Amazon Neptune

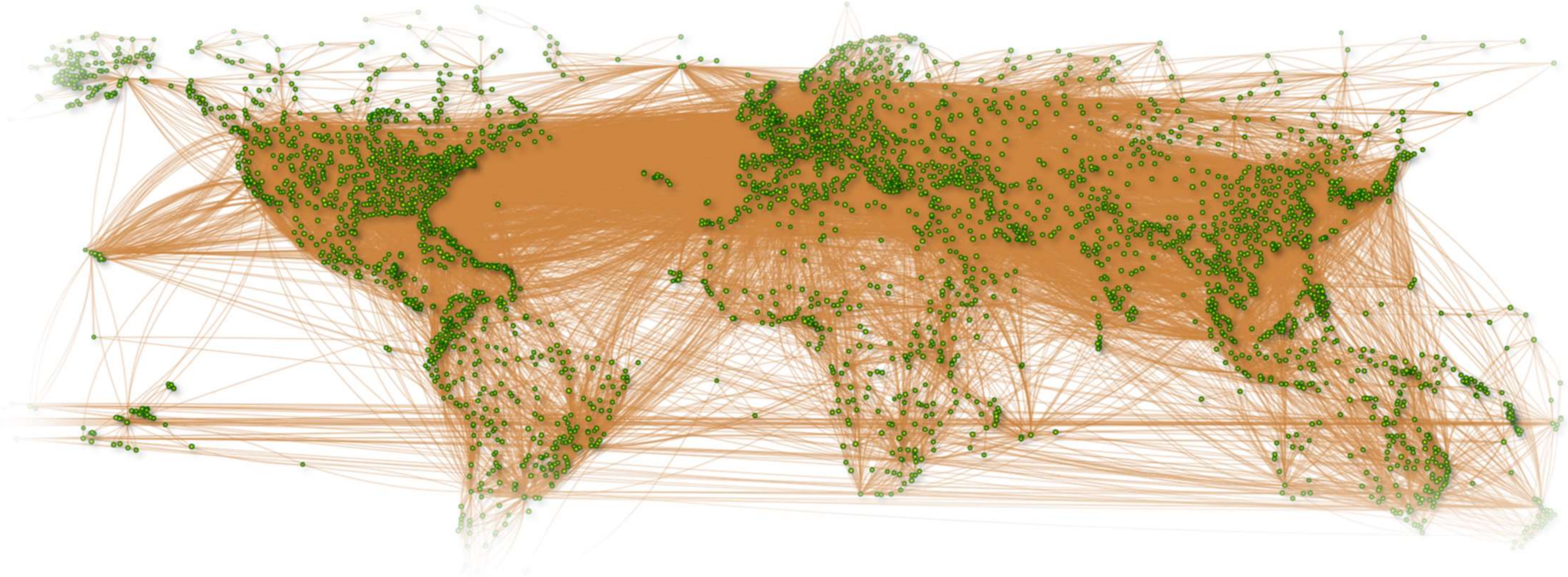
Fast, fully managed graph database service

Karthik Bharathy
Product Leader, Neptune

Agenda

- Graph and its use cases
- Amazon Neptune overview
- Architecture
- Graph Data Model
- Capabilities
- Resources

Graphs are all around us



Why graph?

Relationships enable new applications. Explore connections, paths and patterns in connected data

Knowledge Graphs

Fraud Detection

Recommendations

Social Networking

Life Sciences

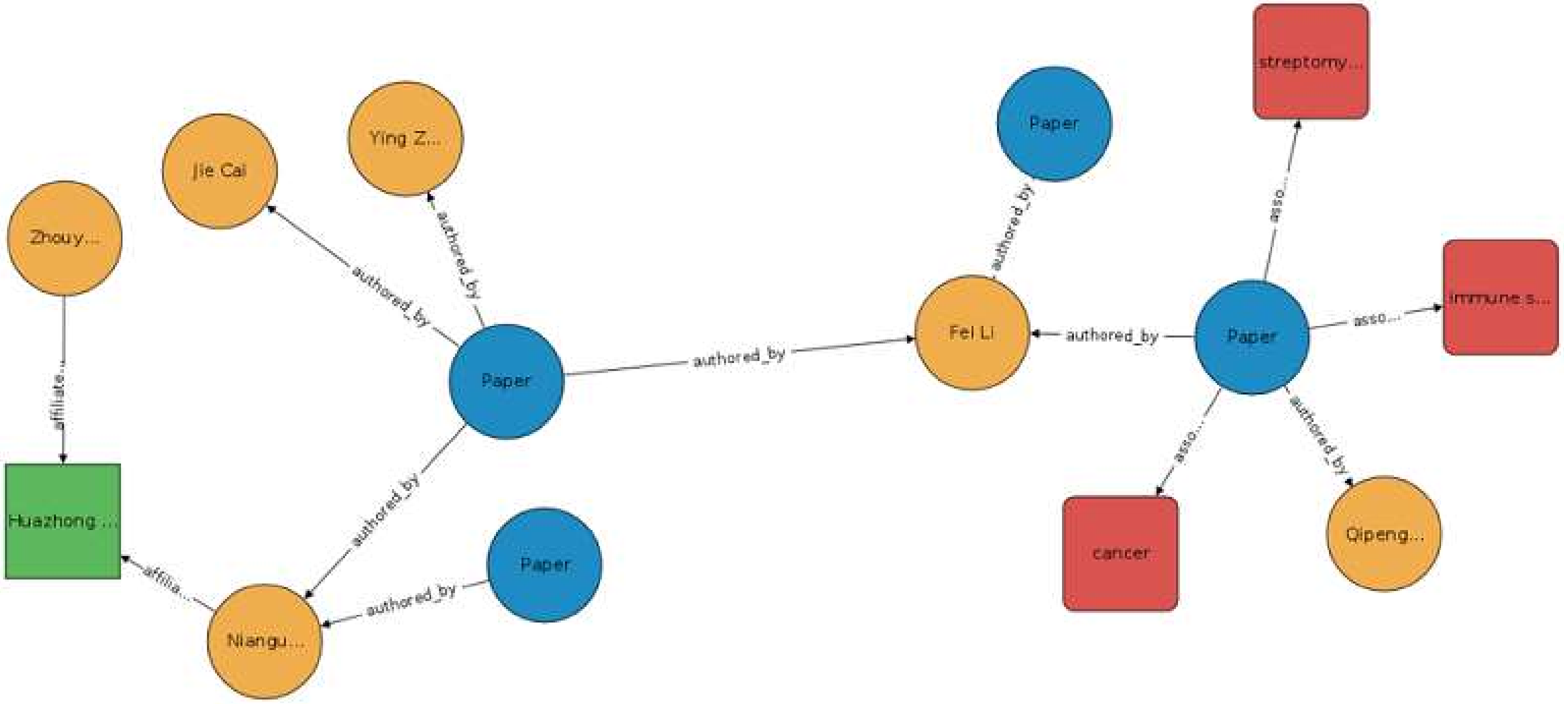
Network & IT Operations

Connected Data Queries

Navigate (variably) connected structure

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

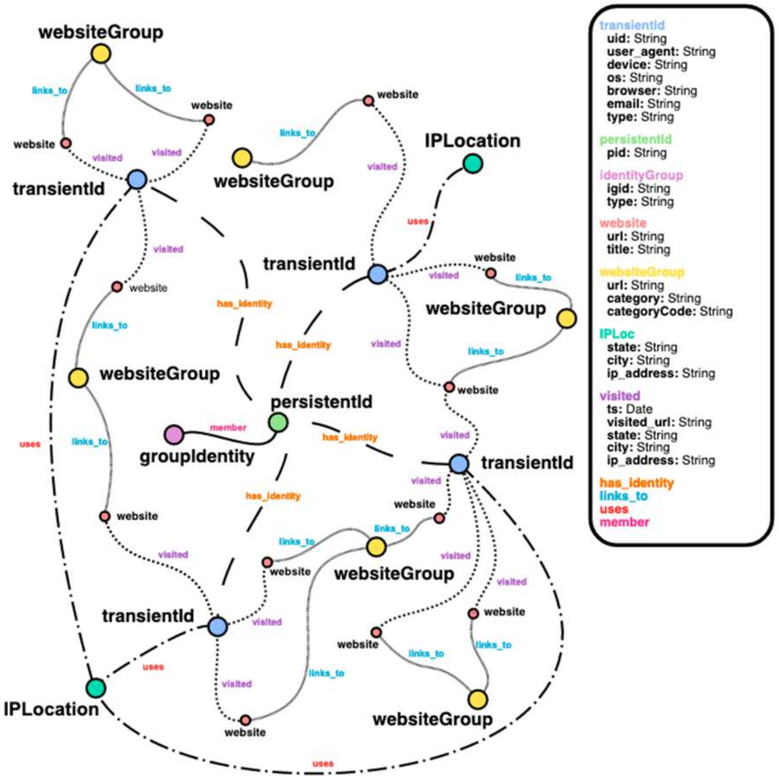
Knowledge Graph



Brings context and semantic meaning by **linked** entities and events

https://aws.amazon.com/blogs/apn/exploring-knowledge-graphs-on-amazon-neptune-using-metaphactory/
https://aws.amazon.com/blogs/database/building-and-querying-the-aws-covid-19-knowledge-graph/

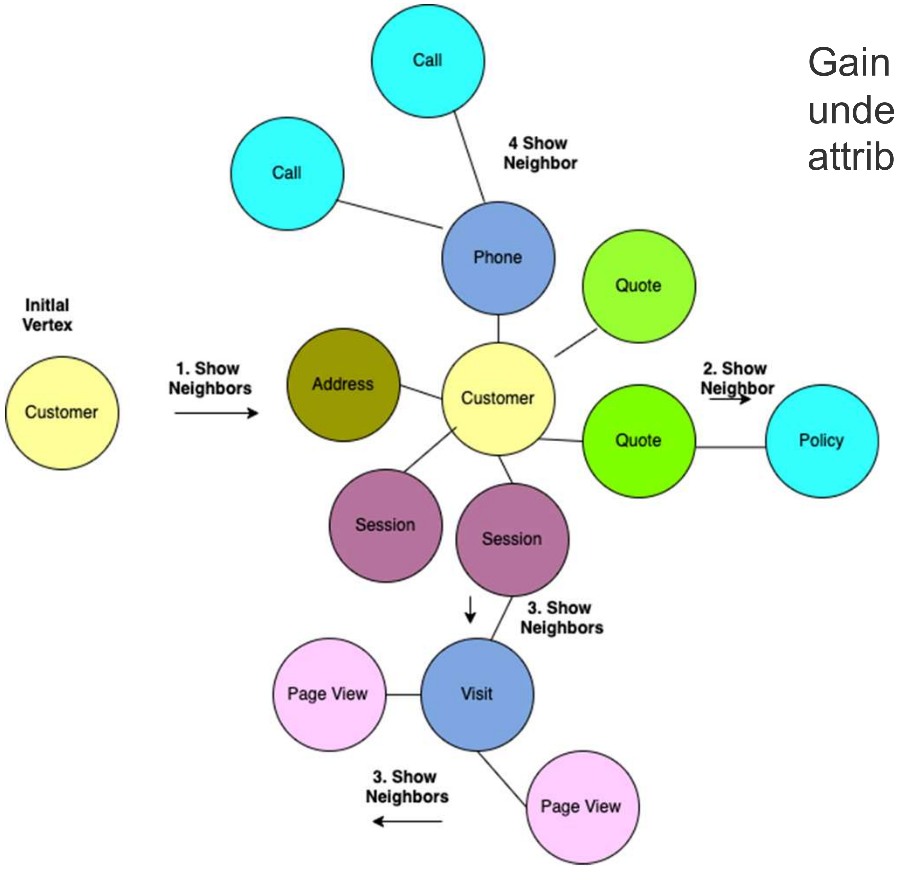
Identity Graph



A single unified view of customers and prospects based on their **interactions** with a product or website across a set of devices and identifiers

https://aws.amazon.com/blogs/database/building-a-customer-identity-graph-with-amazon-neptune/

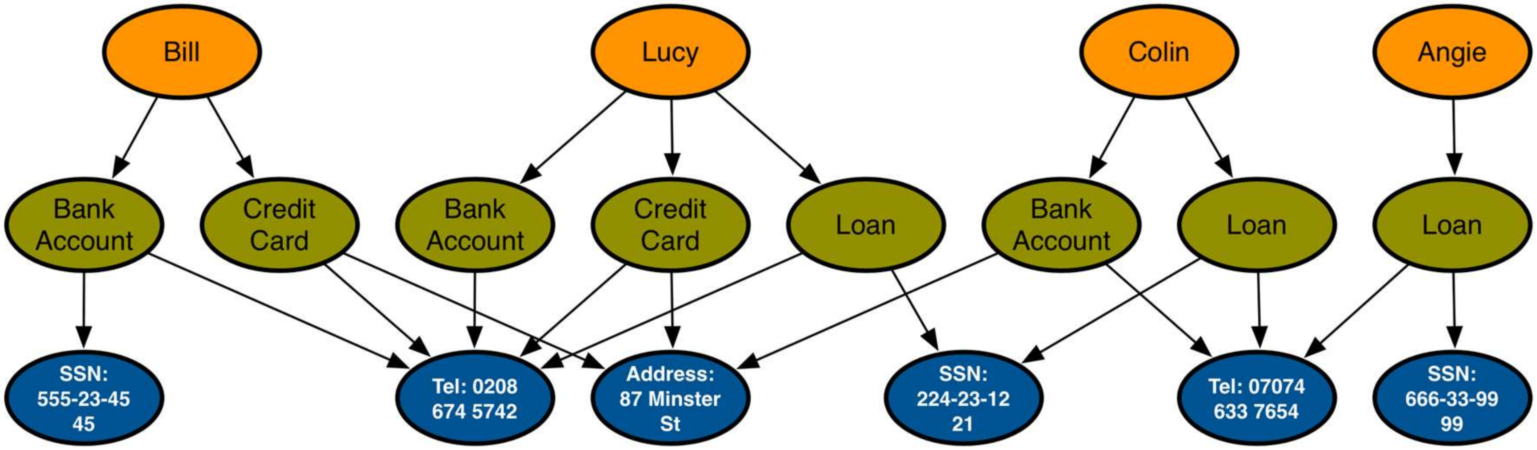
Customer 360



Gain a 360° view of your customers so you can better understand purchasing **patterns** and improve marketing attribution.

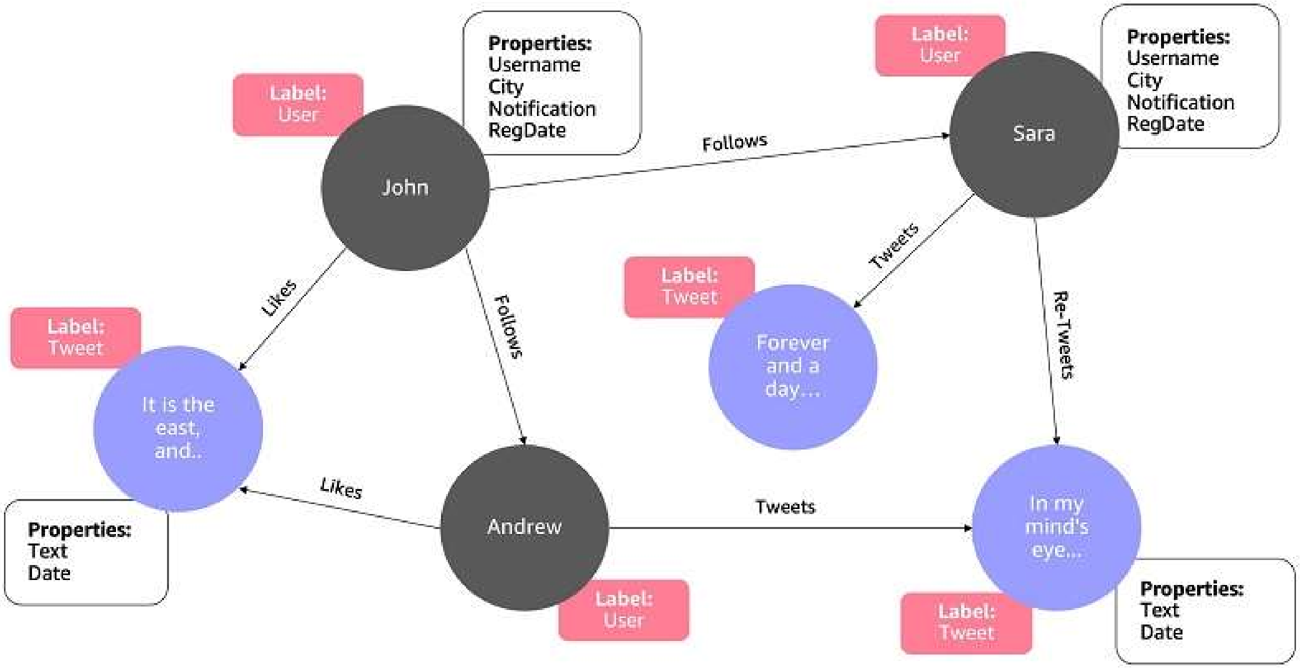
https://aws.amazon.com/blogs/database/building-a-customer-360-knowledge-repository-with-amazon-neptune-and-amazon-redshift/

Fraud Detection



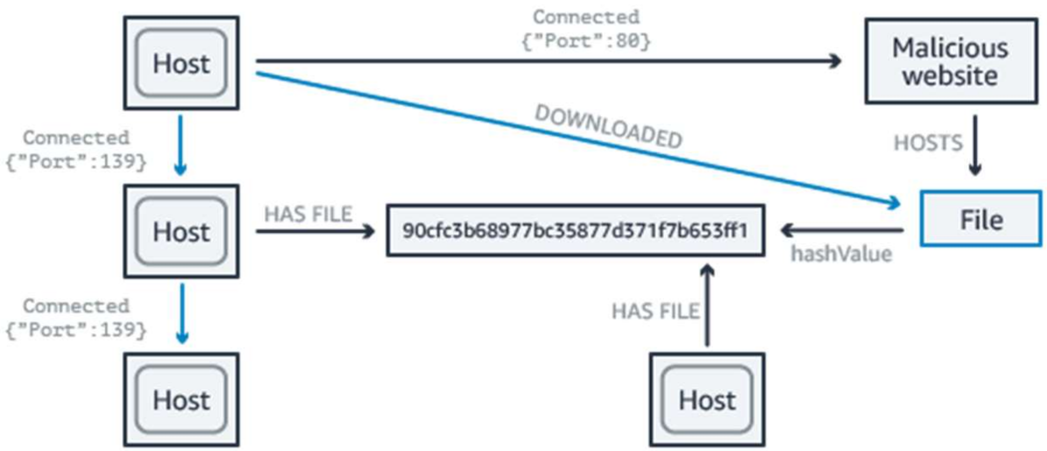
Detect fraud **patterns** - a potential purchaser is using the same email address and credit card, multiple people associated with an email address, or multiple people sharing the same IP address but residing in different physical addresses.

Social Recommendation



Manage **relationships** between information such as customer interests, friends, or purchase history in a graph and quickly query it to make recommendations that are personalized.

Entitlements and access checks



If you detect a malicious file was downloaded on a host, a graph can help you to find the **connections** between the hosts that spread the malicious file, and enable you to trace it to the original host that downloaded it.

Highly connected data and relational database

Unnatural for querying graph

Inefficient graph processing

Rigid schema inflexible for changing data

Challenges of existing graph databases

Difficult to scale

Difficult to maintain high availability

Too expensive

Limited support for open standards

Amazon 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 & W3C RDF graph models

Neptune customers

THOMSON REUTERS™

Finra

SIEMENS

AstraZeneca

LIFEOMIC

amazon alexa

intuit.

FIREEYE™

SAMSUNG

Blackfynn

Uber ATG

NETFLIX

NBCUniversal

HuUUGE

asurion

PaySense

zeta

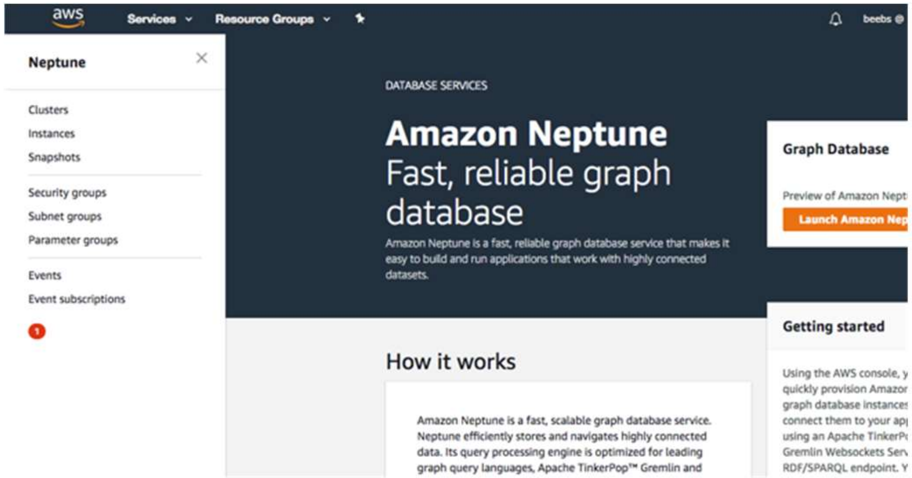
Pearson

MARINUS ANALYTICS

noonum

Neptune General Availability

- Announced on 5/30/2018
- 18 AWS Regions: Americas (N. Virginia, Montréal, Ohio, Oregon), Europe (Frankfurt, Ireland, London, Paris, Stockholm), Middle East (Bahrain), Asia Pacific (Mumbai, Seoul, Singapore, Sydney, Tokyo), AWS China (Ningxia), and AWS GovCloud (US-East) and (US-West)
- Encryption-at-rest with AWS Key Management Service (AWS KMS)
- Encryption-in-transit with TLS 1.2 client connections
- ISO, HIPAA, SOC, PCI/DSS compliance certifications



Amazon Neptune
Fast, reliable graph database

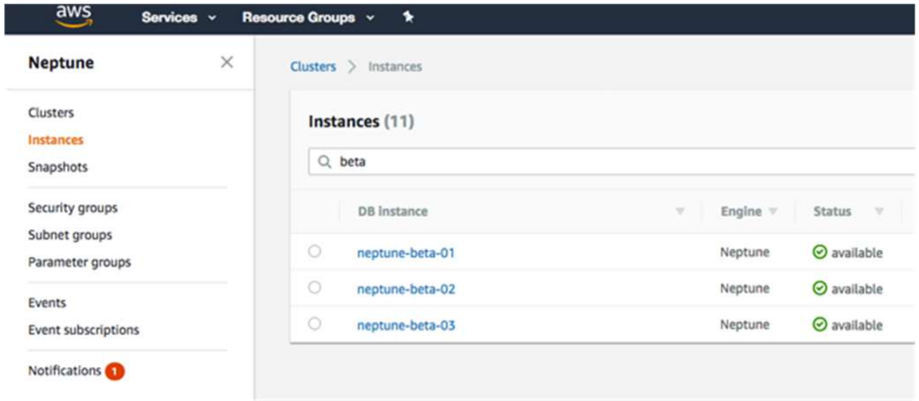
Amazon Neptune is a fast, reliable graph database service that makes it easy to build and run applications that work with highly connected datasets.

Preview of Amazon Nept

How it works

Amazon Neptune is a fast, scalable graph database service. Neptune efficiently stores and navigates highly connected data. Its query processing engine is optimized for leading graph query languages, Apache TinkerPop™ Gremlin and

Using the AWS console, y
quickly provision Amazor
graph database instance:
connect them to your ap|
using an Apache TinkerP
Gremlin Websockets Serv
RDF/SPARQL endpoint. Y



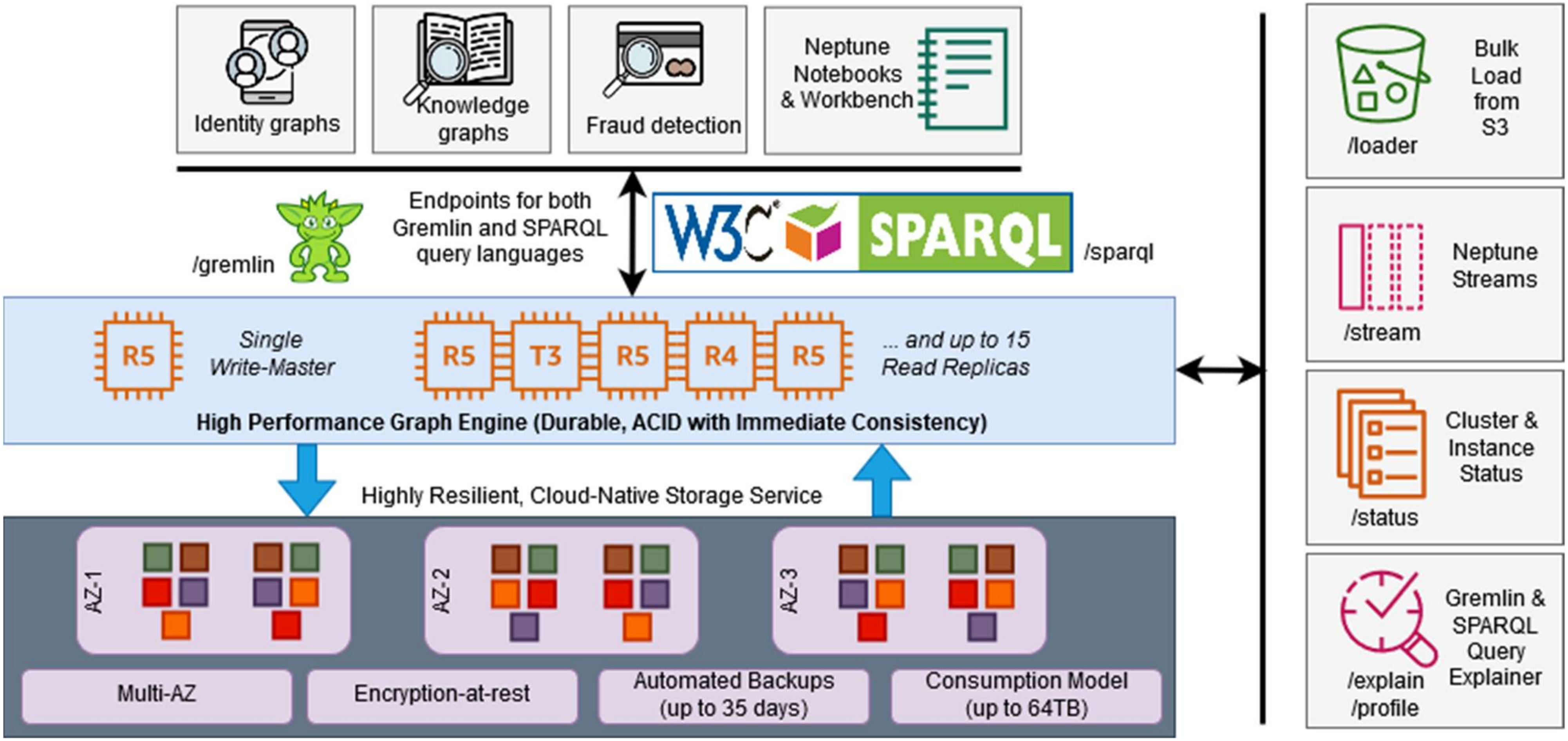
Instances (11)

beta

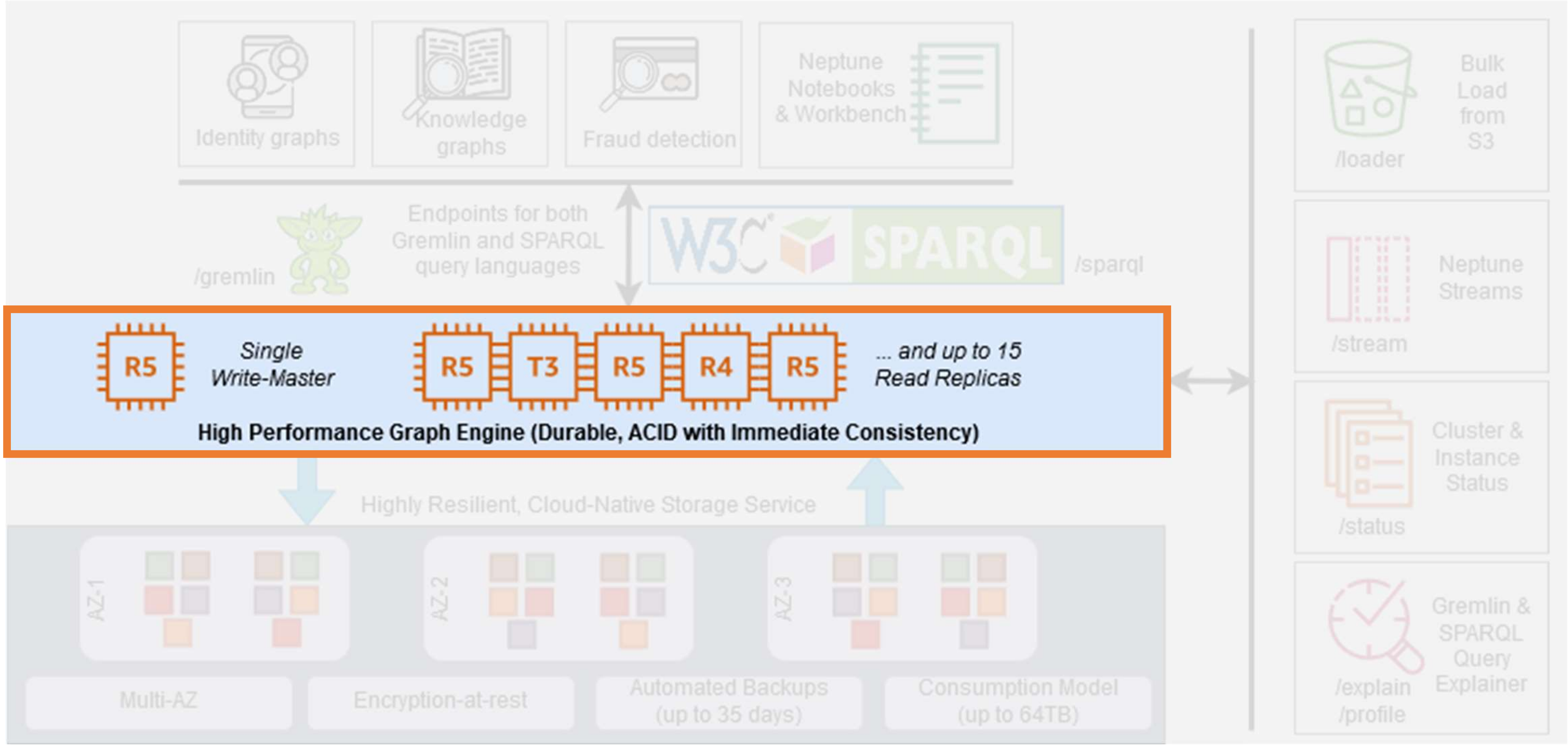
	DB Instance	Engine	Status
○	neptune-beta-01	Neptune	✓ available
○	neptune-beta-02	Neptune	✓ available
○	neptune-beta-03	Neptune	✓ available

Architecture

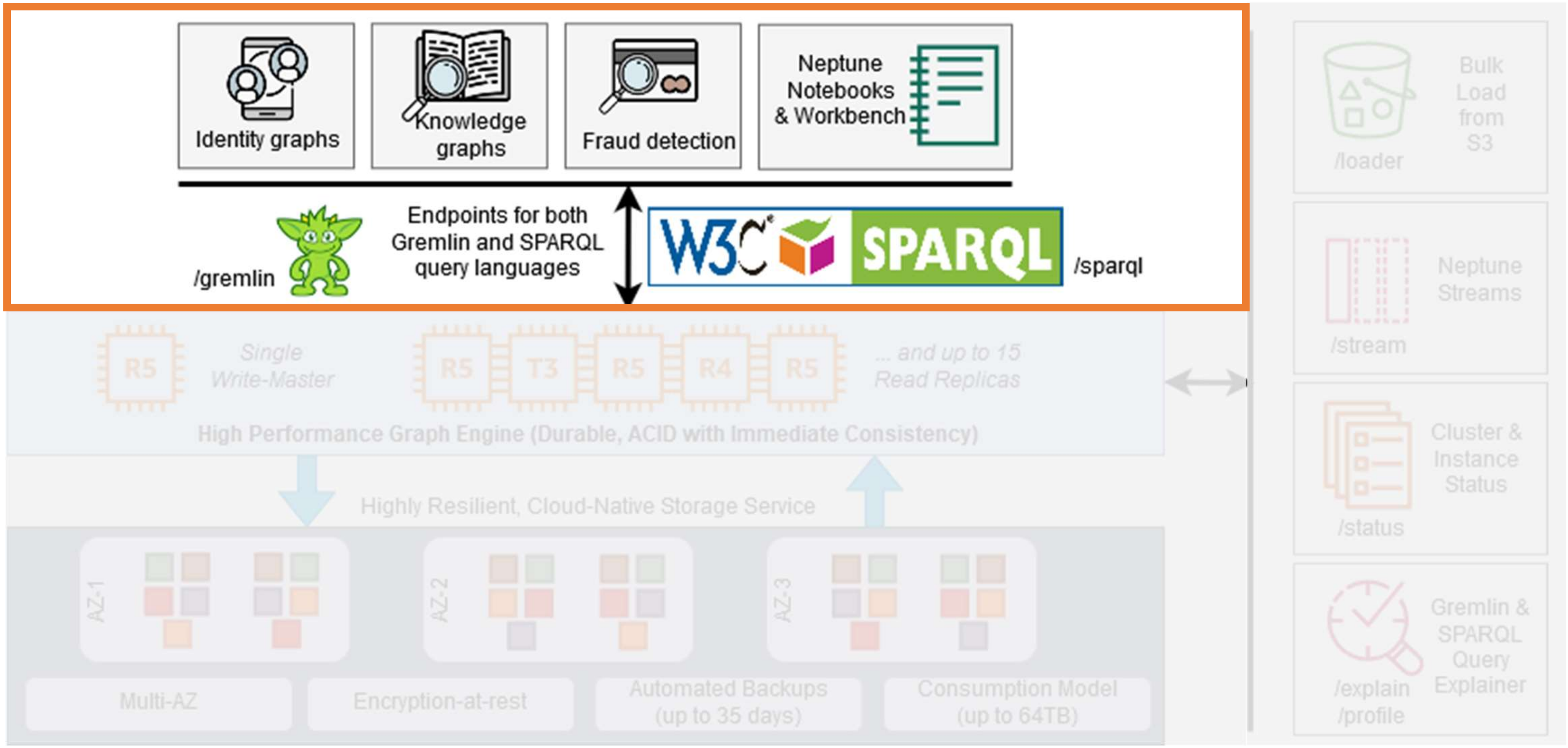
Fast, fully managed graph database service


Identity graphs
Knowledge graphs
Fraud detection
Neptune Notebooks & Workbench
/gremlin
Endpoints for both Gremlin and SPARQL query languages
W3C SPARQL
/sparql
R5
Single Write-Master
R5
T3
R5
R4
R5
... and up to 15 Read Replicas
High Performance Graph Engine (Durable, ACID with Immediate Consistency)
Highly Resilient, Cloud-Native Storage Service
AZ-1
AZ-2
AZ-3
Multi-AZ
Encryption-at-rest
Automated Backups (up to 35 days)
Consumption Model (up to 64TB)
Bulk Load from S3
/loader
Neptune Streams
/stream
Cluster & Instance Status
/status
Gremlin & SPARQL Query Explainer
/explain
/profile

High performance graph engine

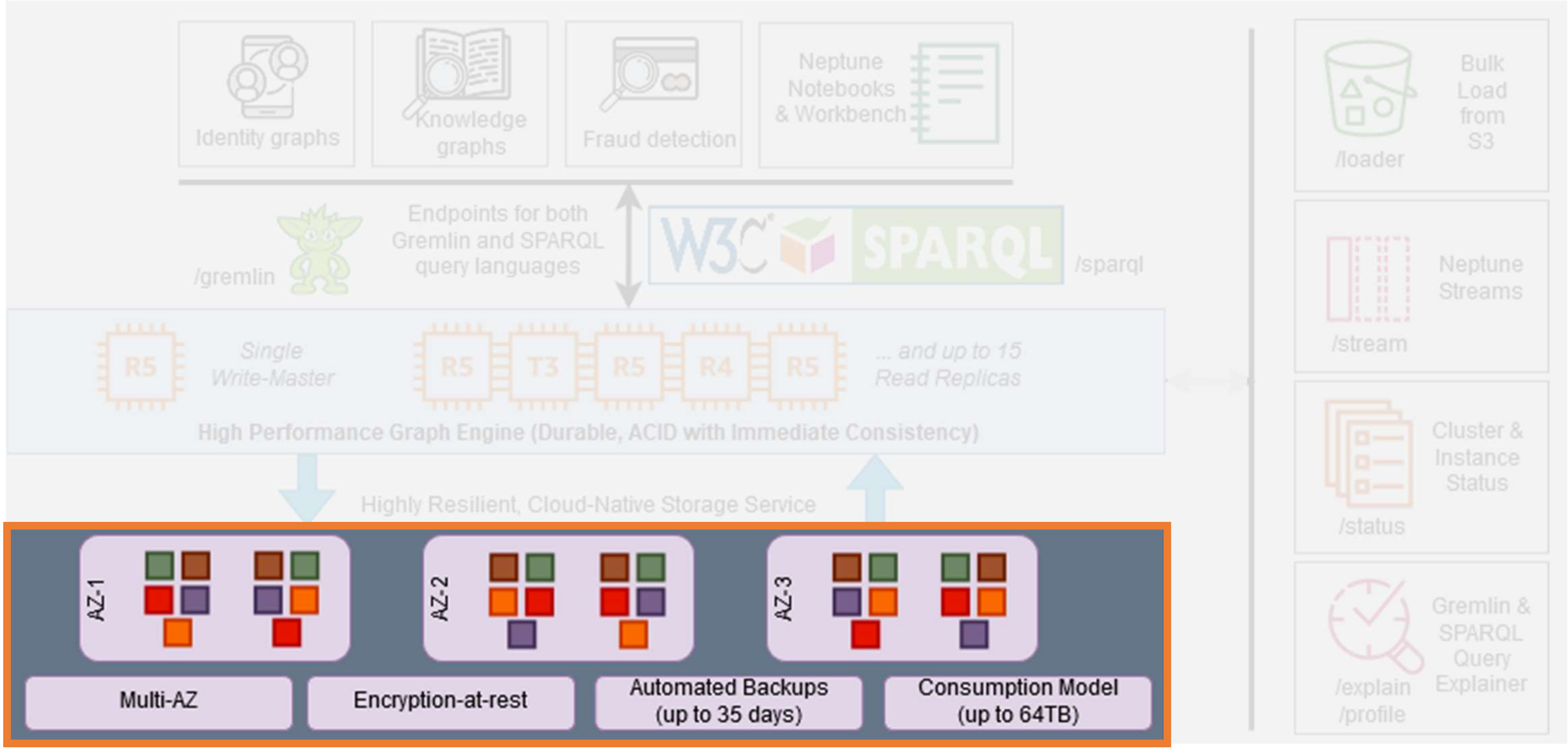

Identity graphs
Knowledge graphs
Fraud detection
Neptune Notebooks & Workbench
/loader
Bulk Load from S3
/gremlin
Endpoints for both Gremlin and SPARQL query languages
W3C SPARQL
/sparql
Neptune Streams
/stream
R5
Single Write-Master
R5
T3
R5
R4
R5
... and up to 15 Read Replicas
High Performance Graph Engine (Durable, ACID with Immediate Consistency)
Cluster & Instance Status
/status
Highly Resilient, Cloud-Native Storage Service
AZ-1
AZ-2
AZ-3
Multi-AZ
Encryption-at-rest
Automated Backups (up to 35 days)
Consumption Model (up to 64TB)
Gremlin & SPARQL Query Explainer
/explain
/profile

Graph models and query languages

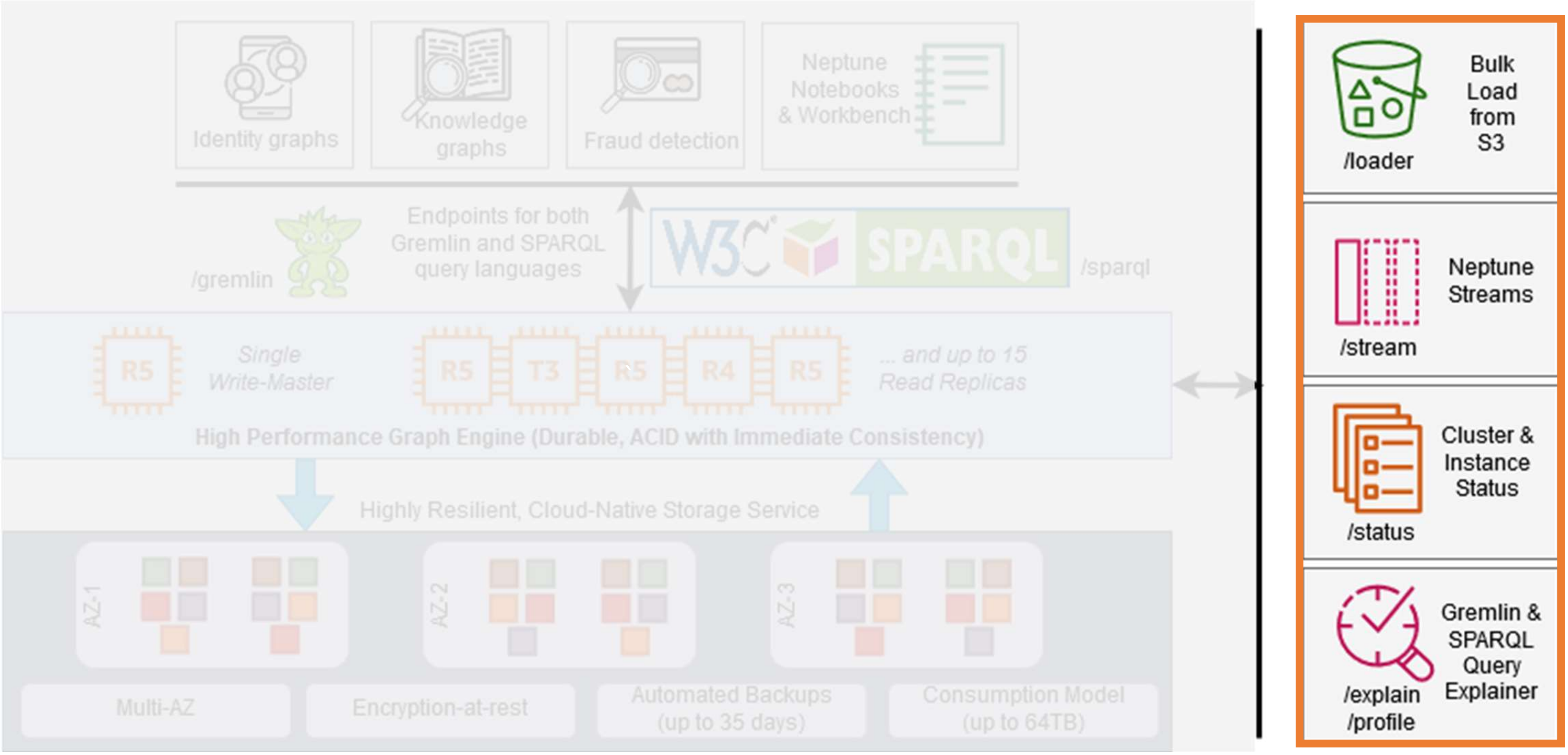


Identity graphs
Knowledge graphs
Fraud detection
Neptune Notebooks & Workbench
/gremlin
Endpoints for both Gremlin and SPARQL query languages
W3C
SPARQL
/sparql
R5
Single Write-Master
R5
T3
R5
R4
R5
... and up to 15 Read Replicas
High Performance Graph Engine (Durable, ACID with Immediate Consistency)
Highly Resilient, Cloud-Native Storage Service
AZ-1
AZ-2
AZ-3
Multi-AZ
Encryption-at-rest
Automated Backups (up to 35 days)
Consumption Model (up to 64TB)
/loader
Bulk Load from S3
/stream
Neptune Streams
/status
Cluster & Instance Status
/explain /profile
Gremlin & SPARQL Query Explainer

Shared storage service - multi AZ upto 64TB

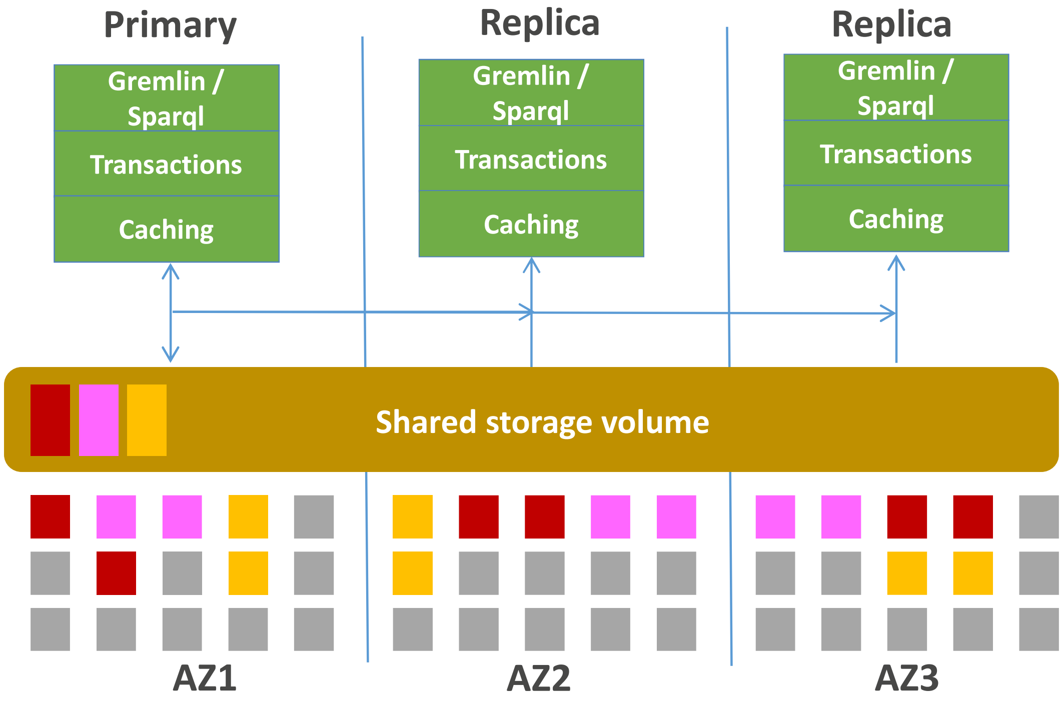

Identity graphs
Knowledge graphs
Fraud detection
Neptune Notebooks & Workbench
Endpoints for both Gremlin and SPARQL query languages
/gremlin
W3C SPARQL
/sparql
R5
Single Write-Master
R5
T3
R5
R4
R5
... and up to 15 Read Replicas
High Performance Graph Engine (Durable, ACID with Immediate Consistency)
Highly Resilient, Cloud-Native Storage Service
AZ-1
AZ-2
AZ-3
Multi-AZ
Encryption-at-rest
Automated Backups (up to 35 days)
Consumption Model (up to 64TB)
Bulk Load from S3
/loader
Neptune Streams
/stream
Cluster & Instance Status
/status
Gremlin & SPARQL Query Explainer
/explain
/profile

Endpoints for Load, Streams, Status, Explain, Profile


Identity graphs
Knowledge graphs
Fraud detection
Neptune Notebooks & Workbench
Endpoints for both Gremlin and SPARQL query languages
/gremlin
W3C SPARQL
/sparql
R5
Single Write-Master
R5
T3
R5
R4
R5
... and up to 15 Read Replicas
High Performance Graph Engine (Durable, ACID with Immediate Consistency)
Highly Resilient, Cloud-Native Storage Service
AZ-1
AZ-2
AZ-3
Multi-AZ
Encryption-at-rest
Automated Backups (up to 35 days)
Consumption Model (up to 64TB)
/loader
Bulk Load from S3
/stream
Neptune Streams
/status
Cluster & Instance Status
/explain
/profile
Gremlin & SPARQL Query Explainer

Neptune: Distributed storage architecture

- Performance, availability, durability
- Scale-out replica architecture
- Shared storage volume with 10 GB segments striped across hundreds of nodes
- Data is replicated 6 times across 3 AZs
- Hotspot rebalance, Fast database recovery
- Log applicator embedded in storage layer



Delivered as a managed service

Read Replicas and High Availability

Performance

- Applications can scale out read traffic across up to 15 read replicas

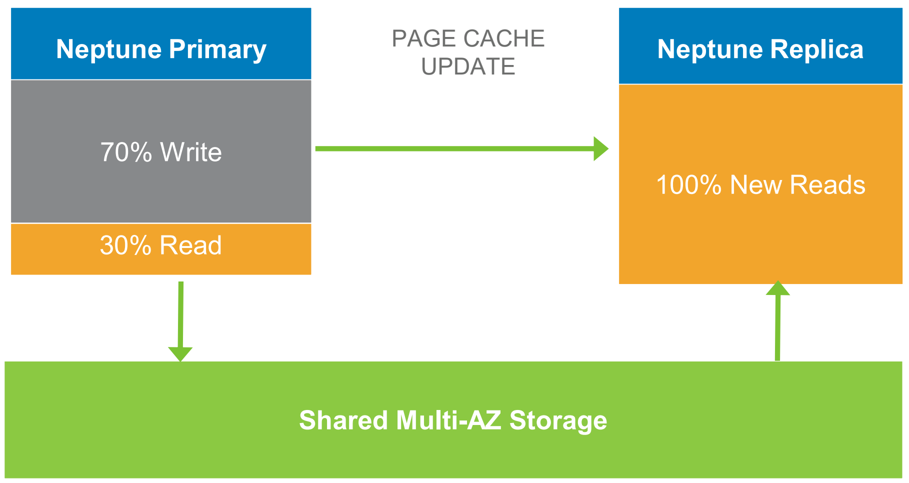
Low Replica Lag

- Typically < 10ms
- Master ships redo logs to replica
- Cached pages have redo applied
- Un-cached pages from shared storage

Availability

- Failing database nodes are automatically detected and replaced
- If primary fails, a replica replaces it (typically < 60s failover time)
- Primary upgrade by forced failover

Amazon Neptune read scaling



Graph Data Model

Leading graph models and frameworks

PROPERTY GRAPH

Open Source Apache TinkerPop™

Gremlin Traversal Language

RESOURCE DESCRIPTION FRAMEWORK (RDF)

W3C Standard

SPARQL Query Language

Many customers want both.

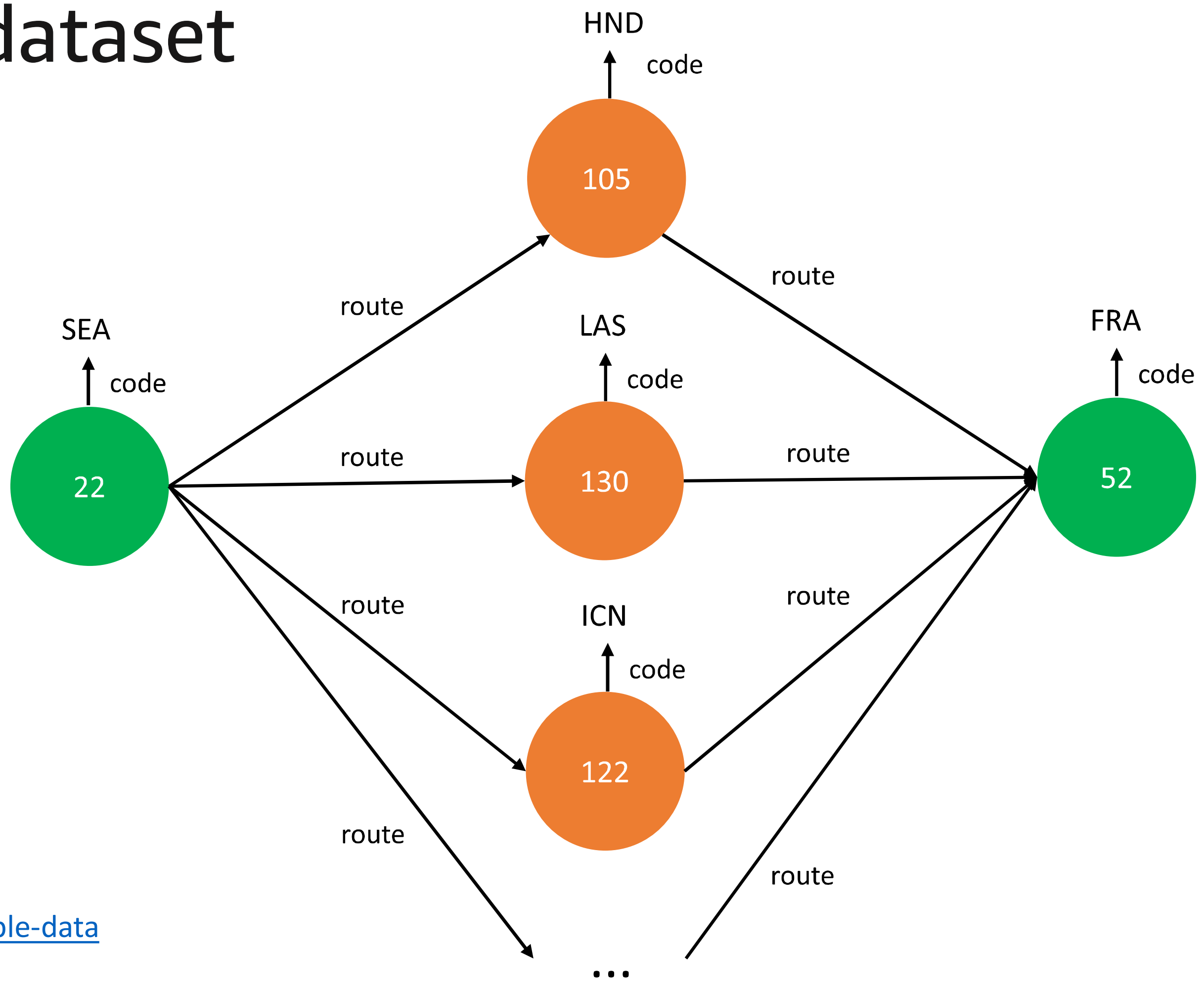
Example: Air routes dataset

- Models the world’s airline route network
- Queries operating over the airport connectivity graph
- Sample query

  Given source and target airports

  Find all one-stop connections

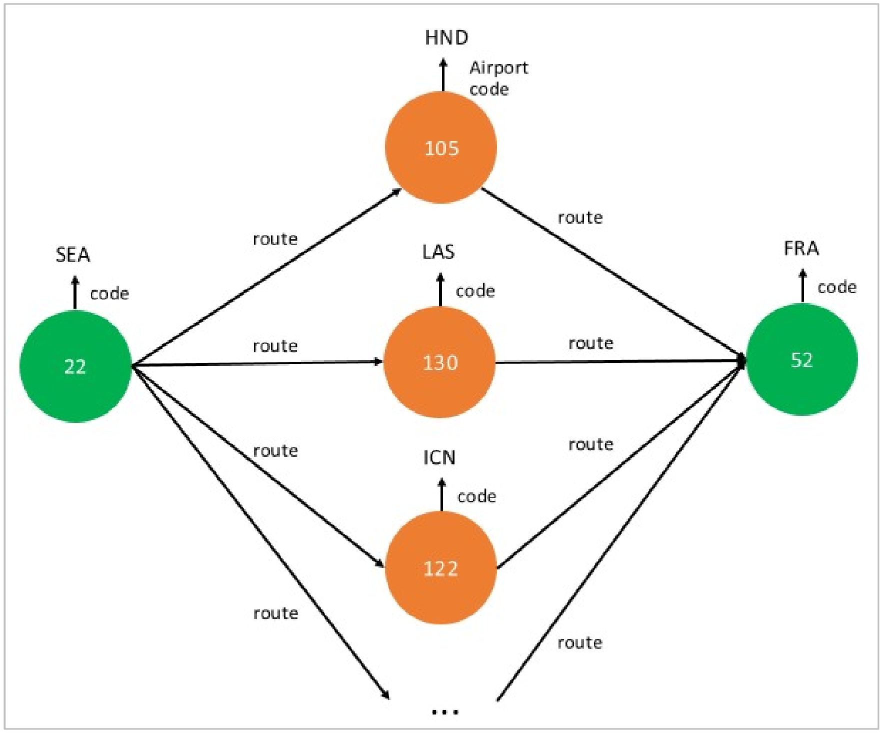
https://github.com/krlawrence/graph/tree/master/sample-data



“Find all of the airport codes for one-stop connections from SEA to FRA”

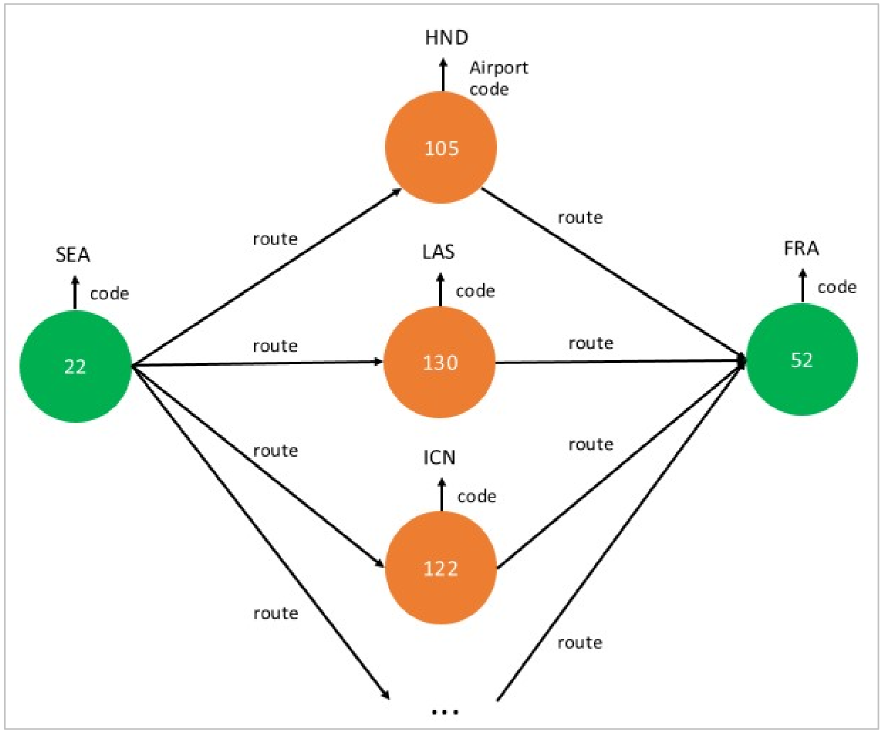
Gremlin

```
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
```



Neptune graph data model

Apache TinkerPop



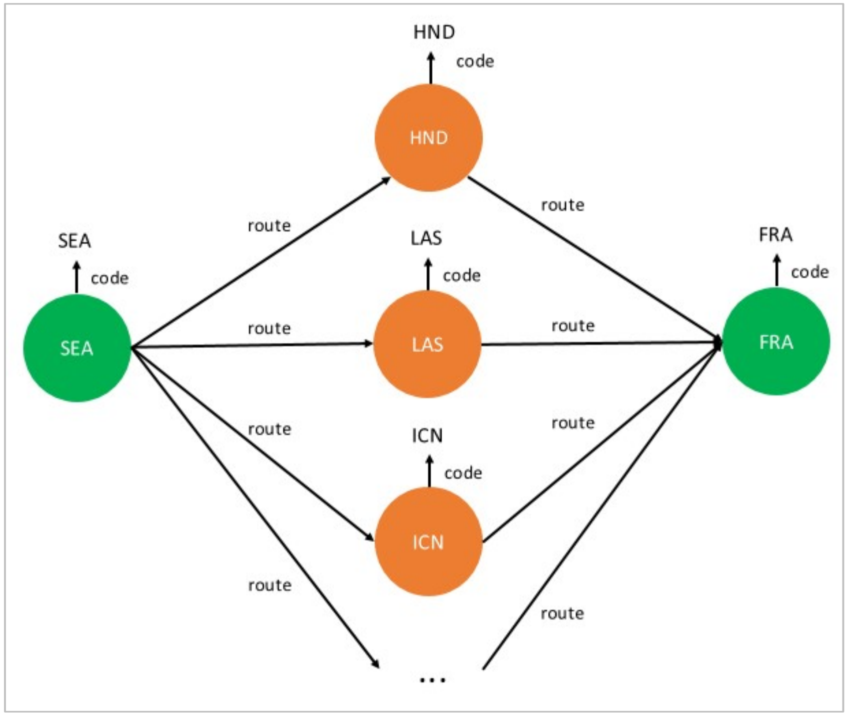
Subject	Predicate	Object	Graph
22	code	"SEA"	default
22	route	105	e1
22	route	130	e2
22	route	122	e3
105	code	"HND"	default
105	route	52	e4
130	code	"LAS"	default
130	route	52	e5
122	code	"ICN"	default
122	route	52	e6
52	code	"FRA"	default

https://docs.aws.amazon.com/neptune/latest/userguide/feature-overview-data-model.html

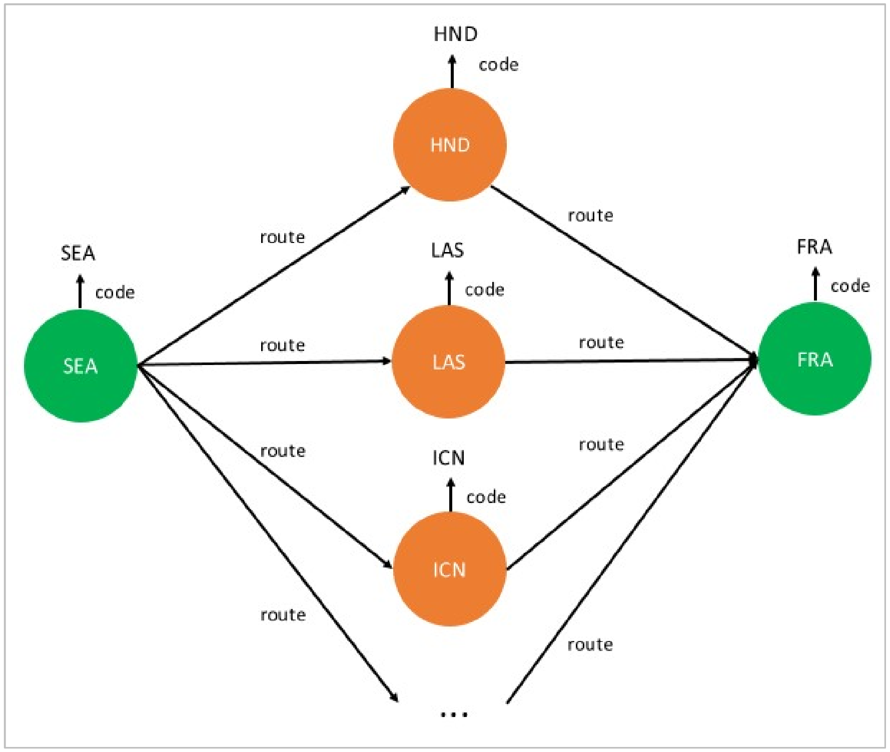
”Find all of the airport codes for one-stop connections from SEA to FRA”

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 ?viaCode WHERE {

    airport:SEA edge:route ?via .

    ?via prop:code ?viaCode .

    ?via edge:route airport:FRA .

}
```



Neptune graph data model



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

Subject	Predicate	Object	Graph
airport:SEA	prop:code	"SEA"	default
airport:SEA	edge:route	airport:HND	resource:1
airport:SEA	edge:route	airport:LAS	resource:2
airport:SEA	edge:route	airport:ICN	resource:3
airport:HND	prop:code	"HND"	default
airport:HND	edge:route	airport:FRA	resource:4
airport:LAS	prop:code	"LAS"	default
airport:LAS	edge:route	airport:FRA	resource:5
airport:ICN	prop:code	"ICN"	default
airport:ICN	edge:route	airport:FRA	resource:6
airport:FRA	prop:code	"FRA"	default

https://docs.aws.amazon.com/neptune/latest/userguide/feature-overview-data-model.html

Neptune graph data model – indices

Subject Predicate Object Graph

SPOG – Uses a key composed of Subject + Predicate + Object + Graph

Efficient lookup whenever a prefix of the positions, such as the vertex (subject) or vertex and property identifier, is bound: *Find airport:SEA (22) with code "SEA"*

POGS – Uses a key composed of Predicate + Object + Graph + Subject

Efficient access when only the edge or property label stored in P position is bound:
What nodes have code "SEA"?

GPSO – Uses a key composed of Graph + Predicate + Subject + Object

Efficient access with the graph (or edge ID) and a property identifier is bound:
What edges are have routes to "FRA"?

https://docs.aws.amazon.com/neptune/latest/userguide/feature-overview-data-model.html

Capabilities

New features in Amazon Neptune

Neptune Streams	Complete sequence of change-log entries, which record every change made to graph
SPARQL 1.1 Federated Query	Use SPARQL to express queries across diverse data sources
Transaction semantics	Formalized semantics to help you avoid data anomalies
Gremlin/SPARQL Explain	Gain insights into the query plan and evaluation order
Gremlin sessions	Queries run during the session are committed as part of a single transaction
Database cloning	Create multiple clones of a DB cluster using copy-on-write semantics
Elasticsearch integration	Full-text search using Elasticsearch with graph data in Neptune
Neptune Workbench	In-console notebook experience to query your graph
Low cost T3 instances	Next generation burstable general-purpose instance type as low as 10 cents/Hr
Delete Protection	Configure a cluster with deletion protection to prevent accidental deletes by any user
Start/Stop cluster	Stop databases when it is not required to be running all of the time
Cross region snapshot copy	Copy snapshots across regions for testing and disaster recovery
Enforce SSL connections	Enforce SSL connections with option to disable SSL in regions where both are supported.
Simplified Console Management	Simplified console experience to manage cluster, instances and their properties.

Connect to Neptune

Gremlin

$G = (V, E)$

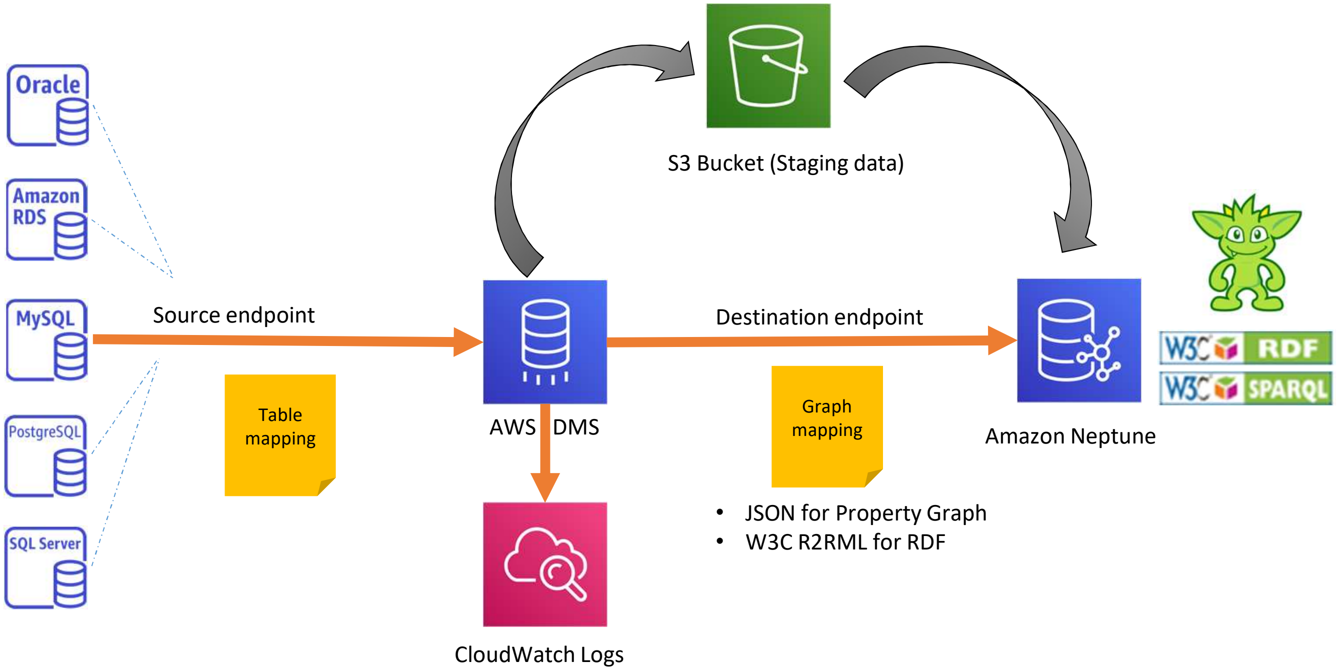
- HTTP or Web Socket (WS)
- Can send queries using curl
- Gremlin console
- Gremlin Language Variants (GLV)
  - Java (reference impl.)
  - Python, Node.js, .Net
  - Other OSS clients available.
- Queries sent as strings or as Gremlin “Byte Code” from GLVs.
- Tuning of client, batching and threading helps with performance.
- Need special client for SigV4 signing if IAM authentication is enabled on Neptune.

W3C SPARQL

- HTTP
- Can send queries using curl
- Can use open source Eclipse RDF4J console.
- Send queries from an application as text.
- Eclipse RDF4J also has Java libraries.
- Apache Jena has full RDF support.
- Many open source client libraries for languages such as Python.
- Client tuning helps overall throughput.
- Many third party open source and commercial tools available for query development.
- https://www.w3.org/2001/sw/wiki/Tools

https://docs.aws.amazon.com/neptune/latest/userguide/intro.html

Copy data from relational DB to Neptune using DMS



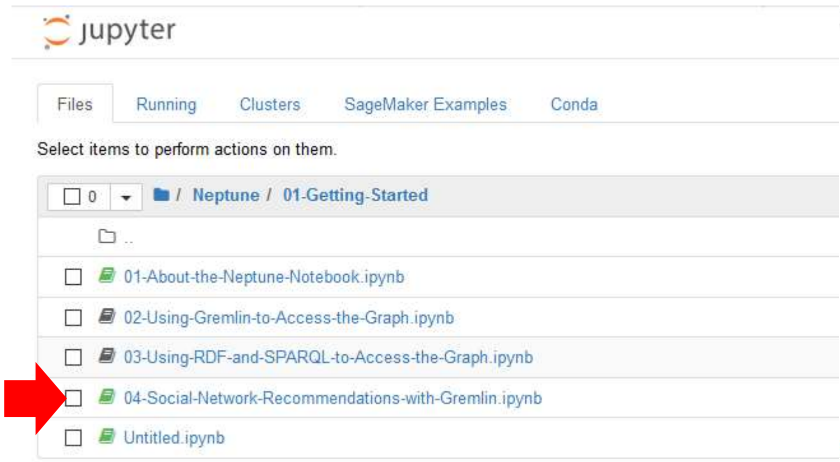
- JSON for Property Graph
- W3C R2RML for RDF

Other ways to load data

- Bulk import data from S3 bucket for Property Graph or RDF
- Use insert queries
- Import data from Neo4j

Neptune Workbench – Query Neptune

Query easily using Jupyter notebooks



Select items to perform actions on them.

☐ 0 / Neptune / 01-Getting-Started

..

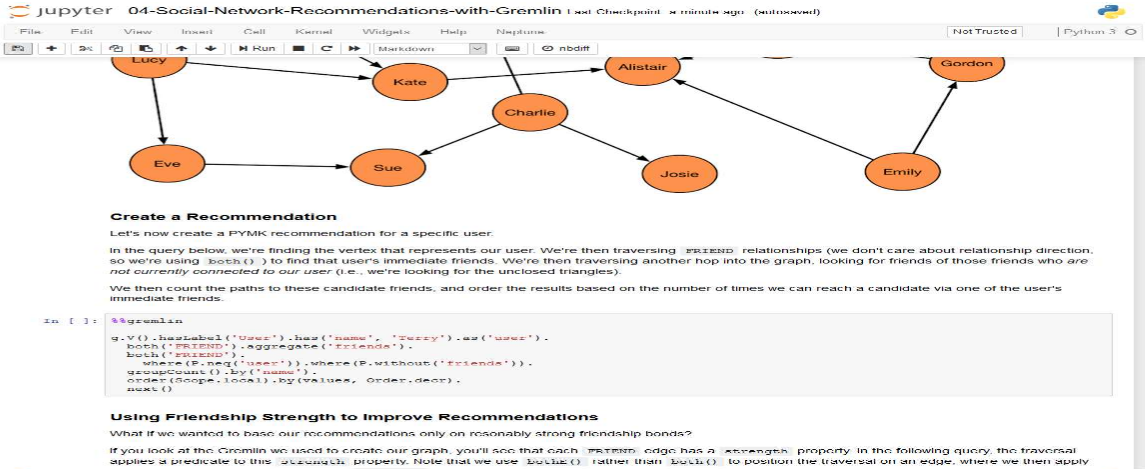
☐ 01-About-the-Neptune-Notebook.ipynb

☐ 02-Using-Gremlin-to-Access-the-Graph.ipynb

☐ 03-Using-RDF-and-SPARQL-to-Access-the-Graph.ipynb

☐ 04-Social-Network-Recommendations-with-Gremlin.ipynb

☐ Untitled.ipynb



Create a Recommendation

Let's now create a PYMK recommendation for a specific user.

In the query below, we're finding the vertex that represents our user. We're then traversing `FRIEND` relationships (we don't care about relationship direction, so we're using `both()`) to find that user's immediate friends. We're then traversing another hop into the graph, looking for friends of those friends who *are not currently connected to our user* (i.e., we're looking for the unclosed triangles).

We then count the paths to these candidate friends, and order the results based on the number of times we can reach a candidate via one of the user's immediate friends.

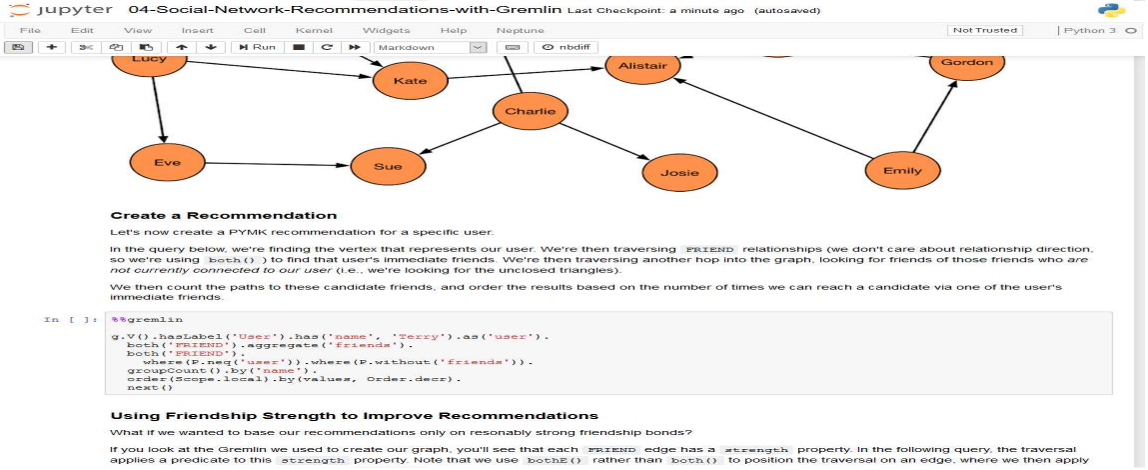
In [ ]:

```
%%gremlin

g.V().hasLabel('User').has('name', 'Terry').as('user').
  both('FRIEND').aggregate('friends').
  both('FRIEND').
    where(P.neq('user')).where(P.without('friends')).
  groupCount().by('name').
  order(Scope.local).by(values, Order.decr).
  next()
```

Using Friendship Strength to Improve Recommendations

What if we wanted to base our recommendations only on resonably strong friendship bonds?

If you look at the Gremlin we used to create our graph, you'll see that each `FRIEND` edge has a `strength` property. In the following query, the traversal applies a predicate to this `strength` property. Note that we use `bothE()` rather than `both()` to position the traversal on an edge, where we then apply



jupyter 04-Social-Network-Recommendations-with-Gremlin Last Checkpoint: a minute ago (autosaved)

Create a Recommendation

Let's now create a PYMK recommendation for a specific user.

In the query below, we're finding the vertex that represents our user. We're then traversing `FRIEND` relationships (we don't care about relationship direction, so we're using `both()`) to find that user's immediate friends. We're then traversing another hop into the graph, looking for friends of those friends who *are not currently connected to our user* (i.e., we're looking for the unclosed triangles).

We then count the paths to these candidate friends, and order the results based on the number of times we can reach a candidate via one of the user's immediate friends.

In [ ]:

```
%%gremlin

g.V().hasLabel('User').has('name', 'Terry').as('user').
  both('FRIEND').aggregate('friends').
  both('FRIEND').
    where(P.neq('user')).where(P.without('friends')).
  groupCount().by('name').
  order(Scope.local).by(values, Order.decr).
  next()
```

Using Friendship Strength to Improve Recommendations

What if we wanted to base our recommendations only on resonably strong friendship bonds?

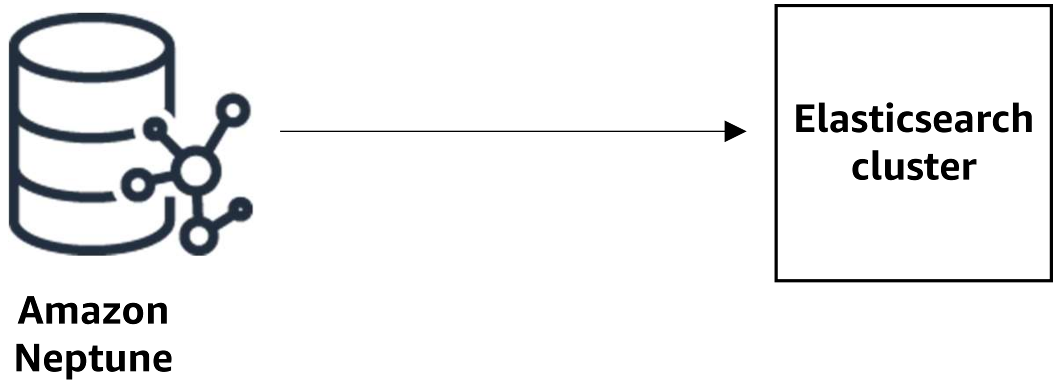
If you look at the Gremlin we used to create our graph, you'll see that each `FRIEND` edge has a `strength` property. In the following query, the traversal applies a predicate to this `strength` property. Note that we use `bothE()` rather than `both()` to position the traversal on an edge, where we then apply

Magic commands:

- Gremlin queries
- SPARQL queries
- Explain and Profile
- Status query
- Bulk load data

Neptune Elasticsearch integration

- Full-text search queries using Elasticsearch
- Use match, fuzzy, prefix, query_string options
- Supported for both SPARQL and Gremlin queries



Neptune Elasticsearch integration

Leverage Elasticsearch for graph data in Amazon Neptune

SPARQL

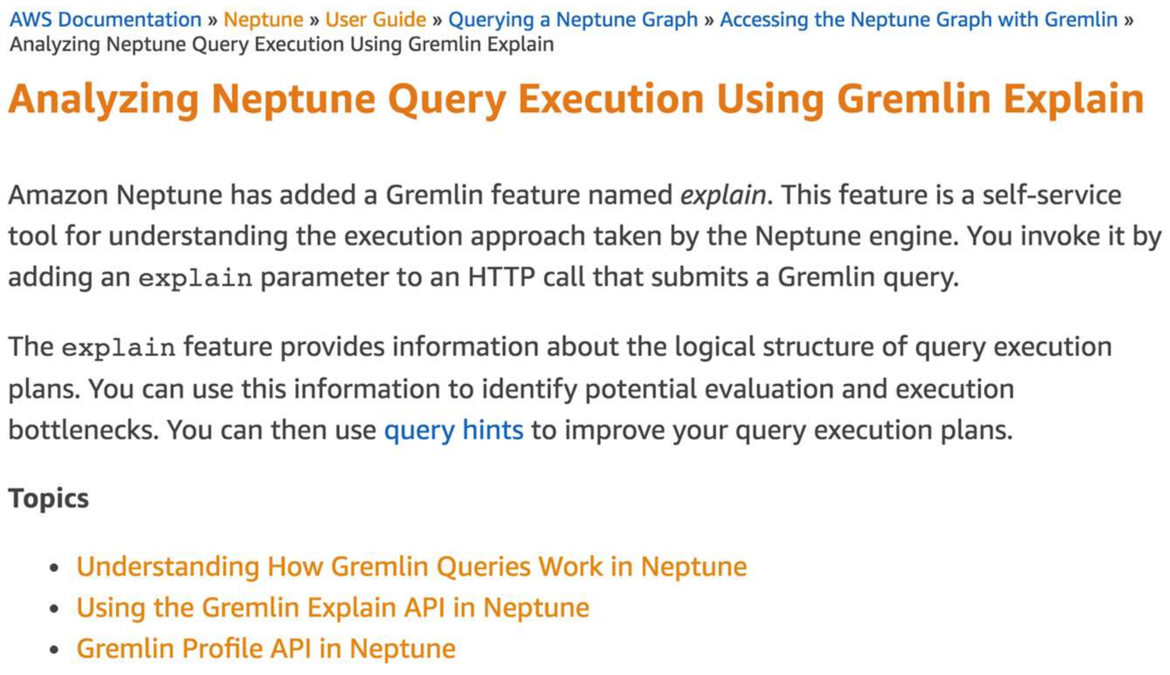
```
SELECT * WHERE {
  SERVICE
<http://aws.amazon.com/neptune/vocab/v01/fts#search>
{
    ?desc fts:query "regional" .
    ?desc fts:maxResults 100 .
  }
}
```

Gremlin

```
g.withSideEffect("Neptune#fts.endpoint",
"https://....amazonaws.com").
V().has('desc','Neptune#fts regional').
local(values('code','desc').fold()).
limit(100)
```

```
==>[HYA, Barnstable Municipal Boardman Polando Field]
==>[SPS, Sheppard Air Force Base-Wichita Falls Municipal Airport]
==>[ABR, Aberdeen Regional Airport]
==>[SLK, Adirondack Regional Airport]
==>[BFD, Bradford Regional Airport]
```

Making the most of your graph queries



AWS Documentation » Neptune » User Guide » Querying a Neptune Graph » Accessing the Neptune Graph with Gremlin » Analyzing Neptune Query Execution Using Gremlin Explain

Analyzing Neptune Query Execution Using Gremlin Explain

Amazon Neptune has added a Gremlin feature named *explain*. This feature is a self-service tool for understanding the execution approach taken by the Neptune engine. You invoke it by adding an `explain` parameter to an HTTP call that submits a Gremlin query.

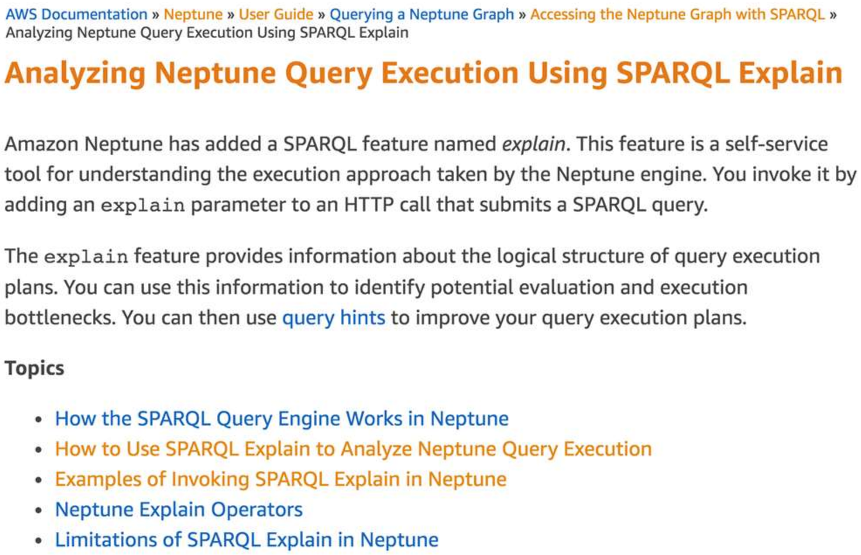
The `explain` feature provides information about the logical structure of query execution plans. You can use this information to identify potential evaluation and execution bottlenecks. You can then use query hints to improve your query execution plans.

Topics

- Understanding How Gremlin Queries Work in Neptune
- Using the Gremlin Explain API in Neptune
- Gremlin Profile API in Neptune

https://docs.aws.amazon.com/neptune/latest/userguide/gremlin-explain.html

Making the most of your graph queries



AWS Documentation » Neptune » User Guide » Querying a Neptune Graph » Accessing the Neptune Graph with SPARQL » Analyzing Neptune Query Execution Using SPARQL Explain

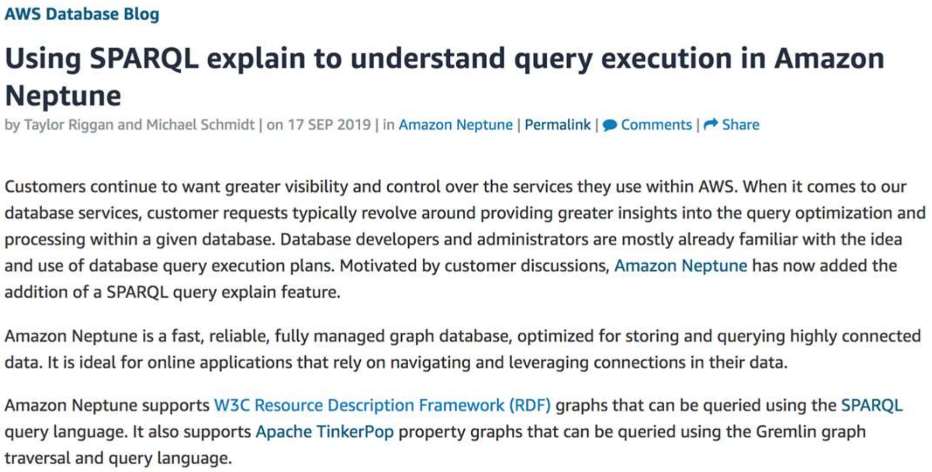
Analyzing Neptune Query Execution Using SPARQL Explain

Amazon Neptune has added a SPARQL feature named *explain*. This feature is a self-service tool for understanding the execution approach taken by the Neptune engine. You invoke it by adding an `explain` parameter to an HTTP call that submits a SPARQL query.

The `explain` feature provides information about the logical structure of query execution plans. You can use this information to identify potential evaluation and execution bottlenecks. You can then use query hints to improve your query execution plans.

Topics

- How the SPARQL Query Engine Works in Neptune
- How to Use SPARQL Explain to Analyze Neptune Query Execution
- Examples of Invoking SPARQL Explain in Neptune
- Neptune Explain Operators
- Limitations of SPARQL Explain in Neptune



AWS Database Blog

Using SPARQL explain to understand query execution in Amazon Neptune

by Taylor Riggan and Michael Schmidt | on 17 SEP 2019 | in Amazon Neptune | Permalink | Comments | Share

Customers continue to want greater visibility and control over the services they use within AWS. When it comes to our database services, customer requests typically revolve around providing greater insights into the query optimization and processing within a given database. Database developers and administrators are mostly already familiar with the idea and use of database query execution plans. Motivated by customer discussions, Amazon Neptune has now added the addition of a SPARQL query explain feature.

Amazon Neptune is a fast, reliable, fully managed graph database, optimized for storing and querying highly connected data. It is ideal for online applications that rely on navigating and leveraging connections in their data.

Amazon Neptune supports W3C Resource Description Framework (RDF) graphs that can be queried using the SPARQL query language. It also supports Apache TinkerPop property graphs that can be queried using the Gremlin graph traversal and query language.

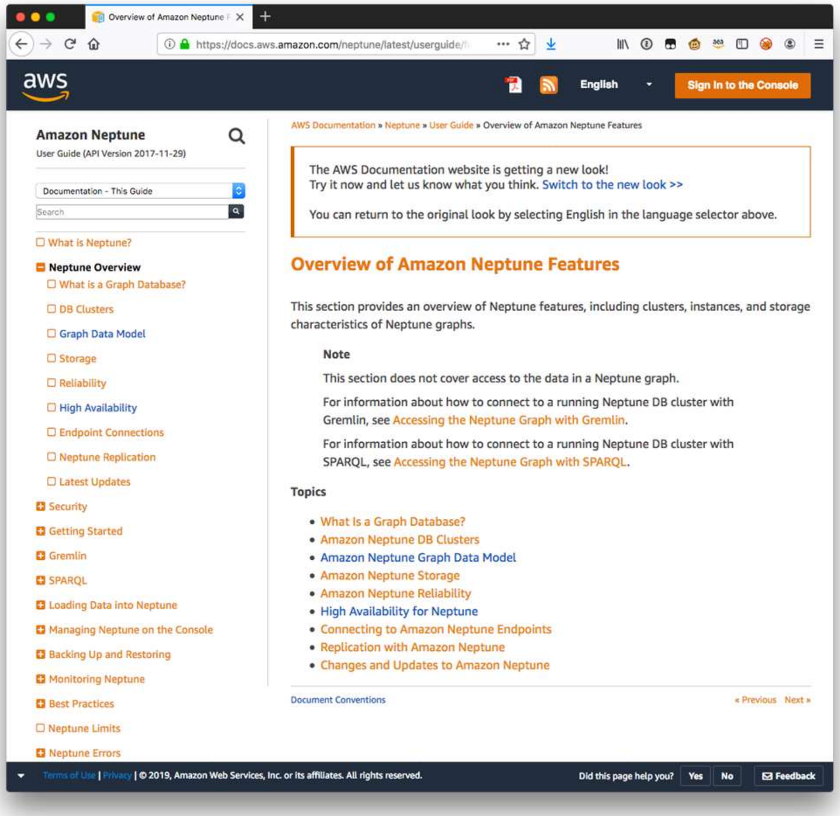
https://aws.amazon.com/blogs/database/using-sparql-explain-to-understand-query-execution-in-amazon-neptune/

https://docs.aws.amazon.com/neptune/latest/userguide/sparql-explain.html

Resources

Documentation

Start with the ‘What is Neptune?’ and ‘Neptune Overview’ sections



The AWS Documentation website is getting a new look!
Try it now and let us know what you think. Switch to the new look >>

You can return to the original look by selecting English in the language selector above.

Overview of Amazon Neptune Features

This section provides an overview of Neptune features, including clusters, instances, and storage characteristics of Neptune graphs.

Note

This section does not cover access to the data in a Neptune graph.

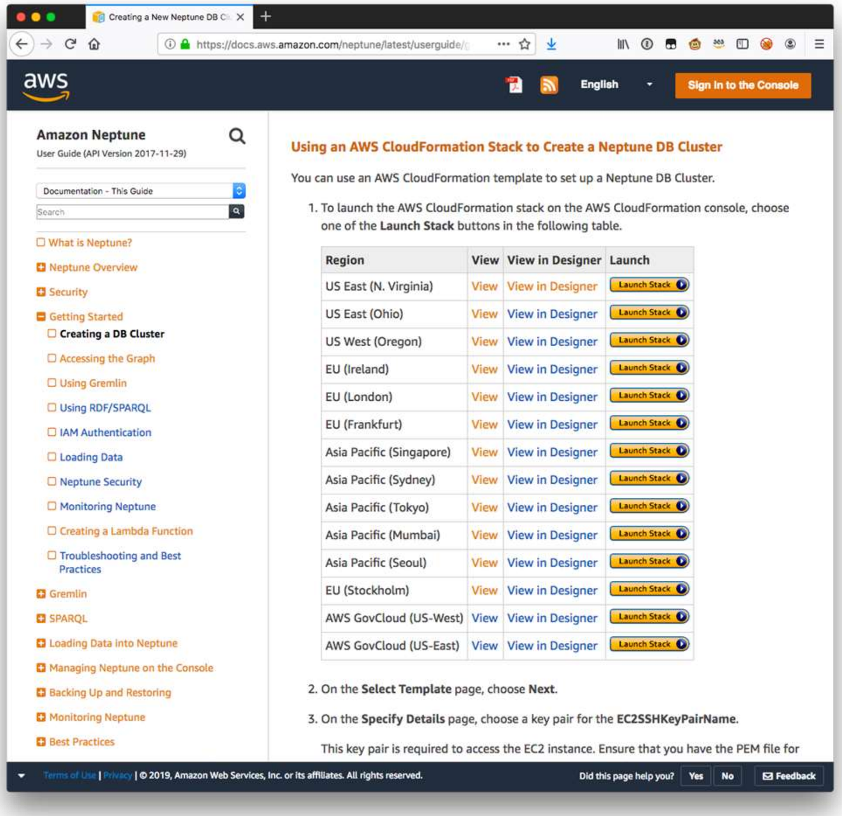
For information about how to connect to a running Neptune DB cluster with Gremlin, see Accessing the Neptune Graph with Gremlin.

For information about how to connect to a running Neptune DB cluster with SPARQL, see Accessing the Neptune Graph with SPARQL.

Topics

- What Is a Graph Database?
- Amazon Neptune DB Clusters
- Amazon Neptune Graph Data Model
- Amazon Neptune Storage
- Amazon Neptune Reliability
- High Availability for Neptune
- Connecting to Amazon Neptune Endpoints
- Replication with Amazon Neptune
- Changes and Updates to Amazon Neptune

Document Conventions
« Previous Next »
Terms of Use | Privacy | © 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Did this page help you? Yes No
Feedback


Creating a New Neptune DB C
https://docs.aws.amazon.com/neptune/latest/userguide/g
aws
English
Sign In to the Console
Amazon Neptune
User Guide (API Version 2017-11-29)
Documentation - This Guide
Search
☐ What is Neptune?
Neptune Overview
Security
Getting Started
☐ Creating a DB Cluster
☐ Accessing the Graph
☐ Using Gremlin
☐ Using RDF/SPARQL
☐ IAM Authentication
☐ Loading Data
☐ Neptune Security
☐ Monitoring Neptune
☐ Creating a Lambda Function
☐ Troubleshooting and Best Practices
Gremlin
SPARQL
Loading Data into Neptune
Managing Neptune on the Console
Backing Up and Restoring
Monitoring Neptune
Best Practices

Using an AWS CloudFormation Stack to Create a Neptune DB Cluster

You can use an AWS CloudFormation template to set up a Neptune DB Cluster.

1. To launch the AWS CloudFormation stack on the AWS CloudFormation console, choose one of the **Launch Stack** buttons in the following table.

Region	View	View in Designer	Launch
US East (N. Virginia)	View	View in Designer	Launch Stack
US East (Ohio)	View	View in Designer	Launch Stack
US West (Oregon)	View	View in Designer	Launch Stack
EU (Ireland)	View	View in Designer	Launch Stack
EU (London)	View	View in Designer	Launch Stack
EU (Frankfurt)	View	View in Designer	Launch Stack
Asia Pacific (Singapore)	View	View in Designer	Launch Stack
Asia Pacific (Sydney)	View	View in Designer	Launch Stack
Asia Pacific (Tokyo)	View	View in Designer	Launch Stack
Asia Pacific (Mumbai)	View	View in Designer	Launch Stack
Asia Pacific (Seoul)	View	View in Designer	Launch Stack
EU (Stockholm)	View	View in Designer	Launch Stack
AWS GovCloud (US-West)	View	View in Designer	Launch Stack
AWS GovCloud (US-East)	View	View in Designer	Launch Stack

2. On the **Select Template** page, choose **Next**.

3. On the **Specify Details** page, choose a key pair for the **EC2SSHKeyPairName**.

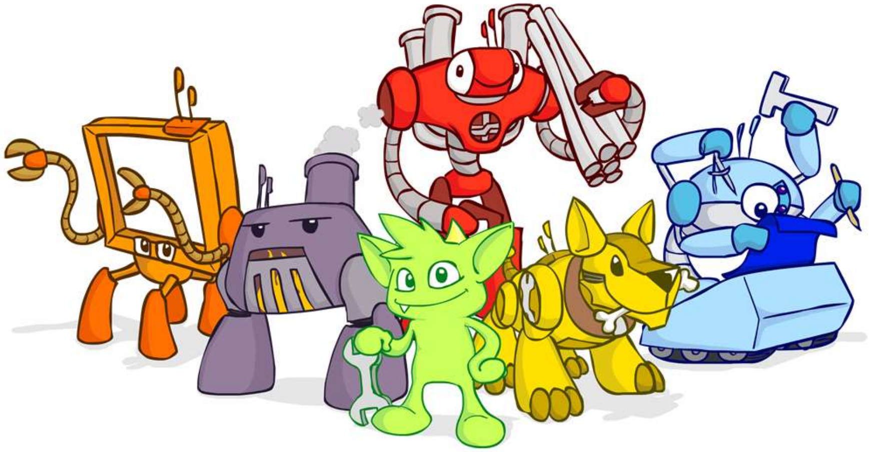
This key pair is required to access the EC2 instance. Ensure that you have the PEM file for

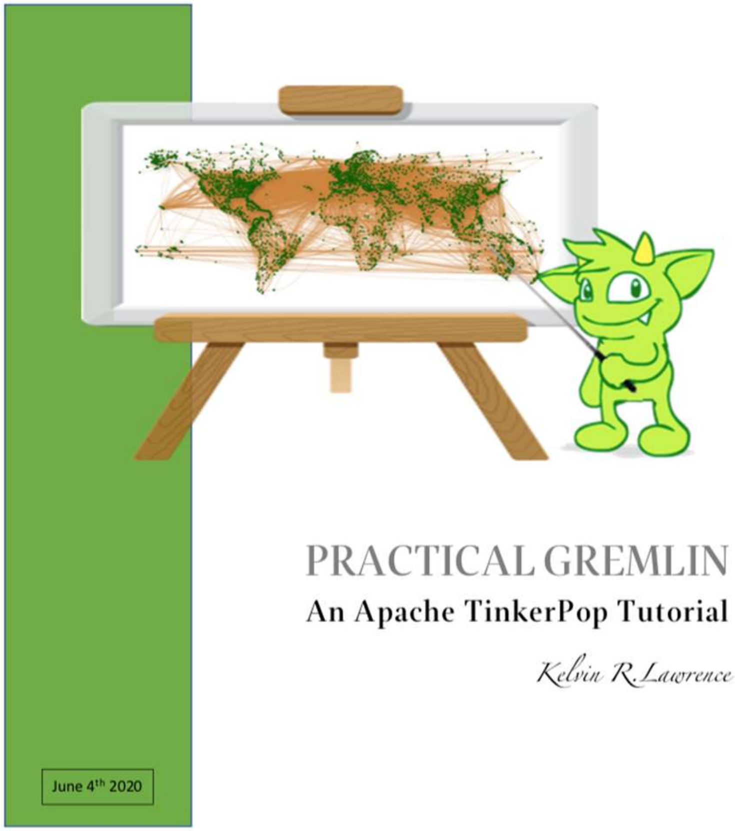
Terms of Use | Privacy | © 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Did this page help you? Yes No
Feedback

Learn Gremlin

http://kelvinlawrence.net/book/Gremlin-Graph-Guide.html

https://github.com/krlawrence/graph





PRACTICAL GREMLIN

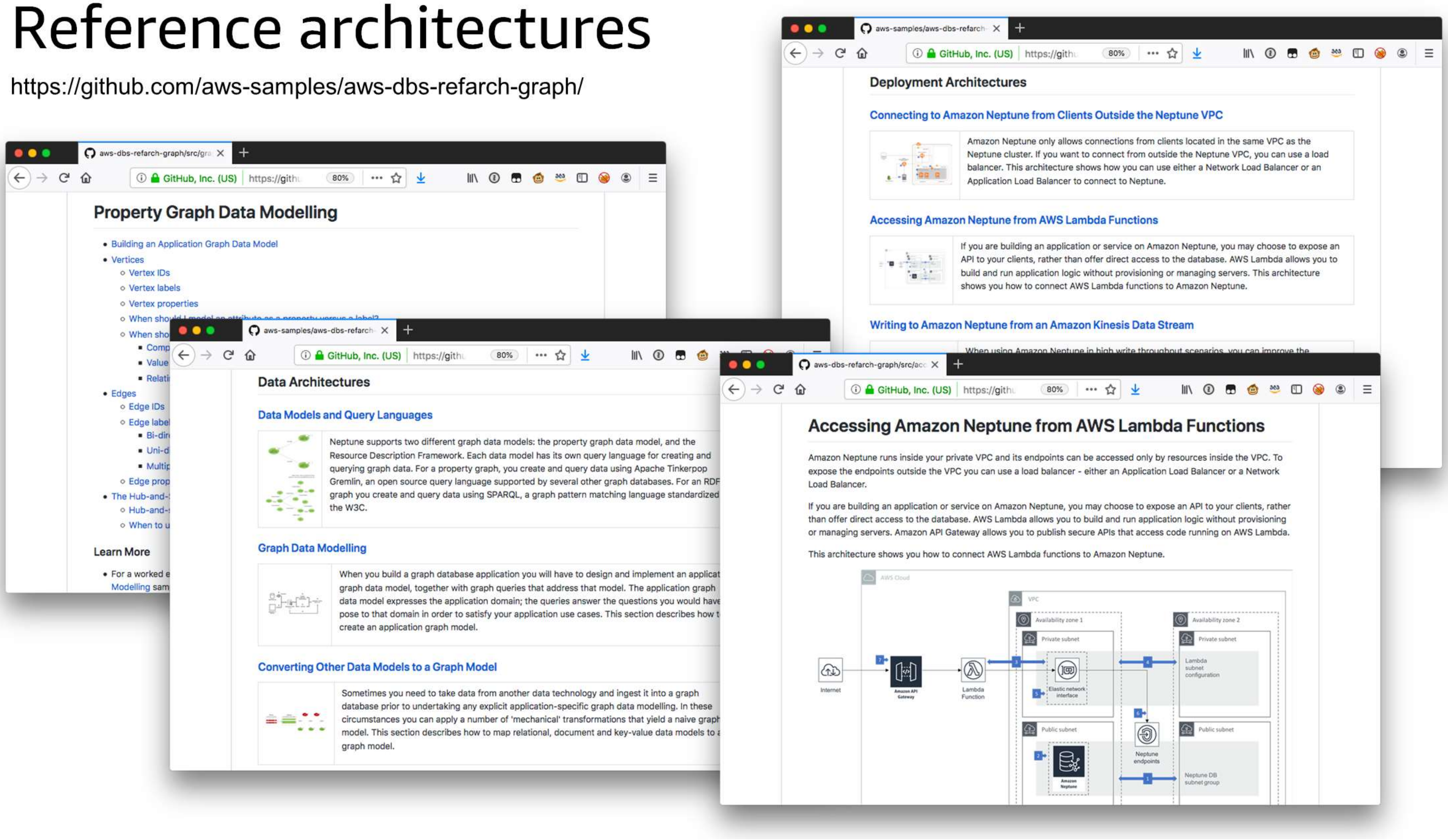
An Apache TinkerPop Tutorial

Kelvin R. Lawrence

June 4th 2020

Reference architectures

https://github.com/aws-samples/aws-dbs-refarch-graph/



Property Graph Data Modelling

- Building an Application Graph Data Model
- Vertices
  - Vertex IDs
  - Vertex labels
  - Vertex properties
  - When should I model an attribute as a property versus a label?
  - When sho
    - Comp
    - Value
    - Relati
- Edges
  - Edge IDs
  - Edge label
    - Bi-dir
    - Uni-d
    - Multi
  - Edge prop
- The Hub-and-
  - Hub-and-
  - When to u

Learn More

- For a worked e
  Modelling sam

Data Architectures

Data Models and Query Languages

Neptune supports two different graph data models: the property graph data model, and the Resource Description Framework. Each data model has its own query language for creating and querying graph data. For a property graph, you create and query data using Apache Tinkerpop Gremlin, an open source query language supported by several other graph databases. For an RDF graph you create and query data using SPARQL, a graph pattern matching language standardized the W3C.

Graph Data Modelling

When you build a graph database application you will have to design and implement an applicat graph data model, together with graph queries that address that model. The application graph data model expresses the application domain; the queries answer the questions you would have pose to that domain in order to satisfy your application use cases. This section describes how t create an application graph model.

Converting Other Data Models to a Graph Model

Sometimes you need to take data from another data technology and ingest it into a graph database prior to undertaking any explicit application-specific graph data modelling. In these circumstances you can apply a number of 'mechanical' transformations that yield a naive graph model. This section describes how to map relational, document and key-value data models to a graph model.

Deployment Architectures

Connecting to Amazon Neptune from Clients Outside the Neptune VPC

Amazon Neptune only allows connections from clients located in the same VPC as the Neptune cluster. If you want to connect from outside the Neptune VPC, you can use a load balancer. This architecture shows how you can use either a Network Load Balancer or an Application Load Balancer to connect to Neptune.

Accessing Amazon Neptune from AWS Lambda Functions

If you are building an application or service on Amazon Neptune, you may choose to expose an API to your clients, rather than offer direct access to the database. AWS Lambda allows you to build and run application logic without provisioning or managing servers. This architecture shows you how to connect AWS Lambda functions to Amazon Neptune.

Writing to Amazon Neptune from an Amazon Kinesis Data Stream

When using Amazon Neptune in high write throughput scenarios, you can improve the

Accessing Amazon Neptune from AWS Lambda Functions

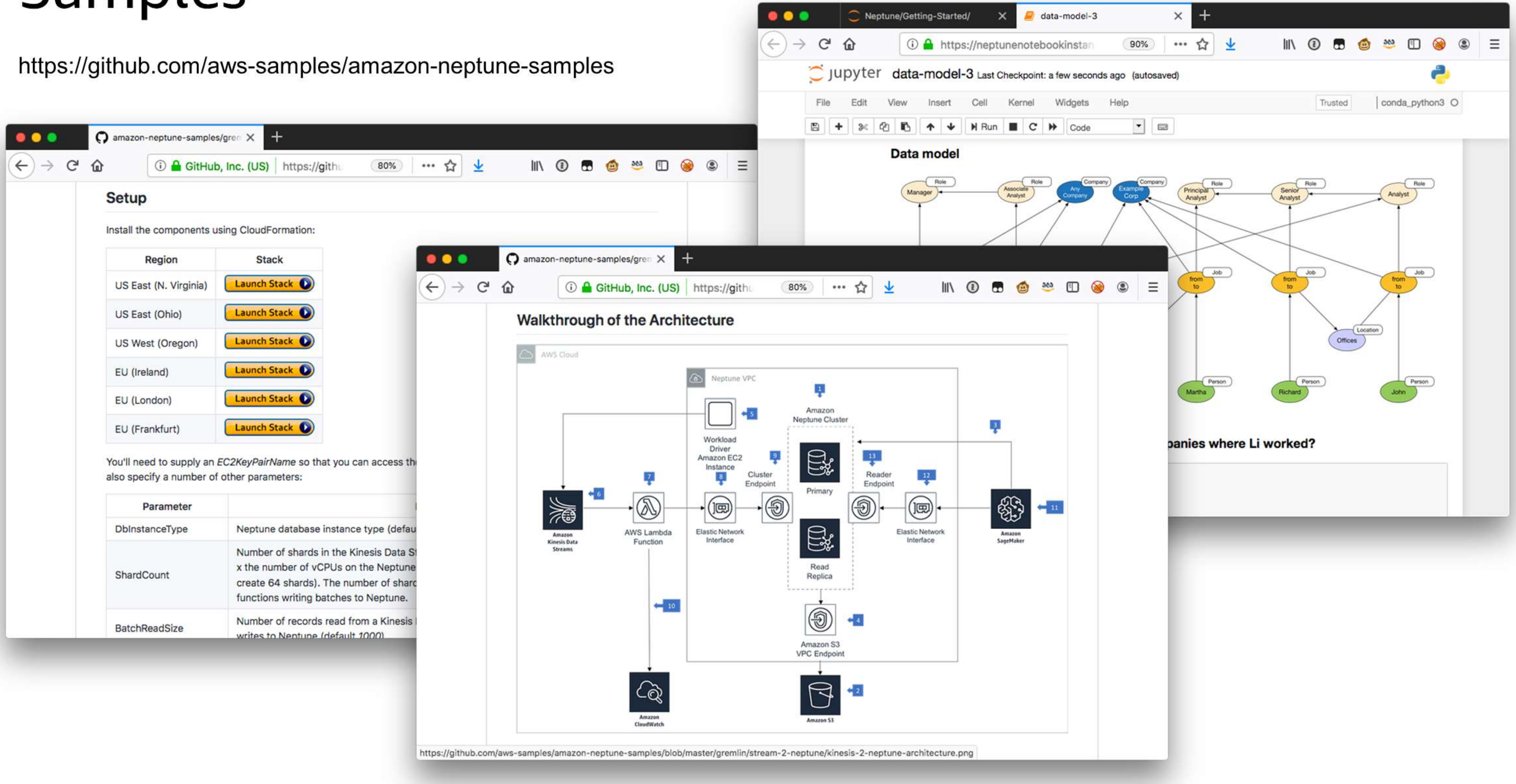
Amazon Neptune runs inside your private VPC and its endpoints can be accessed only by resources inside the VPC. To expose the endpoints outside the VPC you can use a load balancer - either an Application Load Balancer or a Network Load Balancer.

If you are building an application or service on Amazon Neptune, you may choose to expose an API to your clients, rather than offer direct access to the database. AWS Lambda allows you to build and run application logic without provisioning or managing servers. Amazon API Gateway allows you to publish secure APIs that access code running on AWS Lambda.

This architecture shows you how to connect AWS Lambda functions to Amazon Neptune.

Samples

https://github.com/aws-samples/amazon-neptune-samples



Setup

Install the components using CloudFormation:

Region	Stack
US East (N. Virginia)	Launch Stack
US East (Ohio)	Launch Stack
US West (Oregon)	Launch Stack
EU (Ireland)	Launch Stack
EU (London)	Launch Stack
EU (Frankfurt)	Launch Stack

You'll need to supply an *EC2KeyPairName* so that you can access th
also specify a number of other parameters:

Parameter	
DbInstanceType	Neptune database instance type (defau
ShardCount	Number of shards in the Kinesis Data St x the number of vCPUs on the Neptune create 64 shards). The number of shard functions writing batches to Neptune.
BatchReadSize	Number of records read from a Kinesis writes to Neptune (default *1000*)

Walkthrough of the Architecture

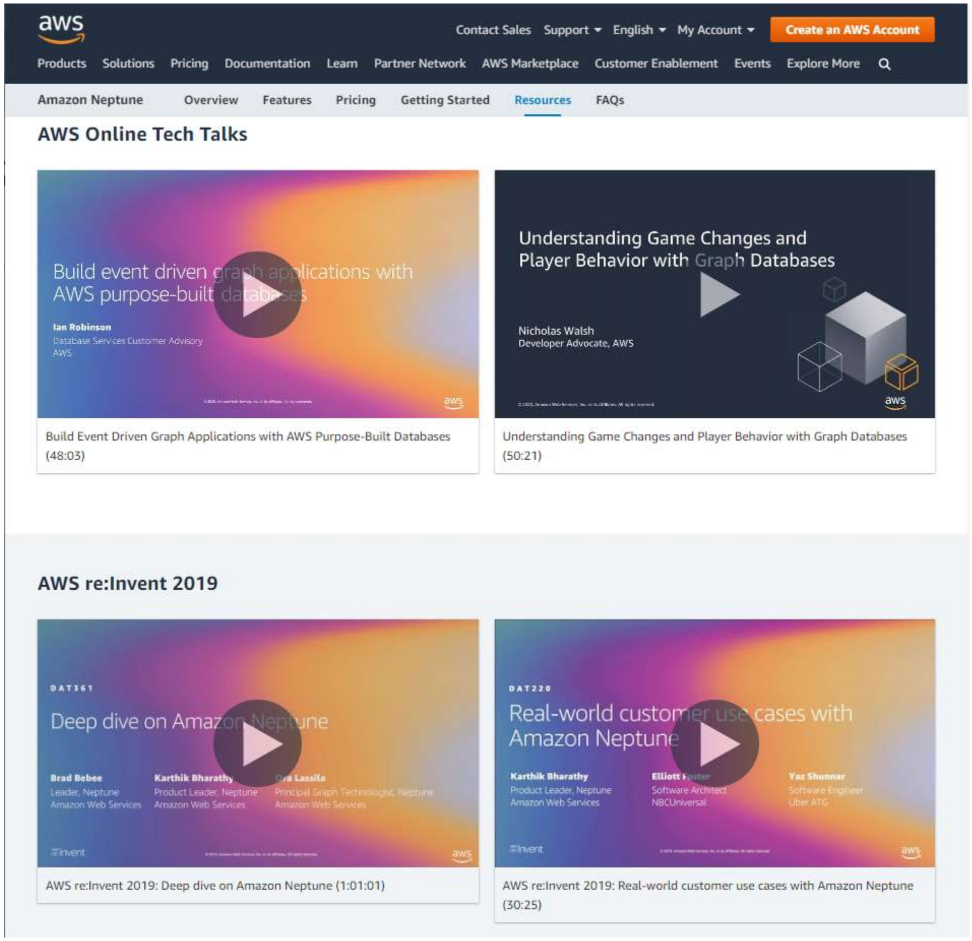
https://github.com/aws-samples/amazon-neptune-samples/blob/master/gremlin/stream-2-neptune/kinesis-2-neptune-architecture.png

Data model

panies where Li worked?

Use cases, videos, blog posts, code

https://aws.amazon.com/neptune/developer-resources/



AWS Online Tech Talks

Build event driven graph applications with AWS purpose-built databases
Ian Robinson
Database Services Customer Advisory
AWS
Understanding Game Changes and Player Behavior with Graph Databases
Nicholas Walsh
Developer Advocate, AWS

Build Event Driven Graph Applications with AWS Purpose-Built Databases (48:03)

Understanding Game Changes and Player Behavior with Graph Databases (50:21)

AWS re:Invent 2019

AWS re:Invent 2019: Deep dive on Amazon Neptune (1:01:01)

AWS re:Invent 2019: Real-world customer use cases with Amazon Neptune (30:25)

Questions?

Thank you!