

# これで開発効率アップ Cloud Spanner 最新アップデート 2022 ～PostgreSQL インターフェースと ORM の紹介～

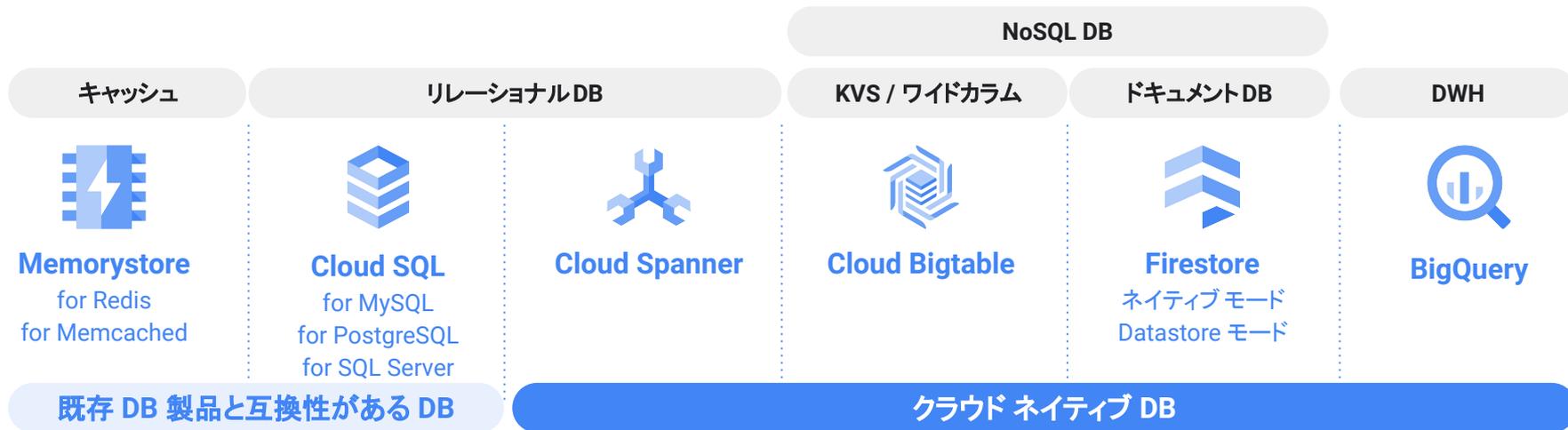
Google Cloud  
データベース スペシャリスト  
佐藤 貴彦

Google Cloud  
データベース スペシャリスト  
江川 大地

01

# 運用いらずのフルマネージド RDBMS Cloud Spanner の概要

# Google Cloud におけるマネージド データベースの選択肢



# クラウド ネイティブなデータベース Cloud Spanner の特徴



## 特徴 1 - 運用いらずのフルマネージドRDBMS

フルマネージドデータベースで、メンテナンス含めあらゆる運用は全自動  
テーブルに対して SQL でのクエリや、ACID トランザクションをサポート



## 特徴 2 - リージョン障害にも耐えられる高可用性

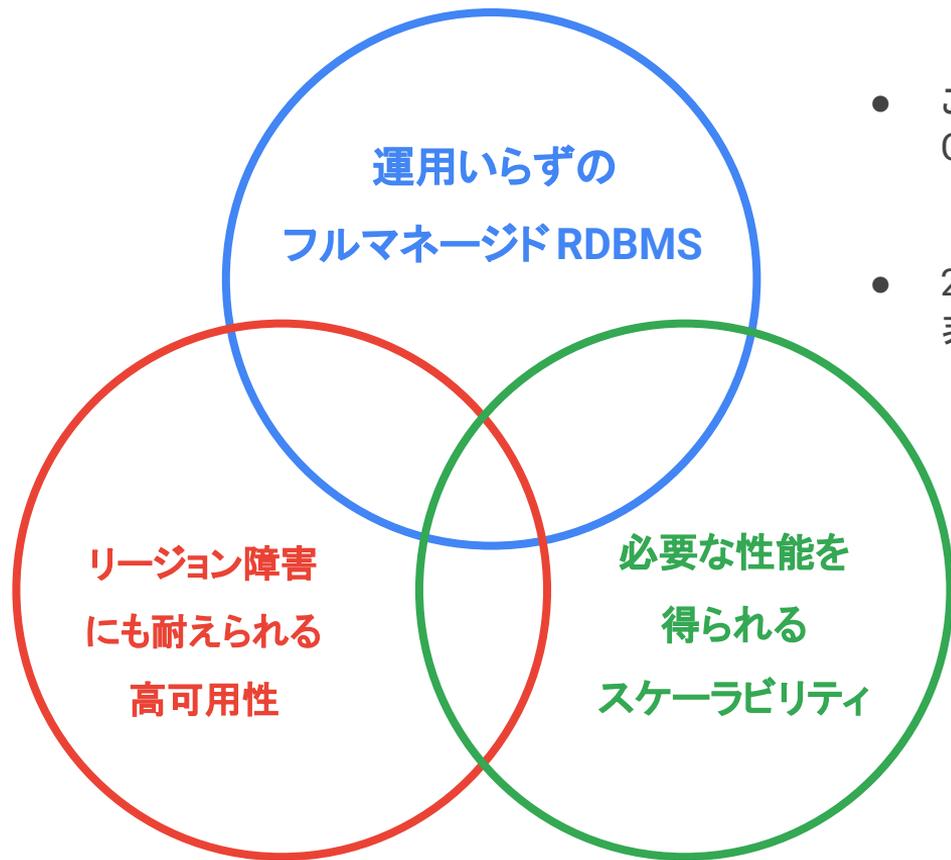
ゾーン障害はもちろんリージョン障害にも耐え、最大99.999 % の可用性を提供  
メンテナンスやノード数変更時のダウンタイムなども無し



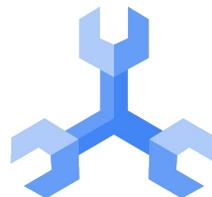
## 特徴 3 - 必要な性能を容易に得られるスケーラビリティ

必要な性能に合わせて、無停止でスケールアウトやスケールインが可能  
テーブル内のデータは自動シャーディングされるため、一切の運用負担はなし

# まずは覚えて欲しい Cloud Spanner の 3 つの特徴



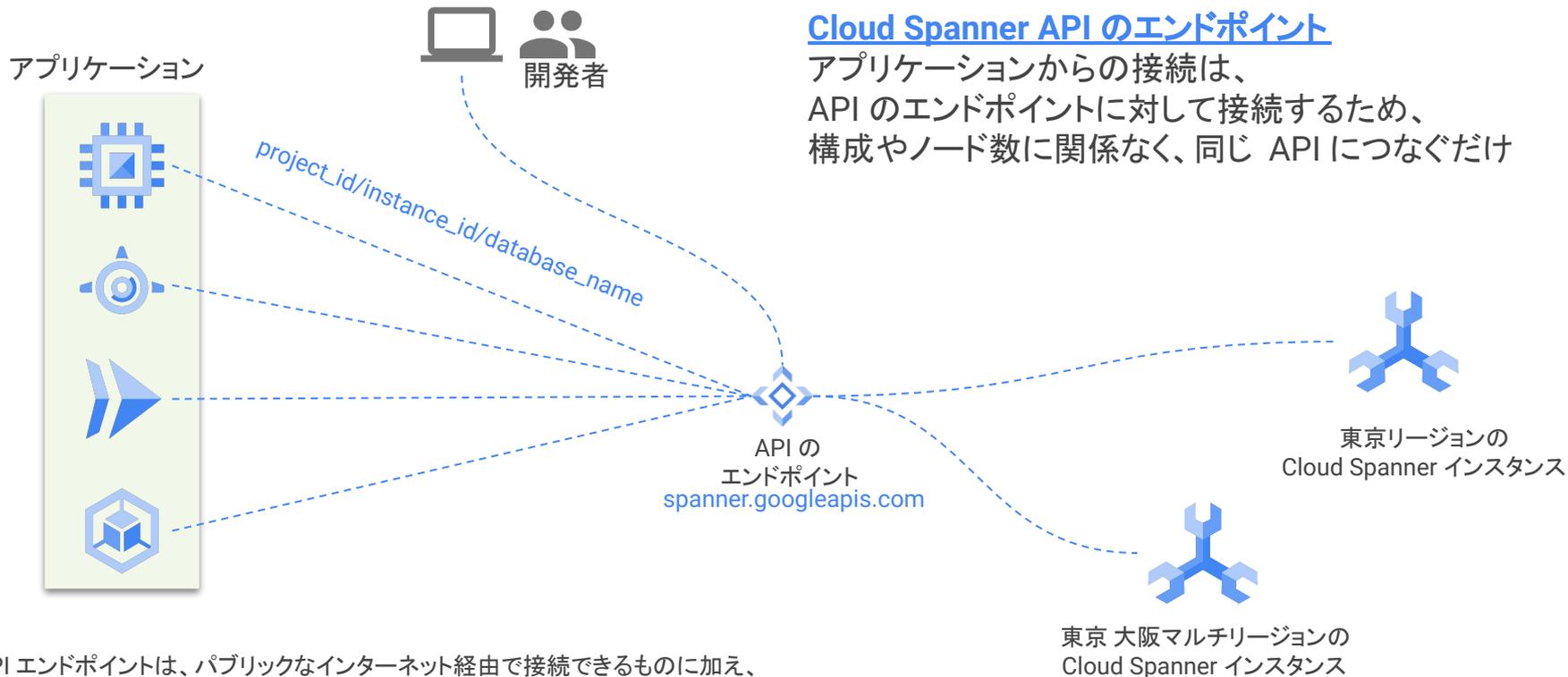
- このうちの 1 つでも必要なシステムなら Cloud Spanner を検討
  - 例) ダウンタイムを無くすために利用
- 2 つ以上同時に必要なら、Cloud Spanner は非常によい選択肢
  - 例) 運用いらずで高可用性が欲しい
  - 例) 運用いらずでスケーラビリティが欲しい



02

# アプリケーションから Cloud Spanner への接続方式

# アプリケーションから API に接続して利用する RDBMS



## Cloud Spanner API のエンドポイント

アプリケーションからの接続は、API のエンドポイントに対して接続するため、構成やノード数に関係なく、同じ API につなぐだけ

API エンドポイントは、パブリックなインターネット経由で接続できるものに加え、VPC や、VPN や Interconnect で接続されたオンプレミス環境のネットワークといった、閉域網のみから接続できるものもある。

# Cloud Spanner は SQL による柔軟なクエリが可能

## 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。

```
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 ?

19.66 msec

CPU 時間 ?

17.72 msec

返された行数

1

スキャンされた行数

415

item

[スキーマ](#)

[インデックス](#)

[データ](#)

列	タイプ	Nullable
<input checked="" type="radio"/> i_id	INT64	<input type="radio"/>
i_data	STRING(50)	<input type="radio"/>
i_im_id	INT64	<input type="radio"/>
i_name	STRING(24)	<input type="radio"/>
i_price	FLOAT64	<input type="radio"/>

画面は Cloud Console 上の GUI から操作している例

# Cloud Spanner に接続するアプリの作成方法

## 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、各種 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、SQLAlchemy、Ruby の Active Record など  
各種 ORM 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 ();
}
```

Java から JDBC 経由で接続している例



# Cloud Spanner のクライアントのドキュメント

Cloud Spanner 概要 ガイド

フィルタ

クライアント ライブラリとドライバ

- クライアント ライブラリ
  - 概要
    - C++ リファレンス
    - C# リファレンス
    - Go リファレンス
    - Java リファレンス
    - Node.js リファレンス
    - PHP リファレンス
    - Python リファレンス
    - Ruby リファレンス
  - カスタム タイムアウトと再試行の構成
    - トランザクションの commit に関する統計情報を取得する
- JDBC ドライバ
- R2DBC ドライバ

Index (5) » Google » Cloud » Spanner

**Class List**  
Classes | Methods | Files  
Search:

Top Level Namespace

- Google
  - Cloud
    - Spanner**
      - Admin
      - Backup < Object
      - BatchClient < Object
      - BatchSnapshot < Object
      - BatchUpdate < Object
      - BatchUpdateError < Error
      - Client < Object
      - ClientClosedError < Error
      - ColumnValue < Object
      - Commit < Object
      - CommitResponse < Object
      - Credentials < Credentials
      - Data < Object
      - Database < Object
      - DuplicateNameError < Error
      - Fields < Object
      - Instance < Object
      - Partition < Object
      - Policy < Object
      - Project < Object

Module: Google::Cloud::Spanner  
Defined in: lib/google/cloud/spanner.rb

## Overview

### Cloud Spanner

Cloud Spanner is a fully managed, mission-critical, relational database service that offers transactional consistency at global scale, schemas, SQL (ANSI 2011 with extensions), and automatic, synchronous replication for high availability.

For more information about Cloud Spanner, read the [Cloud Spanner Documentation](#).

See [Spanner Overview](#).

### Defined Under Namespace

Modules: Admin Classes: Backup, BatchClient, BatchSnapshot, BatchUpdate, BatchUpdateError, Client, ClientClosedError, ColumnValue, Commit, CommitResponse, Credentials, Data, Database, DuplicateNameError, Fields, Instance, Partition, Policy, Project, Range, Results, Rollback, SessionLimitError, Snapshot, Status, Transaction

### Constant Summary

collapse

**VERSION** =  
"2.12.1".freeze

### Class Method Summary

collapse

`.configure { |Google::Cloud.configure.spanner| ... } => Google::Cloud::Config`  
Configure the Google Cloud Spanner library.

クライアント ライブラリごとの、クラスやメソッドの情報などはこちら。  
例えばコネクションを張るときのメソッドなど SQL で操作できる以外の処理。

# Cloud Spanner 用の ORM 一覧

Cloud Spanner 概要 ガイド

≡ フィルタ

NUMERIC アーメの探TF

JSON データの操作 🗨

クエリプラン ビジュアライザを使用したクエリのチューニング

統合

- 他の GCP サービスとの統合
- Active Record との統合**
- Django ORM との統合
- Entity Framework Core との統合
- Hibernate ORM との統合
- Liquibase との統合
- Looker との統合
- Spring Data との統合
- SQLAlchemy との統合
- Terraform との統合

▶ Key Visualizer を使用したトラブルシューティング

チュートリアル

すべてのチュートリアル

## Google Cloud によって OSS として提供

- Java - Hibernate
  - <https://github.com/GoogleCloudPlatform/google-cloud-spanner-hibernate>
- Python - Django ORM
  - <https://github.com/googleapis/python-spanner-django>
- Python - SQLAlchemy
  - <https://github.com/googleapis/python-spanner-sqlalchemy>
- Ruby - Active Record
  - <https://github.com/googleapis/ruby-spanner-activerecord>
- C# Entity Framework
  - <https://github.com/googleapis/dotnet-spanner-entity-framework>

## コミュニティによる提供

- PHP - Laravel
  - <https://github.com/colopl/laravel-spanner>

03

## Cloud Spanner と ORM

～Active Record を試してみる～

# ブログ記事: Rails から Cloud Spanner に接続するお話

デベロッパー

## Cloud Spanner での Active Record のサポートによる Ruby アプリケーションのスケールアップ

Google Cloud Japan Team  
2021年12月27日

Facebook Twitter LinkedIn Email

※この投稿は米国時間 2021 年 12 月 11 日に、Google Cloud blog に投稿されたものの抄訳です。

このたび、Google Cloud Spanner 向け Ruby Active Record Adapter が一般提供されます。Ruby Active Record は、Ruby on Rails にバンドルされている強力なオブジェクトリレーショナルマッピング (ORM) ライブラリです。Active Record は、基礎となるデータベースを抽象化し、スキーマの変更を自動生成する機能やスキーマのバージョン履歴を管理する機能などを提供します。

Active Record は Rails プロジェクトで使用されるのが一般的ですが、Sinatra のような他のフレームワークで使用したり、Ruby アプリケーションのスタンドアロンライブラリとして使用することもできます。このアダプタの一般提供により、Ruby アプリケーションでも、ORM を介して Cloud Spanner の高可用性と外部整合性を大規模に利用できるようになりました。

このアダプタは、Ruby の gem として `ruby-spanner-activerecord` という名前でリリースされています。現在サポートされているオプションは次のとおりです。

- ActiveRecord 6.0.x と Ruby 2.6 および 2.7 の組み合わせ。
- ActiveRecord 6.1.x と Ruby 2.6 以上の組み合わせ。

今回の投稿では、Rails アプリケーションでこのアダプタを使用する方法を説明し、サポートされている機能をご紹介します。

### インストール

Cloud Spanner データベースで Active Record を使用するには、Cloud Spanner API を有効にしたアクティブな Google Cloud プロジェクトが必要です。Cloud Spanner の開始方法の詳細については、Cloud Spanner スタートガイドをご覧ください。

- 以降の手順は、公式ブログ記事に掲載された「Cloud Spanner での Active Record のサポートによる Ruby アプリケーションのスケールアップ」内の手順を、一部補足修正しながら紹介
- 基本的にはブログにかかれているコマンドやコンフィグからコピーして試すことが可能

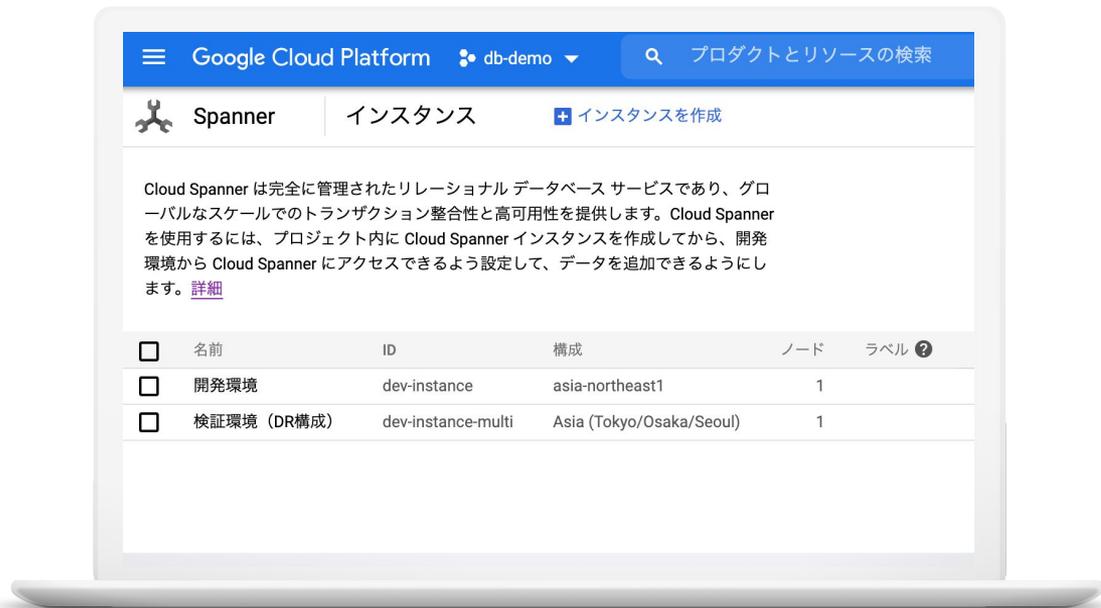
### テーブルを持つデータベースの作成

まず、以下のコマンドを実行してデータベースを作成します。

```
01 bin/rails db:create
```

ブログ内に書かれたコマンド例

# まずは Cloud Spanner インスタンスの作成



# 開発用の 0.1 ノード インスタンスを立てる

← インスタンスの作成

インスタンスの名前を指定

インスタンスには名前と ID の両方があります。名前は表示専用です。ID は変更できない一意の識別子です。

インスタンス名 \*  
test-instance

名前は 4~30 文字で指定してください

インスタンス ID \*  
test-instance

小文字、数字、ハイフンのみ使用できます

構成を選択

ノードとデータの配置を決定します。費用、パフォーマンス、レプリケーションに影響します。マルチリージョン構成では、リーダー レプリカのデフォルトのリーダー リージョンが選択されます。リーダー リージョンは、DDL ステートメントによっていつでも変更できます。 [詳細](#)

COMPARE REGION CONFIGURATIONS

リージョン  
 マルチリージョン

asia-northeast1 (東京)

コンピューティング容量の割り当て

コンピューティング容量によって、インスタンスのデータ スループット、秒間クエリ数 (QPS)、ストレージの制限が決まります。1 ノードは 1,000 処理ユニットに相当します。この選択によって費用は変動します。

単位 \*  
処理ユニット

数量 \*  
100

整数のみ。1,000 に達するまでは 100 単位で入力し、それ以降は 1,000 単位で入力します。

## 今回の Cloud Spanner 環境は以下の通り

- インスタンス名: test-instance
- 構成: 東京リージョン (他のリージョンも可)
- コンピューティング容量: 100 処理ユニット

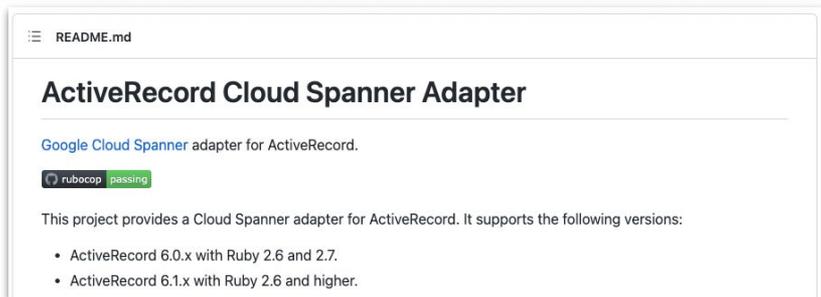
## Cloud Spanner インスタンスの料金

- Cloud Spanner は現在最小単位が 0.1 ノード (100 処理ユニット) から利用可能
  - 東京リージョン: \$87 / 月
  - 台湾リージョン: \$65 / 月

ブログの手順では、`gcloud spanner instances create` コマンドでインスタンスを作っているが、ここでは GUI から行っている

# 今回の Rails アプリケーションの作成

```
-zsh ㊦%1
~ >>> rails _6.1.4.6_ new my_spanner_app
create
create  README.md
create  Rakefile
create  .ruby-version
create  config.ru
create  .gitignore
create  .gitattributes
create  Gemfile
```



## 今回の環境の前提条件

- 今回の例では、手元のマシン上で Rails を実行し、Cloud Spanner に接続する場合を想定している
- あらかじめ Ruby と Rails はインストール状態であることとする
- 今回は macOS Monterey にて実行

## rails new コマンドの実行

- 現時点で ruby-spanner-activerecord は Rails 6 系までのサポート
- `rails _6.1.4.6_ new` にて、6 系を指定

# Gemfile の編集と bundle install の実行

```
-zsh ㊟%1
# Adds support for Capybara system testing and selenium driver
gem 'capybara', '>= 3.26'
gem 'selenium-webdriver'
# Easy installation and use of web drivers to run system tests with browsers
gem 'webdrivers'
end

# Windows does not include zoneinfo files, so bundle the tzinfo-data gem
gem 'tzinfo-data', platforms: [:mingw, :mswin, :x64_mingw, :jruby]

# ActiveRecord Cloud Spanner Adapter
gem 'activerecord-spanner-adapter'
```

```
-zsh ㊟%1
~/my_spanner_app >>> bundle install
Fetching gem metadata from https://rubygems.org/.....
Resolving dependencies...
Using rake 13.0.6
Using concurrent-ruby 1.1.9
Using zeitwerk 2.5.4
Using marcel 1.0.2
Using mini_mime 1.1.2
Using faraday-em_http 1.0.0
```

## Gemfile の編集

- 作成したアプリケーションのディレクトリに行き、Gemfile を編集する
- `gem 'activerecord-spanner-adapter'` を追加する

## bundle install の実行

- `bundle install` 実行し、Gemfile から gem をインストールする。

注意) Apple M1 の macOS 環境で試す場合、Intel 用のバイナリをインストールしてしまうことを防ぐため、`bundle install` 前に、`bundle config set force_ruby_platform true` を実行すること

# Cloud Spanner 接続情報の設定

```
-zsh ㊦%1
default: &default
  adapter: "spanner"
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 10 } %>
  project: gcpx-db-demo
  instance: test-instance

development:
  <<: *default
  database: blog_dev

production:
  <<: *default
  database: blog

~
"config/database.yml" 13L, 237B
```

## config/database.yml の編集

- デフォルトで書かれているものを消して、Cloud Spanner 用のものに置き換える

## project:

- 今回利用している Project ID を入力

## credentials: (今回不使用)

- ブログの記事ではサービスアカウントキーによる例を出しているが、今回はアプリケーションのデフォルト認証情報(ADC)を利用することとするため、この項目は使わない



# アプリケーションのデフォルト認証情報(ADC)の設定

Google Auth Library が Google アカウントへのアクセスをリクエストしています

 takasato@

Google Auth Library に以下を許可します:

- Google Cloud のデータの参照、編集、設定、削除、Google アカウントのメールアドレスの参照。

Google Auth Library を信頼できることを確認

お客様の機密情報をこのサイトやアプリと共有することがあります。アクセス権の確認、削除は、[Google アカウント](#)でいつでも行えます。

Google でデータ共有を安全に行う方法についての説明をご覧ください。

Google Auth Library のプライバシーポリシーと利用規約をご覧ください。

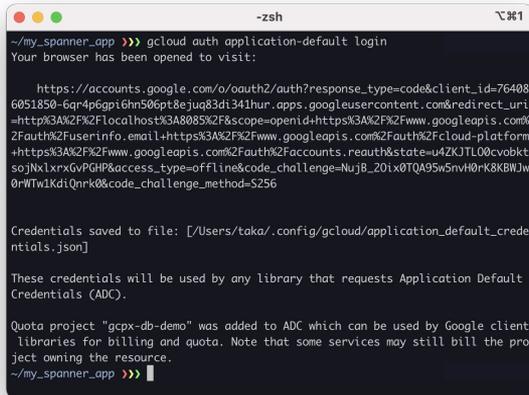
キャンセル

許可

## ADC の設定

- ターミナル上で `gcloud auth application-default login` を実行するとブラウザに飛ぶので、許可をクリックする
- これでローカルのアプリケーションは、必要な認証情報を自動検出し、自動で認証を行うことができる
- あわせて、不要な WARNING 出力を抑制するため、以下の環境変数をセットしてから、以降の手順へすすむ

```
export GOOGLE_AUTH_SUPPRESS_CREDENTIALS_WARNINGS=1
```



```
~/my_spanner_app >>> gcloud auth application-default login
Your browser has been opened to visit:

  https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=764086051850-6ar4p6p16hn506pt8ejuq83di341hur_apps.googleusercontent.com&redirect_uri=https%3A%2F%2Flocalhost%3A8085%2F&scope=openid%2F%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.email%2F%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform%2F%3A%2F%2Fwww.googleapis.com%2Fauth%2Faccounts_reauth&state=u4ZKJTL00evobktsojNxlxxvGvPGHP&access_type=offline&code_challenge=NujB_20iX0TQ95w5nvH0rK8KBWJw0rWTw1KdiQnrk0&code_challenge_method=S256

Credentials saved to file: [/Users/taka/.config/gcloud/application_default_credentials.json]

These credentials will be used by any library that requests Application Default Credentials (ADC).

Quota project "gcp-x-db-demo" was added to ADC which can be used by Google client libraries for billing and quota. Note that some services may still bill the project owning the resource.

~/my_spanner_app >>> █
```

参考ドキュメント:

[Google Cloud でアプリケーションを安全に認証するためのベストプラクティス](#)  
<https://cloud.google.com/docs/authentication/best-practices-applications>

# db:create によるデータベースの作成

```
-zsh ㉿%1
~/my_spanner_app >>> bin/rails db:create
Running via Spring preloader in process 59725
Created database 'blog_dev'
```

## db:create によるデータベースの作成

- `bin/rails db:create` を実行する
- 成功すると `blog_dev` という名前でデータベースが作成されたことが確認できる

Spanner

すべてのインスタンス > インスタンス test-instance: 概要

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

概要

名前	test-instance
ID	test-instance
構成	asia-northeast1

コンピューティング容量 ⓘ  
処理ユニット: 100  
ノード: 0

CPU 使用率 (平均)  
-

オペレーション  
読み取り: 0.00/秒  
書き込み: -/秒

データベース + データベースを作成

フィルタ データベースをフィルタ

<input type="checkbox"/>	名前 ↑	CPU 使用率	サイズ	バージョンの保持期間 ⓘ
<input type="checkbox"/>	blog_dev	-	-	1 時間

# データモデルの作成とテーブルの作成

```
-zsh ㄟ%1
~/my_spanner_app >>> bin/rails generate model Article title:string body:text
Running via Spring preloader in process 59787
  invoke  active_record
  create   db/migrate/20220222063001_create_articles.rb
  create   app/models/article.rb
  invoke   test_unit
  create   test/models/article_test.rb
  create   test/fixtures/articles.yml
```

```
-zsh ㄟ%1
~/my_spanner_app >>> bin/rails db:migrate
Running via Spring preloader in process 61576
== 20220222063001 CreateArticles: migrating =====
-- create_table(:articles)
   -> 24.8298s
== 20220222063001 CreateArticles: migrated (24.8300s) =====
```

## データモデルを作成

- `bin/rails generate model Article title:string body:text`により、title と body を持ったモデルが作成される

## データモデルを元に、テーブルを作成

- `bin/rails db:migrate` を実行すると、作成したモデルに応じたテーブルが Cloud Spanner 上に作成される

スキーマ

名前: articles  
スキーマの更新: 最近の更新はありません

主キー: id (昇順)

列	種類	Nullable	注文
id	INT64	x	昇順
title	STRING(MAX)	○	-
body	STRING(MAX)	○	-
created_at	TIMESTAMP	x	-
updated_at	TIMESTAMP	x	-

## 同等な DDL

これは、このテーブルのデータベース定義言語 (DDL) ステートメントです。

```
CREATE TABLE articles (
  id INT64 NOT NULL,
  title STRING(MAX),
  body STRING(MAX),
  created_at TIMESTAMP NOT NULL,
  updated_at TIMESTAMP NOT NULL,
) PRIMARY KEY(id);
```

# rails console を用いて対話的にデータの読み書きをする

```
-zsh ㉿%1
~/my_spanner_app >>> bin/rails console
Running via Spring preloader in process 61666
Loading development environment (Rails 6.1.4.6)
irb(main):001:0> Article.all
Article Load (28.8ms) SELECT `articles`.* FROM `articles` /* loading for inspect */ LIMIT @p1
=> #<ActiveRecord::Relation []>
irb(main):002:0>
```

## rails console の起動

- `bin/rails console` を実行すると、REPL が起動し、Rails を用いて対話的に Cloud Spanner を読み書きできるようになる

```
-zsh ㉿%1
irb(main):002:0> article = Article.new(title: "Hello Rails", body: "I am on Rails!")
=> #<Article id: nil, title: "Hello Rails", body: "I am on Rails!", created_at: nil>
irb(main):003:0> article.save
SQL (0.1ms) BEGIN
Article Create (32.8ms) INSERT INTO `articles` (`title`, `body`, `created_at`, `updated_at`, `id`) VALUES (@p1, @p2, @p3, @p4, @p5)
SQL (16.8ms) COMMIT
=> true
irb(main):004:0> Article.all
Article Load (26.2ms) SELECT `articles`.* FROM `articles` /* loading for inspect */ LIMIT @p1
=> #<ActiveRecord::Relation [#<Article id: 2618879713266101224, title: "Hello Rails", body: "I am on Rails!", created_at: "2022-02-22 07:06:10.406552000 +0000", updated_at: "2022-02-22 07:06:10.406552000 +0000">]>
irb(main):005:0>
```

## article テーブルを読み書き

- `Article.all` で `SELECT * FROM article` が実行される
- `article` に対して `Article.new` で新しいデータを作成して格納し、`article.save` で保存、すなわち実際にデータベースに対して更新トランザクションが実行される
- `INSERT` が実行されている様子が見える
- その後再度 `Article.all` すると、データが表示される

## ブログ手順: 既存データベースの移行(スキーママイグレーション)

# 既存テーブルに対して新たな列を追加する

```
-zsh ㊦%1
~/my_spanner_app >>> bin/rails generate migration AddCommentAndRateToArticles co
mment:text rating:integer
Running via Spring preloader in process 61779
  invoke  active_record
  create   db/migrate/20220222071035_add_comment_and_rate_to_articles.rb
~/my_spanner_app >>> bin/rails db:migrate
Running via Spring preloader in process 61829
== 20220222071035 AddCommentAndRateToArticles: migrating =====
-- add_column(:articles, :comment, :text)
-> 25.6492s
-- add_column(:articles, :rating, :integer)
-> 24.6286s
== 20220222071035 AddCommentAndRateToArticles: migrated (50.2783s) =====
```

### 追加列の定義

- `bin/rails generate migration AddCommentAndRateToArticles comment:text rating:integer` を実行すると、新たな列として **comment** 列と **rating** 列を追加する

### 追加列のスキーママイグレーション

- `bin/rails db:migrate` の実行で、新たに追加した列が、実際に Cloud Spanner に反映される

スキーマ

名前 articles

スキーマの更新 最近の更新はありません

主キー: id (昇順)

列	種類	Nullable	注文
id	INT64	x	昇順
title	STRING(MAX)	○	-
body	STRING(MAX)	○	-
created_at	TIMESTAMP	x	-
updated_at	TIMESTAMP	x	-
comment	STRING(MAX)	○	-
rating	INT64	○	-



# 詳しい情報は各種 ORM の GitHub リポジトリを参照

## ruby-spanner-activerecord の詳細ドキュメント

- 詳細な使い方は、ruby-spanner-activerecord の GitHub リポジトリにドキュメントがある

### Examples

To get started with Rails, read the tutorial under [examples/rails/README.md](#).

You can also find a list of short self-contained code examples that show how to use ActiveRecord with Cloud Spanner under the directory [examples/snippets](#). Each example is directly runnable without the need to setup a Cloud Spanner database, as all samples will automatically start a Cloud Spanner emulator in a Docker container and execute the sample code against that emulator. All samples can be executed by navigating to the sample directory on your local machine and then executing the command `bundle exec rake run`. Example:

```
cd ruby-spanner-activerecord/examples/snippets/quickstart
bundle exec rake run
```

NOTE: You do need to have [Docker](#) installed on your local system to run these examples.

Some noteworthy examples in the snippets directory:

- **quickstart**: A simple application that shows how to create and query a simple database containing two tables.
- **migrations**: Shows a best-practice for executing migrations on Cloud Spanner.
- **read-write-transactions**: Shows how to execute transactions on Cloud Spanner.
- **read-only-transactions**: Shows how to execute read-only transactions on Cloud Spanner.
- **bulk-insert**: Shows the best way to insert a large number of new records.
- **mutations**: Shows how you can use **mutations** instead of DML for inserting, updating and deleting data in a Cloud Spanner database. Mutations can have a significant performance advantage compared to DML statements, but do not allow read-your-writes semantics during a transaction.
- **array-data-type**: Shows how to work with `ARRAY` data types.

Active Record を使った  
サンプルコード

```
39 Lines (34 sloc) | 1.34 KB
1 # Copyright 2021 Google LLC
2 #
3 # Use of this source code is governed by an MIT-style
4 # license that can be found in the LICENSE file or at
5 # https://opensource.org/licenses/MIT.
6
7 require "io/console"
8 require_relative "../config/environment"
9 require_relative "models/singer"
10 require_relative "models/album"
11
12 class Application
13   def self.run # rubocop:disable Metrics/AbcSize
14     from_album = nil
15     to_album = nil
16     # Use a read/write transaction to execute multiple statements as an atomic unit.
17     ActiveRecord::Base.transaction do
18       # Transfer a marketing budget of 10,000 from one album to another.
19       from_album = Album.all.sample
20       to_album = Album.where.not(id: from_album.id).sample
21
22       puts ""
23       puts "Transferring 10,000 marketing budget from #{from_album.title} (#{from_album.marketing_budget}) "\
24         "to #{to_album.title} (#{to_album.marketing_budget})"
25       from_album.update marketing_budget: from_album.marketing_budget - 10000
26       to_album.update marketing_budget: to_album.marketing_budget + 10000
27     end
28     puts ""
29     puts "Budgets after update:"
30     puts "Marketing budget #{from_album.title}: #{from_album.reload.marketing_budget}"
31     puts "Marketing budget #{to_album.title}: #{to_album.reload.marketing_budget}"
32
33     puts ""
34     puts "Press any key to end the application"
35     STDIN.getch
36   end
37 end
38
39 Application.run
```

<https://github.com/googleapis/ruby-spanner-activerecord>

04

# PostgreSQL Interface

# Cloud Spanner PostgreSQL Interface

プレビュー

## Spanner PostgreSQL のインターフェース

### 使い慣れたスキル及びツールを利用可能

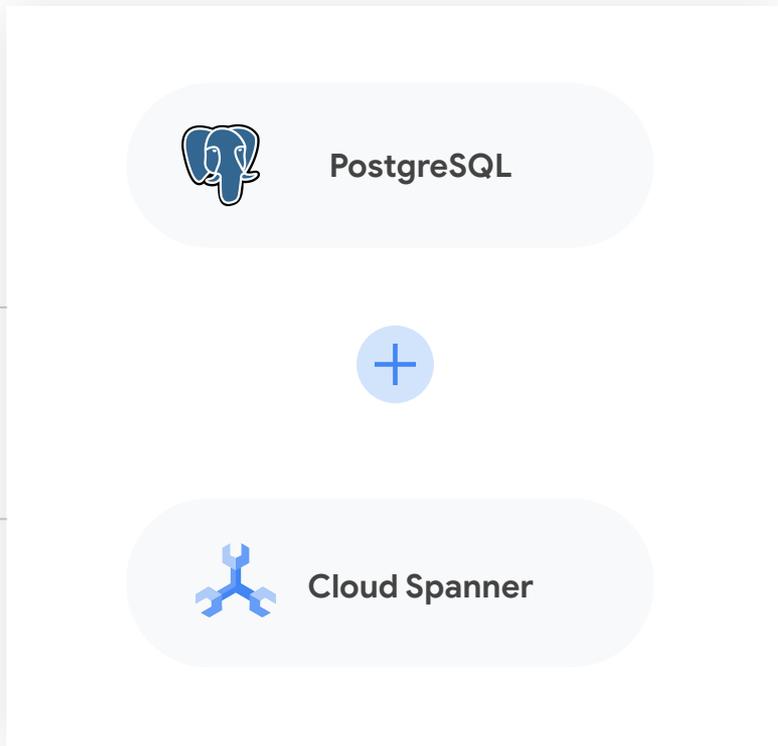
これまで培ったスキルやツールを利用しながら、Spanner が持つスケーラビリティ、99.999% の可用性を享受可能

### アプリケーションのポータビリティ向上

他の PostgreSQL でも実行可能な SQL なので、別の PostgreSQL 環境へも移植可能

### 開発効率の向上

既存の PostgreSQL リソース及びナレッジを活用することで、学習コストを削減



# Spanner Capabilities, Postgres “API”

- プレビュー時点では、Cloud Spanner の基本機能を PostgreSQL ツールを通じて利用可能
- Cloud Spanner が持つスケラビリティ、一貫性、パフォーマンス、運用性をそのまま利用可能
- PostgreSQL エンジニア、開発者にとって Cloud Spanner がより扱いやすく  
(PostgreSQL との 100% の互換性がゴールではない)



# PostgreSQL との Compatibility

## Cloud Spanner の PostgreSQL インターフェース

Preview



Cloud Spanner

Interleaved tables

Spanner Clients

- JDBC
- Java
- Go
- Python

External consistency

Optimizer, query plans

Statistics

Query hints    Provisioning  
Monitoring

Coming soon: *TTL, import/export*

DQL: SELECT ...FROM  
DML: INSERT INTO...  
UPDATE...SET  
DELETE FROM  
DDL: CREATE TABLE...

Functions  
Operators

INFORMATION\_SCHEMA

Coming soon: *JSON, Views, Default values, ARRAY*



PostgreSQL

Data types

- TEXT, VARCHAR
- NUMERIC
- BIGINT
- TIMESTAMP
- FLOAT
- DOUBLE
- BOOL
- BYTEA

psql

Ecosystem clients  
Stored Procedures

Triggers

SERIAL

Privileges

Concurrency control

Sequences

Nested transactions

Transactional DDL

Partial indexes

Extensions

Foreign data wrappers

Google Cloud

# Cloud SQL との使い分けの観点(一例)



Cloud SQL

*Maximum community  
compatibility*

- ✓ PostgreSQL コミュニティバージョンへの完全な準拠
- ✓ リードレプリカによる参照の水平スケール
- ✓ バージョン、拡張、フラグの選択
- ✓ DMS によるシンプルなマイグレーション
- ✓ 99.95% の可用性



Cloud Spanner  
PostgreSQL Interface

*Preview*

*Unlimited global scale and  
99.999% availability*

- ✓ PostgreSQL インターフェース
- ✓ 無制限のスケラビリティ
- ✓ メンテナンス時もゼロダウンタイム
- ✓ 強整合性
- ✓ 99.999% の可用性

- リフト&シフト、早急な移行
- シンプルなコスト削減

- モダナイズ
- よりチャレンジングな要件への対応

# ユーザーエクスペリエンス

PostgreSQL クエリを Cloud Spanner インターフェースへ実行 + PostgreSQL プロトコル

- Cloud Spanner のデータベース作成時に SQL 方言を指定
- PostgreSQL の SQL を PostgreSQL プロトコルで実行可能
  - PostgreSQL コミュニティのツールを利用  
(現時点では **psql** をサポート)
- Cloud Spanner のエンドポイントへ直接 PostgreSQL の SQL を実行
  - オープンソースのドライバを利用  
(現時点では **JDBC, Go, Python** をサポート)
  - gcloud CLI
  - Cloud Console UI
- リソースの作成、モニタリングは既存の Cloud Spanner と同様

Name your database

Enter a permanent name for your database of at least two characters, starting with a letter.

Database name \*

Tradestore\_PROD

Lowercase letters, numbers, hyphens, underscores allowed



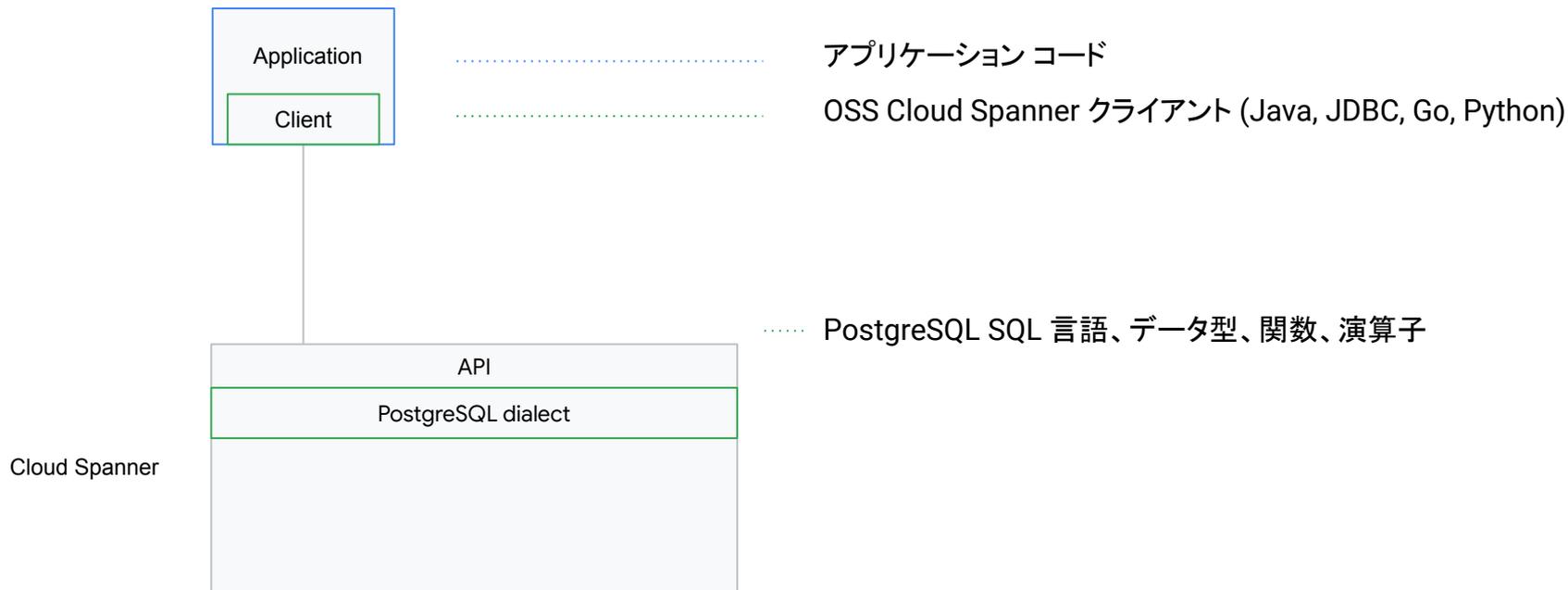
Google Standard SQL



PostgreSQL

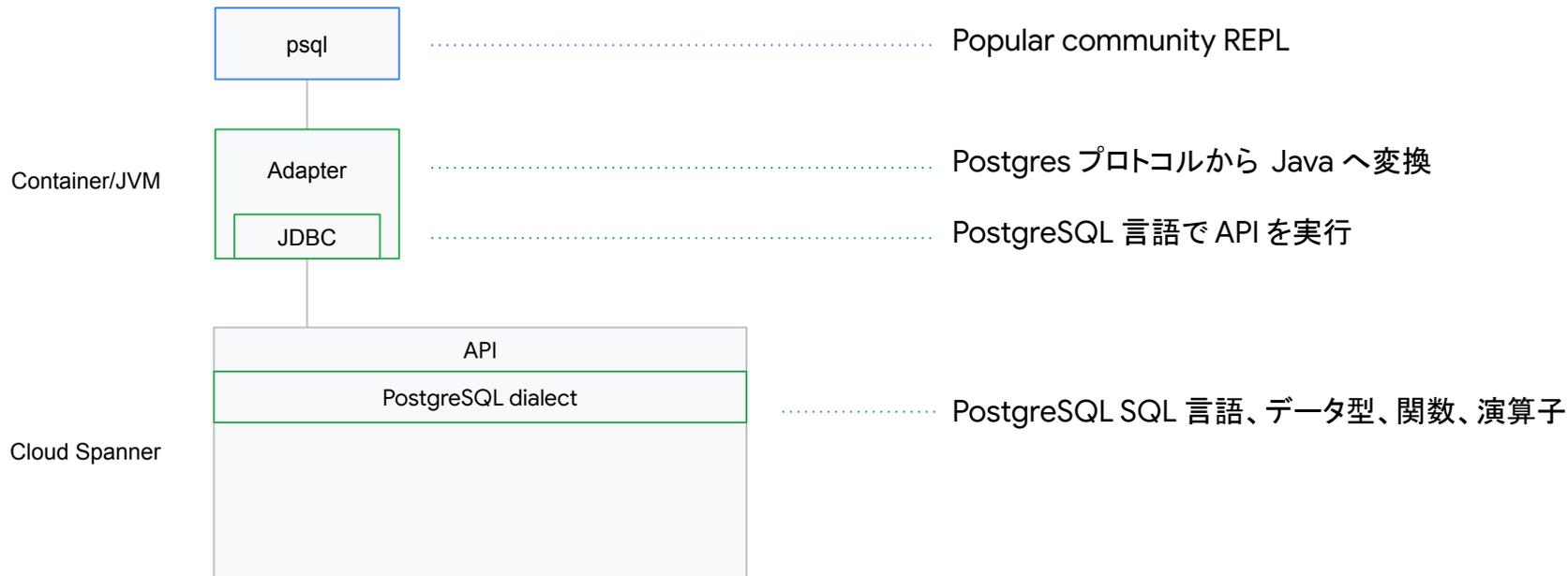
# ユーザーエクスペリエンス: Application Development

既存の Cloud Spanner clients を活用し、直接接続



# ユーザーエクスペリエンス: Ecosystem Tools

PGAdapter 経由での PostgreSQL プロトコルによる接続



05

**Cloud Spanner**

**開発者向け重要アップデート**

# BigQuery からの連携クエリによるリアルタイム分析

## 特徴

- Cloud Spanner へ BigQuery から直接データを取得することで、リアルタイムに分析操作が可能に
- BigQuery 側で外部データソースの設定を行い、EXTERNAL\_QUERY 関数を使用
- リージョンやデータ型の対応関係はドキュメントを参照
- パフォーマンス、Cloud Spanner に与える負荷などは検証の上での利用を推奨

### 外部データソース

接続タイプ  
Cloud Spanner

接続 ID \*  
sales-db

データのロケーション  
asia-northeast1 (東京)

わかりやすい名前  
Sales DB Prod

説明  
売り上げ用のトランザクションデータ

Database name \*  
projects/demo/instances/spanner/databases/sales-db

データを同時に読み込む

# View をサポート

## 特徴

- View を利用することで、開発者に見せる  
カラムの制限、複雑なクエリの簡素化が可能に
- View 作成時にデータ型を明示的に  
キャストすることで、スキーマ変更の  
影響を減らすことが可能(元テーブルの  
データ型を変更してもView に影響がない  
ようにすることができる)
- INFORMATION\_SCHEMA.VIEWS  
テーブルで、定義済みのビューを確認可能

```
CREATE OR REPLACE VIEW SingerNames
SQL SECURITY INVOKER
AS SELECT
  CAST(Singers.SingerId AS INT64) AS SingerId,
  CAST(Singers.FirstName AS STRING) || " " ||
  CAST(Singers.LastName AS STRING) AS Name
FROM Singers;
```

# JSON 型をサポート

## 特徴

- JSON をデータ型として利用可能に
- RDBMS の特性を享受しつつ、スキーマレスなデータ構造を利用可能
- プライマリーキーの指定やインデックスの登録はできない
  - JSON 列からスカラー値を抽出して生成した列はインデックスを付与可能
- パフォーマンスなどは検証の上、利用を推奨

```
ALTER TABLE blog_metadata ADD COLUMN tags JSON;
```

# Granular instance sizing でよりきめ細やかな リソース割当が可能に

## 特徴

- 「Processing Unit (処理ユニット)」という単位でコンピューティング容量を指定可能
  - 従来の 1/10 からリソースの割当が可能 (コストも 1/10 から利用可能)
  - 1 ノード = 1,000 Processing Unit(PU)
- 想定する用途
  - 開発環境、テスト環境
  - (初期費用を抑えたい) 新規サービス、マイクロサービス単位の DB
- 1 ノード以下の利用では上限も小さくなる点に注意

### コンピューティング容量の割り当て

コンピューティング容量によって、インスタンスのデータ スループット、秒間クエリ数 (QPS)、ストレージの制限が決まります。1 ノードは 1,000 処理ユニットに相当します。この選択によって費用は変動します。

単位 \*  
処理ユニット

数量 \*  
100

整数のみ。1,000 に達するまでは 100 単位で入力し、それ以降は 1,000 単位で入力します。



1,000 PU 未満のインスタンスは公開プレビュー中であり、現在 Spanner SLA の対象ではありません。

0.1 node  
100 PU

1 node  
1,000 PU

2 node  
2,000 PU

.....



**Thank you.**