【AWS Black Belt Online Seminar】 Amazon Neptune

アマゾン ウェブ サービス ジャパン株式会社
ソリューションアーキテクト　五十嵐 建平

2018.07.03

自己紹介

五十嵐 建平

技術統括本部
ソリューションアーキテクト

好きなサービス

- Amazon Aurora
- Amazon Performance Insights
- Amazon Neptune

AWS Black Belt Online Seminarとは

AWSJのTechメンバがAWSに関する様々な事を紹介するオンラインセミナーです

【火曜 12:00～13:00】
主にAWSのソリューションや
業界カットでの使いどころなどを紹介
(例：IoT、金融業界向け etc.)

【水曜 18:00～19:00】
主にAWSサービスの紹介や
アップデートの解説
(例：EC2、RDS、Lambda etc.)

※開催曜日と時間帯は変更となる場合がございます。最新の情報は下記をご確認下さい。
オンラインセミナーのスケジュール＆申し込みサイト https://aws.amazon.com/jp/about-aws/events/webinars/

内容についての注意点

- 本資料では2018年7月3日時点のサービス内容および価格についてご説明しています。最新の情報はAWS公式ウェブサイト(http://aws.amazon.com)にてご確認ください。

- 資料作成には十分注意しておりますが、資料内の価格とAWS公式ウェブサイト記載の価格に相違があった場合、AWS公式ウェブサイトの価格を優先とさせていただきます。

- 価格は税抜表記となっています。日本居住者のお客様が東京リージョンを使用する場合、別途消費税をご請求させていただきます。

- AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

このセミナーの内容

- グラフデータベースについて
- Amazon Neptuneの特徴
- Amazon Neptuneの使い方
  - 構成
  - インターフェイス
  - 可視化
  - 学び方
- まとめ

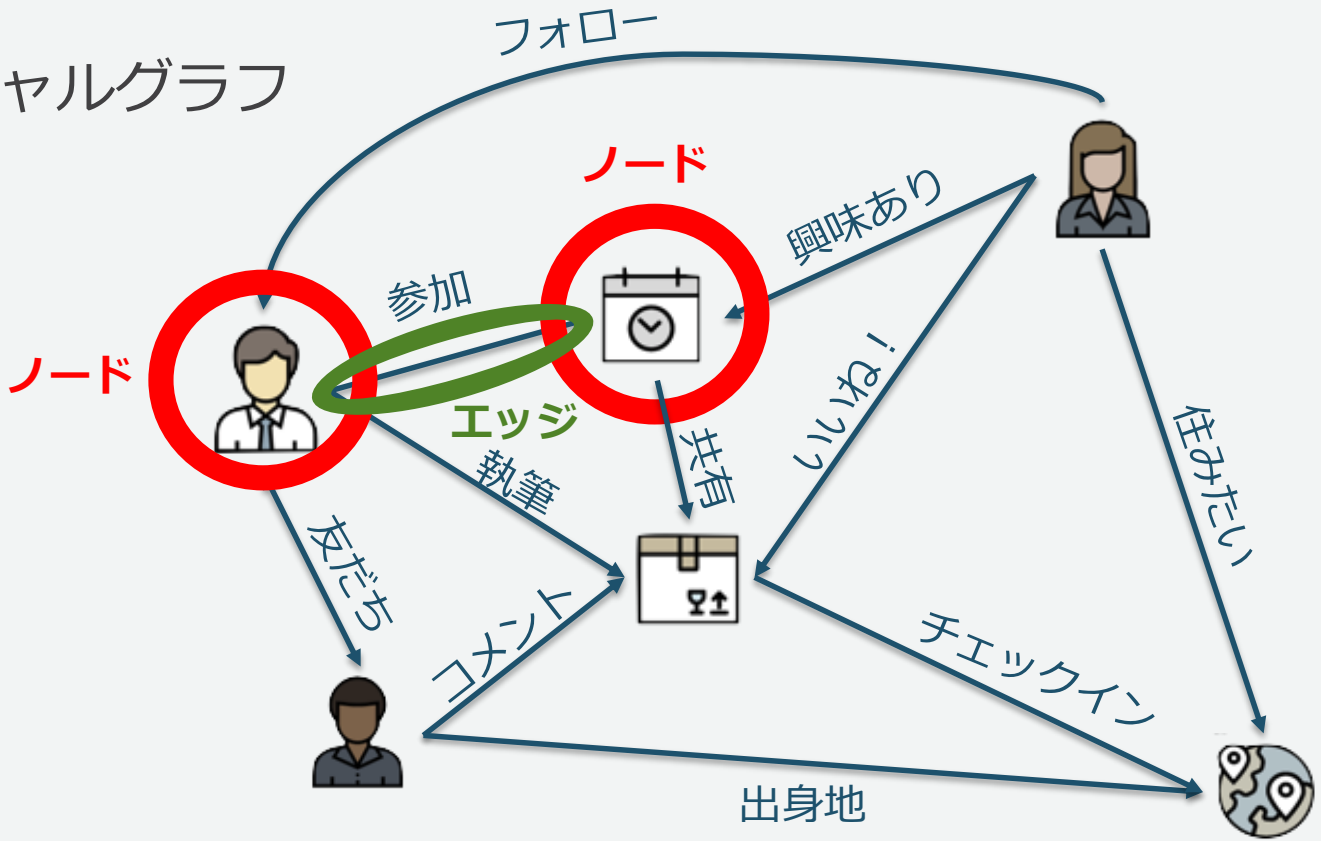
このセミナーの内容

- グラフデータベースについて
- Amazon Neptuneの特徴
- Amazon Neptuneの使い方
  - 構成
  - インターフェイス
  - 可視化
  - 学び方
- まとめ

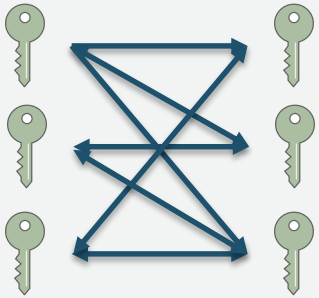
グラフ構造

ノードとエッジ

ソーシャルグラフ



データの関連と実装の分類

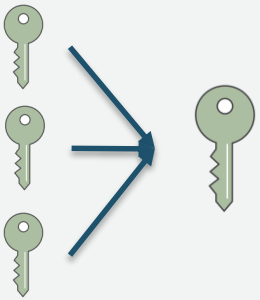


多対多

グラフデータベース

New

Amazon Neptune



多対1

RDBMS

Amazon RDS

Amazon Redshift

1対1

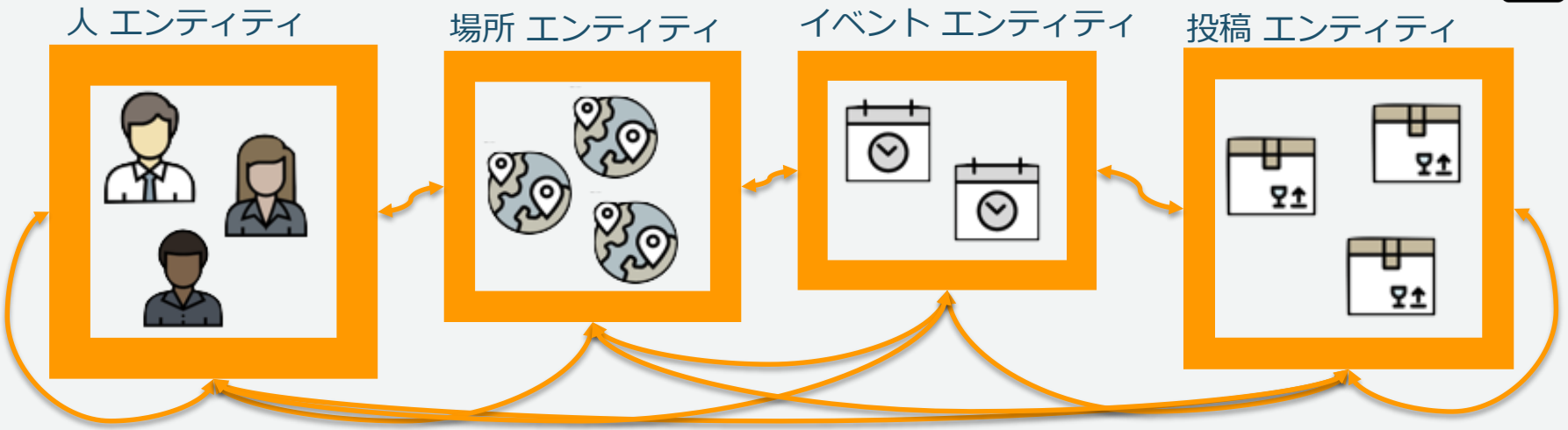
Key-Value ストア

Amazon DynamoDB

Amazon ElastiCache

aws

ソーシャルグラフの例：すべてが多対多



人に対して関係する投稿は一意に決まるか? → No (複数投稿する)
投稿に対して関係する人は一意に決まるか? → No (投稿する人, いいねをする人…)
人に対して関係する人は一意に決まるか? → No (複数の友だちがいる)
人に対して関係するイベントは一意に決まるか? → No (複数のイベントに参加/いいねが可能)
イベントが決まると人が一意に決まるか? → No (複数の人が関わる)
…(省略)

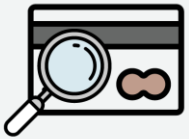
すべてのエンティティ関連が多対多となる

グラフデータ(高度に連結されたデータ)のユースケース

ソーシャル
グラフ

レコメンデーション

ナレッジグラフ



不正検出

ライフサイエンス

ネットワーク/IT運用

aws

RDBMSは多対多のデータを直接表現できない

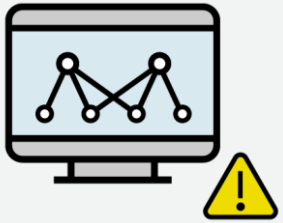
RDBMSは原則として多対1の関連の結合のみを扱う
→グラフ構造の「エッジ」も表(関連表)にする必要がある
(person_id, …) → (person_id, event_id) ← (event_id, …)

ソーシャルグラフの例では、本質的には4つのエンティティであるにも関わらず、
自己関連表を含めて新たに10個もの関連表を定義する必要がある。
しかも列数/列名なども事前に設計する必要がある。

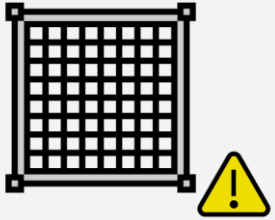
- エンティティが増加した場合、関連表の数も劇的に増加する
- ちょっとしたグラフ探索でもクエリの複雑性が非常に高くなりやすくなる

RDBによるグラフデータ(高度に連結したデータ)での課題

SQLはグラフへの
問合せをするには
不自然



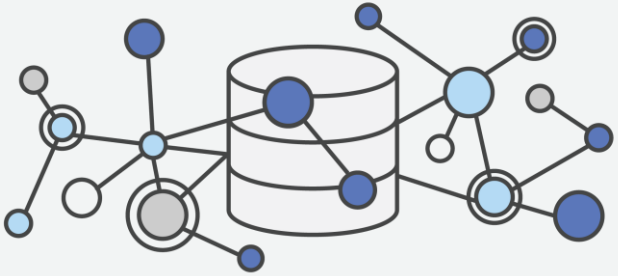
グラフ処理には
機能が不十分



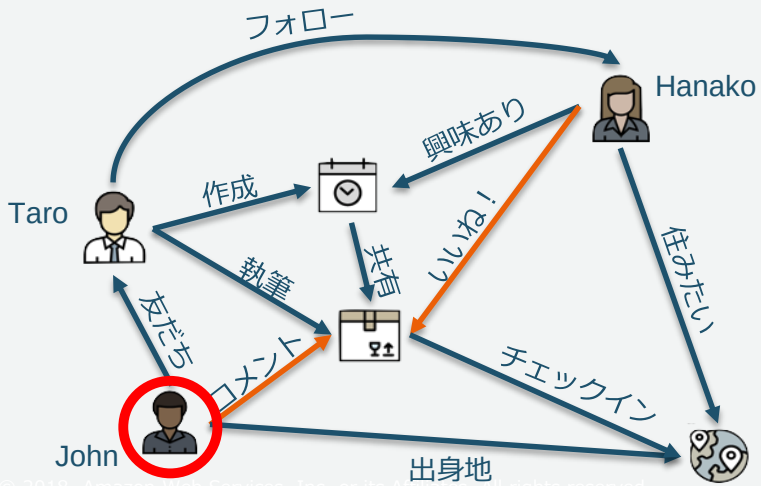
データ変更に対して
スキーマ定義が厳格で
柔軟性に欠ける

aws

グラフデータベース



グラフデータベースは高度に連結されたデータのために
最適化された効率的なストレージと探索エンジン



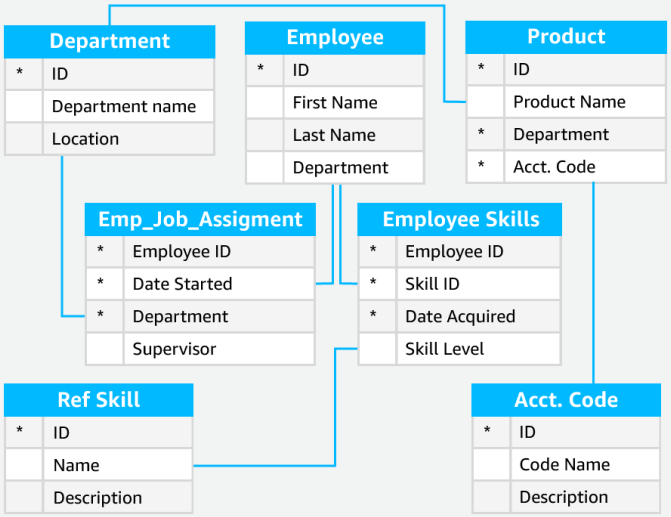
「Hanakoがいいね!をした記事にコメントした他の人」
をリストアップするGremlinの例

```
g.V().has('name','Hanako').as('her')
.out('like').in('comment').where(neq('her'))
```


aws

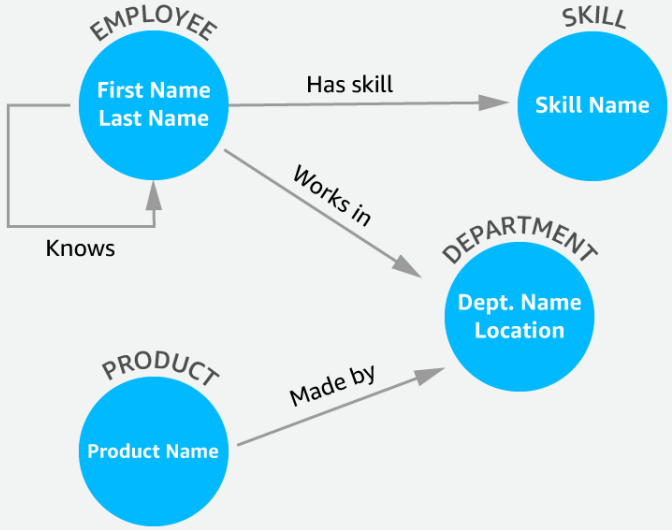
高度に連結されたデータに対する異なるアプローチ

リレーショナルデータ



ビジネスプロセス要件のために
作られる

グラフデータ

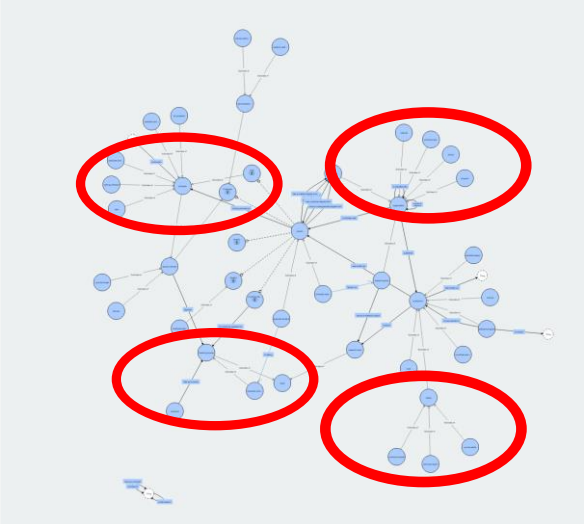


関連性に対する質問に答えるために
作られる

グラフ処理の分類

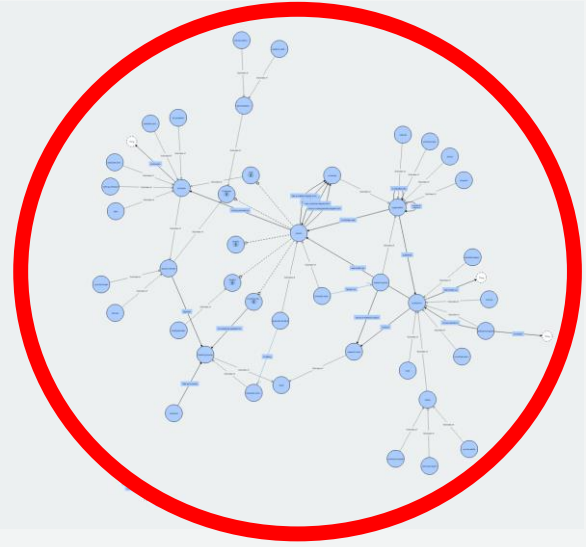
OLTP

1処理単位でグラフの一部を探索
リコメンデーション、不正検知など
並列かつ大量に実行される



OLAP

1処理単位でグラフのすべてを探索
クラスタリング、全ノードの重みバランスの調整など
1度の処理が大きい



従来のグラフデータベースの課題

スケールが難しい

高可用性を維持するのが難しい

高価

オープンスタンダードに対して限定的なサポートしかない

このセミナーの内容

- グラフデータベースについて
- Amazon Neptuneの特徴
- Amazon Neptuneの使い方
  - 構成
  - インターフェイス
  - 可視化
  - 学び方
- まとめ

Amazon Neptune
フルマネージドなグラフデータベースサービス

Relational Databases

Amazon RDS

Aurora
- MySQL
- PostgreSQL

Commercial
- Oracle
- Microsoft SQL Server

Community
- MySQL
- PostgreSQL
- MariaDB

Amazon Redshift

Data Warehouse

Non-Relational Databases

NEW!

Amazon DynamoDB

Key Value
- Document

Amazon ElastiCache

In-Memory Data Store
- redis
- Memcached

Amazon Neptune

Graph

AWS Database Migration Service

aws

Amazon Neptune の特徴

高速

億レベルの
リレーションシップを
ミリ秒のレイテンシで
問合せ可能

高信頼性

3つのAZに跨がる
6つのレプリカを
フルバックアップ/
リストアと共に提供

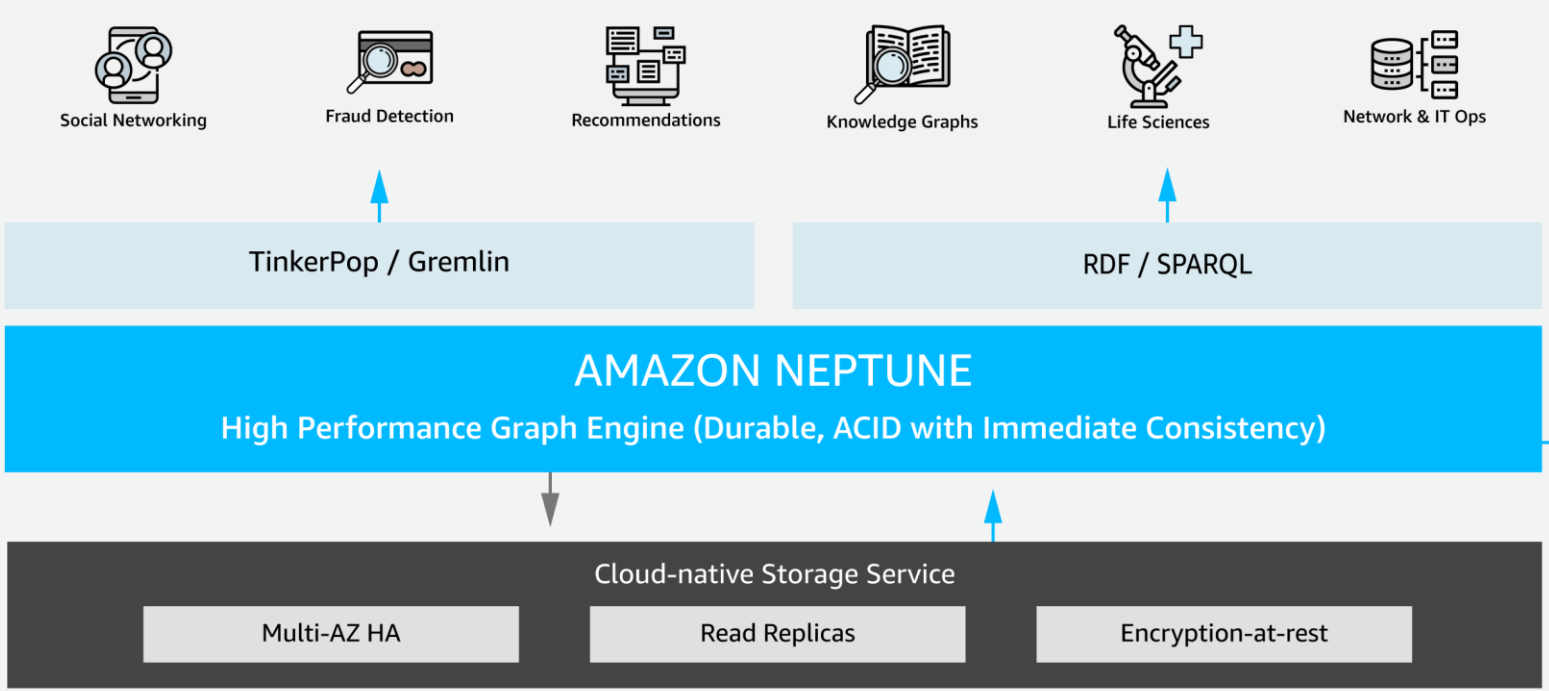
簡単

GremlinとSPARQLに
よる簡単でパワフル
なクエリを実行

オープン

Apache TinkerPop
& W3C RDF グラフ
モデルをサポート

Amazon Neptune ハイレベルアーキテクチャ

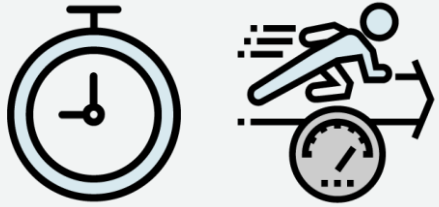


クエリ性能

- OLTPのグラフワークロードにおいて
  毎秒100,000件のクエリをサポートするよう設計
    - 最大15個のリードレプリカ
    - 専用に設計されたクラウドネイティブなストレージサービス
    - 最適化されたインメモリアーキテクチャ
    - クエリの最適化
- 索引を作成する必要がない
    - Neptune側で自動的に最適なI/Oパスを実行

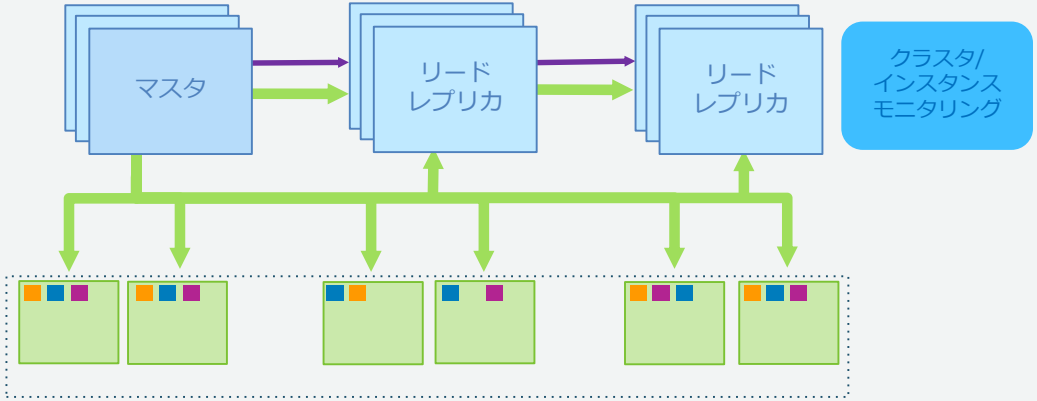
aws

リードレプリカ



可用性

- データベースノードの障害は自動的に検知され、交換される
- データベース処理の障害は自動的に検知され、リソースはリサイクルされる
- リードレプリカは必要に応じて自動的にマスタに昇格する
- どのリードレプリカに対して優先的にフェイルオーバーさせるかを指定できる



パフォーマンス

- リードレプリカによってアプリケーションの読み込みトラフィックをスケールアウトさせることができる
- 読み込みエンドポイントによってリードレプリカを跨がって負荷が分散される

aws

クラウドネイティブストレージエンジンの概要

データは**3つのアベイラビリティーゾーンに跨がった6つのレプリカ**にコピーされる

継続的に堅牢な Amaon S3 へバックアップされる

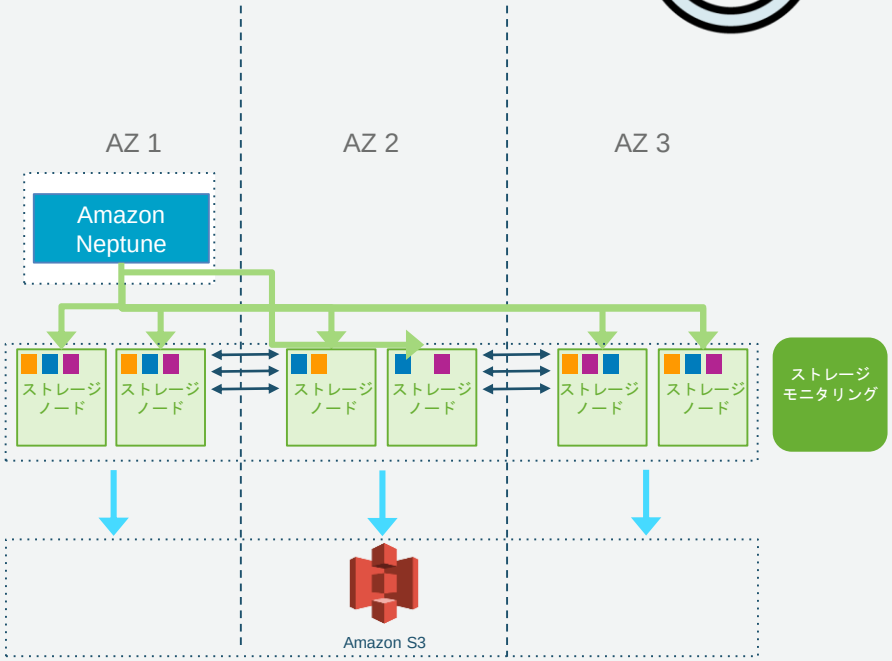
継続的にノードやディスクが修復される

修復やホットスポットのリバランスのために**10GBのセグメントユニット**で管理されている

読み書きには**レイテンシ耐性を持つクォーラムシステム**を使用している

クォーラムのメンバーシップが変更されたとしても書き込みは阻害されない

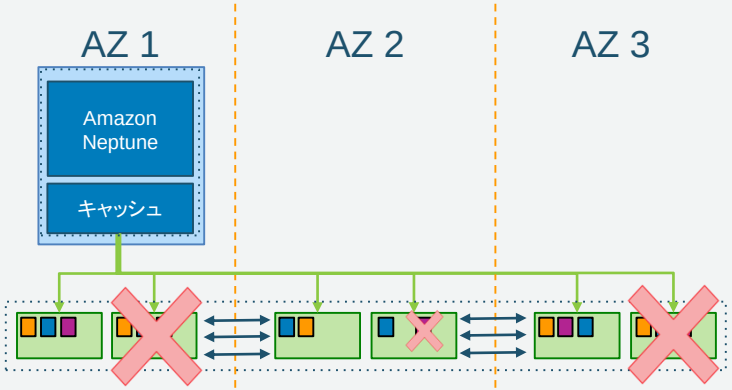
ストレージは使用に応じて**自動的に64TBまで拡張**される

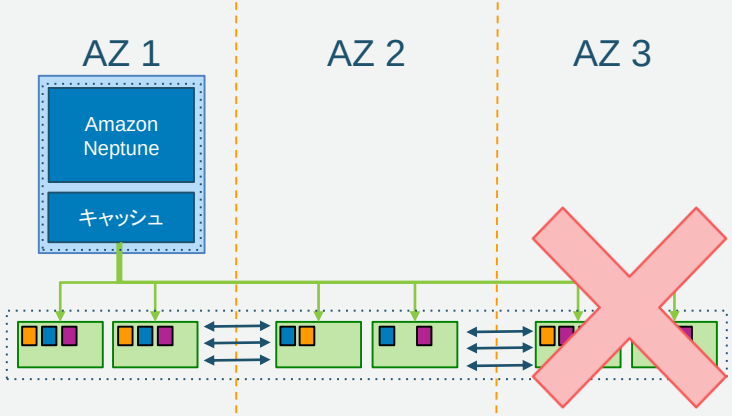


クラウドネイティブストレージエンジンの耐障害性

・どこに障害が発生すると仮定するか？
　セグメント障害(ディスク)
　ノード障害(マシン)
　AZ障害(ネットワークやデータセンター)

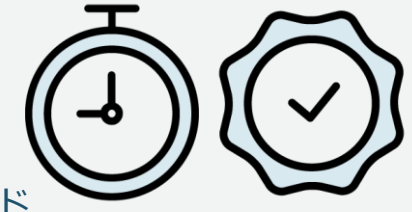
・想定される障害に対する最適化
　6つ中4つの書き込みクォーラム
　6つ中3つの読み込みクォーラム
　修復時には各ノード同士で
　レプリケーションを実施

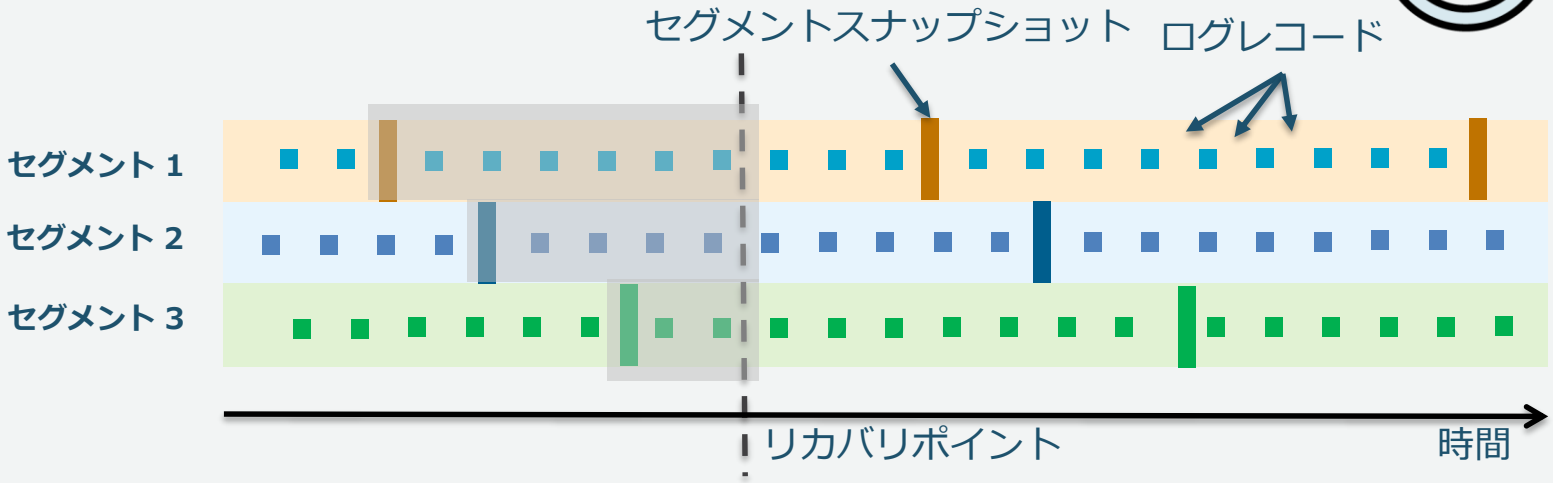




aws

S3への継続的バックアップ

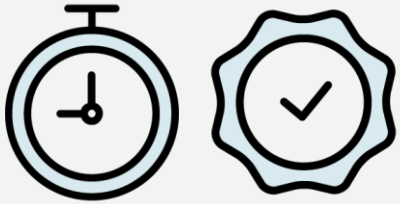


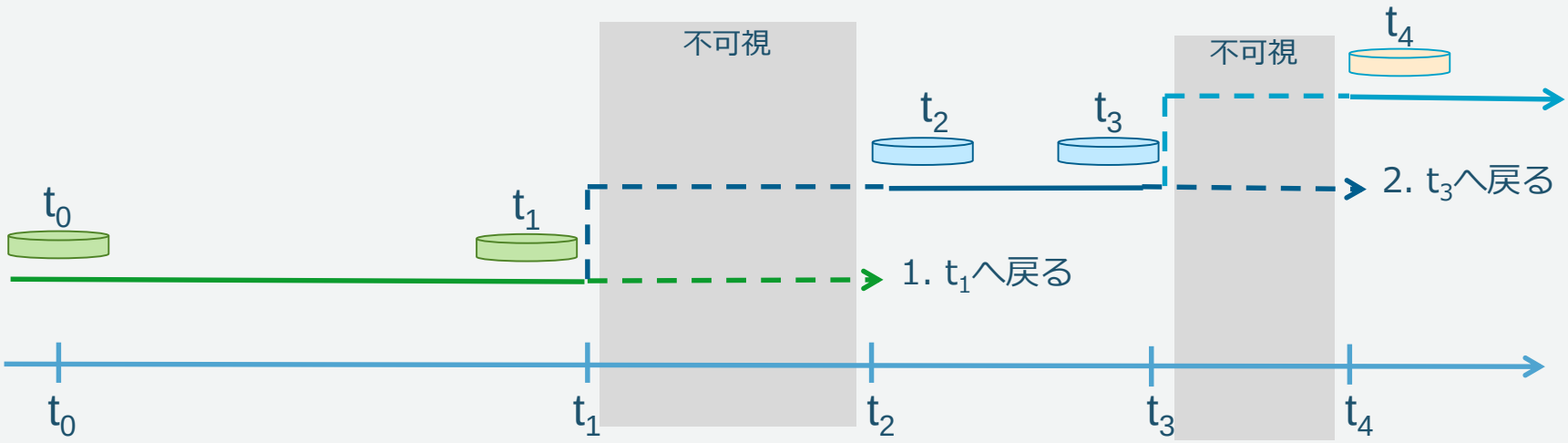


- セグメント毎に定期的にS3へ並列でバックアップが取得されている
- バックアップはパフォーマンスや可用性にインパクトを与えることなく絶えず行われている
- リストア時には、セグメント毎に適切なスナップショットと必要なログレコードが抽出される
- スナップショットへのログレコードの適用は並列に非同期的に実行される

aws

Point-in-Time リストア





オンラインでの point-in-time リストアにより、バックアップからリストアすることなく指定した時間にデータベースの状態を戻すことができる

aws

リクエストのIAM認証

IAMによるリクエスト認証が可能

- AWS署名バージョン4を使用して都度リクエストに署名する必要がある
- 署名のコーディングの必要がある
- 署名や認証のオーバーヘッドに注意

https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/iam-auth.html

暗号化

通信の暗号化

- 現状TLSは無効
- セキュリティグループによるアクセス制御
- IAMロール認証

格納されたデータの暗号化

- データベース作成時にのみ指定可能(変更/解除不可)
- クライアントからは透過的
- AES-256を使用した暗号化
- KMSを使用(既存キーも使用可能)

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

モニタリング

CloudWatch

Neptune用の複数のメトリクスを定義済み

- 例: スケールアップが必要な状況かどうかCPU使用率を用いて判断する
- 例: トラバーサルのエラーが発生していないかを確認する

メトリクス(一部)	説明
CPUUtilization	CPU 使用率。
GremlinErrors	Gremlin がトラバーサルしたエラー数。
SparqlRequests	SPARQL エンジンへのリクエスト数。

Audit Logの出力が可能

- 接続元IPアドレス、接続先IPアドレス、RAWメッセージがすべて保存される
- インスタンスタイプに応じて複数のファイルに分割して出力され、100MB毎にローテーションする
- マネジメントコンソールから閲覧やダウンロードが可能

```
1530539742245, /10.0.2.59:47354, /10.0.2.112:8182, Websocket, "RequestMessage{, requestId=21ad8a13-
6489-4f9e-8067-71f7dbd6f8cd, op='eval', processor='', args={gremlin=g.V(), bindings={},
batchSize=64}}"
```

2つのグラフモデルとフレームワーク

プロパティグラフ

Apache TinkerPop™ (OSS)

Gremlin Traversal Language

Apache TinkerPop

RDF

W3C標準

SPARQL Query Language

Semantic Web

RDF

SPARQL

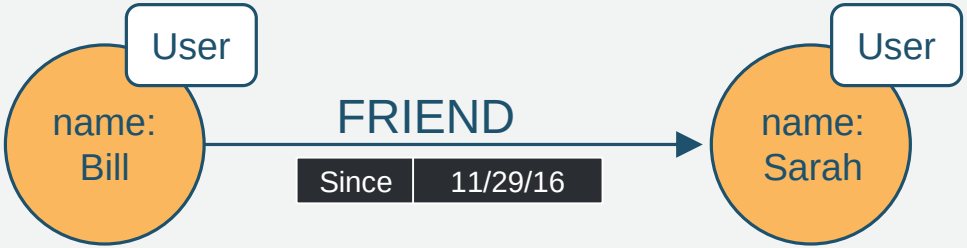
※トランザクションもサポート(バルクロード時除く)

https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/access-graph-gremlin-differences.html

aws

プロパティグラフ

Vertex、Edge、付随するプロパティから構成される一般的なグラフ構造



- Vertex はエンティティ/ドメインを表す
- Edge はVertex間の直接の関係を表す
  - 各Edgeは関係の種類を示すラベルを持つ
- VertexとEdgeはそれぞれユニークな識別子を持つ
- VertexとEdgeはプロパティを持つことができる
  - プロパティはVertexとEdgeに関する型付きの非リレーショナルな情報を表現する
  - Tinkerpop 3では label というデフォルトのプロパティを必ず持つ必要がある
  - 型はブール、整数、浮動小数点、文字列、日付

aws

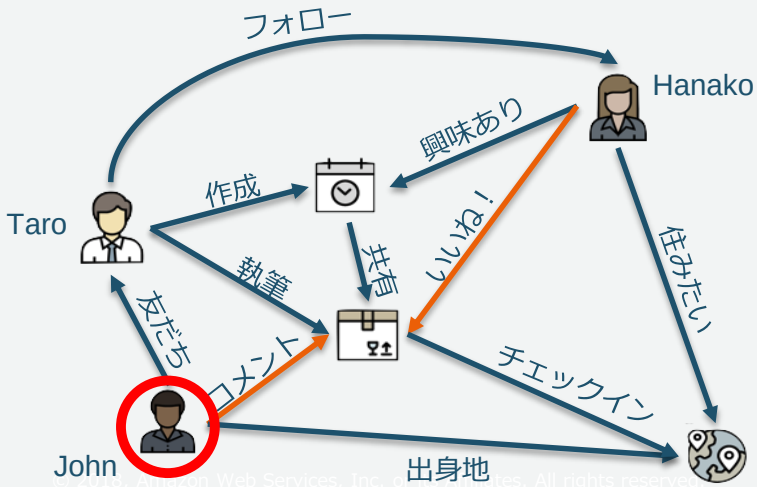
再掲: ソーシャルグラフでの例

ノード用CSV

```
~id,~label,name:String,age:Int,created:Date
p1,person,"Hanako",25,
p2,person,"Taro",21,
p3,person,"John",45,
a1,article,,,"2018-07-03"
e1,event,"Special Event",,"2018-06-18"
c1,checkinplace,"Hokkaido",,,
```

エッジ用CSV

```
~id,~from,~to,~label,created:Date,comment:String
e1,p1,a1,like,"2018-07-03",
e2,p1,e1,interest,"2018-06-29",
e3,p1,c1,wanttolive,,
e4,p2,p1,follow,,
e5,p2,a1,make,"2018-07-03",
e6,p2,e1,make,"2018-06-18",
e7,p3,p2,friend,,
e8,p2,p3,friend,,
e9,p3,a1,comment,,"good article"
e10,a1,c1,checkin,,
e11,p3,c1,hometown,,
```



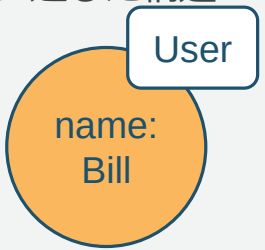
「Hanakoがいいね!をした記事にコメントした他の人」をリストアップするGremlinの実行例

```
gremlin>
g.V().has('name','Hanako').as('her').out('like')
.in('comment').where(neq('her')).valueMap()
==>{name=[John], age=[45]}
```

RDF (Resource Description Framework)

- 外部と交換／外部に公開される厳密なセマンティックグラフを表現するのに適した構造
- トリプル(主語, 述語, 目的語) でデータを表現する

Subject → `<http://www.socialnetwork.com/person#1>`
Predicate → `rdf:type contacts:User;`
Object (literal) → `contact:name: "Bill" .`

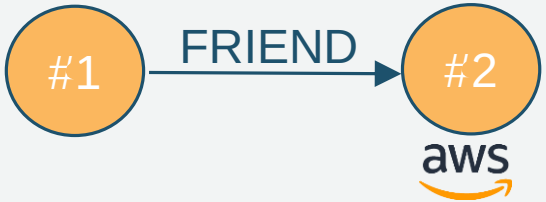


- Internationalized Resource Identifiers (IRIs) で主語を一意に特定する

IRI → `<http://www.socialnetwork.com/person#1>`

- 目的語は IRI あるいはリテラルを使う
  - RDFにおけるリテラルはプロパティに似ており、XMLデータ型に対応する
  - 目的語が IRI の場合、グラフにおける Edge となる

Subject → `<http://www.socialnetwork.com/person#1>`
Predicate → `contacts:friend`
Object (IRI) → `<http://www.socialnetwork.com/person#2> .`



料金構成

要素	詳細
時間単位のオンデマンドインスタンス料金	db.r4.large～db.r4.8xlarge の 5タイプ
データベースストレージおよびI/O	容量(GB) I/Oリクエスト数(100万リクエスト単位)
バックアップストレージ	容量(GB) ※リージョンのNeptuneデータベースストレージ合計を超えるまでは無料
データ転送	リージョン外への通信量(GB) ※1GBまでは無料

https://aws.amazon.com/jp/neptune/pricing/

月額料金計算例

米国東部リージョンの場合

db.r4.large/容量50GB(バックアップ50GB)
/平均77.2 iops (月間2億I/Oリクエスト)/リージョン外への通信量 10GB

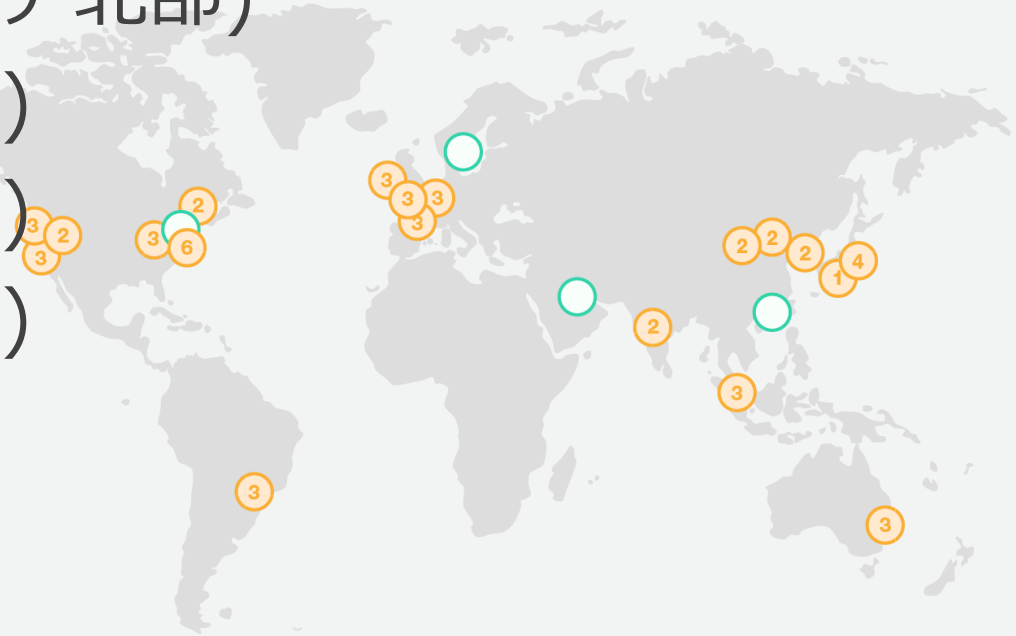
要素	詳細
時間単位のオンデマンドインスタンス料金	$250.56 ($0.348 * 24 * 30)
データベースストレージおよびI/O	容量 $5 ($0.10 * 50) I/Oリクエスト $40 ($0.20 * 200)
バックアップストレージ	$0 ※データベースストレージ容量まで無料
データ転送	$0.81 ($0.09 * 9) ※1GBまでは無料
	合計 $296.37

https://aws.amazon.com/jp/neptune/pricing/

リージョン

- 米国東部 (バージニア北部)
- 米国東部 (オハイオ)
- 米国西部 (オレゴン)
- 欧州 (アイルランド)

順次拡大予定



このセミナーの内容

- グラフデータベースについて
- Amazon Neptuneの特徴
- Amazon Neptuneの使い方
  - 構成
  - インターフェイス
  - 可視化
  - 学び方
- まとめ

EC2からのアクセス

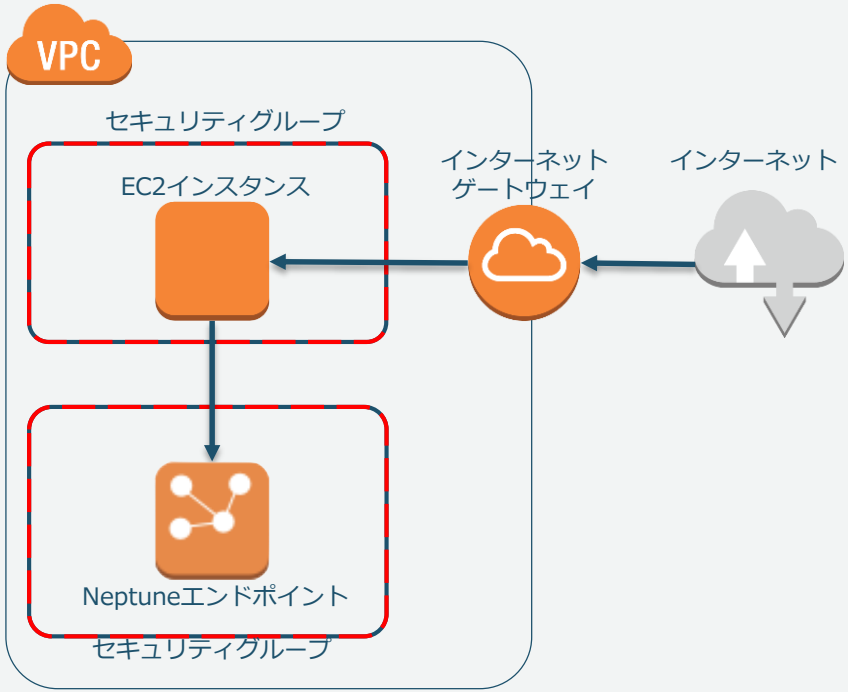
NeptuneはVPC内に配置される

- Public IPアドレスは付与できない

→ インターネットから直接アクセスは不可なのでVPC内部のEC2などからアクセスする

アクセス制御はセキュリティグループで実施

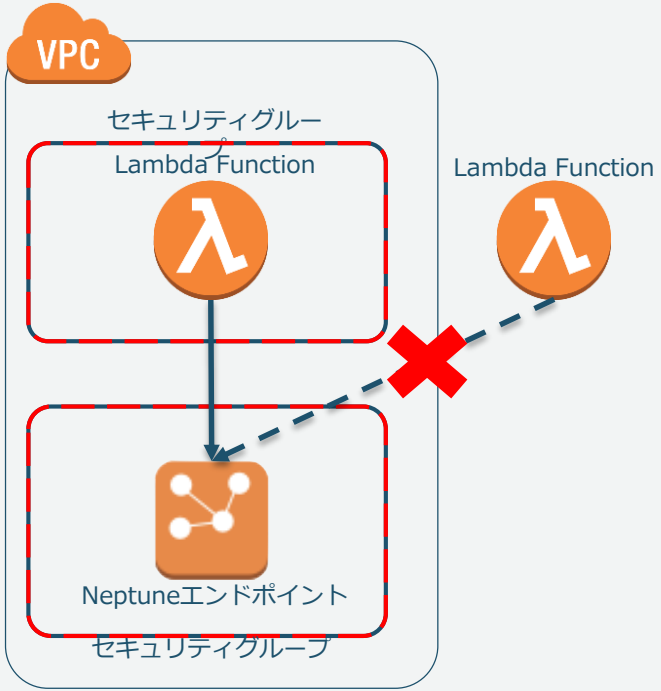
→ IPアドレス範囲またはセキュリティグループをソースとして、アクセスを許可するポート(デフォルト8182番)を指定



https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/access-graph-launch-ec2-instance.html

Lambdaからのアクセス

VPC外からのLambdaからはアクセス不可
→ VPC内のLambdaから利用する



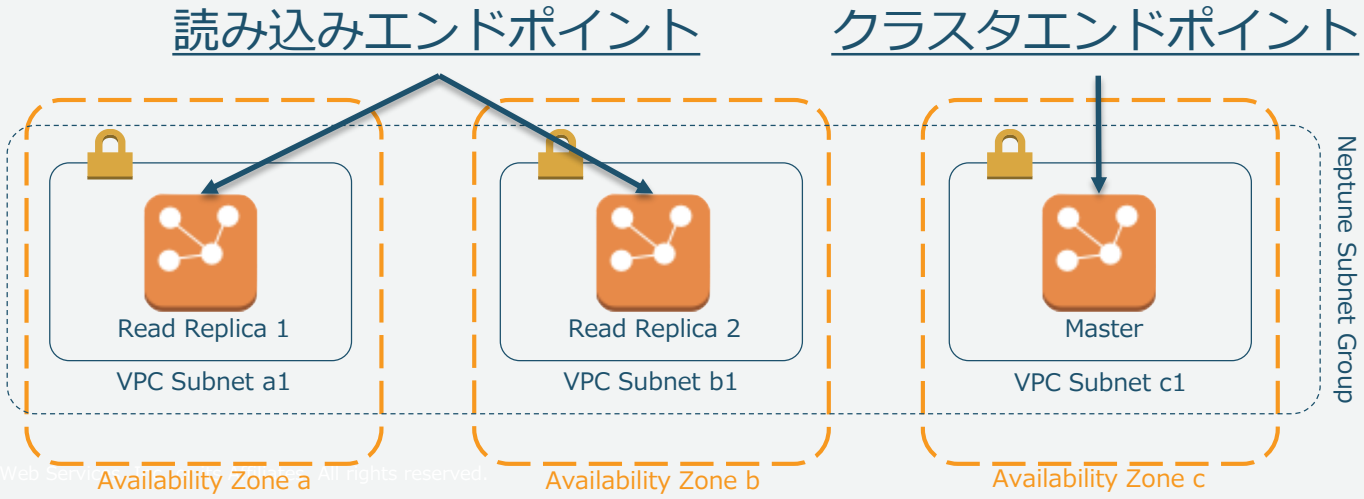
サブネットグループとエンドポイント

サブネットグループ

- インスタンスを起動するサブネットを定義
- リージョン内の少なくとも2つのAZにサブネットが必要

エンドポイント

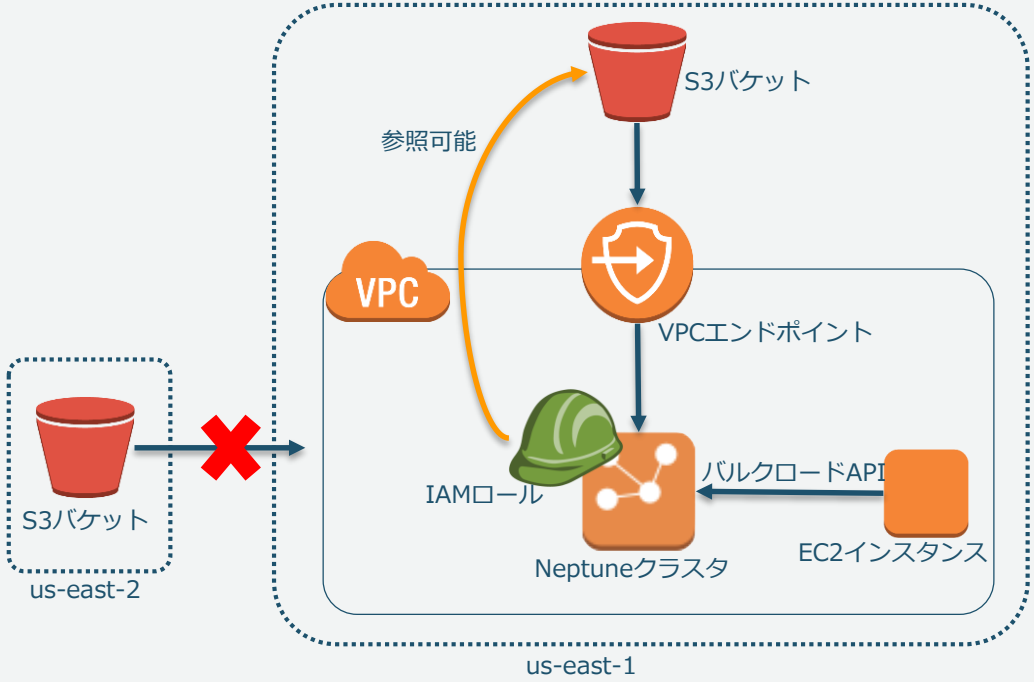
- クラスタエンドポイントは書き込み可能なマスタインスタンスに解決される
- 読み込みエンドポイントはいずれかのリードレプリカに解決される



バルクロード用S3バケット

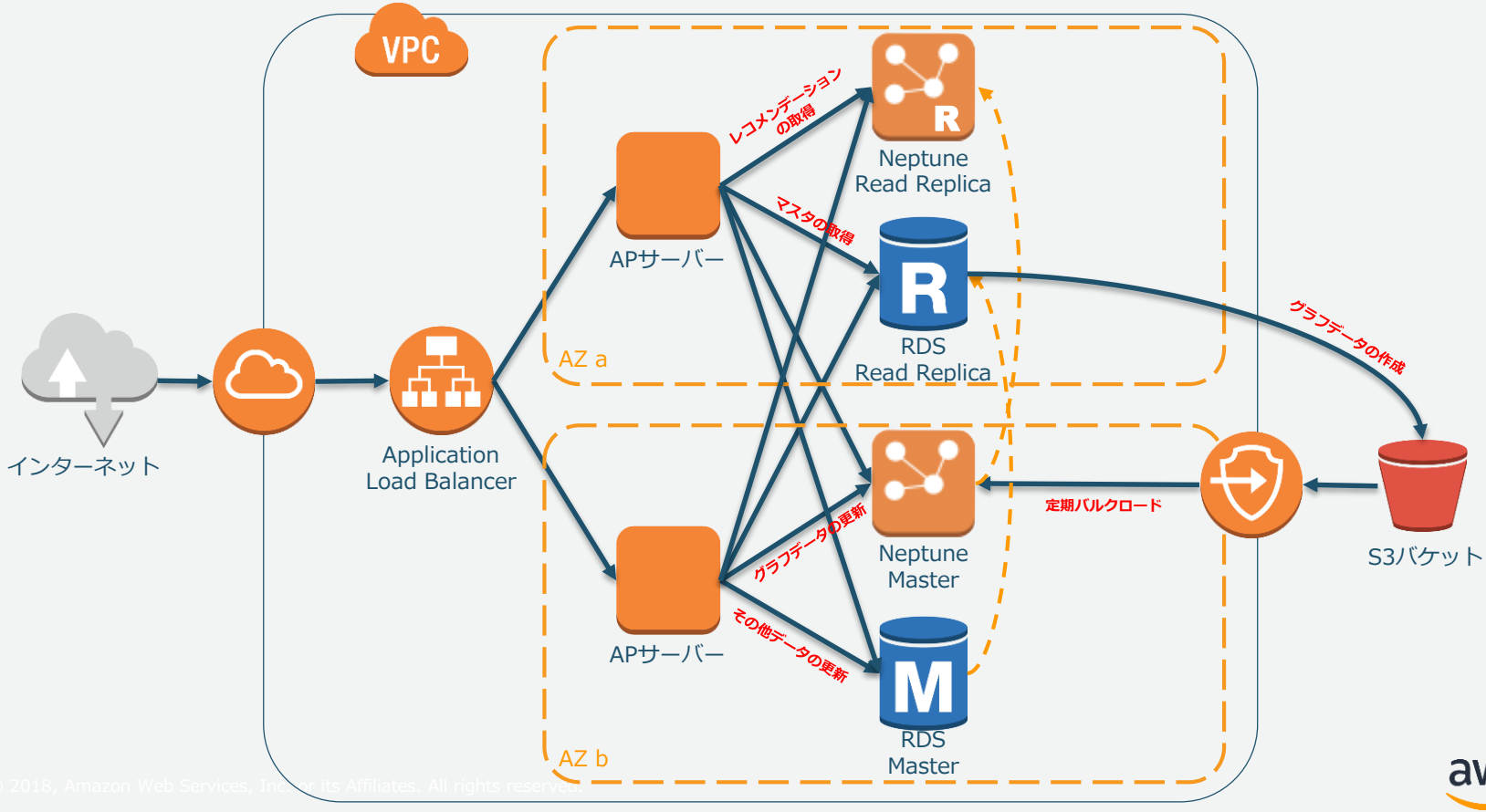
バルクロード用のS3バケットの条件

- 同一リージョン
- VPCエンドポイント
- アクセス可能なIAMロールを Neptuneクラスタにアタッチ



https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/bulk-load.html

Webアプリケーションにおけるスケール可能な構成例



このセミナーの内容

- グラフデータベースについて
- Amazon Neptuneの特徴
- Amazon Neptuneの使い方
  - 構成
  - インターフェイス
  - 可視化
  - 学び方
- まとめ

ドキュメントに記載されたインターフェイス一覧

	Tinkerpop/Gremlin	RDF/SPARQL	バルクロード	Neptuneの管理
コンソール	Gremlin Console	RDF4J Console RDF4J Workbench	-	AWS CLI
Java	Gremlin Driver	RDF4J	-	AWS SDK for Java
Python	Gremlin-Python Driver	-	-	boto3
.NET	Gremlin.NET	-	-	AWS SDK for .NET
HTTP REST	HTTP REST (Gremlin)	HTTP REST (SPARQL)	Neptune Loader API	Health Status
REST以外の HTTP	Gremlin HTTP WebSocket API	SPARQL HTTP	-	-

本セミナーでは赤字のインターフェイスについてご紹介します

[Gremlin] Gremlin Console

Groovyベースのコンソール
用途: Gremlinの学習/アプリケーション開発/アドホック分析/データベース運用

```
$ cat conf/neptune-remote.yaml
hosts: [<Neptune end point>]
port: 8182
serializer: { className: (省略)}
```

サーバー定義

```
$ cat neptune.groovy
:remote connect tinkerpop.server conf/neptune-remote.yaml
:remote console
```

Neptuneに接続する初期化スクリプト

```
$ cat bin/mygremlin.sh
bin/gremlin.sh -i neptune.groovy
```

初期化スクリプトを呼び出す起動シェル

```
$ bin/mygremlin.sh
         ¥,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated:
tinkerpop.utilities
plugin activated:
tinkerpop.tinkergraph
gremlin> g.V().limit(1)
==>v[Luke]
gremlin>
```

起動シェルの実行

Neptuneへのクエリ

https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/access-graph-gremlin-console.html
http://tinkerpop.apache.org/docs/current/reference/#connecting-via-console
https://github.com/tinkerpop/gremlin/wiki/Getting-Started

aws

[Gremlin] Gremlin Driver (Java)

Javaベースのアプリケーションで使用可能なGremlinドライバ
用途: Javaアプリケーション

pom.xml

```
<dependency>
   <groupId>org.apache.tinkerpop</groupId>
   <artifactId>gremlin-driver</artifactId>
   <version>3.3.3</version>
</dependency>
```

ソースコード

```
Cluster.Builder builder = Cluster.build();
builder.addContactPoint("your-neptune-endpoint");
builder.port(8182);
Cluster cluster = builder.create();
GraphTraversalSource g = EmptyGraph.instance()
      .traversal().withRemote(
      DriverRemoteConnection.using(cluster));
GraphTraversal t = g.V().limit(1).valueMap();
t.forEachRemaining(e ->  System.out.println(e));
cluster.close();
```

Neptuneへのクエリ

https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/access-graph-gremlin-java.html
http://tinkerpop.apache.org/docs/current/reference/#connecting-via-java

aws

[Gremlin] Gremlin-Python Driver

Pythonで使用可能なGremlin Driver
Javaの Gremlin Driver を基にしており一部制限がある
用途: Gremlinの学習/Pythonアプリケーション

インストール

```
$ pip install --user gremlinpython
…
Successfully installed gremlinpython-3.3.3
```

ソースコード

```
from __future__  import print_function
from gremlin_python import statics
from gremlin_python.structure.graph import Graph
from gremlin_python.process.graph_traversal import __
from gremlin_python.process.strategies import *
from gremlin_python.driver.driver_remote_connection import DriverRemoteConnection

graph = Graph()
g = graph.traversal().withRemote(DriverRemoteConnection(
      'ws://your-neptune-endpoint:8182/gremlin','g'))

print(g.V().limit(1).toList())
```

Neptuneへのクエリ

https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/access-graph-gremlin-python.html
http://tinkerpop.apache.org/docs/current/reference/#connecting-via-python

aws

[Gremlin] HTTP

HTTP REST/WebSocketインタフェース

ドライバが不要

用途: ドライバのない言語・環境からのアクセス/HTTPベースのGremlinアプリケーションやツール/動作確認

※変数のバインド(bindings)は使用不可

(NGの例: `curl -X POST -d '{"gremlin":"g.V().limit(x)", "bindings":{"x":1}}' http://your-end-point:8182/gremlin`

HTTP RESTの例

```
$ curl -X POST -d '{"gremlin":"g.V().limit(1)"}' http://your-end-point:8182/gremlin
```

Neptuneへのクエリ


```
{"requestId":"e2b22100-f81c-4f31-6862-6dc525b3a543",
"status":{"message":"","code":200,"attributes":{"@type":"g:Map","@value":[]}},
"result":{"data":{"@type":"g:List","@value":[{"@type":"g:Vertex",
"@value":{"id":"Luke","label":"person","properties":
{"GamerAlias":[{"@type":"g:VertexProperty","@value":
{"id":{"@type":"g:Int32","@value":-1735852585},"value":"skywalker123",
"label":"GamerAlias"}}]}}}]},"meta":{"@type":"g:Map","@value":[]}}}
```


https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/access-graph-gremlin-rest.html
https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/gremlin-api-reference.html
http://tinkerpop.apache.org/docs/current/reference/#_connecting_via_http

aws

[SPARQL] RDF4J Console

インタラクティブにSPARQLを発行可能なRDF4Jのコンソール (RDF4J WorkbenchというGUIツールもあります)
用途: SPARQLの学習/アプリケーション開発/アドホック分析

```
$ bin/console.sh
...
> create sparql
Please specify values for the following variables:
SPARQL query endpoint: http://your-end-point:8182/sparql
Local repository ID [endpoint@localhost]: neptune
...
> open neptune
neptune> sparql select ?s ?p ?o where {?s ?p ?o} limit 1
Evaluating SPARQL query...
+------------------------+------------------------+------------------------+
| s                      | p                      | o                      |
+------------------------+------------------------+------------------------+
| <http://aws.amazon.com/neptune/John_Smith>| <http://www.nist.gov/units/weight>| _:b33558799
|
+------------------------+------------------------+------------------------+
1 result(s) (46 ms)
neptune>
```

エンドポイント定義

リポジトリ名

Neptuneへのクエリ

https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/access-graph-sparql-rdf4j-console.html
http://docs.rdf4j.org/server-workbench-console/#_rdf4j_console

[SPARQL] RDF4J Runtime (Java)

Javaベースのアプリケーションで使用可能なRDF4Jのランタイムライブラリ

用途: Javaアプリケーション

pom.xml

```
<dependency>
   <groupId>org.eclipse.rdf4j</groupId>
   <artifactId>rdf4j-runtime</artifactId>
   <version>2.2</version>
</dependency>
```

ソースコード

```
Repository repo = new SPARQLRepository(
  "http://your-neptune-endpoint:8182/sparql");
repo.initialize();
try (RepositoryConnection conn = repo.getConnection()) {
      TupleQuery tupleQuery =
        conn.prepareTupleQuery(QueryLanguage.SPARQL,
        "SELECT ?s ?p ?o WHERE { ?s ?p ?o } limit 10");
                        Neptuneへのクエリ
      try (TupleQueryResult result = tupleQuery.evaluate()) {
            while (result.hasNext()) {
                  BindingSet bindingSet = result.next();
                  Value s = bindingSet.getValue("s");
                  Value p = bindingSet.getValue("p");
                  Value o = bindingSet.getValue("o");
                  System.out.print(s + "¥t" + p + "¥t" + o);
            }
      }
}
```

https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/access-graph-sparql-java.html

http://docs.rdf4j.org/programming/

aws

[SPARQL] HTTP

HTTP REST または SPARQL HTTPインタフェース

ドライバが不要

用途: ドライバのない言語・環境からのアクセス/SPARQL Protocolを用いるアプリケーションやツール/動作確認

HTTP RESTの例

```
$ curl -X POST --data-binary 'query=select ?s ?p ?o where {?s ?p ?o} limit 1' ¥
  http://your-end-point:8182/sparql
{
  "head" : {"vars" : [ "s", "p", "o" ]},
  "results" : {
    "bindings" : [ {
      "s" : {"type" : "uri", "value" : "http://aws.amazon.com/neptune/John_Smith"},
      "p" : {"type" : "uri", "value" : "http://www.nist.gov/units/weight"},
      "o" : {"type" : "bnode","value" : "b33558799"}
    } ]
  }
}
```

Neptuneへのクエリ

https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/access-graph-sparql-http-rest.html
https://www.w3.org/TR/sparql11-protocol/

バルクロード

S3バケットからNeptuneインスタンスにバルクロードする HTTP REST API を用意

- バケットはNeptuneインスタンスと同一リージョンにある必要がある
- フォーマットによってGremlinかSPARQLかを判断 (CSV形式の場合はGremlin)
- 非同期ジョブとして実行され、キャンセルやキャンセル後の再開が可能
- NeptuneクラスタにアタッチしたIAMロールによってS3に対する認証を実施
  (accessKey/secretKeyによる認証は廃止されます)

ロードコマンド例

```
curl -X POST ¥
    -H 'Content-Type: application/json' ¥
    http://<your-end-point>:8182/loader -d '
    {
      "source" : "s3://<bucket-and-key>/",
      "iamRoleArn" : "<iam-role-arn>",
      "format" : "csv",
      "region" : "us-east-1",
      "failOnError" : "FALSE"
    }'
```

https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/bulk-load.html

ヘルスステータス

データベースステータスを簡単に確認できるURLを用意

- アプリケーションから接続可能かどうかのチェック
- セキュリティグループの設定が正しいかのチェック

URL: http(s)://*your-neptune-endpoint*:8182/status
正常時応答: {"status": "healthy”}
異常時応答: ステータス500, タイムアウトなど

コマンド例

```
$ curl -G http://your-neptune-endpoint:8182/status
{"status":"healthy"}
```

https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/access-graph-status.html

Neptuneの管理 - AWS CLI/SDK

インスタンスの作成/参照/削除やフェイルオーバーなど、
Neptune管理のためのCLI/SDKが用意されています。
※VPC外からの実行も可能です

CLIによるリードレプリカへのフェイルオーバーの例

```
$ aws neptune failover-db-cluster --db-cluster-identifier rdfcluster --region us-east-1
{
    "DBCluster": {
        "MasterUsername": "admin",
        "ReaderEndpoint": ”your-end-point",
        "ReadReplicaIdentifiers": [],
...(省略)
}
```

https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/load-api-reference-load.html

このセミナーの内容

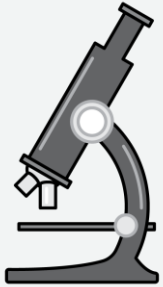
- グラフデータベースについて
- Amazon Neptuneの特徴
- Amazon Neptuneの使い方
  - 構成
  - インターフェイス
  - 可視化
  - 学び方
- まとめ

何のための可視化か？

全体を見るための
可視化

視覚的に全体的な構造を把握
クラスタリングの仮説を立てる
ドリルダウン

実トランザクションの大規模グラフ
ではすべてのデータを描画しても
情報過多となる



一部を見るための
可視化

視覚的にターゲット周辺の構造を把握
トラバーサルの仮説を立てる

大規模グラフのOLTP用途では
こちらが中心となる

aws

オープンなインターフェイスの活用

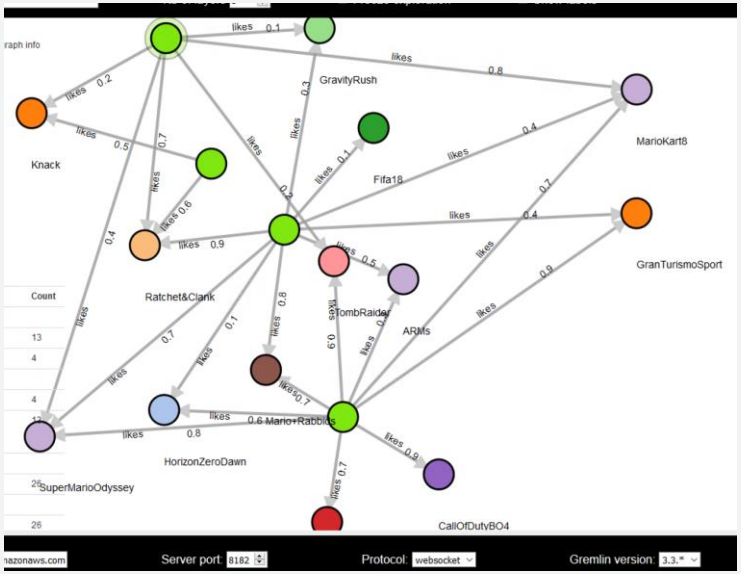
GremlinまたはRDF/SPARQLに対応した可視化ツールを使用
※ Neptune自体は可視化機能を提供しない

可視化ツールの例	カテゴリ	インターフェイス
Grashexp	オープンソース	Gremlin
Tom Sawyer Software	商用	Gremlin RDF/SPARQL
Metaphactory	商用	Gremlin RDF/SPARQL
Cambridge Intelligence / Keylines	商用	RDF/SPARQL

※可視化ツールはあくまで補助的なものであり、ユースケースに合わせた設計と処理が最も大切

aws

Graphexp



JavaScriptベースの可視化ツール

GitHubリポジトリ以下のファイル一式をWebサーバに配置するだけのシンプルさ

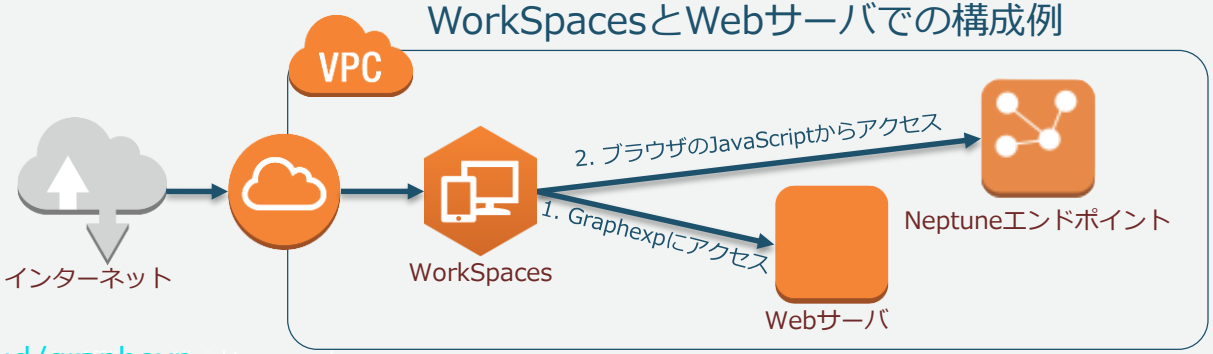
以下設定でNeptuneに対応

graphConf.js

```
const SINGLE_COMMANDS_AND_NO_VARS = true;
```

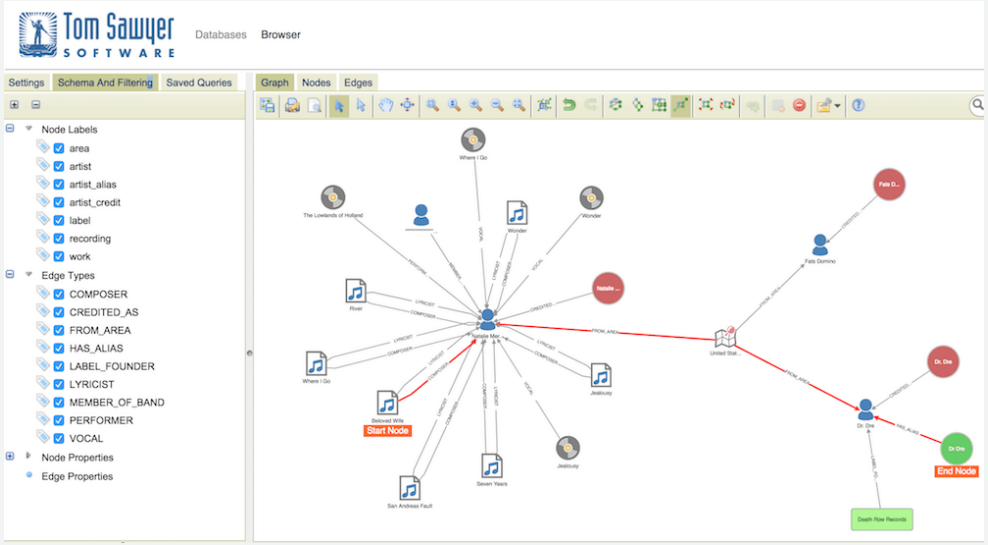
Webブラウザから直接Neptuneエンドポイントに
アクセスできる必要がある

WorkSpacesとWebサーバでの構成例



https://github.com/bricaud/graphexp

Tom Sawyer



多機能で歴史の長い
商用データ可視化ツール

- GremlinとSPARQLの両方に対応

re:Invent 2017において
Neptuneのサポートとパートナーシップを発表

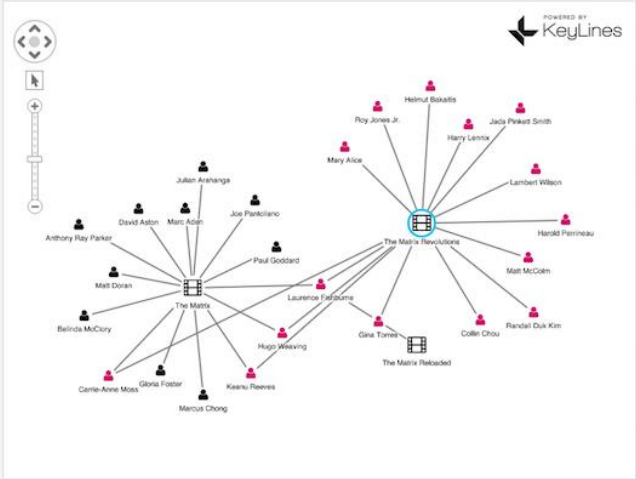
https://www.tomsawyer.com/aws-amazon-neptune-partner/

- 今後 The Tom Sawyer Perspectives Graph Database Browser を AWS MarketPlace 上で公開予定

Cambridge Intelligence / Keylines

グラフ可視化アプリケーションを作成するための高機能な商用ライブラリ

- GremlinとSPARQLの両方に対応
- JavaScriptベース



Cambridge Intelligence社による Amazon Neptune 対応に関するブログ

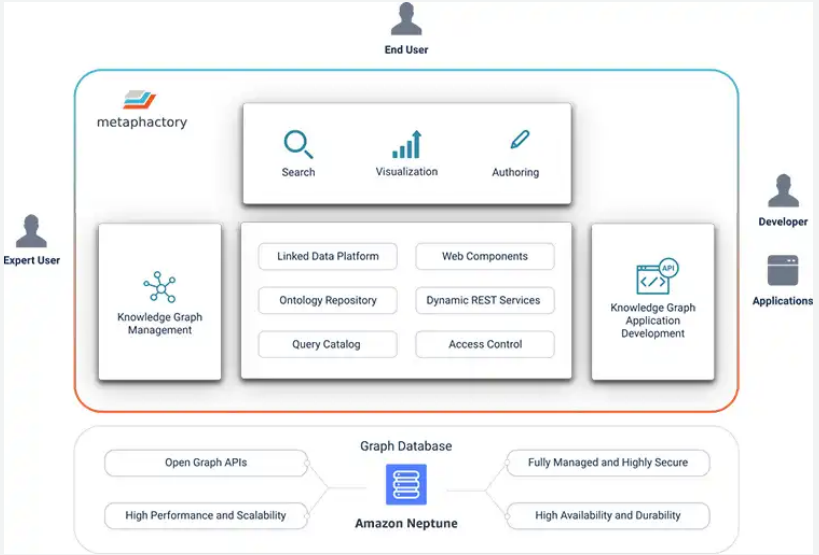
https://cambridge-intelligence.com/visualizing-the-amazon-neptune-database-with-keylines/

Metaphactory

エンタープライズ向け
ナレッジグラフプラットフォーム

- RDF/SPARQLベース
- グラフデータベースとして
  Amazon Neptuneをサポート

http://www.metaphacts.com/amazon-neptune



Metaphactory + Amazon Neptune のユースケースセッション

AWS re:Invent 2017: NEW LAUNCH! Amazon Neptune Overview and Customer Use Cases (DAT319)
https://www.youtube.com/watch?v=9pmQXua9LWA
※Metaphactoryについては38:50あたりから紹介

このセミナーの内容

- グラフデータベースについて
- Amazon Neptuneの特徴
- Amazon Neptuneの使い方
  - 構成
  - インターフェイス
  - 可視化
  - 学び方
- まとめ

サンプルデータ

AWSによるサンプルデータ

[Gremlin]

協調フィルタリングの例

https://github.com/aws-samples/amazon-neptune-samples/tree/master/gremlin/collaborative-filtering

→ Neptune バルクロードして使用

[RDF/SPARQL]

Coming Soon

オープンなサンプルデータの例

[Gremlin]

GitHub: apache / tinkerpop / data

https://github.com/apache/tinkerpop/tree/master/data

→ 次ページのNeptune ToolsでCSVに変換後バルクロード

[RDF/SPARQL]

RDF Examples by W3C

https://www.w3.org/2000/10/rdf-tests/

→ Neptune でそのままバルクロード可能

[Gremlin] Neptune Tools

GraphML ファイルをNeptuneでのバルクロードが可能な CSV ファイルに変換するツール

https://github.com/awslabs/amazon-neptune-tools/tree/master/graphml2csv

TinkerpopのModernデータを変換する例

```
# TinkerpopのModernデータのうちGraphML形式の tinkerpop-modern.xml をダウンロード
$ curl https://raw.githubusercontent.com/apache/tinkerpop/master/data/tinkerpop-
modern.xml -o tinkerpop-modern.xml

# Pythonスクリプトを実行してノードとエッジの2つのCSVファイルに変換
$ ./graphml2csv.py -i tinkerpop-modern.xml
infile = tinkerpop-modern.xml
Processing tinkerpop-modern.xml
Wrote 6 nodes and 18 attributes to tinkerpop-modern-nodes.csv.
Wrote 6 edges and 12 attributes to tinkerpop-modern-edges.csv.

# あとは変換されたCSVファイルをS3バケットに配置してNeptuneにバルクロードすればOK
```

[Gremlin] 学習パスの例

1. Gremlin Console(とGraphexp)を用いてクエリを実行しながら学習

- AWS SamplesのTutorialに沿ってクエリを実行してみる

https://github.com/aws-samples/amazon-neptune-samples/tree/master/gremlin/collaborative-filtering

- TinkerpopのTutorialで学習する

http://tinkerpop.apache.org/docs/3.3.3/tutorials/gremlins-anatomy/
http://tinkerpop.apache.org/docs/3.3.3/tutorials/the-gremlin-console/

※データは前ページで紹介したModernデータを用いる

※Neptuneにおける実装での相違点に留意しながら実行する
https://docs.aws.amazon.com/ja_jp/neptune/latest/userguide/access-graph-gremlin-differences.html

2. Graphexpで構造を確認しながらオリジナルのグラフとクエリを作成

[RDF/SPARQL] 学習パスの例

1. 公開されたグローバルなナレッジベースを用いてSPARQLを学習
- 例: DBPedia http://ja.dbpedia.org/sparql

SPARQLチュートリアル@第36回SWO研究会-DBpediaシンポジウム
https://docs.google.com/spreadsheets/d/1zlrplSjBXviaJaEHWBrKqiSYXS1K9ojDKDqOGl5BGOI/edit#gid=0
既存のクエリを少しずつ改変しながら学ぶなどが有効。

2. 独自のデータを Neptune に作成しオリジナルのクエリを作成

このセミナーの内容

- グラフデータベースについて
- Amazon Neptuneの特徴
- Amazon Neptuneの使い方
  - 構成
  - インターフェイス
  - 可視化
  - 学び方
- まとめ

従来のグラフデータベースの課題

スケールが難しい

高可用性を維持するのが難しい

高価

オープンスタンダードに対して限定的なサポートしかない

Amazon Neptune の特徴

高速

億レベルの
リレーションシップを
ミリ秒のレイテンシで
問合せ可能

高信頼性

3つのAZに跨がる
6つのレプリカを
フルバックアップ/
リストアと共に提供

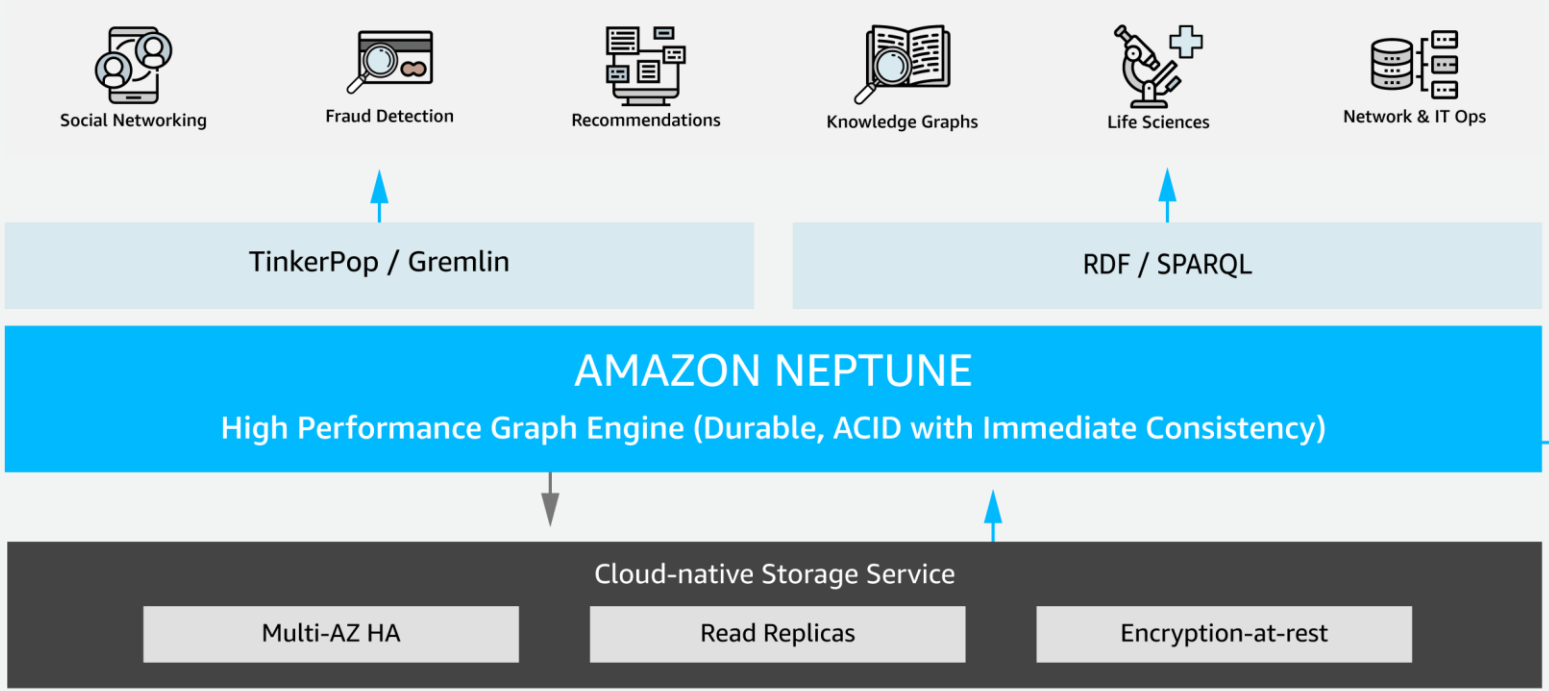
簡単

GremlinとSPARQLに
よる簡単でパワフル
なクエリを実行

オープン

Apache TinkerPop
& W3C RDF グラフ
モデルをサポート

Amazon Neptune ハイレベルアーキテクチャ



Neptuneの使い方 – 様々なインターフェイス

	Tinkerpop/Gremlin	RDF/SPARQL	バルクロード	Neptuneの管理
コンソール	Gremlin Console	RDF4J Console RDF4J Workbench	-	AWS CLI
Java	Gremlin Driver	RDF4J	-	AWS SDK for Java
Python	Gremlin-Python Driver	-	-	boto3
.NET	Gremlin.NET	-	-	AWS SDK for .NET
HTTP REST	HTTP REST (Gremlin)	HTTP REST (SPARQL)	Neptune Loader API	Health Status
REST以外のHTTP	Gremlin HTTP WebSocket API	SPARQL HTTP	-	-

さいごに

Amazon Neptune で
スケーラブルで堅牢なグラフデータベースを
実現しましょう！

Amazon Neptune についてもっと知りたい方はこちら

- 公式ページ
  https://aws.amazon.com/jp/neptune/
- ドキュメントやチュートリアルなどのリソース
  https://aws.amazon.com/jp/neptune/developer-resources/
- Amazon Neptune Forum
  https://forums.aws.amazon.com/forum.jspa?forumID=253

aws

AWS オンラインセミナースケジュール

AWS オンラインセミナースケジュールは以下より確認できます
https://amzn.to/JPWebinar

オンラインセミナー資料の配置場所

AWS クラウドサービス活用資料集

- https://aws.amazon.com/jp/aws-jp-introduction/

Amazon Web Services ブログ

- 最新の情報、セミナー中のQ&A等が掲載されています。
- https://aws.amazon.com/jp/blogs/news/

公式Twitter/Facebook
AWSの最新情報をお届けします

@awscloud_jp

もしくは
http://on.fb.me/1vR8yWm

最新技術情報、イベント情報、お役立ち情報、
お得なキャンペーン情報などを日々更新しています！

AWSの導入、お問い合わせのご相談

AWSクラウド導入に関するご質問、お見積、資料請求をご希望のお客様は以下のリンクよりお気軽にご相談下さい。

https://aws.amazon.com/jp/contact-us/aws-sales/

※「AWS 問い合わせ」で検索して下さい。

AWS Well Architected 個別技術相談会お知らせ

- Well Architectedフレームワークに基づく数十個の質問項目を元に、お客様がAWS上で構築するシステムに潜むリスクやその回避方法をお伝えする個別相談会です。
  https://pages.awscloud.com/well-architected-consulting-2017Q4-jp.html
- 参加無料
- 毎週火曜・木曜開催