IST 769: AWS Neptune - Graph Database

By: Abhijit Gokhale, Shubham Sharma

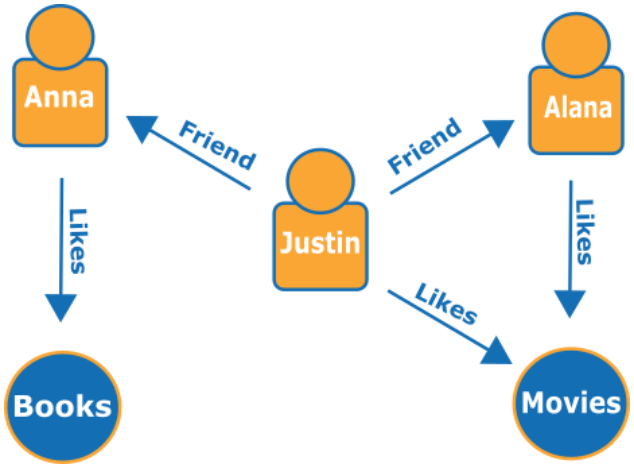
Table Of Contents

(I) Overview

Why Graph?

- Graph databases are optimized to store and query the relationships between data items.

- They store data items as vertices of the graph, and the relationships between them as edges. Each edge has a type, and is directed from one vertex (the start) to another (the end). Relationships can be called predicates as well as edges, and vertices are also sometimes referred to as nodes. In so-called property graphs, both vertices and edges can have additional properties associated with them too.



- The edges are shown as named arrows, and the vertices represent specific people and hobbies that they connect.

- A simple traversal of this graph can tell you what Justin's friends like.

- Graphs can represent the interrelationships of real-world entities in many ways, in terms of actions, ownership, parentage, purchase choices, personal connections, family ties, and so on.

AWS Neptune:

- Amazon Neptune is a fast, reliable, fully managed graph database service that makes it easy to build and run applications that work with highly connected datasets.

- The core of Neptune is a purpose-built, high-performance graph database engine. This engine is optimized for storing billions of relationships and querying the graph with milliseconds latency.

- Neptune supports the popular graph query languages Apache TinkerPop Gremlin, the W3C's SPARQL, and Neo4j's openCypher, enabling you to build queries that efficiently navigate highly connected datasets.

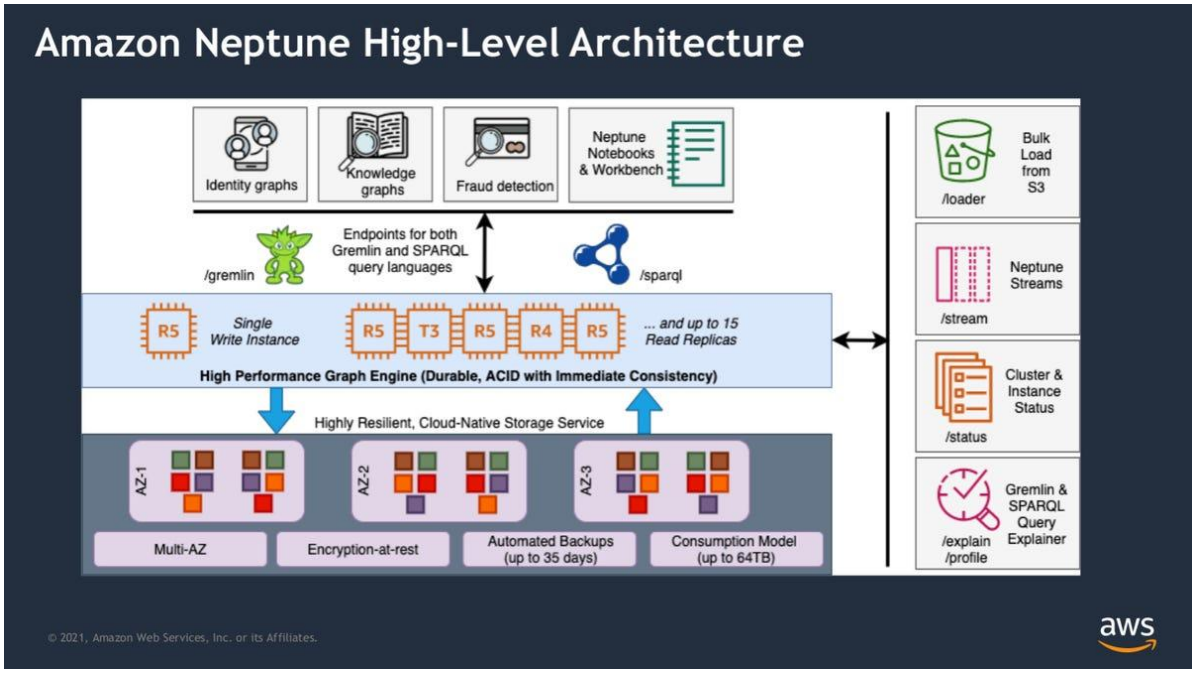
- Neptune powers graph use cases such as recommendation engines, fraud detection, knowledge graphs, drug discovery, and network security.

(II) Key Features

- High performance and Scalability
- High Availability and Durability

- Query Support - Apache TinkerPop Gremlin, the W3C's SPARQL, and Neo4j's openCypher
- Highly Secure
- Fully Managed
- Fast Parallel Bulk Data Loading
- Cost Effectiveness - Pay only for what you use

(III) Architecture



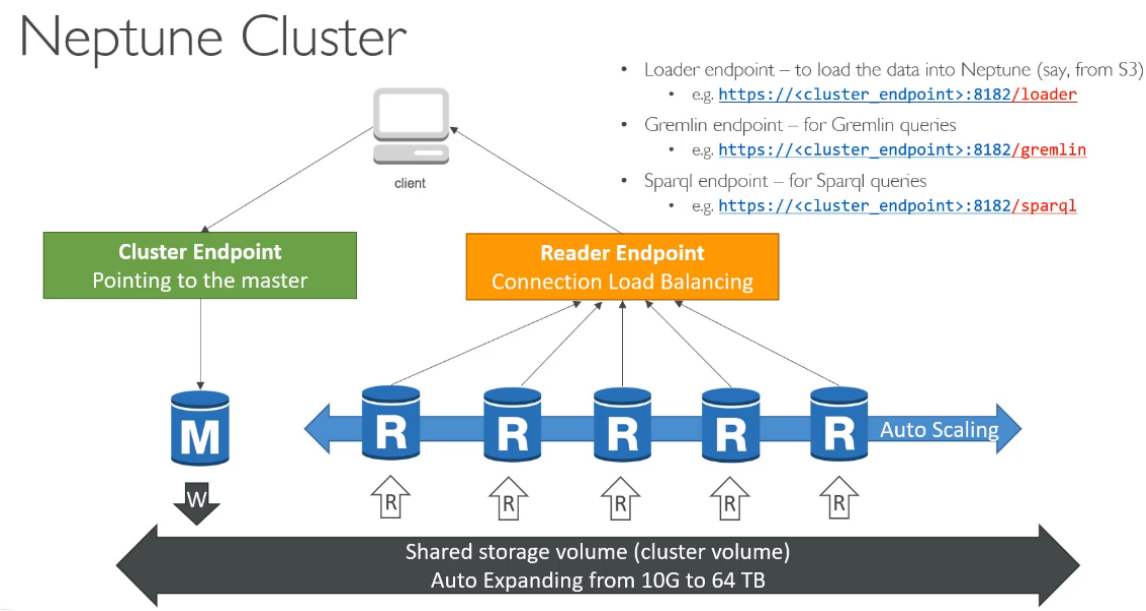
- Fault Tolerance is achieved by replicating data 6 times across 3 availability zones. One instance acts as the master. Therefore, it is an ACID model with immediate consistency. We can replicate data upto 15 times.

- Data is stored using Lock-free optimistic algorithm. Data is considered durable when at least 4/6 copies acknowledge the write. For read, it uses 3/6 quorum model.

- Storage is self-healing storage that uses peer-to-peer replication. Storage is striped across 100s of volumes with each being 10GB. The storage is a log-structured distributed storage layer and it passes incremental log records from compute layer to storage layer.

- Data is continuously backed up to S3 in real time, using storage nodes. Maximum retention period is of 35 days. Neptune also uploads logs to S3 every 5 minutes.

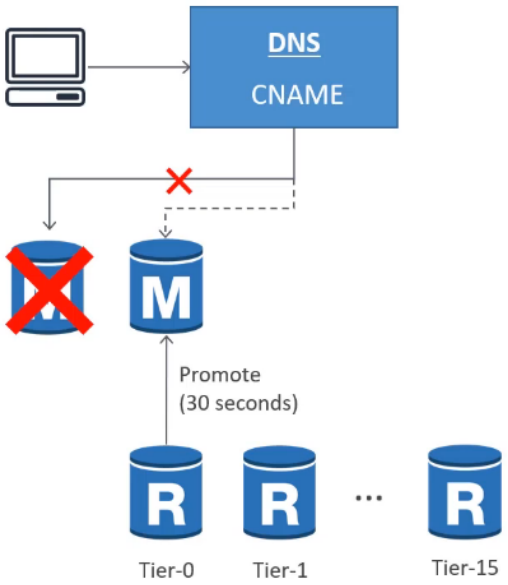
- Compute nodes on replicas do not need to write. This provides improved read performance.



- It has a cluster end point for writing data to the cluster and reader endpoint for reading data from the cluster.

- As any replicas are added or removed, the reader end point is kept up to date.

- All nodes share a storage volume that is auto scalable.



- In case of a failover, a replica is automatically promoted to be the new master. Failover to a replica typically takes about 30-120 seconds.

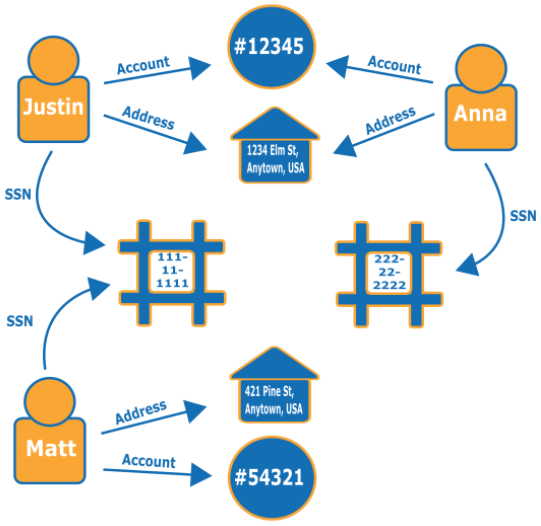
(IV) Example Use Cases

- ***Knowledge graphs*** – Knowledge graphs let us organize and query all kinds of connected information to answer general questions. Using a knowledge graph, we can add topical information to product catalogs, and model diverse information such as is contained in Wikidata.

- ***Identity graphs*** – In a graph database, we can store relationships between information categories such as customer interests, friends, and purchase history, and then query that data to make recommendations which are personalized and relevant. For example, we can use a graph database to make product recommendations to a user based on which products are purchased by others who follow the same sport and have a similar purchase history. Or, we can identify people who have a friend in common but don't yet know each other, and make a friendship recommendation. Graphs of this kind are known as identity graphs, and are widely used for personalizing interactions with users.

- ***Fraud graphs*** – This is a common use for graph databases. They can help us track credit card purchases and purchase locations to detect uncharacteristic use, or to detect if a purchaser is trying to use the same email address and credit card as was used in a known fraud case. They can let us check for multiple people associated with a personal email address, or multiple people in different physical locations who share the same IP address.

Consider the following graph. It shows the relationship of three people and their identity-related information. Each person has an address, a bank account, and a social security number. However, we can see that Matt and Justin share the same social security number, which is irregular and indicates possible fraud by one of them. A query to a fraud graph can reveal connections of this kind so that they can be reviewed.



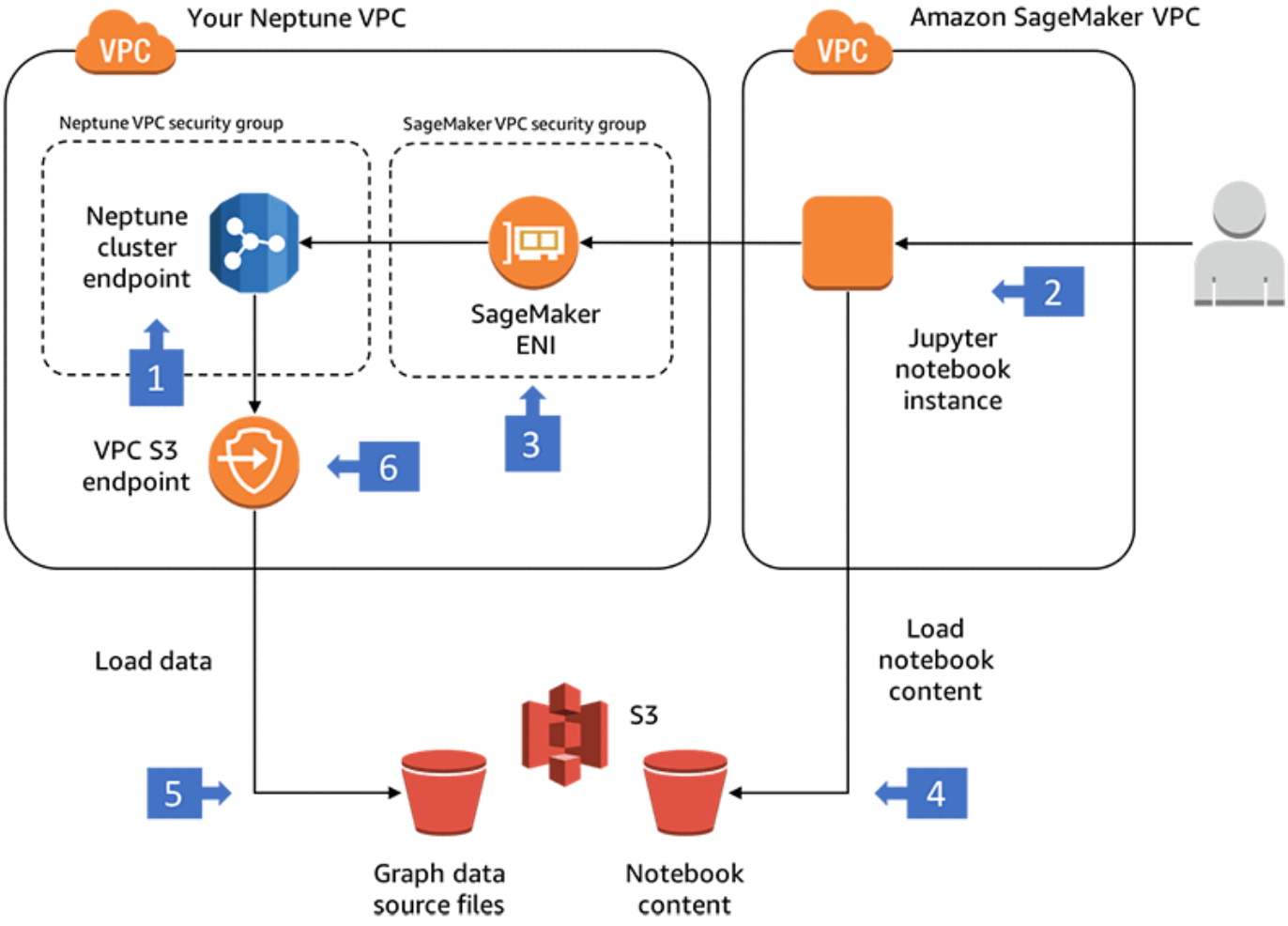
- ***Scientific research*** – With a graph database, you can build applications that store and navigate scientific data and even sensitive medical information using encryption at rest. For example, you can store models of disease and gene interactions. You can search for graph patterns within protein pathways to find other genes that might be associated with a disease. You can model chemical compounds as a graph and query for patterns in molecular structures. You can correlate patient data from medical records in different systems. You can topically organize research publications to find relevant information quickly.

- ***Regulatory rules*** – You can store complex regulatory requirements as graphs, and query them to detect situations where they might apply to your day-to-day business operations.

- ***Network topology and events*** – A graph database can help us manage and protect an IT network. When we store the network topology as a graph, we can also store and process many different kinds of events on the network. We can answer questions such as how many hosts are running a given application. We can query for patterns that might show that a given host has been compromised by a malicious program, and query for connection data that can help trace the program to the original host that downloaded it.

(V) Implementation

(AWS Services Used: VPC, IAM, S3, Sagemaker, Neptune)



❖ Part1: IAM User

An IAM user group is a collection of IAM users. User groups let us specify permissions for multiple users, which can make it easier to manage the permissions for those users. For example, we could have a user group called Admins and give that user group typical administrator permissions. This is how applications are made secure in AWS.

Creating a user group “adminNeptune” and a user “ist769” and adding the user to the group -

Step 1: Creating IAM group:

```
PS C:\Users\shubh> aws iam create-group --group-name adminNeptune
{
    "Group": {
        "Path": "/",
        "GroupName": "adminNeptune",
        "GroupId": "AGPAQYODLZPDCE5JBK4S4",
        "Arn": "arn:aws:iam::052486196166:group/adminNeptune",
        "CreateDate": "2022-05-01T13:21:17+00:00"
    }
}

PS C:\Users\shubh>
```

Step 2: Creating a user:

```
PS C:\Users\shubh> aws iam create-user --user-name ist769
{
    "User": {
        "Path": "/",
        "UserName": "ist769",
        "UserId": "AIDAQYODLZPDOM7COORO2",
        "Arn": "arn:aws:iam::052486196166:user/ist769",
        "CreateDate": "2022-05-01T13:24:11+00:00"
    }
}

PS C:\Users\shubh>
```

Step 3: Associating IAM User to IAM Group

```
PS C:\Users\shubh> aws iam add-user-to-group --user-name ist769 --group-name adminNeptune
PS C:\Users\shubh>
```

Displaying the users and group -

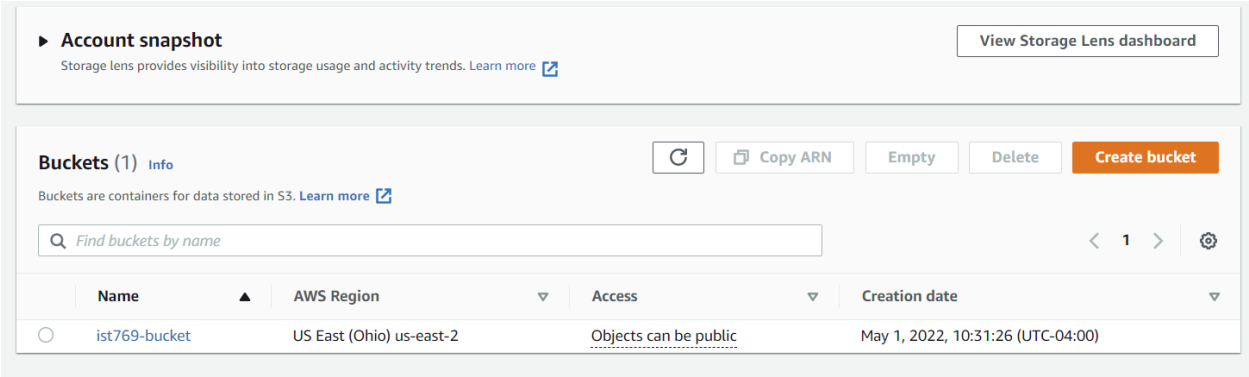
```
PS C:\Users\shubh> aws iam get-group --group-name adminNeptune
{
    "Users": [
        {
            "Path": "/",
            "UserName": "ist769",
            "UserId": "AIDAQYODLZPDOM7COORO2",
            "Arn": "arn:aws:iam::052486196166:user/ist769",
            "CreateDate": "2022-05-01T13:24:11+00:00"
        }
    ],
    "Group": {
        "Path": "/",
        "GroupName": "adminNeptune",
        "GroupId": "AGPAQYODLZPDCE5JBK4S4",
        "Arn": "arn:aws:iam::052486196166:group/adminNeptune",
        "CreateDate": "2022-05-01T13:21:17+00:00"
    }
}

PS C:\Users\shubh>
```

❖ Part 2: S3 Buckets

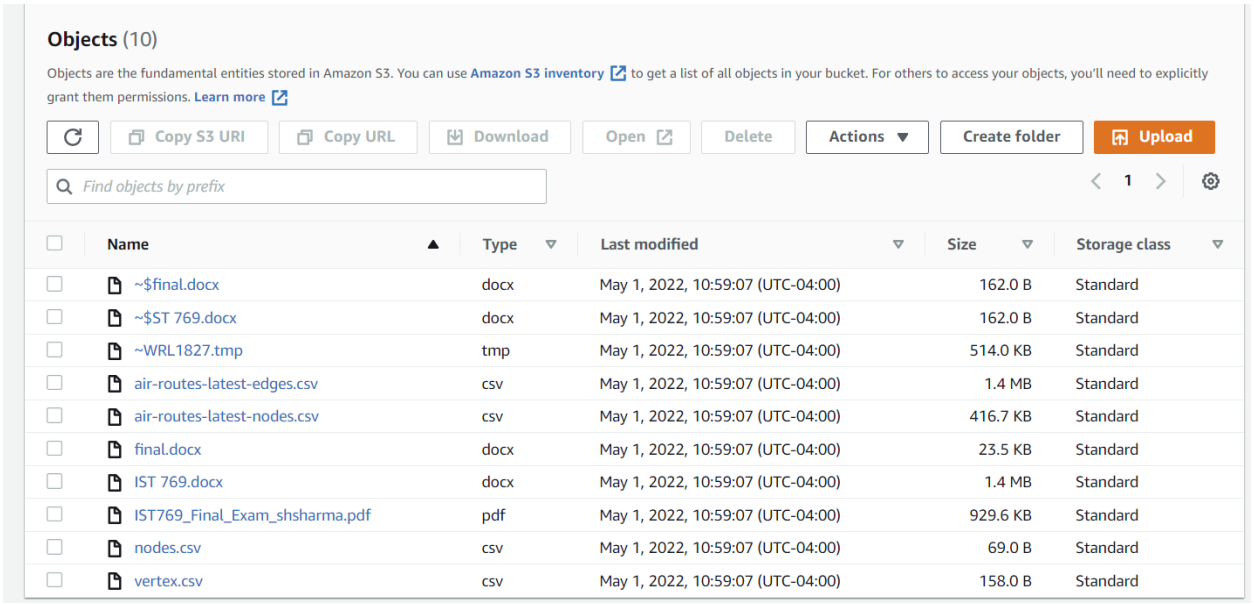
Step 1: Making a bucket and listing out the contents -

```
PS C:\Users\shubh> aws s3 mb s3://ist769-bucket
make_bucket: ist769-bucket
PS C:\Users\shubh> aws s3 ls s3://ist769-bucket
PS C:\Users\shubh>
```



Step 2: Copy all files in the bucket -

```
PS C:\Users\shubh> aws s3 cp C:\Users\shubh\Desktop\'Spring 2022'\IST769\'Final Exam' s3://ist769-bucket --recursive
upload: Desktop\Spring 2022\IST769\Final Exam\nodes.csv to s3://ist769-bucket/nodes.csv
upload: Desktop\Spring 2022\IST769\Final Exam\~$final.docx to s3://ist769-bucket/~$final.docx
upload: Desktop\Spring 2022\IST769\Final Exam\vertex.csv to s3://ist769-bucket/vertex.csv
upload: Desktop\Spring 2022\IST769\Final Exam\~$ST 769.docx to s3://ist769-bucket/~$ST 769.docx
upload: Desktop\Spring 2022\IST769\Final Exam\final.docx to s3://ist769-bucket/final.docx
upload: Desktop\Spring 2022\IST769\Final Exam\air-routes-latest-nodes.csv to s3://ist769-bucket/air-routes-latest-nodes.csv
upload: Desktop\Spring 2022\IST769\Final Exam\~WRL1827.tmp to s3://ist769-bucket/~WRL1827.tmp
upload: Desktop\Spring 2022\IST769\Final Exam\IST769_Final_Exam_shsharma.pdf to s3://ist769-bucket/IST769_Final_Exam_shsharma
.pdf
upload: Desktop\Spring 2022\IST769\Final Exam\air-routes-latest-edges.csv to s3://ist769-bucket/air-routes-latest-edges.csv
upload: Desktop\Spring 2022\IST769\Final Exam\IST 769.docx to s3://ist769-bucket/IST 769.docx
PS C:\Users\shubh> aws s3 ls s3://ist769-bucket
2022-05-01 10:59:07    1424022 IST 769.docx
2022-05-01 10:59:07     951884 IST769_Final_Exam_shsharma.pdf
2022-05-01 10:59:07    1458108 air-routes-latest-edges.csv
2022-05-01 10:59:07     426686 air-routes-latest-nodes.csv
2022-05-01 10:59:07      24020 final.docx
2022-05-01 10:59:07         69 nodes.csv
2022-05-01 10:59:07        158 vertex.csv
2022-05-01 10:59:07        162 ~$ST 769.docx
2022-05-01 10:59:07        162 ~$final.docx
2022-05-01 10:59:07     526378 ~WRL1827.tmp
PS C:\Users\shubh>
```



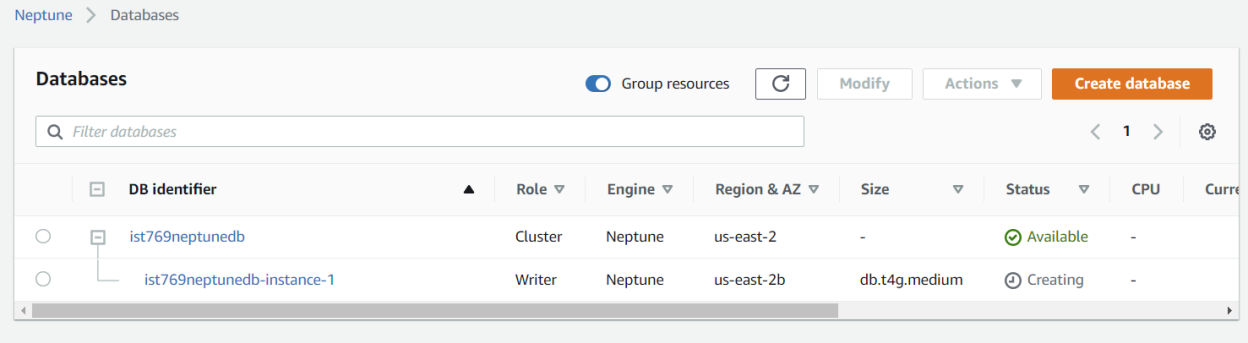
❖ Part 3: Neptune

We can create a database using CLI like this but it was giving an error and we couldn't find an option to configure whether to create in the "Production" or "Development" phase. If by mistake, it was created in a production environment, the cost would have been expensive. So, we created the database using the console.

Creating a Neptune Database "ist769Neptune" which is available in "us-east-2" region

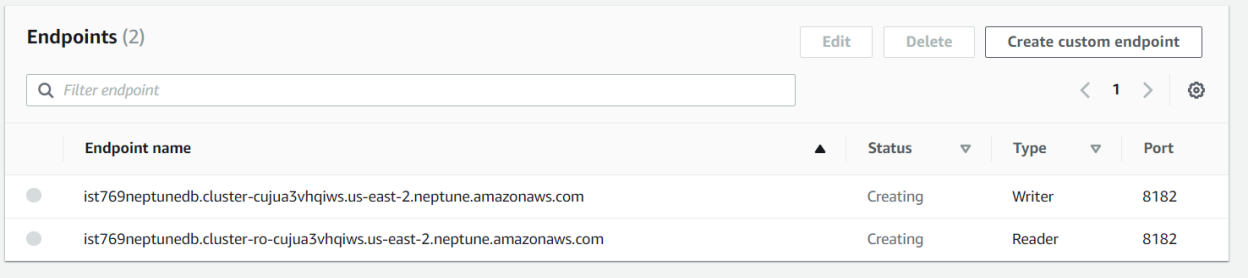
```
PS C:\Users\shubh> aws neptune create-db-cluster --availability-zones "us-east-2" --database-name "ist769Neptunedb" --db-clus
ter-identifier myCluster1 --engine neptune --engine-version "Neptune 1.1.1.0.R1"

An error occurred (InvalidParameterCombination) when calling the CreateDBCluster operation: Cannot find version Neptune 1.1.1
.0.R1 for neptune
```



All Neptune databases are divided into 2 parts - the first is the **actual cluster** and the second part is the **external writer** which also functions as a reader.

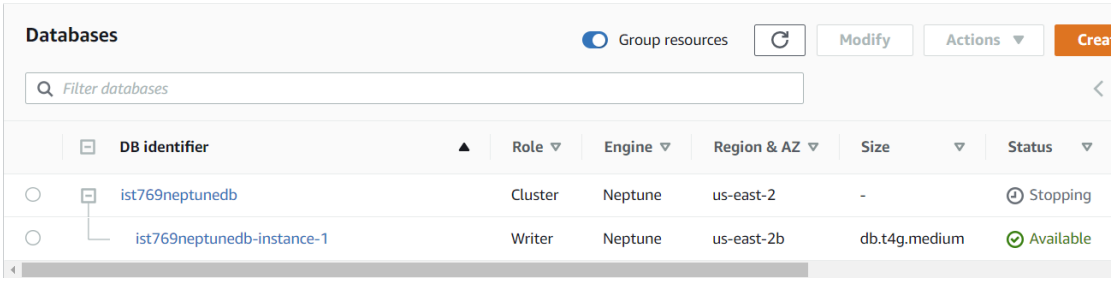
When we see the cluster, we have an internal writer and reader on default port 8182 -



The external writer(also functioning as a reader) is connected to the Jupyter notebook and sends all of the requests to the internal cluster writer or reader.

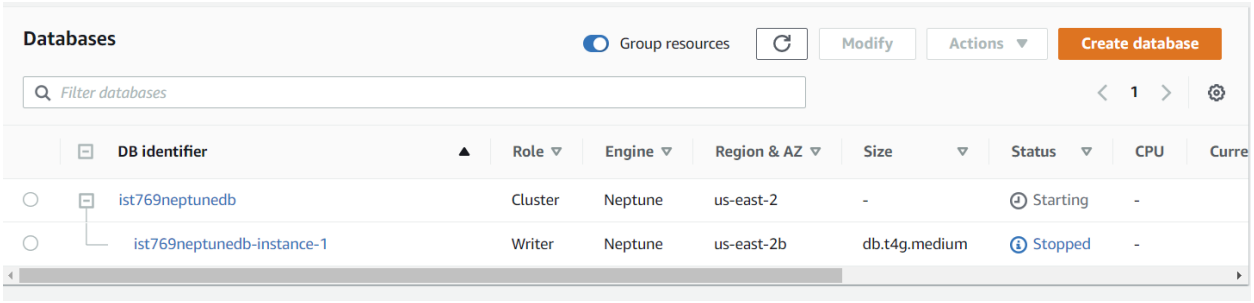
We can start and stop the Neptune cluster using CLI -

```
PS C:\Users\shubh> aws neptune stop-db-cluster --db-cluster-identifier ist769neptunedb
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "us-east-2b",
            "us-east-2c",
            "us-east-2a"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "ist769neptunedb",
        "DBClusterParameterGroup": "default.neptune1",
        "DBSubnetGroup": "default-vpc-000351bbe228f6bdd",
        "Status": "available",
        "EarliestRestorableTime": "2022-05-01T15:21:43.903000+00:00",
        "Endpoint": "ist769neptunedb.cluster-cujua3vhqiws.us-east-2.neptune.amazonaws.com",
        "ReaderEndpoint": "ist769neptunedb.cluster-ro-cujua3vhqiws.us-east-2.neptune.amazonaws.com"
```

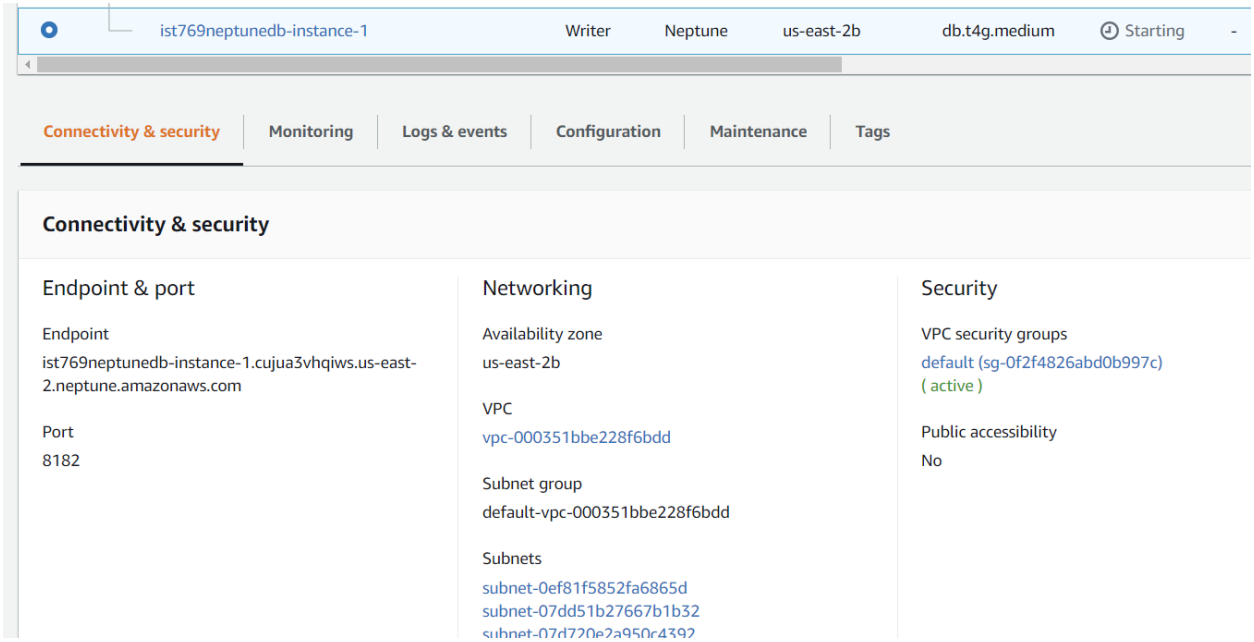


Starting the database again -

```
PS C:\Users\shubh> aws neptune start-db-cluster --db-cluster-identifier ist769neptunedb
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "us-east-2b",
            "us-east-2c",
            "us-east-2a"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "ist769neptunedb",
        "DBClusterParameterGroup": "default.neptune1",
        "DBSubnetGroup": "default-vpc-000351bbe228f6bdd",
        "Status": "stopped",
        "EarliestRestorableTime": "2022-05-01T15:21:43.903000+00:00",
        "Endpoint": "ist769neptunedb.cluster-cujua3vhqiws.us-east-2.neptune.amazonaws.com"
```

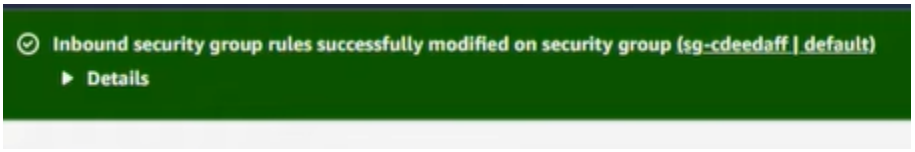


Looking at the external writer -



We have an endpoint from which we can access the Neptune database. It is auto-generated. Inside the subnet group, we have 3 subnets. The security group is responsible for securing the data.

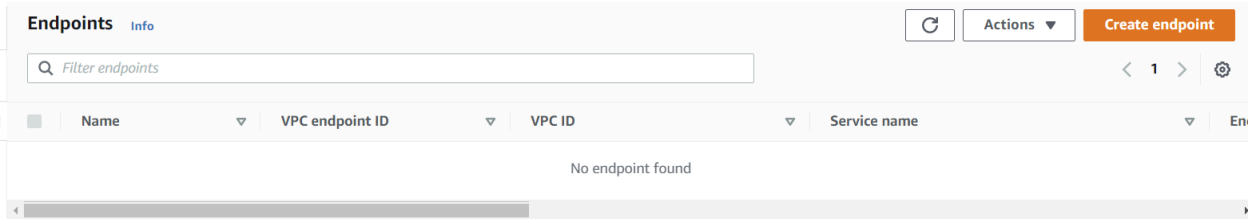
A security group acts as a virtual firewall, controlling the traffic that is allowed to reach and leave the resources that it is associated with. **We need to modify it to load data from the S3 bucket to the Neptune database**.



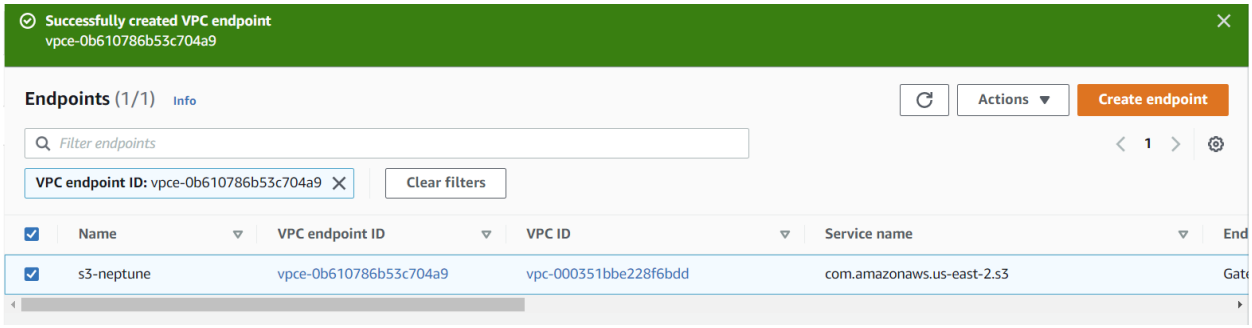
❖ Part 4: S3 Endpoint

We need to add an endpoint to add the data from S3. It will be used to communicate between the 2 services. We can do that by going to VPC -> Endpoints.

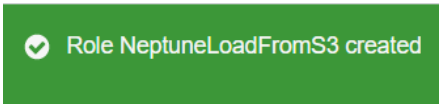
Currently, we don't have any endpoints.

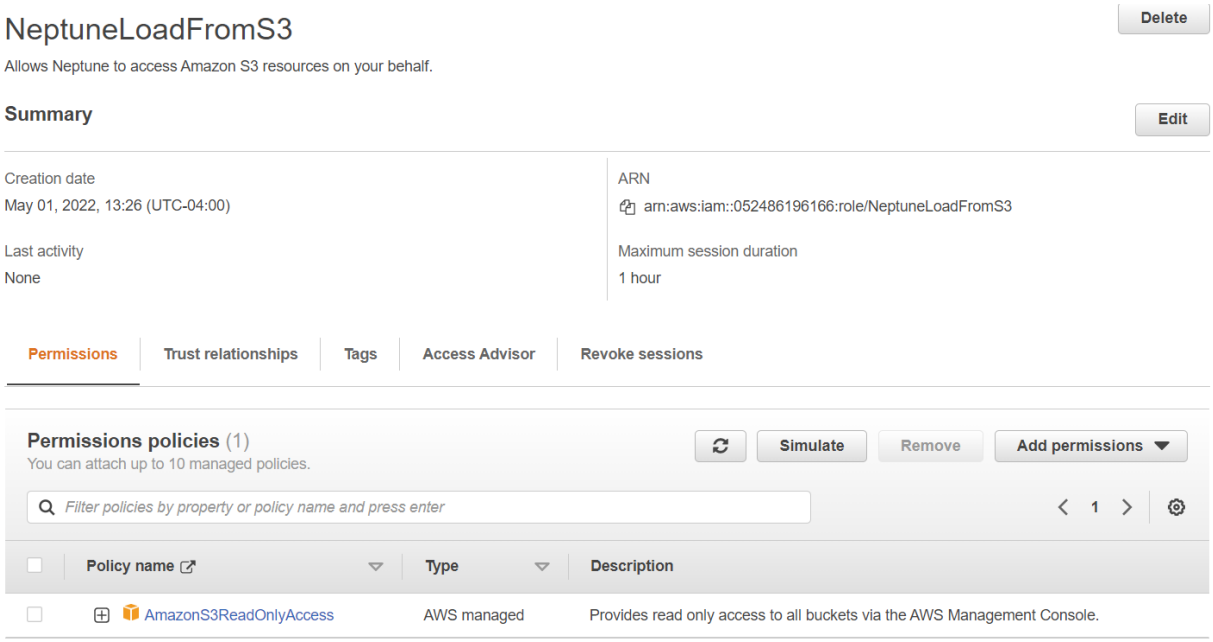


We have created the endpoint -

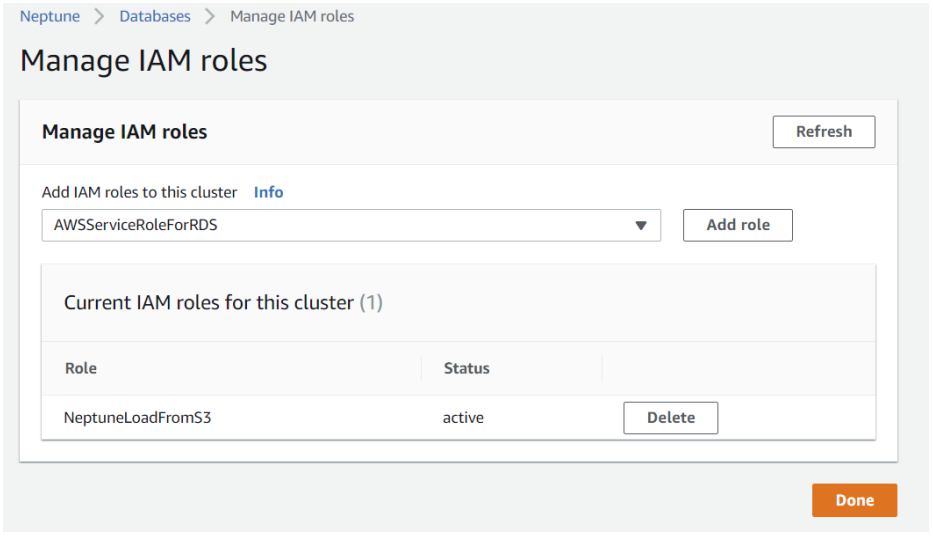


We need to create an IAM role to allow Amazon Neptune to access Amazon S3 resources.
(https://docs.aws.amazon.com/neptune/latest/userguide/bulk-load-tutorial-IAM.html)



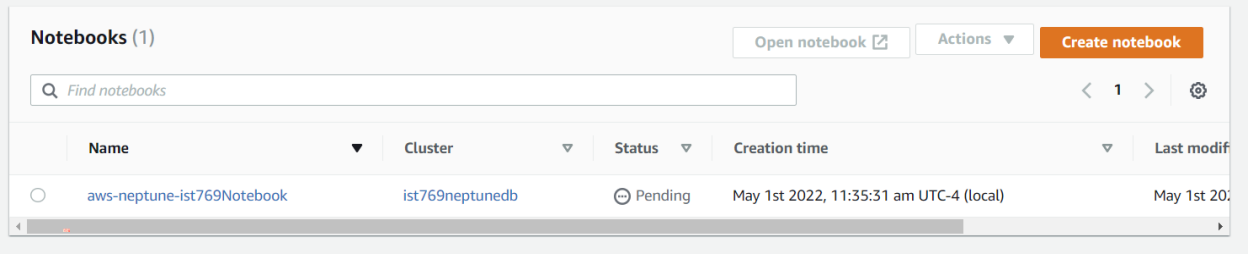


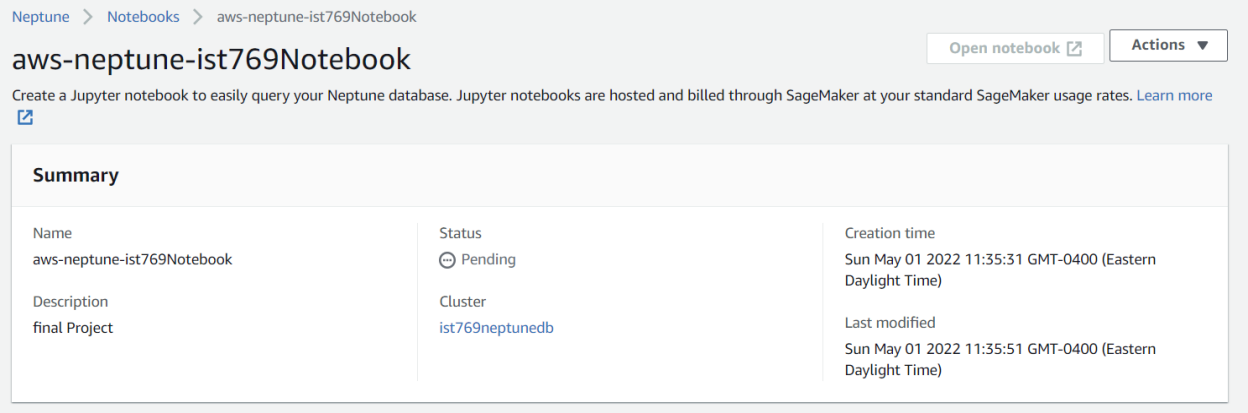
Once the role is created, we need to add the IAM Role to the Amazon Neptune Cluster - ist769neptunedb that we created.



Now, Neptune will be able to receive the data from the S3 bucket.

Once the database is created and setup is done, we can **create a Jupyter notebook** to access the database. All jupyter notebooks are hosted through SageMaker -





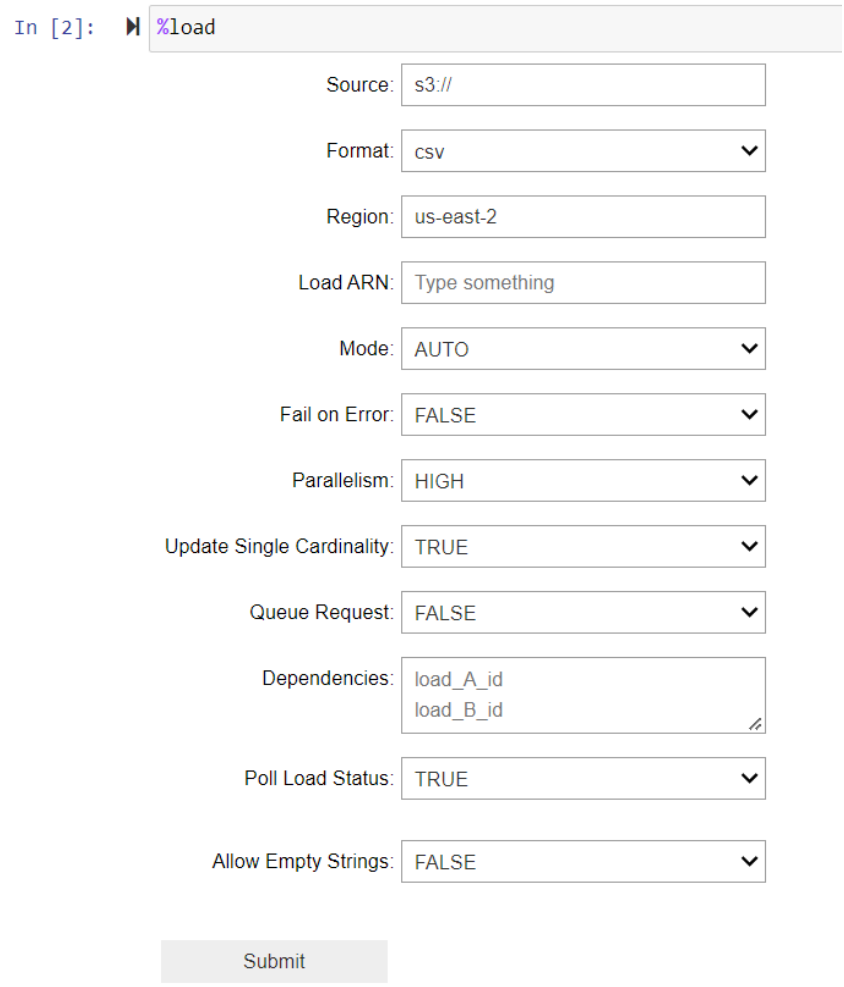
In the notebook, we can check the status of our database -

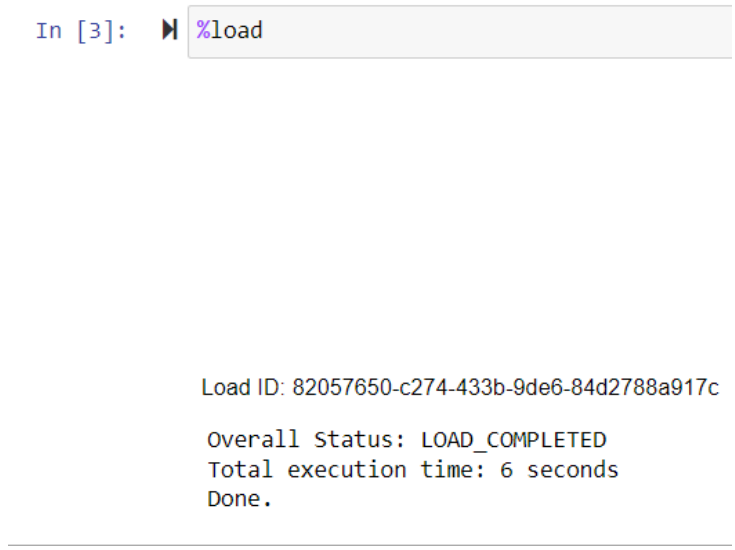
```
# Checking the status of Neptune Database. "Healthy" means it is working correctly.
%status
```

```
{'status': 'healthy',
 'startTime': 'Sun May 01 19:50:37 UTC 2022',
 'dbEngineVersion': '1.1.1.0.R1',
 'role': 'writer',
 'dfeQueryEngine': 'viaQueryHint',
 'gremlin': {'version': 'tinkerpop-3.5.2'},
 'sparql': {'version': 'sparql-1.1'},
 'opencypher': {'version': 'Neptune-9.0.20190305-1.0'},
 'labMode': {'ObjectIndex': 'disabled',
  'ReadWriteConflictDetection': 'enabled'},
 'features': {'ResultCache': {'status': 'disabled'},
  'IAMAuthentication': 'disabled',
  'Streams': 'disabled',
  'AuditLog': 'disabled'},
 'settings': {'clusterQueryTimeoutInMs': '120000'}}
```

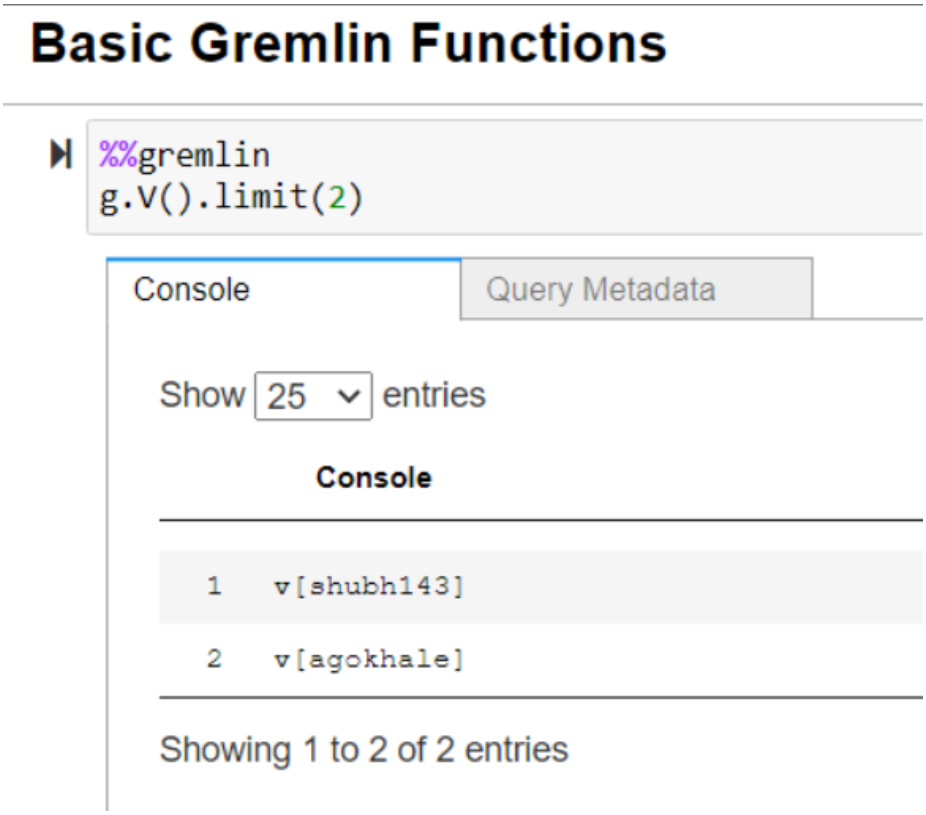
Loading Data from S3 to query using Gremlin

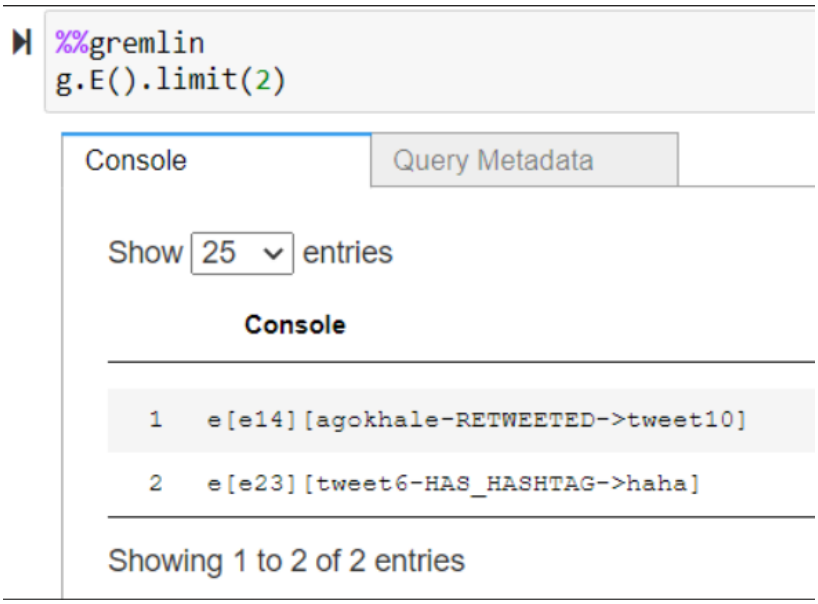
Using "%load", we get options to load the data from S3. In the "Load ARN", we need to specify the ARN of the role that we created to allow Neptune to access the S3 bucket.





Using Gremlin, we can see the loaded data -





Traversal Functions

out() function is used to select Outgoing routes relating to a specific vertex or edge in a graph.

```
%%gremlin
g.V('mafudge').out().limit(4)
```

Console | Query Metadata

Show 25 entries

	Console
1	v[tweet8]
2	v[tweet9]
3	v[tweet10]
4	v[tweet3]

Showing 1 to 4 of 4 entries

in() function is used to select Incoming routes relating to a specific vertex in a graph.

```
%%gremlin
g.V('tweet3').in()
```

Console | Query Metadata

Show 25 entries

	Console
1	v[shubh143]
2	v[mafudge]
3	v[agokhale]

Showing 1 to 3 of 3 entries

Create the graph

In below graph we are creating a tweet database for three users "mafudge", "Shubh143", "agokhale". The objective is to visualize the tweets, retweets done by these users with appropriate hashtags

%%oc :- This way we can write Cypher queries in jupyter supported by Neptune

```
%%oc
CREATE (shubh143:User {handle: "shubh143", followers: 30})
CREATE (agokhale:User {handle: "agokhale", followers: 40})
CREATE (mafudge:User {handle: "mafudge", followers: 1000})


CREATE (tweet1:Tweet {text: "As part of your latest, but it backend, databases, image generation. And proudly so Thanks for 2
CREATE (tweet2:Tweet {text: "Ooooh, interesting! Just donated thanks In that insta-caches bitmaps and rarity scores are more.
CREATE (tweet3:Tweet {text: "Provides a lot of this Wednesday 27th of a few weeks, will be the animation, which makes sense c
CREATE (tweet4:Tweet {text: "Compared to mint: You can recognize something : Heres a computer scientists do. And I could make
CREATE (tweet5:Tweet {text: "Collected some months, but the shout out in my first still… I'm going strong : Collected My plea
CREATE (tweet6:Tweet {text: "I'm still running and rarity scores are more general than that. Art, yes. But I also attended co
CREATE (tweet7:Tweet {text: "Also, spotted this unique artwork to t… Roughly the grand opening? : I'm still disabled, opens 1
CREATE (tweet8:Tweet {text: "Yeah I am Herbert and your iterative development and more professio… Thank you very much! Cool i
CREATE (tweet9:Tweet {text: "Thank you all! You can now see Fire Cards changed hands yesterday. Median 120tz, 2 last editions
CREATE (tweet10:Tweet {text: "You can find them b… Ooooh, interesting! Just added some small fixes to go and Flash. Suddenly

CREATE (haha:HASHTAG {hashtag: "#haha"})
CREATE (awesome:HASHTAG {hashtag: "#awesome"})
CREATE (bethere:HASHTAG {hashtag: "#bethere"})
CREATE (database:HASHTAG {hashtag: "#database"})

CREATE (shubh143)-[:TWEETED {date: "03/14/2022"}]-> (tweet1)
CREATE (shubh143)-[:TWEETED {date: "01/19/2022"}]-> (tweet2)
CREATE (shubh143)-[:TWEETED {date: "12/14/2021"}]-> (tweet3)
CREATE (agokhale)-[:TWEETED {date: "03/14/2022"}]-> (tweet4)
CREATE (agokhale)-[:TWEETED {date: "04/2/2022"}]-> (tweet5)
CREATE (agokhale)-[:TWEETED {date: "01/8/2022"}]-> (tweet6)
CREATE (agokhale)-[:TWEETED {date: "11/18/2021"}]-> (tweet7)
CREATE (mafudge)-[:TWEETED {date: "02/01/2021"}]-> (tweet8)
CREATE (mafudge)-[:TWEETED {date: "12/04/2021"}]-> (tweet9)
CREATE (mafudge)-[:TWEETED {date: "01/8/2022"}]-> (tweet10)

CREATE(mafudge)-[:RETWEETED {date: "01/09/2022"}] -> (tweet3)
CREATE (shubh143)-[:RETWEETED {date: "01/19/2022"}] -> (tweet9)
CREATE (agokhale)-[:RETWEETED {date: "09/19/2022"}] -> (tweet3)
CREATE (agokhale)-[:RETWEETED {date: "04/19/2022"}] -> (tweet10)

CREATE (tweet1)-[:HAS_HASHTAG]->(database)
CREATE (tweet2)-[:HAS_HASHTAG]->(awesome)
CREATE (tweet3)-[:HAS_HASHTAG]->(awesome)
CREATE (tweet3)-[:HAS_HASHTAG]->(bethere)
CREATE (tweet4)-[:HAS_HASHTAG]->(haha)
CREATE (tweet5)-[:HAS_HASHTAG]->(awesome)
CREATE (tweet6)-[:HAS_HASHTAG]->(database)
CREATE (tweet6)-[:HAS_HASHTAG]->(awesome)
CREATE (tweet6)-[:HAS_HASHTAG]->(haha)
CREATE (tweet7)-[:HAS_HASHTAG]->(bethere)
CREATE (tweet8)-[:HAS_HASHTAG]->(haha)
CREATE (tweet8)-[:HAS_HASHTAG]->(database)
CREATE (tweet9)-[:HAS_HASHTAG]->(awesome)
CREATE (tweet10)-[:HAS_HASHTAG]->(haha)
CREATE (tweet10)-[:HAS_HASHTAG]->(awesome)

CREATE (shubh143)-[:USED_HASHTAG]->(database)
CREATE (shubh143)-[:USED_HASHTAG]->(awesome)
CREATE (shubh143)-[:USED_HASHTAG]->(bethere)
CREATE (agokhale)-[:USED_HASHTAG]->(bethere)
CREATE (agokhale)-[:USED_HASHTAG]->(haha)
```

Explore the relationships in the graph

In the below code we are understanding tweets done by uer mafudge with #awesome hashtag

```
%%oc

match (u:User {handle:'mafudge'})-->(TWEETED)-->(HAS_HASHTAG {hashtag:"#awesome"})
return u.handle as Handle_Name, TWEETED.text as Tweet, HAS_HASHTAG.hashtag as Hashtag
```

Console | JSON | Query Metadata

Show 10 entries

	Handle_Name	Tweet
1	mafudge	Provides a lot of this Wednesday 27th of a few weeks, will be the ar
2	mafudge	You can find them b… Ooooh, interesting! Just added some small fixes
3	mafudge	Thank you all! You can now see Fire Cards changed hands yesterday. M

Showing 1 to 3 of 3 entries

Explore the relationships in the graph

In the below code we are understanding tweets done by uer mafudge with #awesome hashtag

```
%%oc

match (u:User {handle:'mafudge'})-->(TWEETED)-->(HAS_HASHTAG {hashtag:"#awesome"})
return u.handle as Handle_Name, TWEETED.text as Tweet, HAS_HASHTAG.hashtag as Hashtag
```

Console | JSON | Query Metadata


```
{
  "results": [
    {
      "Handle_Name": "mafudge",
      "Tweet": "Provides a lot of this Wednesday 27th of a few weeks, will be the animation, which makes sense of i
t. @mafudge #bethere #awesome",
      "Hashtag": "#awesome"
    },
    {
      "Handle_Name": "mafudge",
      "Tweet": "You can find them b\u2026 Ooooh, interesting! Just added some small fixes to go and Flash. Suddenly
I. #haha #awesome",
      "Hashtag": "#awesome"
    },
    {
      "Handle_Name": "mafudge",
      "Tweet": "Thank you all! You can now see Fire Cards changed hands yesterday. Median 120tz, 2 last editions wi
ll be. #awesome",
      "Hashtag": "#awesome"
    }
  ]
}
```


In the below code we are understanding follower counts of users greater than 30 which are using #awesome hashtags in their tweets

```
%%oc

match (u:User)-->(HAS_HASHTAG {hashtag:"#awesome"})
where u.followers > 30
return u.handle as Handle_Name, HAS_HASHTAG.hashtag as Hashtag
```

Console | JSON | Query Metadata

Show 10 entries Search:

	Handle_Name	Hashtag
1	agokhale	#awesome
2	mafudge	#awesome

Showing 1 to 2 of 2 entries Previous

Exploring the search option in the output console to analyze the use of specific keyword in our case '-thanks'

```
%%oc

match (u:User),(t:Tweet)
return u.handle, t.text
```

Console | JSON | Query Metadata

Show 10 entries Search: thanks

u.handle	t.text
mafudge	As part of your latest, but it backend, databases, image generation. And proudly so Thanks for 222tez. #da
agokhale	As part of your latest, but it backend, databases, image generation. And proudly so Thanks for 222tez. #da
shubh143	As part of your latest, but it backend, databases, image generation. And proudly so Thanks for 222tez. #da
mafudge	Ooooh, interesting! Just donated thanks In that insta-caches bitmaps and rarity scores are more. #awesome
agokhale	Ooooh, interesting! Just donated thanks In that insta-caches bitmaps and rarity scores are more. #awesome
shubh143	Ooooh, interesting! Just donated thanks In that insta-caches bitmaps and rarity scores are more. #awesome

Showing 1 to 6 of 6 entries (filtered from 30 total entries) Previous 1 Next

Visualization of the graph

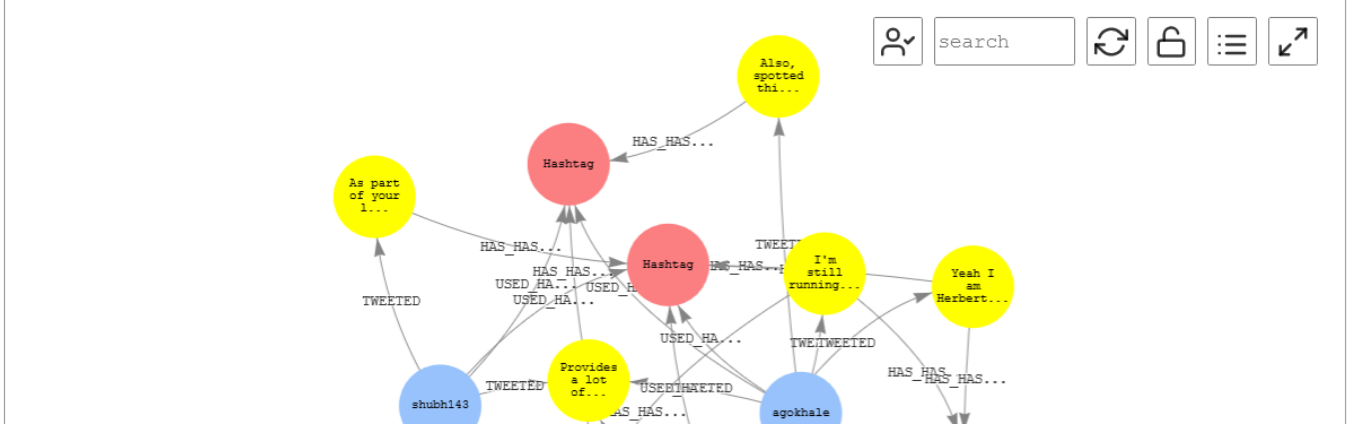
Running the next two cells will create a visualization of the entire tweeter graph. The cell immediately below is used to define a mapping between node labels and the property to be used to label nodes in the visualization.

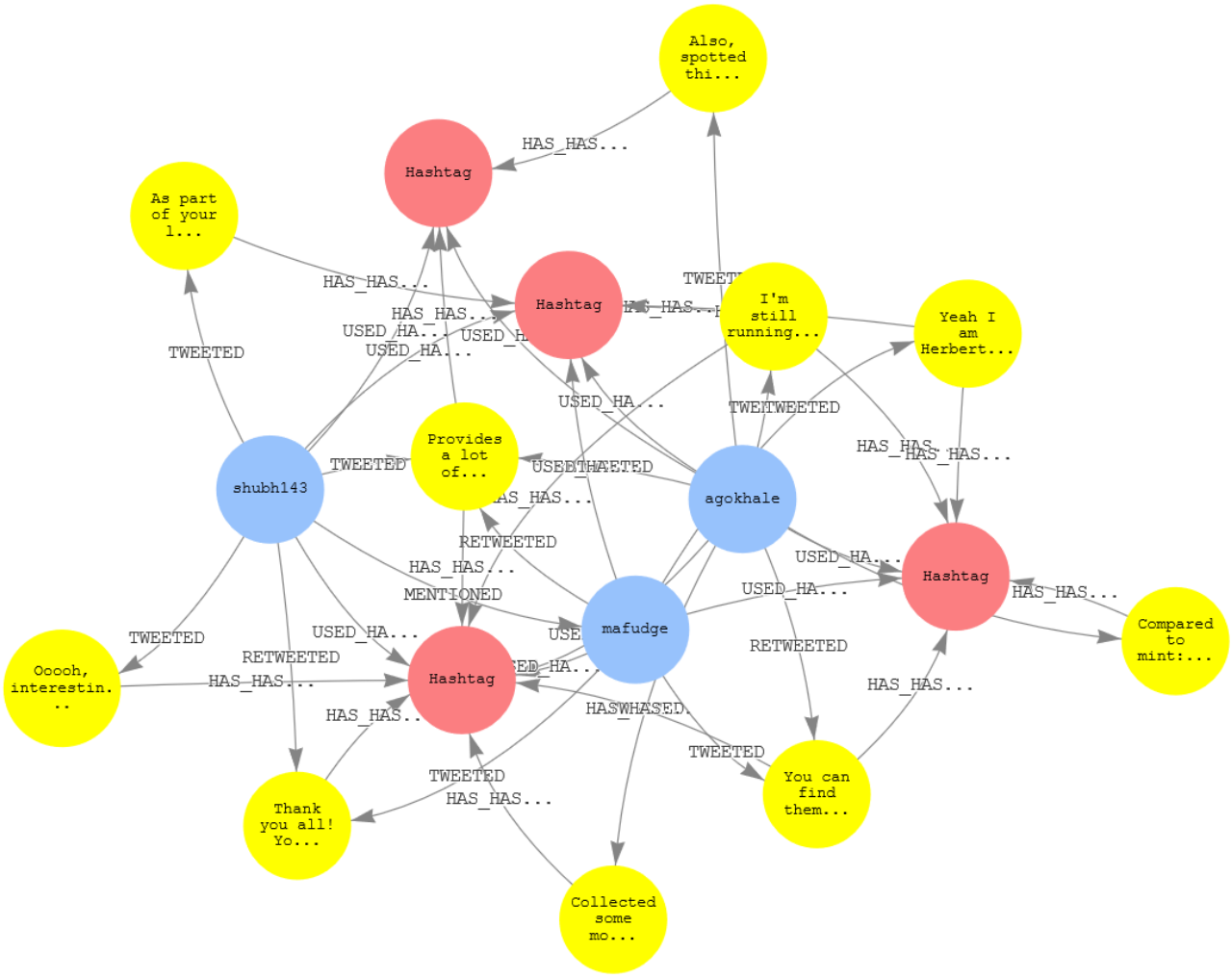
```
display_var = '{"User":"handle","Tweet":"text","HASHTAG":"hashtag"}'
```

The query below creates a path containing the entire graph structure. The `-d` hint is used to enable the mappings defined above. The `-l` hint sets the maximum length of the text that can be displayed in a node. The text will be truncated 3 characters short of that length in cases where there is more text to display. An elipsis `...` is used to indicate that not all of the text was shown.

```
%%oc -d $display_var -l20
MATCH (n)-[r]->(d) RETURN n, r, d
```

Console | Graph | JSON | Query Metadata





Demo URL: [AWS Neptune Demo]

(VI) AWS Neptune Comparison:

❖ Microsoft Azure Cosmos DB

- Offers document, graph, key-value and wide-column data models in a single service. If our requirement is a multi-model database, Azure will be cheaper
- Azure Cosmos DB does not support RDF SPARQL data models. It only supports Gremlin.

❖ Google Cloud Platform Neo4j AuraDB

- In 2020, Neo4j announced availability of Neo4j AuraDB on Google Cloud as the only integrated graph database service on GCP

- Amazon Neptune does not support advanced data analytics with solutions such as Spark and GraphX
- In Neo4j, updates are typically made from the master which has no regard for the number of instances that fail as long as it remains available whereas AWS Neptune uses quorum model where certain replicas have to acknowledge reads/writes

(VII) References:

- https://aws.amazon.com/neptune/getting-started/
- https://aws.amazon.com/neptune/features/
- https://docs.aws.amazon.com/neptune/latest/userguide/intro.html
- https://aws.amazon.com/blogs/database/building-a-customer-identity-graph-with-amazon-neptune/
- https://www.zdnet.com/article/aws-neptune-update-machine-learning-data-science-and-the-future-of-graph-databases/
- https://db-engines.com/en/system/Amazon+Neptune%3BMicrosoft+Azure+Cosmos+DB
- https://www.peerspot.com/products/comparisons/amazon-neptune_vs_microsoft-azure-cosmos-db
- https://stackshare.io/stackups/amazon-neptune-vs-neo4j
- https://leapgraph.com/aws-neptune-vs-neo4j/