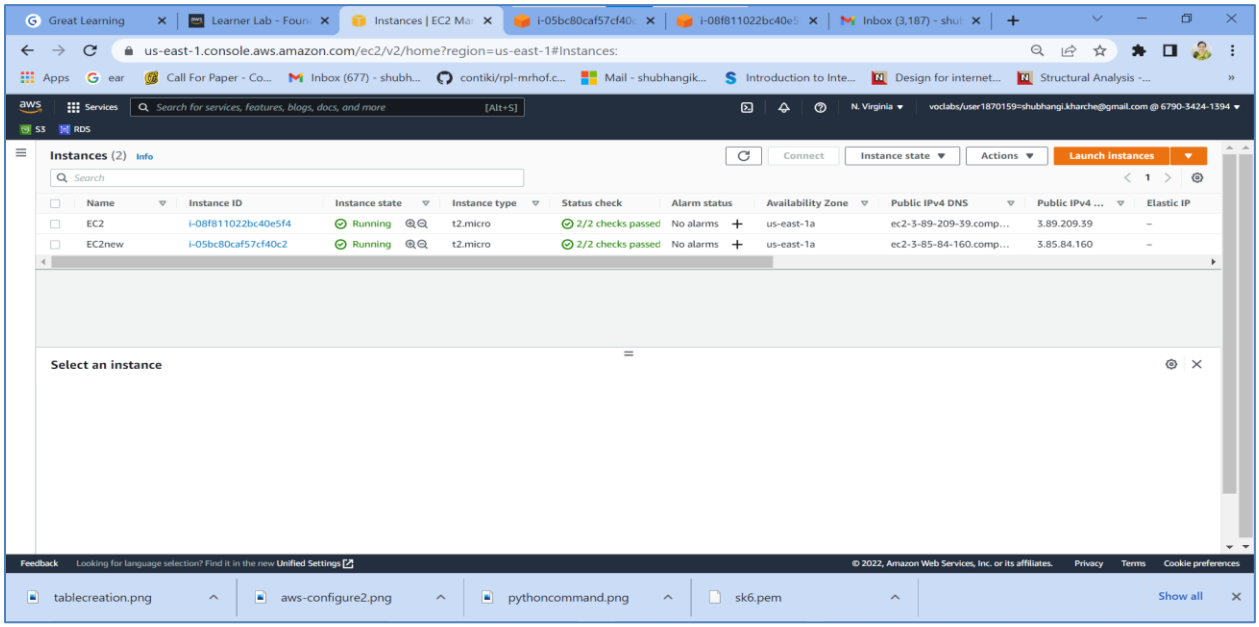

Fig.6 a: Successful creation of another EC2 instance (optional screenshot)

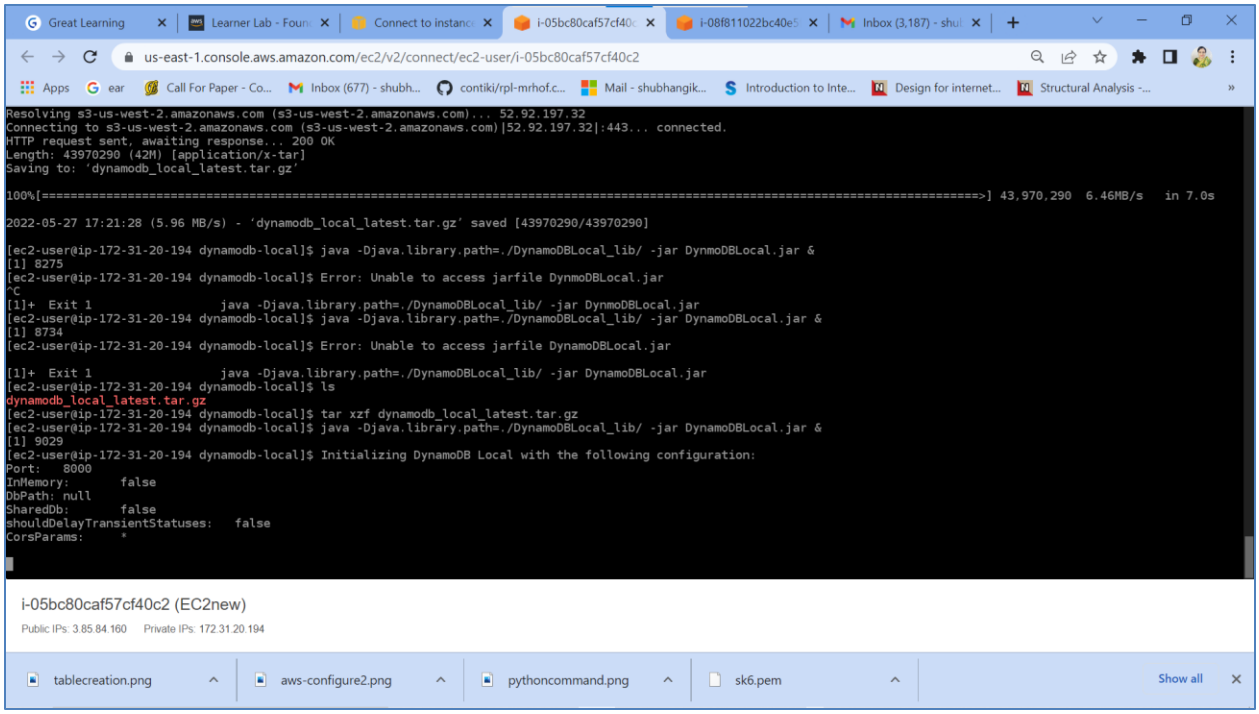

Fig.6 b: Successful execution of the Java command on another EC2 instance (after getting dynamodb_local_latest.tar.gz file and un-taring it) (optional screenshot)

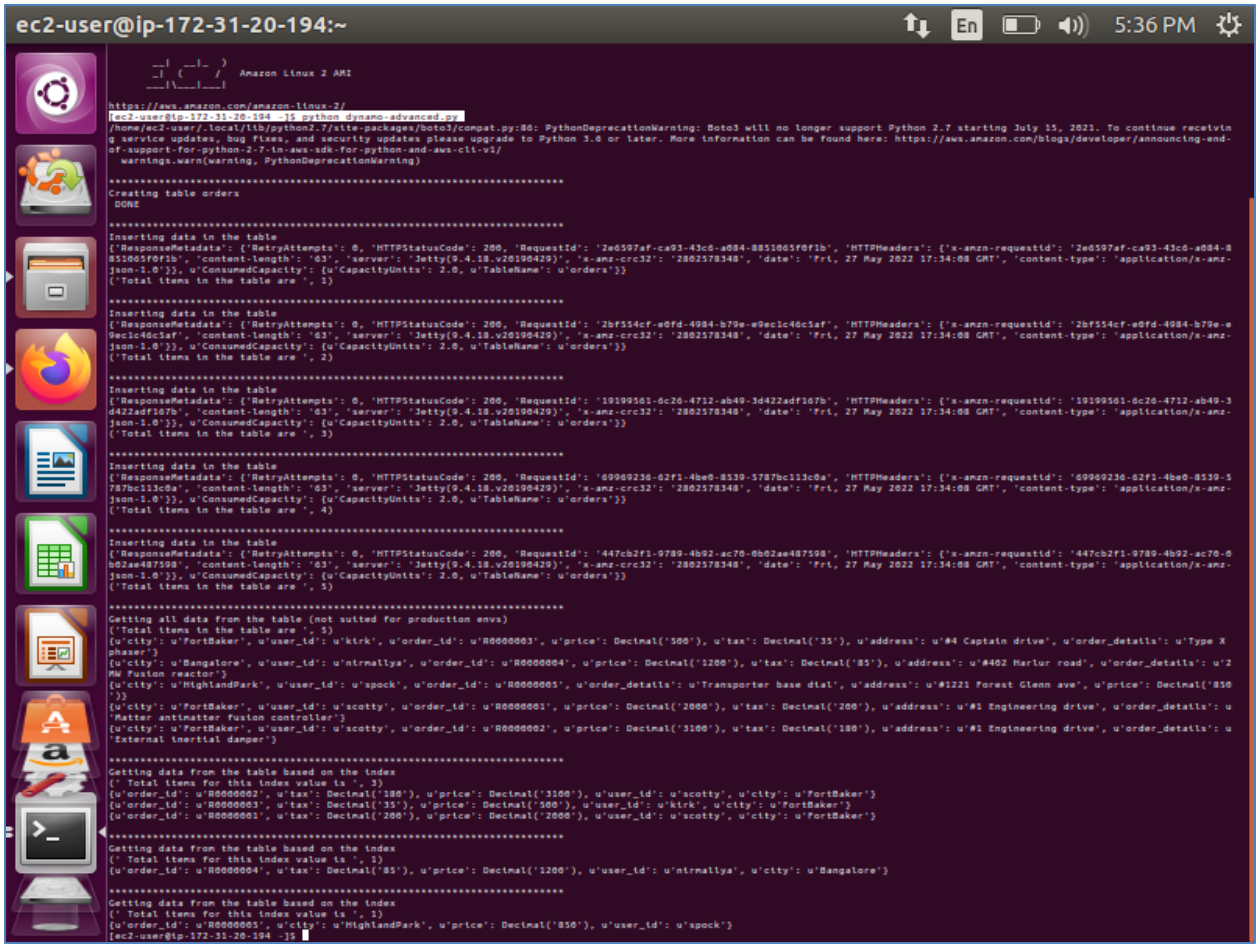

Fig.6 c: Successful execution of the python program

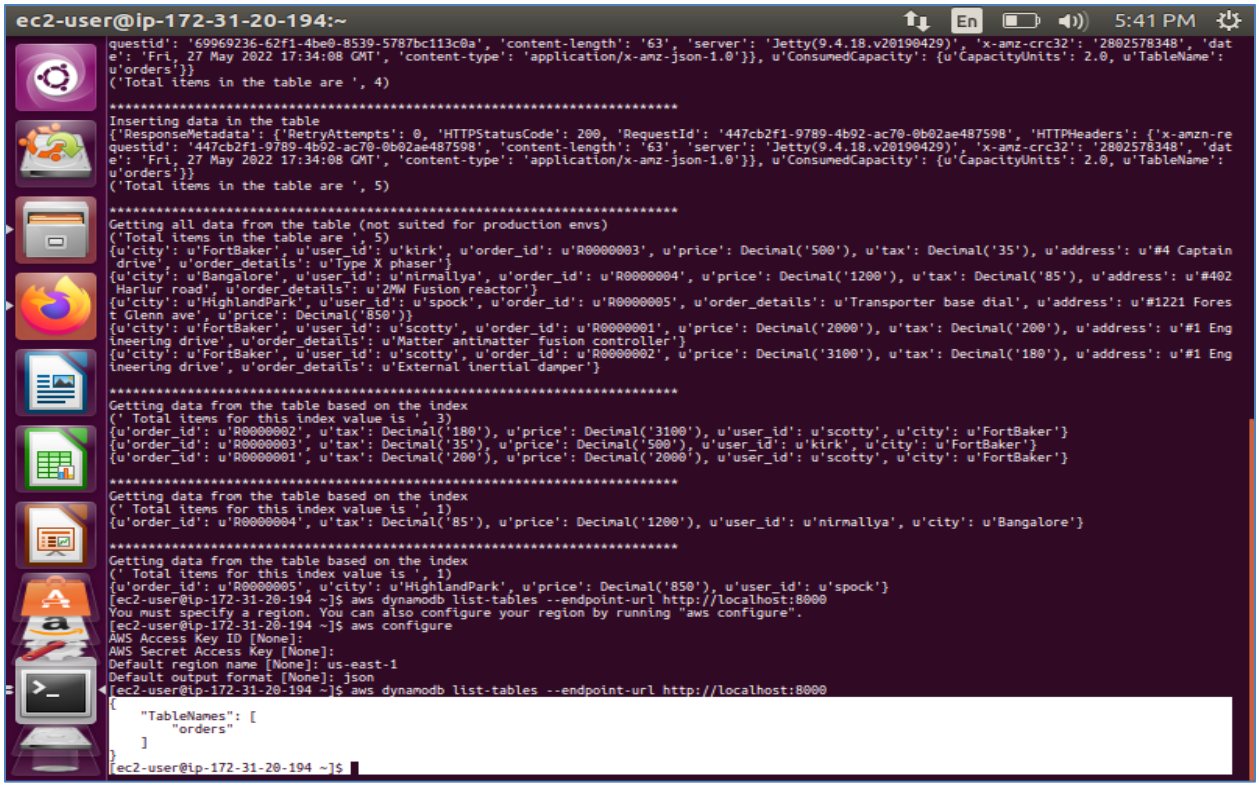

Fig.7: Verification of table creation

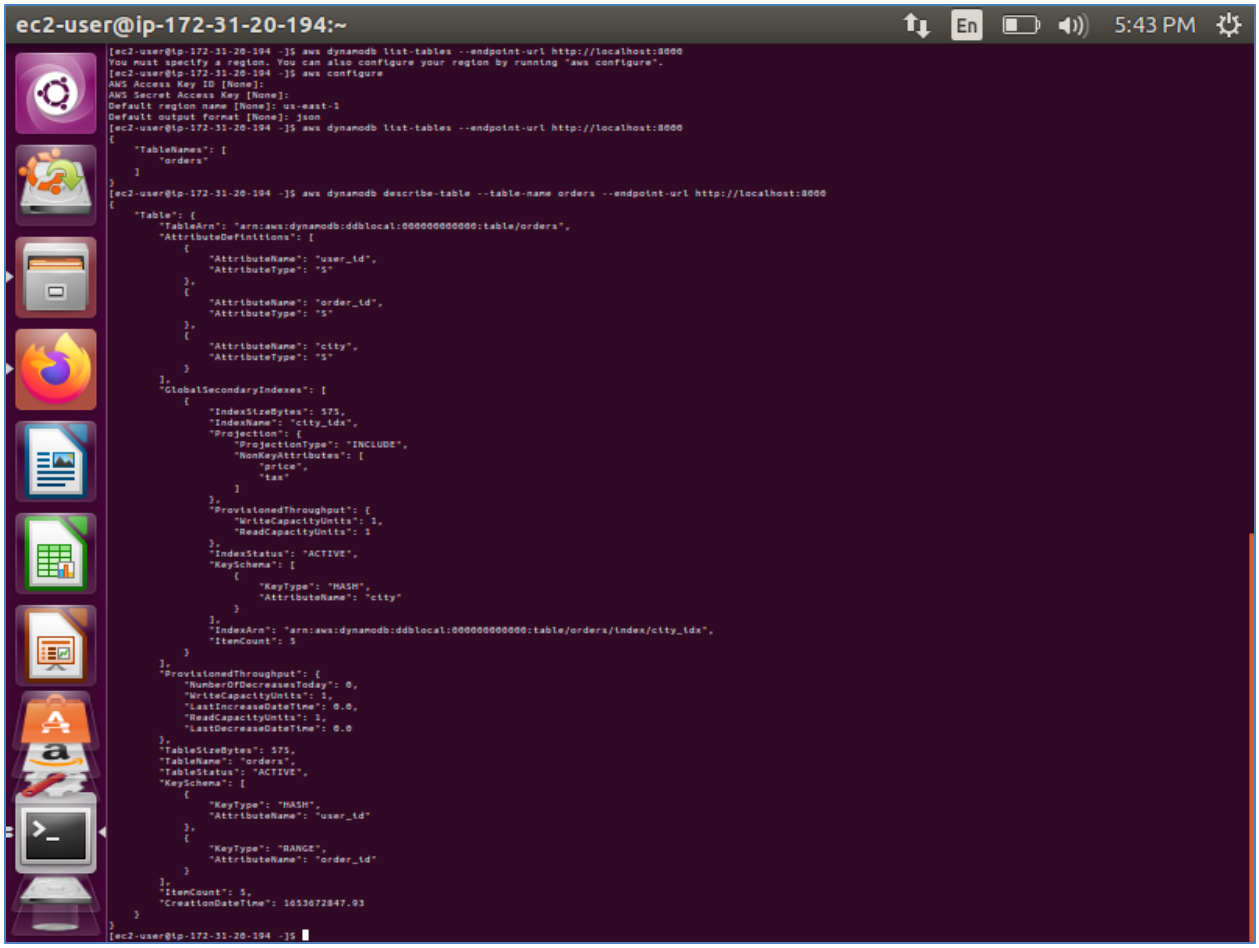

Fig.8: Verification of table properties

Grades distribution	
MCQs	5 (1 mark each)
Subjective questions	11 marks (3+8)
Implementation screenshots	24 marks (3marks each)
Total	40 marks

MCQ

1. What will be the expected size of a row in DynamoDB table which has the primary key with only the partition key and no sort keys?
    a. Question is incomplete, velocity specification is required
    b. This is the case of a wide row issue
    c. One record
    d. Sort keys are mandatory so table creation will fail

Ans: c. This is the case of a wide row issue

2. Transactions are always a good idea in DynamoDB. It should be used very liberally to ensure strong data integrity.
    a. True
    b. False

Ans: a. True

3. Identify the correct statement (LSI = Local Secondary Index, GSI, Global secondary index, RK = Partition key, SK = Sort key). Select 2 options.
    a. LSI needs same RK & different SK from the table's primary key→ Answer
    b. LSI needs different RK & same SK from the table's primary key
    c. GSI needs same RK & different SK from the table's primary key
    d. GSI needs different RK & different SK from the table's primary key→ Answer

Ans: a. and d.

4. Which statement is true about the number of secondary indices a table can have (can be global or local)?
    a. Number is LSI is limited but GSI is unlimited→ Answer
    b. Both LSI and GSI have limits
    c. LSI is unlimited but GSI is limited
    d. Both do not have any limits

Ans: a.

5. A good data model in a NoSQL such as DynamoDB must factor in the access patterns upfront.
    a. True→ Answer
    b. False

Ans: a. True

Subjective questions

1. Analyze the following text and answer the questions

   “Many database workloads are cyclical in nature or are difficult to predict in advance. For example, consider a social networking app where most of the users are active during daytime hours. The database must be able to handle the daytime activity, but there's no need for the same levels of throughput at night. Another example might be a new mobile gaming app that is experiencing rapid adoption. If the game becomes too popular, it could exceed the available database resources, resulting in slow performance and unhappy customers. These kinds of workloads often require manual intervention to scale database resources up or down in response to varying usage levels.

   DynamoDB auto scaling uses the AWS Application Auto Scaling service to dynamically adjust provisioned throughput capacity on your behalf, in response to actual traffic patterns. This enables a table or a global secondary index to increase its provisioned read and write capacity to handle sudden increases in traffic, without throttling. When the workload decreases, Application Auto Scaling decreases the throughput so that you don't pay for unused provisioned capacity.

   To understand how DynamoDB auto scaling works, suppose that you have a table named `ProductCatalog`. The table is bulk-loaded with data infrequently, so it doesn't incur very much write activity. However, it does experience a high degree of read activity, which varies over time. By monitoring the CloudWatch metrics for `ProductCatalog`, you determine that the table requires 1,200 read capacity units (to avoid DynamoDB throttling read requests when activity is at its peak). You also determine that `ProductCatalog` requires 150 read capacity units at a minimum, when read traffic is at its lowest point. Within the range of 150 to 1,200 read capacity units, you decide that a target utilization of 70 percent would be appropriate for the `ProductCatalog` table. *Target utilization* is the ratio of consumed capacity units to provisioned capacity units, expressed as a percentage. Application Auto Scaling uses its target tracking algorithm to ensure that the provisioned read capacity of `ProductCatalog` is adjusted as required so that utilization remains at or near 70 percent.”

   Question : Can you describe a scenario where we will not require autoscaling of a table as described above? (3 marks)

   Answer: IoT based weather monitoring scenario where Writes required are constant as per specific interval defined & reads can be defined basis where the collected data is used.

Kindly refer Fig. 9 below for details

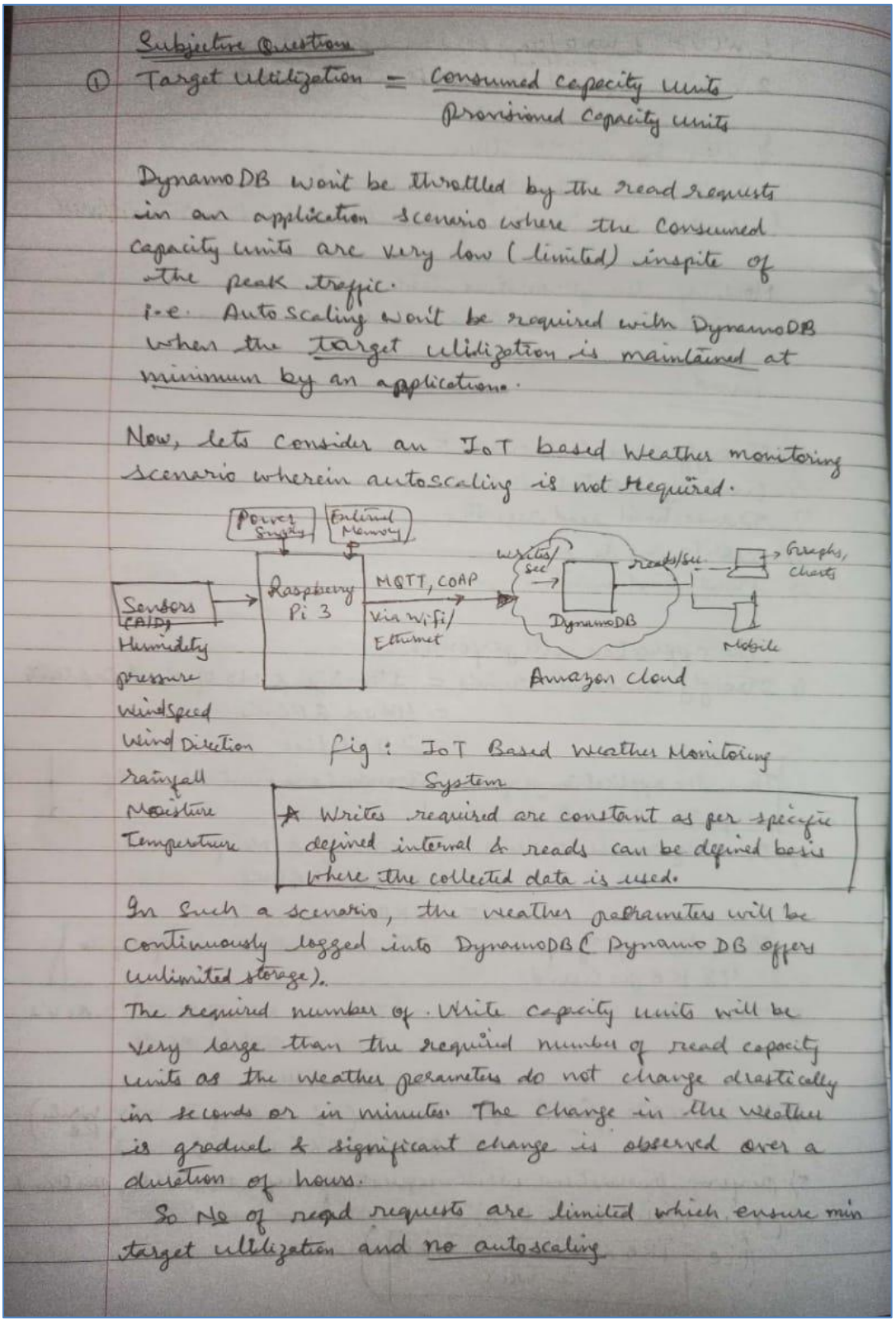

Fig. 9: IoT based Weather monitoring scenario wherein DynamoDB can be used without autoscaling

2. Please read the following text and answer the question that follows it.
“If you choose provisioned mode, you specify the number of reads and writes per second that you require for your application.

One read capacity unit represents one strongly consistent read per second, or two eventually consistent reads per second, for an item up to 4 KB in size. Transactional read requests require two read capacity units to perform one read per second for items up to 4 KB.

One write capacity unit represents one write per second for an item up to 1 KB in size. If you need to write an item that is larger than 1 KB, DynamoDB must consume additional write capacity units. Transactional write requests require 2 write capacity units to perform one write per second for items up to 1 KB.”

Question: Suppose that you create a provisioned table with 6 read capacity units and 6 write capacity units. With these settings, your application could do the following (8 marks):

1. Perform strongly consistent reads of up to **24** KB per second **(4KB x 6 RCUs)**
2. Perform eventually consistent reads of up to **48** KB per second **(2 x 4KB x 6 RCUs)**
3. Write up to **6** KB per second **(1KB x 6 WCUs)**
4. Perform transactional write requests of up to **3** KB per second **(1KB x 6 WCUs / 2 WCUs)**

Please refer images below for detailed solution

Solution:

Given that:

The provisioned table has

Read capacity units = 6

Write capacity units = 6

Item Size = 4 KB

1 RCU = 1 Strongly consistent read/sec

1 RCU = 2 eventually consistent reads/sec

2 RCU = 1 transactional read request for 1 read/s

} for 4KB item

If Item Size = 8 KB Then

1 Strongly consistent read/sec = $\frac{8KB}{4KB}$ = 2 RCU's

1 Eventually consistent read/sec = $\frac{8KB}{2 \times 4KB}$ = 1 RCU

1 transactional read request = $\frac{8KB}{4KB} \times 2$ = 4 RCU's

1 WCU = 1 Write/sec ; for item upto 1 KB
2 WCU's = 1 transactional write request ; for item upto 1 KB

If item size is 2 KB then
2 WCU's are required for 1 Write/sec
& 4 WCU's are required for 1 transactional write request.

Now, if the provisioned table has;
6 RCU's & 6 WCU's then

Find:

1) Strongly consistent reads = _____
2) Eventually consistent reads = _____
3) Transactional read requests = _____
4) Writes up to = _____
5) Transactional write requests = _____

The application will perform:

1) Strongly consistent reads = Item Size x No of Read Cap. Units
= 4KB x 6 RCU's
= 24 KB/Sec.

Thus, the application performs strongly consistent reads of up to 24 KB per Second.

2) Eventually consistent reads = 2 x Item Size x No of RCU's.
= 2 x 4KB x 6 RCU's
= 48 KB/Sec.

Thus, the appl^{n} performs eventually consistent reads of up to 48 KB per second.

3) Transactional read requests = (Item Size x No of RCU's)/2 RCU's
= 4KB x 6 RCU's/2 RCU's
= 12 KB/Sec

4) Write up to 6 KB per second (∵ Each of the WCU writes 1 KB)
(i.e. 1KB x 6 WCU's)

5) Perform transactional write requests of up to 3KB per second.
(∵ 2 WCU's write 1 KB)
(i.e. $\frac{1KB \times 6 WCU's}{2 WCU's} = 3KB$)