Google Cloud

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 ～

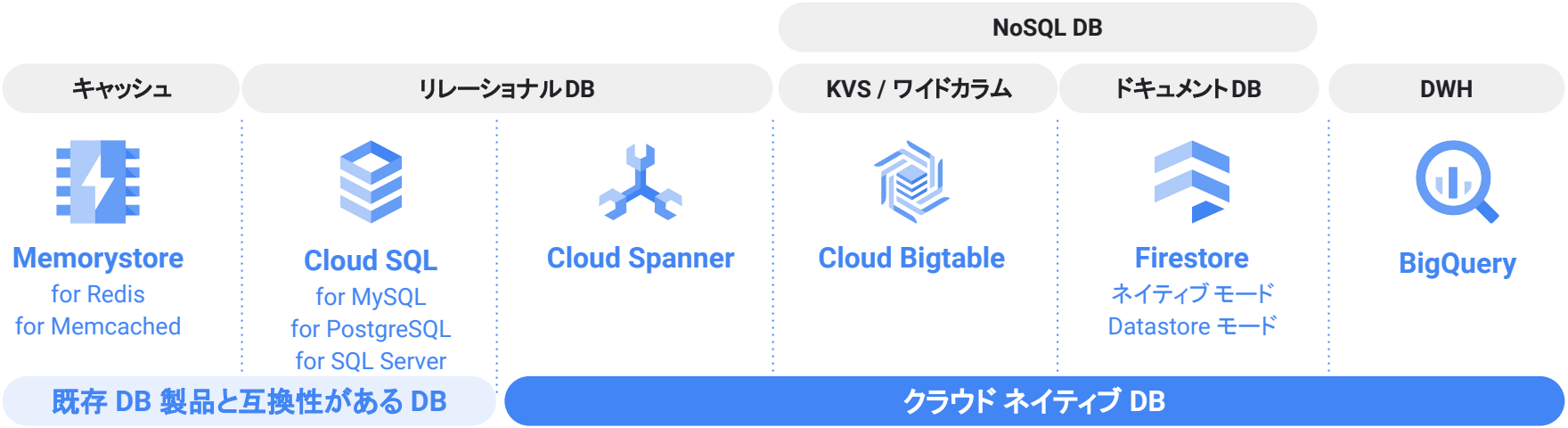
Google Cloud
データベース スペシャリスト
佐藤 貴彦

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

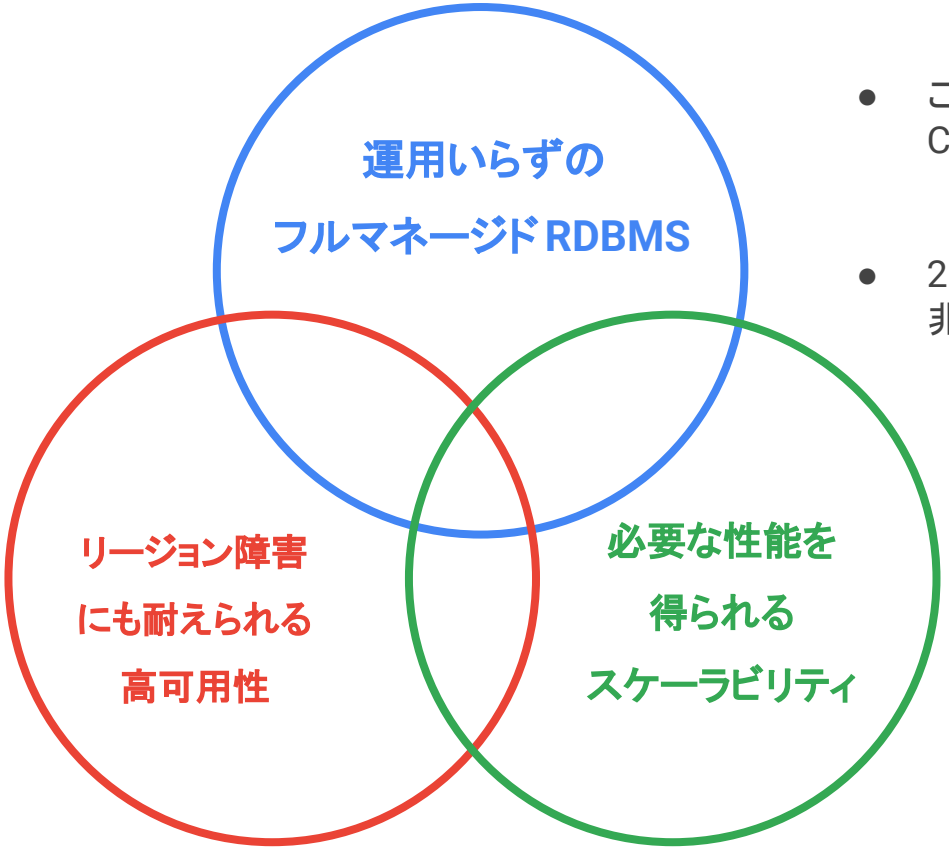
メンテナンスやノード数変更時のダウンタイムなども無し

特徴 3 - 必要な性能を容易に得られるスケーラビリティ

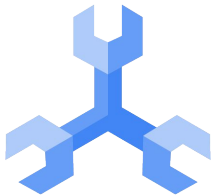
必要な性能に合わせて、無停止でスケールアウトやスケールインが可能

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



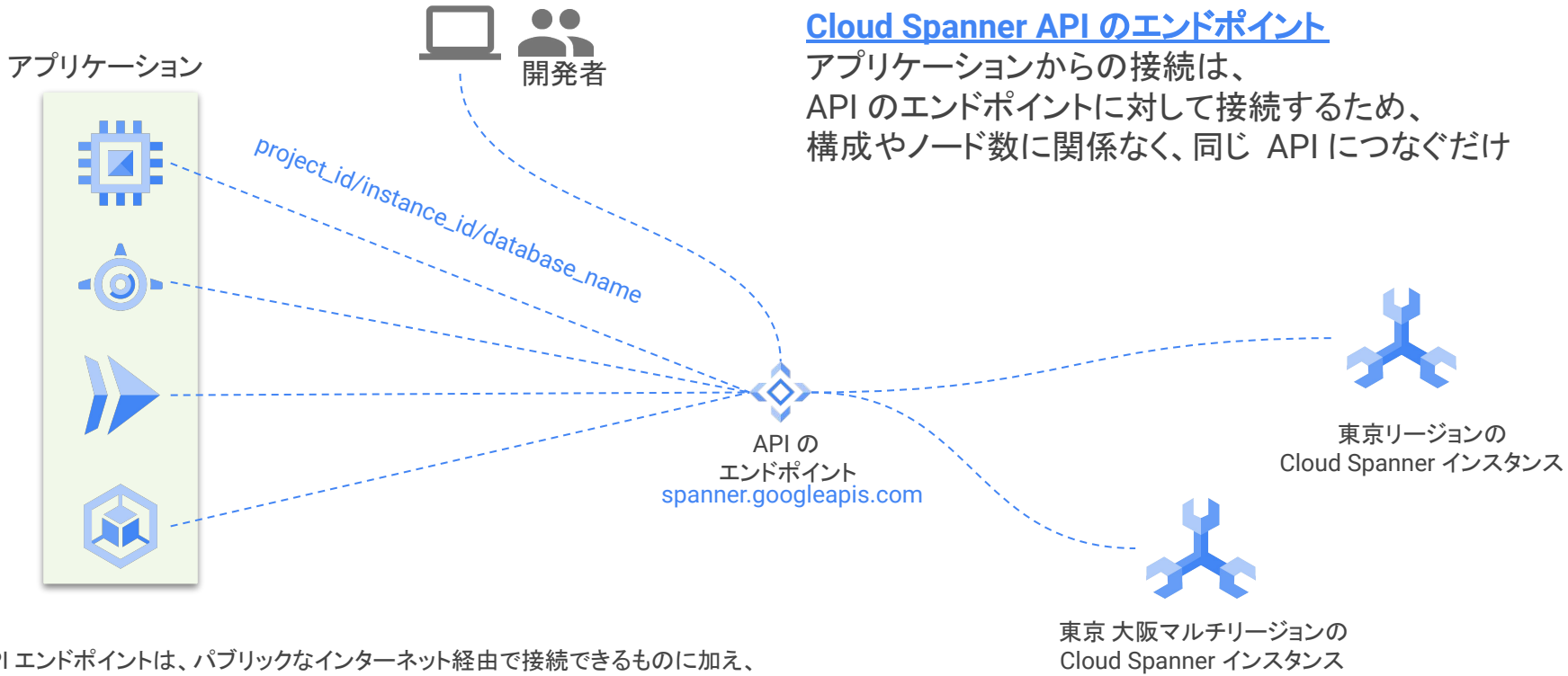
- このうちの 1 つでも必要なシステムなら Cloud Spanner を検討
  - 例) ダウンタイムを無くすために利用
- 2 つ以上同時に必要なら、Cloud Spanner は非常によい選択肢
  - 例) 運用いらずで高可用性が欲しい
  - 例) 運用いらずでスケーラビリティが欲しい



02

アプリケーションから Cloud Spanner への接続方式

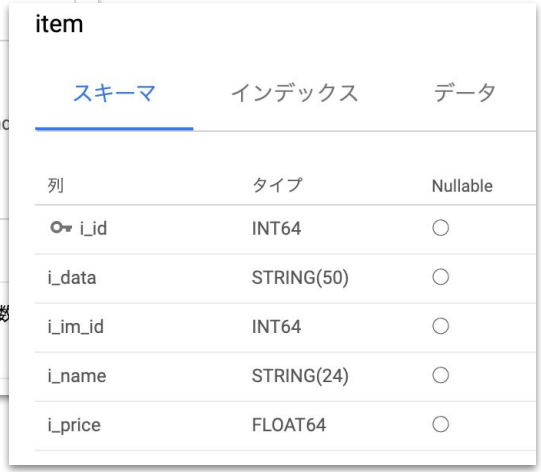
アプリケーションから API に接続して利用する RDBMS



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。

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。

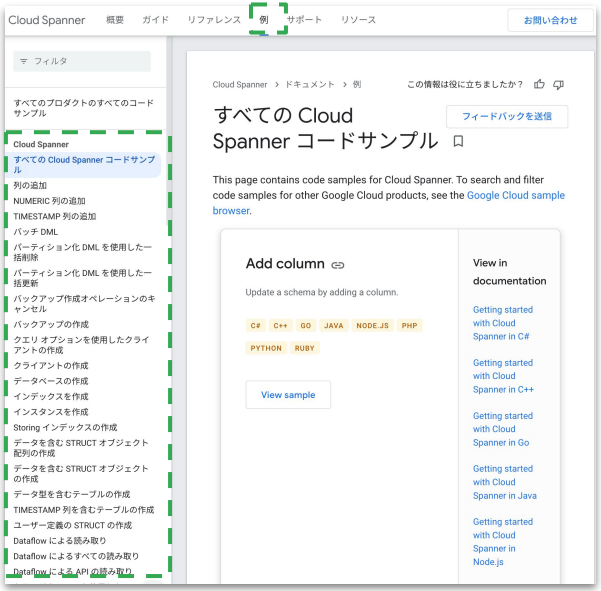
```
public int updateRecordUsingJDBC (int id, long col1)
throws SQLException {
  PreparedStatement ps = connection.prepareStatement (
          "UPDATE table01 SET col1 = ? WHERE id = ?");
  ps.setLong (1, col1);
  ps.setLong (2, id);
  ps.executeUpdate ();
```

Java から JDBC 経由で接続している例

アプリケーションを書くためのドキュメントの場所

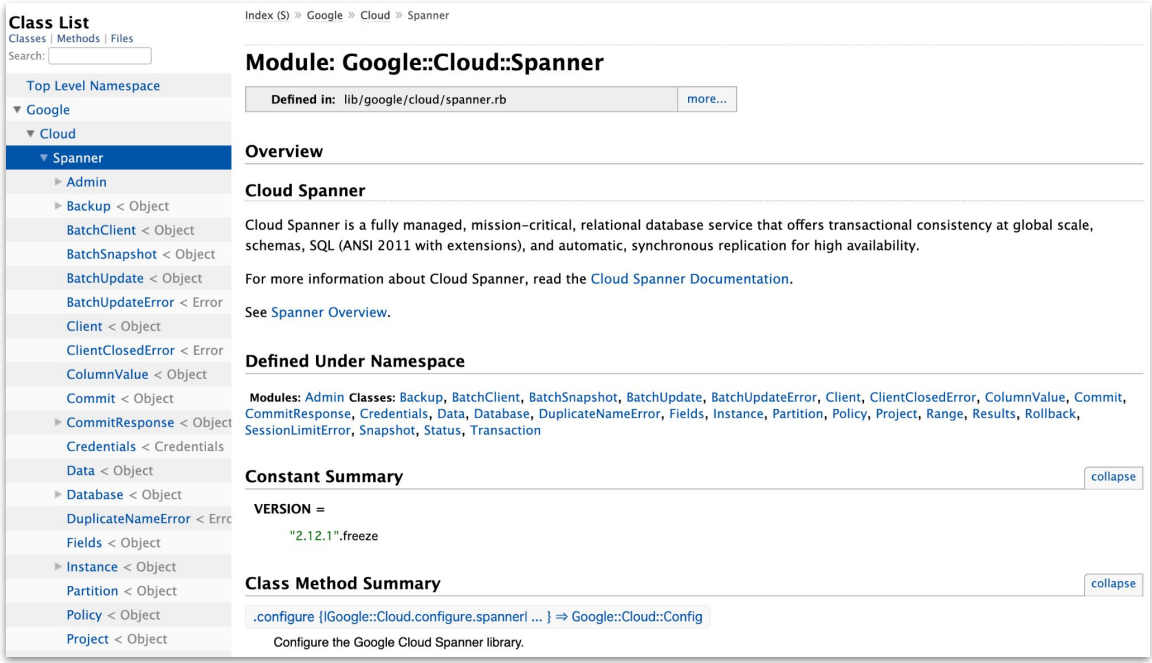
各種言語のクライアント ライブラリを使った開発のチュートリアルとサンプルコード

クライアント ライブラリとドライバのドキュメントへのリンク一覧



応用的な操作のコードサンプル一覧

Cloud Spanner のクライアントのドキュメント



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

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リング

Google Cloud Japan Team

2021年12月27日

※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、基盤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リング」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。

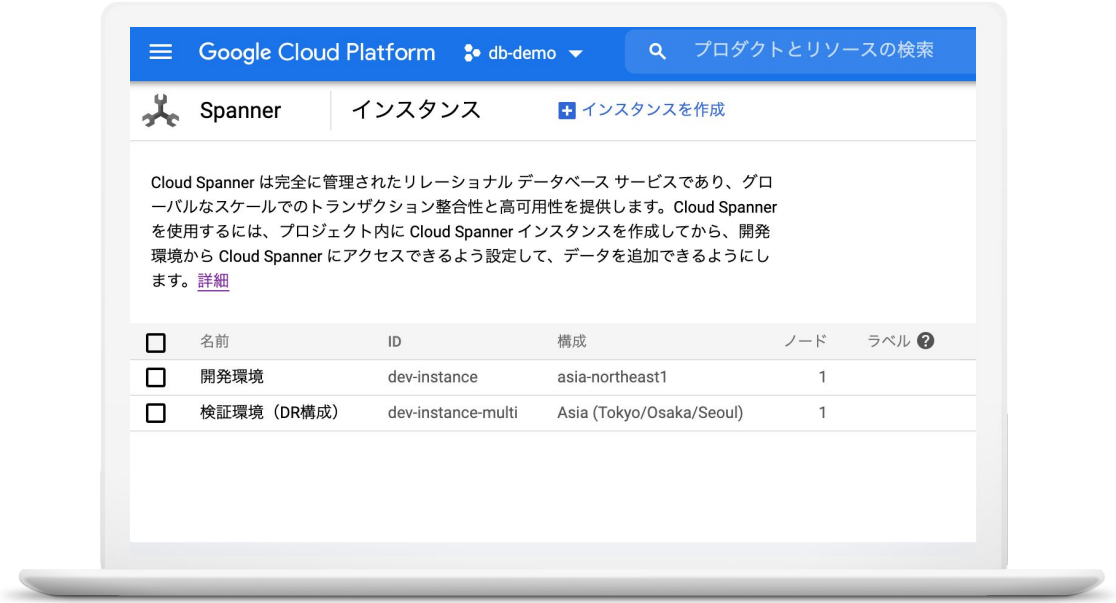
```
01  bin/rails db:create
```

ブログ内に書かれたコマンド例

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リング| Google Cloud Blog
https://cloud.google.com/blog/ja/topics/developers-practitioners/scale-your-ruby-applications-active-record-support-cloud-spanner

ブログ手順：インストール

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

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

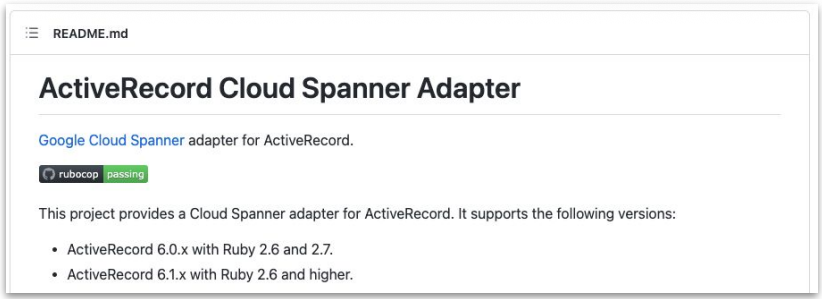
Cloud Spanner インスタンスの料金

- 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
```


README.md

ActiveRecord Cloud Spanner Adapter

Google Cloud Spanner adapter for ActiveRecord.

rubocop passing

This project provides a Cloud Spanner adapter for ActiveRecord. It supports the following versions:

- ActiveRecord 6.0.x with Ruby 2.6 and 2.7.
- ActiveRecord 6.1.x with Ruby 2.6 and higher.

今回の環境の前提条件

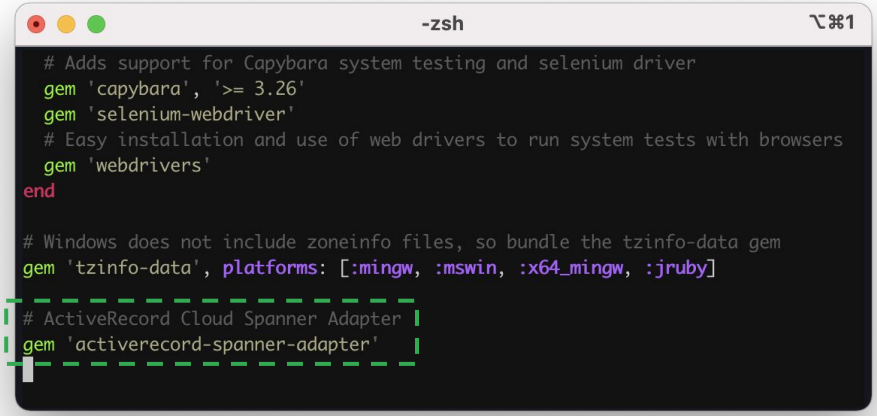
- 今回の例では、手元のマシン上で Rails を実行し、Cloud Spanner に接続する場合を想定している
- あらかじめ Ruby と Rails はインストール状態であることとする
- 今回は macOS Monterey にて実行

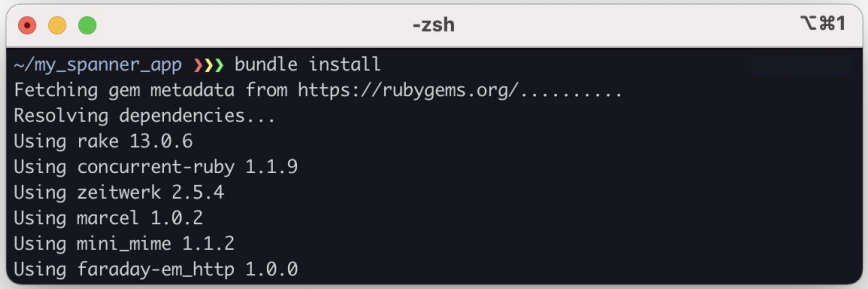
rails new コマンドの実行

- 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

GitHub - googleapis/ruby-spanner-activerecord
https://github.com/googleapis/ruby-spanner-activerecord

Gemfile の編集と bundle install の実行





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

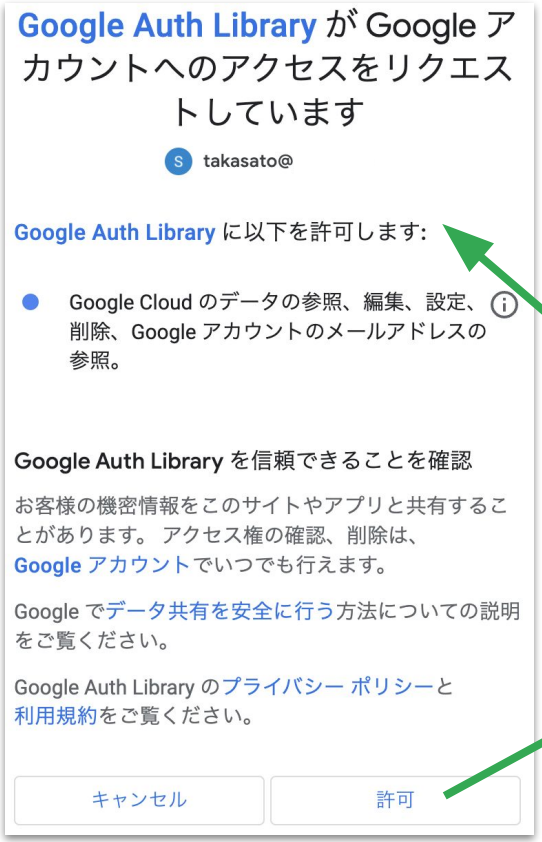
project:

- 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

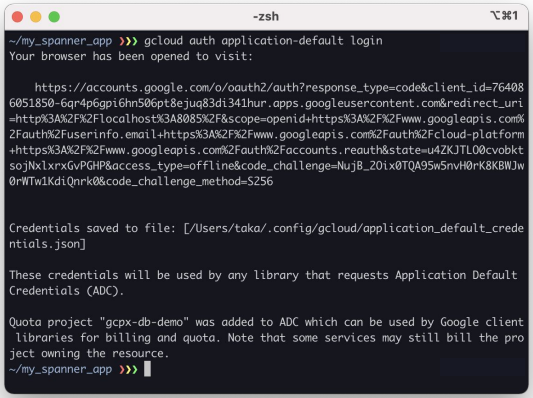
アプリケーションのデフォルト認証情報(ADC)の設定



ADC の設定

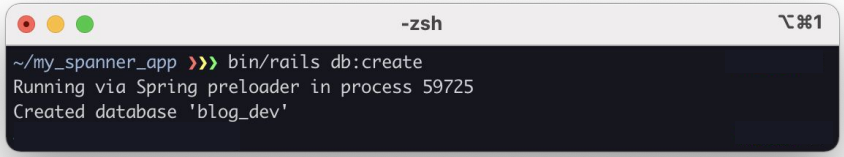
- ターミナル上で gcloud auth application-default login を実行するとブラウザに飛ぶので、許可をクリックする
- これでローカルのアプリケーションは、必要な認証情報を自動検出し、自動で認証を行うことができる
- あわせて、不要な WARNING 出力を抑制するため、以下の環境変数をセットしてから、以降の手順へすすむ

export GOOGLE_AUTH_SUPPRESS_CREDENTIALS_WARNINGS=1



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



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

データモデルの作成とテーブルの作成

```
-zsh                                                    ⌥⌘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
```

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

```
-zsh                                                    ⌥⌘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

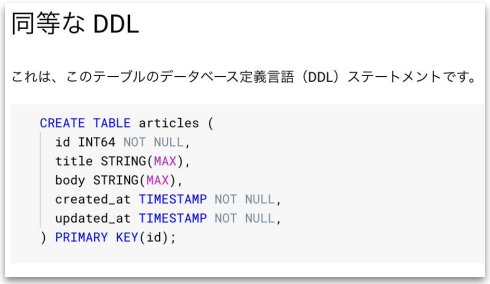
- bin/rails db:migrate を実行すると、作成したモデルに応じたテーブルが Cloud Spanner 上に作成される


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)

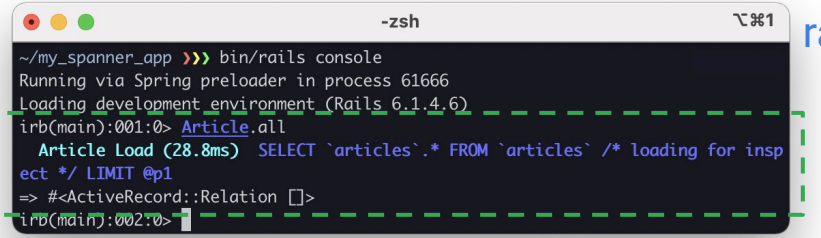
	列	種類	Nullable	注文
🔑	id	INT64	×	昇順
	title	STRING(MAX)	○	–
	body	STRING(MAX)	○	–
	created_at	TIMESTAMP	×	–
	updated_at	TIMESTAMP	×	–


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。

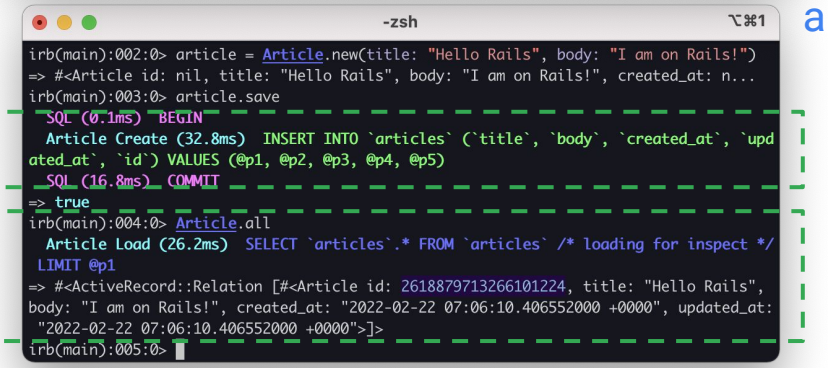
```
CREATE TABLE articles (
  id INT64 NOT NULL,
  title STRING(MAX),
  body STRING(MAX),
  created_at TIMESTAMP NOT NULL,
  updated_at TIMESTAMP NOT NULL,
) PRIMARY KEY(id);
```

rails console を用いて対話的にデータの読み書きをする



rails console の起動

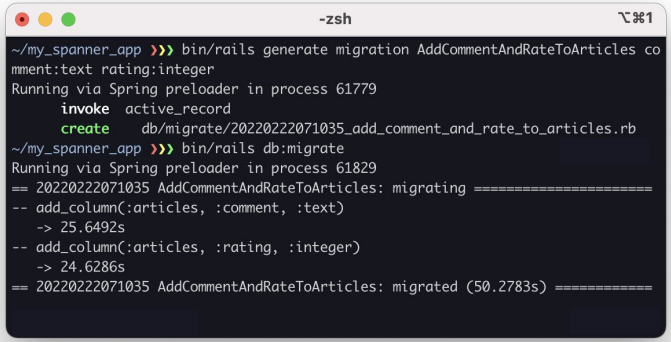
- bin/rails console を実行すると、REPL が起動し、Rails を用いて対話的に Cloud Spanner を読み書きできるようになる



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

既存テーブルに対して新たな列を追加する



db:migrate で実際に反映

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

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.

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

Active Record を使ったサンプルコード

39 lines (34 sloc) | 1.34 KB

```ruby
# Copyright 2021 Google LLC
#
# Use of this source code is governed by an MIT-style
# license that can be found in the LICENSE file or at
# https://opensource.org/licenses/MIT.

require "io/console"
require_relative "../config/environment"
require_relative "models/singer"
require_relative "models/album"

class Application
  def self.run # rubocop:disable Metrics/AbcSize
    from_album = nil
    to_album = nil
    # Use a read/write transaction to execute multiple statements as an atomic unit.
    ActiveRecord::Base.transaction do
      # Transfer a marketing budget of 10,000 from one album to another.
      from_album = Album.all.sample
      to_album = Album.where.not(id: from_album.id).sample

      puts ""
      puts "Transferring 10,000 marketing budget from #{from_album.title} (#{from_album.marketing_budget}) "\
           "to #{to_album.title} (#{to_album.marketing_budget})"
      from_album.update marketing_budget: from_album.marketing_budget - 10000
      to_album.update marketing_budget: to_album.marketing_budget + 10000
    end
    puts ""
    puts "Budgets after update:"
    puts "Marketing budget #{from_album.title}: #{from_album.reload.marketing_budget}"
    puts "Marketing budget #{to_album.title}: #{to_album.reload.marketing_budget}"

    puts ""
    puts "Press any key to end the application"
    STDIN.getch
  end
end

Application.run
```

04

PostgreSQL Interface

Cloud Spanner PostgreSQL Interface

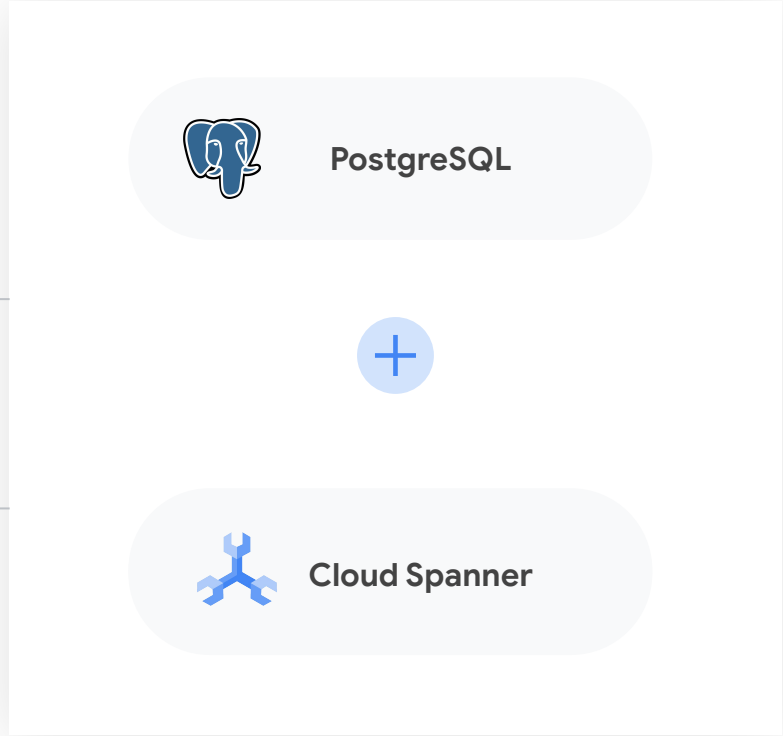
プレビュー

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

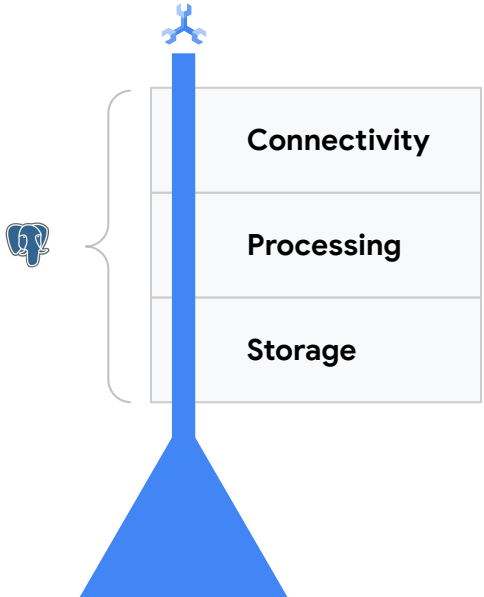
開発効率の向上
既存の PostgreSQL リソース及びナレッジを
活用することで、学習コストを削減



詳細: https://cloud.google.com/spanner/docs/postgresql-interface

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 が持つスケーラビリティ、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)



プロトコル、ツール、ドライバ、ORM

クエリ言語、関数、演算子

データ型、DDL、インデックス、メタデータ、メトリクス

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`

Data types
- `TEXT, VARCHAR`
- `NUMERIC`
- `BIGINT`
- `TIMESTAMP`
- `FLOAT`
- `DOUBLE`
- `BOOL`
- `BYTEA`

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

PostgreSQL

psql

Ecosystem clients
Stored Procedures
Triggers
`SERIAL`
Privileges
Concurrency control
Sequences
Nested transactions
Transactional DDL
Partial indexes
Extensions
Foreign data wrappers

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
- 無制限のスケーラビリティ
- 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

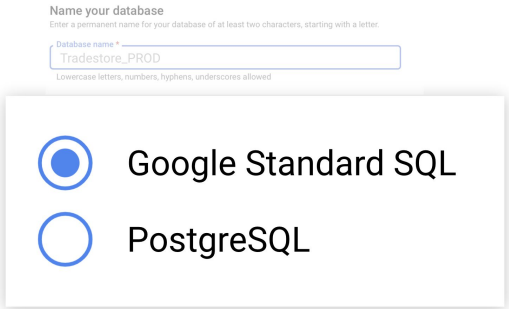
- リフト＆シフト、早急な移行
- シンプルなコスト削減

- 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

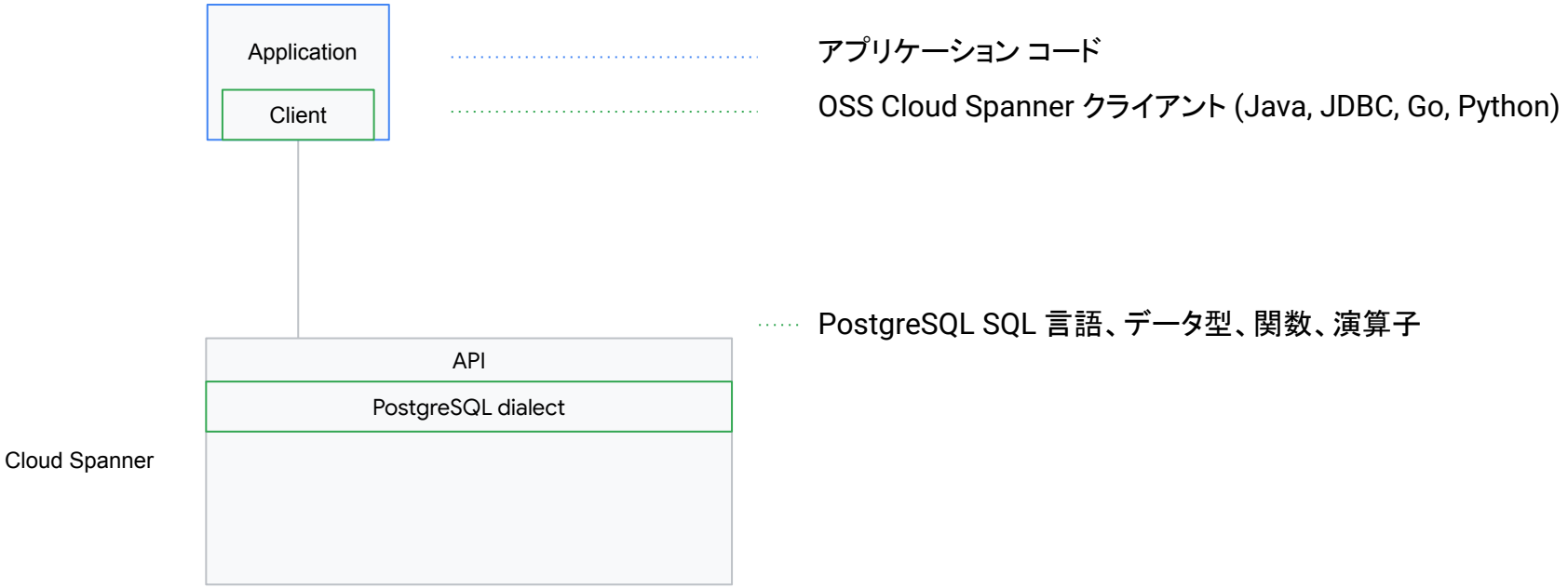
PostgreSQL クエリを Cloud Spaner インターフェースへ実行 + PostgreSQL プロトコル

- 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



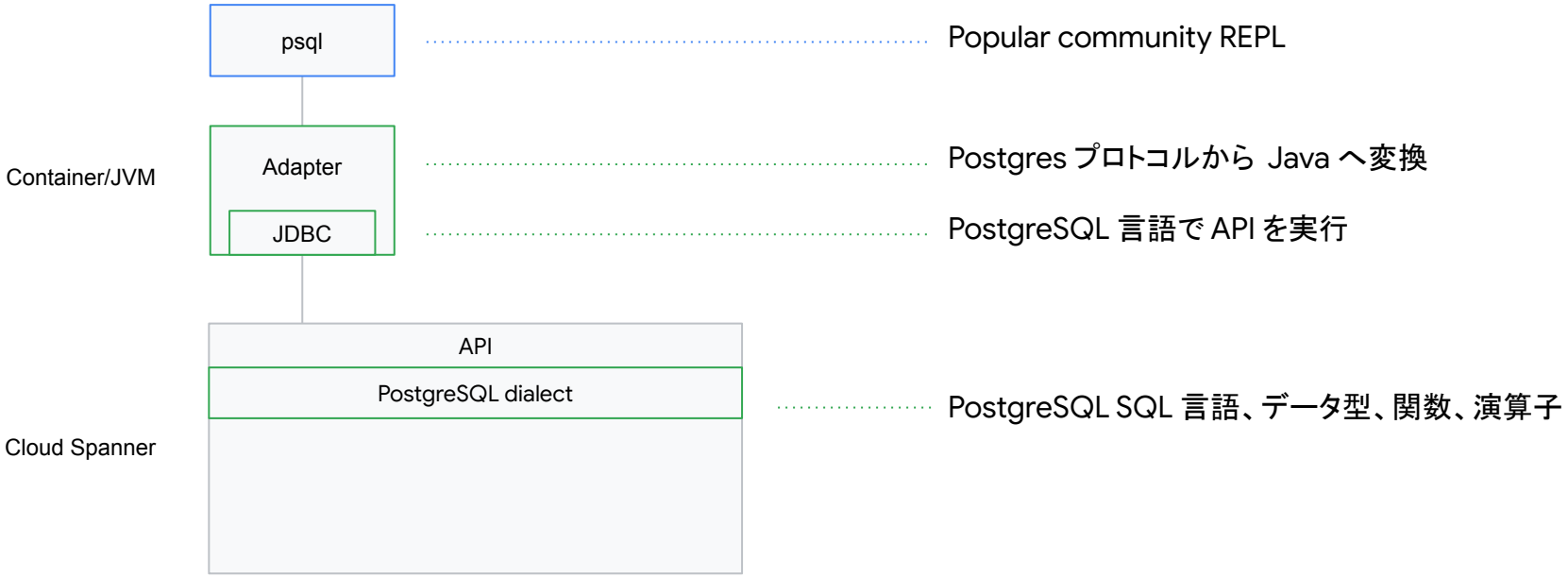
ユーザー エクスペリエンス: Application Development

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

詳細: https://cloud.google.com/bigquery/docs/cloud-spanner-federated-queries

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;
```

詳細: https://cloud.google.com/spanner/docs/views

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;
```

詳細: https://cloud.google.com/spanner/docs/working-with-json

Public Preview

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

詳細: https://cloud.google.com/spanner/docs/instance-configurations#moving_an_instance_to_a_different_configuration

Thank you.

Google Cloud