1 ノードからはじめる Cloud Spanner の 実践活用方法

グーグル・クラウド・ジャパン合同会社
データマネジメント スペシャリスト
佐藤 貴彦

本日の内容

- データベースに求められる非機能要件
- Cloud Spanner の 3 つの特徴
- Cloud Spanner のユースケース
- 1 ノードからはじめる実践活用方法

データベースに求められる非機能要件

01

システムに求められる様々な非機能要件

可用性	システムの継続、耐障害性、災害対策などの要件
性能と拡張性	通常時及びピーク時のレイテンシやスループット、 リソースの拡張性などの要件
運用と保守性	バックアップ、メンテナンス、障害時運用などの要件
セキュリティ	認証、認可、監査、暗号化などの要件
移行性	システム移行の場合の、移行時期、データ量などの要件

IPA 非機能要件グレード2018 より抜粋

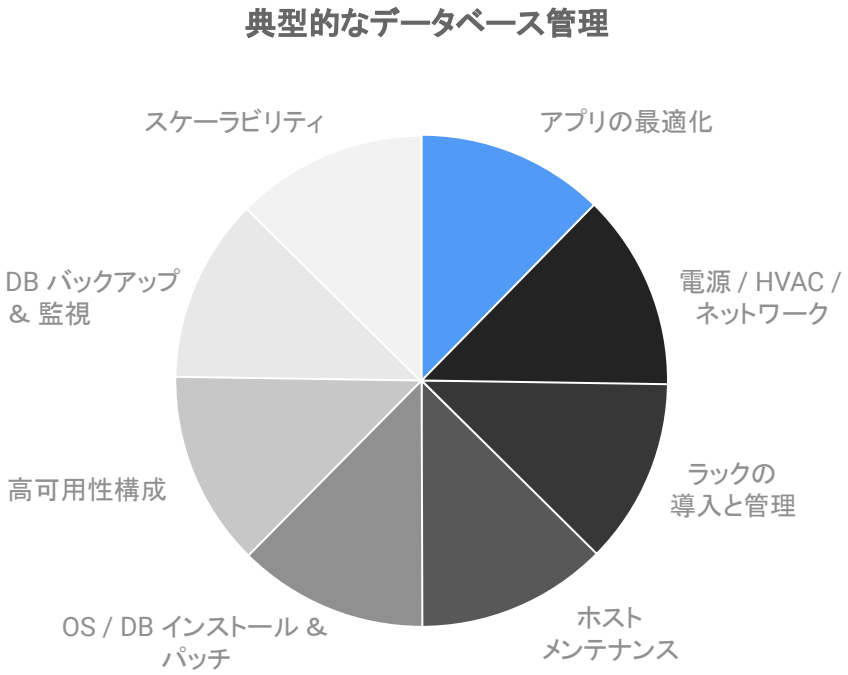
データベースで考える様々な非機能要件

可用性	DB の稼働率、RPO、RTO など
性能と拡張性	DB のスケールアウト、ストレージの拡張など
運用と保守性	DB の運用監視、バックアップ、メンテナンス、障害時運用など
セキュリティ	DB アクセスに対する認証、認可、監査、及びデータの暗号化など

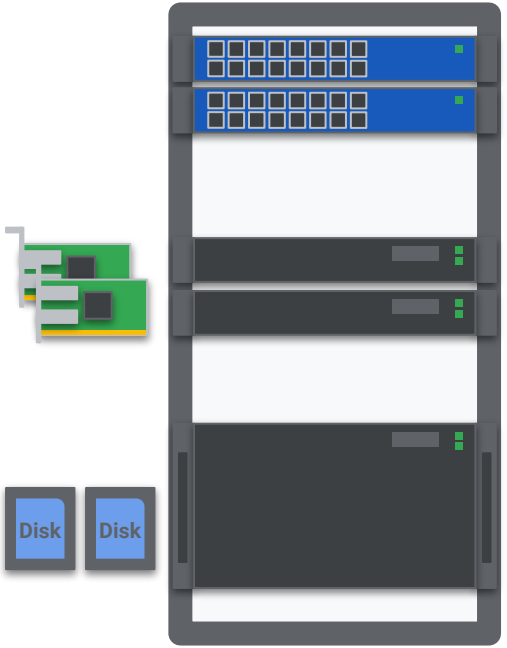
※「移行性」は移行プロジェクト絡みの話なので今回は割愛

非機能要件を満たすために膨れる運用負担

- 非機能要件を満たしていくには、データベース サーバーに様々なものが求められる。
- オンプレミス環境では、各ハードウェアや、DB サーバー上の OS など、様々な運用負担が生じる。
- 特に高可用性の実現するには、上記全てについて冗長構成を構築し適切に運用が必要。



非機能要件を満たすために膨れる運用負担



ネットワークの冗長化

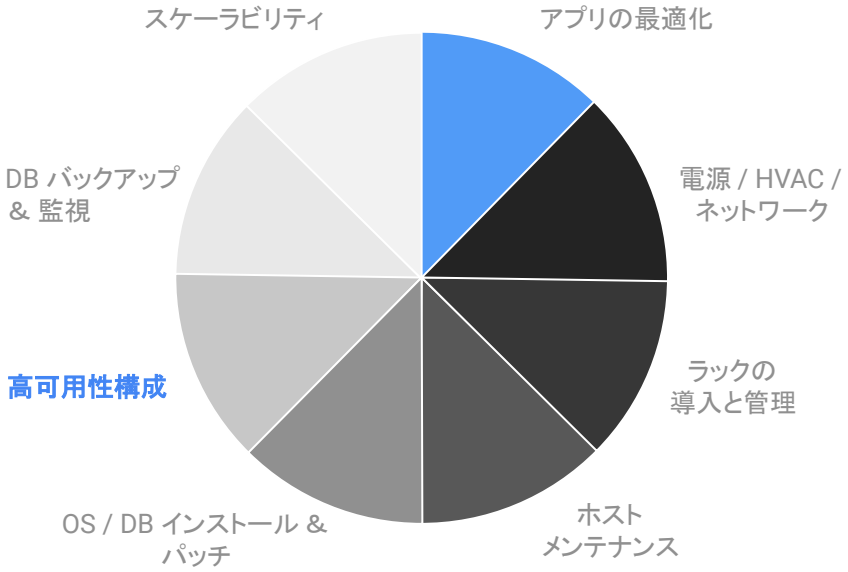
DB サーバーの冗長化

ストレージの冗長化

電源の冗長化

高可用性構成

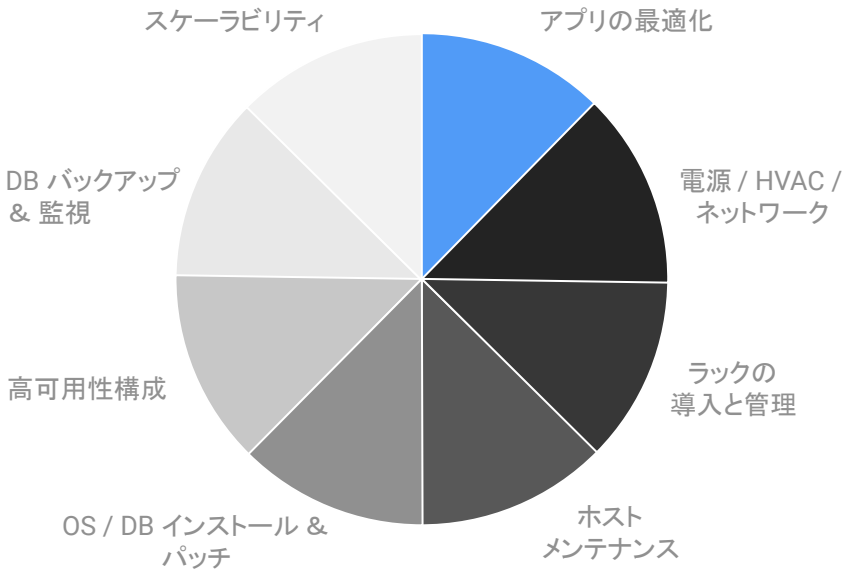
典型的なデータベース管理



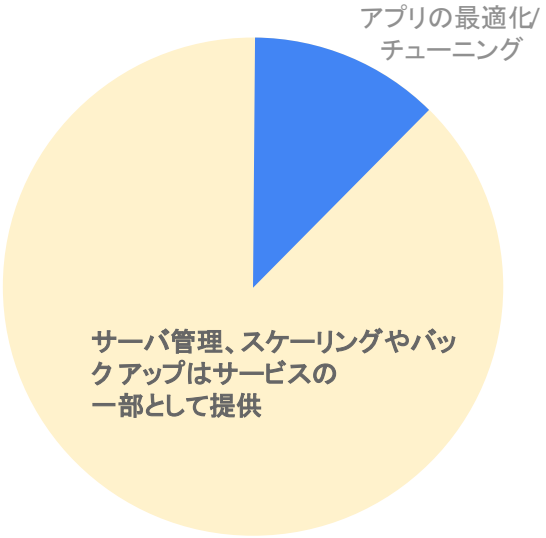
オンプレミス環境の DB サーバー

マネージド データベースによって大幅に低減する負担

典型的なデータベース管理



**DBaaS, Cloud Native DB での
データベース管理**



Google Cloud におけるマネージド データベースの選択肢

キャッシュ

移行に適した
OSS および 商用 DB

モダナイズに適した
クラウドネイティブ DB

データ
ウェアハウス

Cloud Memorystore

マネージド
Redis＆memcached

Cloud SQL

マネージド RDBMS
MySQL &
PostgreSQL &
SQL Server

Cloud Bigtable

低レイテンシで
スケーラブルな
ワイド カラムストア

Cloud Spanner

スケーラブルで
可用性の高い RDBMS

本日のテーマ

Cloud Firestore

サーバーレスで
スケーラブルな
ドキュメントストア

BigQuery

サーバーレスで
スケーラブルな
エンタープライズ
DWH

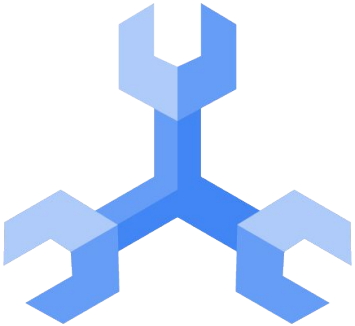
Cloud Spanner

リレーショナル データベースの構造の利点と、

非リレーショナルのスケーラビリティを組み合わせ、

世界中に分散され、強い整合性を持った、

エンタープライズ グレードの**フルマネージド データベース**。



Cloud Spanner の 3 つの特徴

02

クラウド ネイティブなデータベース Cloud Spanner の特徴

特徴 1 - 運用が簡単なフルマネージドRDBMS

フルマネージド データベースで、セキュリティ対応、メンテナンスなども全自動

テーブル構造に対して **SQL** でのクエリや、**ACID トランザクション**をサポート

特徴 2 - 最大 99.999% の高可用性

複数のゾーンやリージョンにまたがって1 つのインスタンスが自動構築され

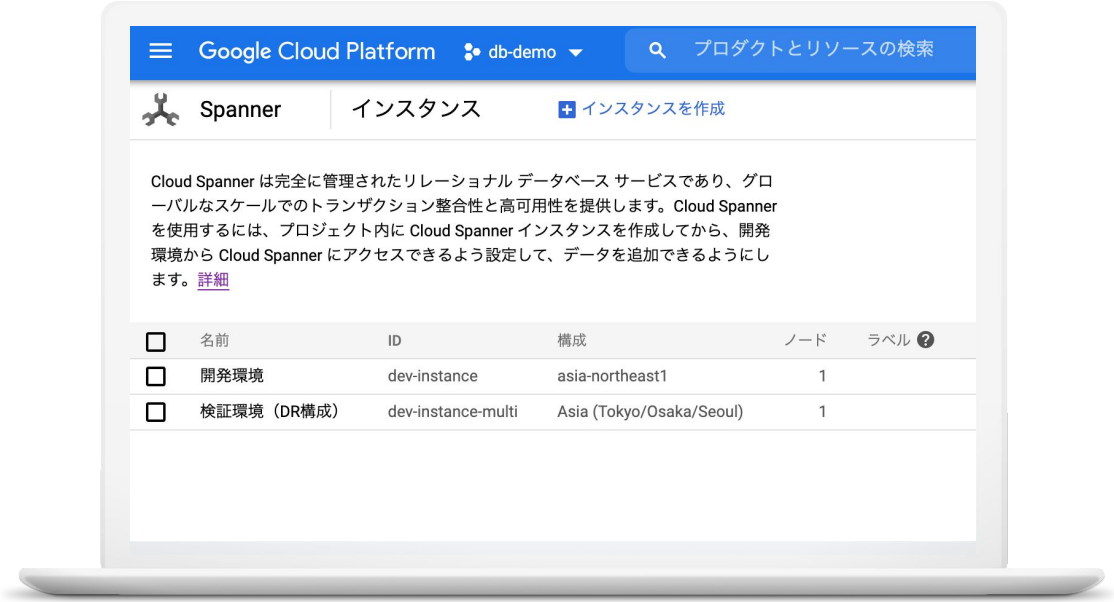
最大 **99.999% の可用性**を提供、メンテナンスやノード追加によるダウンタイムも無し

特徴 3 - 自動シャーディング

ノード数と負荷状況に応じてテーブル内のデータは**自動シャーディング** され、

スケールアウトはもちろん、ノード数を減らしてスケールインも簡単

Cloud Spanner インスタンスの作成画面



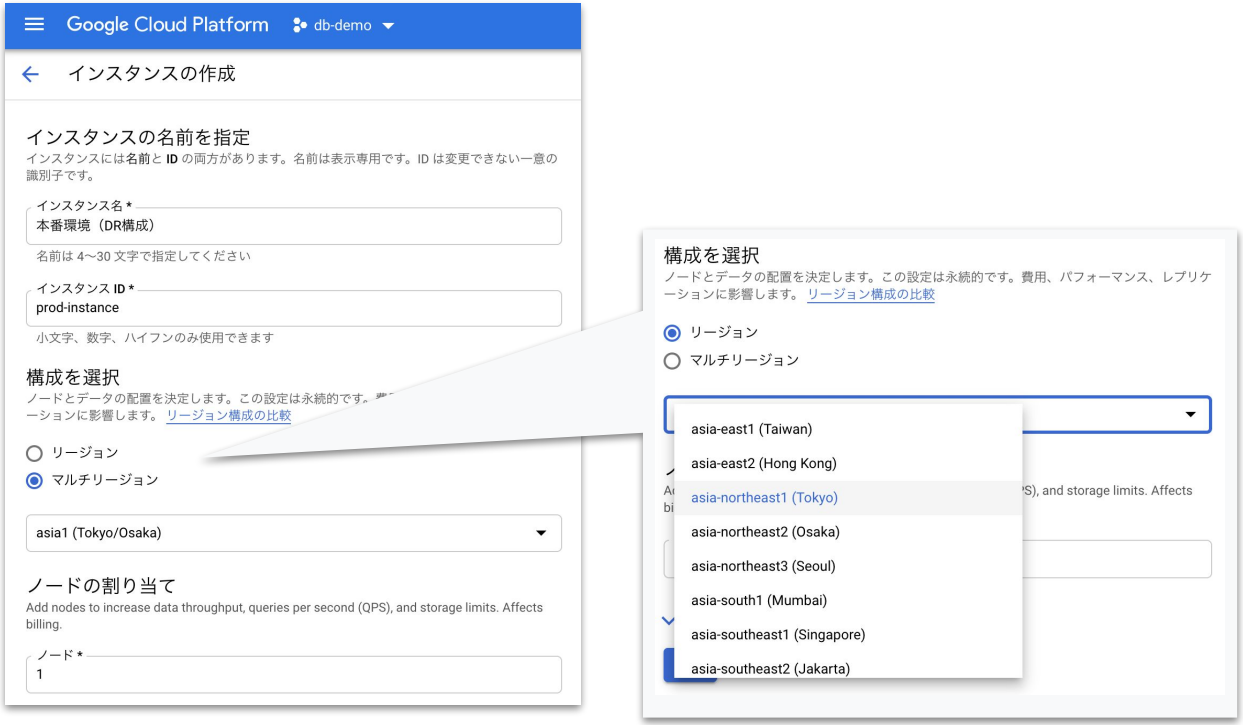
Cloud Spanner インスタンスの作成画面

必要な入力項目は
わずか 3 種類

1. インスタンス名 / ID
任意の名前を入力

2. 構成
設置するリージョンを選択

3. ノード数
必要なスループット性能に
応じてノード数を入力



Cloud Spanner は SQL による柔軟なクエリが可能

```
1  SELECT
2      COUNT(DISTINCT s.s_i_id)
3  FROM
4      district d
5      INNER JOIN order_line ol ON ol.w_id = d.w_id
6       AND ol.d_id = d.d_id
7       AND ol.o_id BETWEEN d.d_next_o_id - 20
8       AND d.d_next_o_id - 1
9      INNER JOIN stock s ON s.w_id = ol.w_id
10      AND s.s_i_id = ol.ol_i_id
11 WHERE
12     d.w_id = 1 AND d.d_id = 1
13      AND s.s_quantity < 50
```

クエリを実行 | クエリをクリア | SQL クエリのヘルプ

候補: Cmd + Space クエリを実行: Cmd

スキーマ　結果表　説明

このクエリはオプティマイザーのバージョン 2 を使用して実行されました

合計経過時間	CPU 時間	返された行数	スキャンされた行数
19.66 msecs	17.72 msecs	1	415

item

スキーマ　インデックス　データ

列	タイプ	Nullable
i_id	INT64	○
i_data	STRING(50)	○
i_im_id	INT64	○
i_name	STRING(24)	○
i_price	FLOAT64	○

スキーマ及び SQL

一般的な RDBMS と同様に、スキーマや SQL をサポート。複雑な JOIN を伴うクエリも実行可能。

Cloud Spanner は SQL による柔軟なクエリが可能

Cloud Spanner　概要　ガイド　リファレンス　例　サポート　リソース

ホーム › Cloud Spanner › ドキュメント › リファレンス　評価とクチコミ

クエリ構文

フィードバックを送信

クエリ ステートメントは、1 つ以上のテーブルまたは式をスキャンし、計算結果の行を返します。このトピックでは、Cloud Spanner SQL での SQL クエリの構文について説明します。

SQL 構文

```
query_statement:
    [ statement_hint_expr ][ table_hint_expr ][ join_hint_expr ]
    query_expr

statement_hint_expr:
    '@{' statement_hint_key = statement_hint_value [, ...] '}'

statement_hint_key:
    { USE_ADDITIONAL_PARALLELISM | OPTIMIZER_VERSION  | LOCK_SCANNED_RANGES }

query_expr:
    [ WITH with_query_name AS ( query_expr ) [, ...] ]
    { select | ( query_expr ) | query_expr set_op query_expr }
    [ ORDER BY expression [{ ASC | DESC }] [, ...] ]
    [ LIMIT count [ OFFSET skip_rows ] ]

select:
    SELECT [ AS { STRUCT | VALUE } ] [{ ALL | DISTINCT }]
        { [ expression. ]* [ EXCEPT ( column_name [, ...] ) ]
            [ REPLACE ( expression [ AS ] column_name [, ...] ) ]
        | expression [ [ AS ] alias ] } [, ...]
    [ FROM from_item [ tablesample_type ] [, ...] ]
```


https://cloud.google.com/spanner/docs/query-syntax?hl=ja

Cloud Spanner はトランザクションをサポート

```
spanner> begin;
Query OK, 0 rows affected (0.04 sec)

spanner(rw txn)> select s_i_id as `商品ID`, w_id as `倉庫ID`, s
_quantity as `在庫数` from stock where s_i_id = 1 and w_id = 1;
+--------+--------+-----------+
| 商品ID | 倉庫ID | 在庫数    |
+--------+--------+-----------+
| 1      | 1      | 77.000000 |
+--------+--------+-----------+
1 rows in set (2.16 msecs)

spanner(rw txn)> update stock set s_quantity = 76 where s_i_id
= 1 and w_id = 1;
Query OK, 1 rows affected (0.01 sec)

spanner(rw txn)> commit;
ERROR: spanner: code = "Aborted", desc = "Transaction was abort
ed. It was wounded by a higher priority transaction due to conf
lict on keys in range [[1,1], [1,1]), column s_quantity in tabl
e stock."
spanner>
```

ACID トランザクション
一般的な RDBMS と同様に、トランザクションをサポート。トランザクション分離レベルとしては SERIALIZABLE であり、OLTP 系のワークロードに対して、データの整合性を崩すことなく更新が可能。

様々なトランザクションをサポート
読み書きトランザクション以外にも、読み込み専用のトランザクションをサポート。
トランザクション間の整合性を保ちつつ、他のトランザクションを妨げない。過去の時間断面に対して、クエリを投げることも可能。

様々な言語環境で Cloud Spanner 用アプリを開発

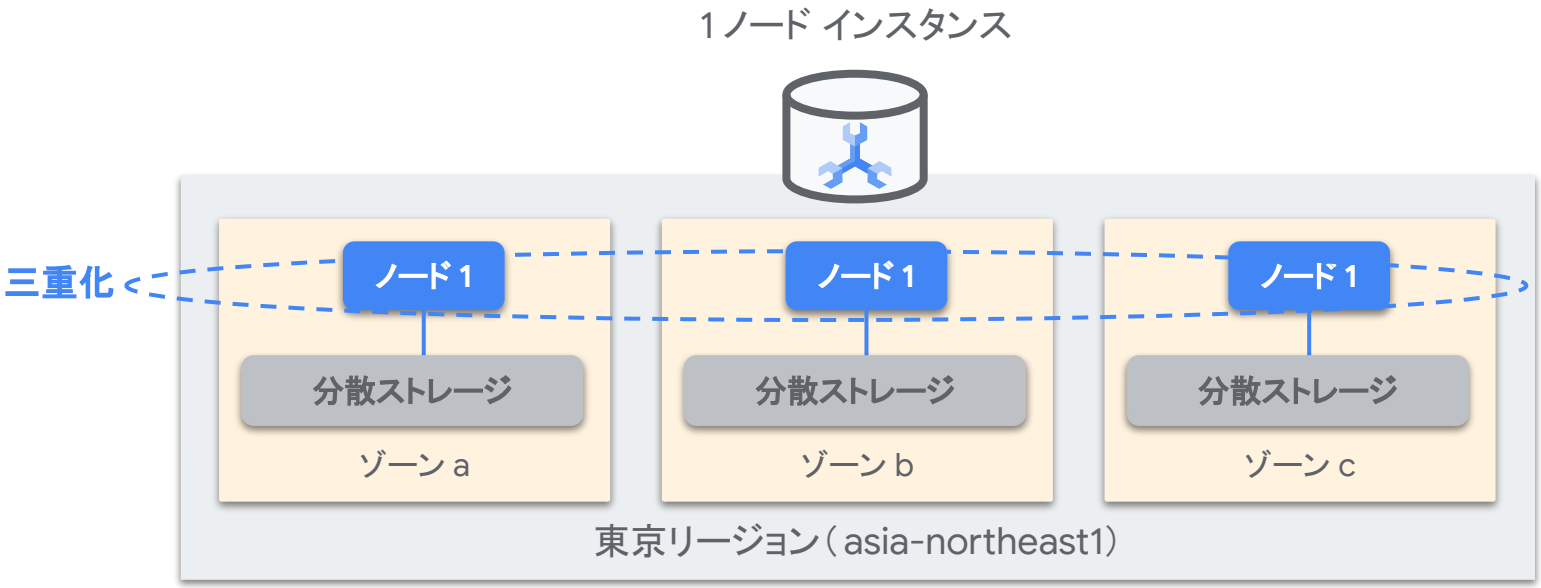
各種言語ごとのクライアント ライブラリ
C++, C#, Go, Java, PHP, Python, Ruby, Node.js といった主要言語のネイティブ クライアント ライブラリを提供。Cloud Spanner の API を利用したデータの操作や、SQL を利用した操作が可能。

JDBC ドライバー、Hibernate ORM
JDBC ドライバーを提供。汎用的な JDBC を用いた開発による開発コスト削減だけではなく、JDBC に対応した既存アプリケーションを Cloud Spanner に対応させることも容易に可能。Java の Hibernate ORM や Python の Django ORM なども提供。

```
public int updateRecordUsingJDBC (int id, long col1)
throws SQLException {
   PreparedStatement ps = connection.prepareStatement (
           "UPDATE table01 SET col1 = ? WHERE id = ?");
   ps.setLong (1, col1);
   ps.setLong (2, id);
   ps.executeUpdate ();
```

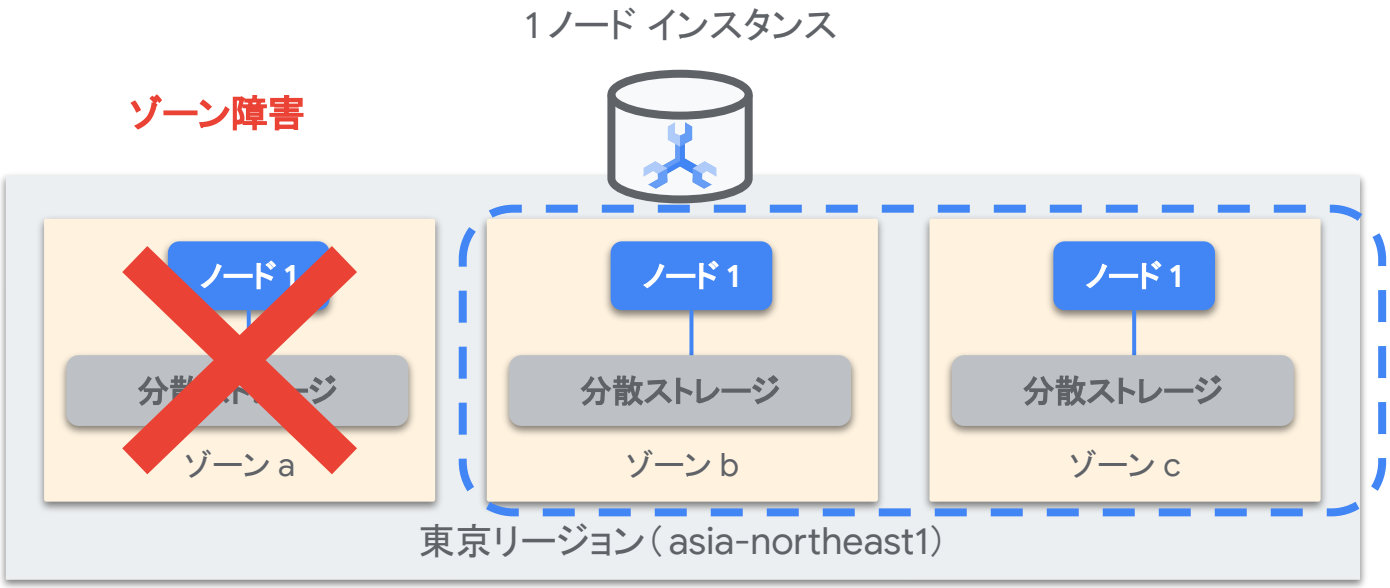
Cloud Spanner は 1 ノードであっても冗長化されている

Cloud Spanner インスタンスとは、ゾーンごとに処理用のタスクが起動しており、1 ノード構成であっても可用性を保つ構成になっている。



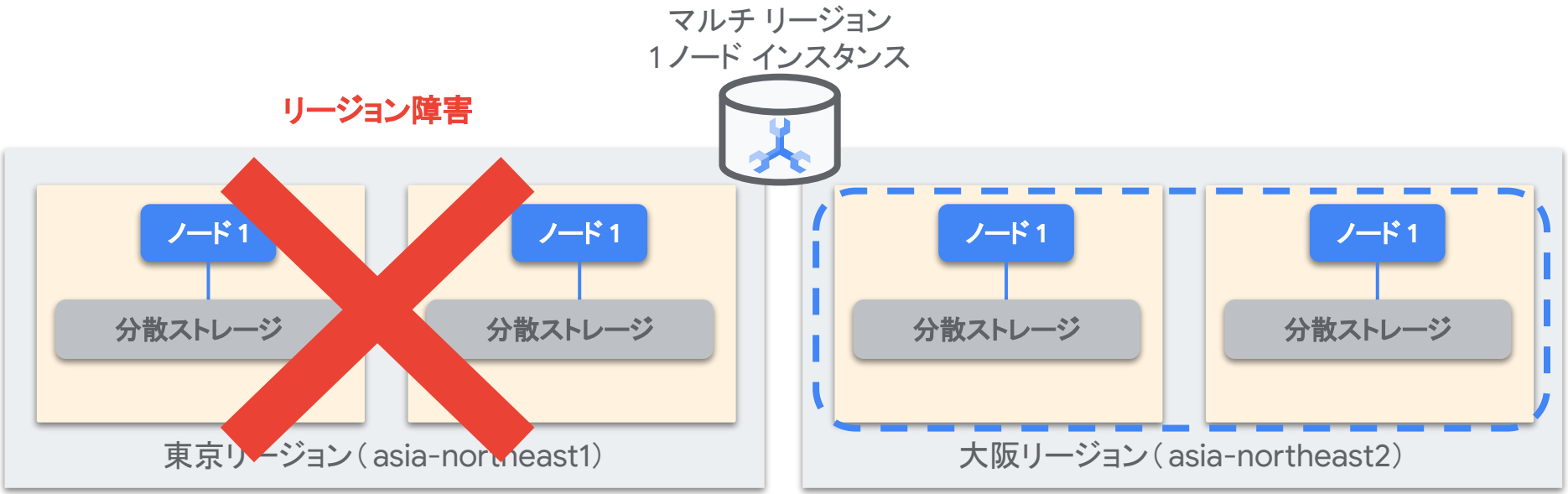
Cloud Spanner は 1 ノードであっても冗長化されている

単一リージョン構成では、ゾーン障害が起こっても処理を継続可能であり、
可用性 99.99% を満たす。



Cloud Spanner は 1 ノードであっても冗長化されている

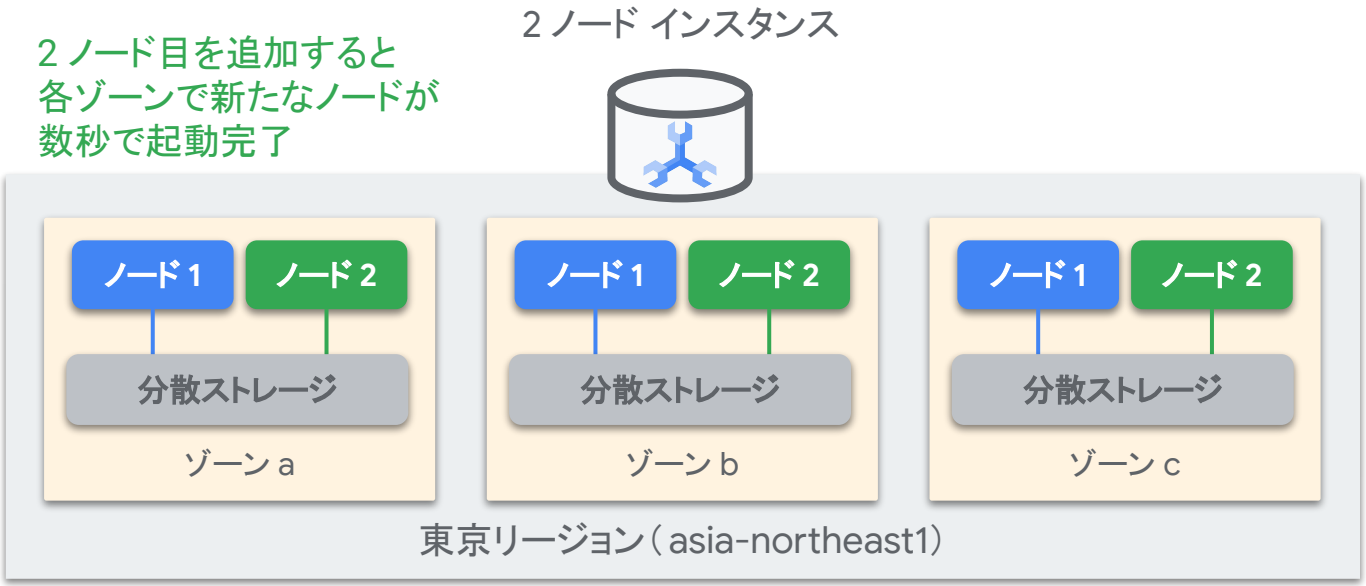
マルチ リージョン構成では、リージョン障害が起こっても処理を継続可能であり、
可用性 99.999% を満たす。



Cloud Spanner のノード追加はわずか数秒で完了

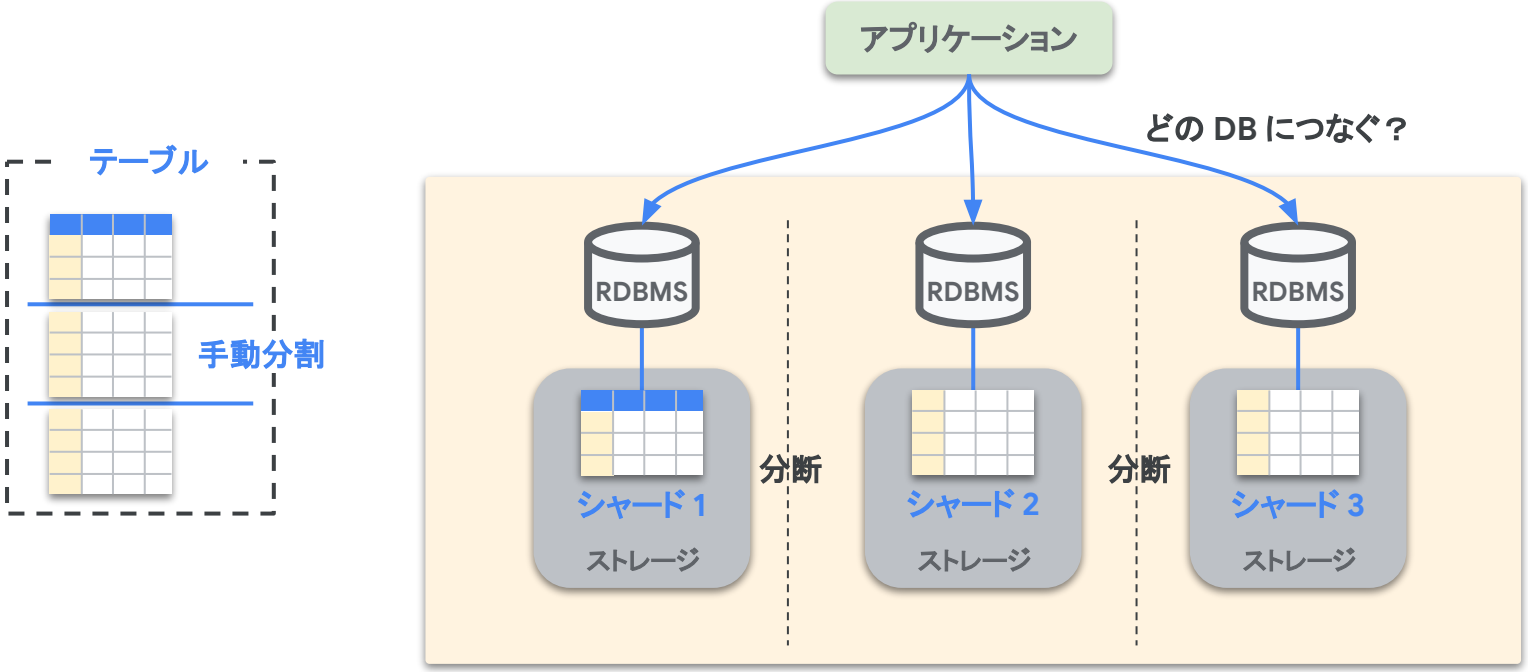
ノード追加とは、新たなコンテナが起動するだけ。データ自体は分散ストレージ経由で共有されているため、ノードの追加と削除は速やかに完了する。

2 ノード目を追加すると
各ゾーンで新たなノードが
数秒で起動完了



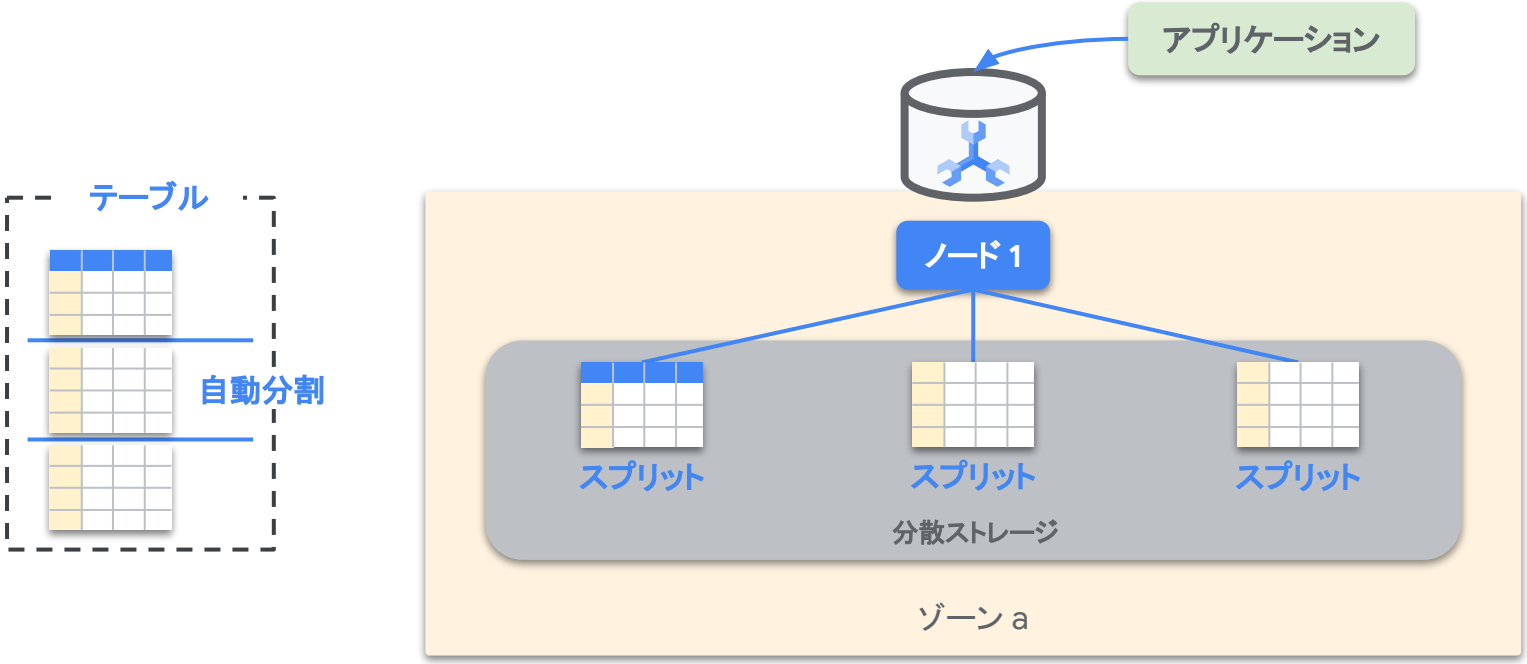
一般的な RDBMS の手動シャーディング

典型的な RDBMS でスケールアウトをするためには、ユーザー管理によって DB を手動で分割。
各シャードは物理的に分割されているため、接続先 DB 含めてアプリ側で意識して扱う必要有り。



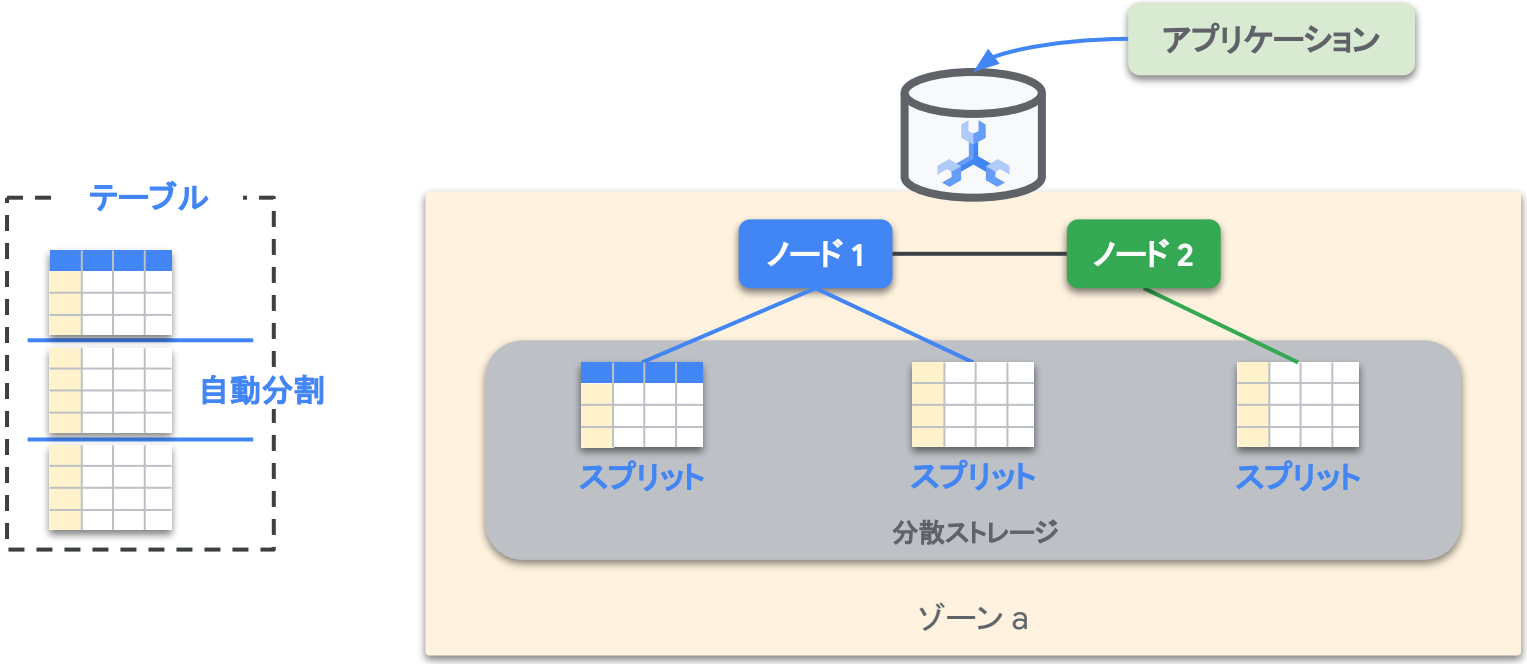
Cloud Spanner の自動シャーディング（1 ノード）

Cloud Spanner に作られたテーブルは、主キー（ PK）のレンジで、自動的に分割されて保存されている。この分割単位をスプリットと呼ぶ。



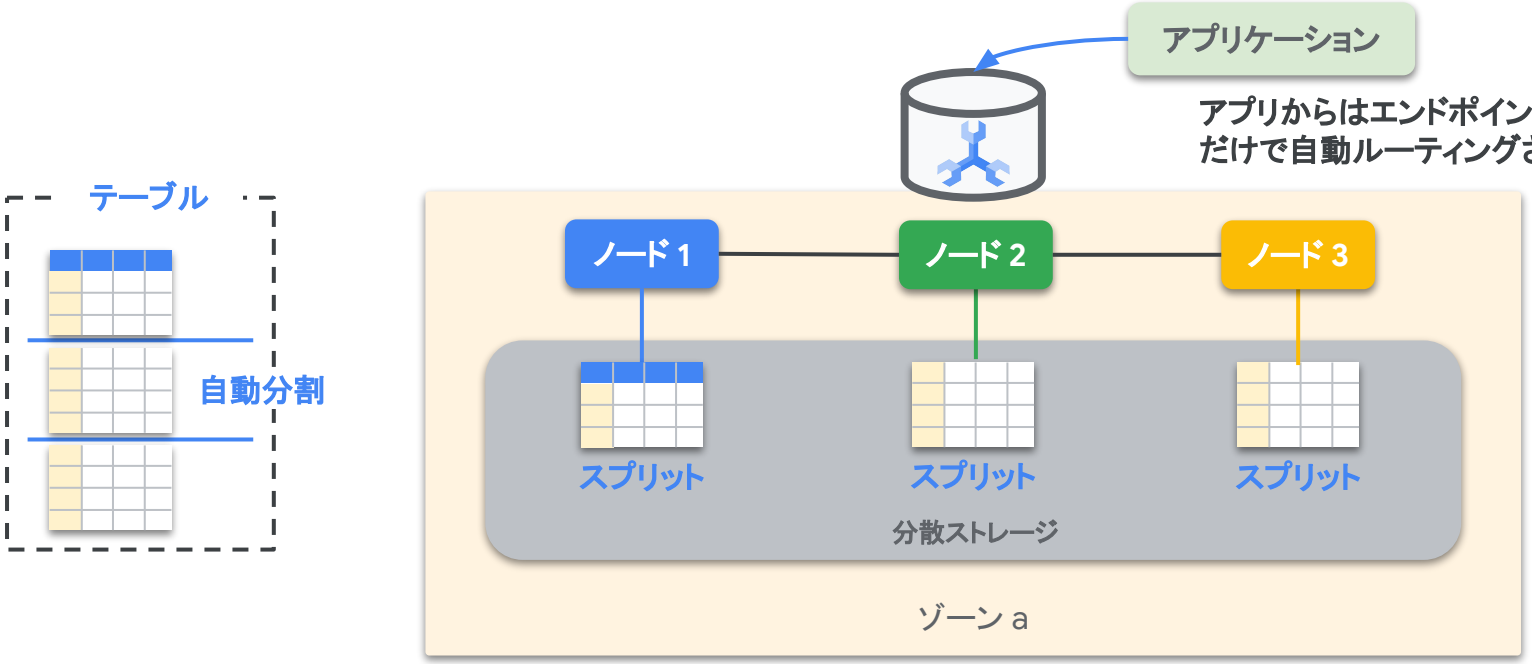
Cloud Spanner の自動シャーディング（2 ノード）

スプリットは分散ストレージに保存されているため、ノードが増えると担当するスプリットを変更することによって、性能がスケールするようになっている。



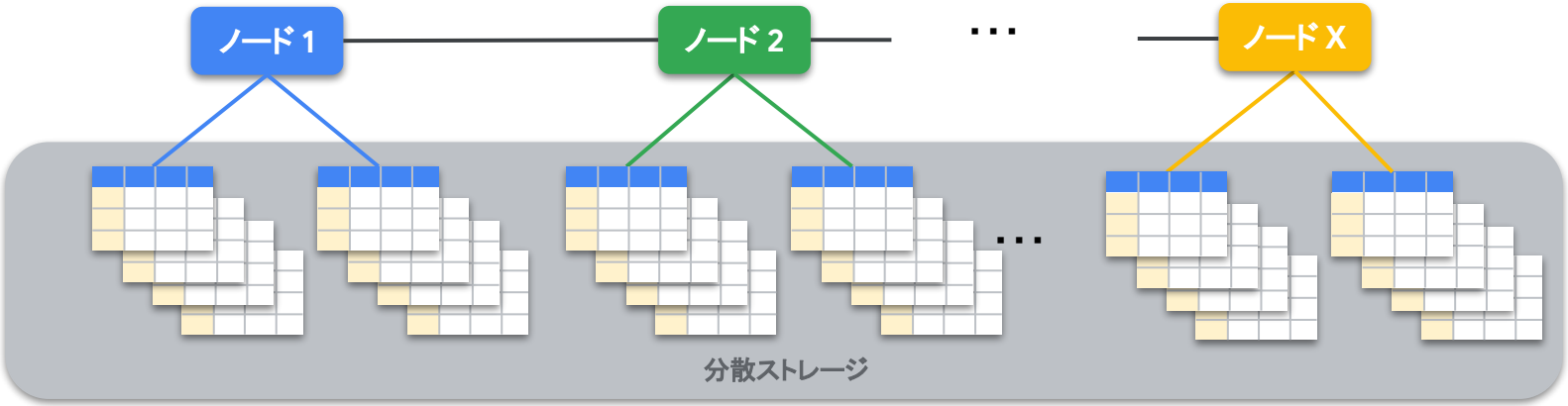
Cloud Spanner の自動シャーディング（3 ノード）

スプリットは分散ストレージに保存されているため、ノードが増えると担当するスプリットを変更することによって、性能がスケールするようになっている。



自在にスケール可能な Cloud Spanner

このようにして Cloud Spanner は、1 ノードからスモール スタートし、小規模構成から数百ノードもの大規模構成まで、柔軟にスケールさせることが可能。分散ストレージ内のスプリットは、データサイズや負荷状況に応じて分割や結合が行われる。



Cloud Spanner の
ユースケース

03

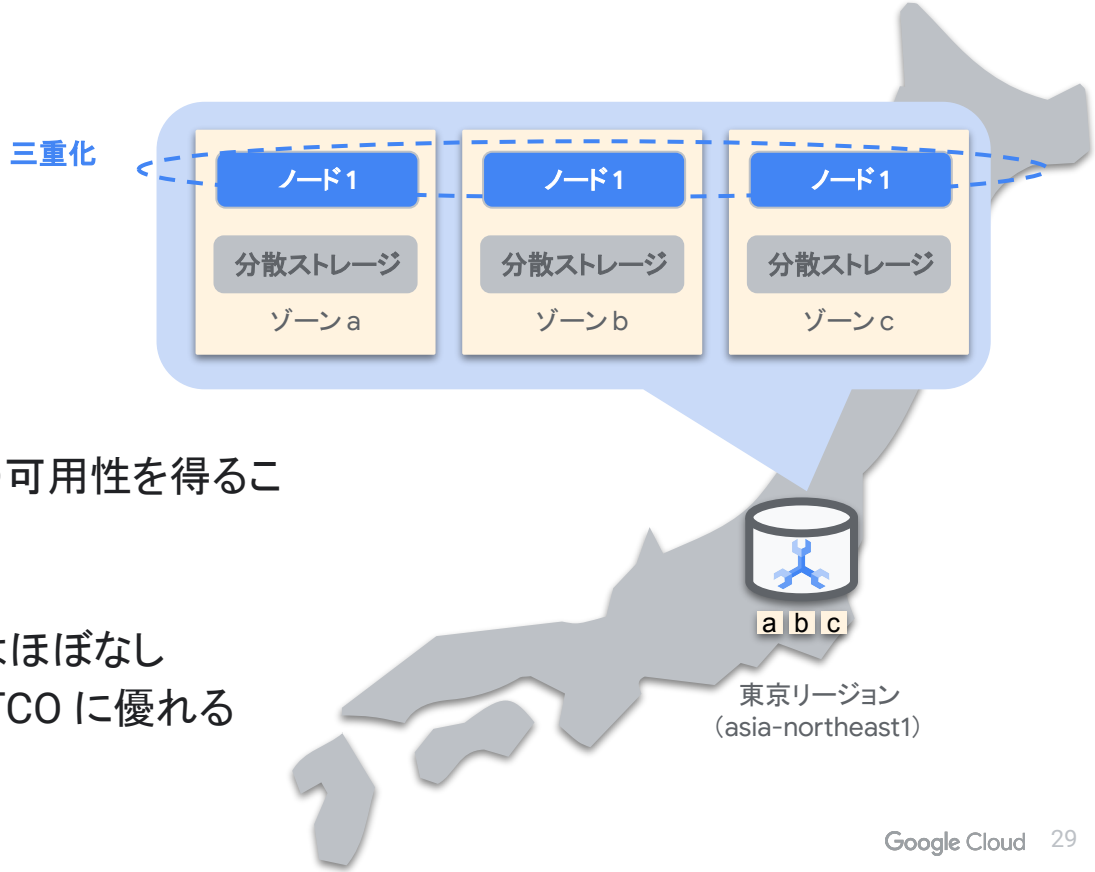
ユースケース 1: 高可用性が欲しい、でも運用は楽したい

対象ユーザー

- 高可用な RDBMS が欲しい
- 運用は簡単に済ませたい

なぜ Cloud Spanner？

- 1 ノードでも 99.99%〜99.999% の可用性を得ることができる
- 東京大阪の DR 構成も簡単
- フルマネージドであり、運用負担はほぼなし
- DB にかかる工数を減らせるのでTCO に優れる



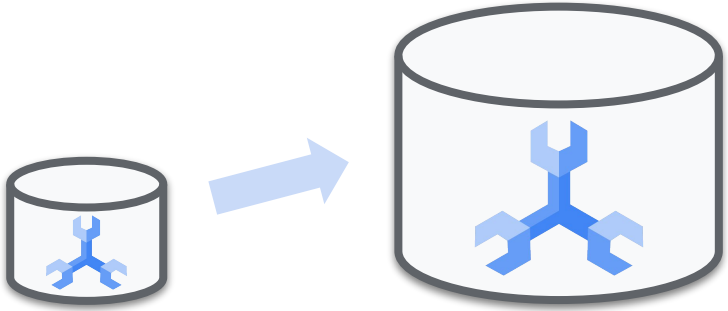
ユースケース 2 : スモール スタートしたい

対象ユーザー

- まずはサービスをスモールスタートしたい
- でも将来拡張する必要があるかも

なぜ Cloud Spanner？

- 1 ノードでも 99.99%～99.999% の可用性を得ることができるため、スモールスタートに向いている
- 性能が必要になったらあとからノード追加も簡単に行える
- DB にかかる工数を削減しTCO に優れるため、サービスやアプリの改良に専念できる



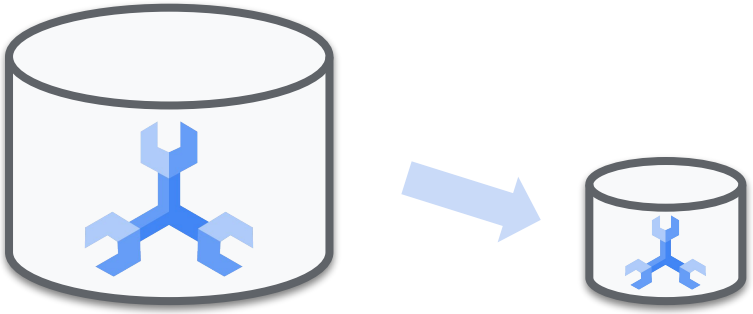
ユースケース 3 : 最初は大規模であとから規模縮小したい

対象ユーザー

- ローンチ直後に最も性能が必要
- 後ほどサービス縮小する可能性もある

なぜ Cloud Spanner？

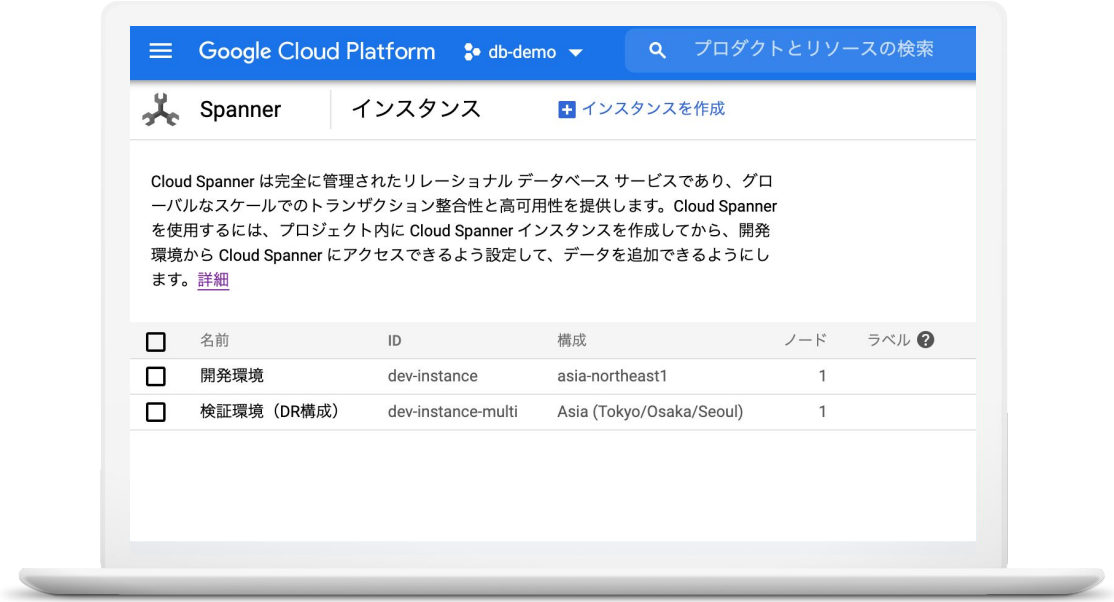
- 自動シャーディング機能は、ノード追加だけでなくノード削除にも対応
- 性能が必要なフェーズではノードを多めにし、性能に余裕が出てきたらノードを減らすことができる
- 平日と週末でノード数を変える運用も



1ノードから
はじめる
実践活用方法

04

Cloud Spanner インスタンスを作ろう



Cloud Spanner インスタンスの作成画面

1. インスタンス名 / ID
任意の名前を入力。

2. 構成
単一リージョンかマルチリージョンかを選び、利用するリージョンを選択。

3. ノード数
必要なスループット性能に応じてノード数を入力。

可用性に影響する設定

性能（スループット）に影響

ノード数はスループットの性能指標

- ノード数はスループットの性能指標であり、可用性とは無関係
- シングル リージョンの性能
  - 1 ノードあたり Read 10,000 QPS、Write 2,000 QPS
- マルチ リージョンの性能
  - 1 ノードあたり Read 7,000 QPS、Write 1,800 QPS
- あくまで目安なので、実際には性能検証を行うこと

パフォーマンス

上記のベスト プラクティスに準拠すると、各 Cloud Spanner ノードは最大 10,000 QPS の読み取りまたは最大 2,000 QPS の書き込み（行ごとに 1 KB データ）を実現できます。

https://cloud.google.com/spanner/docs/instances?hl=ja#regional-performance

アクセス権限の設定を行う

Google Cloud Platform　db-demo　プロダクトとリソースの検索

Spanner　すべてのインスタンス　情報パネルを非表示

インスタンス　インスタンスを作成

Cloud Spanner は完全に管理されたリレーショナル データベース サービスであり、グローバルなスケールでのトランザクション整合性と高可用性を提供します。Cloud Spanner を使用するには、プロジェクト内に Cloud Spanner インスタンスを作成してから、開発環境から Cloud Spanner にアクセスできるよう設定して、データを追加できるようにします。詳細

フィルタ　プロパティ名または値を入力

名前	ID	構成	ノード	ストレージの利用率	ラベル
開発環境	dev-instance	asia-northeast1	1	33 GB / 2 TB	
開発環境（DR構成）	dev-instance-multi	Asia (Tokyo/Osaka/Seoul)	1	0 B / 2 TB	
本番環境（東京）	prod-instance-tokyo	asia-northeast1	1	38.1 GB / 2 TB	

本番環境（東京）

権限　ラベル

下の権限を編集または削除するか、[メンバーを追加] で新たに権限を付与するメンバーを追加します　メンバーを追加

継承された権限を表示

フィルタ　プロパティ名または値を入力

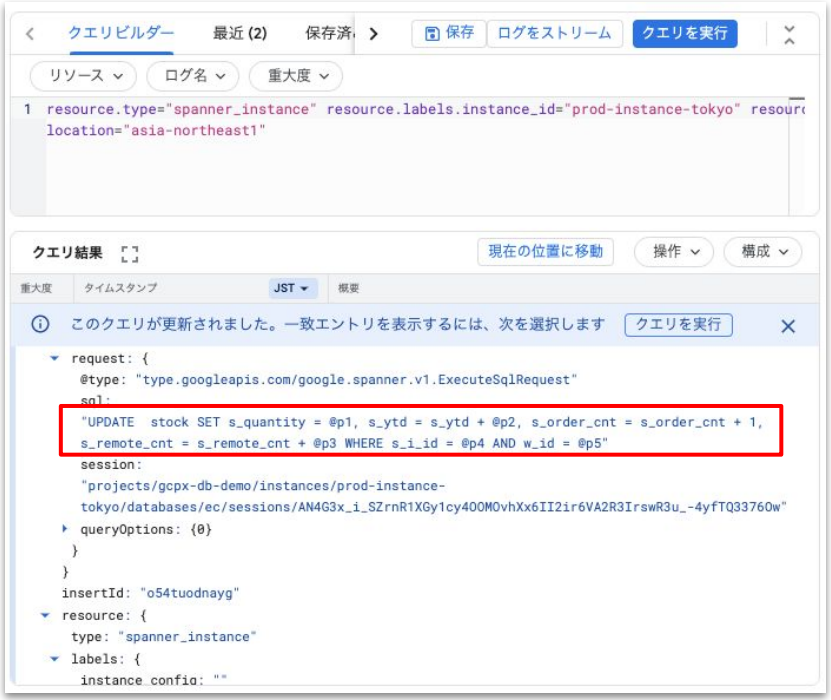
ロール / メンバー　継承

- Cloud Data Fusion API サービス エージェント (1)
- Cloud Spanner データベース ユーザー (1)
- Cloud Spanner データベース管理者 (2)
- Cloud Spanner 管理者 (1)
- オーナー (2)
- 編集者 (4)

アクセス権限の設定

Cloud Spanner インスタンスにアクセスできるユーザー（ DBA など）や、サービスアカウントを適切に設定する。

監査ログの設定を行う



監査ログの有効化

IAM と管理ページにある監査ログ設定より、Cloud Spanner の監査ログを有効にできる。
監査を有効にしても DB への性能影響はなし。

監査ログの閲覧

監査ログは Cloud Logging で閲覧可能。

Cloud Spanner インスタンスの管理画面

Google Cloud Platform　db-demo　プロダクトとリソースの検索

Spanner

インスタンス
- 概要
- モニタリング
- インポート / エクスポート
- バックアップ / 復元

すべてのインスタンス > インスタンス prod-instance-tokyo: 概要

インスタンスを編集　インスタンスを削除　情報パネルを表示

ノード数（性能）変更

概要

名前	本番環境（東京）
ID	prod-instance-tokyo
構成	asia-northeast1

ノード	CPU 使用率（平均）	オペレーション	スループット	データベース ストレージの合計
1	56.90%	読み取り: 2,407.23/秒 書き込み: 84.18/秒	読み取り: 329.18 KB/秒 書き込み: 14.93 KB/秒	38.2 GB / 2 TB

データベースの作成と管理

データベース

データベースの作成

フィルタ　データベースをフィルタ

名前	CPU 使用率	サイズ	Version retention period
ec	56.90%	38.2 GB (2%)	7 日

Cloud Spanner のノード数（性能）変更

ノードの追加と削除

Cloud Spanner のノード数を変更する場合、編集画面を開きノードの割り当て数を変更するだけで完了。

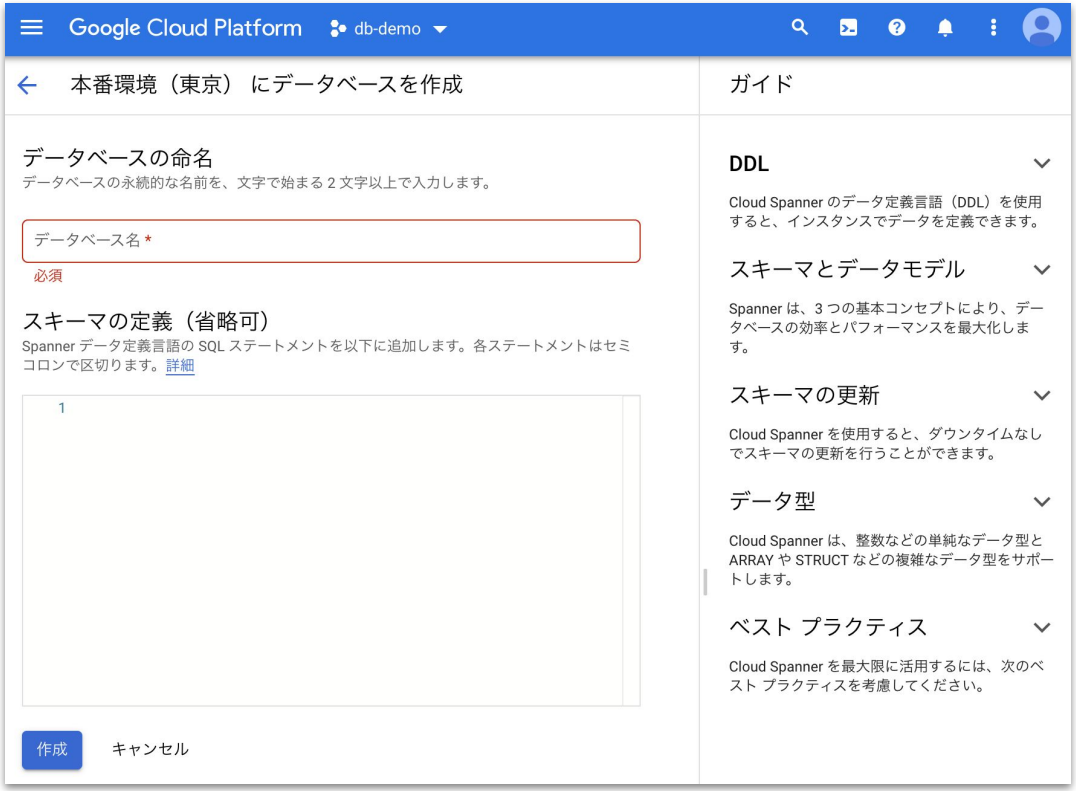
ノード数変更にダウンタイム無し

ノード追加であってもノード削減であっても、一切のダウンタイムなく実施することが可能。

データベースの作成と管理

DB の作成
1 つのインスタンス内に、最大 100 個の DB を持つことができる。

スキーマ
CREATE TABLE や CREATE INDEX といった、DDL でスキーマを定義する。



データベースの管理画面

Google Cloud Platform　db-demo　プロダクトとリソースの検索

Spanner

すべてのインスタンス > インスタンス prod-instance-tokyo: 概要 > データベース ec: 概要　データベースの削除　情報パネルを

データベース

- 概要
- モニタリング
- クエリの統計データ
- インポート / エクスポート
- バックアップ / 復元
- クエリ

概要

データベース名	ec
バージョンの保持期間	7 日

CPU 使用率（平均）	オペレーション	スループット	データベース ストレージの合計
54.34%	読み取り: 2,361.55/秒 書き込み: 82.60/秒	読み取り: 321.88 KB/秒 書き込み: 14.56 KB/秒	38.26 GB

テーブル

テーブルを作成

フィルタ　テーブルのフィルタ

名前	インデックス	インターリーブされたテーブル
customer	1	district
district	–	warehouse
history	–	–

Cloud Spanner の DB 運用管理は何をすれば良い？

DB を作成してテーブルさえ作ってしまえば、インスタンス及び DB の運用はフルマネージドなため**基本的にやることはほぼ無い**。

運用監視でユーザーがアクションを取る必要があるもの

- 計算リソース（CPU）またはストレージが足りなくなったら**ノードを追加**
- 必要に応じて**データのバックアップ**をする

計算リソース不足やストレージはどこで判断する？

計算リソース不足の確認
インスタンス全体のモニタリング ページより計算リソース（CPU）不足を確認できる。

ストレージ不足の確認
インスタンス全体の概要ページより状況が確認できる。1 ノードあたり 2 TB のデータを処理可能。

ノード追加を検討する CPU 使用率のしきい値

インスタンスのモニタリング
CPU 使用率がグラフにかかれている推奨最大値を超えている場合、計算リソースが逼迫してきていることを意味するため、**ノード追加などを検討する。**

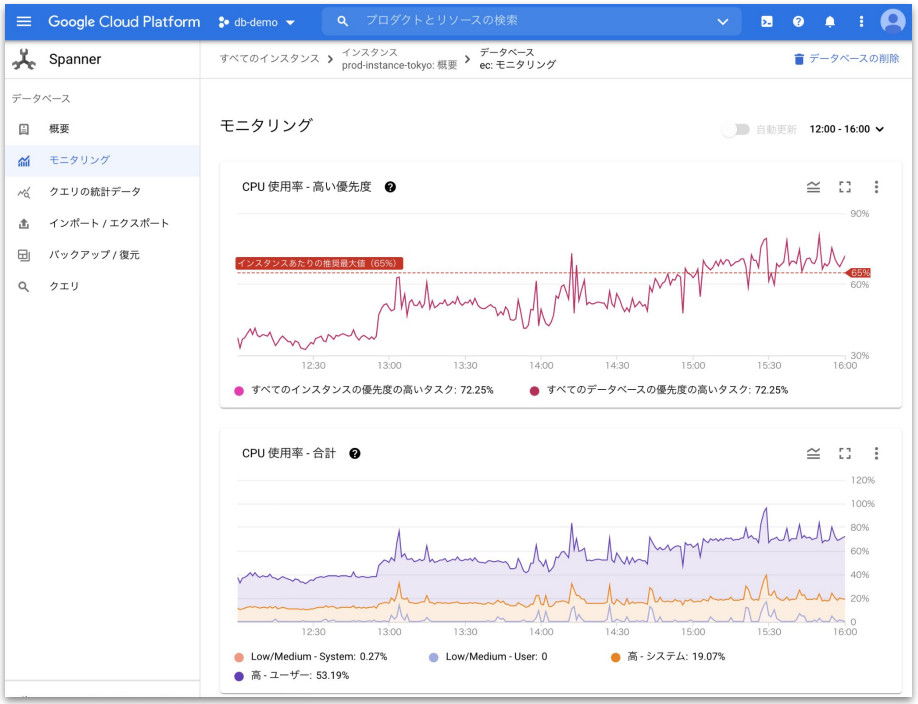
CPU 使用率 - 移動平均
移動平均 24 時間でみた、CPU 使用率全体。

CPU 使用率 - 高い優先度
リアルタイムの CPU 使用率のうち、優先度が高い処理のもの。ユーザーのクエリや更新処理は高優先度に含まれる。

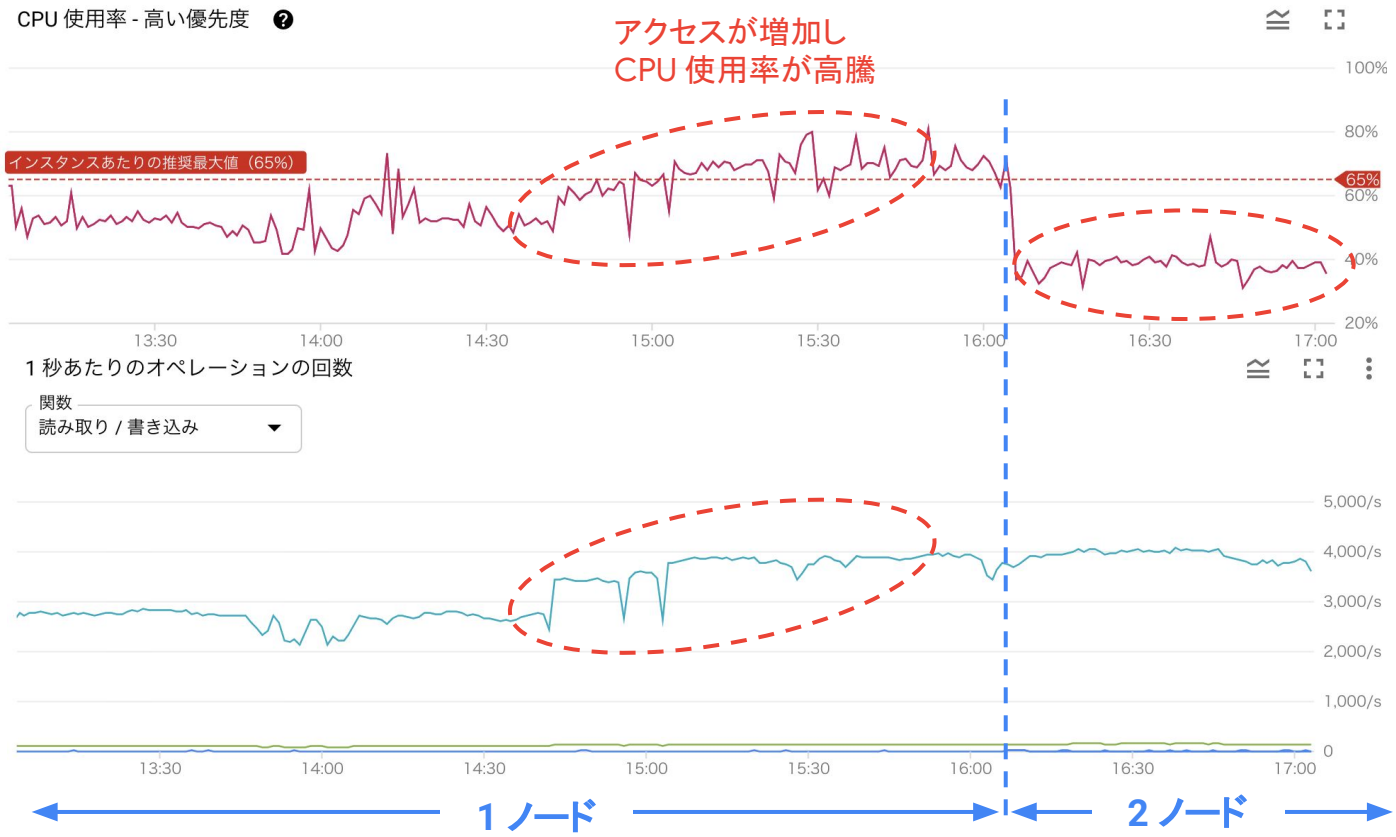
個々のデータベースのモニタリング

各データベースのモニタリング
モニタリングのページは、インスタンス全体のものと、個々のデータベースのものの両方がある。ノード追加が必要かどうかについては、インスタンス全体で確認し、その後必要に応じて個々のデータベースごとの状況を確認する。

Cloud Monitoring
Cloud Monitoring の Metrics Explorer では、Cloud Spanner のモニタリングのページでグラフ化されていない様々な情報を確認可能。
チューニングや性能分析をさらに行っていく場合は、こちらが効果的。



CPU 使用率の逼迫に対してノードを追加を行う



ノード追加削除を自動で行う Autoscaler

- Cloud Spanner の負荷状況を自動でチェックし、必要に応じてノードの追加削除を行う
- ノードの増減のスケジューリングについては、設定ファイルにて細かく制御することが可能
- OSS として Cloud Spanner Ecosystem で公開されている

データベース

新しい Autoscaler で Spanner インスタンスを適切なサイズに自動変更

https://cloud.google.com/blog/ja/products/databases/cloud-database-scales-instance-sizes-easily

Cloud Spanner のバックアップ

マネージド バックアップ リストア

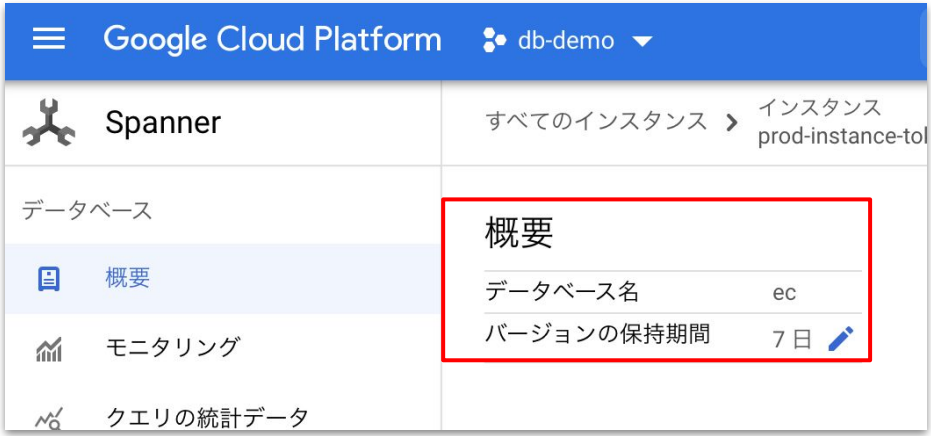
インスタンス内の DB 単位でバックアップ取得可能。バックアップは、インスタンスに紐付いた専用領域に保管され、最大 1 年間保持可能。

また、同一インスタンス及び、同一リージョンの別のインスタンスに対してリストアが可能。

任意の時間断面でのバックアップ（PITR）

時刻指定で、任意の時間断面でのバックアップを取得することが可能。これは DB の過去のバージョンのバックアップであり、バージョン保存期間（デフォルト 1 時間）分だけさかのぼって実施できる。

バージョン保存期間の変更をして過去のバックアップを取る



バージョンの保存期間
デフォルト 1 時間で、最大 7 日まで伸ばすことができる。過去のバージョンがを残すことで、
バックアップだけでなく、過去のバージョンを
直接 SELECT するなど活用が可能。

データの過去のバージョンを直接 SELECT する

Stale Read を行うことで、過去のデータをタイムスタンプ指定で SELECT することもできる。

例）過去の在庫数を確認する

右図は、ある商品の、現在の在庫数と過去の在庫数をそれぞれクエリしている例。

アプリケーションの不具合や、ヒューマンエラーなどでデータを失っても、過去の情報を簡単に復元できる。

```
spanner> select s_i_id as `商品ID`, w_id as `倉庫ID`, s_quantity a
s `在庫数` from stock where s_i_id = 1 and w_id = 1;
+--------+--------+-----------+
| 商品ID | 倉庫ID | 在庫数    |
+--------+--------+-----------+
| 1      | 1      | 77.000000 |
+--------+--------+-----------+
1 rows in set (3.01 msecs)

spanner> BEGIN RO 2021-03-29T09:00:00Z;
Query OK, 0 rows affected (0.01 sec)

spanner(ro txn)> select s_i_id as `商品ID`, w_id as `倉庫ID`, s_qu
antity as `在庫数` from stock where s_i_id = 1 and w_id = 1;
+--------+--------+-----------+
| 商品ID | 倉庫ID | 在庫数    |
+--------+--------+-----------+
| 1      | 1      | 85.000000 |
+--------+--------+-----------+
1 rows in set (4.43 msecs)

spanner(ro txn)>
```

※spanner-cli を使ってクエリを投げている。Cloud Spanner の GUI コンソールではできないので注意。

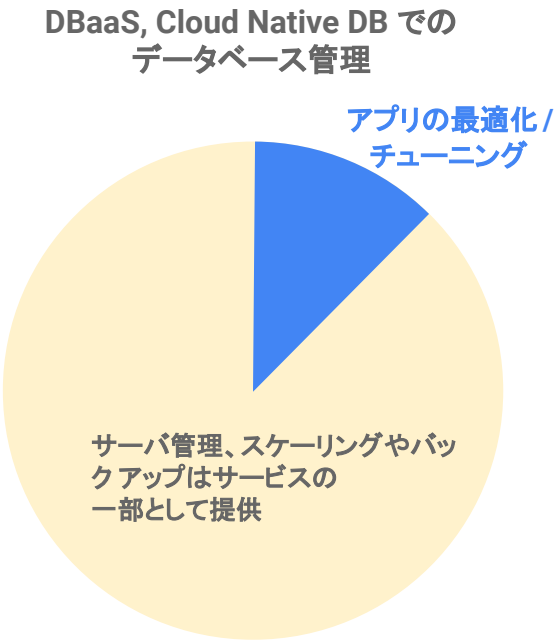
運用開発を助ける spanner-cli

- 対話形式で Cloud Spanner に対して SQL を実行できるツール
- MySQL の mysql コマンド、PostgreSQL の psql コマンドに似たもの
- Cloud Spanner Ecosystem にて OSS として公開されている
- https://github.com/cloudspannerecosystem/spanner-cli

```
spanner> show databases;
+----------+
| Database |
+----------+
| ec       |
+----------+
1 rows in set (0.26 sec)

spanner> show tables;
+--------------+
| Tables_in_ec |
+--------------+
| customer     |
| district     |
| history      |
| item         |
| new_orders   |
| order_line   |
| orders       |
| stock        |
| warehouse    |
+--------------+
9 rows in set (0.08 sec)

spanner> select count(*) from history;
+---------+
|         |
+---------+
| 9102313 |
+---------+
1 rows in set (2.13 secs)

spanner>
```

おわりに

- 1 ノードから使えるフルマネージド データベース Cloud Spanner によって可用性、性能と拡張性、運用と保守性、そしてセキュリティ様々な非機能要件を容易に実現可能
- これによりエンジニアはアプリの最適化やチューニングに時間をさくことができる



可用性　性能と拡張性　運用と保守性　セキュリティ

Thank you