Amazon DynamoDB

2017/08/09
AWS Black Belt Online Seminar 2017
アマゾン ウェブ サービス ジャパン株式会社

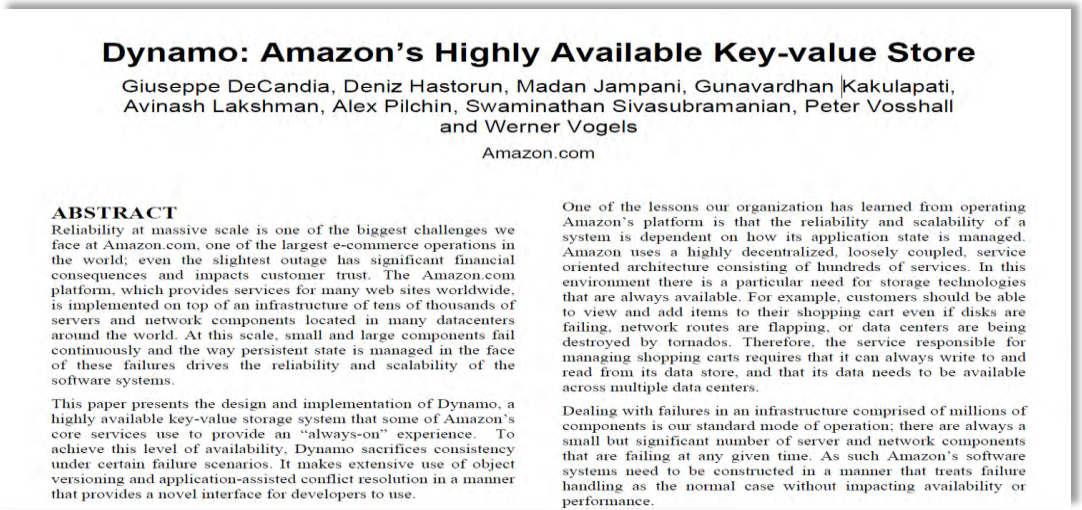
ソリューションアーキテクト　成田 俊

Agenda

- DynamoDBとは
- テーブル設計とサンプル
- Data Modeling
- ユースケース及びベストプラクティス
- TTL
- Auto Scaling
- DAX
- DynamoDB Streams
- AWS Mobile SDKと 2-Tier アーキテクチャ
- 運用関連
- ツールとエコシステム
- まとめ

Amazon DynamoDBの生い立ち

- Amazon.comではかつて全てのアクセスパターンをRDBMSで処理していた
- RDBMSのスケールの限界を超えるため開発されたDynamoが祖先



Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon's software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.

- 結果整合性モデル採用による可用性向上
- HWを追加する毎に性能が向上するスケーラビリティ
- シンプルなクエリモデルによる予測可能な性能

Amazon DynamoDBの特徴

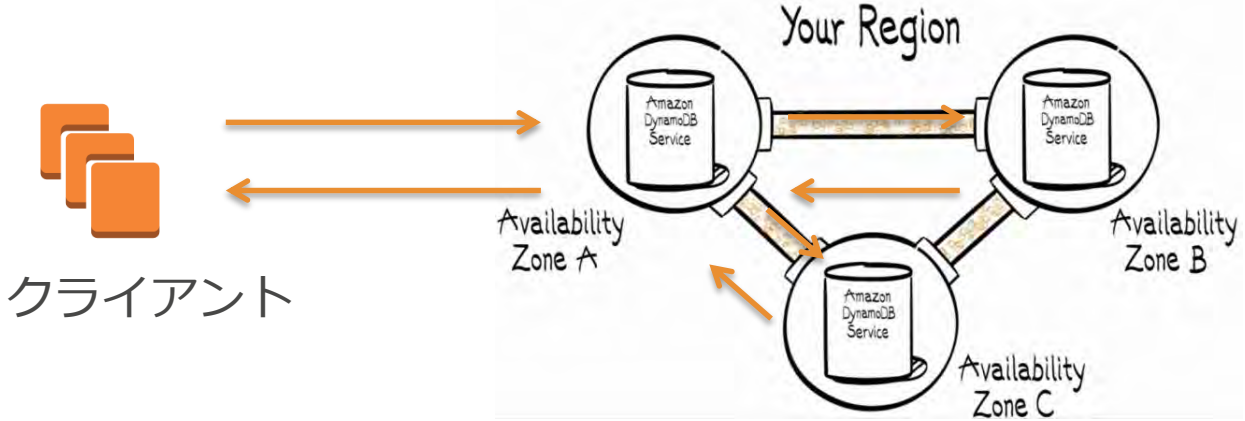
- 完全マネージド型の NoSQL データベースサービス
- ハイスケーラブル、低レイテンシー
- 高可用性– 3x レプリケーション
- シンプル且つパワフルAPI
- ストレージの容量制限がない
- 運用管理必要なし

DynamoDBの特長

- 管理不要で信頼性が高い
- プロビジョンドスループット
- ストレージの容量制限がない

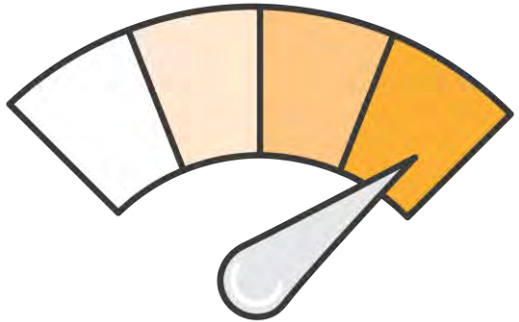
特長１：管理不要で信頼性が高い

- SPOFの存在しない構成
- データは3箇所のAZに保存されるので信頼性が高い
- ストレージは必要に応じて自動的にパーティショニングされる



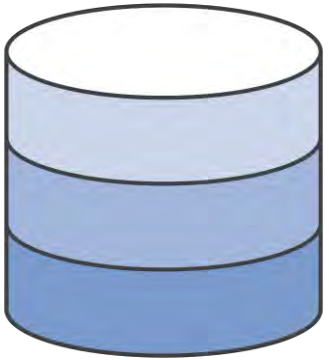
特長２：プロビジョンドスループット

- テーブルごとにReadとWriteそれぞれに対し、必要な分だけのスループットキャパシティを割り当てる（＝プロビジョンする）ことができる
- 例えば下記のようにプロビジョンする
  - Read：1,000
  - Write：100
- 書き込みワークロードが上がってきたら
  - Read：500
  - Write：1,000
- この値はDB運用中にオンラインで変更可能
  - ただし、スケールダウンに関しては日に9回までしかできないので注意



特長３：ストレージの容量制限がない

- 使った分だけの従量課金制のストレージ
- データ容量の増加に応じたディスクやノードの増設作業は一切不要



DynamoDBの整合性モデル

- Write
  - 少なくとも２つのAZでの書き込み完了が確認とれた時点でAck
- Read
  - デフォルト
    - 結果整合性のある読み込み
    - 最新の書き込み結果が即時読み取り処理に反映されない可能性がある
  - Consistent Readオプションを付けたリクエスト
    - GetItem、Query、Scanでは強力な整合性のある読み込みオプションが指定可能
    - Readリクエストを受け取る前までのWriteがすべて反映されたレスポンスを保証
    - Capacity Unitを２倍消費する

DynamoDBの料金体系

- プロビジョンドスループットで決まる時間料金
  - Read/Writeそれぞれプロビジョンしたスループットによって時間あたりの料金がきまる
  - 大規模に利用するのであればリザーブドキャパシティによる割引もあり
- ストレージ利用量
  - 保存したデータ容量によって決まる月額利用料金
  - 計算はGBあたりの単価が適用される
  - 月当たり初めの25GB分は無料
  - GBあたり$0.285(2015/08/05現在@東京リージョン)

http://aws.amazon.com/jp/dynamodb/pricing/

DynamoDBの料金体系

- プロビジョンドスループット
  - Read・Writeそれぞれ25キャパシティユニットまでは無料
  - 書き込み
    - $0.00742 :10 ユニットの書き込み容量あたり/1 時間
  - 読み込み
    - $0.00742：50 ユニットの読み込み容量あたり/1 時間
- キャパシティユニット

  上記で「ユニット」と呼ばれている単位のこと
  - 書き込み
    - 1ユニット：最大1KBのデータを1秒に1回書き込み可能
  - 読み込み
    - 1ユニット：最大4KBのデータを1秒に1回読み込み可能

      （強い一貫性を持たない読み込みであれば、1秒辺り2回）

DynamoDBの料金例

- 米国東部 (バージニア北部) リージョンで実行されているアプリケーションで、DynamoDB テーブルで 8 GB のデータを格納しながら、1 日に 500 万回の書き込みと 500 万回の結果的に整合性のある読み込みを実行する必要があるとします。分かりやすいように、ワークロードは 1 日を通じて比較的一定で、使用しているテーブル項目のサイズは 1 KB 以下であるとします。

- 書き込みキャパシティーユニット (WCU): 1 日あたり 500 万回の書き込みは、1 秒あたり 57.9 回の書き込みと同じです。1 WCU は 1 秒あたり 1 回の書き込みを処理できるので、58 WCU が必要です。1 か月あたり 1 WCU あたり 0.47 USD のため、58 WCU の料金は 1 か月あたり 27.26 USD です。

- 読み込みキャパシティーユニット (RCU): 1 日あたり 500 万回の読み込みは、1 秒あたり 57.9 回の読み込みと同じです。1 RCU は 1 秒あたり 2 回の結果的に整合性のある読み込みを処理できるので、29 RCU が必要です。1 か月あたり 1 RCU あたり 0.09 USD のため、29 RCU の料金は 1 か月あたり 2.61 USD です。

- データストレージ: テーブルは 8 GB のストレージを占有します。1 か月あたり 1 GB あたり 0.25 USD のため、テーブルの料金は 2.00 USD です。
- 総コストは 1 か月あたり 31.86 USD です (プロビジョニングされた書き込みスループットの 27.14 USD、プロビジョニングされた読み込みスループットの 2.71 USD、およびインデックス化データストレージの 2.00 USD)。
- 無料利用枠 (25 WCU、25 RCU、25 GB のストレージ) を消費していない場合、総コストは 1 か月あたり 15.82 USD (残りの 33 WCU に 15.44 USD、残りの 4 RCU に 0.37 USD、インデックス化データストレージは 0.00 USD)。
- さまざまなリソース要件の DynamoDB の月額料金を見積もるには、簡易見積りツールを使用してください。

DynamoDBが使われているユースケース

- KVSとして
  - ユーザー情報の格納するデータベース
- 広告やゲームなどのユーザー行動履歴DBとして
  - ユーザーIDごとに複数の行動履歴を管理するためのデータベース
- モバイルアプリのバックエンドとして
  - モバイルアプリから直接参照できるデータベースとして
- 他にも
  - バッチ処理のロック管理
  - フラッシュマーケティング
  - ストレージのインデックス

DynamoDBを使い始めるには

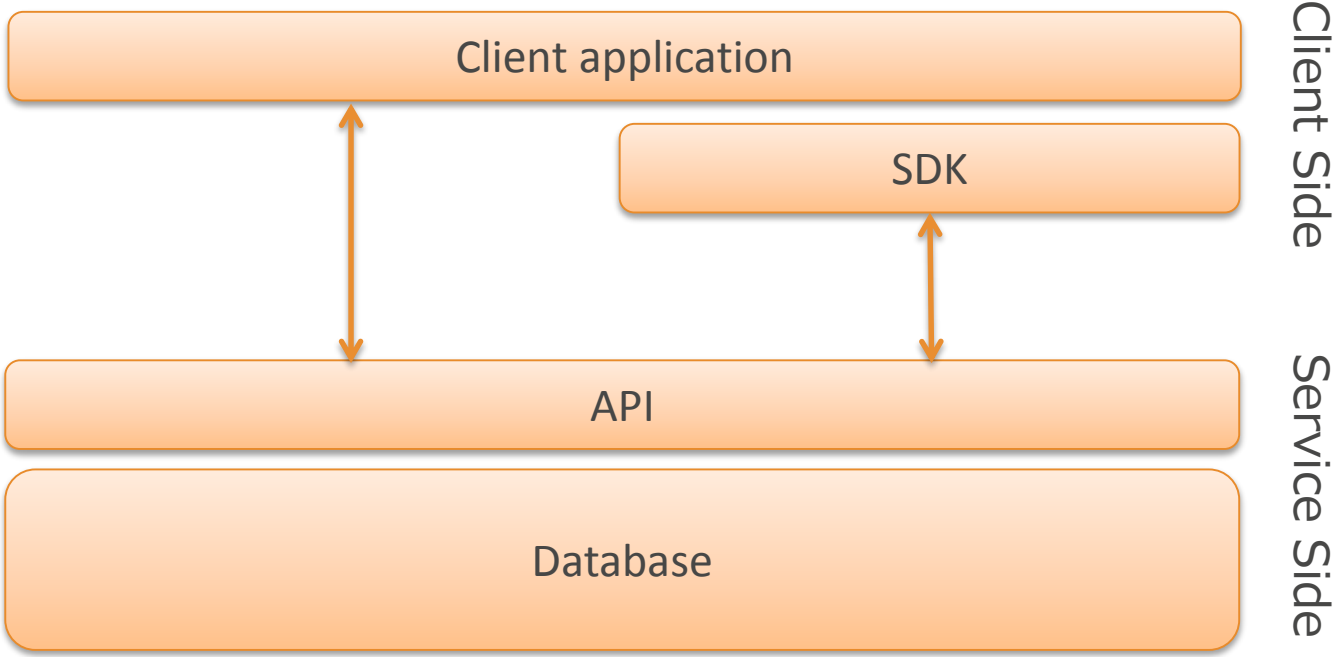
1. テーブルのKeyやIndexを決める

2. Read/Writeそれぞれのスループットを決める

3. DynamoDB Streamsの設定を決める

DynamoDBの構成要素

オペレーションはHTTPベースのAPIで提供されている



提供されているAPI

Table API

- CreateTable
- UpdateTable
- DeleteTable
- DescribeTable
- ListTables

- Query
- Scan

- BatchGetItem
- BatchWriteItem

- GetItem
- PutItem
- UpdateItem
- DeleteItem

DynamoDB

Stream API

- Liststreams
- DescribeStream
- GetShardIterator
- GetRecords

テーブル操作について

よく使われるオペレーションは以下のとおり

- GetItem
  - Partation Keyを条件として指定し、
    一件のアイテムを取得
- PutItem
  - 1件のアイテムを書き込む
- Update
  - 1件のアイテムを更新
- Delete
  - 1件のアイテムを削除
- Query
  - Partation KeyとSort Keyの複合条件にマッチするアイテム群を取得
- BatchGet
  - 複数のプライマリキーを指定してマッチするアイテム群を取得
- Scan
  - テーブル総ナメする

※Query または Scan オペレーションで、
最大 1 MB のデータを取得可能

AWS SDKs and CLI

- 各種言語むけのオフィシャルSDKやCLIを利用

Java

Python

PHP

.NET

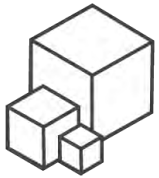
Ruby

nodeJS

Javascript
in the Browser

iOS

Android



AWS CLI

オフィシャルSDK以外にも

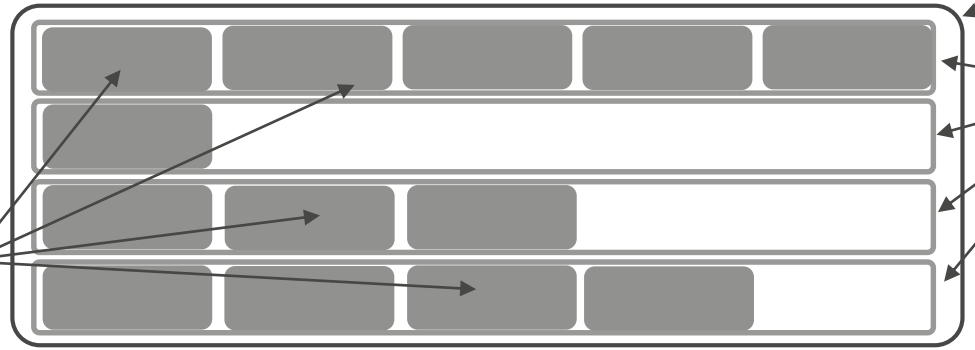
- Perl
  - Net::Amazon::DynamoDB
- Erlang
  - wagerlabs/ddb
    - https://github.com/wagerlabs/ddb
- Go
  - goamz
    - https://github.com/crowdmob/goamz

その他利用できる開発者用リソースはこちら
http://aws.amazon.com/jp/dynamodb/developer-resources/

Agenda

- DynamoDBとは
- テーブル設計とサンプル
- Data Modeling
- ユースケース及びベストプラクティス
- TTL
- Auto Scaling
- DAX
- DynamoDB Streams
- AWS Mobile SDKと 2-Tier アーキテクチャ
- 運用関連
- ツールとエコシステム
- まとめ

テーブル設計
基礎知識



name/value 型、JSON 型等
アイテム間で不揃いであっても問題ない

Partation Key

Sort Key

必須
キーバリュー型のアクセスパターン
データ分散に利用される

オプション
1:Nモデルのリレーションシップ
豊富なQueryをサポート
デフォルトの並べ替え順序は昇順

キー検索用
==, <, >, >=, <=
“begins with”
“between”
sorted results
counts
先頭/末尾 N件
ページ単位出力

Attributesの型

- String
- Number
- Binary
- Boolean
- Null
- 多値データ型
  - Set of String
  - Set of Number
  - Set of Binary
- ドキュメントデータ型
  - List型は、順序付きの値のコレクションを含む
  - Map型は、順序なしの名前と値のペアのコレクションを含む

```
{
    Day: "Monday",
    UnreadEmails: 42,
    ItemsOnMyDesk: [
        "Coffee Cup",
        "Telephone",
        {
            Pens: { Quantity : 3},
            Pencils: { Quantity : 2},
            Erasers: { Quantity : 1}
        }
    ]
}
```

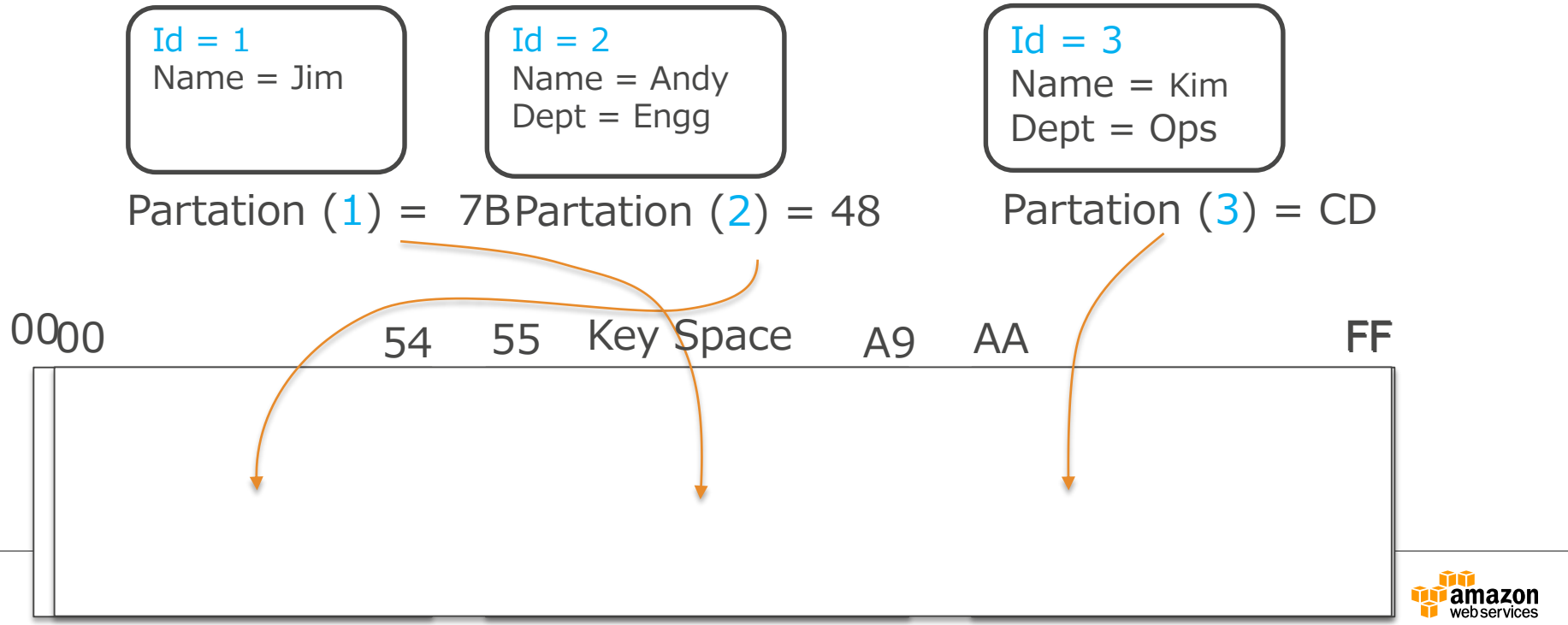
※ Partation Key ,Sort Keyの型は、文字列、数値、またはバイナリでなければならない

DynamoDBのテーブルのプライマリキーの持ち方は２種類

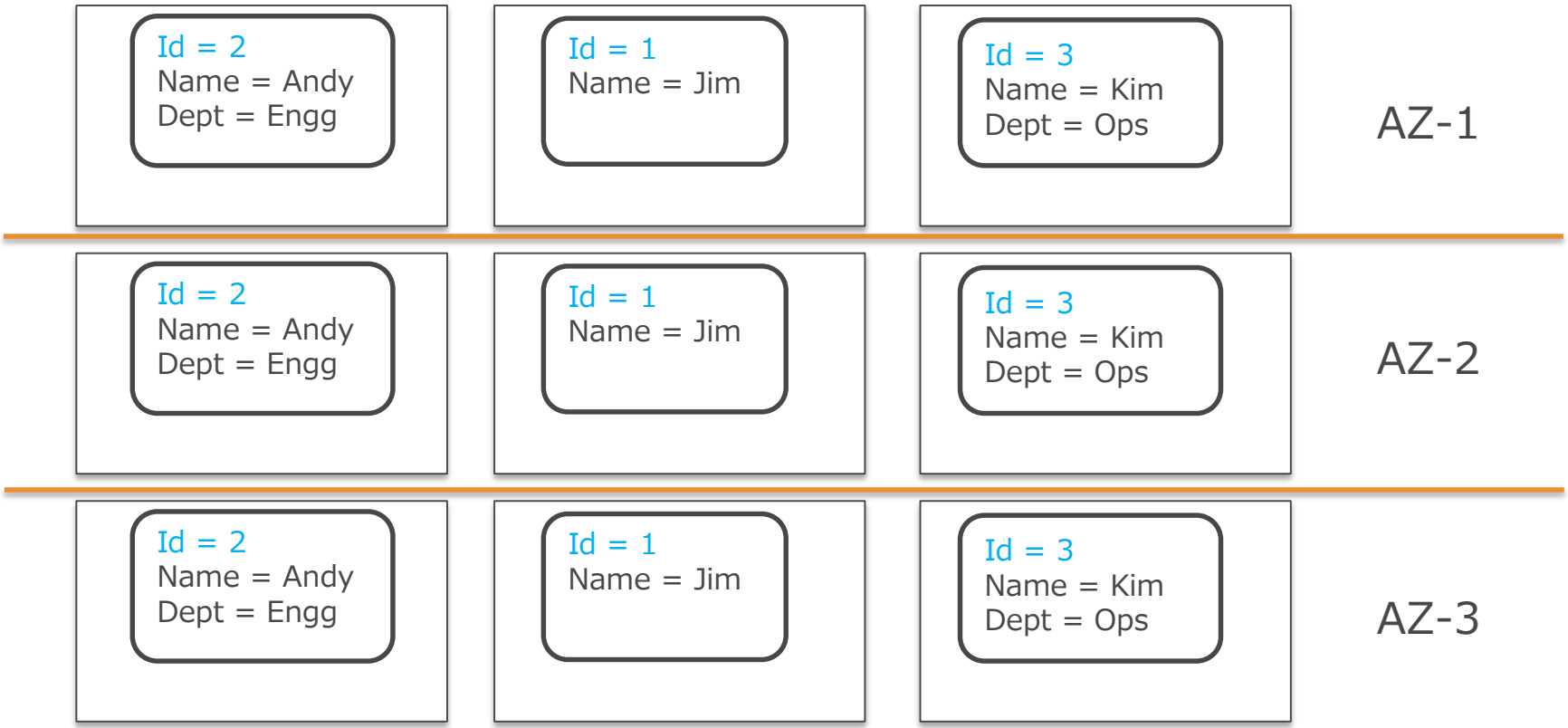
- Partation key
- Partation key & Sort key

Partation Table

- Partation Key は単体でプライマリキーとして利用
- 順序を指定しないハッシュインデックスを構築するためのキー
- テーブルは、性能を確保するために分割(パーティショニング)される場合あり

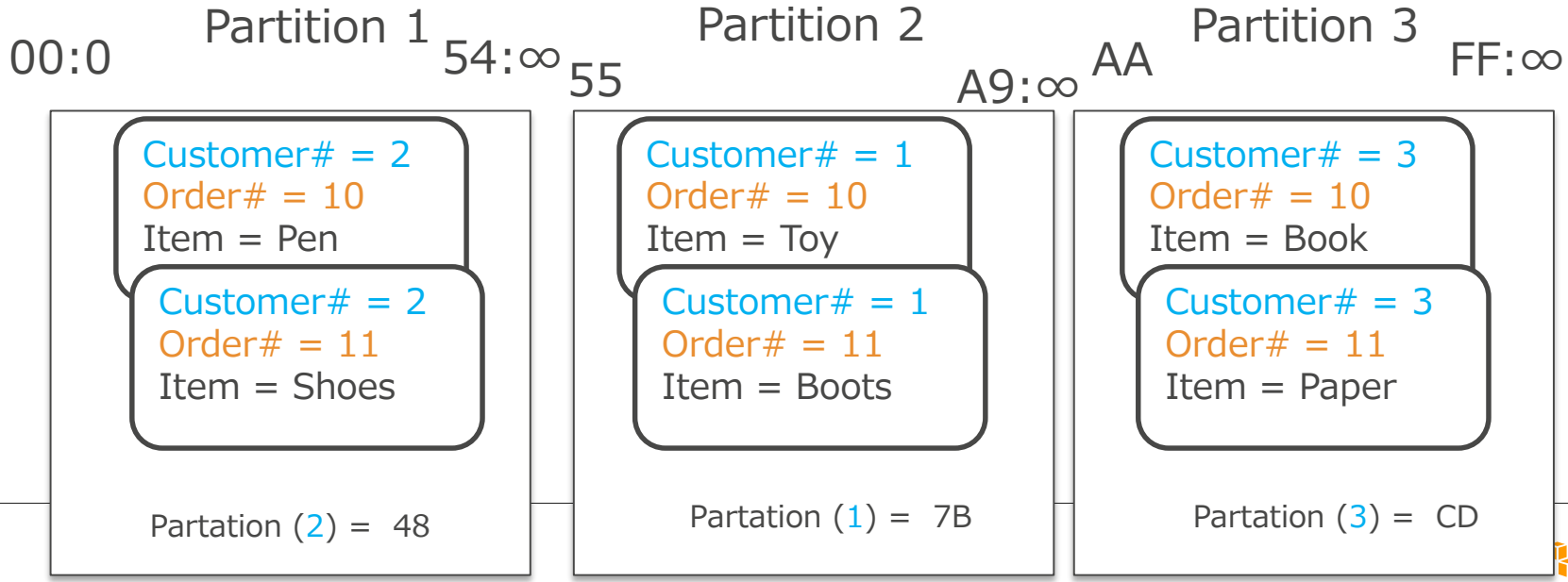


データは3箇所にレプリケーション



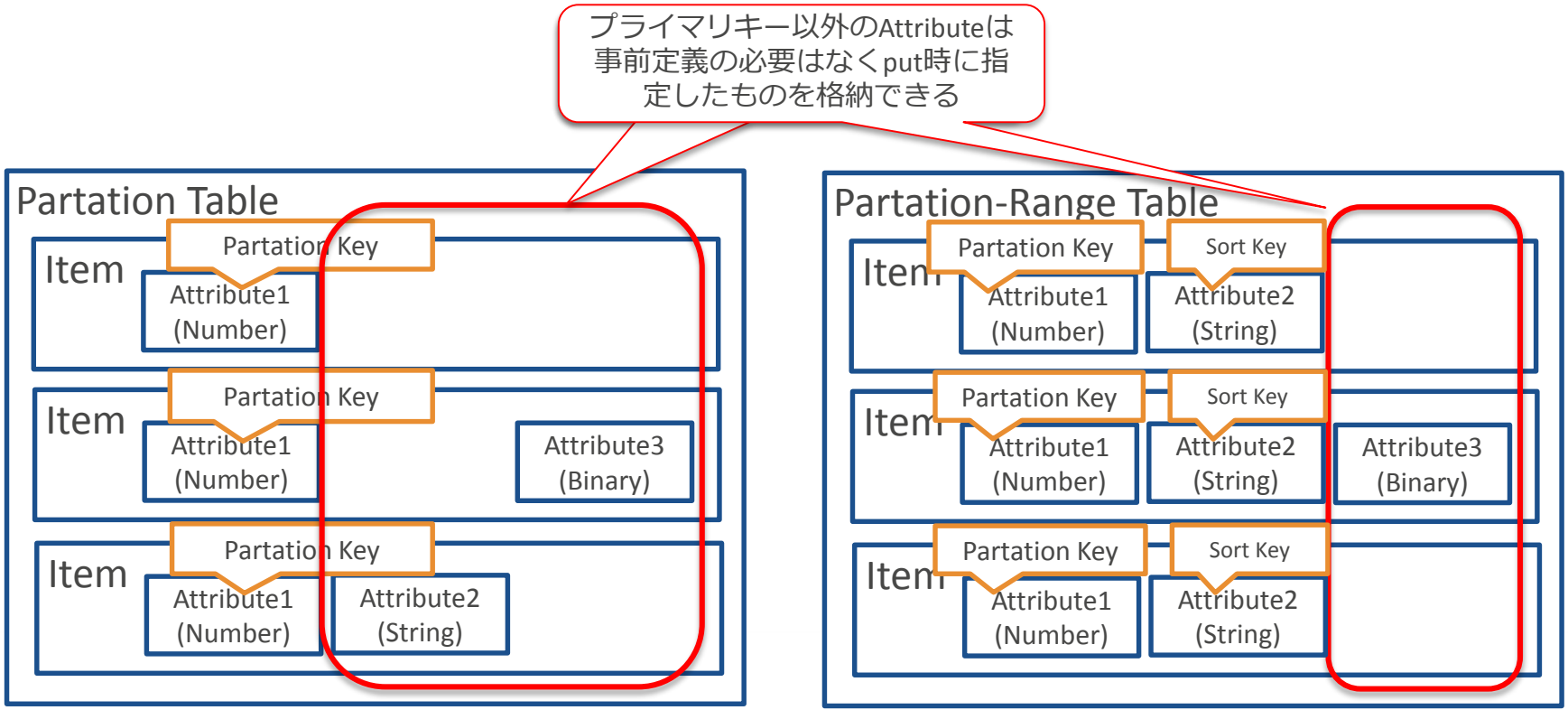
Partation-Sort Table

- Partation + Sortでプライマリキーとすることもできる
- 同一のPartation Keyでのデータの並びを保証するためにSort Keyが使われる
- Partation Keyの数に上限はない（Local Secondary Indexesを使用時はデータサイズ上限あり）



Attributesについて

- Partation、Sortに該当するAttributes以外は事前に定義する必要はない
- レコード間でAttributesが不揃いであっても問題ない



JSONサポート

- JSON(*JavaScript Object Notation*)フォーマットのドキュメントをDynamoDBの1アイテムとして保存可能
  ※サイズ上限として400KB以内が条件
- DynamoDB 内部ではJsonデータをそのまま保持していないため独自に JSON 解析を実装するか、AWS SDK などのライブラリを利用する

```
{
  "person_id" : 123,
  "last_name" : "Barr",
  "first_name" : "Jeff",
  "current_city" : "Tokyo",
  "next_haircut" :
  {
    "year" : 2014,
    "month" : 10,
    "day" : 30
  },
  "children"  :
    [ "SJB", "ASB", "CGB", "BGB", "GTB" ]
}
```

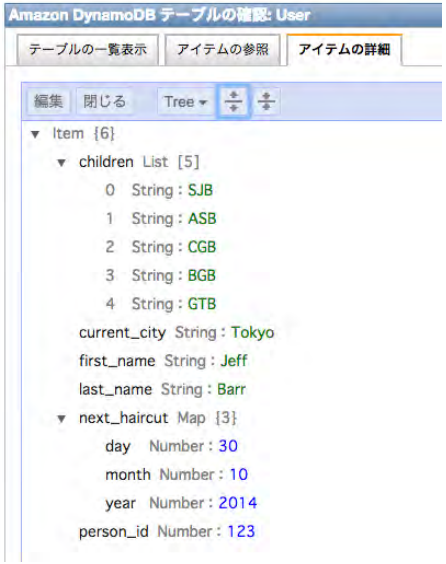
DynamoDBの内部フォーマット

```
{
 "current_city": {
  "S": "Tokyo"
 },
 "first_name": {
  "S": "Jeff"
 },
 "last_name": {
  "S": "Barr"
 },
 "next_haircut": {
  "M": {
   "day": {
    "N": "30"
   },
   "month": {
    "N": "10"
   },
   "year": {
    "N": "2014"
   }
  }
 },
 "person_id": {
  "N": "123"
 }
```

AWSマネージメントコンソールのJSON対応

- AWSマネージメントコンソールからJSONドキュメントの取得、追加、編集が可能

Tree形式



テキスト形式

JSON Document SDK

- シンプルなプログラミングモデル
- JSONから変換、JSONへの変換が可能
- Java, JavaScript, Ruby, .NET, PHP

https://aws.amazon.com/tools/

Javascript	DynamoDB
string	S
number	N
boolean	BOOL
null	NULL
array	L
object	M

```
var image = { // JSON Object for an image
    imageid: 12345,
    url: 'http://example.com/awesome_image.jpg'
};
var params = {
    TableName: 'images',
    Item: image, // JSON Object to store
};
dynamodb.putItem(params, function(err, data)){
    // response handler
};
```

Scaling

- スループット
  - **テーブル**単位で、読み書きのスループットを指定
    （プロビジョニングされたスループット）
- サイズ
  - テーブルには任意の数のアイテムが追加可能
    - 1 つのアイテムの合計サイズは 400 KB
    - local secondary index について、異なるハッシュキーの値ごとに最大 10GB のデータを格納

スループット

- テーブルレベルによってプロビジョニング
  - **Read Capacity Units (RCU)**
    - 1 秒あたりの読み込み項目数 x 項目のサイズ (4 KB ブロック)
    - 結果整合性のある読み込みをする場合はスループットが 2 倍

例1）アイテムサイズ：1.2KB(1.2/ 4≒0.3 ⇒ 1　繰り上げ)
読み込み項目数1000回/秒
1000 × 1 = **1000** RCU

例2）アイテムサイズ:4.5KB (4.5 / 4≒1.1 ⇒ 2　繰り上げ)
読み込み項目数1000回/秒
1000 × 2 = **2000** RCU
※結果整合性のある読み込みの場合
1000 × 2 × ½ = **1000** RCU

スループット

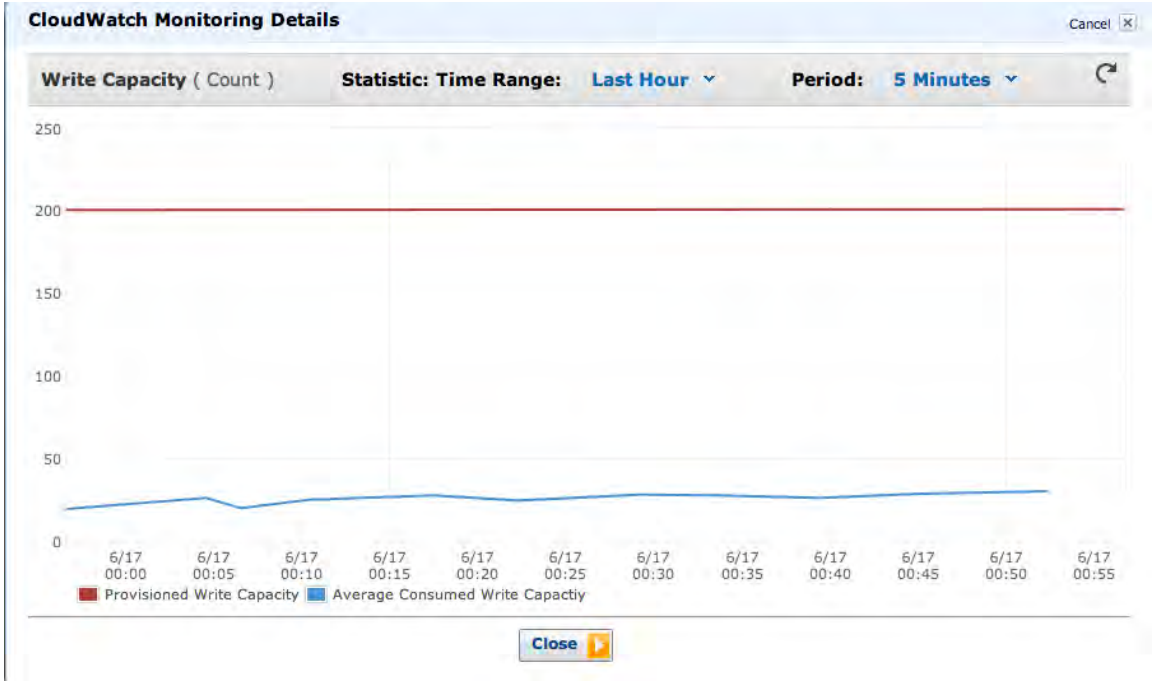
- **Write Capacity Units (WCU)**
  - 1 秒あたりの書き込み項目数 x 項目のサイズ (1 KB ブロック)
  - 1KBを下回る場合は繰り上げられて計算

例1）アイテムサイズ：512B(0.512/ 1≒ 0.5 ⇒ 1　繰り上げ)
書き込み項目数 1000項目/秒
1000 × 1 = **1000** WCU

例2）アイテムサイズ:2.5KB (2.5 / 1 = 2.5 ⇒ 3　繰り上げ)
書き込み項目数1000項目/秒
1000 × 3 = **3000** WCU

- 読み込みと書き込みのキャパシティユニットは個別に設定

スループットの見積もり



①概算の見積もりから、実運用開始時は、キャパシティを大きめに設定

※あまり大きくし設定し過ぎると、パーテション分割時のキャパシティ分割に注意

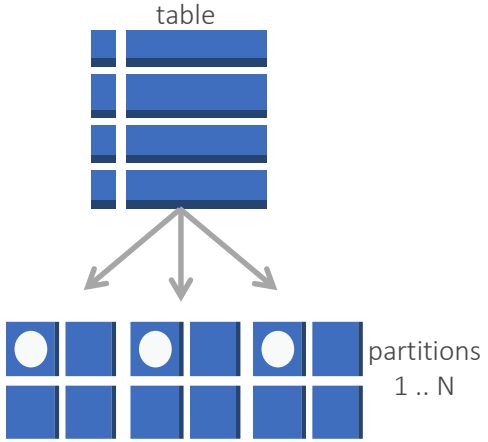
②実運用で様子を見て、キャパシティを調整

パーティション

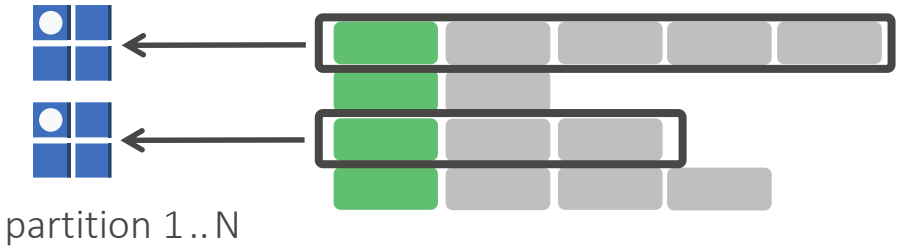
パーティションの数はDynamoDBがマネージするので、ユーザーはパーティション数を気にすることなく運用できるようになっている。逆に言うと直接知る方法はない。しかし、パーティション分割の特性を理解しておくことでより便利にDynamoDBを活用できるようになる。

パーティショニング

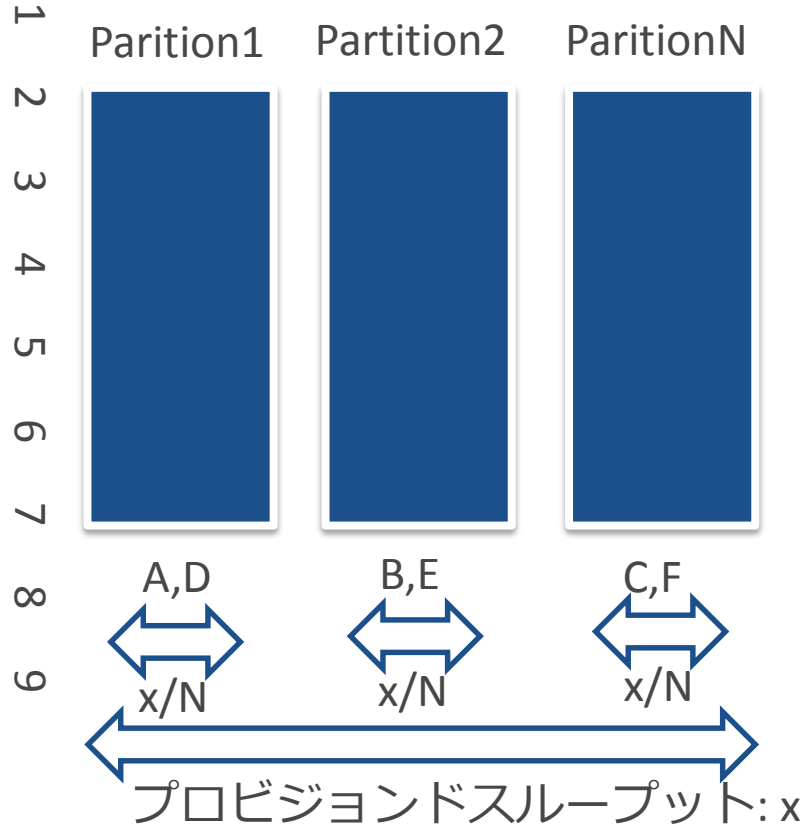
- DynamoDBはプロビジョンされたスループットを確保するためにテーブルを複数のパーティションに分散して格納



- Partation Keyをパーティション間でのデータ分散に利用し、格納ストレージサイズやプロビジョンされたスループットによって自動的にパーティショニングを実施



パーティション内でのスループット



スループットはパーティションに均等に付与される

- プロビジョンしたスループットは各パーティションに均等に付与され、全体で合計値の性能が出るように設計されている
- アクセスされるキーに偏りが発生すると思うように性能がでないという自体を招くため、Partation Keyの設計には注意が必要

パーティショニング数の算出

パーティション数はストレージ容量とスループットで決まる

①DynamoDB では、1 つのパーティションに対して、最大 3,000 個の読み込みキャパシティーユニットまたは 1,000 個の書き込みキャパシティーユニットを割り当てられる。

$$\#\ of\ Partitions_{(for\ throughput)} = \frac{RCU_{for\ reads}}{3000\ RCU} + \frac{WCU_{for\ writes}}{1000\ WCU}$$

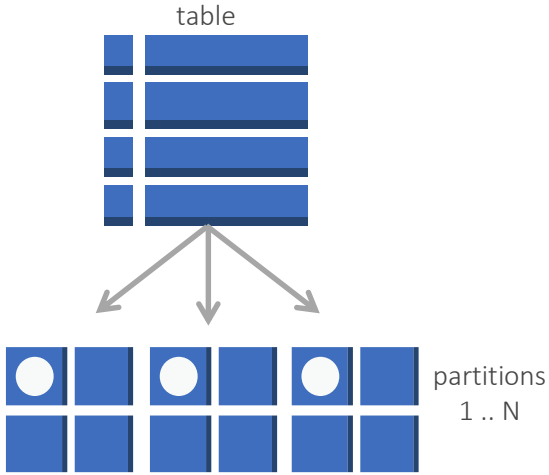
②単一のパーティションには、約 10 GB のデータを保持される

$$\#\ of\ Partitions_{(for\ size)} = \frac{Table\ Size\ in\ GB}{10\ GB}$$

パーティショニング数の算出

- 大きいほうを採用

$$\underset{(total)}{\#\ of\ Partitions} = \mathrm{MAX}(\underset{(for\ size)}{\#of\ Partitions} \mid \underset{(for\ throughput)}{\#of\ Partitions})$$



パーティショニング数の算出例

Table size = 8 GB, RCUs = 5000, WCUs = 500

- スループット

$$\#\ of\ Partitions_{(for\ throughput)} = \frac{5000_{RCU}}{3000\ RCU} + \frac{500_{WCU}}{1000\ WCU} = 2.17 = 3$$

- ストレージサイズ

$$\#\ of\ Partitions_{(for\ size)} = \frac{8\ GB}{10\ GB} = 0.8 = 1$$

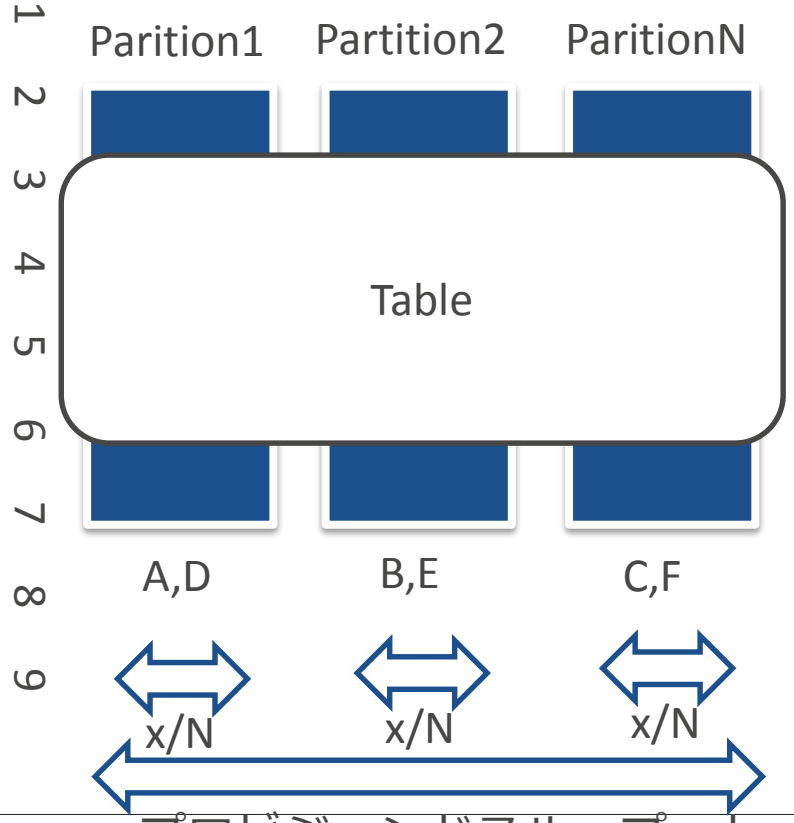
- 大きいほうを採用

$$\#\ of\ Partitions_{(total)} = \mathrm{MAX}(1_{\ for\ size} | 3_{\ for\ throughput}) = 3$$

RCUとWCU の値は均一に各パーテションに割り当てられる

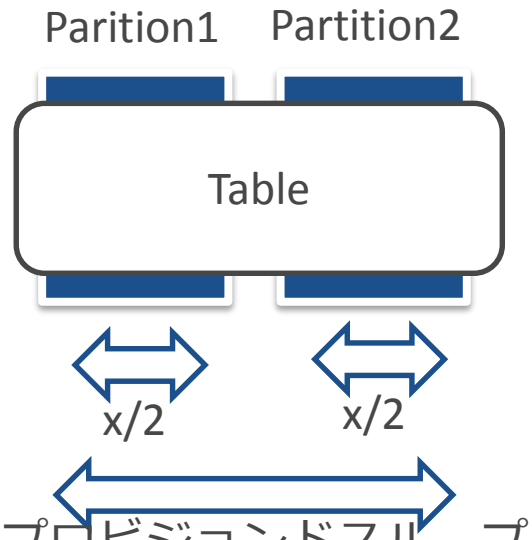
RCUs per partition = 5000/3 = 1666.67
WCUs per partition = 500/3 = 166.67
Data/partition = 8/3 = 2.66 GB

Capacity変更時の注意事項

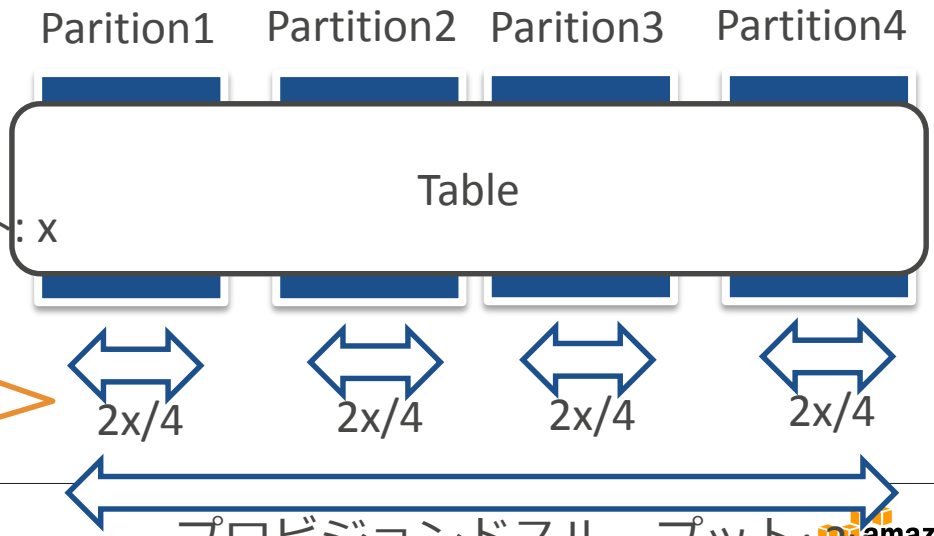


各パーティションには、テーブルの総スループットをパーティション数で割った分が均等にわりあてられている。

Capacity変更時の注意事項

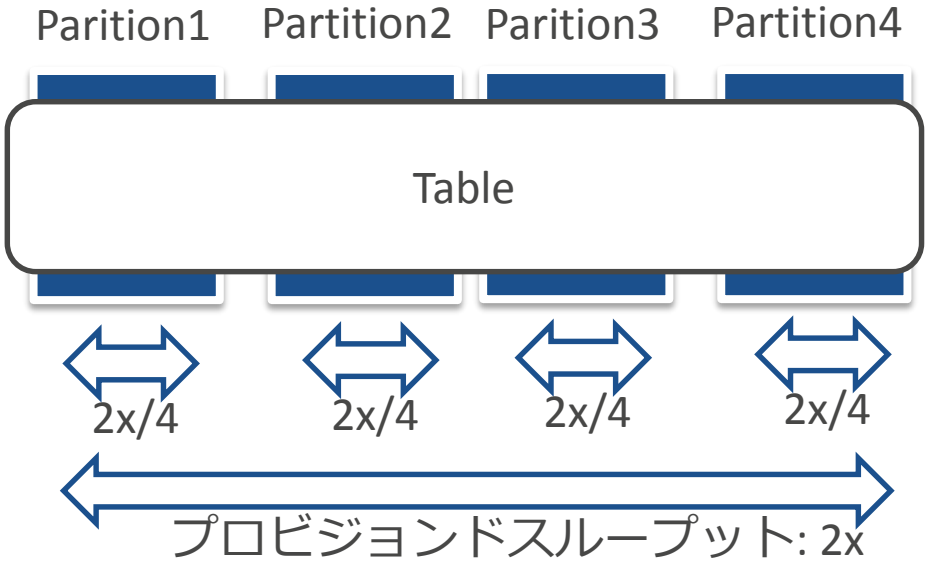


キャパシティのプロビジョンを増やすと、前述の計算式にしたがってパーティション数も増えていく

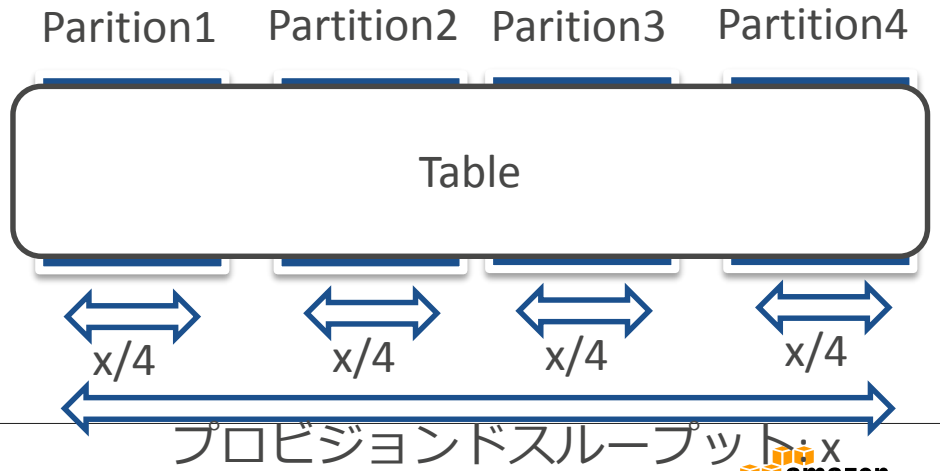


前述の計算式の値をMAXとし、各パーティションにキャパシティがAllocateされていき、MAXを超えるとパーティションが追加される

Capacity変更時の注意事項



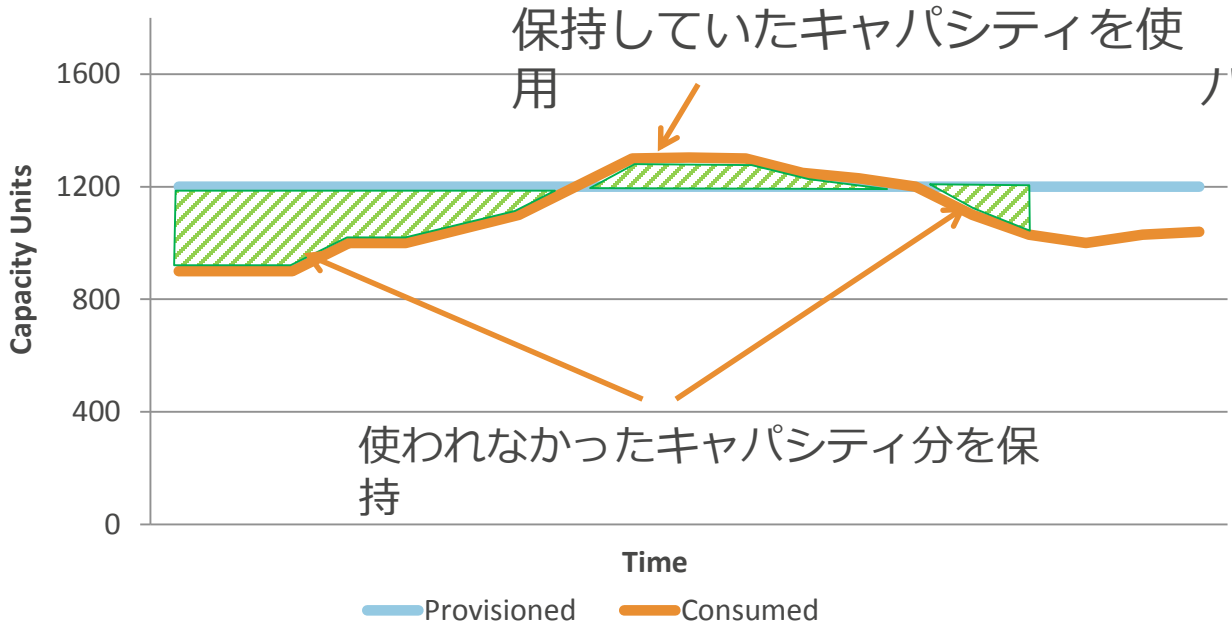
キャパシティを減らすときはパーティション数は減らずに、各パーティションのキャパシティが減るので注意！



ひとつひとつのパーティションのキャパシティが低い状態になる。
結果としてキャパシティエラーを起こしやすい状態になる。

Burst Capacity

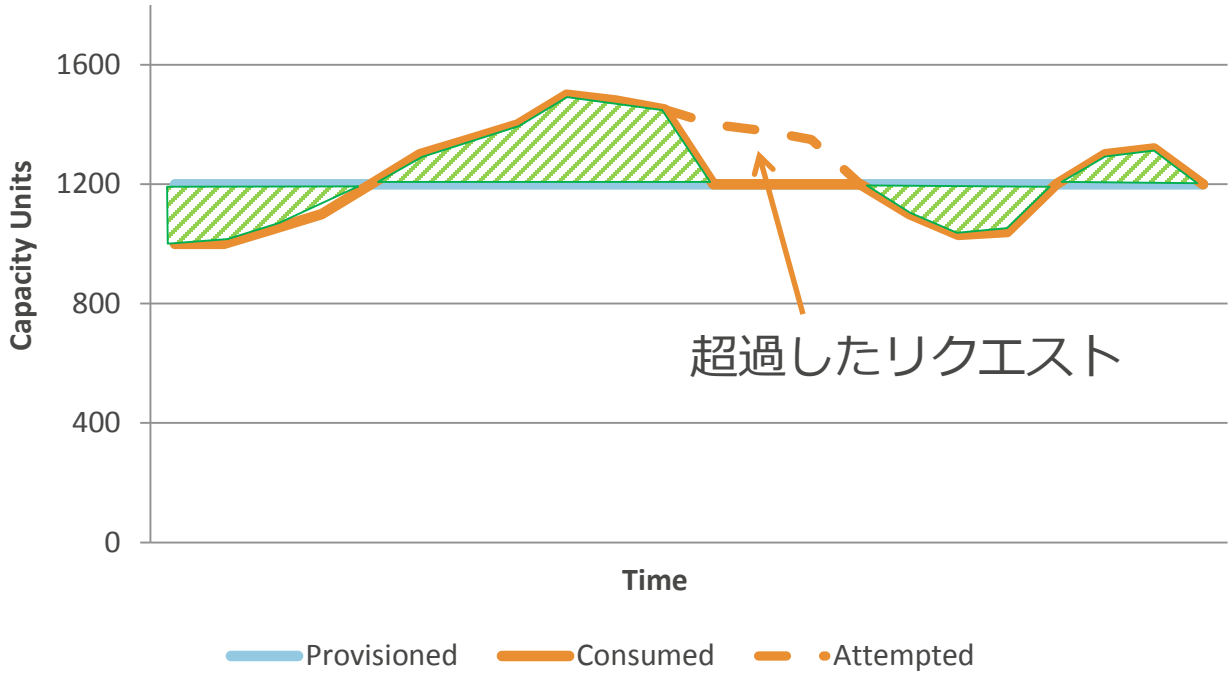
DynamoDBはパーティションごとのキャパシティのうち、利用されなかった分を過去300秒分までリザーブ
プロビジョン分を超えたバーストラフィックを処理するために利用する



バーストキャパシティ: 300秒
(1200 × 300 = 3600 CU)

Burst Capacityでは不十分な場合もある

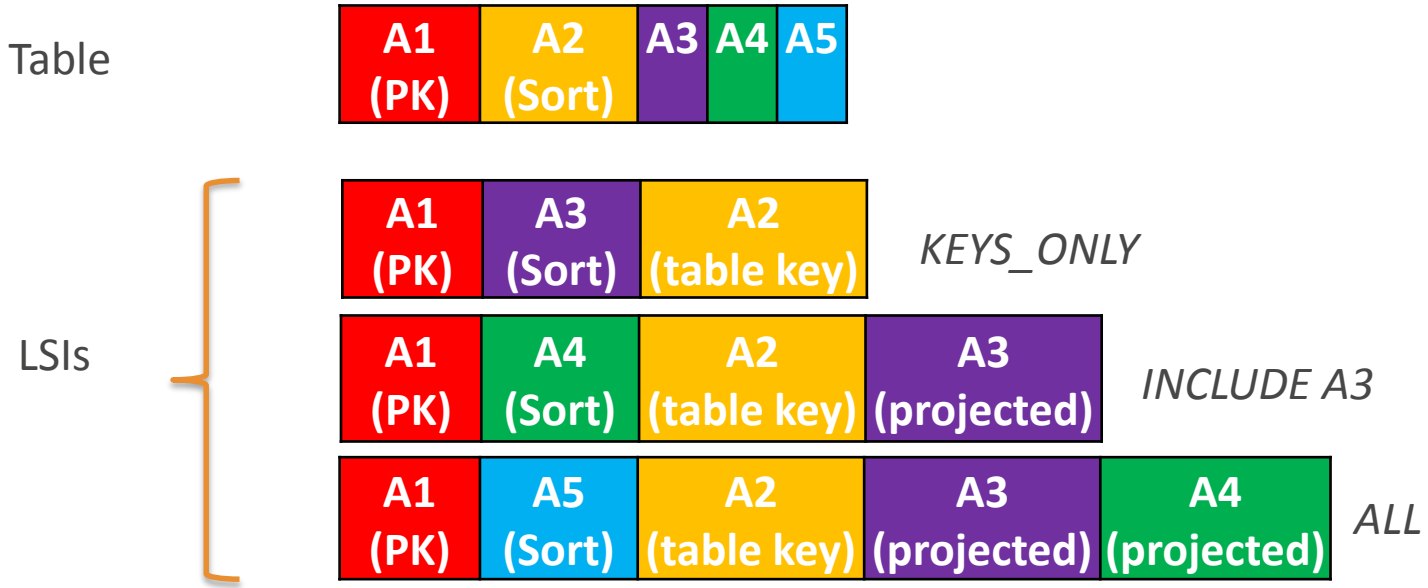
バーストキャパシティ: 300秒
(1200 × 300 = 3600 CU)



バーストキャパシティの機能に依存しないように。

Local Secondary Index (LSI)

- Sort key以外に絞り込み検索を行うkeyを持つことができる
- Partition keyが同一で、他のアイテムからの検索のために利用
- すべての要素（テーブルとインデックス）の合計サイズを、各ハッシュキーごとに 10 GB に制限



LSI利用例：自分の友だちリストの検索用

Table

PK	Sort Key	つながり
太郎	次郎	サッカー
太郎	花子	野球
太郎	三郎	サッカー

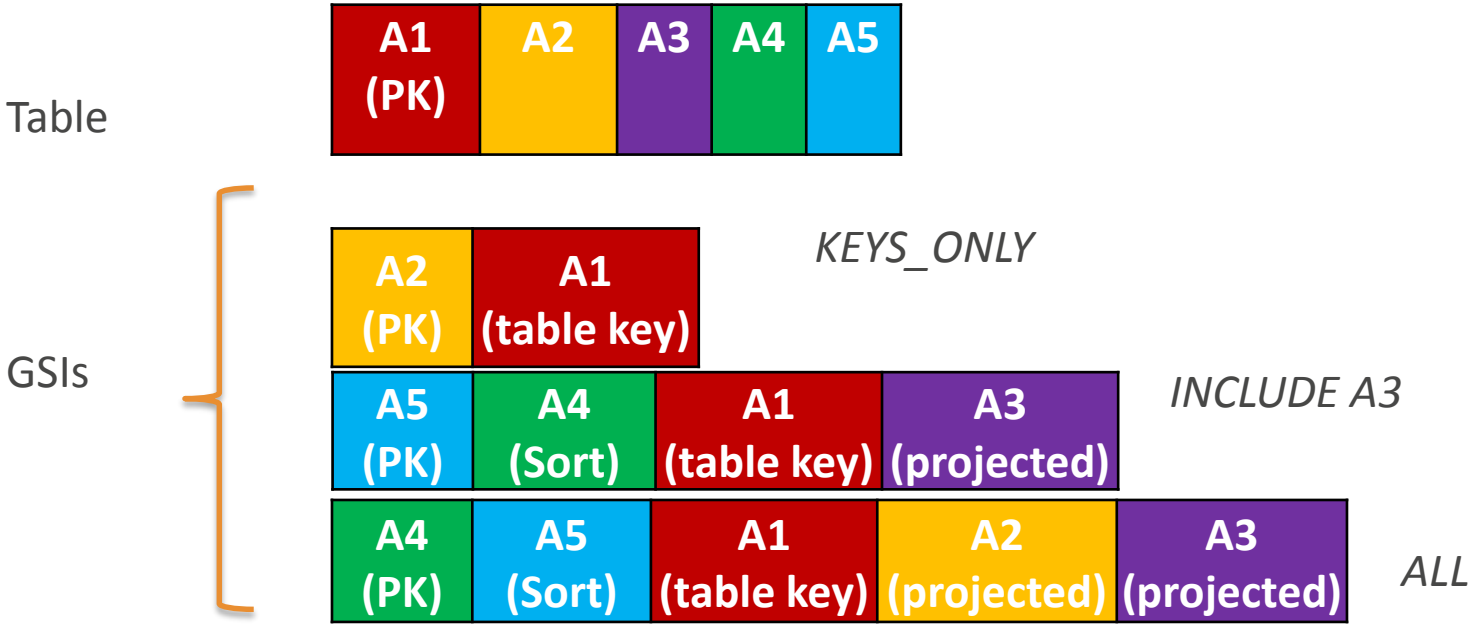
LSI

PK	Sort Key	つながり
太郎	サッカー	次郎
太郎	サッカー	三郎
太郎	野球	花子

- 太郎さんの友達の次郎は何で繋がっている？が元のテーブルでは取得可能
- 太郎さんのサッカー友達は誰？がLSIで検索可能
- Queryを使う事で太郎さんの友達はそもそも誰がいる？太郎さんは何繋がりの友達がいる？も両方のテーブルを使う事で検索可能

Global Secondary Index (GSI)

- Partition Key属性の代わりとなる
- Partition Keyをまたいで検索を行うためのインデックス



GSI利用例：全体友だちリストの検索用

Table

PK	Sort	つながり
太郎	次郎	サッカー
太郎	三郎	野球
幸子	花子	テニス

GSI マッピング用

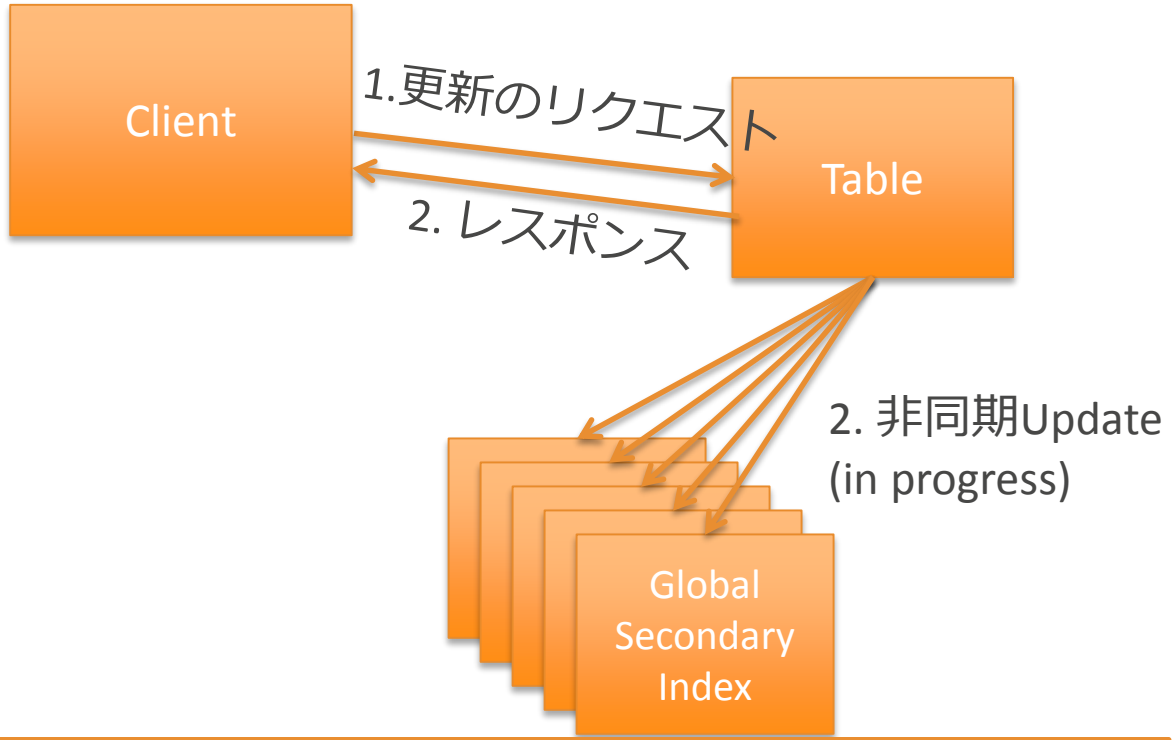
PK	Sort
次郎	太郎
三郎	太郎
幸子	花子

GSI つながりからの検索用

PK	Sort	被友達
サッカー	太郎	次郎
野球	太郎	三郎
テニス	幸子	花子

- 次郎さんの友達は誰がいる？と言った逆引きをマッピング用GSIで検索可能
- テニスで太郎さんや幸子さんと繋がっている人は誰がいる？がつながりからの検索用GSIで検索可能

GSIの更新フロー



GSIにはテーブルとは独立したスループットをプロビジョンして利用するため
十分なスループットが必要

LSI/GSIを使う際の注意点

- LSI/GSIは便利だが、スループットやストレージ容量を追加で必要とする
- 特にインデックスの数が増えれば増えるほど書き込みコストが上がる
- セカンダリインデックスに強く依存するようなテーブル設計になるようであれば、一度RDBで要件を満たせないかを確認してみるのがベター

高度なテーブル操作についてのオペレーション

- 強い一貫性を持ったReadオペレーション
  - GetItem、Query、ScanにはConsistent Readというオプションを指定することができ、これを利用すると、Readリクエストを受け取るよりも前に成功しているすべてのWriteリクエストが反映された結果が返る。Read Capacity Unitを通常の2倍消費するので注意。
- Conditional Write
  - 「キーにマッチするレコードが存在したら/しなかったら」や「この値が○○以上/以下だったら」という条件付き書き込み/更新ができる
- Filter式による結果の絞込
  - Query または Scanでは、必要に応じてフィルタ式を指定して、返された結果を絞り込むことができる
  - Query、Scanの１MBの制限はフィルタ式の適用前に適用。また、消費されるキャパシティユニットもフィルタ式を指定しなくても同じ

高度なテーブル操作についてのオペレーション

- UpdateItemにおけるAttributeへの操作
  - Attributeに対して、UpdateItemでPut、Add、Deleteという3種類の操作が可能
    - Put：Attributeを指定した値で更新
    - Add：AttributeがNumber型なら足し算/引き算、Set型ならそのセットに対して値を追加する
    - Delete：当該Attributeを削除する
- Atomic Counter
  - 上記のAddを利用することによって、Atomicなカウンターを実現することもできる

Data Modeling

1:1 リレーション or キー・バリュー型

- Partition keyを使ったテーブルまたはGSI
- GetItem かBatchGetItem APIを使用

例: UserIDやEmailから要素を抽出する場合

Users Table	
Partition key	Attributes
UserId = bob	Email = bob@gmail.com, JoinDate = 2011-11-15
UserId = fred	Email = fred@yahoo.com, JoinDate = 2011-12-01

Users-email-GSI	
Partition key	Attributes
Email = bob@gmail.com	UserId = bob, JoinDate = 2011-11-15
Email = fred@yahoo.com	UserId = fred, JoinDate = 2011-12-01

1:N リレーション or 親子関係

- Partition key とSort key を使ったテーブル、GSI
- Query APIを使ってアクセス

例:1ユーザでN個のゲームをプレイしている場合

User-Games		
Partition Key	Sort key	Attributes
UserId = bob	GameId = Game1	HighScore = 10500, ScoreDate = 2011-10-20
UserId = fred	GameId = Game2	HIghScore = 12000, ScoreDate = 2012-01-10
UserId = bob	GameId = Game3	HighScore = 20000, ScoreDate = 2012-02-12

N:M リレーション

- table と GSI を使用してPartition key とSort key の要素をスイッチして設計
- Query API を用いてアクセス

例: 1ユーザが複数のゲームをプレイし,

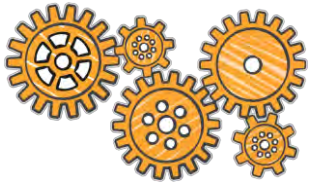
1ゲームで複数のプレイヤーがゲームをしている場合

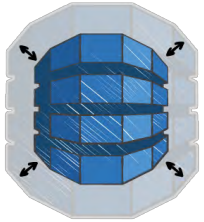
User-Games-Table	
Partition Key	Sort key
UserId = bob	GameId = Game1
UserId = fred	GameId = Game2
UserId = bob	GameId = Game3

Game-Users-GSI	
Partition Key	Sort key
GameId = Game1	UserId = bob
GameId = Game2	UserId = fred
GameId = Game3	UserId = bob

ユースケース及び、ベストプラクティス

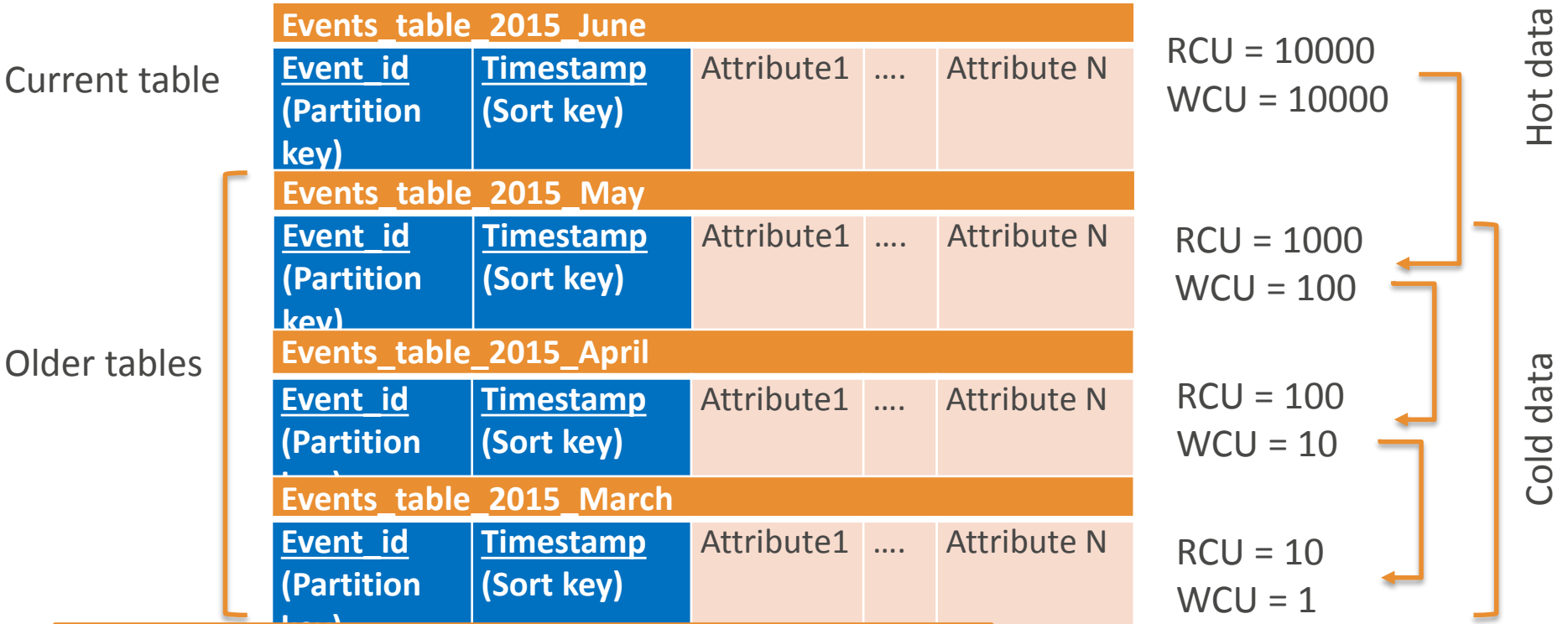
イベントログ





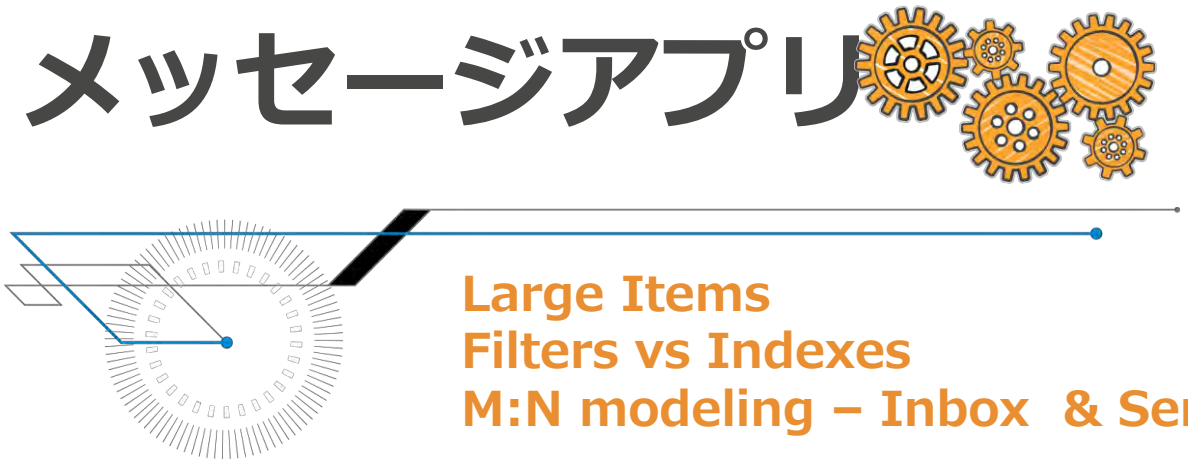
Storing time series data

Time Series Tables



ホットデータとコールドデータは混在させない
アーカイブしたコールドデータはS3へ

メッセージアプリ



Large Items
Filters vs Indexes
M:N modeling – Inbox & Sent Items

Messages App

Messages Table

Inbox

```
SELECT *
FROM Messages
WHERE Recipient='David'
LIMIT 50
ORDER BY Date DESC
```

Outbox

```
SELECT *
FROM Messages
WHERE Sender ='David'
LIMIT 50
ORDER BY Date DESC
```

大小のデータが混在

David

Inbox

```
SELECT *
FROM Messages
WHERE Recipient='David'
LIMIT 50
ORDER BY Date DESC
```

Partition key

Sort key

Messages Table

Recipient	Date	Sender	Message
David	2014-10-02	Bob	…
… 48 more messages for David …			
David	2014-10-03	Alice	…
Alice	2014-09-28	Bob	…
Alice	2014-10-01	Carol	…

(Many more messages)

50 items × 平均 256 KB

大きなメッセージボディーを格納

クエリーコストの計算

$$50_{items} \times 256_{KB} \times \frac{1_{RCU}}{4_{KB}} \times \frac{1_{read}}{2_{e.c.reads}} = \boxed{\mathbf{1600_{RCU}}}$$

1回の問い合わせによって取得されるアイテム数

平均アイテムサイズ

4KB毎に1RCU消費

結果整合性のある読み込み

大きいデータを分けて配置

均等に大きいアイテムを読むように配置

David

(50 sequential items at 128 bytes)

1. Query Inbox-GSI: 1 RCU
2. BatchGetItem Messages: 1600 RCU

(50 separate items at 256 KB)

(Recipientをindex,
Message メタデータを格納)

Inbox-GSI

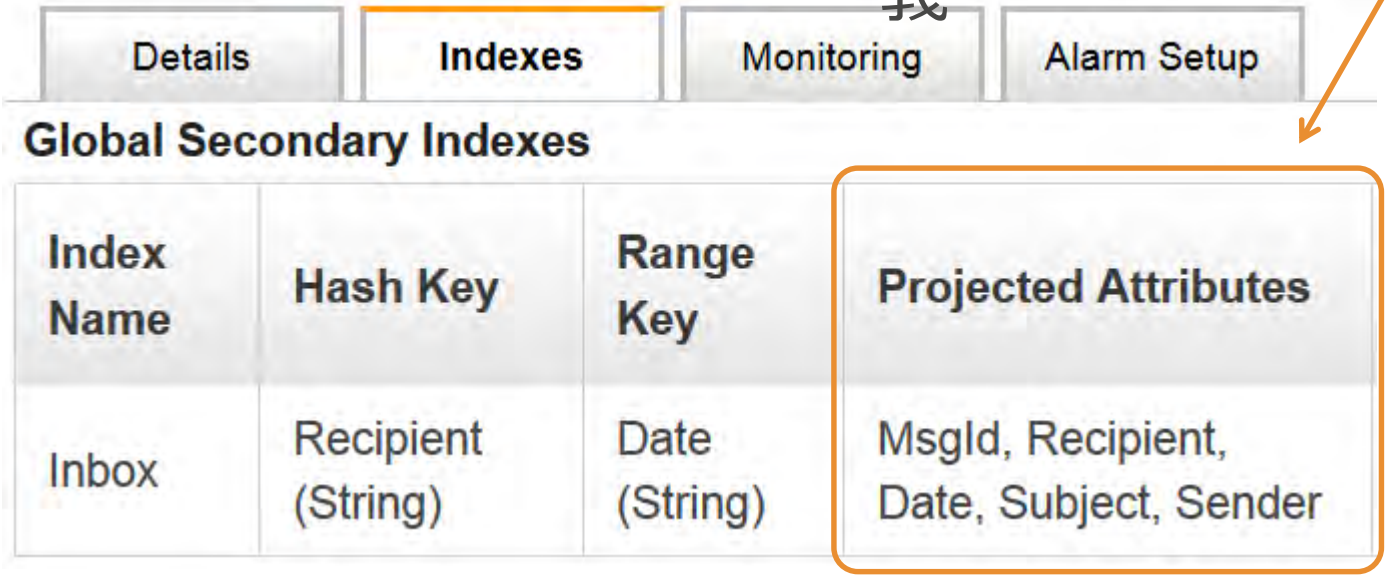
Recipient	Date	Sender	Subject	MsgId
David	2014-10-02	Bob	Hi!...	afed
David	2014-10-03	Alice	RE: The...	3kf8
Alice	2014-09-28	Bob	FW: Ok...	9d2b
Alice	2014-10-01	Carol	Hi!...	ct7r

Messages Table

MsgId	Body
9d2b	...
3kf8	...
ct7r	...
afed	...

Inbox GSI

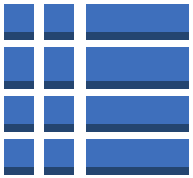
Indexにコピーする属性を定義



書き込み処理を単純化

```
PutItem
{
  MsgId: 123,
  Body: ...,
  Recipient: Steve,
  Sender: David,
  Date: 2014-10-23,
  ...
}
```

David

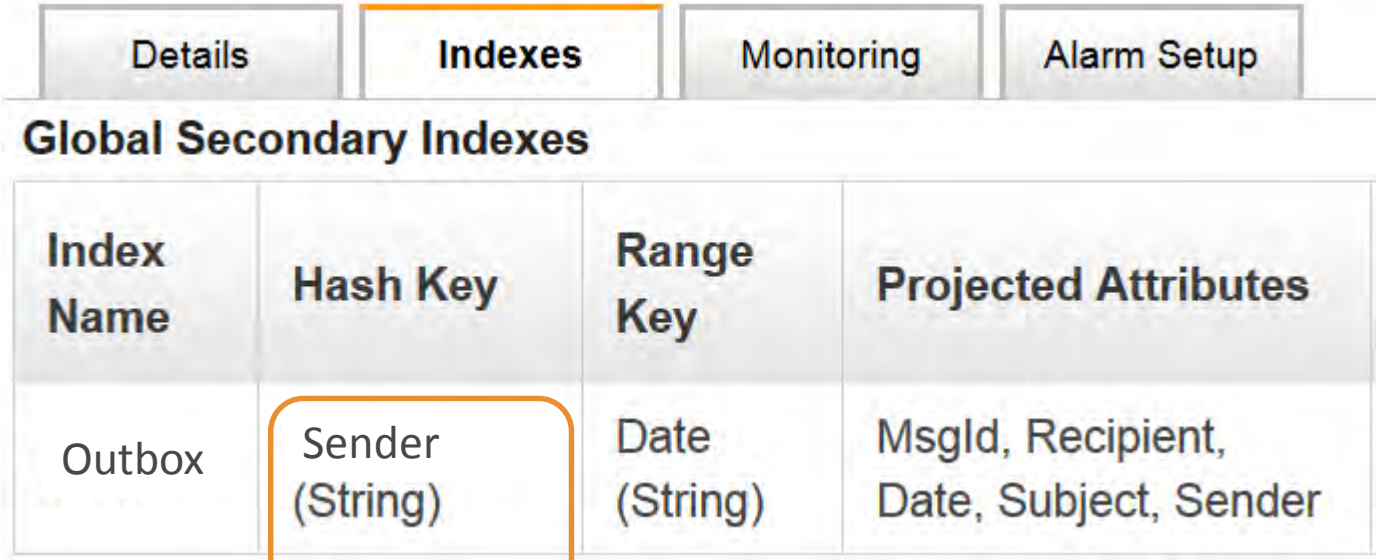


Inbox

Global secondary index

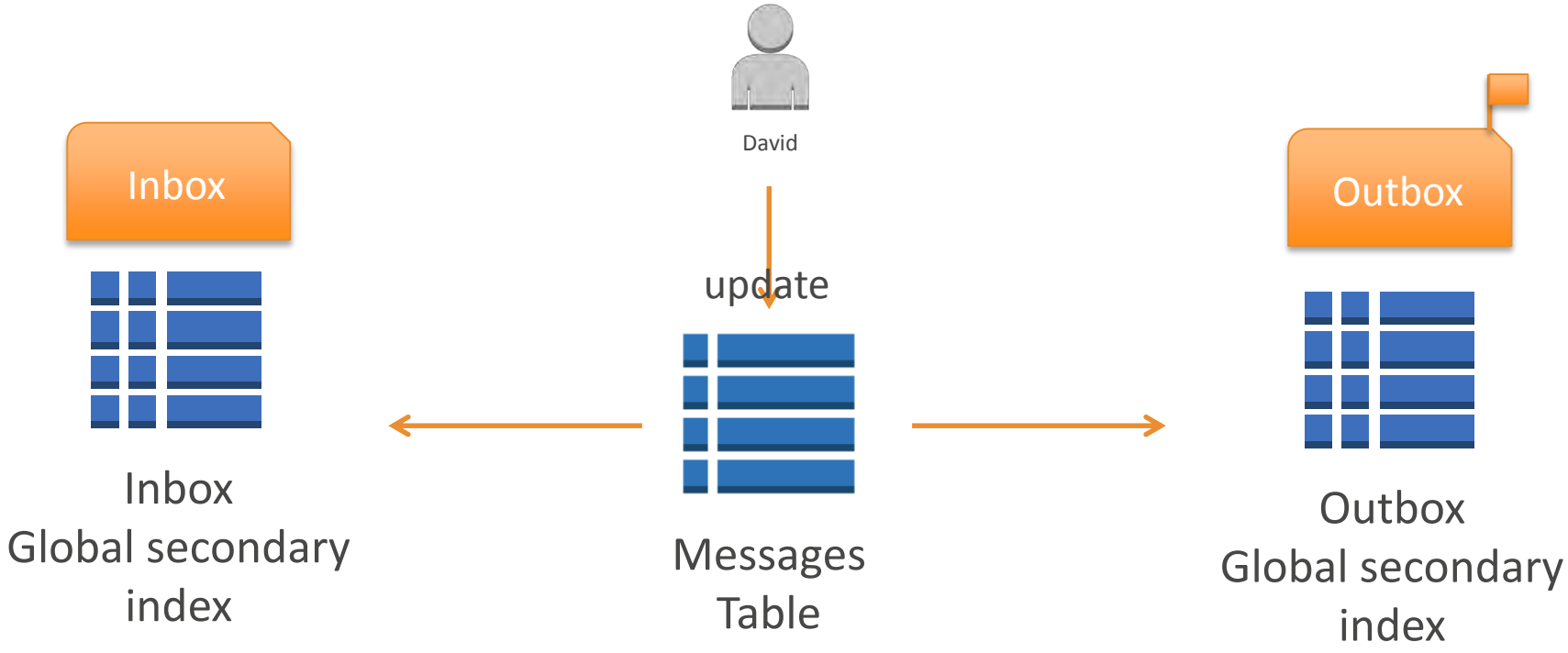
Messages Table

Outbox GSI




```
SELECT *
FROM Messages
WHERE Sender ='David'
LIMIT 50
ORDER BY Date DESC
```

Messaging app



紹介する新機能

- TTL
- Auto Scaling
- DynamoDB Accelerator （DAX）

TTL

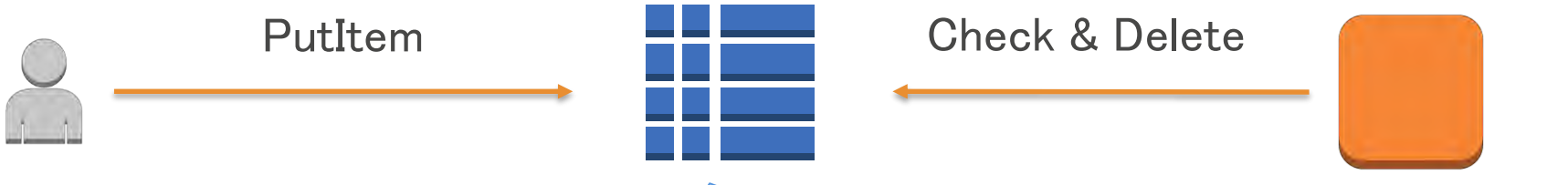
TTL（Time To Live）

- 特徴
- DynamoDB の Time To Live (TTL) では、テーブルの項目の有効期限が切れ、データベースから自動的に削除できるタイミングを定義出来る
- プロビジョニングされたスループットを使用することなく、関連性のないデータのストレージ使用量と保存コストを減らせる
- 追加料金なしで提供
- 既存・新規のテーブルに設定可能
- DynamoDB Streamsとの併用可能

TTL（Time To Live）

- TTL属性にしたい属性を設定し有効にするだけで完了。
- 属性内の値は時間をエポック形式で含む数値データ型のみ

TTLの仕組み



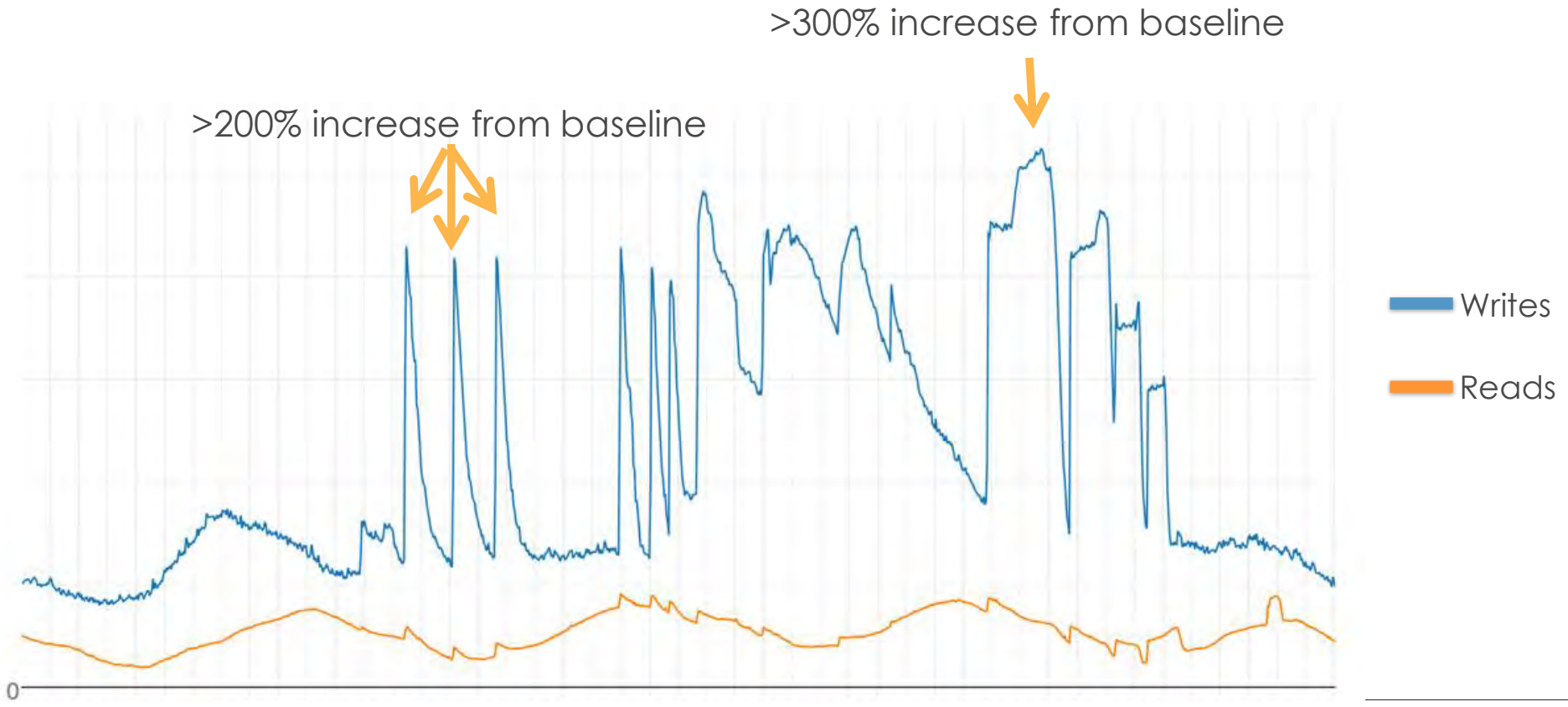
SessionData

UserName	SessionId	CreationTime	ExpirationTime (TTL)
user1	aaaaa	1461931200	1461938400
user2	bbbb	1461920400	1461927600

TTL（Time To Live）

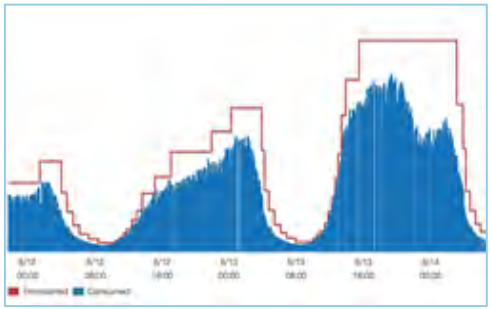
- 注意点
- TTLはUnixTimeで設定するが期限切れになっても即削除・読み取り出来なくなる訳では無い。
- 最大48時間削除まで掛かる（ドキュメント記載）
- その為、読み取り時に期限切れのものを取得しないようにするにはQueryを利用するかアプリ側でフィルタ処理が必要

Auto Scaling


>300% increase from baseline
>200% increase from baseline
Writes
Reads
0

Auto Scaling

Auto Scaling無し



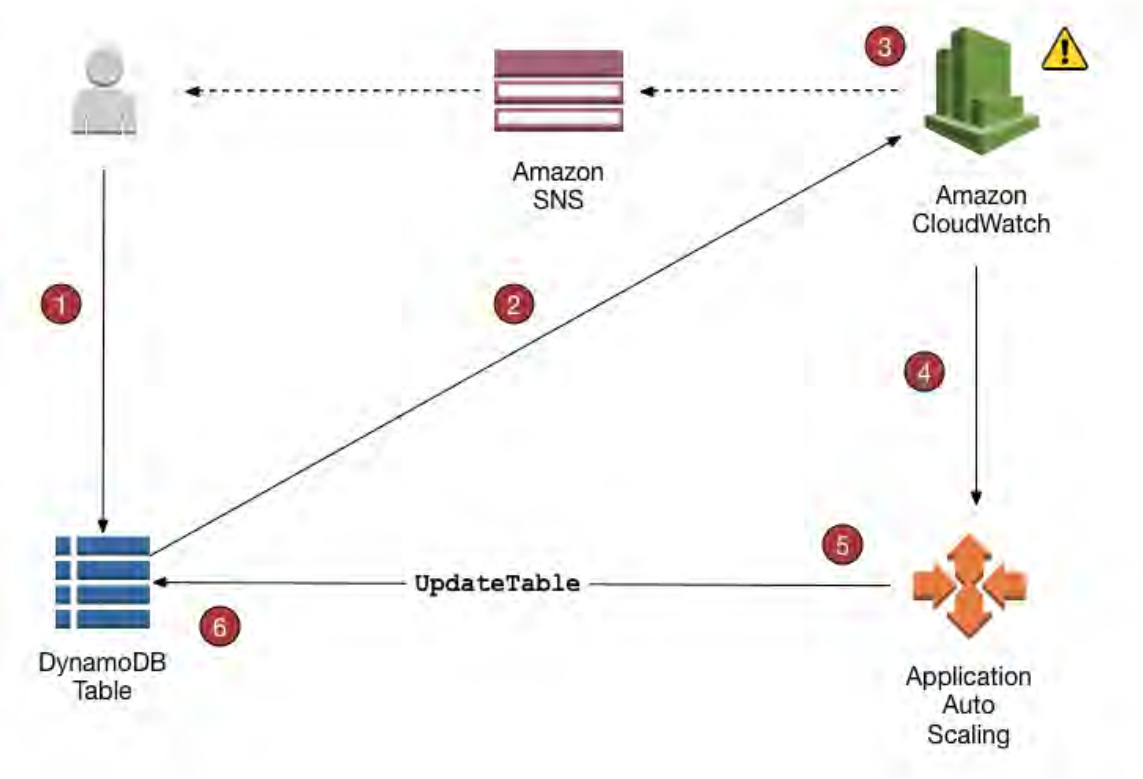
Auto Scaling 有効

- フルマネージドでWCU、RCU、GSIに対する設定を管理
- 設定はターゲット使用率と上限、下限を設定するだけ
- マネジメントコンソール、CLI、SDKでの操作が可能
- 利用は無料

どんな利点が？

- 今まではどれくらい使っていたかを推測してWCU、RCUを設定していた作業から解放
- アプリケーションからのリクエスト数に応じて自動的に容量を拡大
- リクエストが減った時に自動的に容量を縮小してコストを削減
- マネジメントコンソールから可視化された状態で管理が可能

Auto Scalingの概要



Auto Scaling

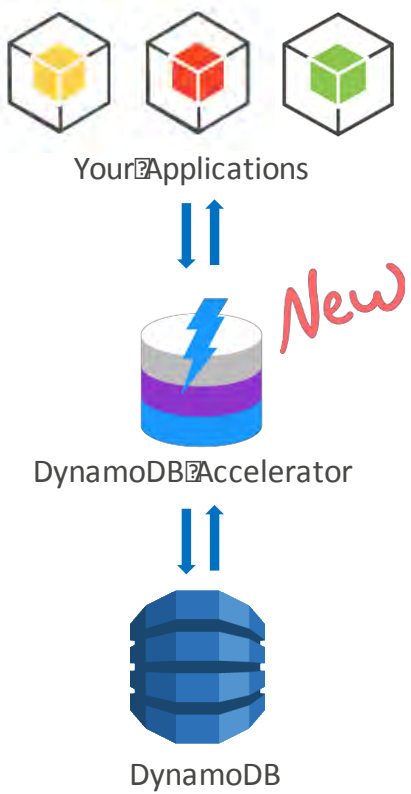
- 設定について
  - 上限と下限と目標使用率を設定することで常にその範囲の中で指定した使用率を維持しようとする。GSIも同様にするか個別にするかなど設定も可能

Auto Scaling

- 注意事項
  - Auto Scalingが発動すると即座に容量が拡大する訳ではない。EC2でAuto Scalingをやるのと同様。
  - 瞬間的なスパイクに対応するには次に紹介するDAXと合わせてアーキテクチャを組むことを推奨
  - 上げる回数の上限は無いが下げる回数は現在最大一日9回
  - 翻訳ブログなどで改めて確認を。
    - https://aws.amazon.com/jp/blogs/news/new-auto-scaling-for-amazon-dynamodb/

DynamoDB Accelerator (DAX)

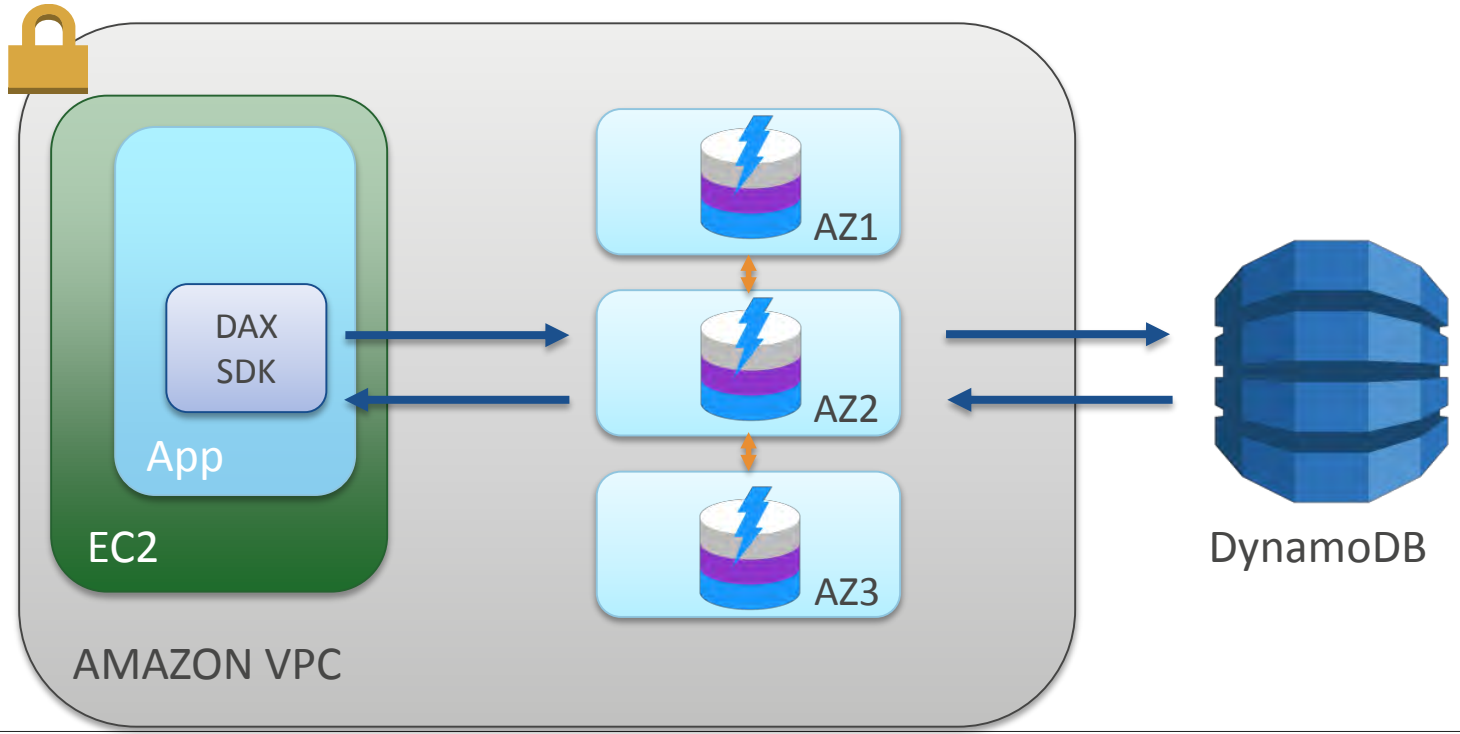
DynamoDB Accelerator (DAX)



機能

- **フルマネージドかつ高可用性:** リージョン内でマルチAZ構成かつキャッシュ情報のレプリケーション、障害時のフェイルオーバーなどをフルマネージドで実現
- **DynamoDB API互換:** 現在のSDKと互換性を保っているのでコードの大部分は書き直す必要が無い
- **Write-through:** ライトスルーでキャッシュを保持
- **Flexible**: 様々なDynamoDBのテーブル状況に対応
- **Scalable**: 最大10ノードまでのスケールアウトに対応
- **Manageability:** Amazon CloudWatch, Tagging for DynamoDB, AWS Consoleなどとの連携も完備
- **Security:** Amazon VPC, AWS IAM, AWS CloudTrail, AWS Organizationsに対応

High Availability



DynamoDB API compatible

Q: How easy is it to add in-memory caching with DAX?

A: Comment out the **this code** and **add this code**

```
public class myApp {

  public static void main(String[] args) throws Exception {

    AmazonDynamoDBClient client = new AmazonDynamoDBClient();

    //** DAX Specific **
    String daxEndpoint = "demo1.zakkqx.clustercfg.dax.use1.cache.amazonaws.com:8111";
    ClientConfig daxConfig = new ClientConfig().withCredentialsProvider(new ProfileCredentialsProvider()).withEndpoints(daxEndpoint);
    AmazonDaxClient client = new ClusterDaxClient(daxConfig);
    //** DAX Specific **

    myTests tests = new myTests();
    tests.setup();

    DynamoDB dynamoDB = new DynamoDB(client);
    Table table = dynamoDB.getTable("Movies");

    tests.getItemTest(table, 10, tests.yearArray, tests.titleArray);

  }
```

1,1 Top

DAX is API compatible with DynamoDB

- **Read APIs:** GetItem, BatchGetItem, Query, Scan
- **Modify APIs:** PutItem, UpdateItem, DeleteItem, BatchWriteItem
- **Control plane APIs:** サポート外(CreateTable, DeleteTable, etc.)

DAX has two caches

Item Cache {GetItem, PutItem}

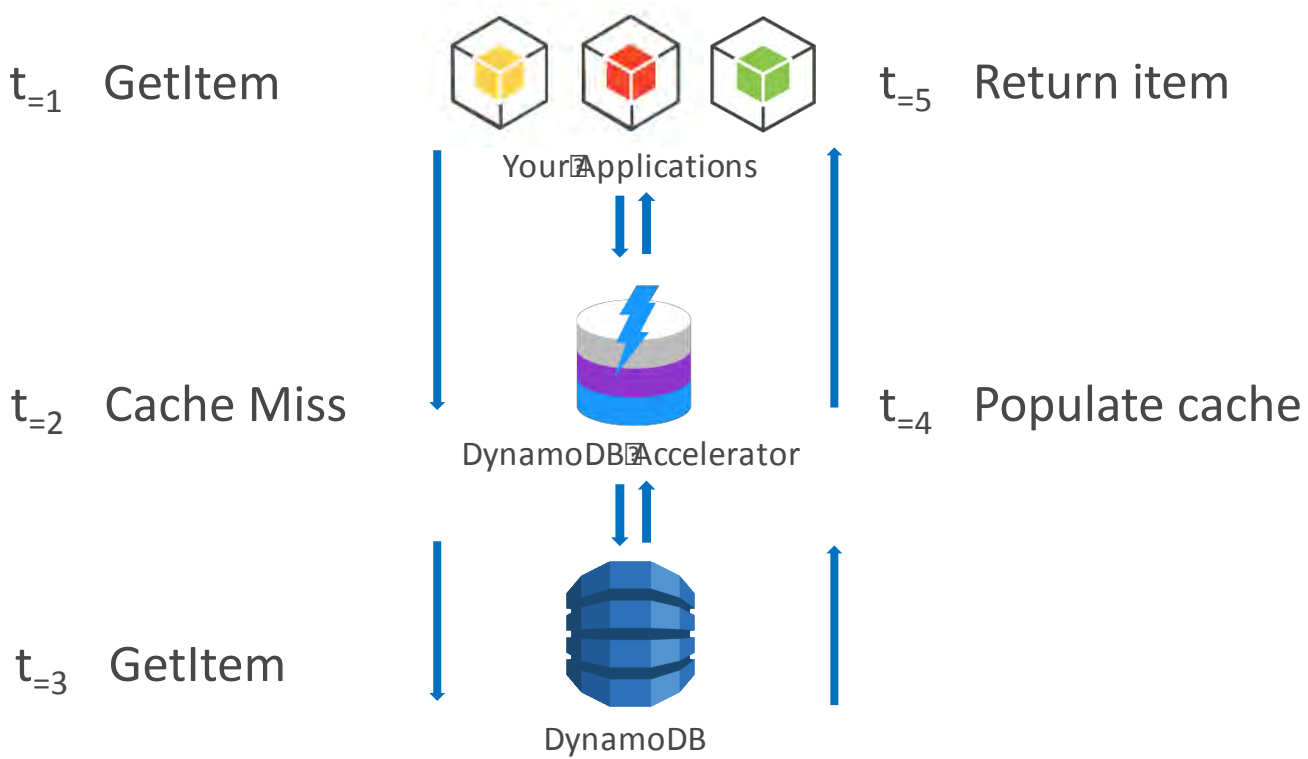
key, value

Query Cache {query, scan}

query text, result set

Write-through

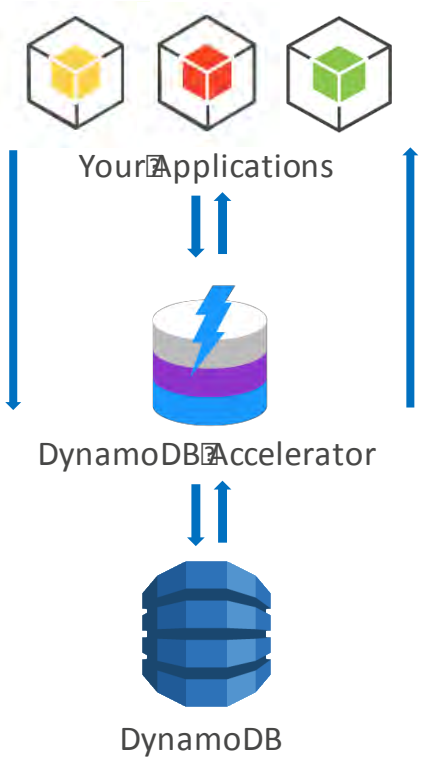
DynamoDB Accelerator (DAX): Reads



DynamoDB Accelerator (DAX): Reads

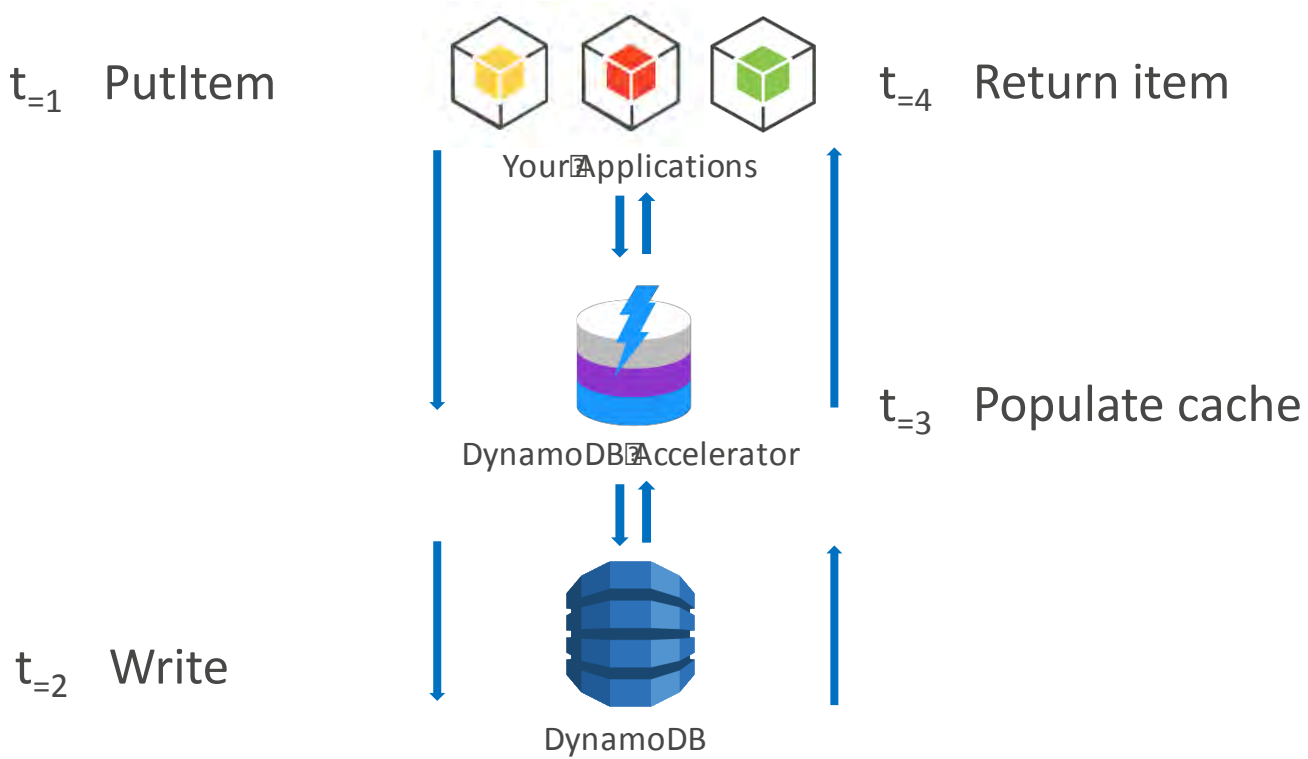
$t_{=1}$ GetItem

$t_{=2}$ Cache Hit



$t_{=3}$ Return item

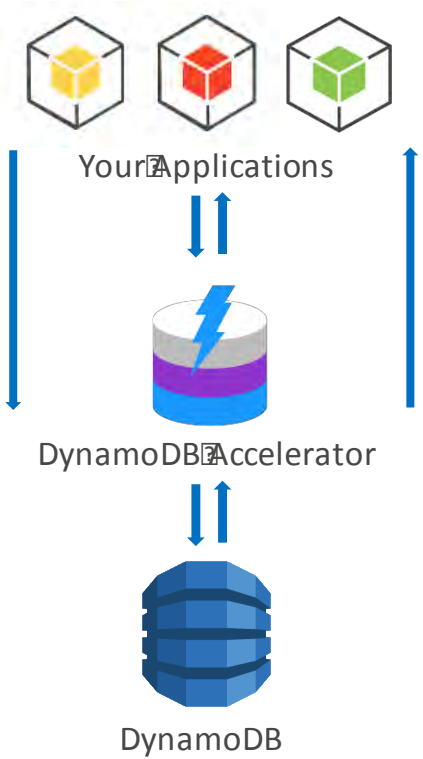
DynamoDB Accelerator (DAX): Write-through



DynamoDB Accelerator (DAX): Read after Write

$t_{=1}$ GetItem

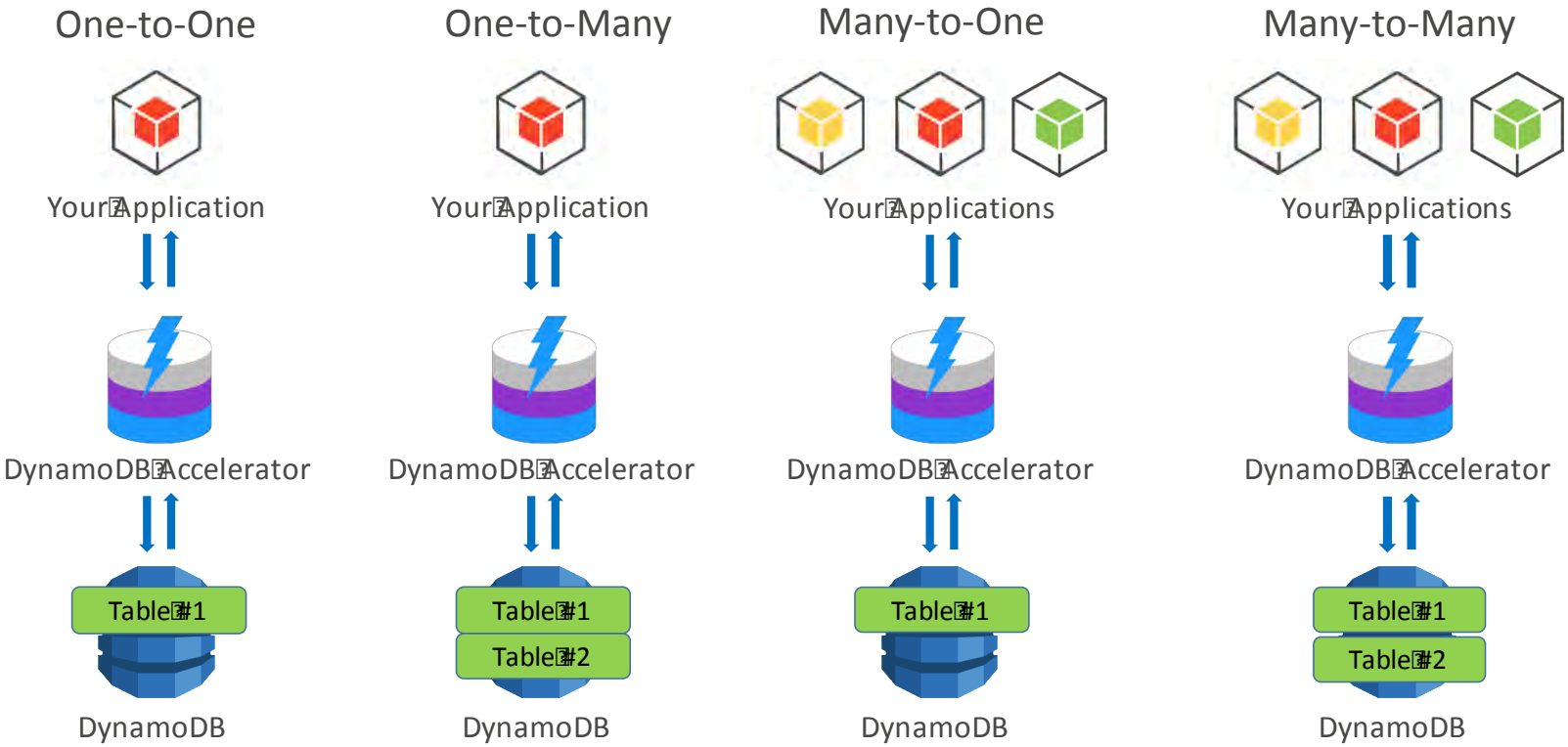
$t_{=2}$ Cache Hit



$t_{=3}$ Return item

Flexible

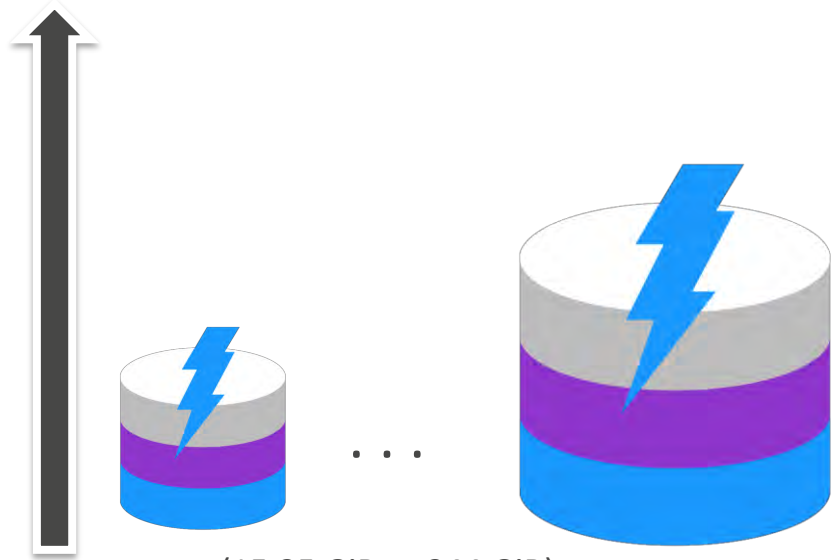
DynamoDB Accelerator (DAX): Flexible



Scalable

Scaling DAX

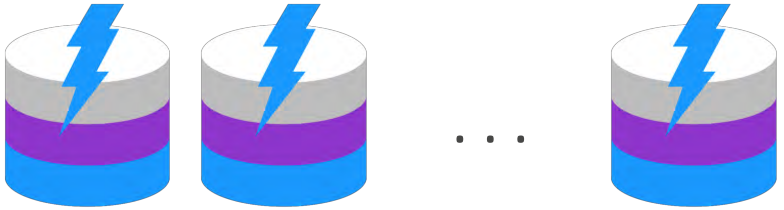
Scale-up



(15.25 GiB to 244 GiB)

Scale Up

Scale-out



(up to 10 replicas)

Scale Out

Manageability

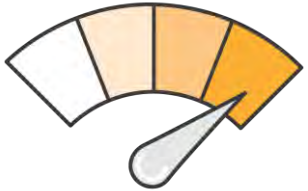
AWS Management Console

Amazon CloudWatch

Cost-based Tagging

Cache Eviction

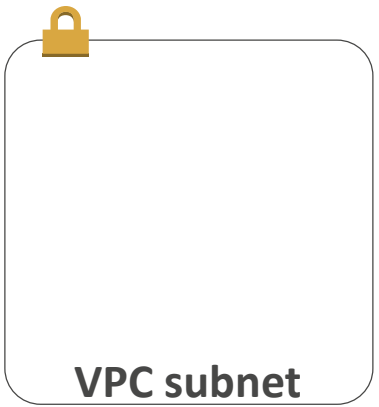
Time-to-live (TTL)



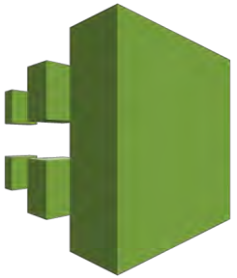
Least Recently Used (LRU)

Write-through eviction

Security



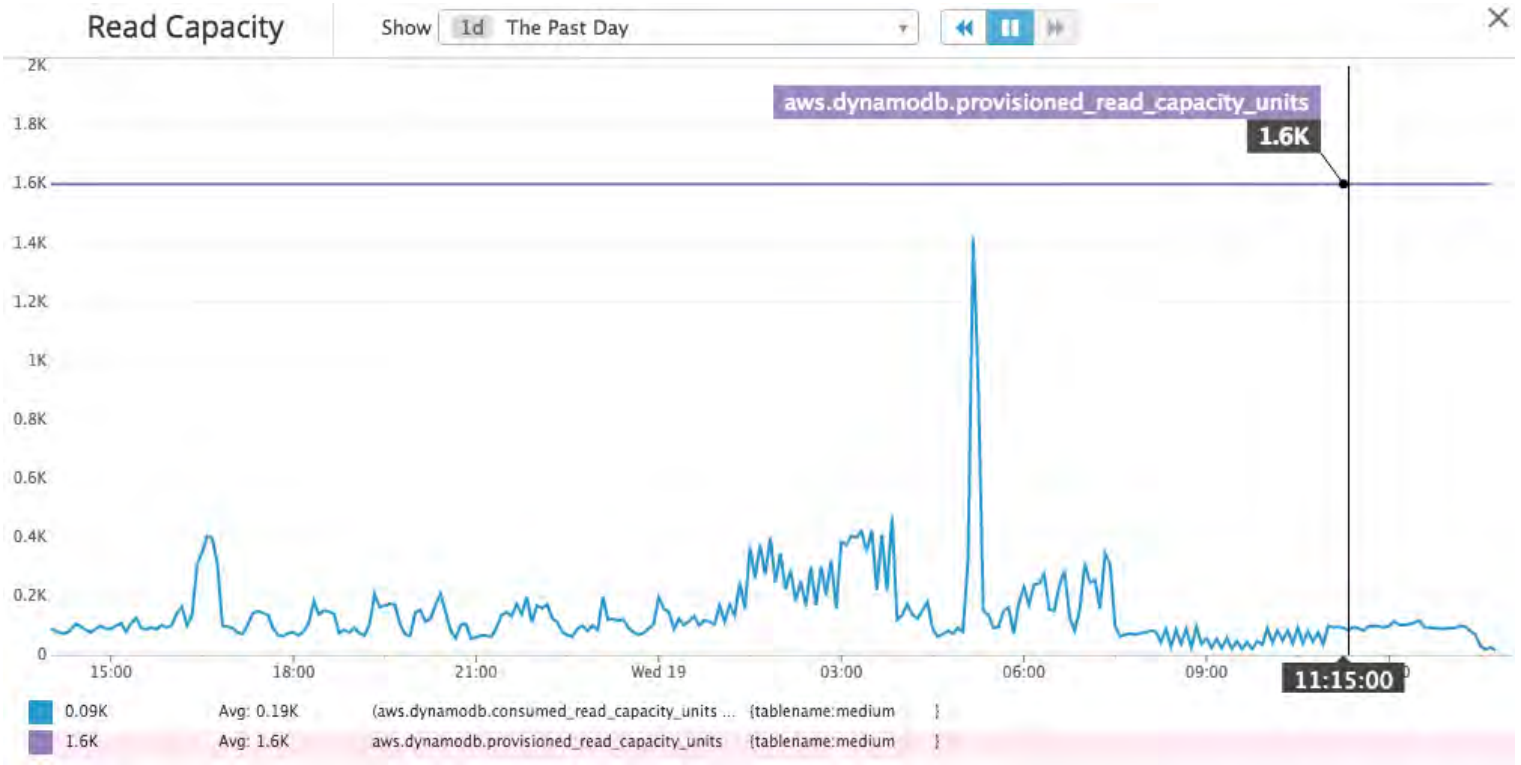
IAM



AWS
CloudTrail

想定されるシナリオ

Use Case #1: Unpredictable spikes



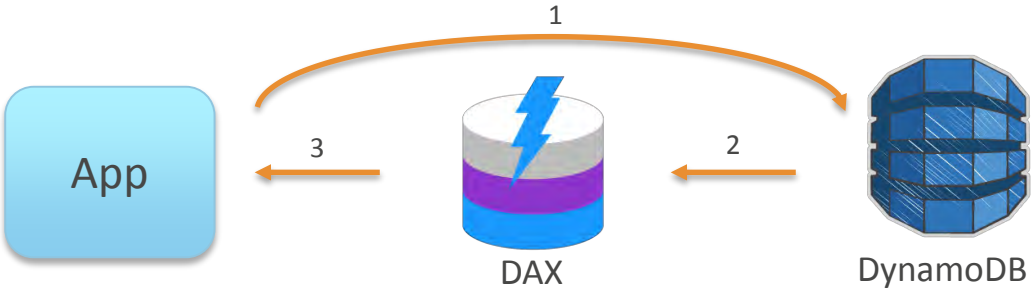
Customer Use Case #2: Speed

Response times in microseconds

Use Cases

Write-Through Cache

Write-Around Cache



価格

アジアパシフィック（東京）(ap-northeast-1)

dax.r3.large	0.322 USD
dax.r3.xlarge	0.644 USD
dax.r3.2xlarge	1.289 USD
dax.r3.4xlarge	2.577 USD
dax.r3.8xlarge	5.154 USD

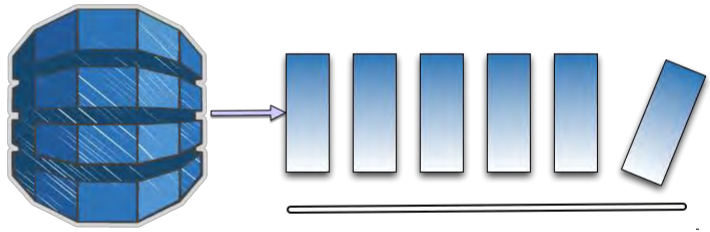
注意事項

- 現在Java SDK only。他言語対応は今後。
- US East (Northern Virginia)、 EU (Ireland)、 US West (Oregon)、 Asia Pacific (Tokyo)、 US West (Northern California) の5つの地域で利用可能
- Instances: r3のみ

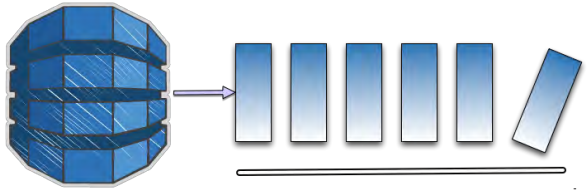
Agenda

- DynamoDBとは
- テーブル設計
- DynamoDB Streams
- AWS Mobile SDKと 2-Tier アーキテクチャ
- 運用関連
- ツールとエコシステム
- まとめ

DynamoDB Streams?

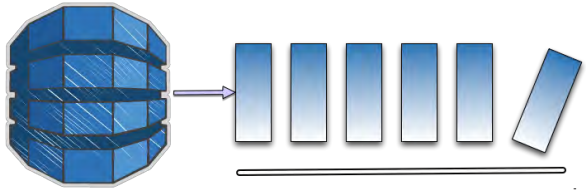


What is DynamoDB Streams?



- DynamoDBに行われた追加、更新、削除の変更履歴を保持しとりだし可能
- 過去 24 時間以内にそのテーブルのデータに対して行われた変更のストリームすべてにアクセス可能。24時間経過したストリームデータは、その後、消去される
- DynamoDB Streams の容量は自動的に管理

DynamoDB Streamsの順番保証



- 操作が行われた順番に沿ってデータは、シリアライズされる。
- 特定のハッシュキーに対して行われた変更は、正しい順序で取得可能
- ハッシュキーが異なる場合は、受信した順序とは異なる順序でDynamoDB Streamsに格納されることがある

  例）ゲームのハイスコア保存しているテーブル

  テーブルの各項目が個別のプレーヤーのポイントを下の順番に実行

  ① プレーヤー 1 ：100 ポイントに変更
  ② プレーヤー 2 ：50 ポイントに変更
  ③ プレーヤー 1 ：125 ポイントに変更

  ①と③は同じ項目（プレーヤー 1）に対する変更なので、①より後に③が表示される。これにより、各プレーヤーの最新のハイスコアを取得できる。

  3 つの更新すべてが正しい順序で表示されるとは限らないが（②が①より後で ③より前に表示されるなど）、各個別プレーヤーのレコードに対する更新は正しい順序になる。

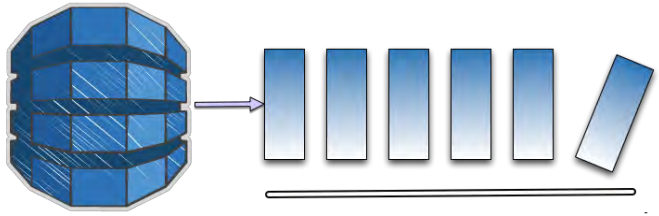
DynamoDB Streamsからの読み取り

- DynamoDB SDK、CLIやKCL(Kinesis Client Library)を用いて読み取り可能
- DynamoDB テーブルのWriteプロビジョニングスループットの最大 2 倍の速度で、DynamoDB Streams から更新を読み取ることが可能

  例)1 秒間に 1,000 項目を更新するのに十分な能力を DynamoDB テーブルにプロビジョニングしている場合、1 秒間に最大 2,000 件の更新を DynamoDB Streams から読み取ることができる。
- DynamoDBへの変更は1 秒未満で反映される

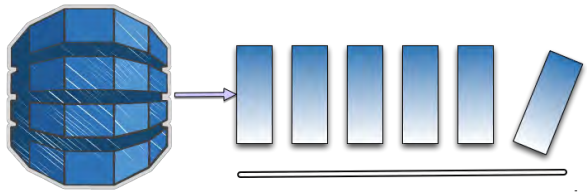
DynamoDB Streamsの使いドコロ

- **クロスリージョンレプリケーション**
- ゲームやソーシャルサイト等のユーザの集計、分析、解析のための非同期集計
- ユーザーが新しい写真をアップロードするとすぐにサークル内のすべての友人のモバイルデバイスに自動的に通知するモバイルアプリケーションの構築等



View types

更新情報（Name = John, Destination = Mars）
⇒(Name = John, Destination = Pluto)



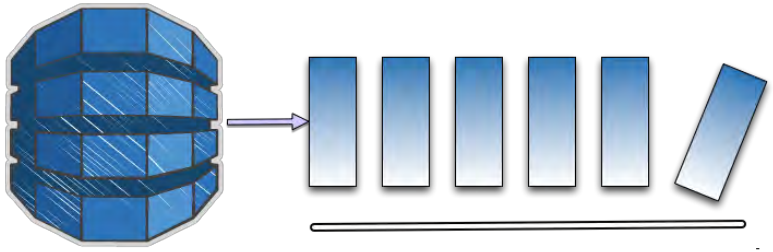
View Type	Destination
古いイメージ – 更新前の情報	Name = John, Destination = Mars
新しいイメージ – 更新後の情報	Name = John, Destination = Pluto
新旧イメージ	Name = John, Destination = Mars Name = John, Destination = Pluto
キーのみ	Name = John

DynamoDB Streams API

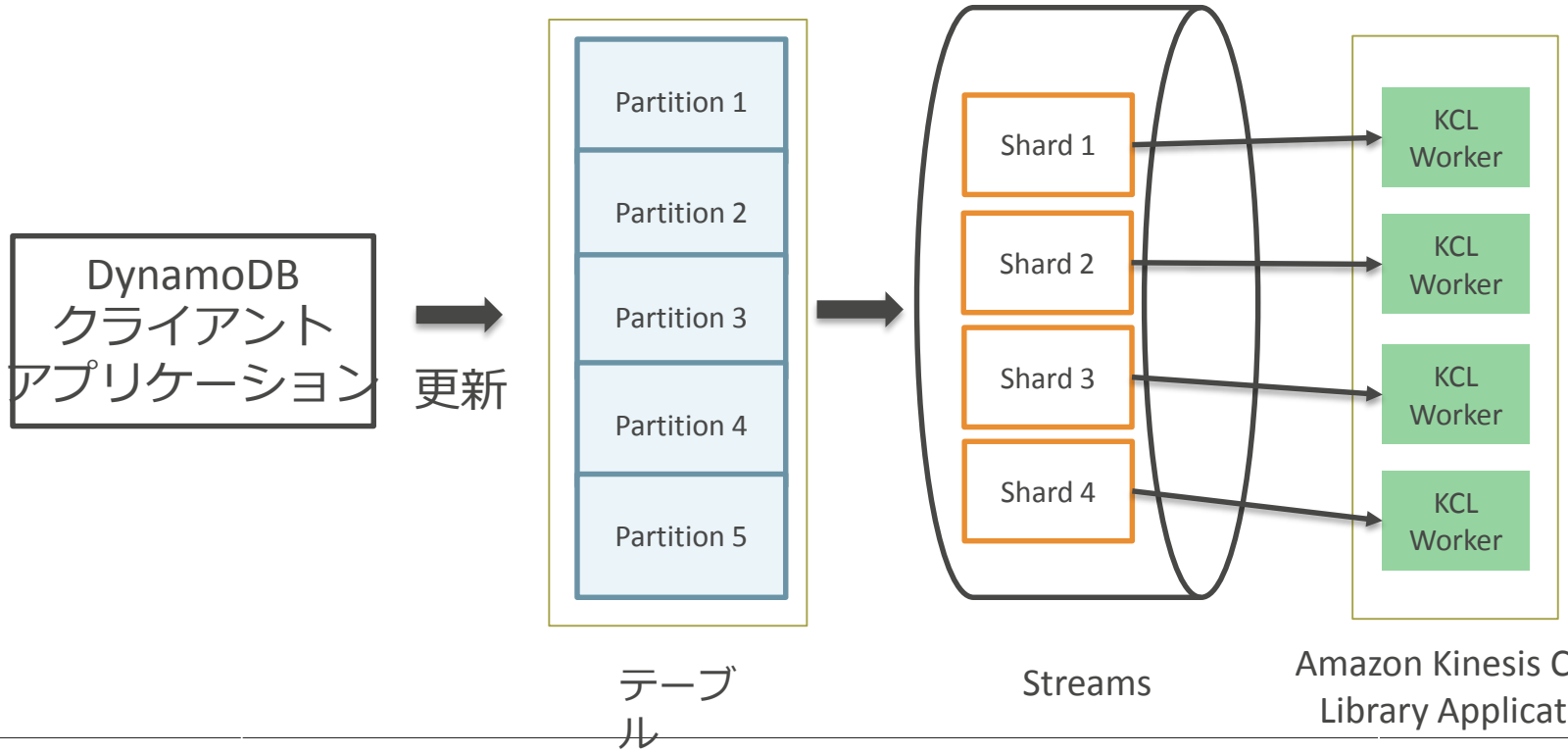
API名	
ListStreams	現在のアカウントのエンドポイントのストリーム記述子（ストリームARN）のリストを返す。
DescribeStream	指定されたストリームの詳細情報を返す。 ストリームの現在の状態、そのアマゾンリソース名（ARN）、そのシャードの構成、およびそれに対応するDynamoDBのテーブルを含むストリームに関する情報を返す
GetShardIterator	シャードイテレータの位置を返す。シャードのどこからデータを読み始めたいかを指定するために使用する。イテレータで最も古いポイントや最新のポイント、またはストリーム内の特定のポイントを指定する。
GetRecords	指定されたシャード内からのストリーム·レコードを返す。GetShardIteratorから取得したシャードイテレータを指定する。 1回で最大 1 MBまたは1000件のストリームレコードを取得

DynamoDB Streams 利用料金（2017/8/9@東京リージョン）

- Streamsの機能を有効化するのは無料
- 毎月最初のReadリクエスト 250万件は無料
- その後は、0.0228 USD/10万

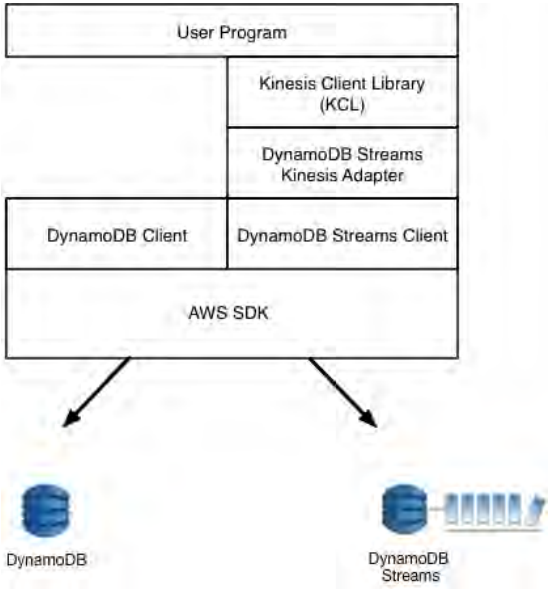


DynamoDB Streams and Amazon Kinesis Client Library

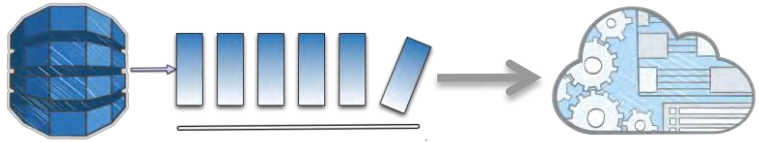


KCL(Kinesis Client Library)

- Kinesis API を使い慣れている開発者は、DynamoDB Streams を簡単に利用可能
- Amazon Kinesis インターフェイスを実装する DynamoDB Streams Adapter を使用すると、アプリケーションで Amazon Kinesis クライアントライブラリ（KCL）を使用して DynamoDB Streams にアクセス可能

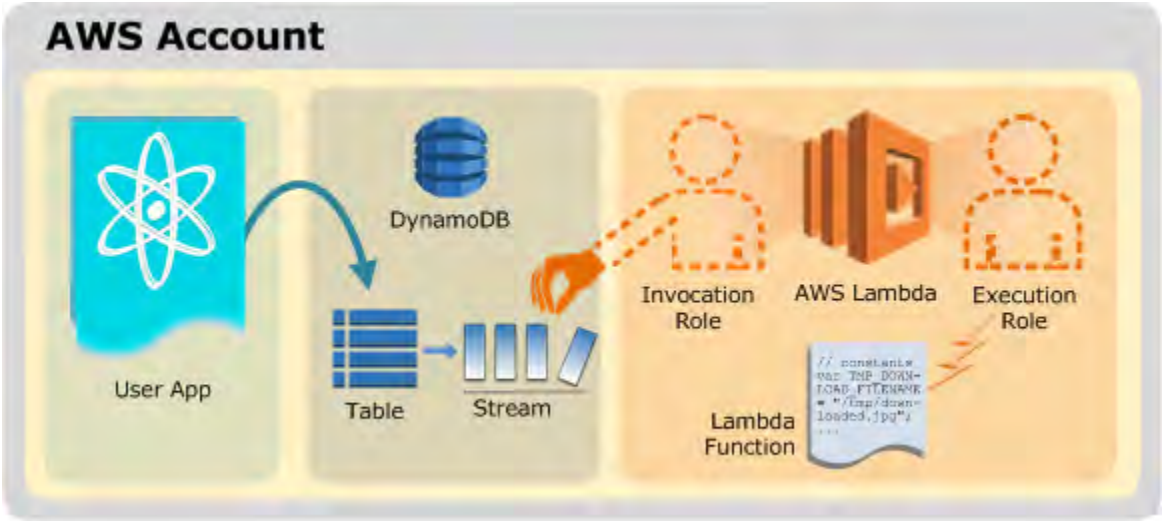


DynamoDB Triggers



DynamoDB + AWS Lambda= DynamoDB Triggers

- AWS Lambda
  - OS、キャパシティ等インフラの管理不要
  - S3、Kinesis、SNS等でのイベント発生を元にユーザが用意したコード（Node.js、Java）を実行
  - ユーザアプリからの同期/非同期呼び出し

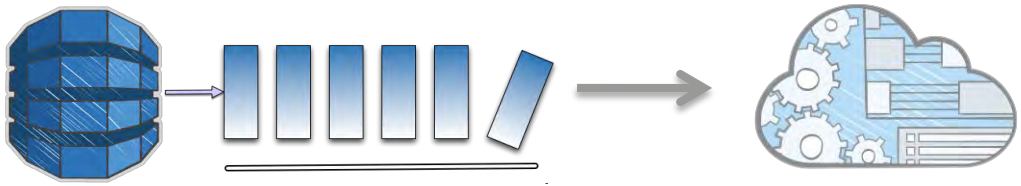


DynamoDB Triggers ユースケース

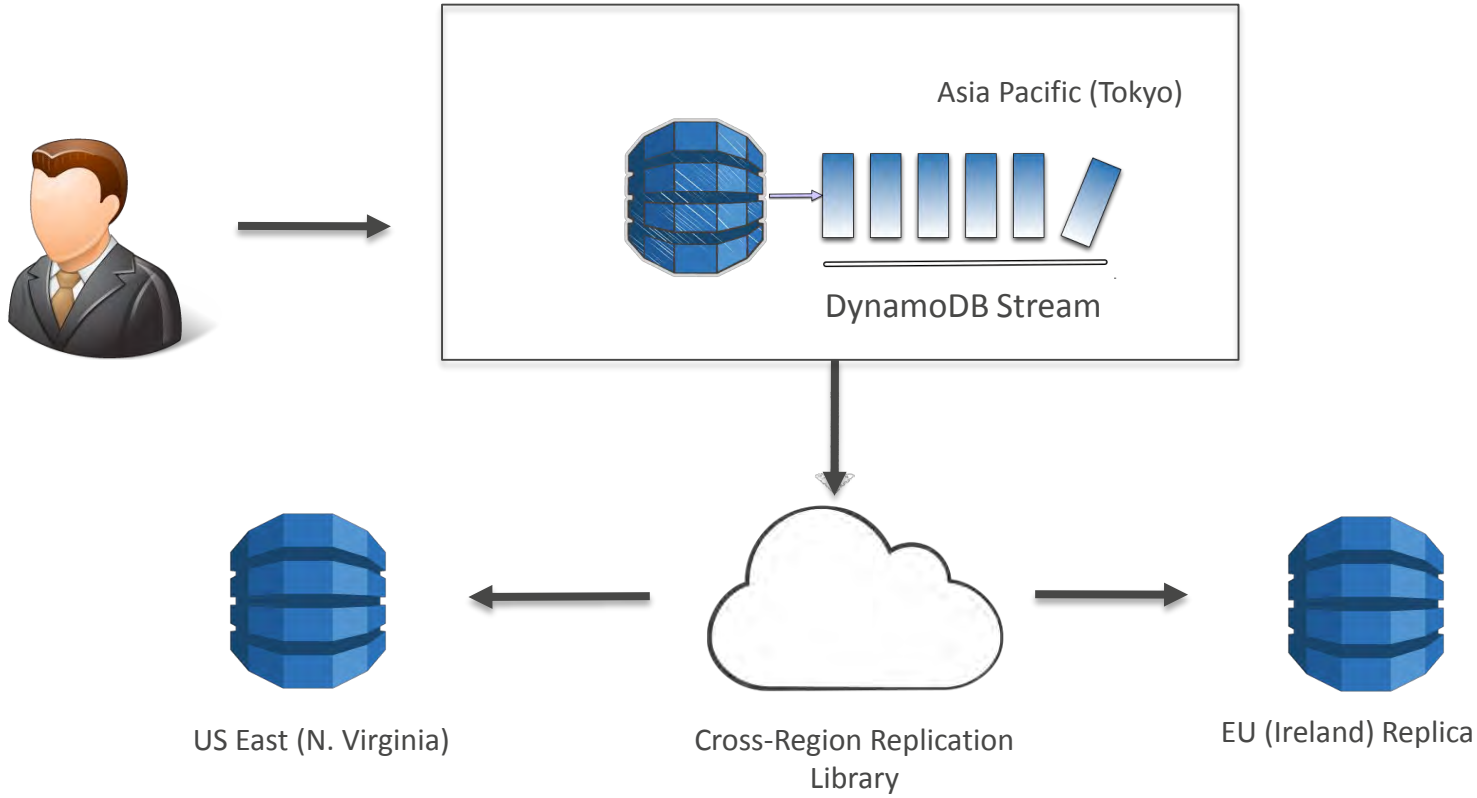
- DynamoDBへの書き込みに応じて値チェックをしつつ別テーブルの更新やプッシュ通知を実行
- DynamoDBの更新状況の監査ログをS3に保存
- ゲームデータなどのランキング集計を非同期に実施

DynamoDB Triggers 利用料金

- AWS Lambda 関数に対するリクエスト回数と AWS Lambda 関数の実行時間の分のみの支払い
- AWS Lambda 関数が、テーブルに関連付けられたストリームに対し行う読み込みについては課金されない



DynamoDB Cross-region Replication



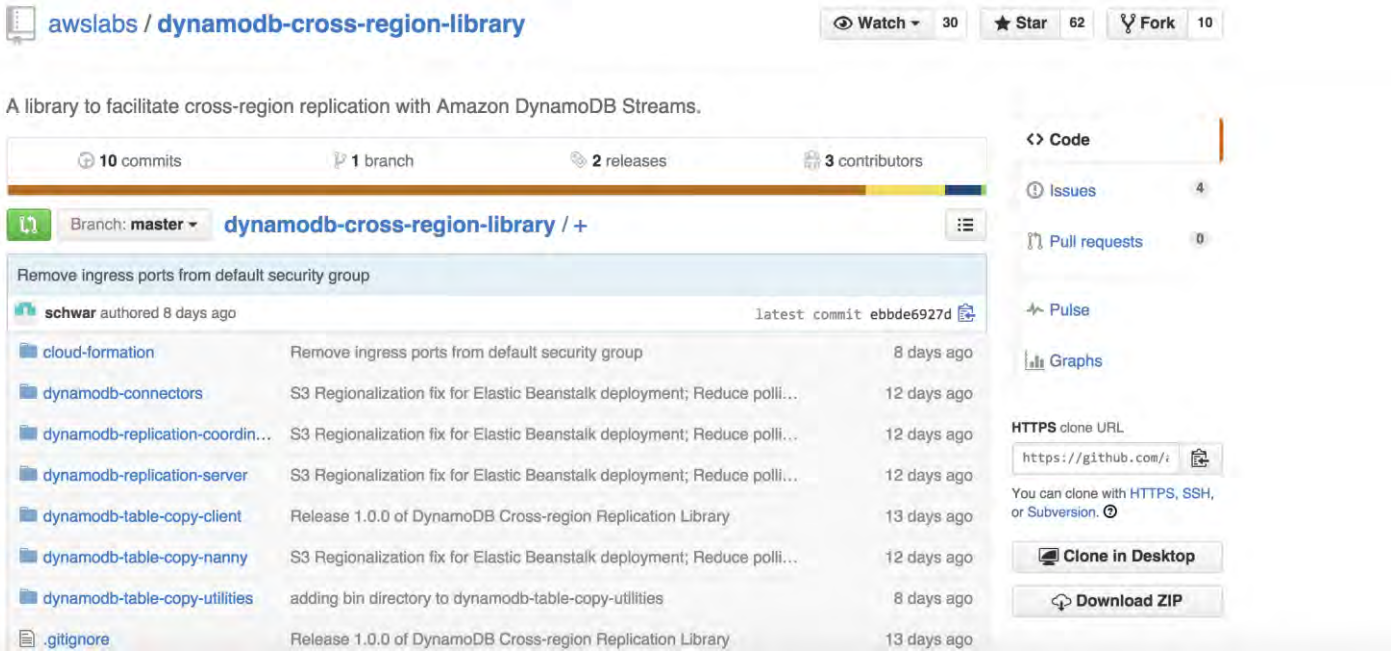
DynamoDB Cross-region Replicationユースケース

- DRとして利用：複数のリージョンへのテーブルを複製することで、データセンターの障害が発生した場合に、他のリージョンへ切り替えることが可能となる
- 速いリード：複数のリージョンにサイトを運営している場合、最も近いAWSのリージョンからDynamoDBのテーブルを読み取ることで、より高速なデータを提供することが可能となる
- RR：テーブル全体の読み取り作業負荷を分散するためにレプリカを使用することにより、マスター表のリードキャパシティを節約することができる
- リージョンマイグレーション：新しいリージョンにリードレプリカを作成し、マスターデータをレプリカにレプリケーションし、簡単にそのリージョンにシステム移行を行う

DynamoDB Cross-region Library

オープンソースとなっており、独自で組み込み可能

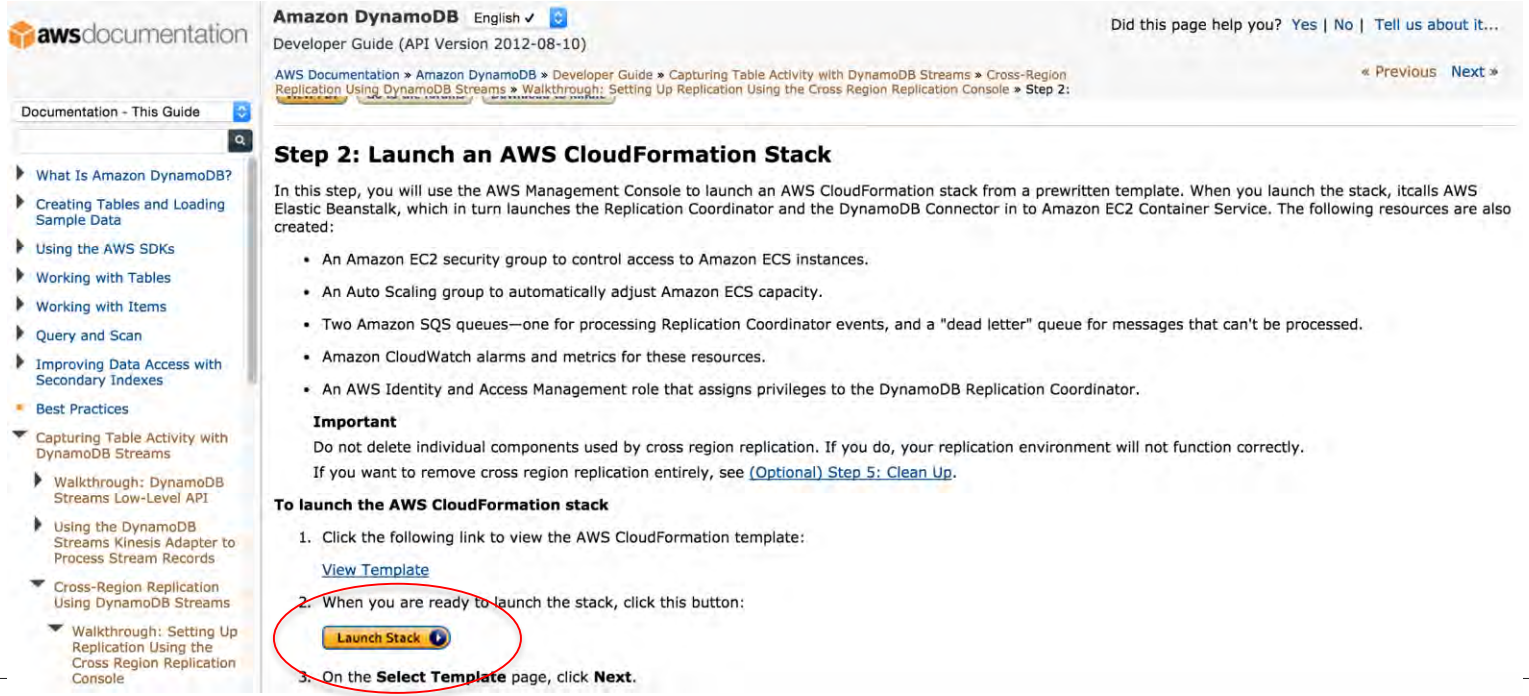
https://github.com/awslabs/dynamodb-cross-region-library



DynamoDB Cross-region Replication

DynamoDB Cross-region Libraryを用いて、CloudFormation化

DynamoDBのドキュメント
http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.CrossRegionRepl.Walkthrough.Step2.html



awsdocumentation

Amazon DynamoDB English ✓
Developer Guide (API Version 2012-08-10)

Did this page help you? Yes | No | Tell us about it...

AWS Documentation » Amazon DynamoDB » Developer Guide » Capturing Table Activity with DynamoDB Streams » Cross-Region Replication Using DynamoDB Streams » Walkthrough: Setting Up Replication Using the Cross Region Replication Console » Step 2:

« Previous Next »

Documentation - This Guide

- What Is Amazon DynamoDB?
- Creating Tables and Loading Sample Data
- Using the AWS SDKs
- Working with Tables
- Working with Items
- Query and Scan
- Improving Data Access with Secondary Indexes
- Best Practices
- Capturing Table Activity with DynamoDB Streams
  - Walkthrough: DynamoDB Streams Low-Level API
  - Using the DynamoDB Streams Kinesis Adapter to Process Stream Records
- Cross-Region Replication Using DynamoDB Streams
  - Walkthrough: Setting Up Replication Using the Cross Region Replication Console

Step 2: Launch an AWS CloudFormation Stack

In this step, you will use the AWS Management Console to launch an AWS CloudFormation stack from a prewritten template. When you launch the stack, itcalls AWS Elastic Beanstalk, which in turn launches the Replication Coordinator and the DynamoDB Connector in to Amazon EC2 Container Service. The following resources are also created:

- An Amazon EC2 security group to control access to Amazon ECS instances.
- An Auto Scaling group to automatically adjust Amazon ECS capacity.
- Two Amazon SQS queues—one for processing Replication Coordinator events, and a "dead letter" queue for messages that can't be processed.
- Amazon CloudWatch alarms and metrics for these resources.
- An AWS Identity and Access Management role that assigns privileges to the DynamoDB Replication Coordinator.

Important

Do not delete individual components used by cross region replication. If you do, your replication environment will not function correctly.

If you want to remove cross region replication entirely, see (Optional) Step 5: Clean Up.

To launch the AWS CloudFormation stack

1. Click the following link to view the AWS CloudFormation template:

   View Template

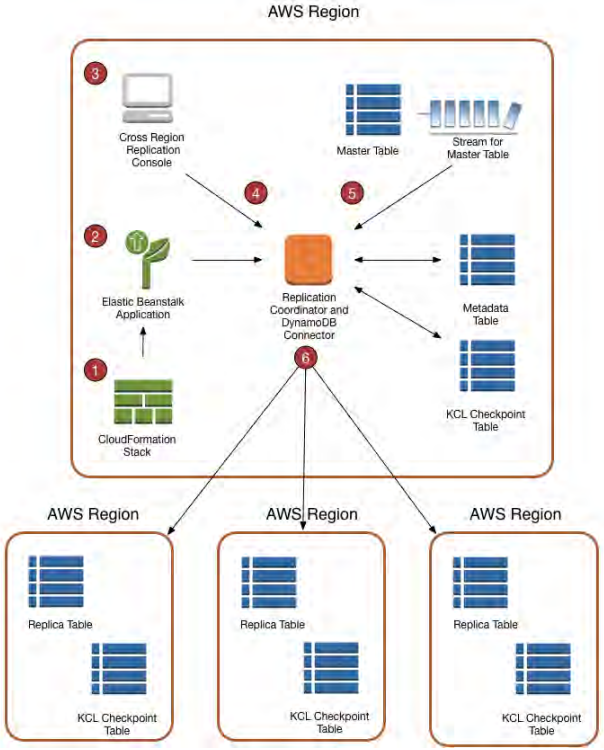
2. When you are ready to launch the stack, click this button:

   Launch Stack

3. On the **Select Template** page, click **Next**.

amazon web services

DynamoDB Cross-region Replication

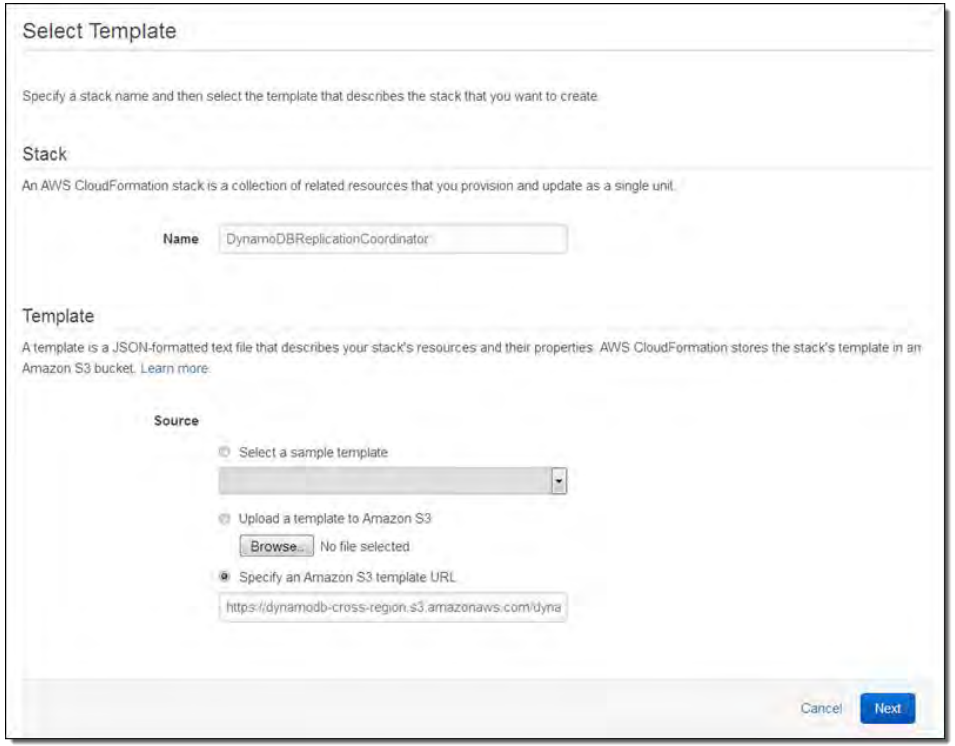


1	事前設定済みのAWS CloudFormationスタックを起動 これが完了するのに約20分かかり、1回限りの操作
2	AWS CloudFormationスタックは、AWS Elastic Beanstalkを使用して Amazon EC2コンテナサービス（アマゾンECS）に レプリケーション·コーディネーターとDynamoDBのコネクタを起動
3	レプリケーショングループを作成するには、クロスリージョンレプリケーションコンソールを利用
4	レプリケーション·コーディネーターは、メタデータ表、その他のリージョンでのレプリカテーブルを含む必要なすべてのリソースを割り当て この操作は、完了するまでに最大30分かかる
5	DynamoDBのコネクタは、DynamoDBの中でマスターテーブル上のストリームからの更新データを読み取り、処理
6	DynamoDBのコネクタは、レプリカ表（複数可）を更新

DynamoDB Cross-region Replication

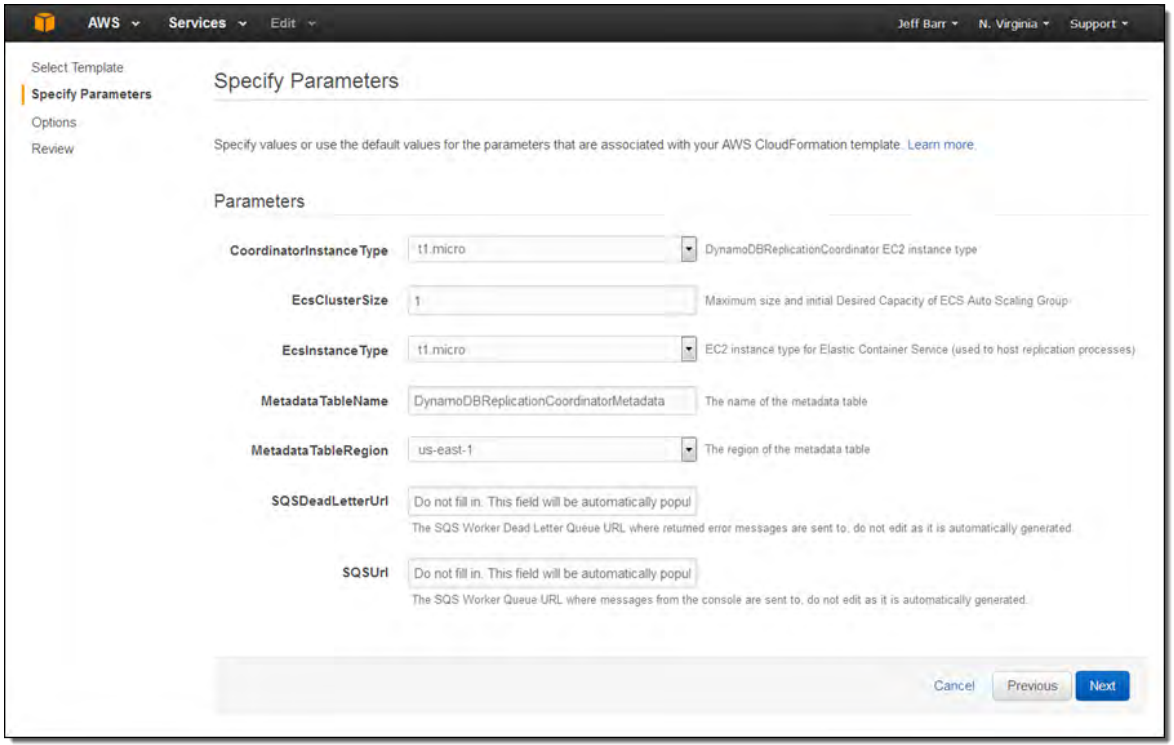
CloudFormationコンソール内から起動プロセスを開始
CloudFormationでは、スタックとコンテナを作成するために必要な情報の入力

スタック（テンプレートによって起動するAWSリソースセットの総称）に名前を付けて、**Next**をクリックして次へ



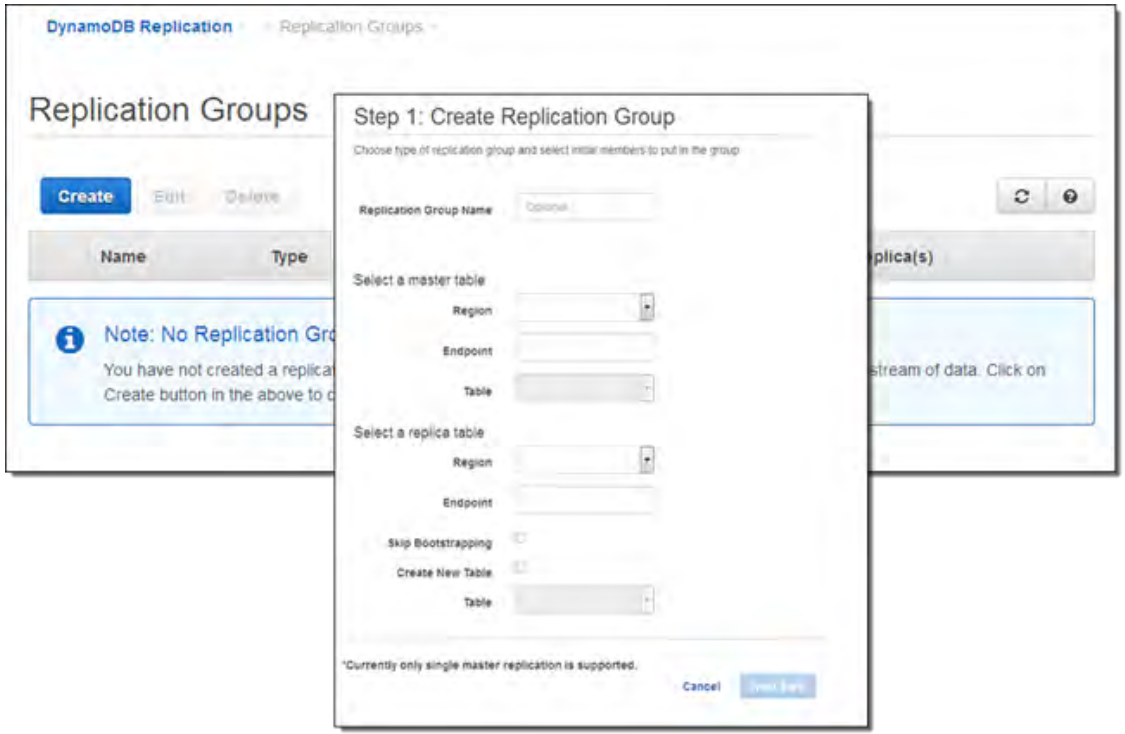
DynamoDB Cross-region Replication

次に、パラメータ等を入力
使用するEC2のインスタンス
タイプ等を入力
（基本的にそのままで大丈夫
なはず）



DynamoDB Cross-region Replication

レプリケーション·アプリを起動した後、EC2上のオンラインのconfig設定ページ（CloudFormationテンプレートからURLが生成される）にアクセスする事が出来る様になり、レプリケーショングループを作成（マスターテーブル、レプリカテーブル設定）し、実行



DynamoDB Cross-region Replication 利用料金

DynamoDB Cross-region Replicationを利用するにあったて下記の料金が発生

- DynamoDBのレプリケーション先のプロビジョニングされたスループット（書き込みと読み込み）およびレプリカテーブルのストレージ料金
- リージョン間データ転送量
- テーブル間の同期を維持するための DynamoDB ストリームからのデータの読み込み
- レプリケーションアプリケーションをホストするためにプロビジョニングされた EC2 インスタンスインスタンスの費用は、選択したインスタンスタイプとインスタンスをホストしているリージョンによって異なる
- アプリケーションからの制御コマンドを登録する SQS キューリクエスト

※詳しい料金は各サービス料金を参照

AWS Mobile SDKと 2-Tier アーキテクチャ

AWS Mobile SDK

クロスプラットフォーム

- Android, Fire OS, iOS, Unity をサポート

共通の認証

- Amazon Cognito により認証機構を迅速に導入可能

ネットワーク状態を自動でハンドリング

- 例： ローカルキャッシュによりオフラインでも利用可能

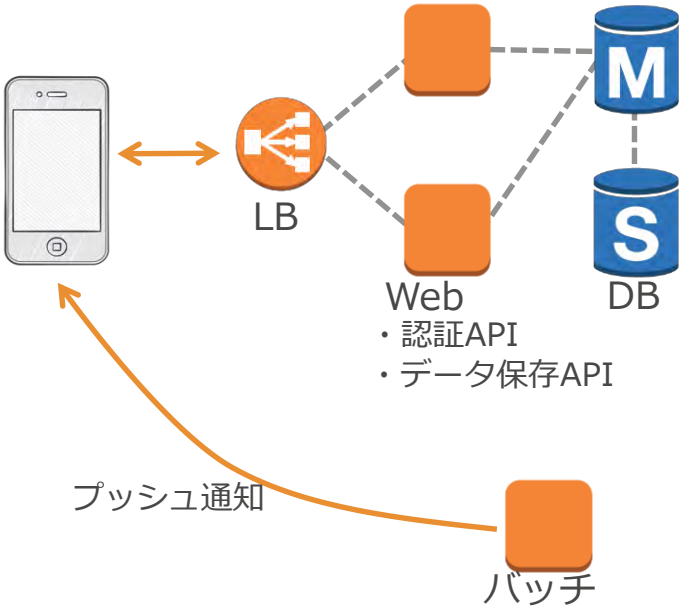
メモリフットプリントの削減

- 導入するパッケージをサービス単位で選択可能

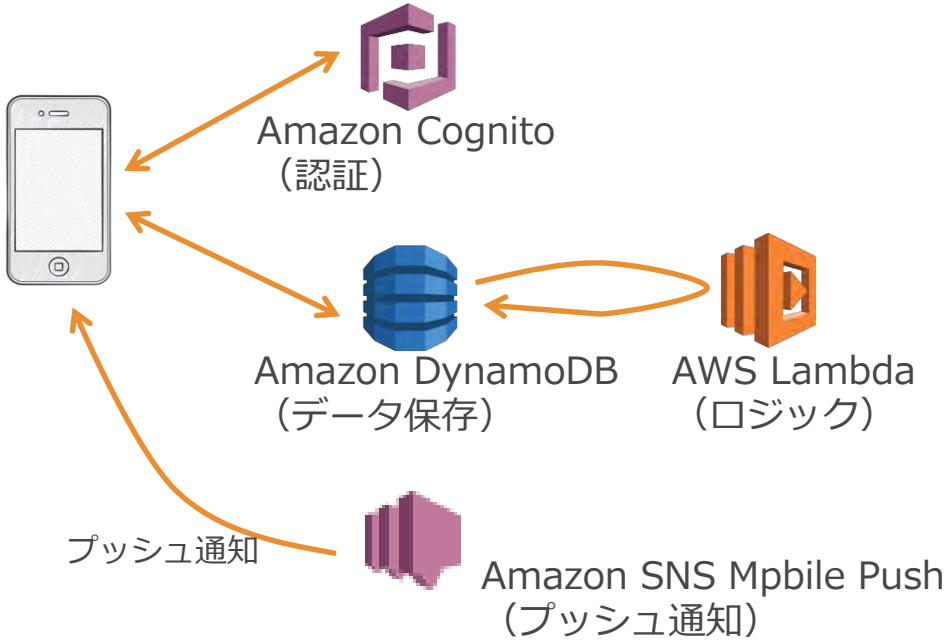
2-Tier アーキテクチャとは？

仮想サーバー(EC2)を利用せずに、クライアントから直接 AWS のサービスを利用する。

従来 アーキテクチャ



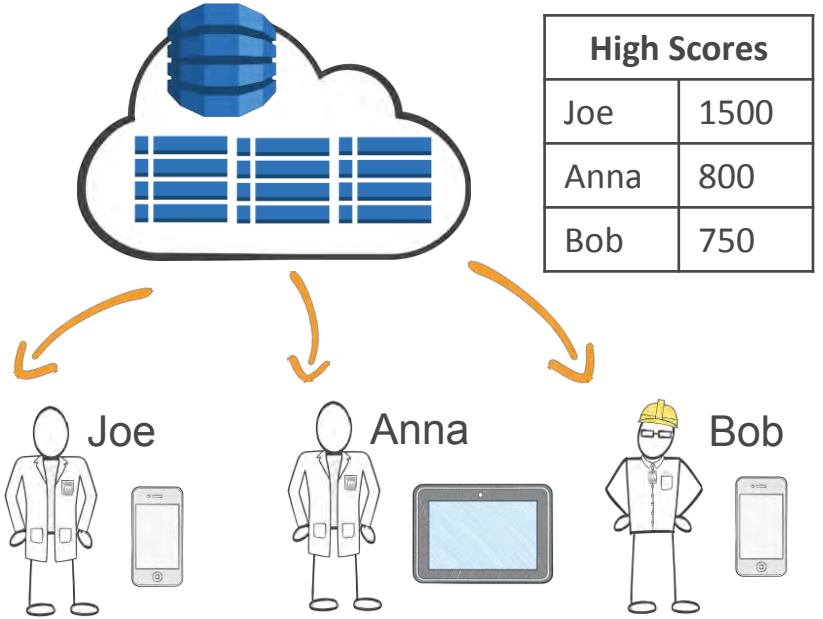
2-Tier アーキテクチャ



Amazon DynamoDB Connector

Object Mapper

- クライアント側のクラスが Amazon DynamoDB テーブルにマッピング
- オブジェクトをテーブルに変換することも、その逆も必要ない



Amazon DynamoDB Streams

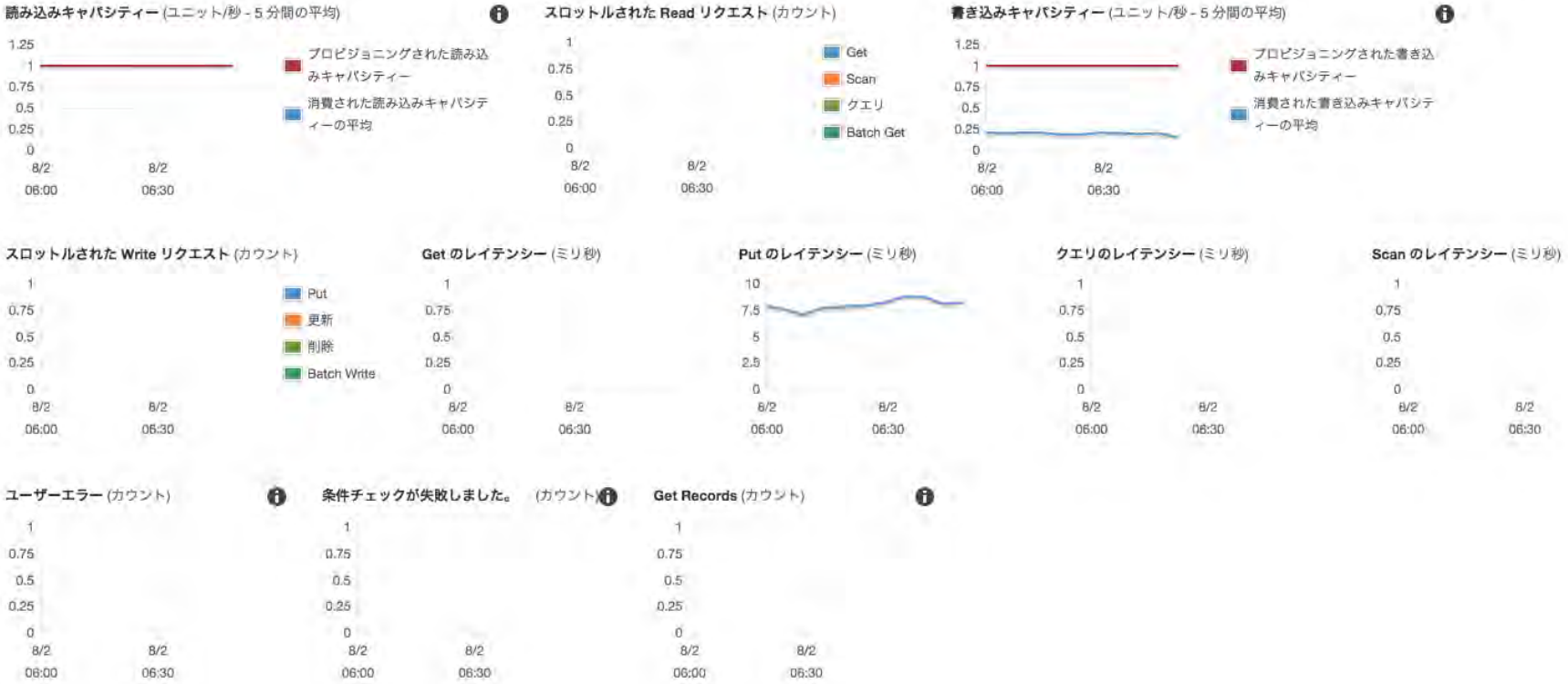
AWS Lambda と連携したり、KCL ワーカーで処理するなど

DynamoDBへの書き込みに応じて値チェックをし別テーブルの更新やプッシュ通知を実行するなど、アイディアはさまざま。

Agenda

- DynamoDBとは
- テーブル設計とサンプル
- DynamoDB Streams
- AWS Mobile SDKと 2-Tier アーキテクチャ
- 運用関連
- ツールとエコシステム
- まとめ

性能監視はCloudWatchで(1/2)



性能監視はCloudWatchで(2/2)

- CloudWatchで監視できる項目
  - 読み込みキャパシティー
    - 消費されているRead Capacity Unit
  - スロットルされたReadリクエスト
    - Capacity Overでエラーを返したReadリクエスト数
  - 書き込みキャパシティー
    - 消費されているWrite Capacity Unit
  - スロットルされたWriteリクエスト
    - Capacity Overでエラーを返したWriteリクエスト数
  - Get のレイテンシー
  - Put のレイテンシー
  - Query のレイテンシー
  - Scan のレイテンシー
  - ユーザエラー
  - 条件チェックが失敗
    - 条件書き込みの失敗数,クライアント側ではConditionalCheckFailedExceptionを受け取っている
  - GetRecord
    - DynamoDB StreamsでのGet Recordsのリクエスト数

データのバックアップ、レプリカ

- DynamoDBの機能ではRDBのように静止点をとってバックアップするという機能は提供されていないが、下記のいくつかの方法でデータのバックアップ、レプリカを保持することができる

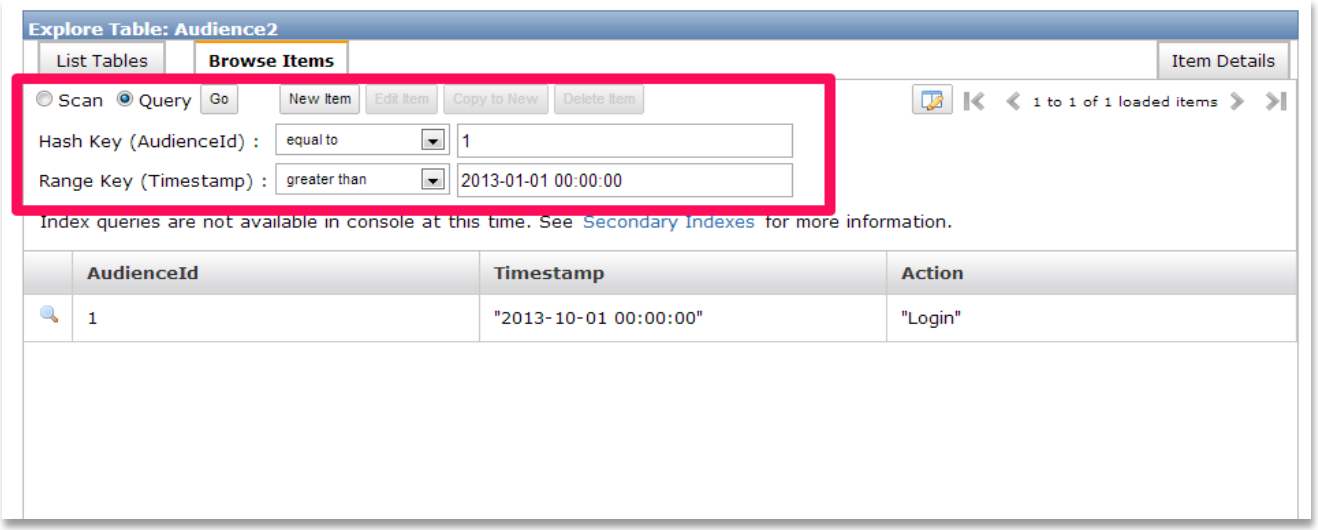
1. DynamoDB Streamsを用いたCross Region Replication
   - 前述したCross-region Repricationを用いて、レプリカを作成、一時的にレプリケーションをStopさせることでPITRバックアップを実現
2. AWS Data Pipelineを使ったデータバックアップ
   - 別のリージョン、同一リージョンのDynamoDBのテーブルに対してコピー、選択したAttributeのみのコピー、選択したAttributeのみのIncrementalコピーのジョブを実行することが可能
3. Amazon Elastic MapReduceを使ったコピー
   - S3や別のDynamoDBに対してAmazon Elastic MapReduceのhiveを使ってデータをコピーすることができる

Agenda

- DynamoDBとは
- テーブル設計
- DynamoDB Streams
- AWS Mobile SDKと 2-Tier アーキテクチャ
- 運用関連
- ツールとエコシステム
- まとめ

データ操作はマネージメントコンソールで

- テーブルに対してSCANやQUERY、PutItemをマネージメントコンソールから実行することができる



DynamoDB Local

- 開発/テスト用のツール
  - 今までは開発やテストをするために実際のDynamoDBのテーブルを作る必要があった。これには**”費用が発生する”、”静的なテスト環境を作れない”、”オフラインで開発できない”**などの問題があった。
  - このツールを利用することにより、開発やCIをより便利に行うことができる
- JARファイルで提供され、ローカルにインストールして動かすことができる(要Java7以上)
- **DynamoDB Streams**も利用可能
- ウェブベース(DynamoDB JavaScript Shell)のUIも利用可能
- あくまでAPIの機能的を再現しているテストツールなので本番では利用しない
- プロビジョンスループットは再現されていないので注意
- 詳細は http://bit.ly/1d9fN5c を参照

Agenda

- DynamoDBとは
- テーブル設計
- DynamoDB Streams
- 運用関連
- ツールとエコシステム
- まとめ

まとめ

- NoSQLはRDBとは全く特性が異なるので、それらを理解した上で適所でうまく活用する！
- 運用はゼロではないが、拡張性の面や性能面ではとても楽ができるのがNoSQLの中でのDynamoDBの特徴
- Mobile SDK やDynamoDB Streamsを利用してより高度で、柔軟なアプリケーションの開発を！

構築・運用にかかる時間を、**本来のビジネス価値を高めることに投資できる！**

Webinar資料の配置場所

- AWS クラウドサービス活用資料集
  - http://aws.amazon.com/jp/aws-jp-introduction/

プロダクト別：				
Amazon S3		AWSマイスターシリーズ Re:Generate Amazon Simple Storage Service (S3)	Slideshare	PDF
Amazon Glacier		AWSマイスターシリーズ Reloaded Amazon Glacier Amazon Glacierのご紹介 機能編	Slideshare（Reloaded） Slideshare（機能編）	PDF（Reloaded） PDF（機能編）
Amazon Route 53		AWSマイスターシリーズ Re:Generate	Slideshare	PDF

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！