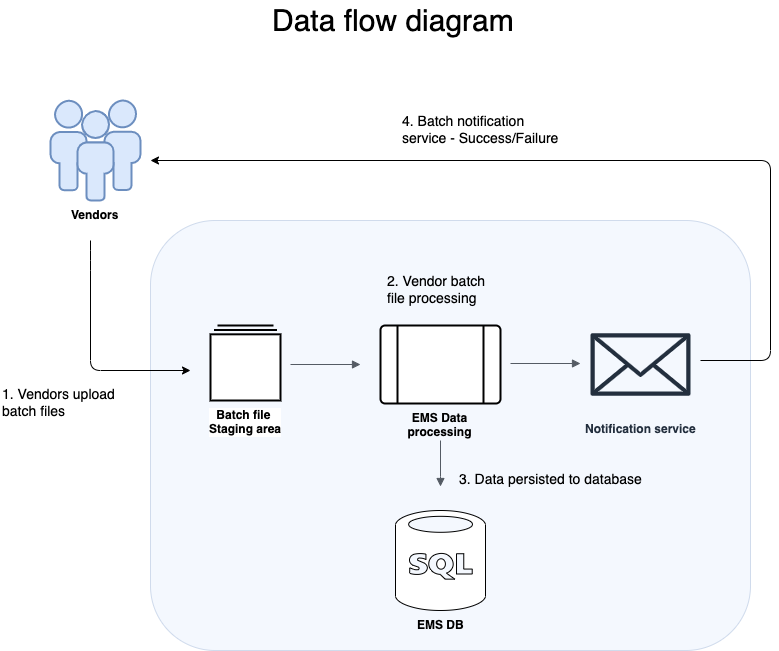
# Lab 1: Working with Amazon Aurora databases

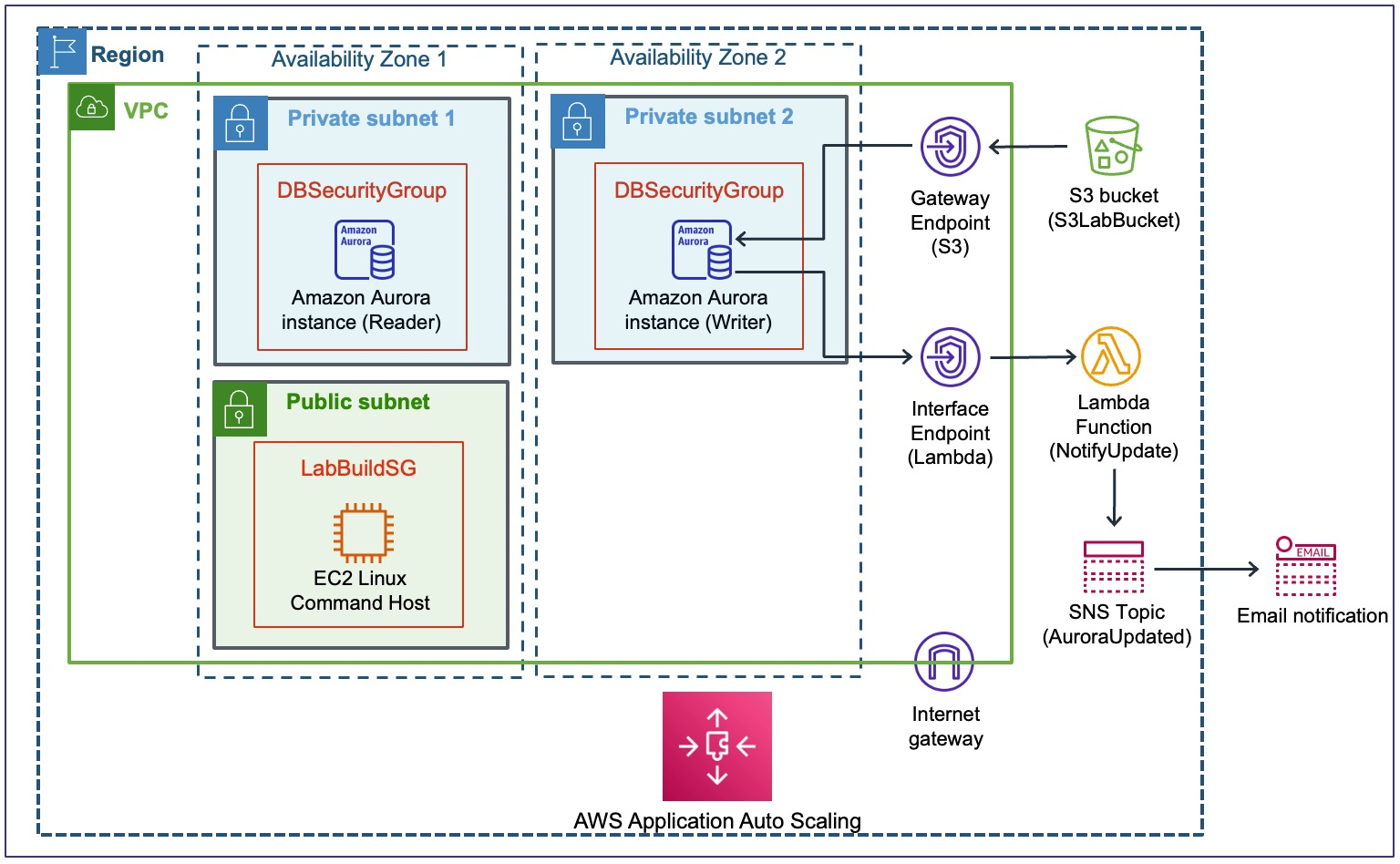
Objectives:

1. Design an Aurora cluster to meet application requirements.
2. Architect a solution on Aurora for high availability and scalability.
3. Integrate Aurora with other AWS services - Amazon S3.



DataFlow: (as example of event mgmt company, which has event vendors:

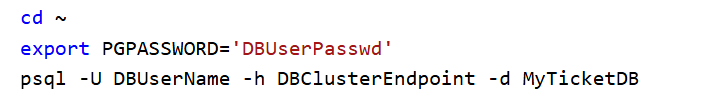
1. Vendor uploads event data files to Amazon S3.
2. The system automatically picks up these files and loads the data into the Aurora database.
3. Vendors are notified about the success or failure of the data loading process.

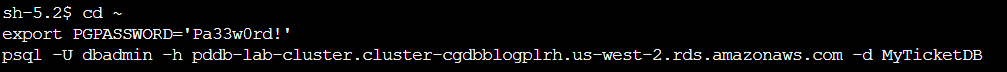


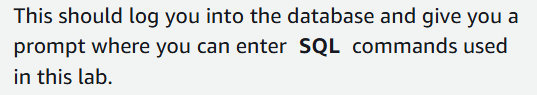
**Connecting to the Database via CLI**

Steps -

1. Open CLI using given URL to connect to PostgreSQL Database
2. Run this

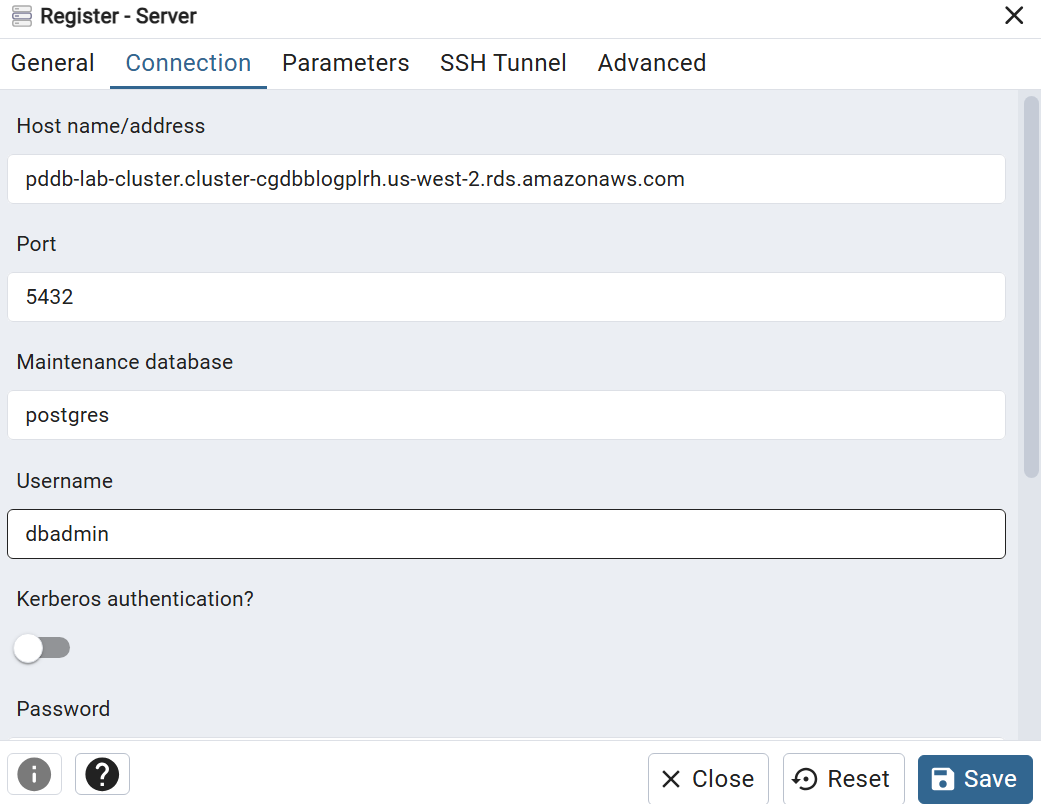
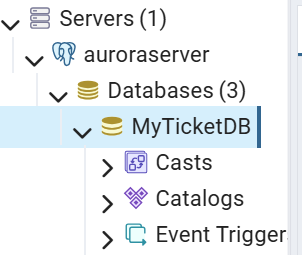






**Connecting to DB via UI**

Steps-

1. Use Url to get to login page, and login
2. 
3. Create a new server as follows
4. 
5. This referenced this with the code we ran in CLI in last task.
6. Open this DB
7. 
8. This is where we’ll run our SQL commands

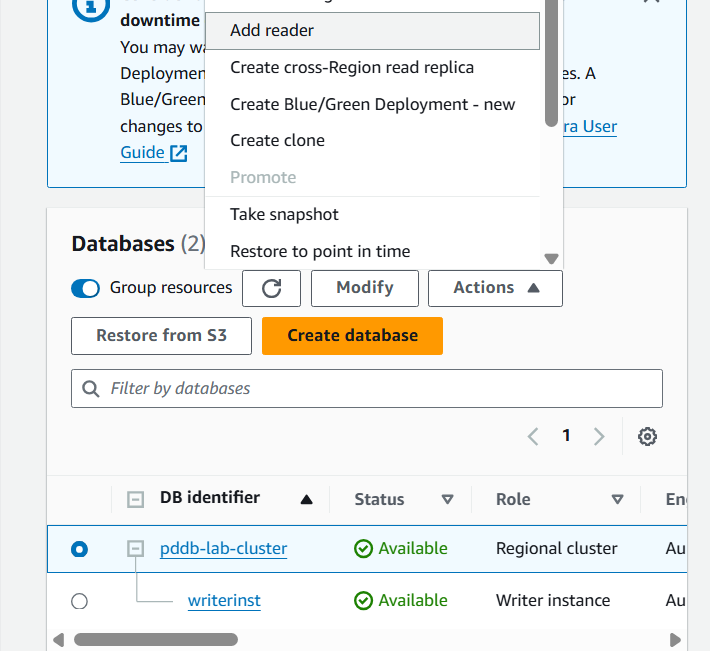
**Task 1: Configure Amazon Aurora for high availability**

**an aurora instance already has a writer instance, how does adding a reader instance add more availability?:**

1. **Load Balancing: With a separate reader instance, read queries can be load-balanced across multiple instances. This means that read-heavy workloads can be distributed among these instances, reducing the load on the writer instance.**
2. **Read Scalability: By allowing read queries to be directed to reader instances, you can scale out your read capacity independently from your write capacity. This is crucial in scenarios where the read workload is much higher than the write workload.**
3. **Fault Isolation: If the writer instance becomes unavailable due to maintenance, upgrades, or failures, the reader instances can still serve read queries. This enhances availability by isolating the impact of any issues on the write instance from the read operations.**

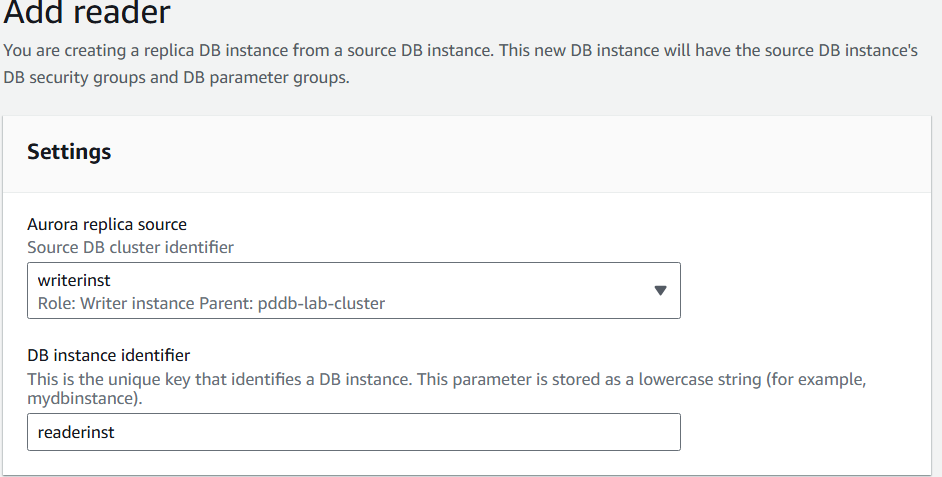
1.1 open console, go to RDS

1.2 Select the created DB

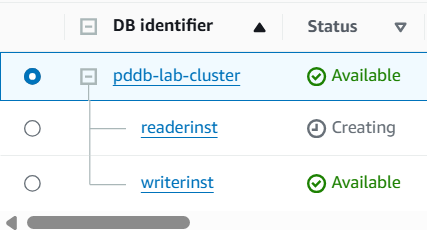


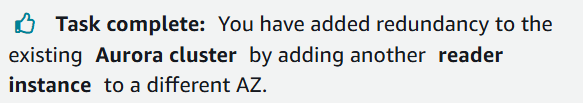
1.3 select add reader (a read-only instance of a database) option, note we already have a writer

1.4 enter the name for reader, and go ahead



1.5 creating



Goal - 

**Task 2: Configure DB to ingest a batch file from Amazon S3**

2.1 Connect to DB (done in pre-task steps), eihter way

2.2 Install S3 extension in sql

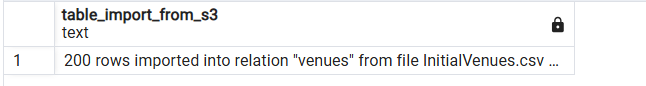
**CREATE EXTENSION IF NOT EXISTS aws\_s3 CASCADE;**

2.3 Run following command to ingest data from S3 to DB

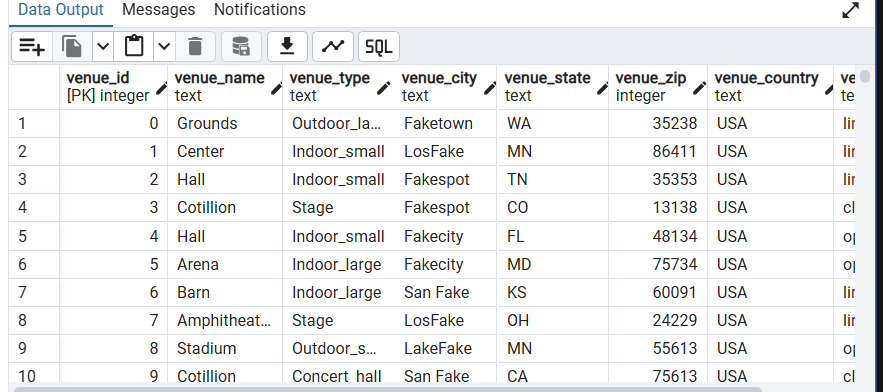


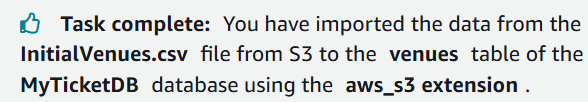
Labstack-a5… is the S3 bucket location

o/p-



2.4 view table





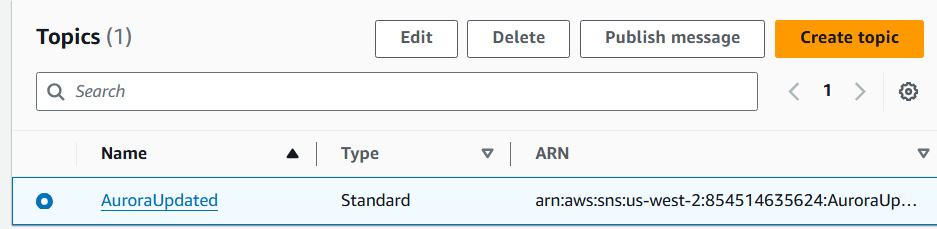
**Task 3: Configure Aurora to notify a vendor of the batch data processing status**

**Subscribe to an SNS topic and**

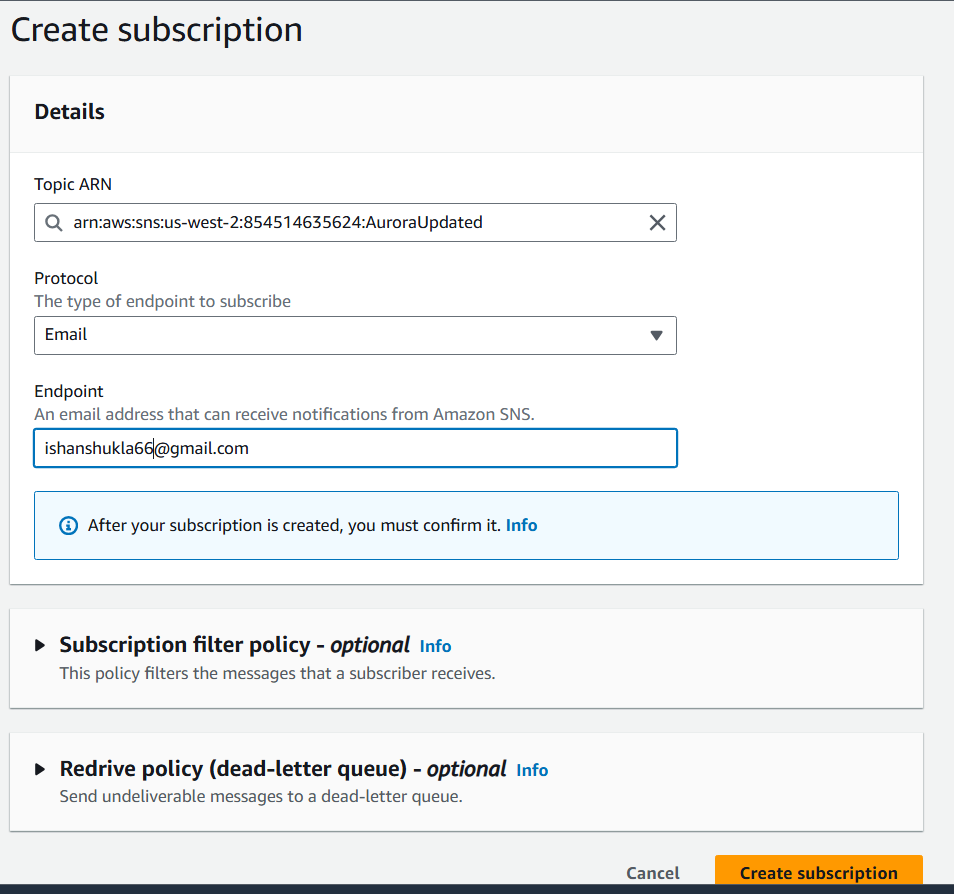
**Configure the database to invoke a Lambda function for notifications.**

TASK 3.1: **SUBSCRIBE TO AN SNS TOPIC**

3.11 open console and open SNS(Simple Notification Service)

3.12 open this topic

3.13 Create a subscription as follows



3.14 open email and click confirmation link

**TASK 3.2: CONFIGURE DATABASE WITH LAMBDA FUNCTION**

3.21 go back to sql

3.22 Install the needed Lambda CASCADE extension with the following SQL command.

CREATE EXTENSION IF NOT EXISTS aws\_lambda CASCADE;

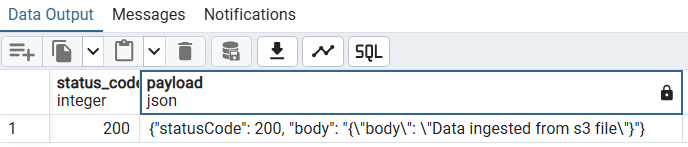
3.23 run the command (why? -it’s answer not given in lab)

SELECT \* from aws\_lambda.invoke(aws\_commons.create\_lambda\_function\_arn('LAMBDAARN', 'AWSREGION'), '{"body": "Data ingested from s3 file"}'::json );

Here LAMBDAARN value it to be replaced with :

arn:aws:lambda:us-west-2:854514635624:function:NotifyUpdate  
  
Note the “notifyupdate” in above line

o/p-



This will notify vendors with the shown message

3.24 email recd.

