Let Me Graph that For You

Amazon Neptune Deep Dive & Workshop

Justin Thomas

Principal Business Development Manager

Agenda

- What is a Graph?
- Why use a Graph?
- Why use a Graph Database?
- Why use Amazon Neptune
- Demos
- Q&A

What is a Graph?

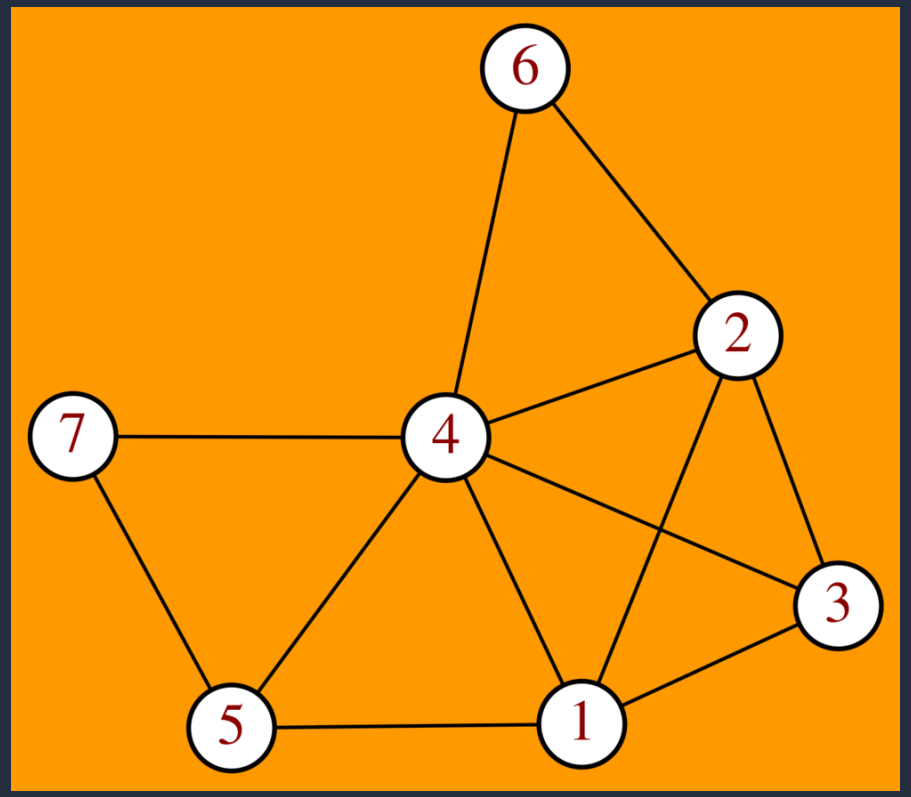
Graphs

Model data based on relationships

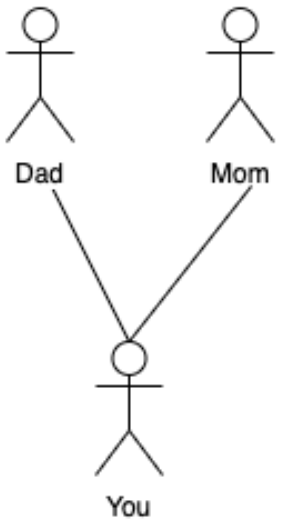
Explore connections and patterns in connected data

Graph fundamentals

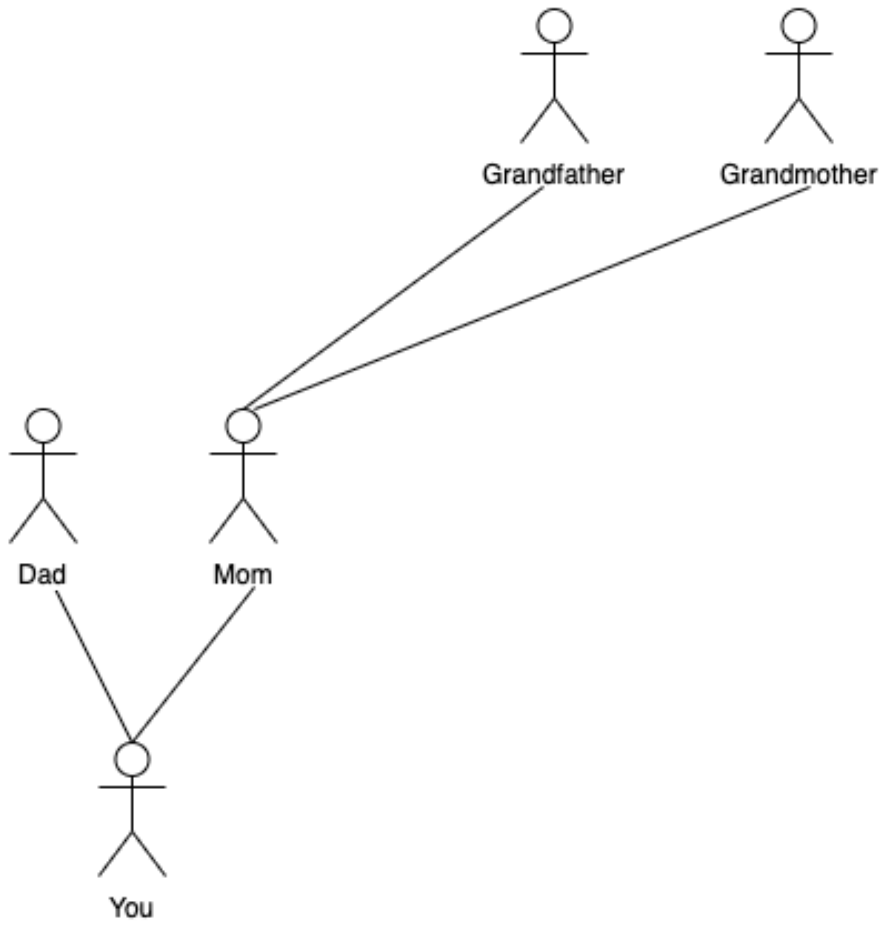
- Vertices or Nodes
- Edges or Relationships
- Edges can have direction



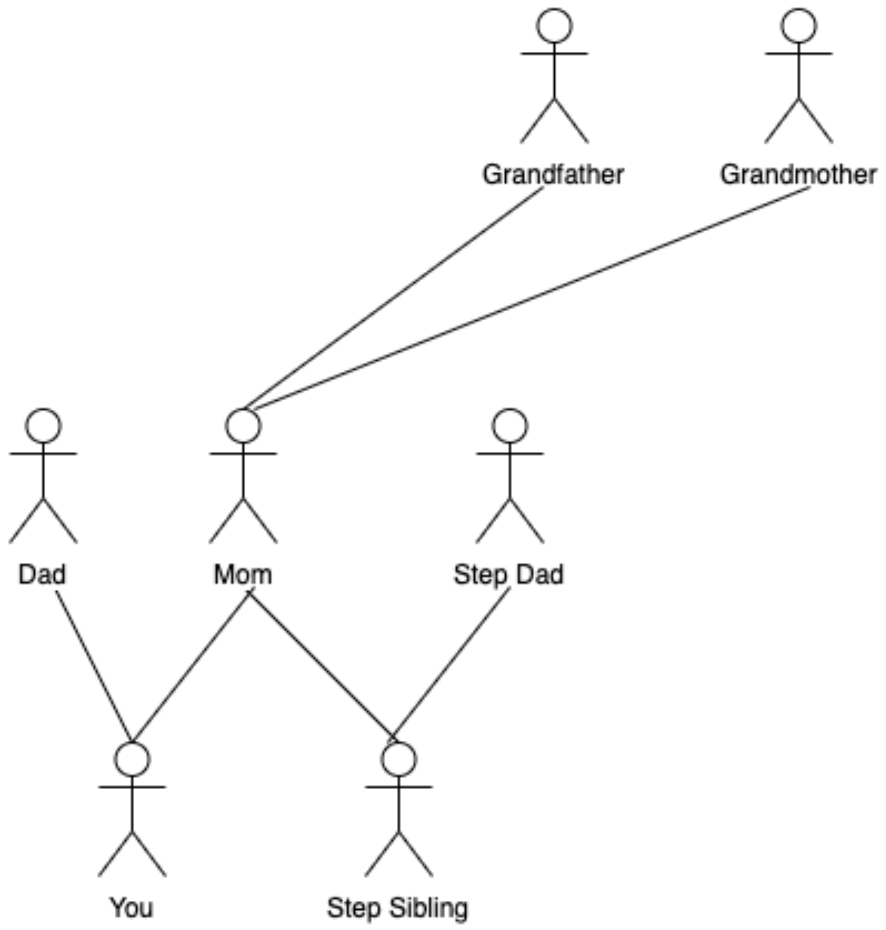
Graphs are intuitive



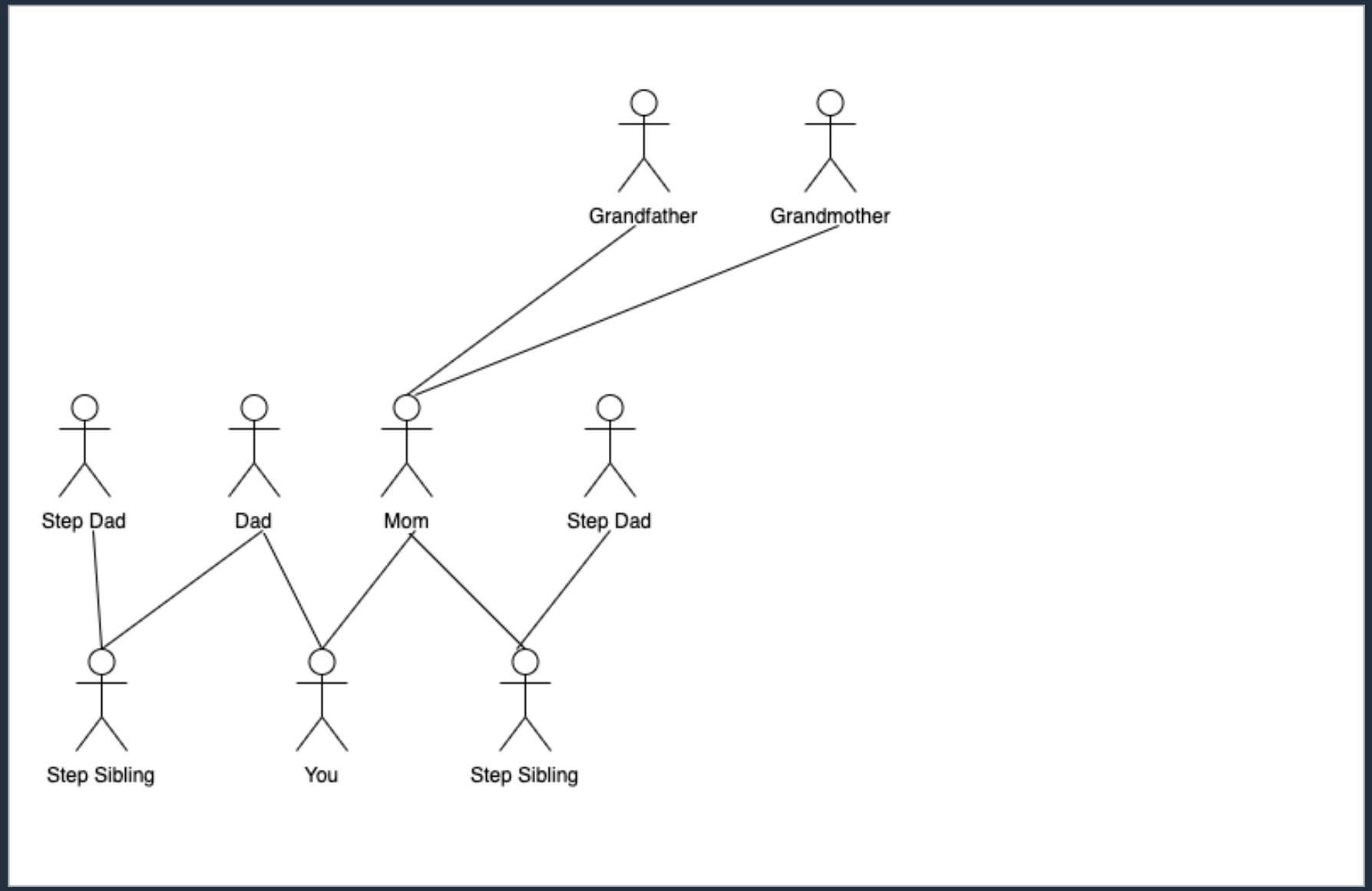
Graphs easily represent connections



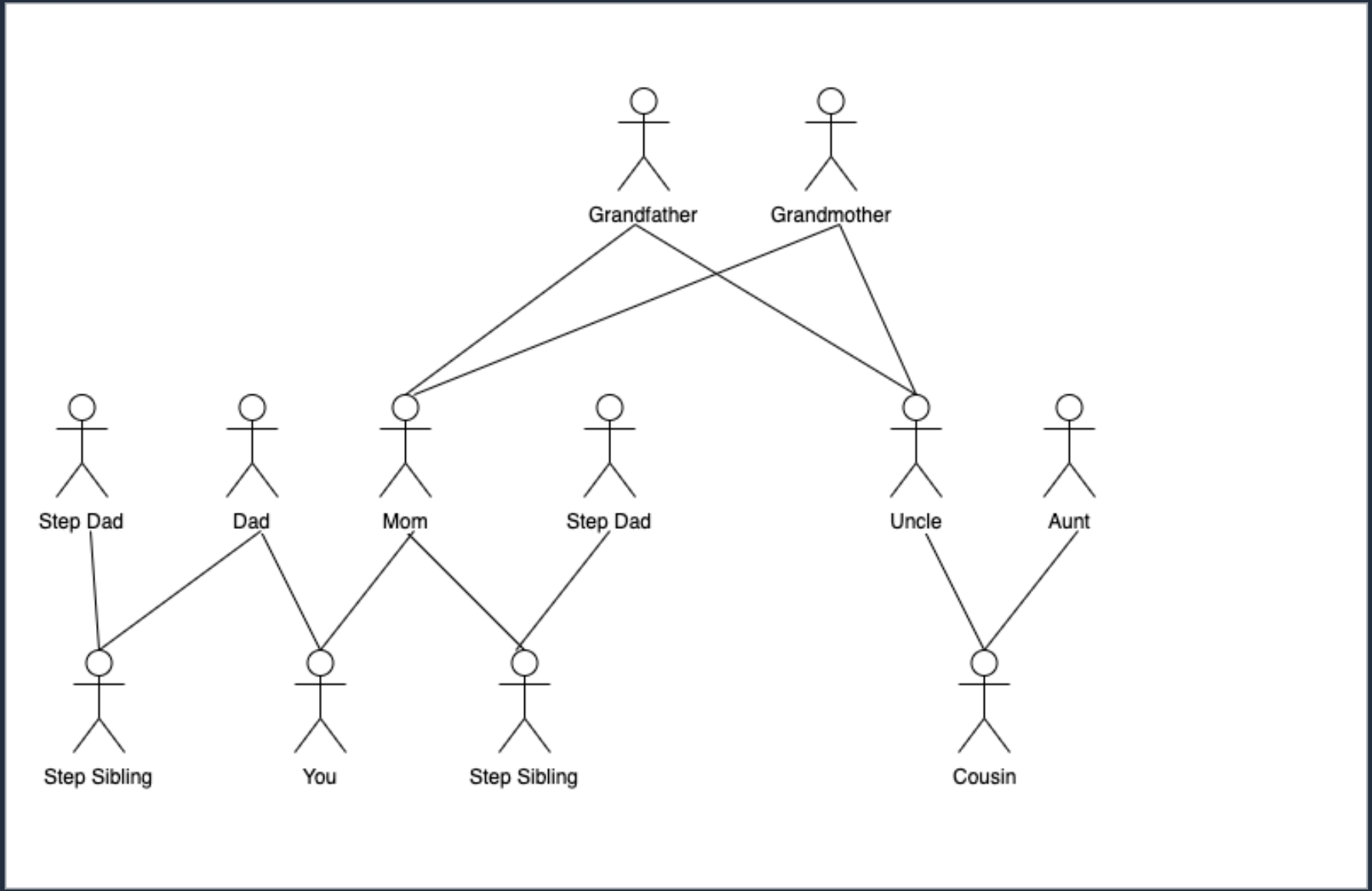
Sometimes the connections are complex



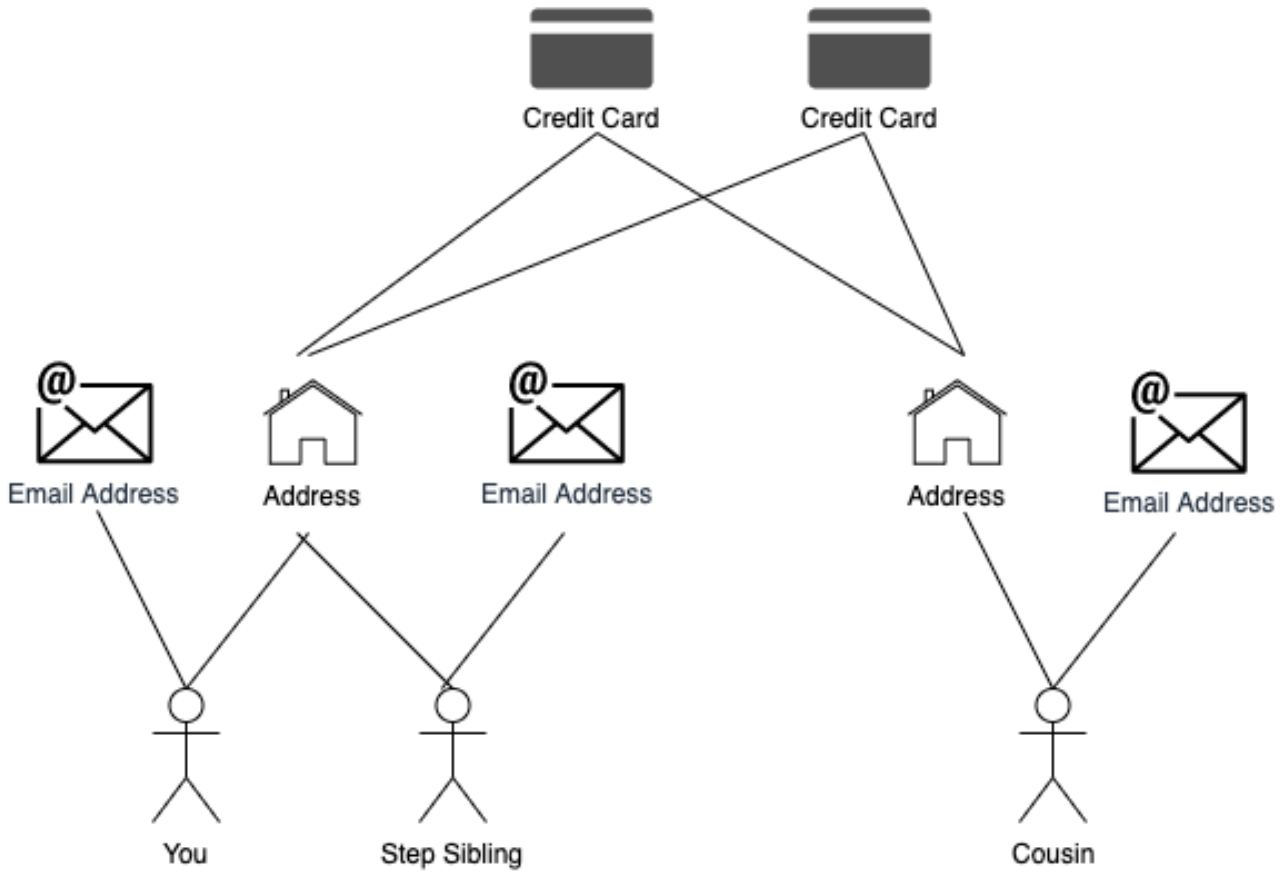
And varied



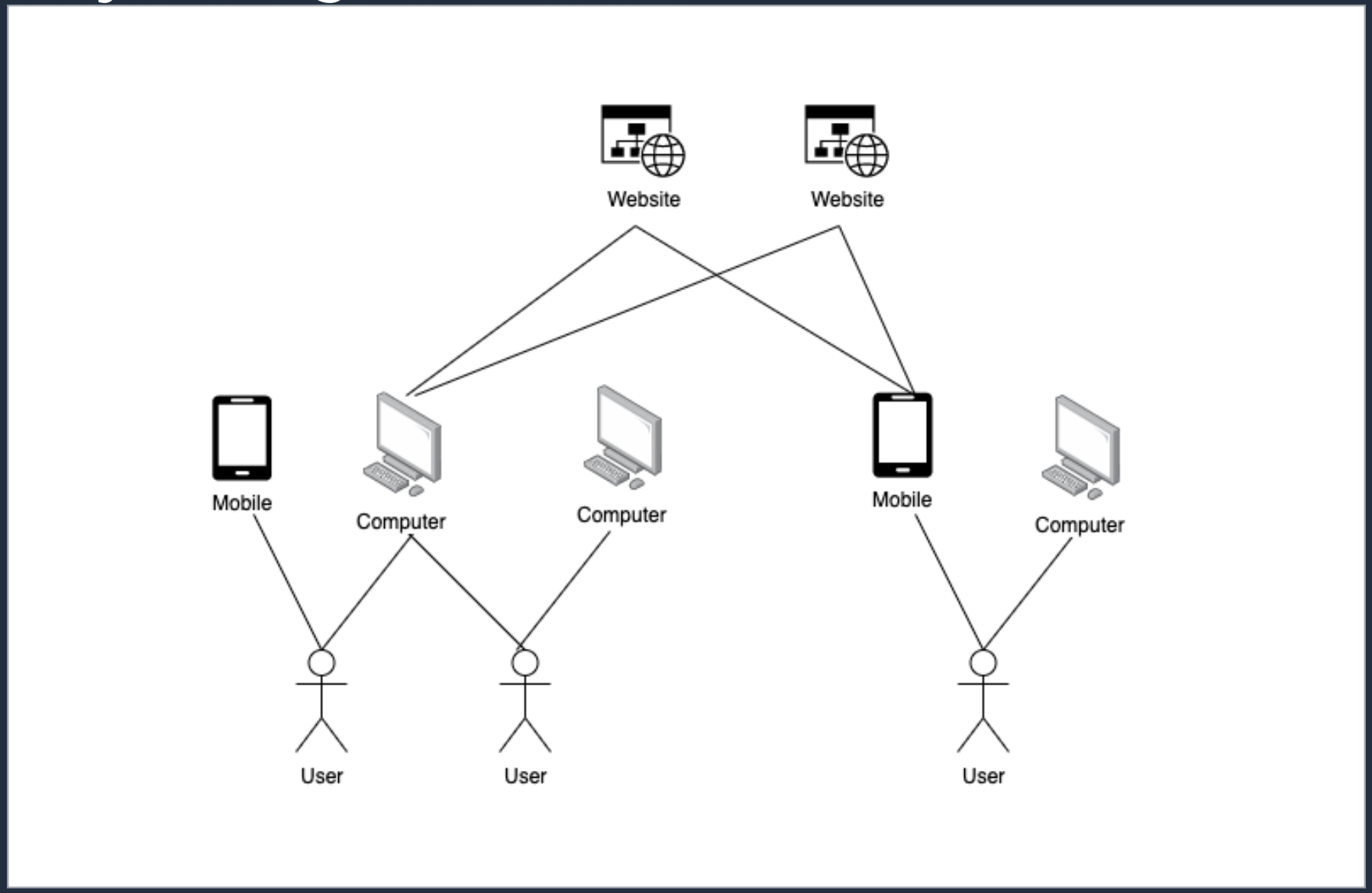
But they represent the world as it is



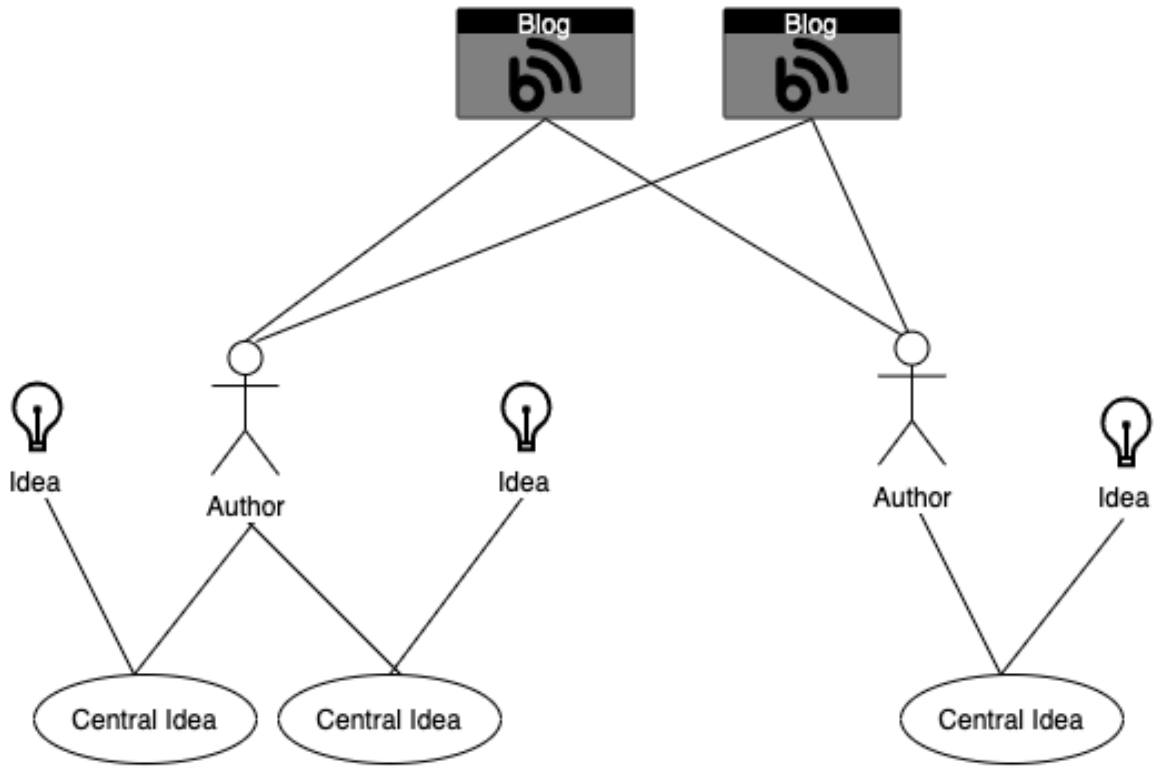
Graphs apply across domains, like Fraud



Or Identity Management



Or Knowledge Graphs



Technical challenges graphs help solve

- Combining data across data silos
- Finding common connections or paths
- Working with heterogenous data with complex relationships
- Data full of many-to-many relationships

It’s the difference between using multiple excel spreadsheets or a mind map tool to think about your data.

Why use a Graph?

Common graph business problems

- We need to better at detecting fraud
- My customers want better or more personalized recommendations
- We need to connect our siloed data sources
- We have multiple websites/applications and we need to link customer identities in these systems
- Our machine learning algorithms need improved

Not so easily recognized graph problems

- Where are the risks in my IT Infrastructure/supply chain?
- Where did this data it come from?
- Why don’t my search results relate to my question?
- How does person X have access to information Y?
- How will this price curve change impact my application?

Graphs solve the Where, Why, and How

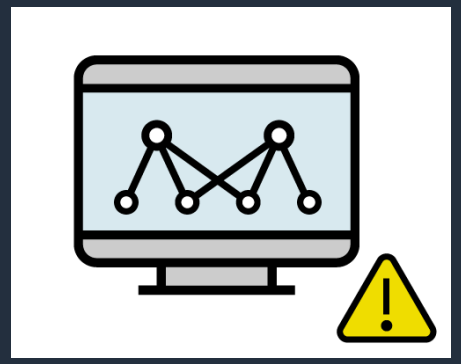
- **Where** are the risks in my IT Infrastructure/supply chain?
- **Where** did this data come from?
- **Why** don’t my search results relate to my question?
- **How** does person X have access to information Y?
- **How** will this price curve change impact my application?

aws

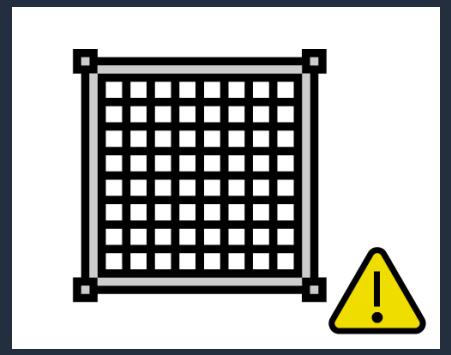
Why use a Graph Database?

Challenges with highly connected data

Unnatural for querying



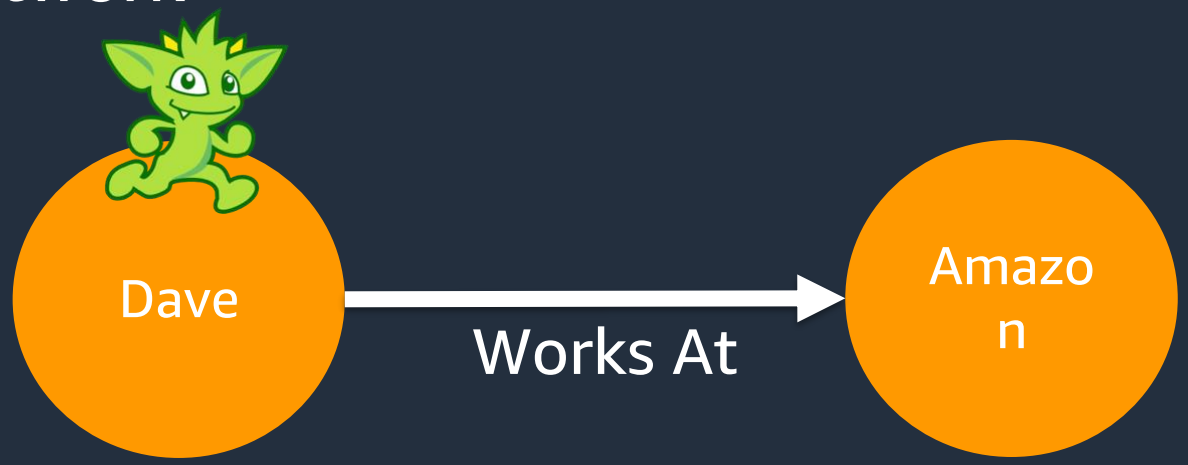
Inefficient processing



Rigid schema inflexible for changing data

Query languages- Designed to move through data

Graphs query languages are optimized to use connections to move through a network



Relational queries work by combining sets of data.

Person
Dave

X

Company
Amazon

=

Person	Company
Dave	Amazon

Efficient Processing

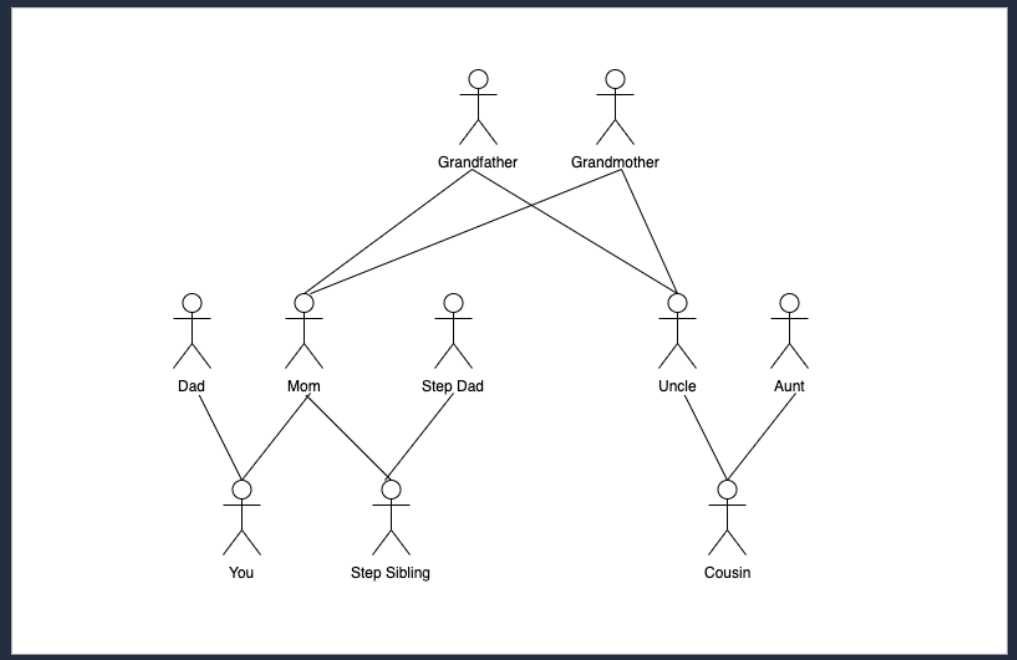
Let's look at the example:

Justin – Works At -> Amazon

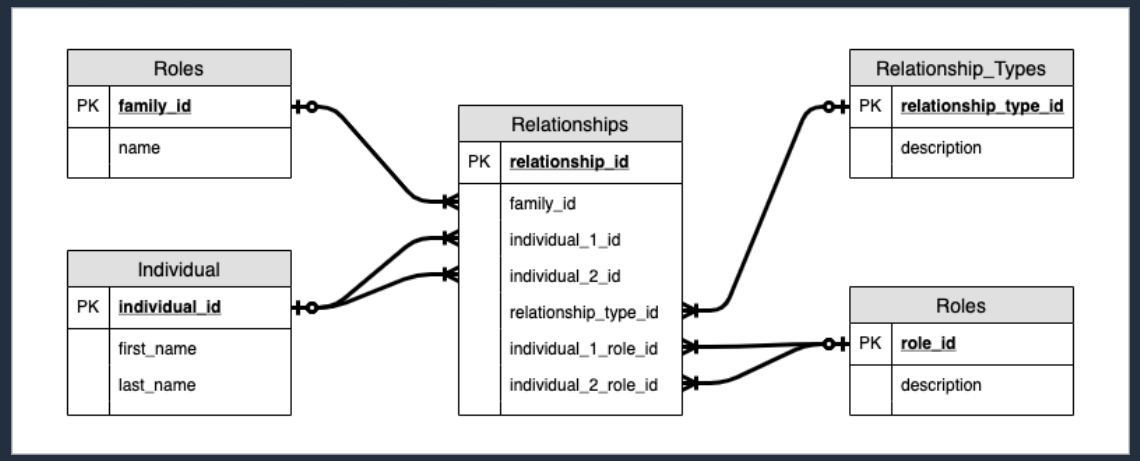
In a Graph database, the ***Works At*** connection is data, when needed the connection is retrieved

In a Relational databases, the ***Works At*** connection is metadata, when needed the connection must be calculated.

Schema flexibility makes adding new data easy



vs



Bonus - Graphs are easier to understand by new and/or non-technical people

Graph use cases are about the questions being asked

Graph databases excel at answering questions like:

- What does this person want to buy?
- How are these two people connected?
- Why did X impact Y?

These questions:

- Navigate (variably) connected structure
- Filter or compute a result on the basis of the *strength*, *weight,* or *quality* of relationships
- Require traversing an unknown number of connections

Other databases solve other questions better

Other databases excel answering to questions like, “How Much”, “Count these items”, ”Find X based on Y”

- What were my sales for April 2021?
- How many people bought product X in Anchorage?
- Find me everyone named ”Dave” in “Alaska”?
- Show me everyone who works at “Amazon”?
- Give me the value for key X?

Graph is not an Island - Purpose-built databases

	Relational	Key-value	Document	In-memory	Graph	Time-series	Ledger	Wide Column
	Referential integrity, ACID transactions, schema-on-write	High throughput, Low latency reads and writes, endless scale	Store documents and quickly access querying on any attribute	Query by key with microsecond latency	Quickly and easily create and navigate relationships between data	Collect, store, and process data sequenced by time	Complete, immutable, and verifiable history of all changes to application data	Scalable, highly available, and managed Apache Cassandra-compatible service
AWS Service(s)	Aurora, RDS	DynamoDB	DocumentDB	ElastiCache	Neptune	Timestream	QLDB	Keyspaces Managed Cassandra
Common Use Cases	Lift and shift, ERP, CRM, finance	Real-time bidding, shopping cart, social, product catalog, customer preferences	Content management, personalization, mobile	Leaderboards, real-time analytics, caching	Fraud detection, social networking, recommendation engine	IoT applications, event tracking	Systems of record, supply chain, health care, registrations, financial	Build low-latency applications, leverage open source, migrate Cassandra to the cloud

Graphs and other databases

Graph databases are frequently used in conjunction with other databases

- Other databases are often used to answer questions where there is a single distinct answer.
    - Whom bought this product?
    - What is the average rating of this movie?

- Graph are good at giving you answers that a range of answers.
    - What product is this person going to buy next?
    - What movie will this person like?

Why use Amazon Neptune?

Amazon Neptune

Fast, reliable graph database built for the cloud

Open

Supports Apache TinkerPop & W3C RDF graph models

Fast

Query billions of relationships with millisecond latency

Reliable

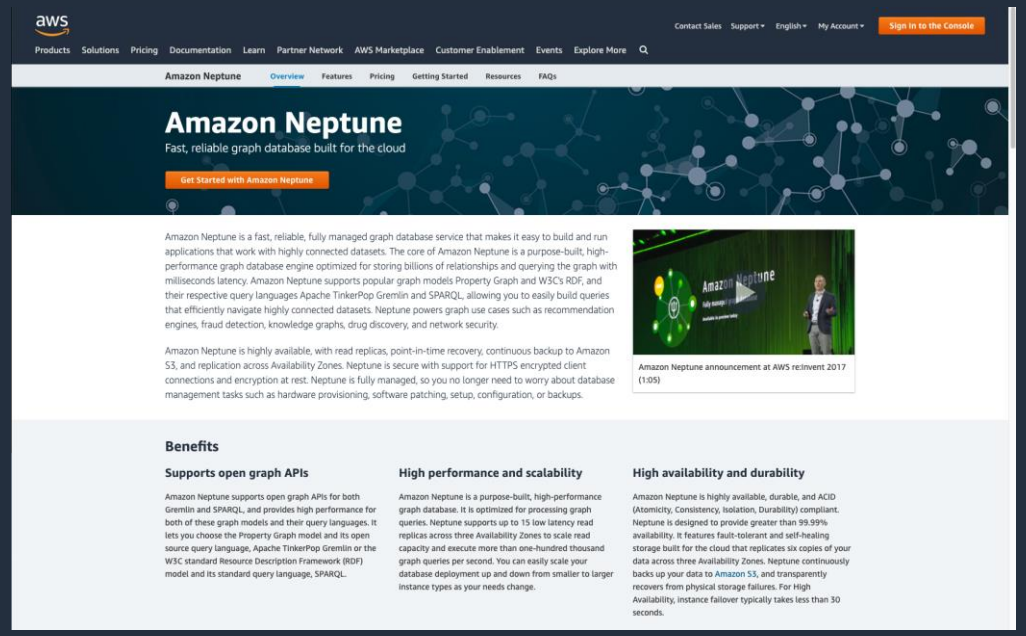
6 replicas of your data across 3 AZs with full backup and restore

Easy

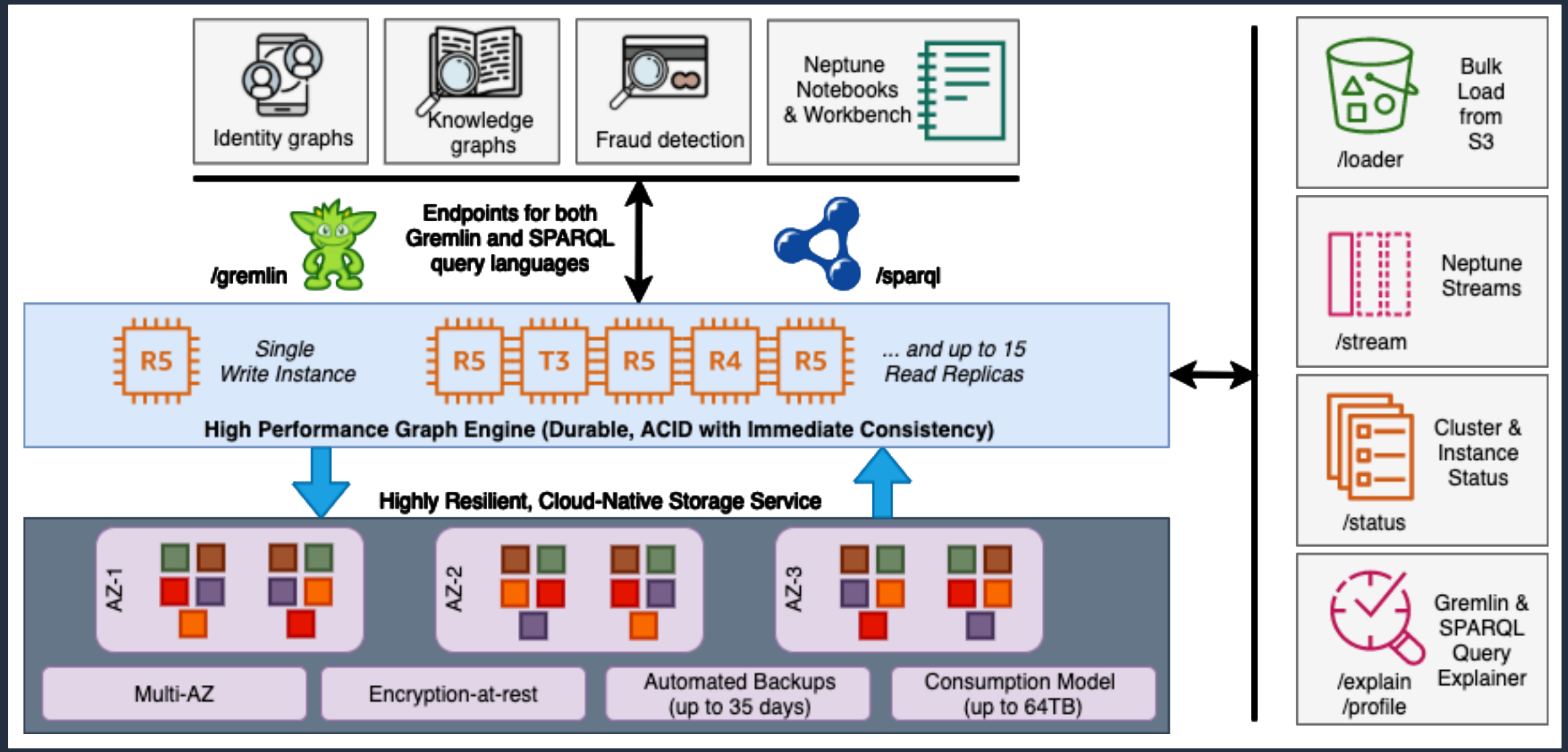
Build powerful queries easily with Gremlin and SPARQL

Neptune General Availability

- Announced on 5/30/2018
- 21 AWS Regions: US East (N. Virginia, Ohio), US West (Oregon, N. California), Canada (Central), Europe (Ireland, London, Frankfurt, Paris, Stockholm), Middle East (Bahrain), Asia Pacific (Mumbai, Singapore, Sydney, Seoul, Tokyo), China (Ningxia), GovCloud (East, West)
- ISO, HIPAA, SOC, PCI/DSS Compliance Certifications



Amazon Neptune High Level Architecture



Leading graph models and frameworks

PROPERTY GRAPH

Open Source Apache TinkerPop™

Gremlin Traversal Language

Programming language drivers

Apache TinkerPop

Gremlin

$G = (V, E)$

RESOURCE DESCRIPTION FRAMEWORK (RDF)

W3C Open Standard

SPARQL Query Language

W3C Semantic Web

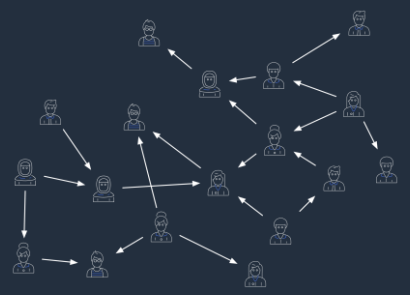
RDF

SPARQL

aws

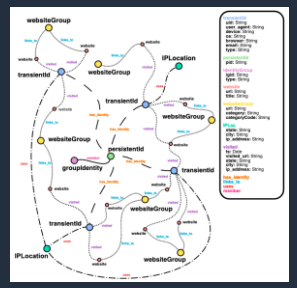
Common Neptune workloads

Social networks



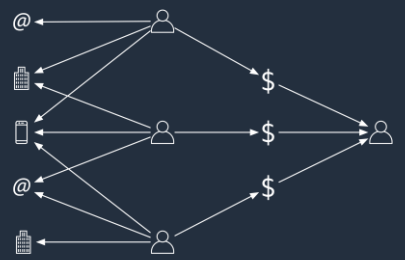
Social connections between friends and colleagues

Identity graphs



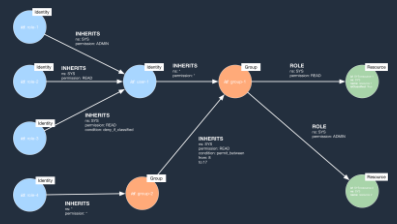
Unified view by linking devices, IP addresses, & browsing behaviors

Fraud detection



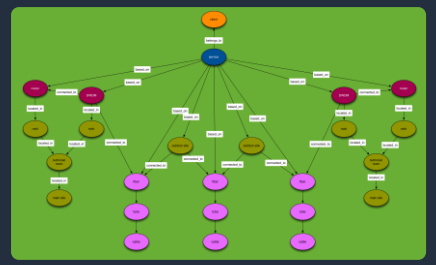
Patterns of connected transactions, senders, receivers, and identity info

Entitlements



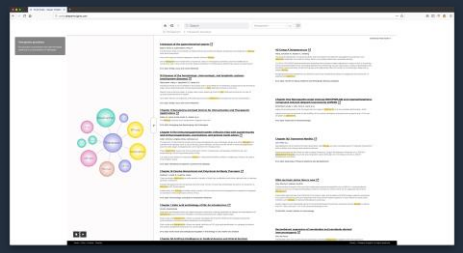
Conditional path from identities to resources

Network analysis



Top-down and bottom-up parent-child traversals

Knowledge graphs



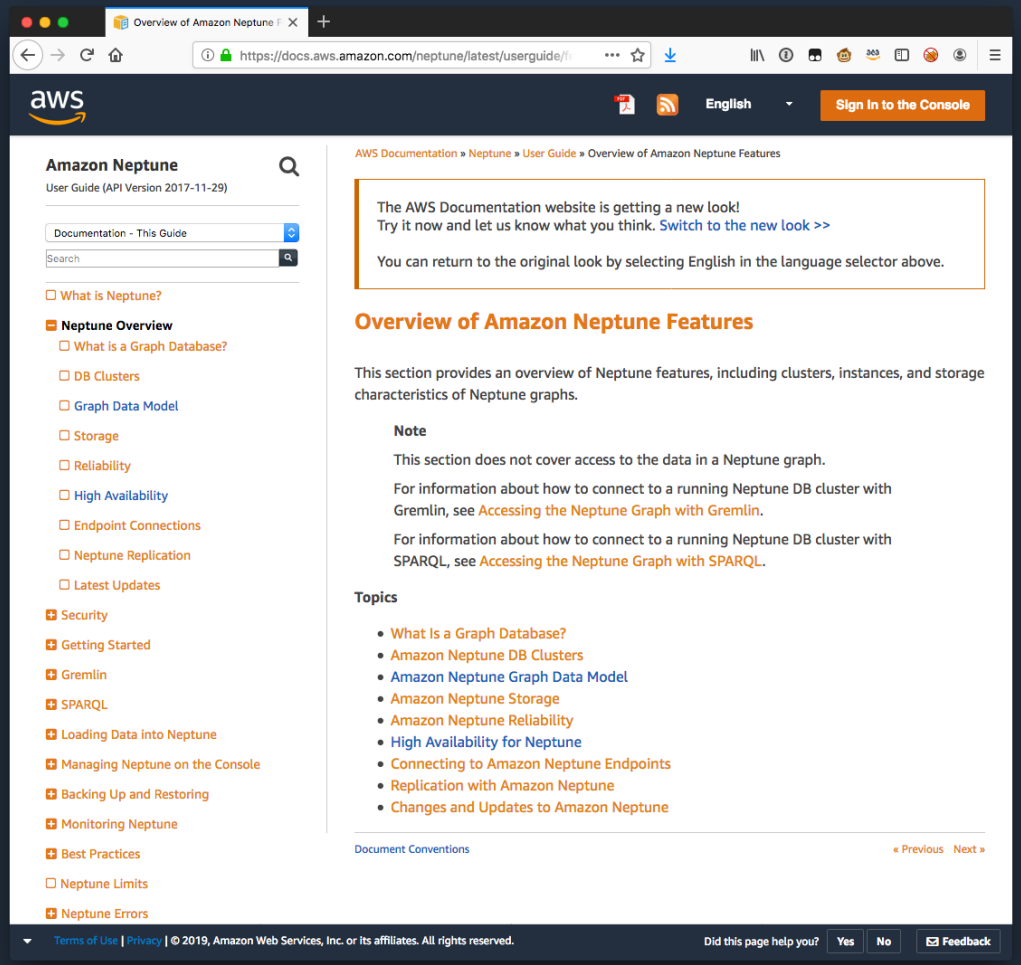
Context created by linked datasets and concepts

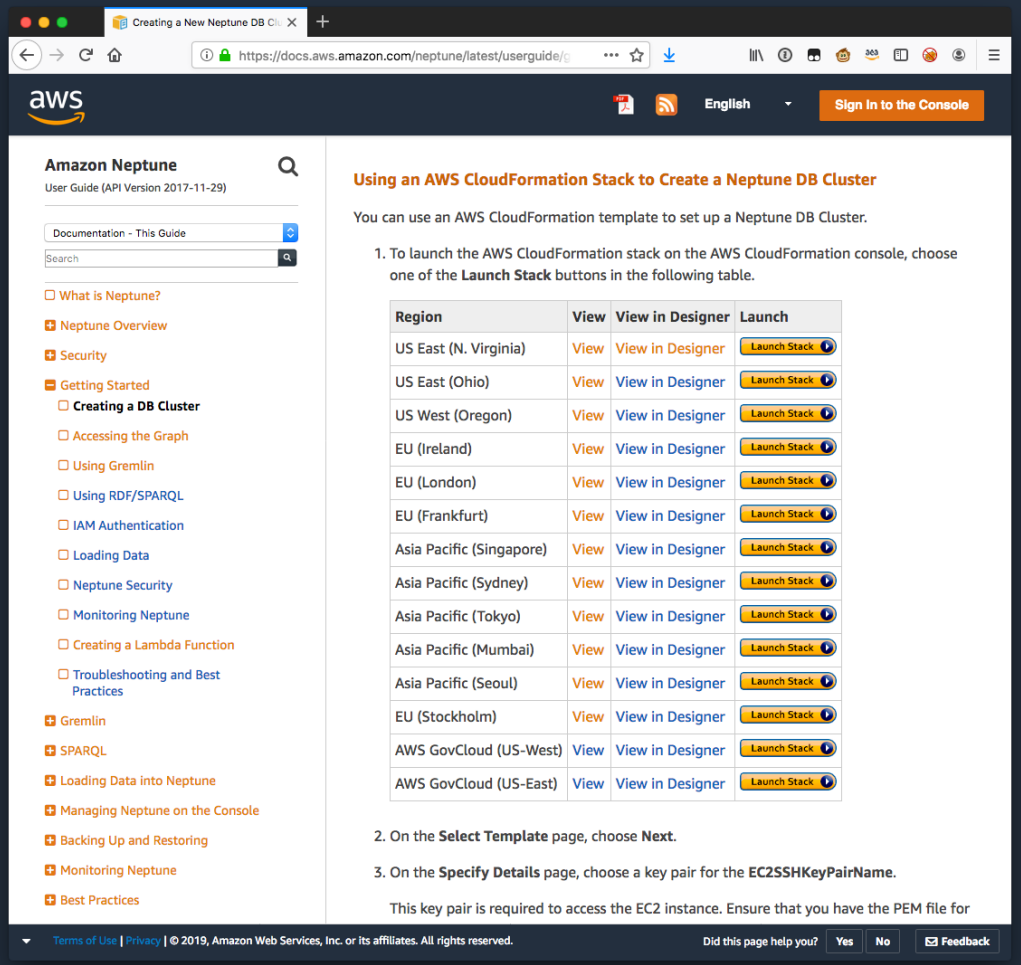
Demos

Resources

Documentation

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





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
- Edges
  - Edge IDs
- Learn More

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.

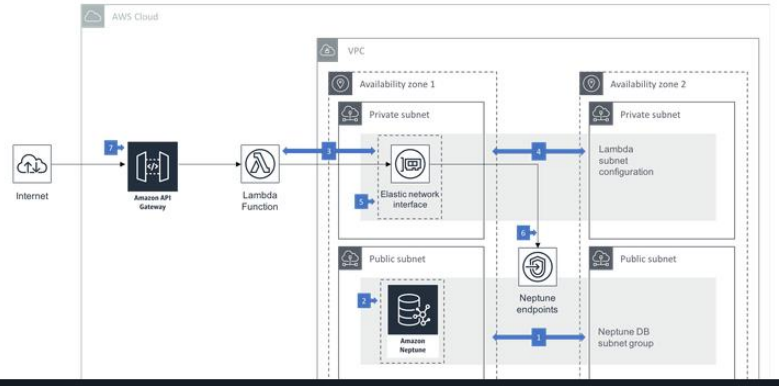
Writing to Amazon Neptune from an Amazon Kinesis Data Stream

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.

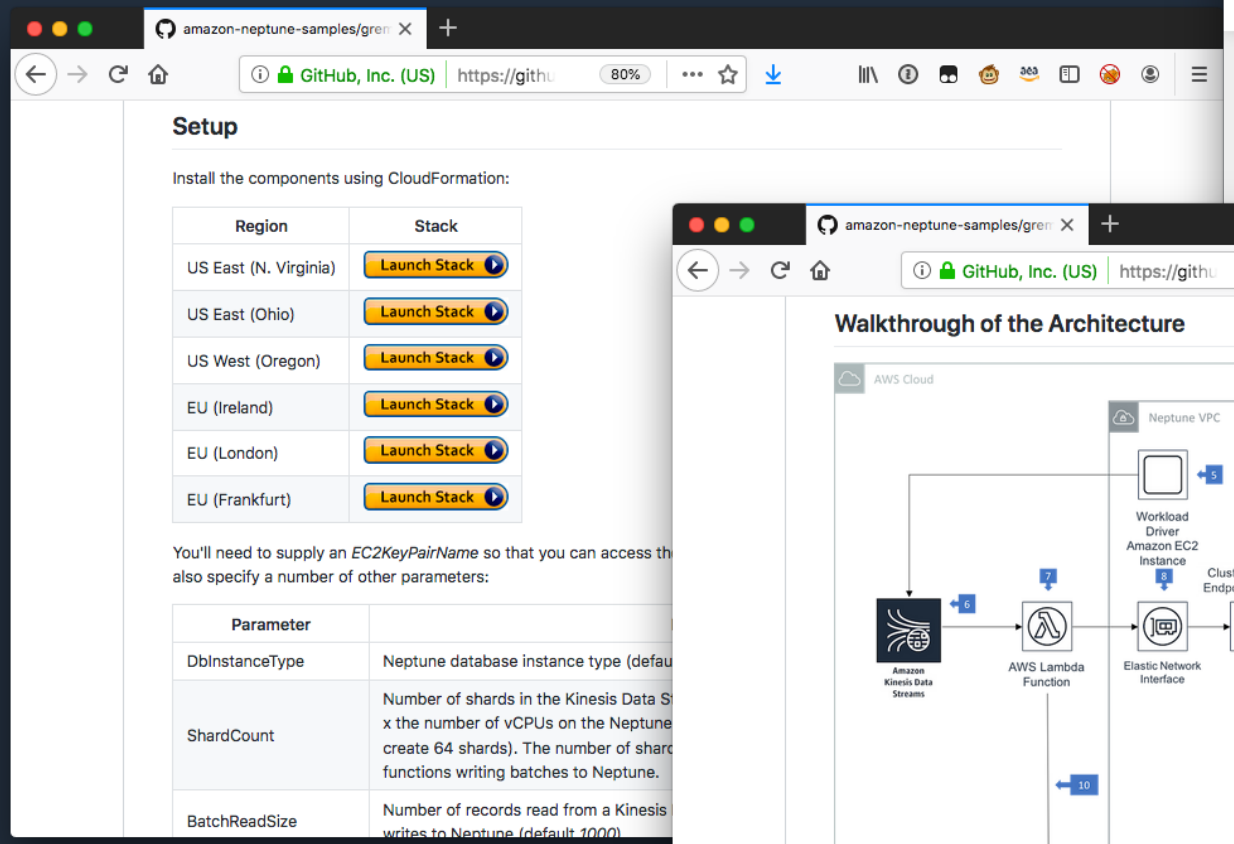
If you are building an application or service on Amazon Neptune, you may choose to expose an API to your clients, rather than offer direct access to the database. AWS Lambda allows you to build and run application logic without provisioning or managing servers. Amazon API Gateway allows you to publish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

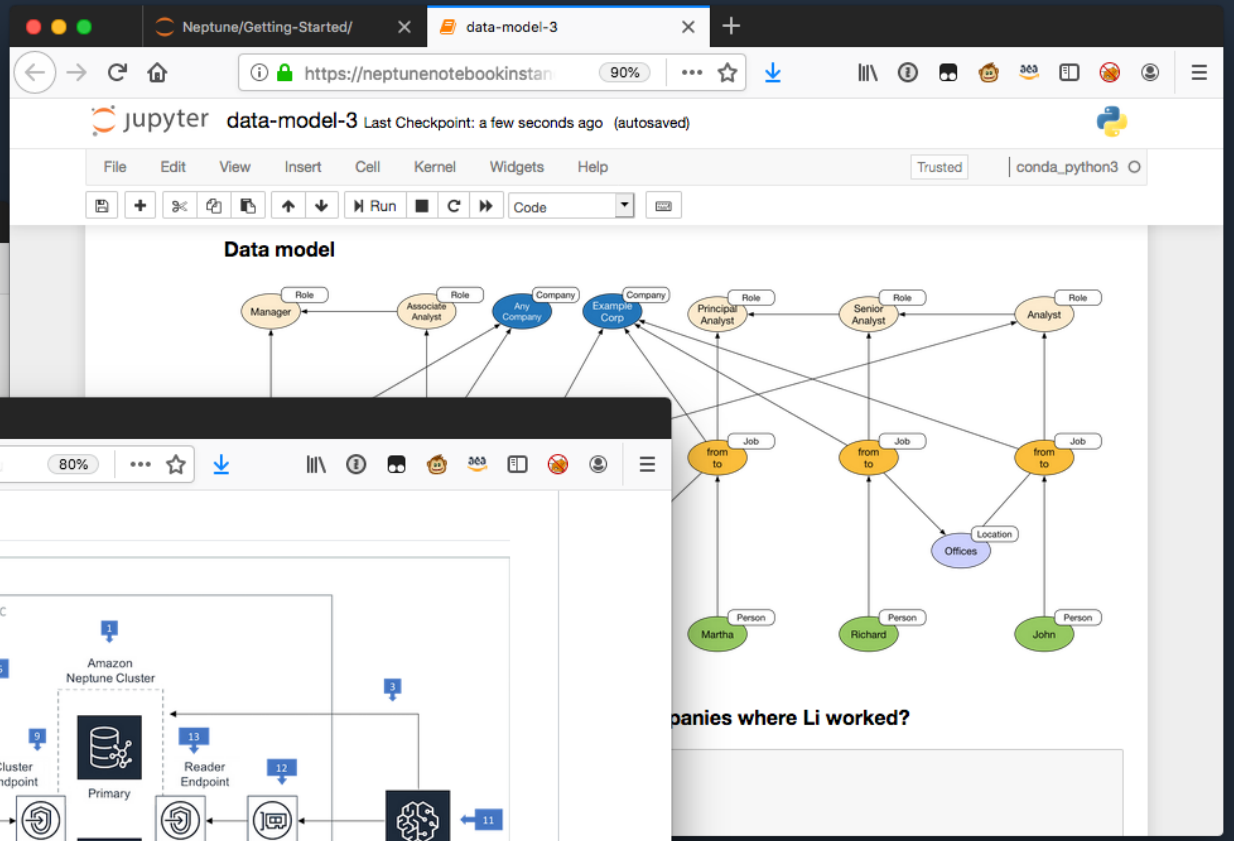
This architecture shows you how to connect AWS Lambda functions to Amazon Neptune.

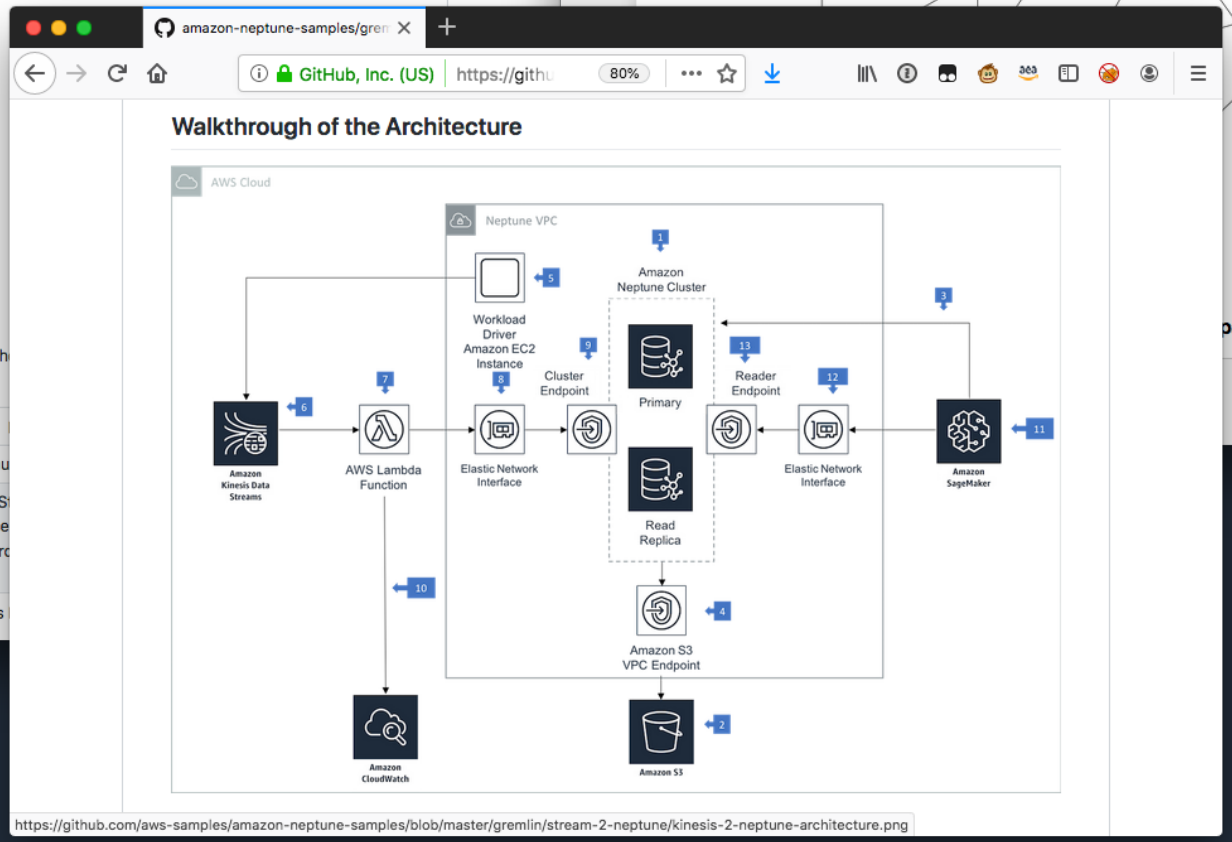


Samples

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







Thank You!