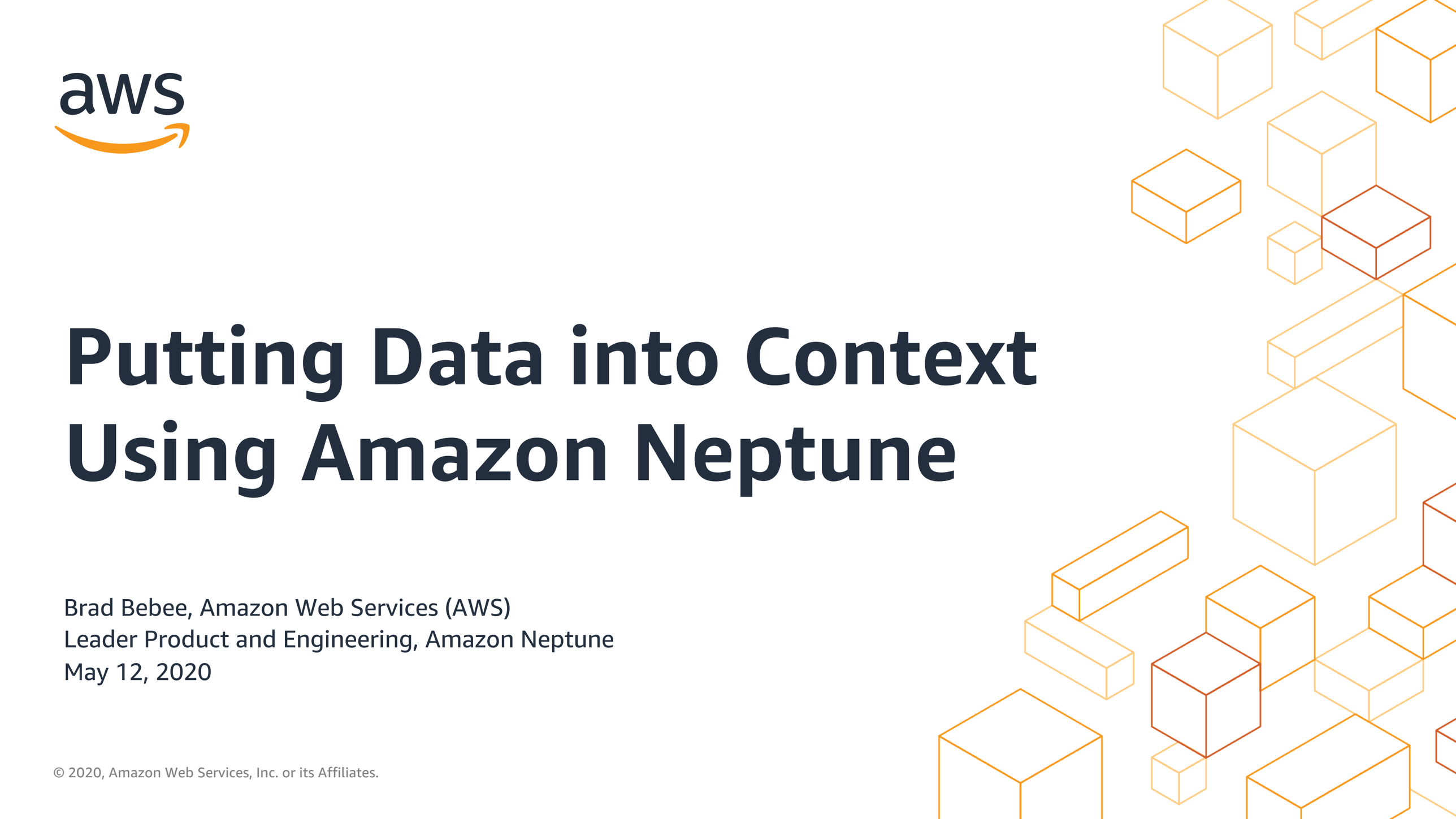


Putting Data into Context Using Amazon Neptune

Brad Bebee, Amazon Web Services (AWS)
Leader Product and Engineering, Amazon Neptune
May 12, 2020

A journey through graph…

2000

2001

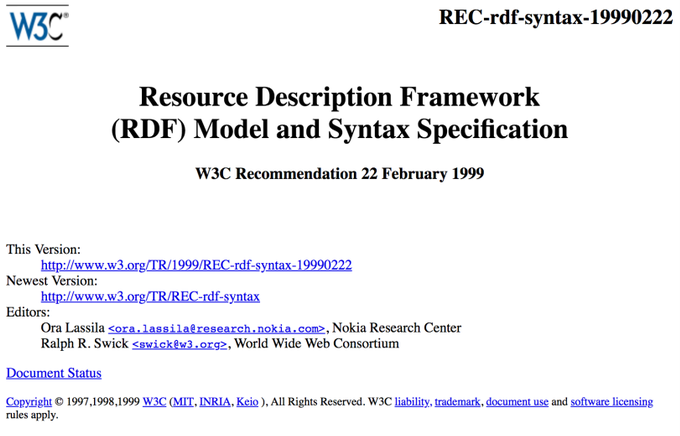
Sem. Web. “Graph” Applications

2005-2009

1999

2004

OWL Web Ontology Language Reference



Wikidata Query Service

2015

2016

Amazon Neptune General Availability

2018

2014

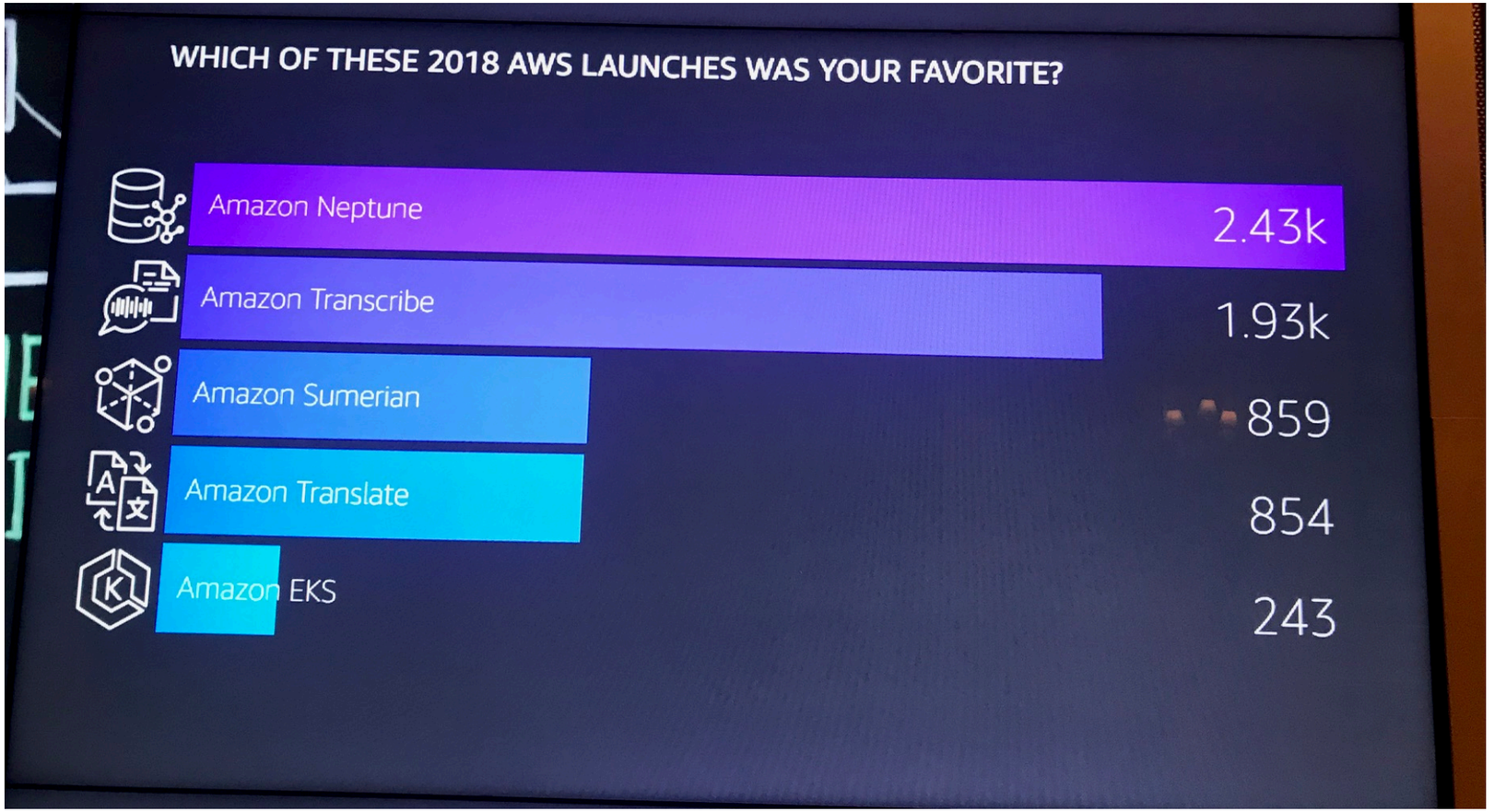
2017

2020

blazegraph™ by SYSTAP

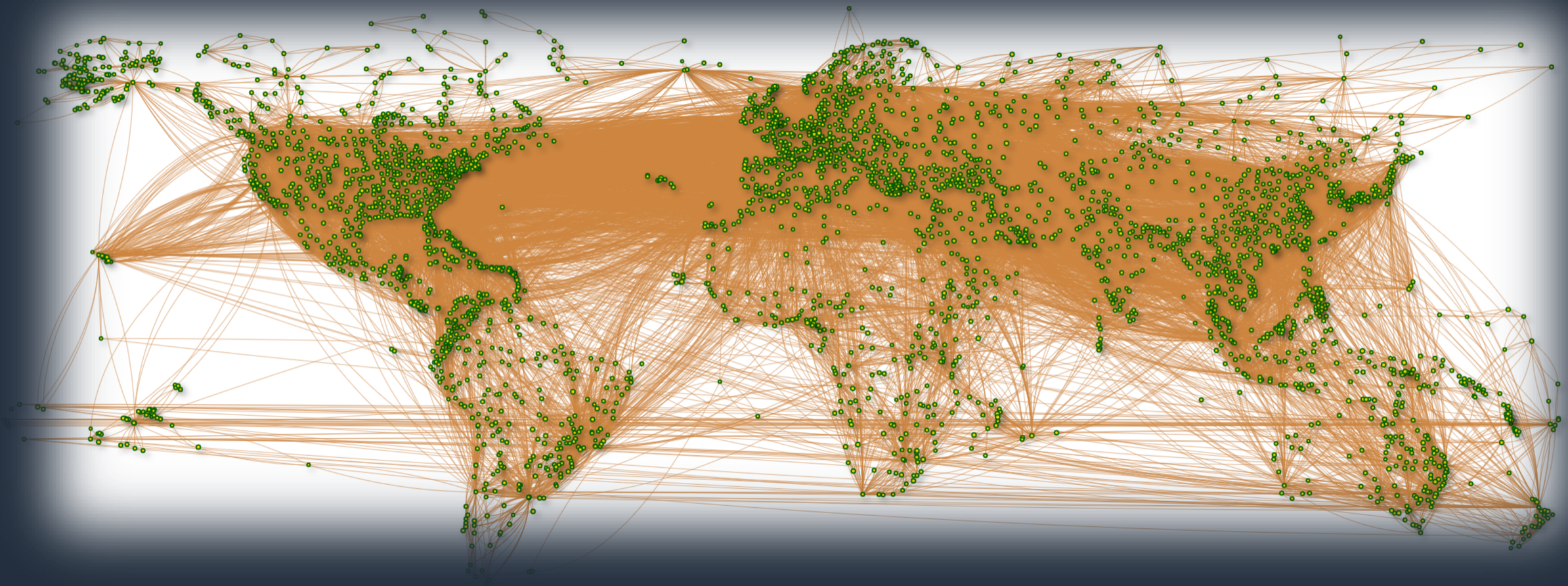
May’2020 Neptune turns two – still day one for graph!

Customers are excited about graph.



AWS re:Invent

Graphs are all around us

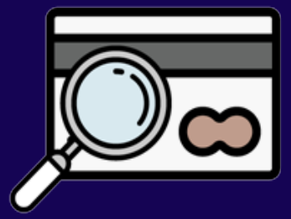


Graph use cases

Social networking

Recommendations

Knowledge graphs



Fraud detection

Life Sciences

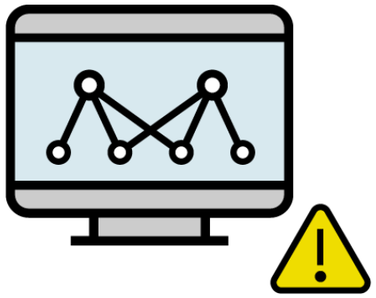
Network & IT operations

Connected data

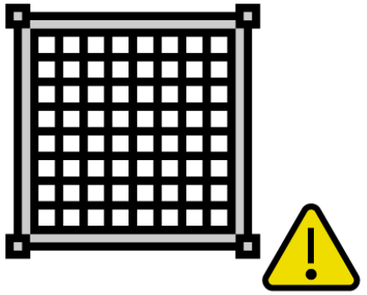
- Navigate (variably) connected structure
- Filter or compute a result on the basis of the *strength*, *weight,* or *quality* of relationships

The challenges of building apps with highly connected data using a relational database (or key-value store)

Unnatural for querying graph



Inefficient graph processing



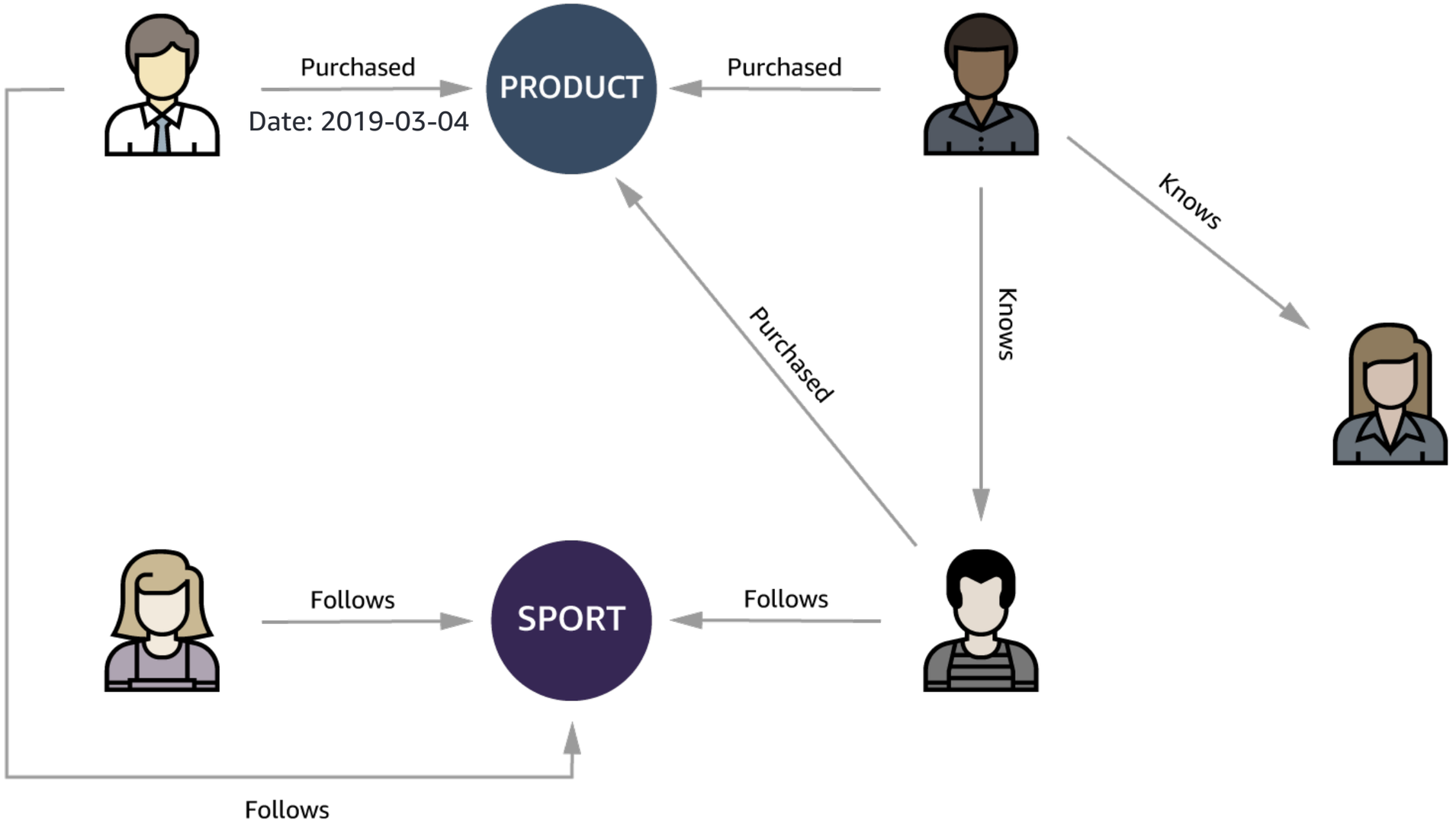
Rigid schema inflexible for changing data

Customers using Neptune

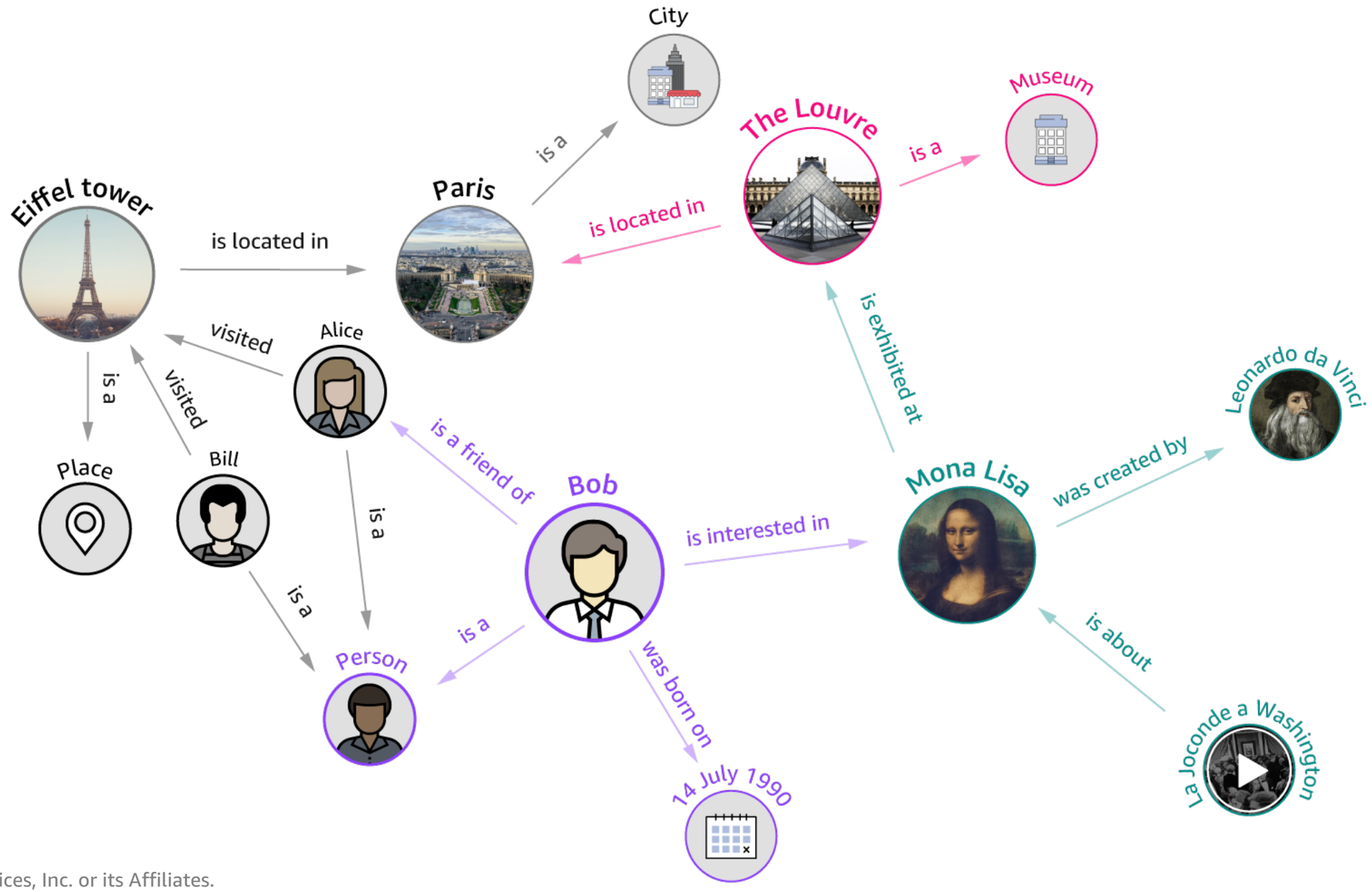
aws

When a customer thinks of a graph

When a customer thinks of a graph



When a customer thinks of a graph



When a customer thinks of a graph

But they’re typically not thinking about…

Graph models and frameworks

Property graph

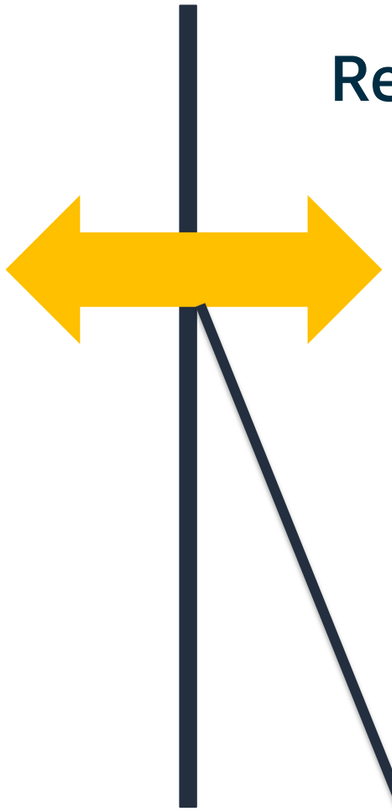
Apache TinkerPop Gremlin

openCypher

Property Graph Query Language (PGQL)

GQL

Others



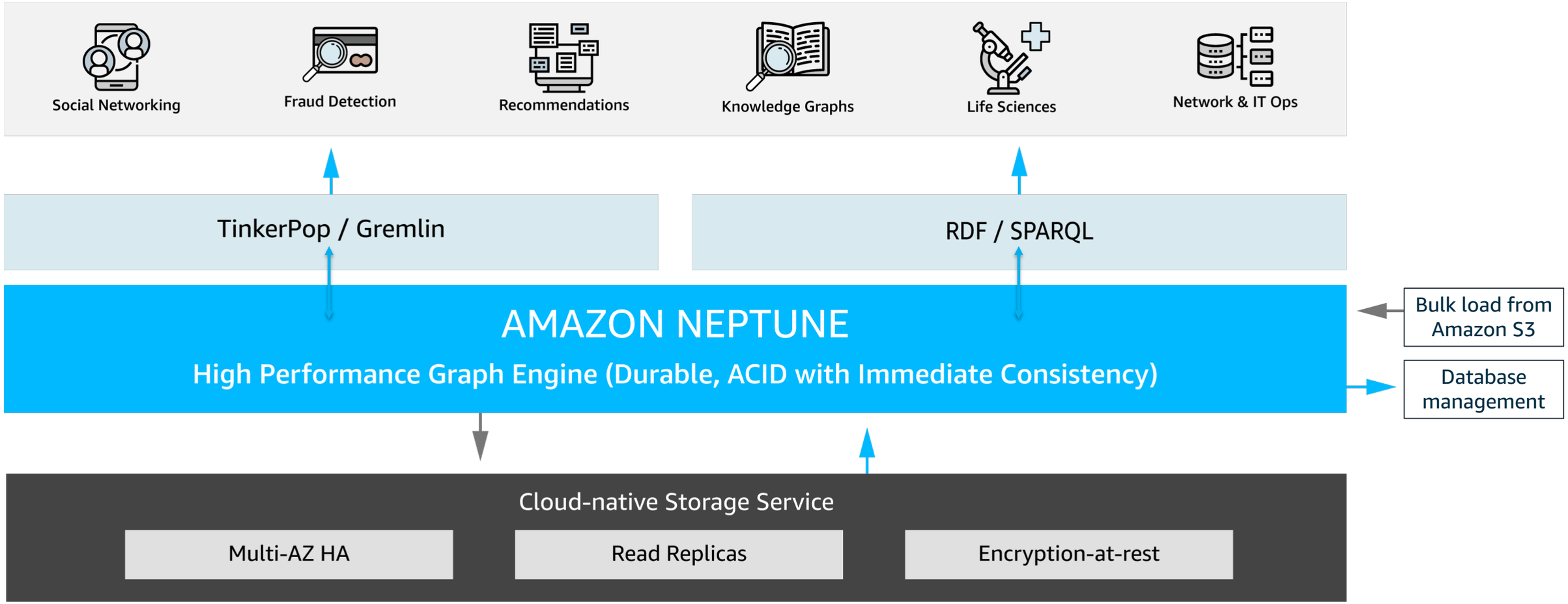
Resource Description Framework (RDF)

W3C standard

SPARQL Query Language

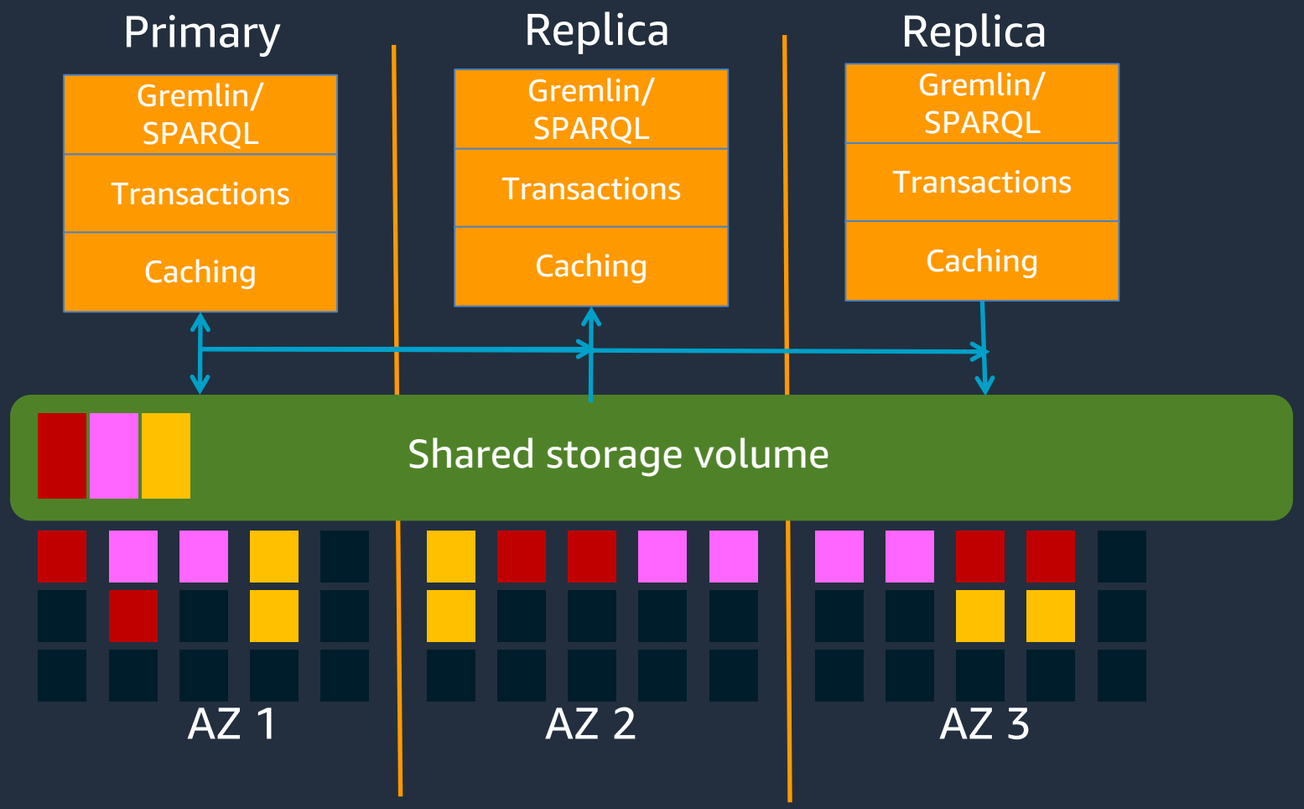
Many customers want both

Supporting both models is important to Amazon Neptune.



Distributed storage architecture

- Performance, availability, durability
- Scale-out replica architecture
- Shared storage volume with 10-GB segments striped across hundreds of nodes
- Data are 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

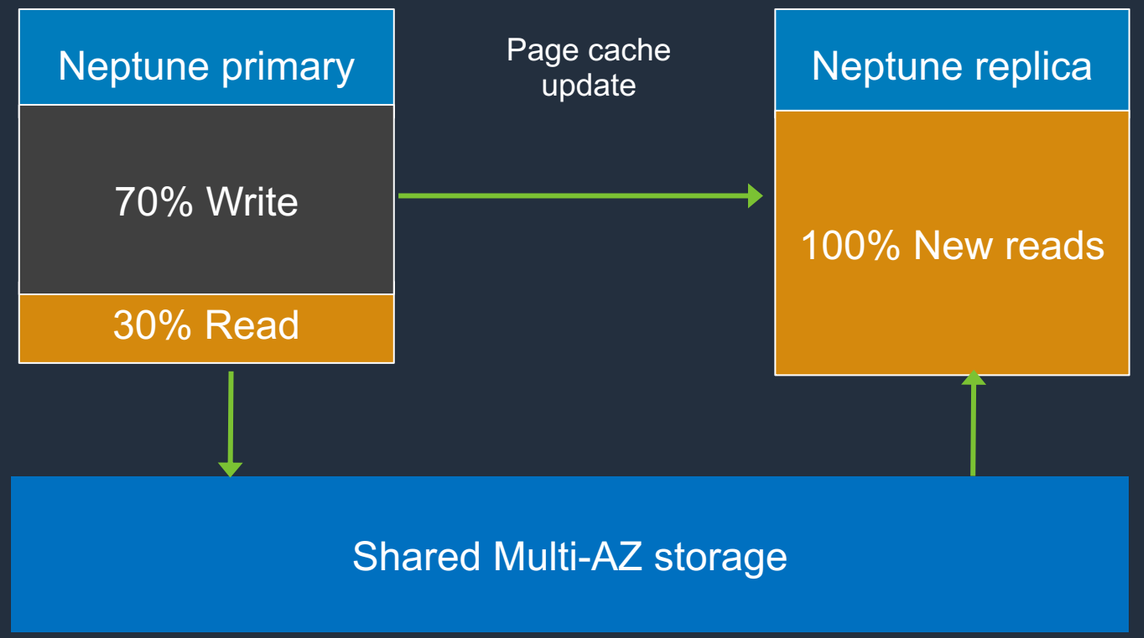
Low replica lag

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

Availability

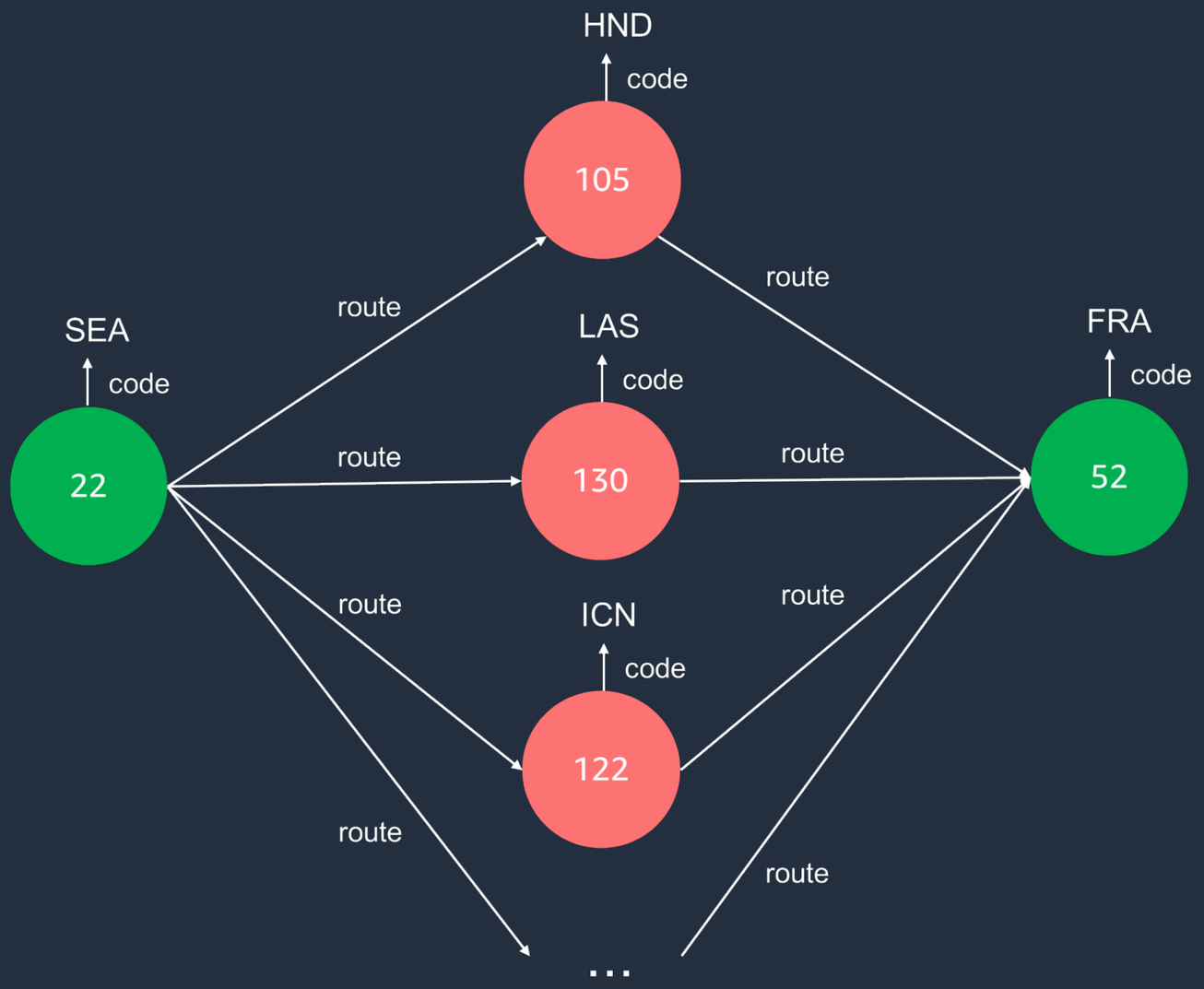
- Failing database nodes are automatically detected and replaced
- If primary fails, a replica replaces it (failover time typically <60 seconds)
- Primary upgrade by forced failover

Amazon Neptune read scaling



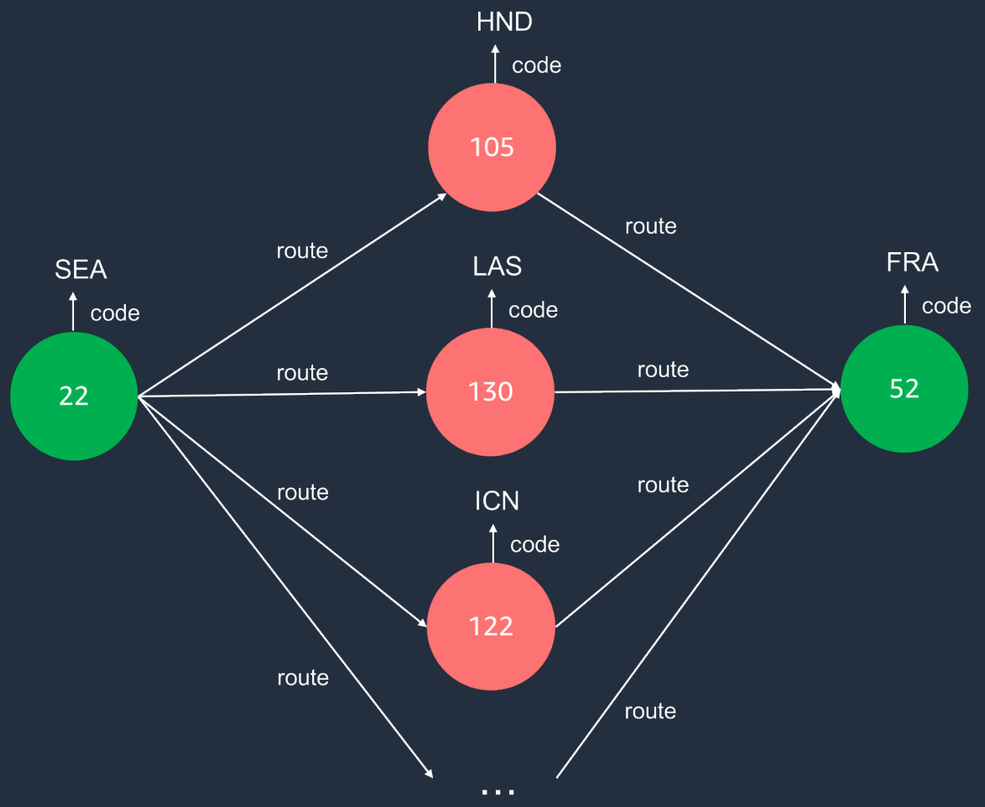
Air routes dataset

- Models the world's airline route network
- Queries operating over the airport connectivity graph
  - Example: I can travel from Seattle (SEA) to Frankfurt (FRA) via Tokyo Haneda (HND), Las Vegas(LAS), or Seoul (ICN).



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

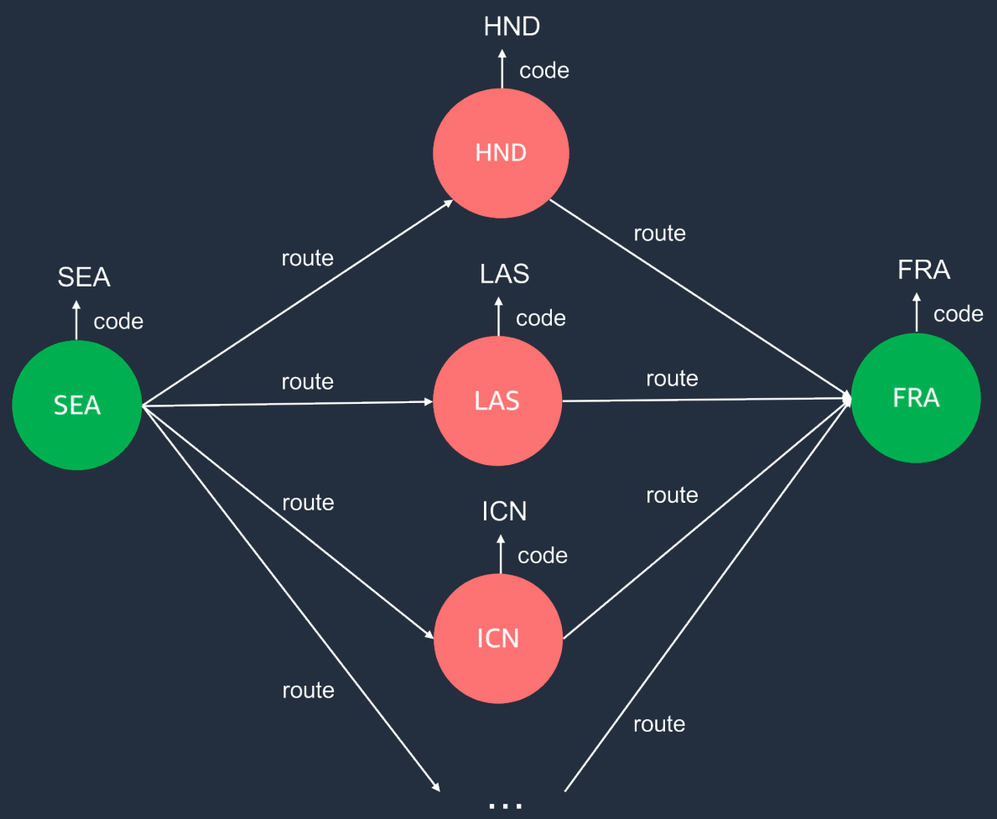
Neptune graph data model



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

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

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

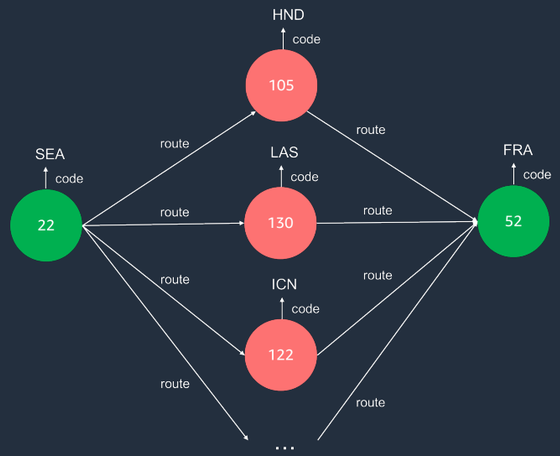
Efficient lookup whenever a prefix of the positions, such as the vertex (subject) or vertex and property identifier, is bound: *Find airport:SEA (22) with code "SEA"*

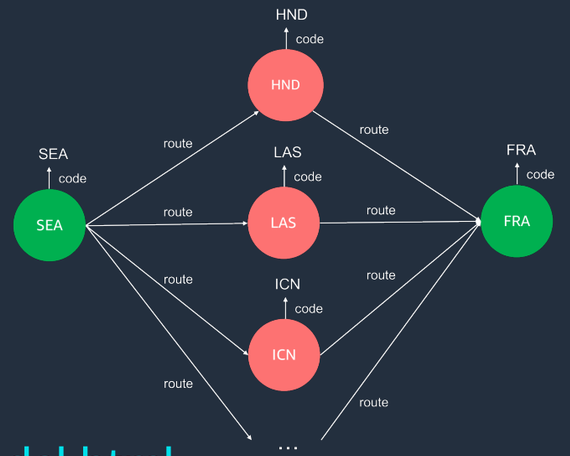
POGS – Uses a key composed of Predicate + Object + Graph + Subject

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"?*

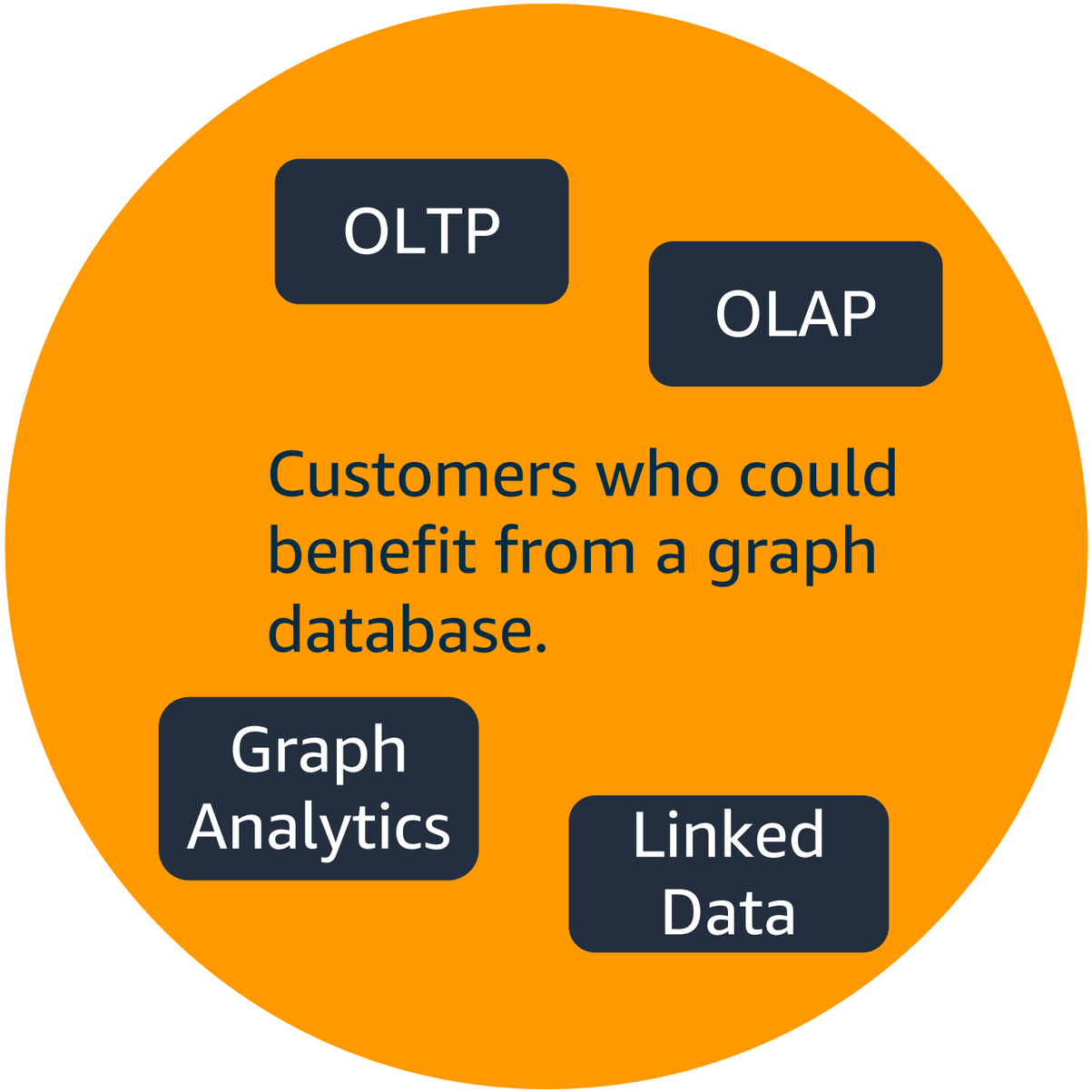




https://docs.aws.amazon.com/neptune/latest/userguide/feature-overview-data-model.html

How big is the graph market segment? What is it?

Customers who know they want a graph database.



Customers want graph and have use cases across graph data models.

- Many customers want both PG and RDF.
    - Developer team preference
    - Existing applications

- Semantic alignment / data canonicalization at an organization level (RDF – typically)
    - Business Applications (PG – often)
    - Graph Analytics

Customers also want to…

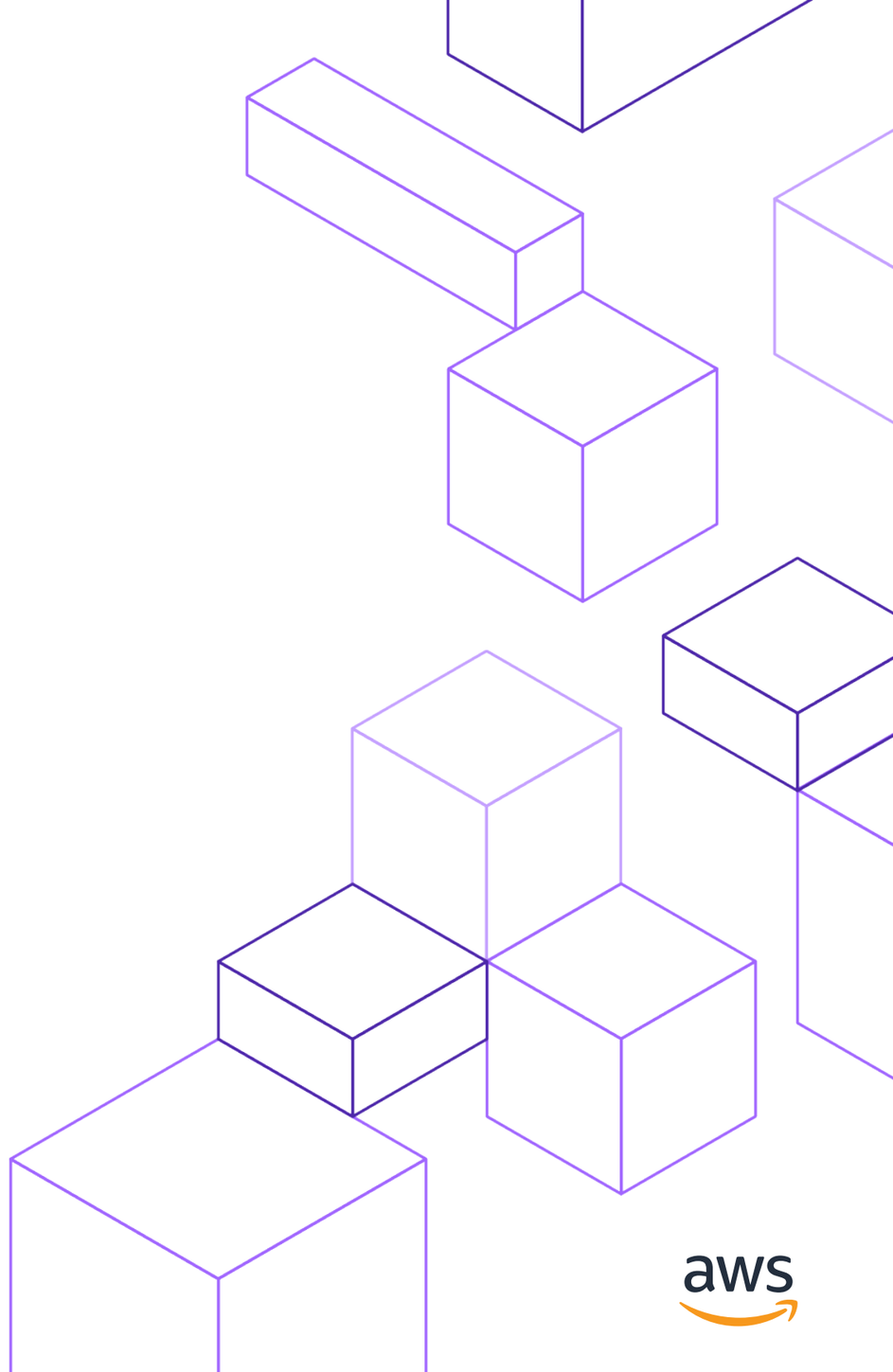
- Understand how to model their data in a graphs
- Query graph data
    - *Which PG query language should I use?*
    - *I can’t query my PG data with SPARQL.*
    - *I have RDF Linked Data I want to use in my PG application.*
- Exchange data
    - Between property graph applications
    - RDF->PG and PG->RDF (like RDF* and SPARQL*)
    - Help to conceptualize graphs in RDF

Building approaches that enable commonality enables a larger set of graph users.

- A rising graph floats all nodes, edges, and properties (or resources).
    - Make it easier for customers to model their data as a graphs.
    - Make graph query that works across models more accessible to a broader set developers.
    - Give customers a way to interchange data between PG and RDF.

It’s just graph; let’s make it that way for graph application builders.

Q&A



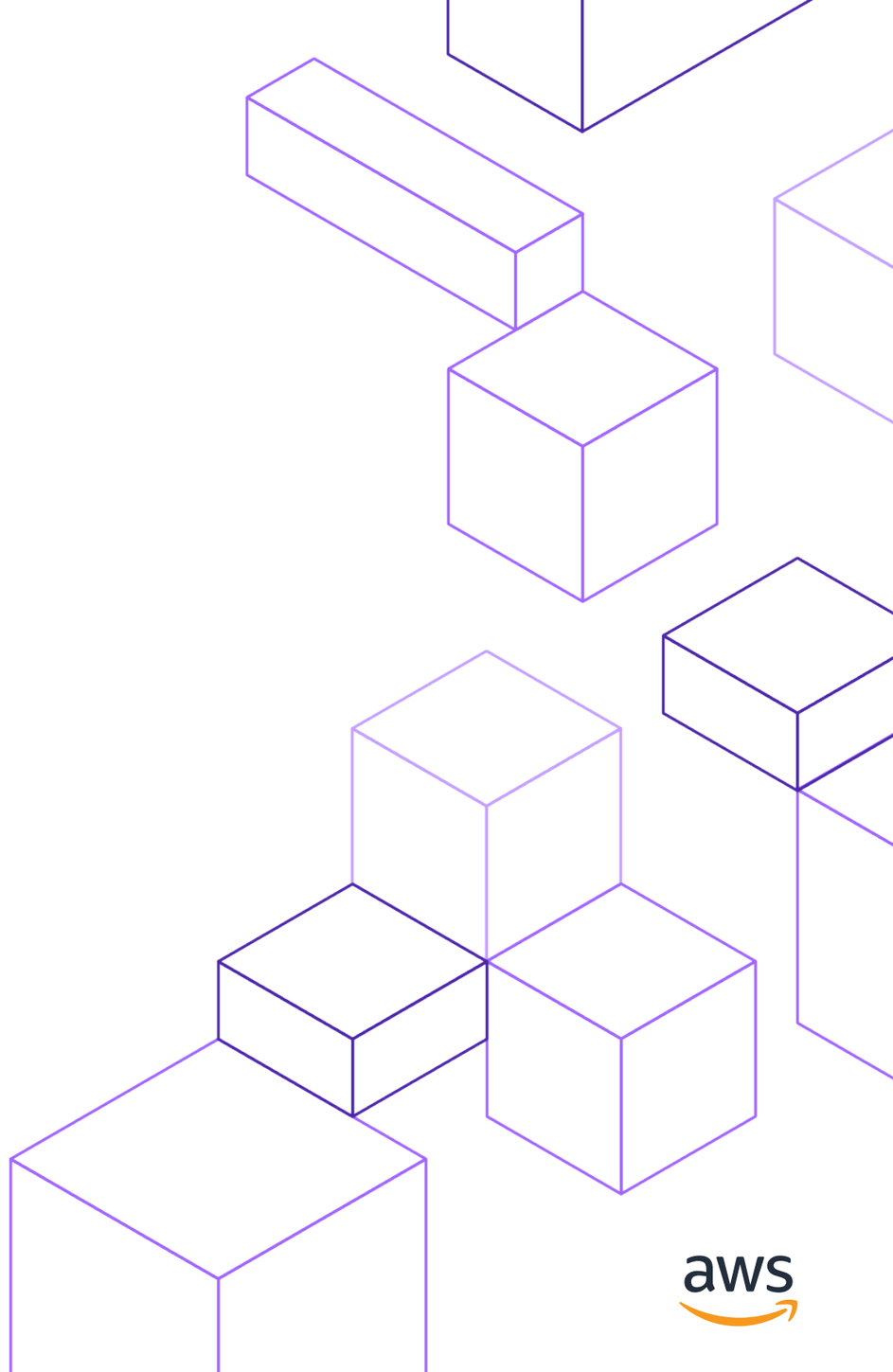
Thank you!

Brad Bebee

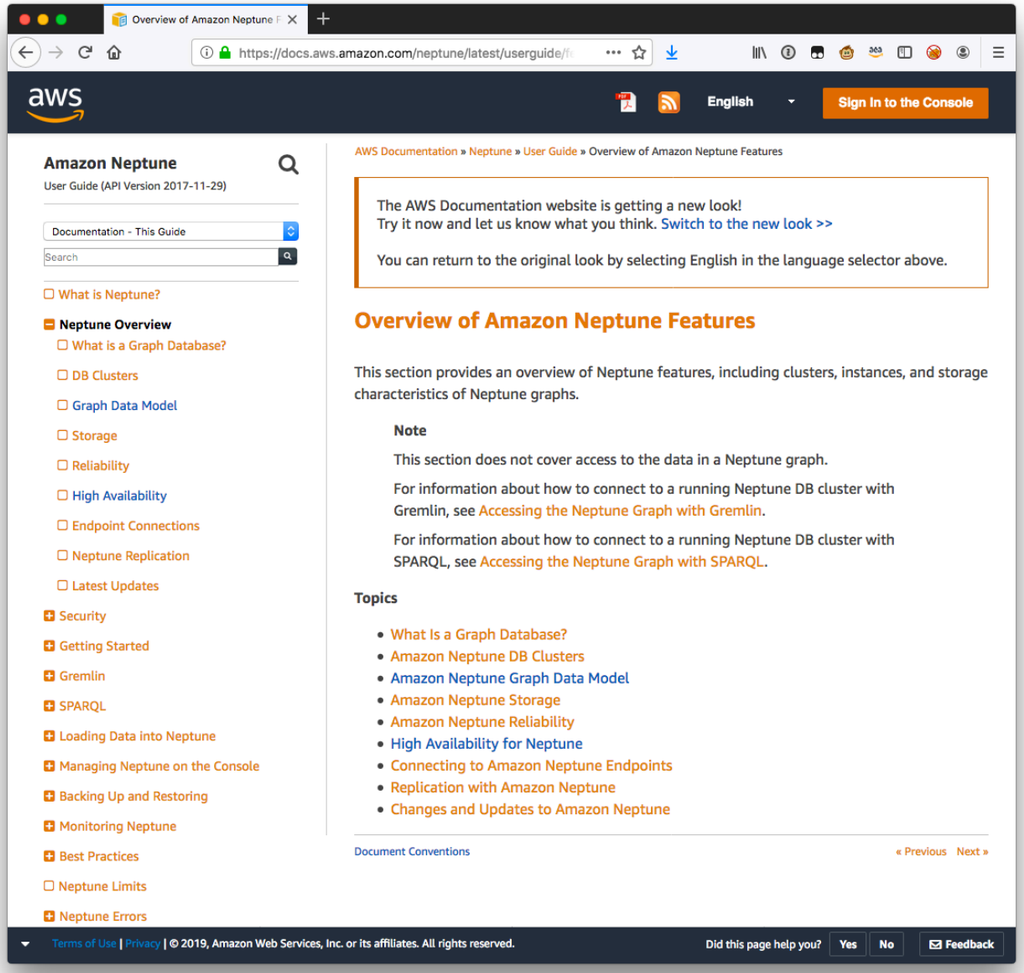
beebs@amazon.com

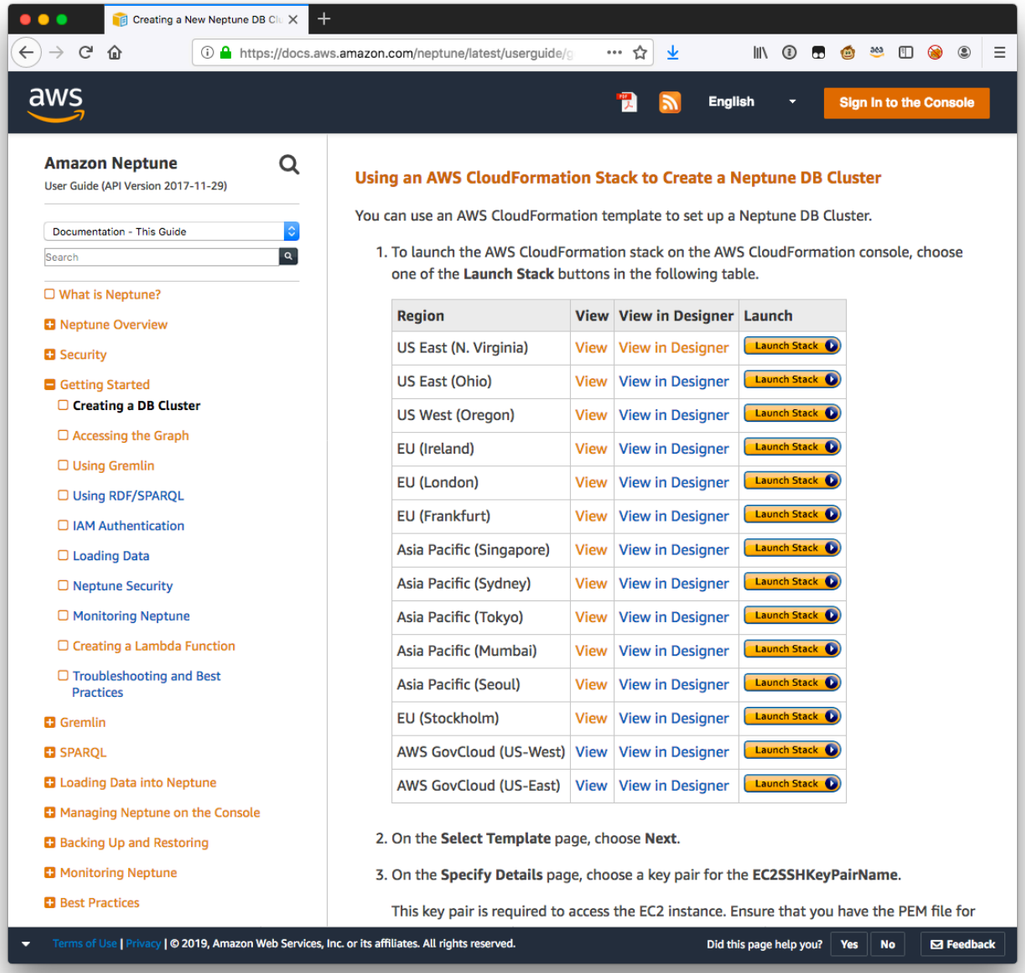
@b2ebs

Resources



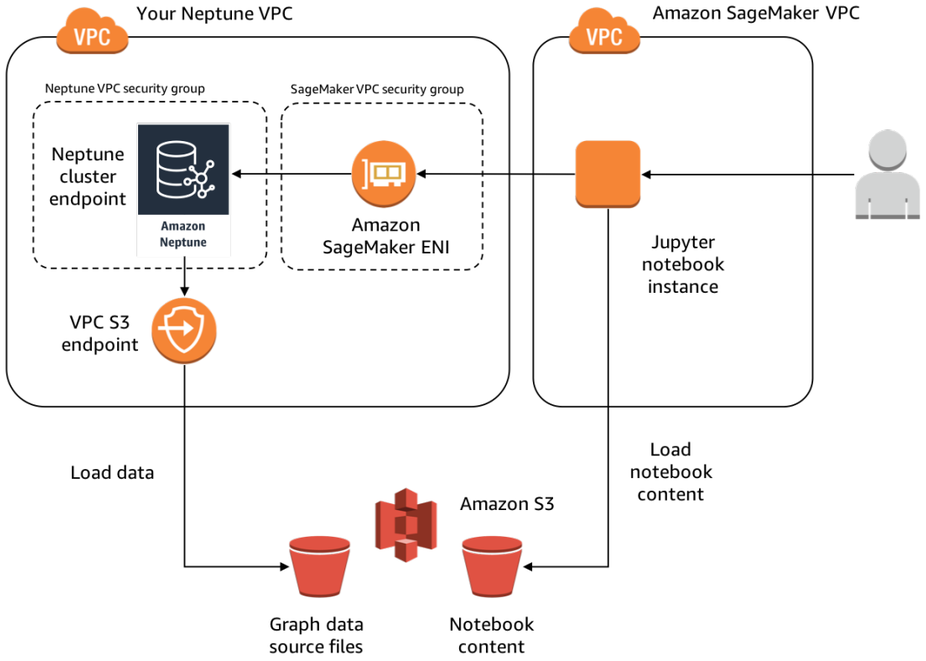
Documentation

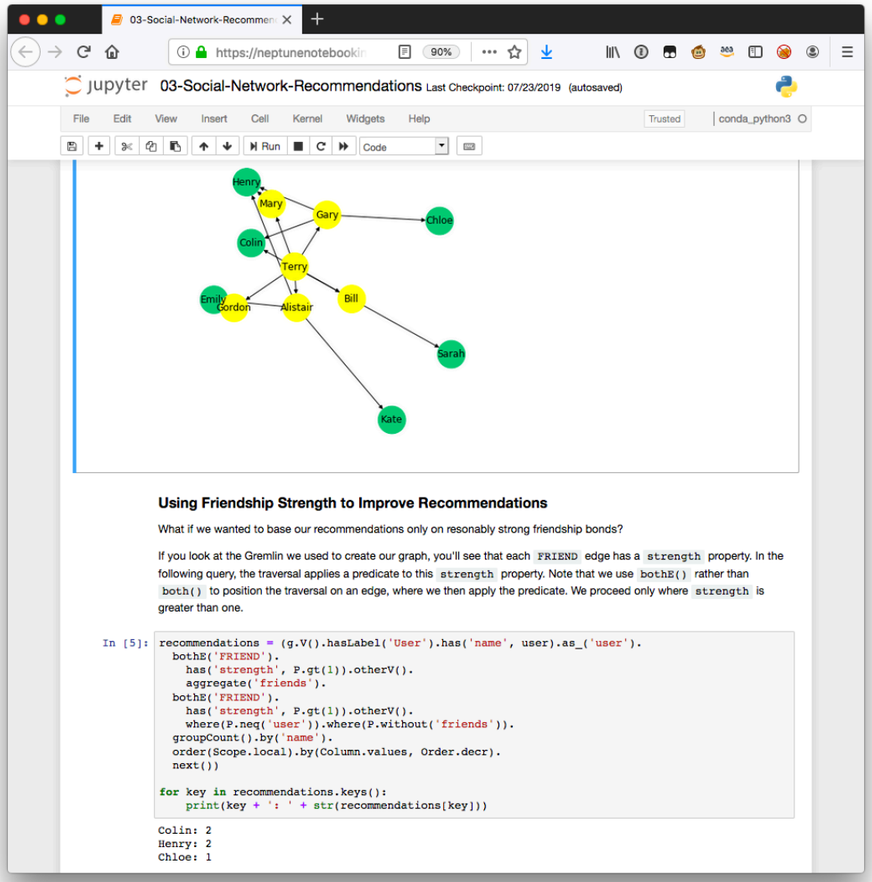




Explore use cases using Neptune

https://aws.amazon.com/blogs/database/analyze-amazon-neptune-graphs-using-amazon-sagemaker-jupyter-notebooks/

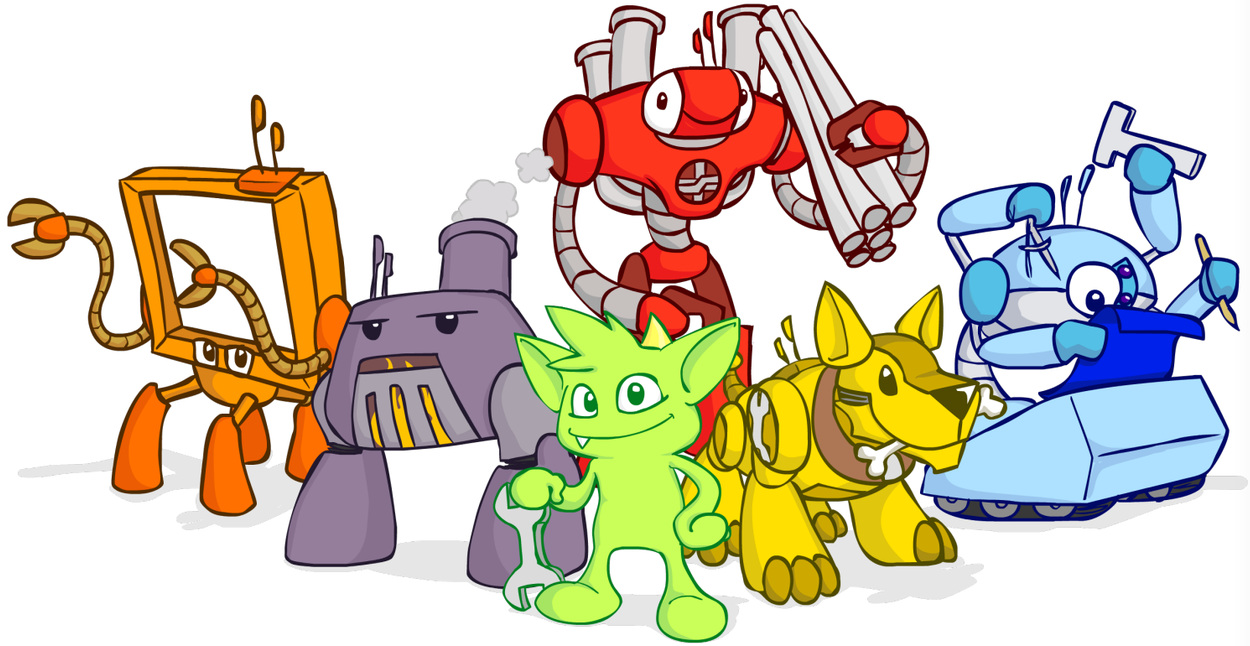


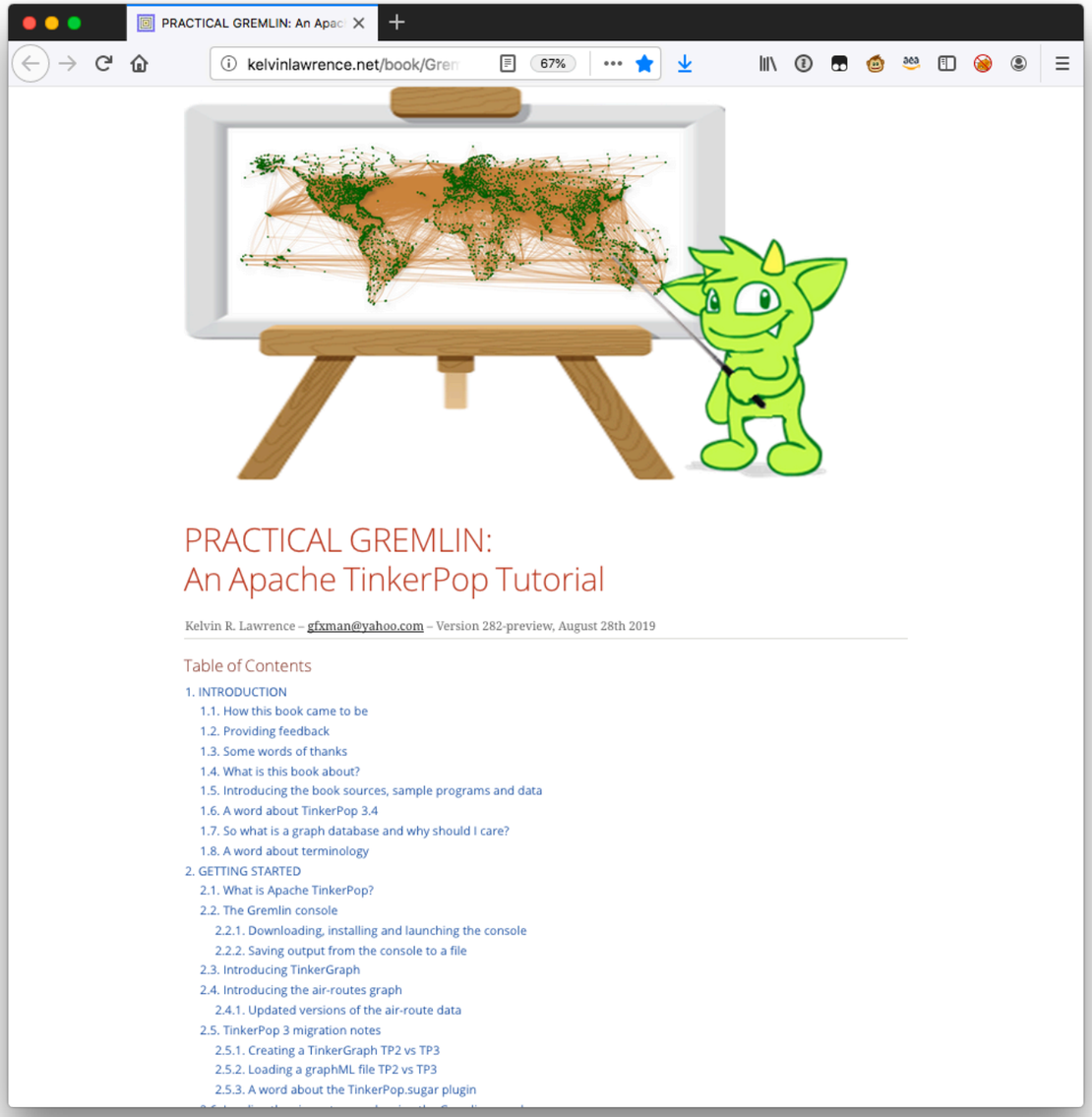


Learn Gremlin

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

https://github.com/krlawrence/graph

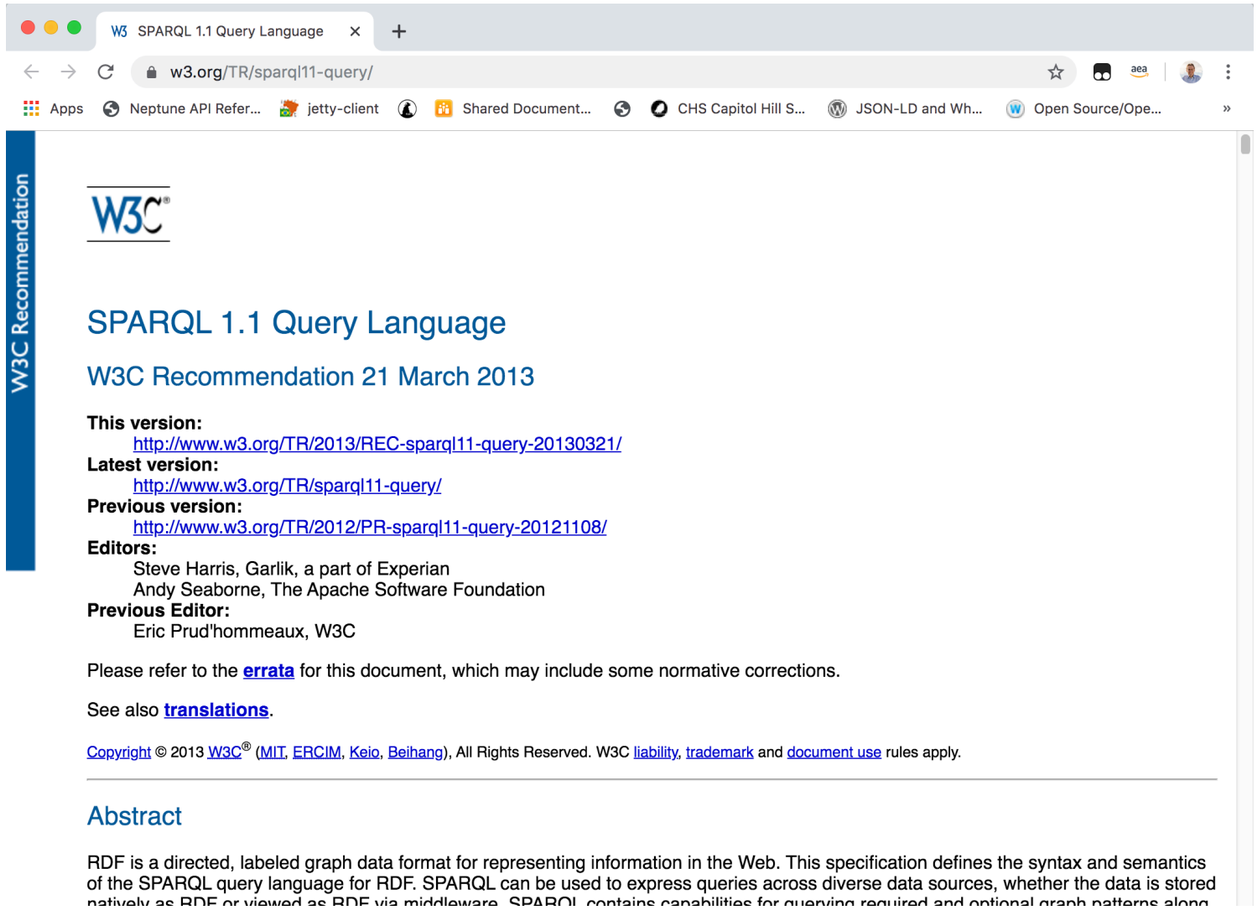




Learn SPARQL

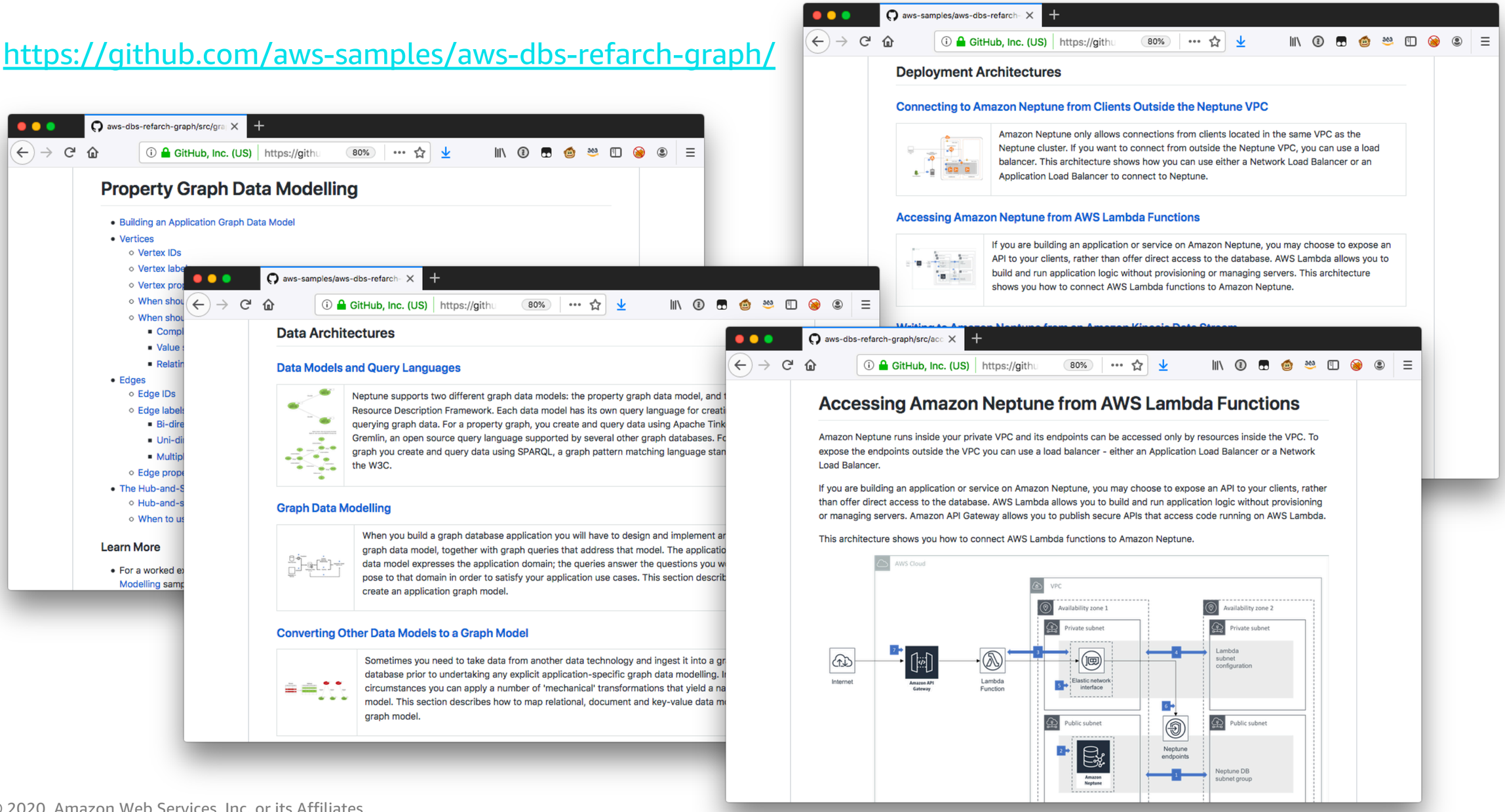
https://logd.tw.rpi.edu/tutorial/a_crash_course_on_sparql

https://www.w3.org/TR/sparql11-query/



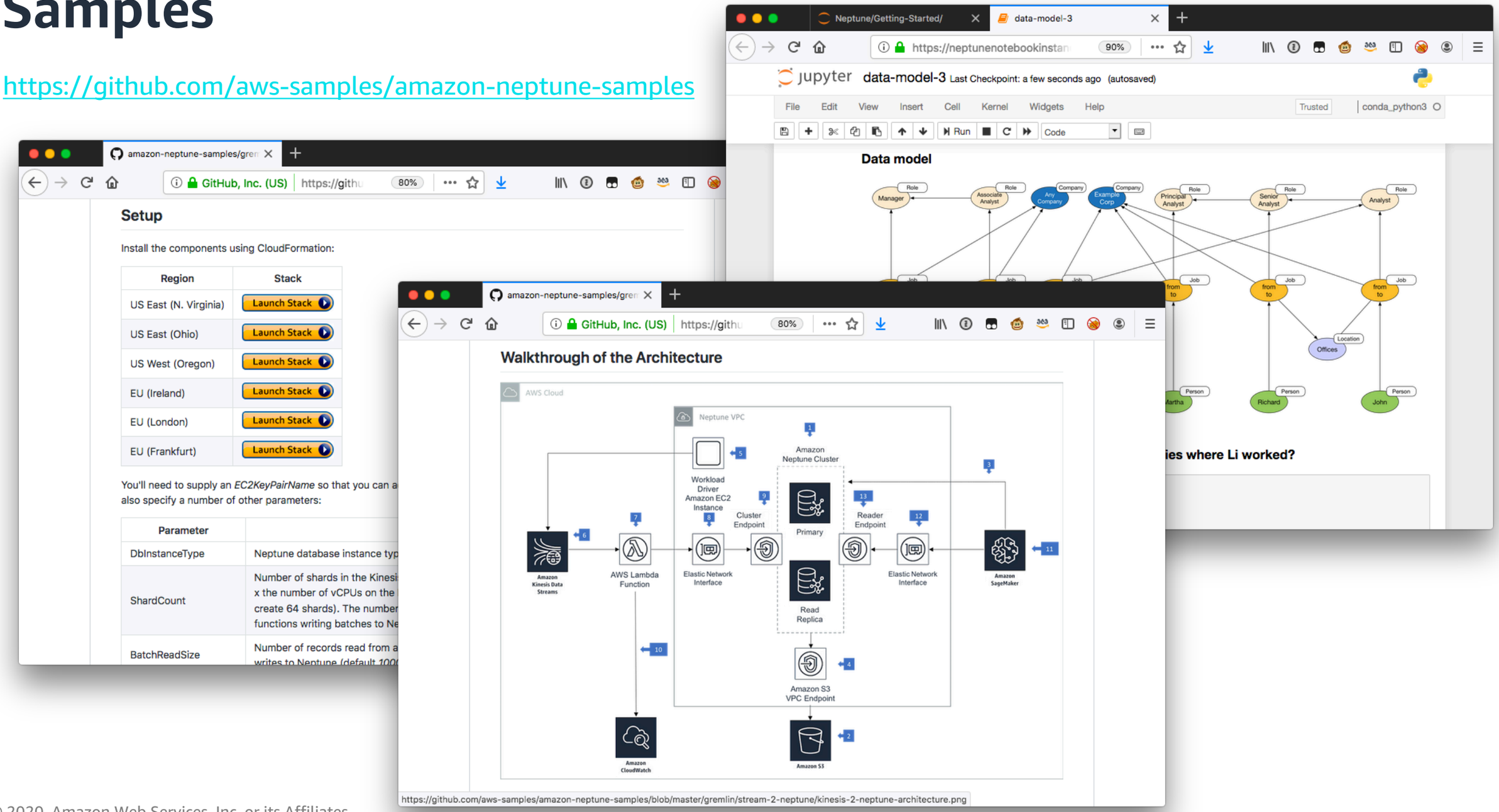
Reference architectures

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



Samples

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



Use cases, videos, blog posts, and code

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