



AWS
re:Invent

D A T 3 4 7 - R

Neptune best practices: How to optimize your graph queries

Michael Schmidt

Sr. Software Development Engineer
Amazon Web Services

Agenda

Amazon Neptune introduction, architecture, and monitoring

Query optimization and performance tuning

Neptune introduction, architecture, and monitoring

Graph use cases



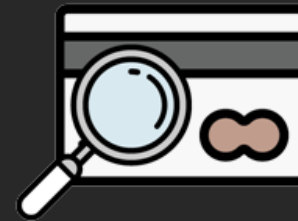
Social
networking



Recommendations



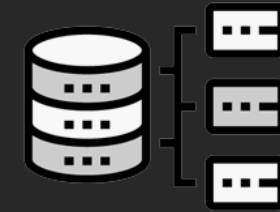
Knowledge
graphs



Fraud detection



Life Sciences



Network and IT
operations

Connected data queries

Navigate (variably) connected structure

Filter or compute a result based on *strength*, *weight*, or *quality* of relationships

Neptune: Fully managed graph database

Fast



Query billions of relationships
with millisecond latency

Reliable



Six replicas of your data across
three AZs with full backup and
restore

Easy



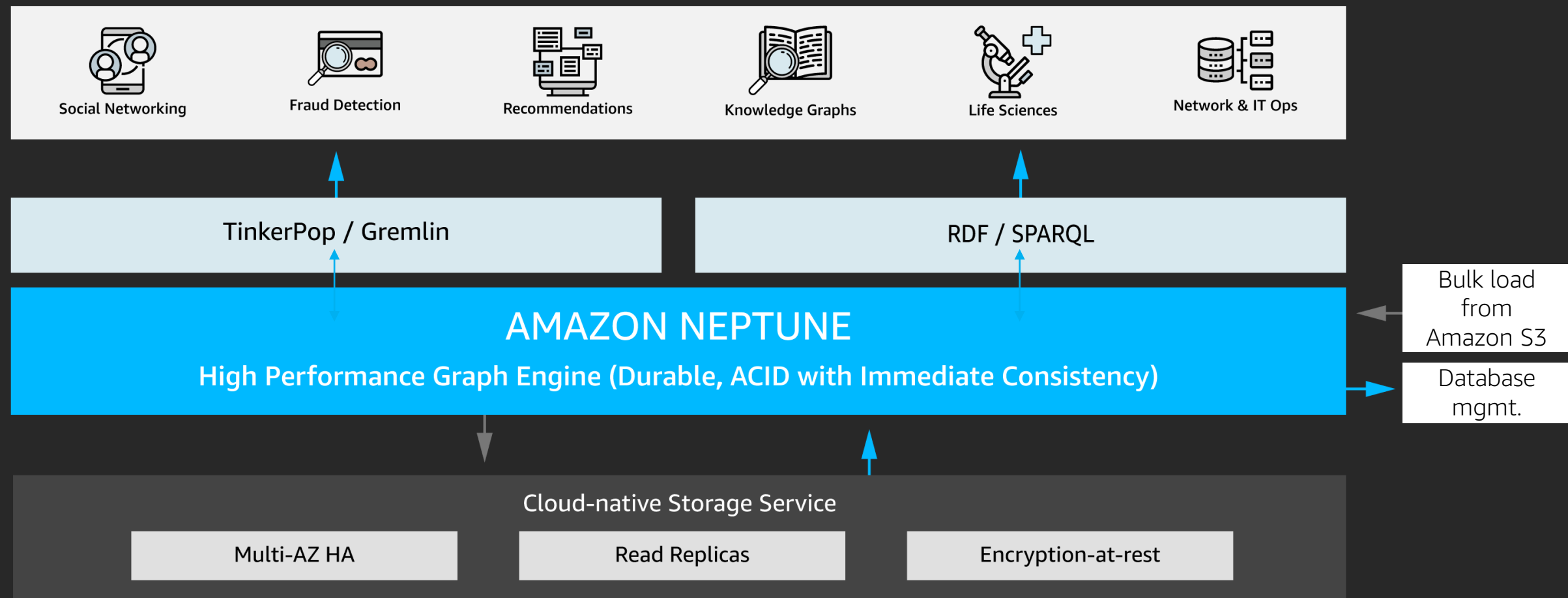
Build powerful queries easily
with Gremlin and SPARQL

Open



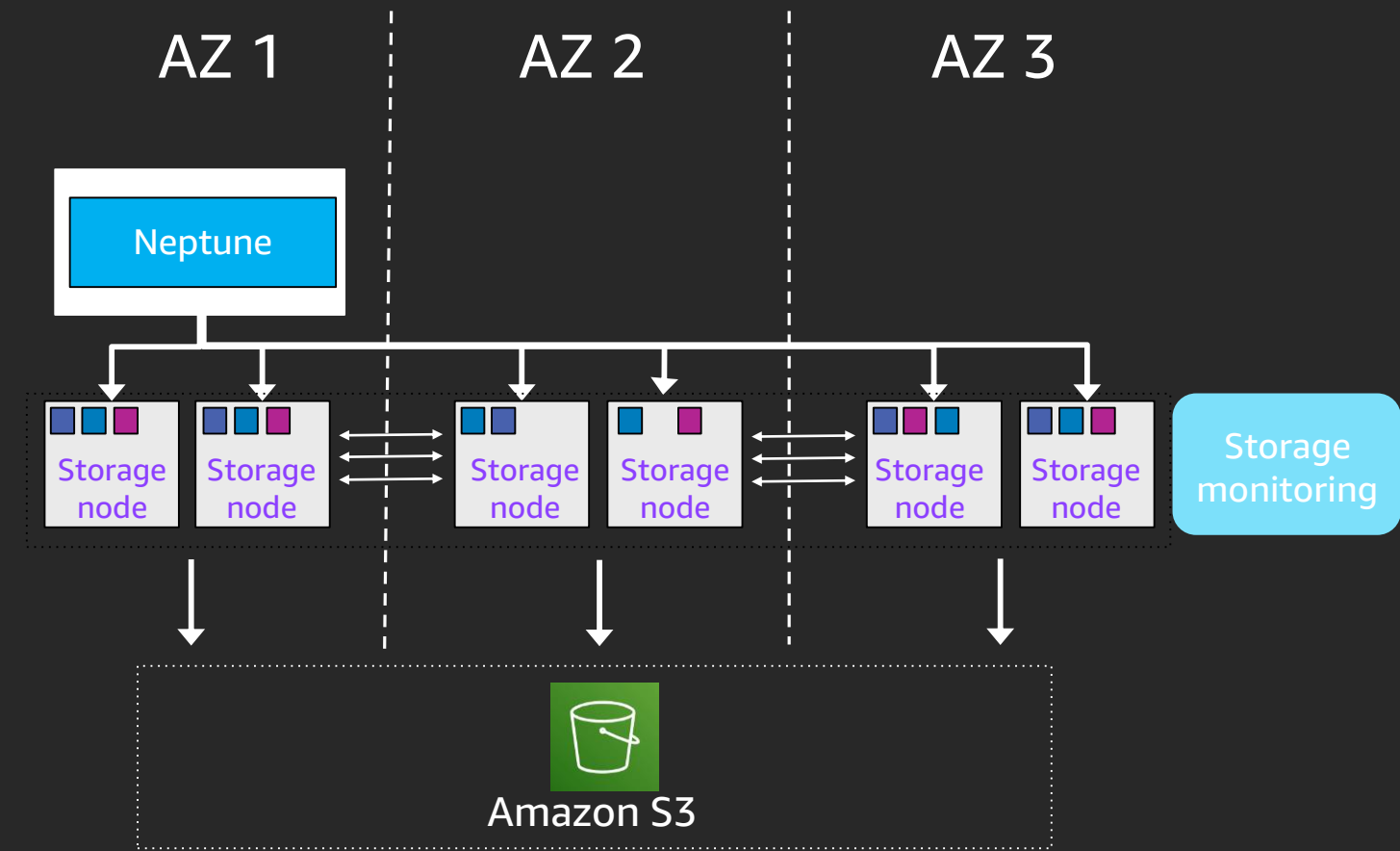
Supports Apache TinkerPop
and W3C RDF graph models

Neptune architecture



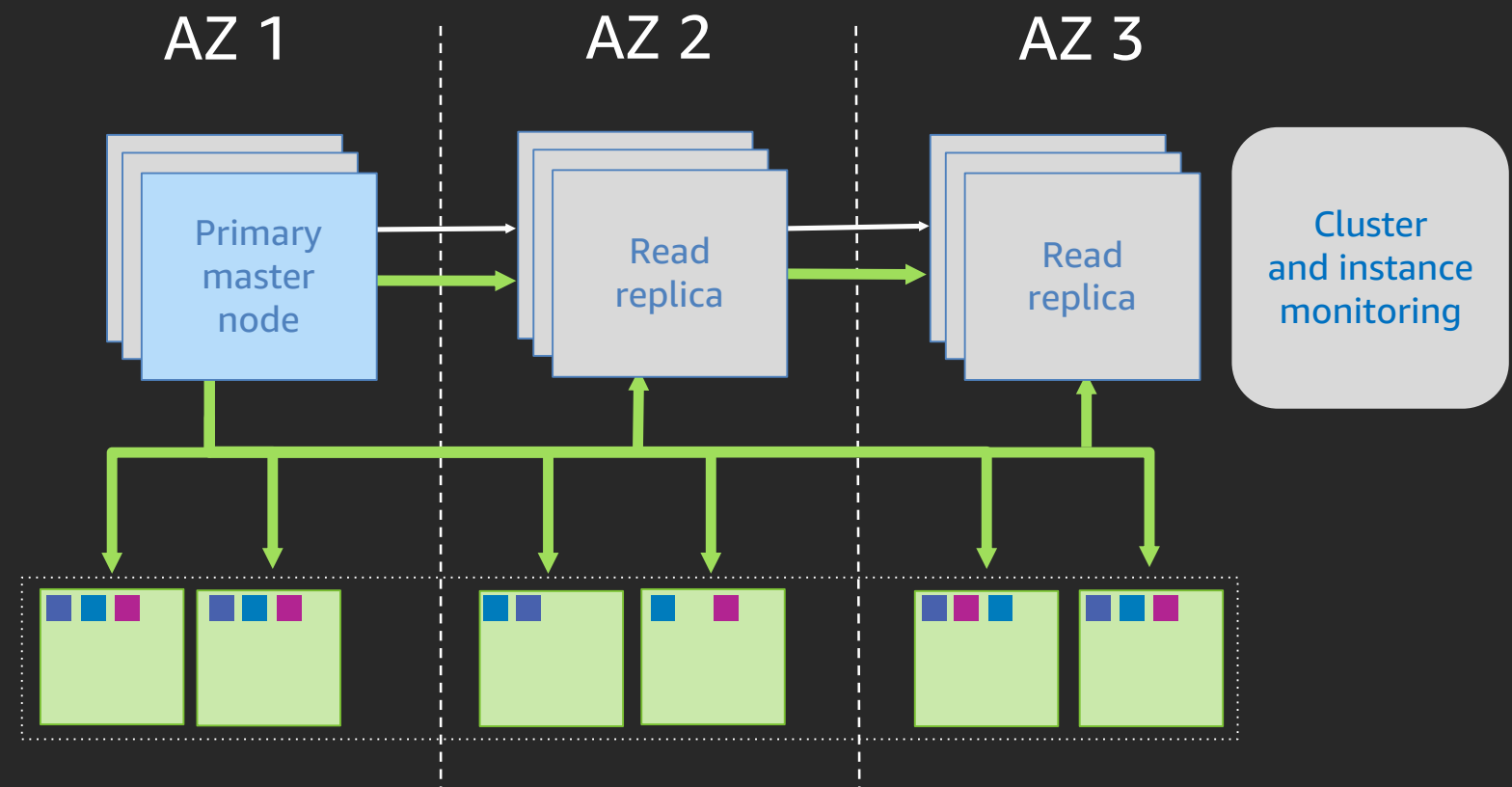
Cloud-native storage

- Data is replicated six times across three AZs
- Continuous backup to Amazon S3
 - Built for eleven nines of durability
- Continuous monitoring of nodes and disks
- 10 GB segments as unit of repair of hotspot rebalance
- Quorum system for read/write; latency tolerant
- Quorum membership changes do not stall writes
- Storage volume automatically grows up to 64 TB



Read replicas

- Availability
- Failing database nodes are automatically detected and replaced
- Failing database processes are automatically detected and recycled
- Replicas are automatically promoted to primary if needed (failover)
- Customer specifiable failover order
- Performance
- Customer applications can scale out read traffic across read replicas
- Read balancing across read replicas
 - Use reader endpoint



Monitoring

AWS CloudTrail

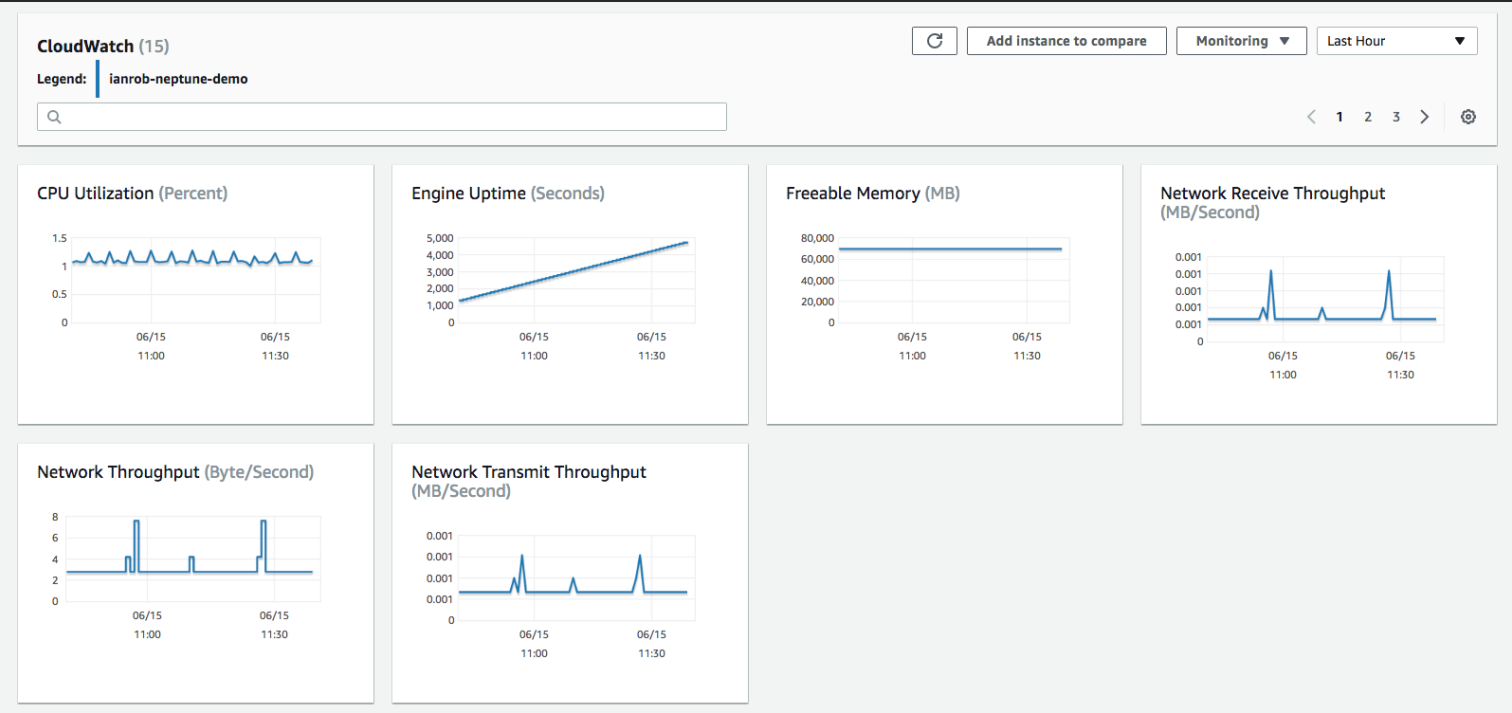
Log all Neptune API calls to S3 buckets

Event notifications

Create Amazon SNS subscription via CLI or SDK

Sources: db-instance | db_cluster |
db-parameter-group | db-security-group |
db-snapshot | db-cluster-snapshot

Amazon CloudWatch



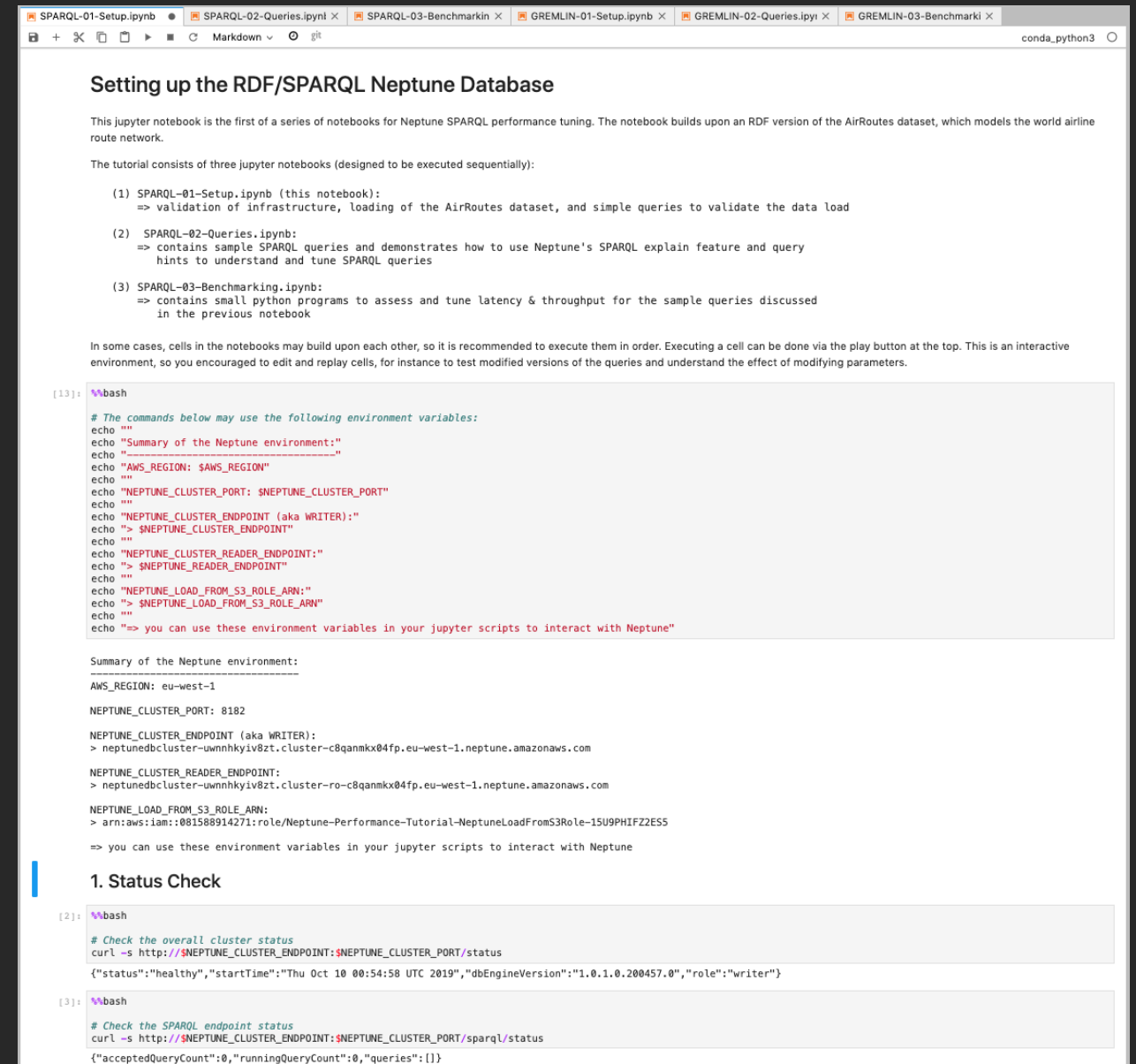
BackupRetentionPeriodStorageUsed	GremlinRequestsPerSec	NumTxCommitted	TotalRequestsPerSec
CPUUtilization	GremlinWebSocketOpenConnections	NumTxOpened	TotalServerErrorsPerSec
ClusterReplicaLag	LoaderRequestsPerSec	NumTxRolledBack	VolumeBytesUsed
ClusterReplicaLagMaximum	MainRequestQueuePendingRequests	SnapshotStorageUsed	VolumeReadIOPs
ClusterReplicaLagMinimum	NetworkReceiveThroughput	SparqlRequestsPerSec	VolumeWriteIOPs
EngineUptime	NetworkThroughput	TotalBackupStorageBilled	
FreeableMemory	NetworkTransmitThroughput	TotalClientErrorsPerSec	

Hands on!

Run the experiments in your Jupyter notebook to learn about:

- Basic Neptune APIs
- Loading data into Neptune

<https://dashboard.eventengine.run/login>



```
SPARQL-01-Setup.ipynb | SPARQL-02-Queries.ipynb | SPARQL-03-Benchmarking | GREMLIN-01-Setup.ipynb | GREMLIN-02-Queries.ipynb | GREMLIN-03-Benchmarking | conda_python3
```

Setting up the RDF/SPARQL Neptune Database

This jupyter notebook is the first of a series of notebooks for Neptune SPARQL performance tuning. The notebook builds upon an RDF version of the AirRoutes dataset, which models the world airline route network.

The tutorial consists of three jupyter notebooks (designed to be executed sequentially):

- (1) SPARQL-01-Setup.ipynb (this notebook):
=> validation of infrastructure, loading of the AirRoutes dataset, and simple queries to validate the data load
- (2) SPARQL-02-Queries.ipynb:
=> contains sample SPARQL queries and demonstrates how to use Neptune's SPARQL explain feature and query hints to understand and tune SPARQL queries
- (3) SPARQL-03-Benchmarking.ipynb:
=> contains small python programs to assess and tune latency & throughput for the sample queries discussed in the previous notebook

In some cases, cells in the notebooks may build upon each other, so it is recommended to execute them in order. Executing a cell can be done via the play button at the top. This is an interactive environment, so you encouraged to edit and replay cells, for instance to test modified versions of the queries and understand the effect of modifying parameters.

```
[1]: %bash
# The commands below may use the following environment variables:
echo ""
echo "Summary of the Neptune environment:"
echo "-----"
echo "AWS_REGION: $AWS_REGION"
echo ""
echo "NEPTUNE_CLUSTER_PORT: $NEPTUNE_CLUSTER_PORT"
echo ""
echo "NEPTUNE_CLUSTER_ENDPOINT (aka WRITER):"
echo "> $NEPTUNE_CLUSTER_ENDPOINT"
echo ""
echo "NEPTUNE_CLUSTER_READER_ENDPOINT:"
echo "> $NEPTUNE_READER_ENDPOINT"
echo ""
echo "NEPTUNE_LOAD_FROM_S3_ROLE_ARN:"
echo "> $NEPTUNE_LOAD_FROM_S3_ROLE_ARN"
echo ""
echo ""=> you can use these environment variables in your jupyter scripts to interact with Neptune"

Summary of the Neptune environment:
-----
AWS_REGION: eu-west-1

NEPTUNE_CLUSTER_PORT: 8182

NEPTUNE_CLUSTER_ENDPOINT (aka WRITER):
> neptunedbcluster-uwnnhkyiv8zt.cluster-c8qanmx04fp.eu-west-1.neptune.amazonaws.com

NEPTUNE_CLUSTER_READER_ENDPOINT:
> neptunedbcluster-uwnnhkyiv8zt.cluster-ro-c8qanmx04fp.eu-west-1.neptune.amazonaws.com

NEPTUNE_LOAD_FROM_S3_ROLE_ARN:
> arn:aws:iam::081588914271:role/Neptune-Performance-Tutorial-NeptuneLoadFromS3Role-15U9PHIFZ2E55

=> you can use these environment variables in your jupyter scripts to interact with Neptune
```

1. Status Check

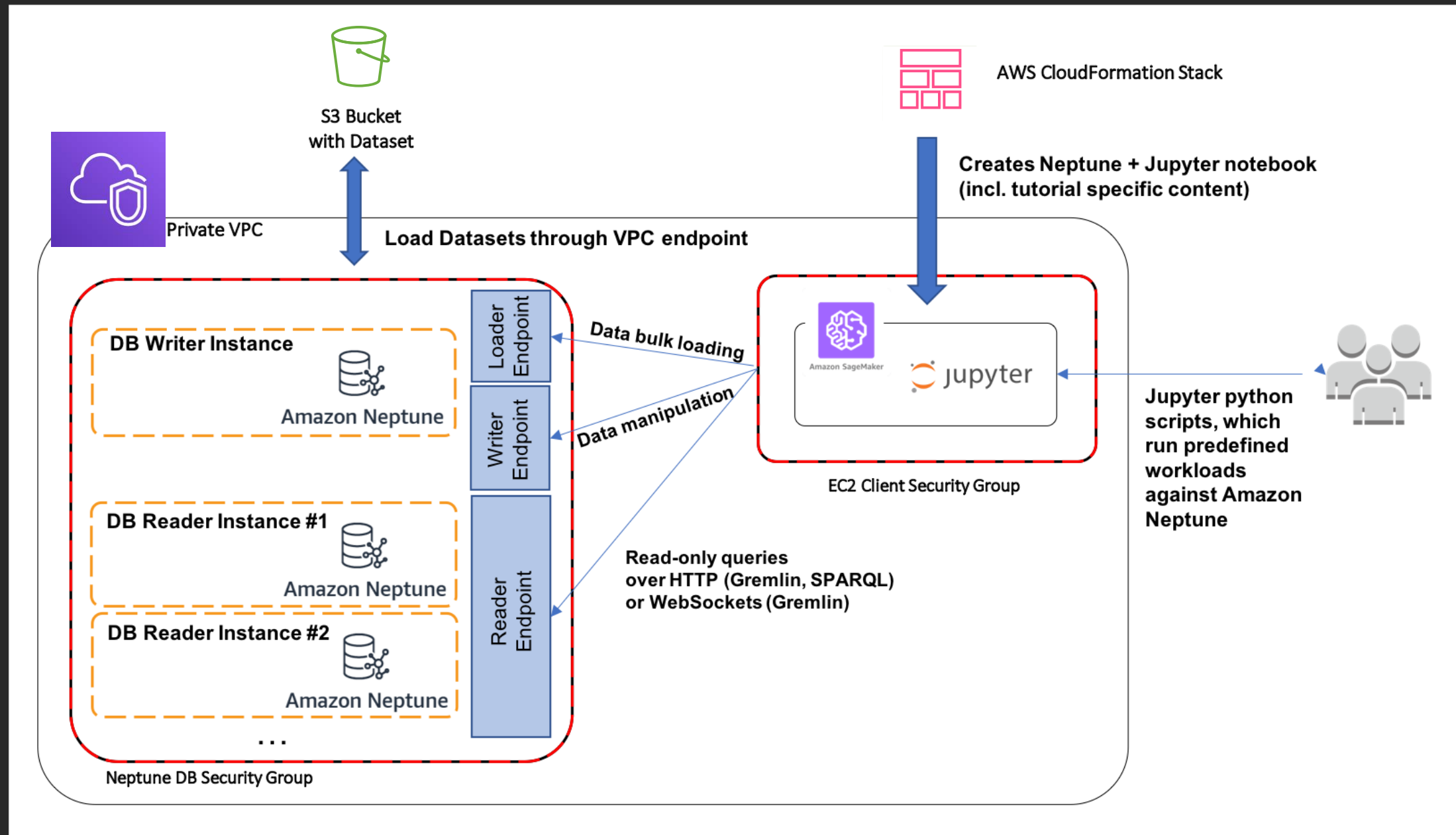
```
[2]: %bash
# Check the overall cluster status
curl -s http://$NEPTUNE_CLUSTER_ENDPOINT:$NEPTUNE_CLUSTER_PORT/status

{"status":"healthy","startTime":"Thu Oct 10 00:54:58 UTC 2019","dbEngineVersion":"1.0.1.0.200457.0","role":"writer"}
```

```
[3]: %bash
# Check the SPARQL endpoint status
curl -s http://$NEPTUNE_CLUSTER_ENDPOINT:$NEPTUNE_CLUSTER_PORT/sparql/status

{"acceptedQueryCount":0,"runningQueryCount":0,"queries":[]}
```

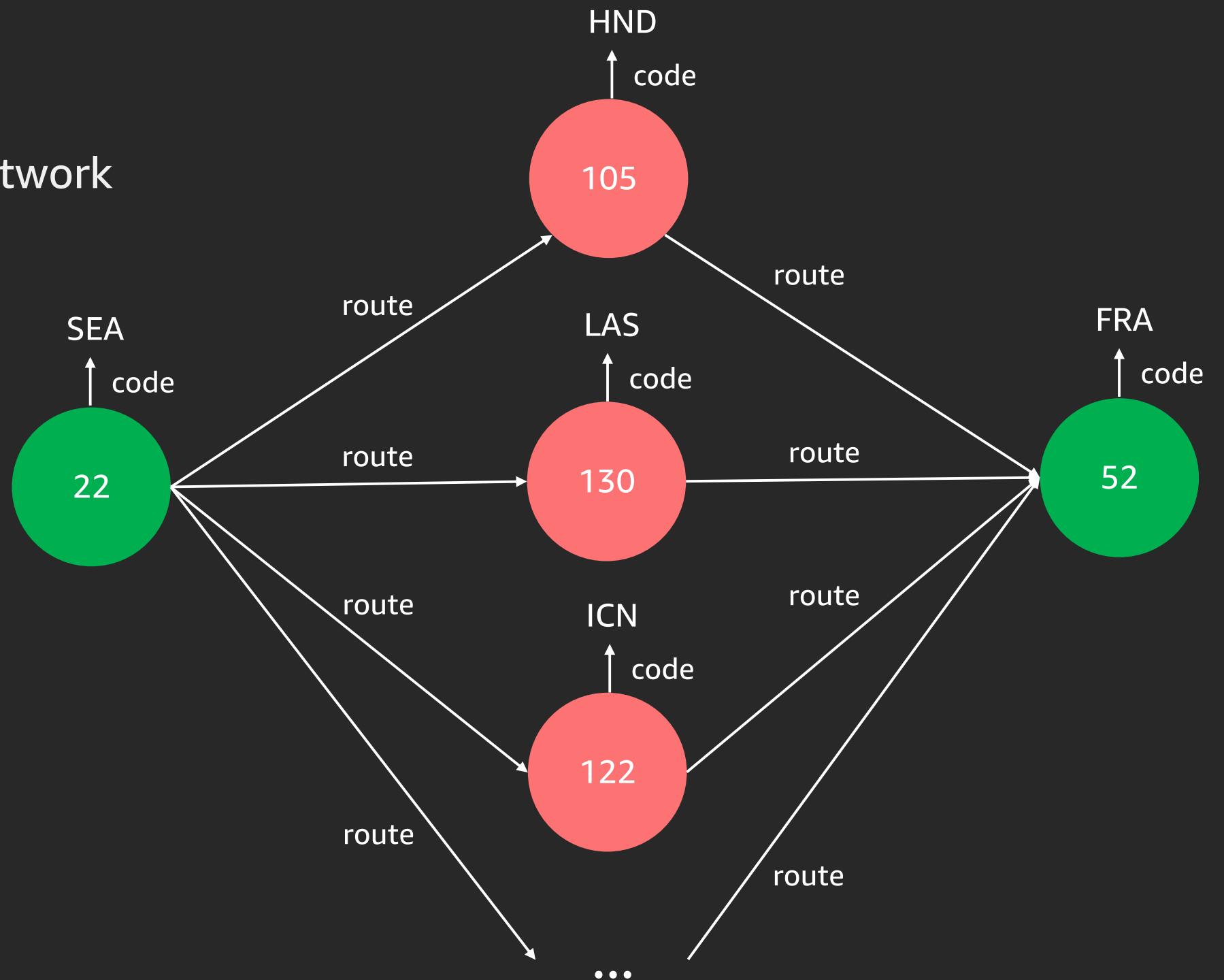
Architecture for hands-on experiments



Query optimization and performance tuning

Air routes dataset

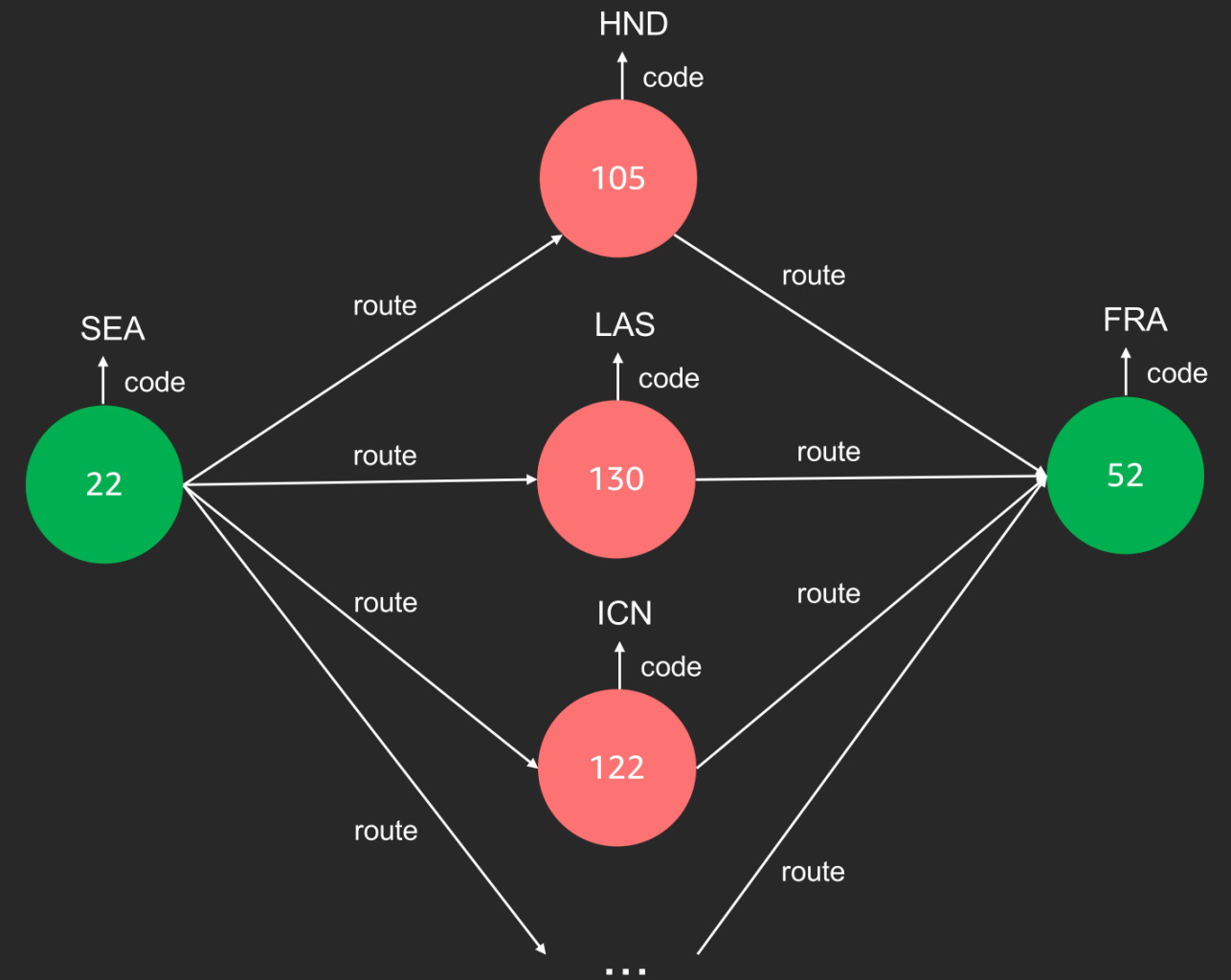
- Models the world's airline route network
- Queries operating over the airport connectivity graph
- Sample queries
 - Given
 - Source and target airport
 - Find
 - All one-stop connections



Sample query: Gremlin

Gremlin

```
g.V()           // start out with all vertices
  .has('code','SEA') // select vertices having code = 'SEA'
  .out('route')     // follow 'route' edge
  .as('via')        // save node in variable 'via'
  .out('route')     // follow 'route' edge again
  .has('code','FRA') // assert we ended up in FRA
  .select('via')    // jump back to the via airport
  .values('code')   // select airport code
```



Sample query: SPARQL

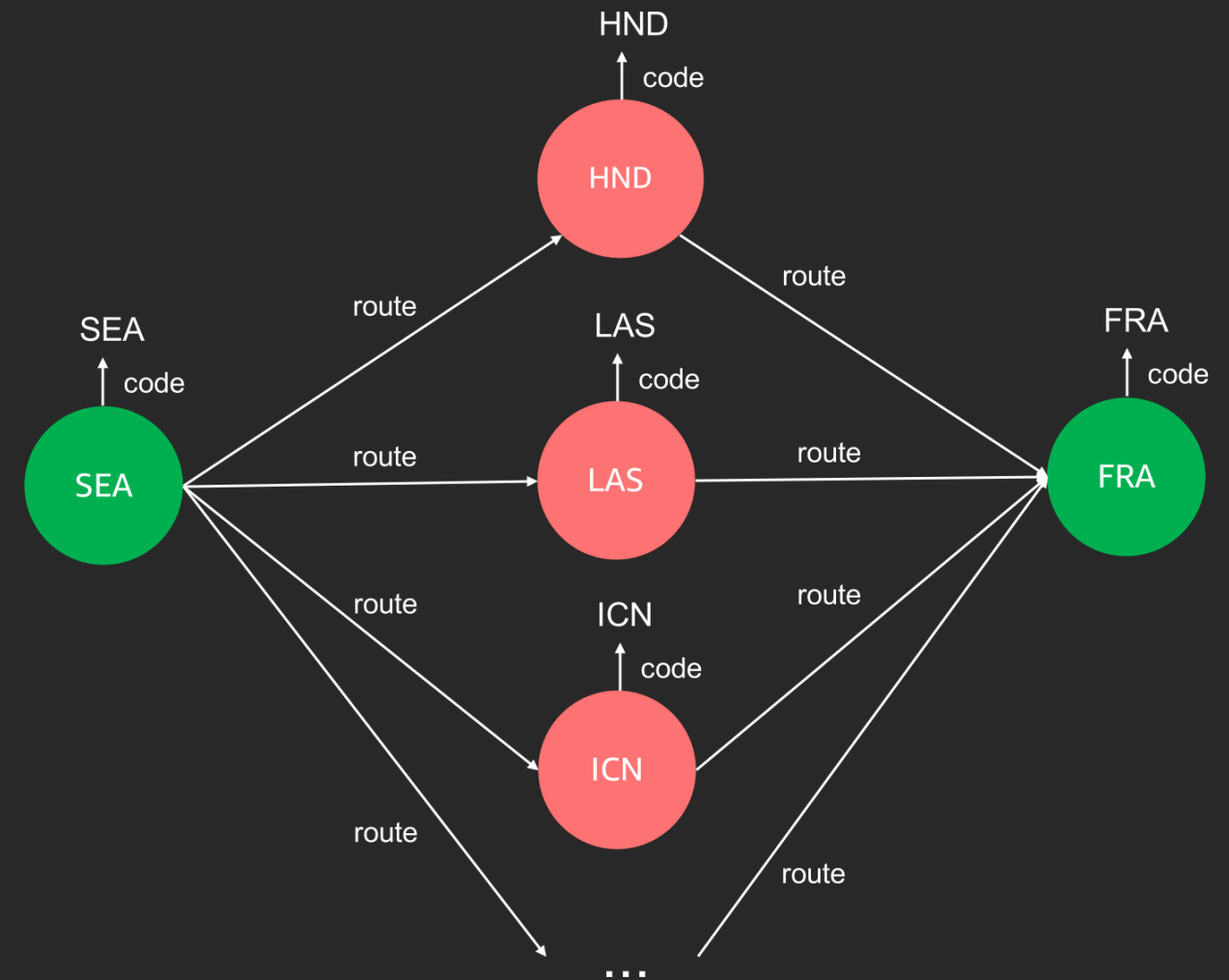
SPARQL

PREFIX airport: <http://kelvinlawrence.net/air-routes/resource/airport/>

PREFIX edge: <http://kelvinlawrence.net/air-routes/objectProperty/>

PREFIX prop: <http://kelvinlawrence.net/air-routes/datatypeProperty/>

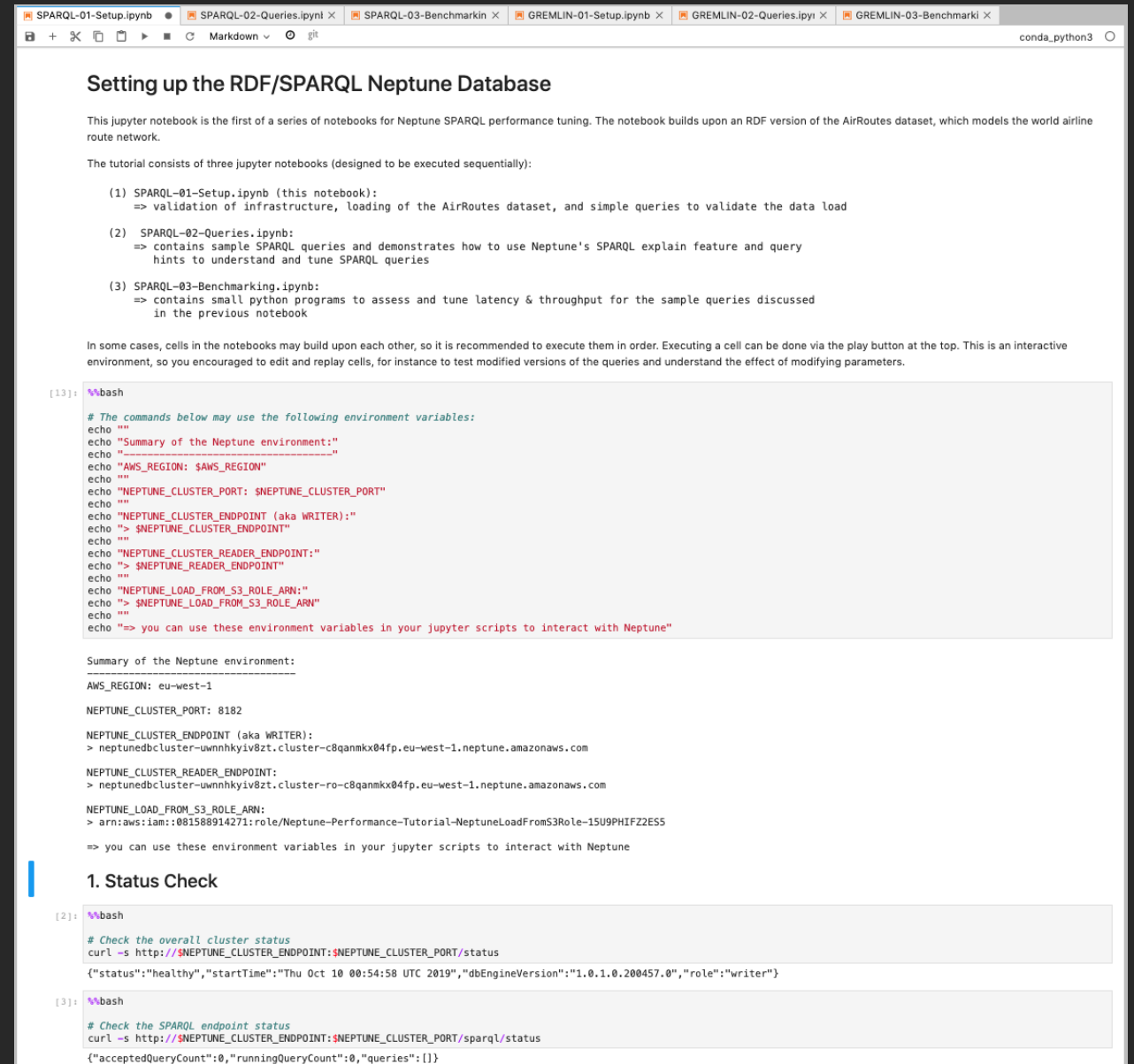
```
SELECT ?via ?viaCode WHERE {  
    airport:SEA edge:route ?via .  
    ?via prop:code ?viaCode .  
    ?via edge:route airport:FRA .  
}
```



Hands on!

Run the experiments in your Jupyter notebook to learn about:

- Running, understanding, and tuning queries
- Performance monitoring via CloudWatch
- Measuring latency and throughput at client side
- Scaling throughput



The screenshot shows a Jupyter notebook interface with multiple tabs at the top: SPARQL-01-Setup.ipynb, SPARQL-02-Queries.ipynb, SPARQL-03-Benchmarking, GREMLIN-01-Setup.ipynb, GREMLIN-02-Queries.ipynb, and GREMLIN-03-Benchmarking. The active notebook is 'SPARQL-01-Setup.ipynb'.

Setting up the RDF/SPARQL Neptune Database

This jupyter notebook is the first of a series of notebooks for Neptune SPARQL performance tuning. The notebook builds upon an RDF version of the AirRoutes dataset, which models the world airline route network.

The tutorial consists of three jupyter notebooks (designed to be executed sequentially):

- (1) SPARQL-01-Setup.ipynb (this notebook):
=> validation of infrastructure, loading of the AirRoutes dataset, and simple queries to validate the data load
- (2) SPARQL-02-Queries.ipynb:
=> contains sample SPARQL queries and demonstrates how to use Neptune's SPARQL explain feature and query hints to understand and tune SPARQL queries
- (3) SPARQL-03-Benchmarking.ipynb:
=> contains small python programs to assess and tune latency & throughput for the sample queries discussed in the previous notebook

In some cases, cells in the notebooks may build upon each other, so it is recommended to execute them in order. Executing a cell can be done via the play button at the top. This is an interactive environment, so you encouraged to edit and replay cells, for instance to test modified versions of the queries and understand the effect of modifying parameters.

```
[13]: %bash
# The commands below may use the following environment variables:
echo ""
echo "Summary of the Neptune environment:"
echo "-----"
echo "AWS_REGION: $AWS_REGION"
echo ""
echo "NEPTUNE_CLUSTER_PORT: $NEPTUNE_CLUSTER_PORT"
echo ""
echo "NEPTUNE_CLUSTER_ENDPOINT (aka WRITER):"
echo "> $NEPTUNE_CLUSTER_ENDPOINT"
echo ""
echo "NEPTUNE_CLUSTER_READER_ENDPOINT:"
echo "> $NEPTUNE_READER_ENDPOINT"
echo ""
echo "NEPTUNE_LOAD_FROM_S3_ROLE_ARN:"
echo "> $NEPTUNE_LOAD_FROM_S3_ROLE_ARN"
echo ""
echo ""
echo "=> you can use these environment variables in your jupyter scripts to interact with Neptune"
```

Summary of the Neptune environment:

AWS_REGION: eu-west-1

NEPTUNE_CLUSTER_PORT: 8182

NEPTUNE_CLUSTER_ENDPOINT (aka WRITER):
> neptunedbcluster-uwnhkyiv8zt.cluster-c8qanmx04fp.eu-west-1.neptune.amazonaws.com

NEPTUNE_CLUSTER_READER_ENDPOINT:
> neptunedbcluster-uwnhkyiv8zt.cluster-ro-c8qanmx04fp.eu-west-1.neptune.amazonaws.com

NEPTUNE_LOAD_FROM_S3_ROLE_ARN:
> arn:aws:iam::081588914271:role/Neptune-Performance-Tutorial-NeptuneLoadFromS3Role-15U9PHIFZ2E55

=> you can use these environment variables in your jupyter scripts to interact with Neptune

1. Status Check

```
[2]: %bash
# Check the overall cluster status
curl -s http://$NEPTUNE_CLUSTER_ENDPOINT:$NEPTUNE_CLUSTER_PORT/status

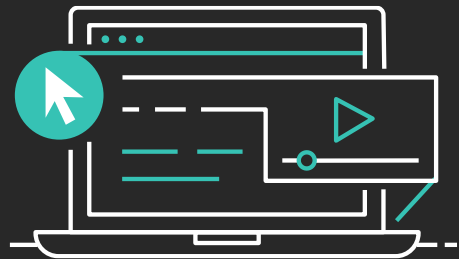
{"status":"healthy","startTime":"Thu Oct 10 00:54:58 UTC 2019","dbEngineVersion":"1.0.1.0.200457.0","role":"writer"}
```

```
[3]: %bash
# Check the SPARQL endpoint status
curl -s http://$NEPTUNE_CLUSTER_ENDPOINT:$NEPTUNE_CLUSTER_PORT/sparql/status

{"acceptedQueryCount":0,"runningQueryCount":0,"queries":[]}
```

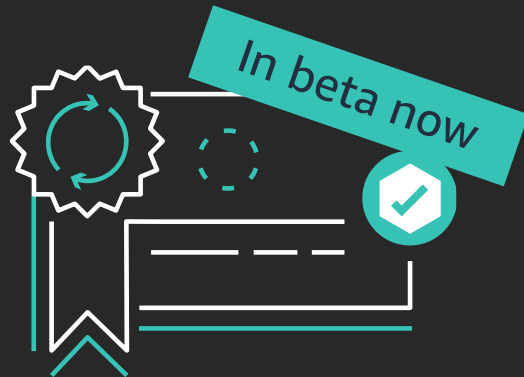
Learn databases with AWS Training and Certification

Resources created by the experts at AWS to help you build and validate database skills



25+ free digital training courses cover topics and services related to databases, including:

- Amazon Aurora
- Amazon Neptune
- Amazon DocumentDB
- Amazon DynamoDB
- Amazon ElastiCache
- Amazon Redshift
- Amazon RDS



Validate expertise with the new **AWS Certified Database - Specialty** beta exam

Visit aws.training

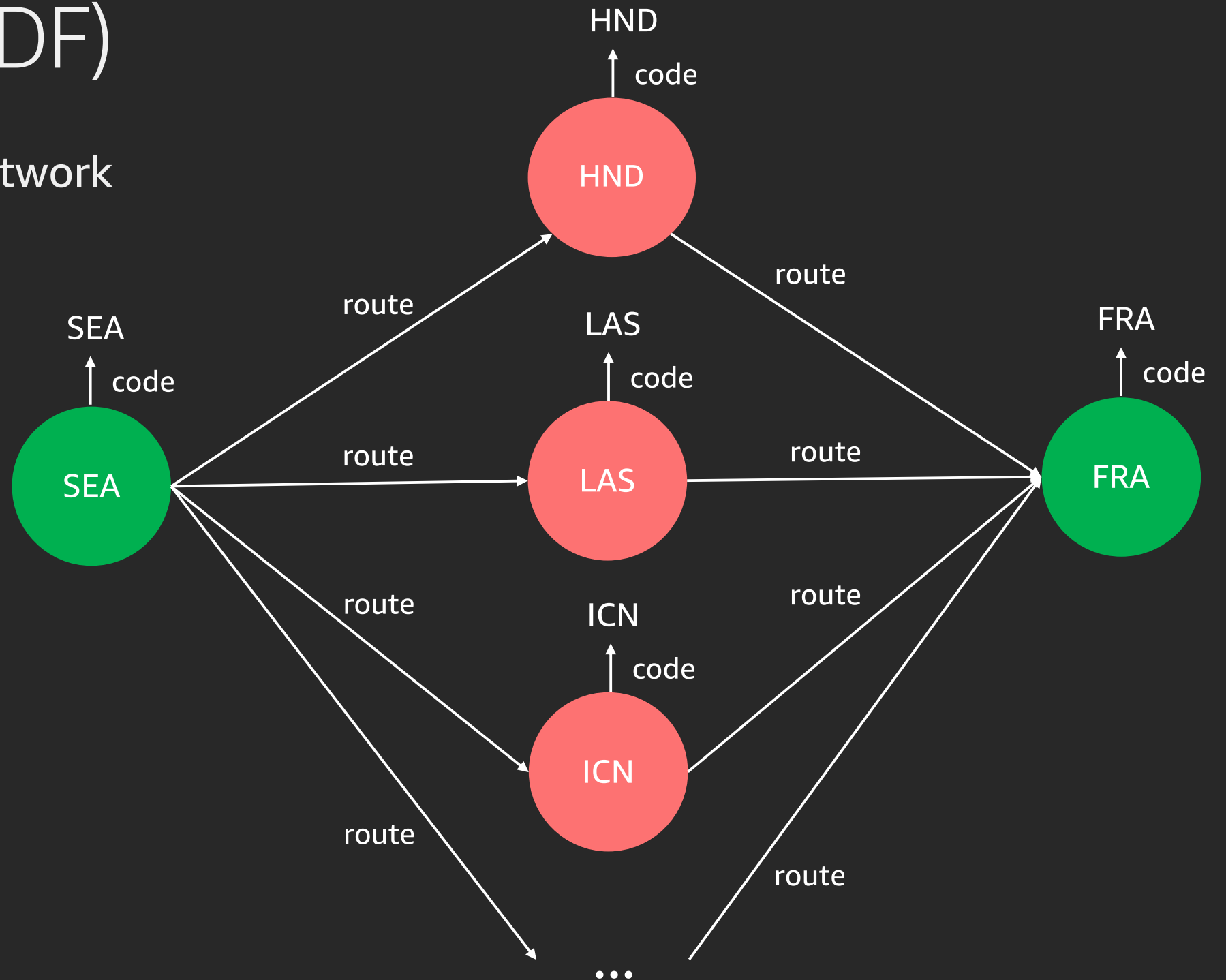
Thank you!

Neptune Reference Customers



Air routes dataset (RDF)

- Models the world's airline route network
- Queries operating over the airport connectivity graph
- Sample queries
 - **Given**
 - Source and target airport
 - **Find**
 - All one-stop connections





Please complete the session
survey in the mobile app.