



E.G.G. Japan

Cloud Run と Firestore で作る Web アプリケーション

Updated: Jun 2021

本日のアジェンダ

1. Google Cloud Platform 入門 - Cloud Run x Firestore 編 - (40 mins)
 - 1.1. Cloud Run, Firestore, Memorystore の概要 (30 mins)
 - 1.2. ハンズオン前提知識編 (10 mins)
2. Break (10 mins)
3. ハンズオン (90 mins)
 - 3.1. Cloud Run のデプロイ (20 mins)
 - 3.2. Firestore の組み込み (30 mins)
 - 3.3. Memorystore の組み込み (30 mins)
 - 3.4. Clean up (10 mins)

Cloud Run 入門

- Cloud Run x Firestore 編 -

Cloud Run

開発者は Kubernetes を直接触りたいのか？

Have to do

コードを書く

\$ docker build

\$ docker push

\$ kubectl apply -f deployment.yml

\$ kubectl apply -f service.yaml

\$ kubectl apply -f autoscal.yaml

他にも監視・ロギングの設定とか..

Want to do

コードを書く

もっと簡単にするならサーバーレス



No Infra
Management



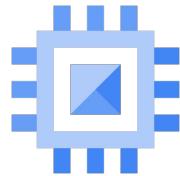
Fully Managed
Security



Pay only
for usage

Google Cloud における Compute の選択肢

Server



Compute
Engine

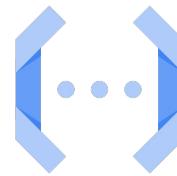
- 仮想マシン



Kubernetes
Engine

- Kubernetes マ
ネージドサービス

Serverless



Cloud
Functions

- イベント駆動
- 関数の実行
- 最大 9 分の実行
時間
- 1 concurrency



App Engine

- Web アプリ基盤
- ユーザー数・ナ
レッジの多さ
- 1 プロジェクトに
つき 1 リージョン



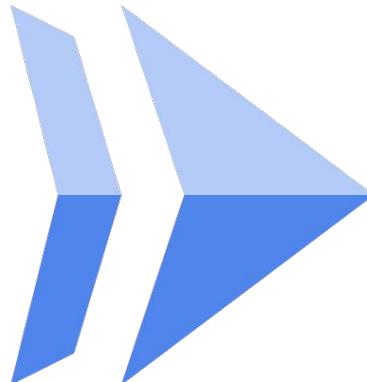
Cloud Run

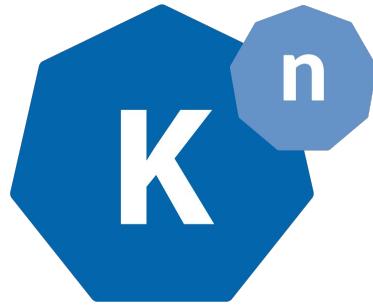
- Knative マネージ
ドサービス
- 1 プロジェクト N
リージョン
- 最大 15 分 の実
行時間

Cloud Run

Knative がベースの新しいサーバーレス

サーバーレスのアジリティを
コンテナ化したアプリに





Knative

Kubernetes の上に
サーバーレス環境を実現する OSS



<https://github.com/knative>

[kay-native]

Cloud Run だとコンテナをシンプルに使える

Kubernetes

コードを書く

```
$ docker build
```

```
$ docker push
```

```
$ kubectl apply -f deployment.yml
```

```
$ kubectl apply -f service.yaml
```

```
$ kubectl apply -f hautoyaml
```

他にも監視・ロギングの設定とか..

Cloud Run

コードを書く

```
$ docker build
```

```
$ docker push
```

```
$ gcloud run deploy..
```

Cloud Run の主な特徴



高速なデプロイ

ステートレスなコンテナ

高速に 0 to N スケール

数秒でデプロイし URL を付与



サーバレス・ネイティブ

管理するサーバーはなし
コードに集中

言語やライブラリの制約なし

きっちり使った分だけお支払い



高いポータビリティ

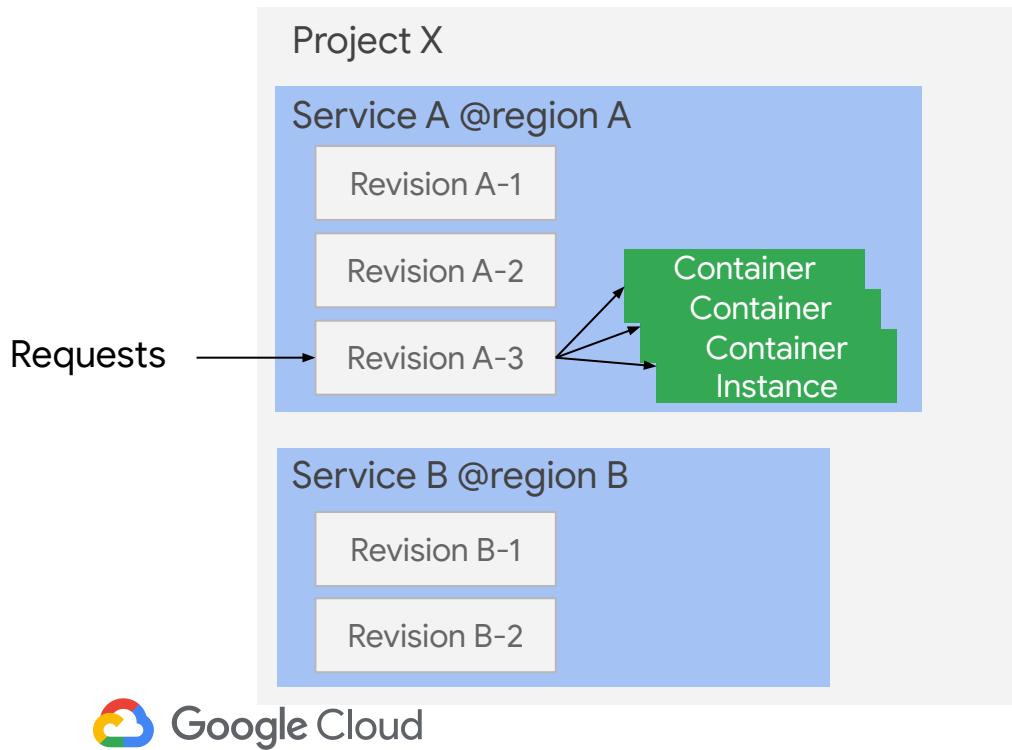
どこでも同じ Developer Experience
フルマネージでも GKE のクラスタ上で
も

Knative API の一貫性

ロックインの排除

Cloud Run を使う前に 知っておいた方がよいこと

Cloud Run のリソースモデル



Service

Cloud Run の主リソース

Service 毎に Endpoint を提供

自動で設定される a.run.app ドメイン、
もしくはカスタム ドメインが選択可能

Revision

デプロイするごとに生成される

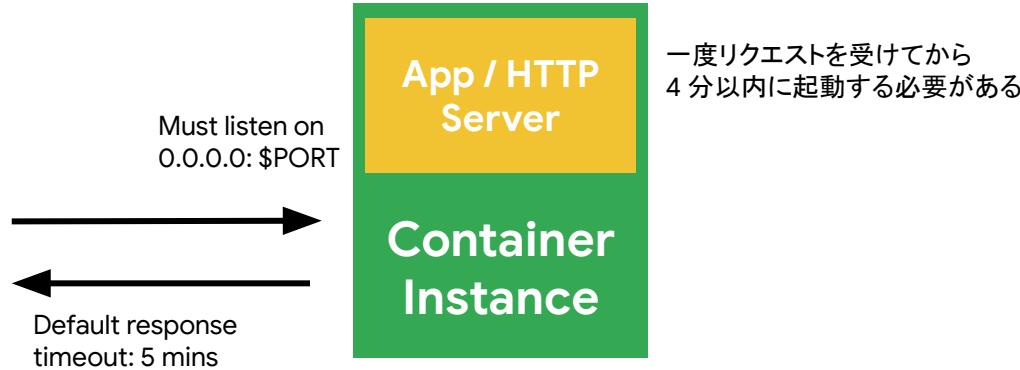
コンテナ イメージとデプロイ時に指定される
環境変数やパラメーターから構成される

Container Instance

実際にリクエストを受けるコンテナ、
リクエストの数に応じて自動的にスケール

コンテナに関する決まりごと

- Linux x86_64 ABI をサポート
- 0.0.0.0 かつ、環境変数 PORT に対してリッスン
- リクエストを受信してから 4 分以内に HTTP サーバーを起動する必要
- レスポンス タイムアウトはデフォルト 5 分 (GA: 最大 15 分 , Beta: 最大 60 分)
- レスポンスタイム内にレスポンスを返さなかった場合は、504 エラーを返す



コンテナのスペック

- CPU
 - デフォルト 1vCPU
 - 変更可、1vCPU, 2vCPU, 4vCPU から選択
 - 4vCPU の場合はメモリを 2GB 以上選択する必要あり
- メモリ
 - デフォルト 256 MB
 - 変更可、最小 128 MB ~ 最大 8 GB
- ファイルシステム
 - 読み書き可能
 - コンテナに割り当てられたメモリ上を利用
 - データの永続性なし

オートスケーリング

Cloud Run ではリクエスト数に応じて

特に明示的に設定をしていなくてもオートスケーリングを行う

リクエストがしばらく(体感 5 分)無い場合、Container Instance が 0 になる

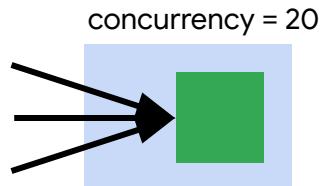
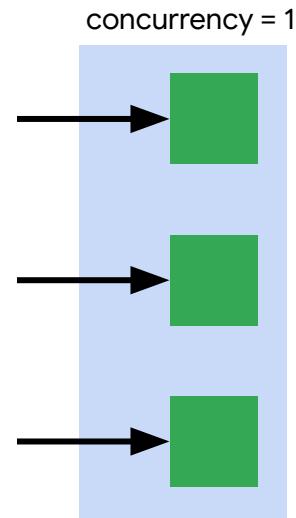


Concurrency(コンカレンシー)

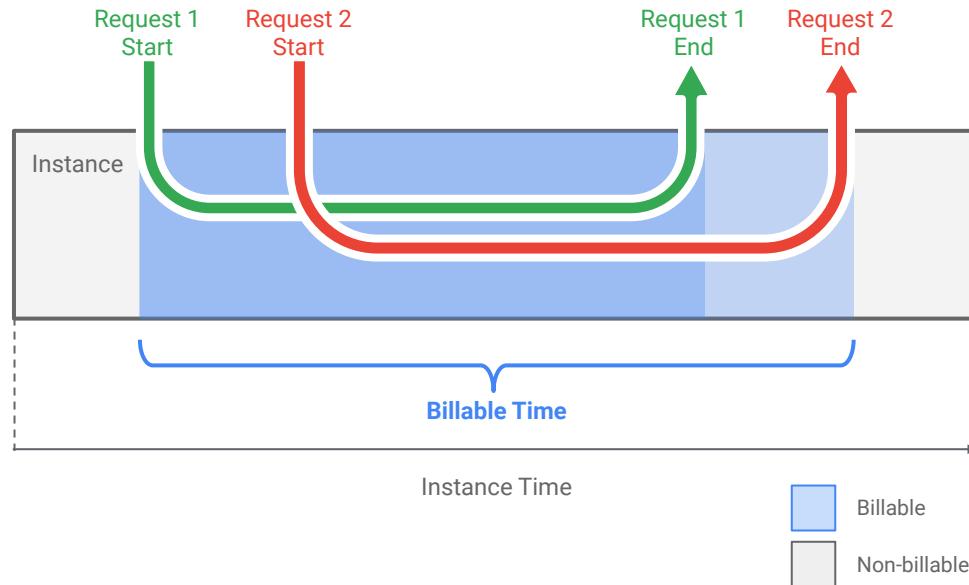
Concurrency とは同時の 1つの Container Instance に投げられるリクエストの最大数。

Google Cloud Functions など一般的な FaaS は一度に 1つのリクエストしかハンドルできない。
なので "concurrency = 1".

Cloud Run の場合、concurrency の値を 1から 80 まで設定できる(default: 80) ので、1つの Container Instance で同時に複数のリクエストを処理することができる。



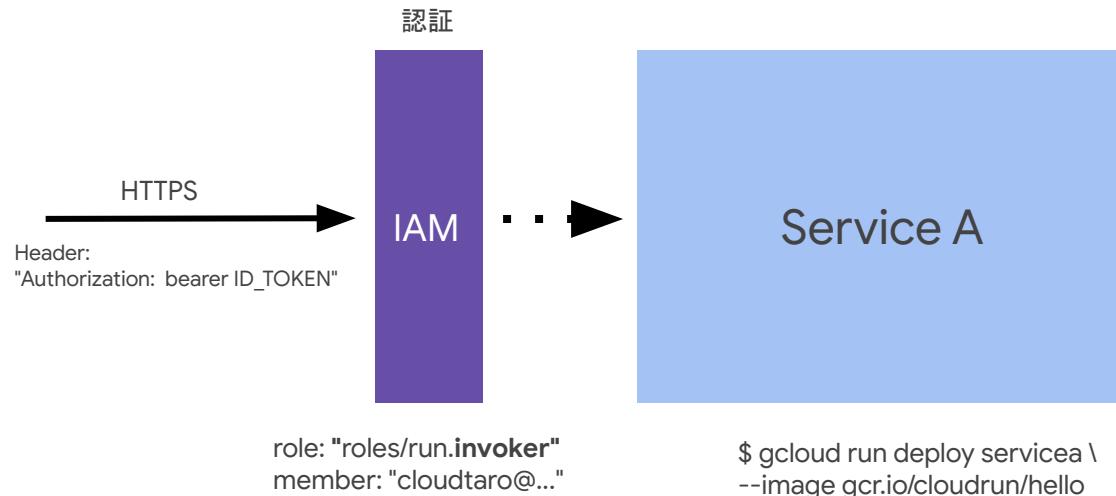
課金時間の考え方



Cloud Run を より実践的に使っていく

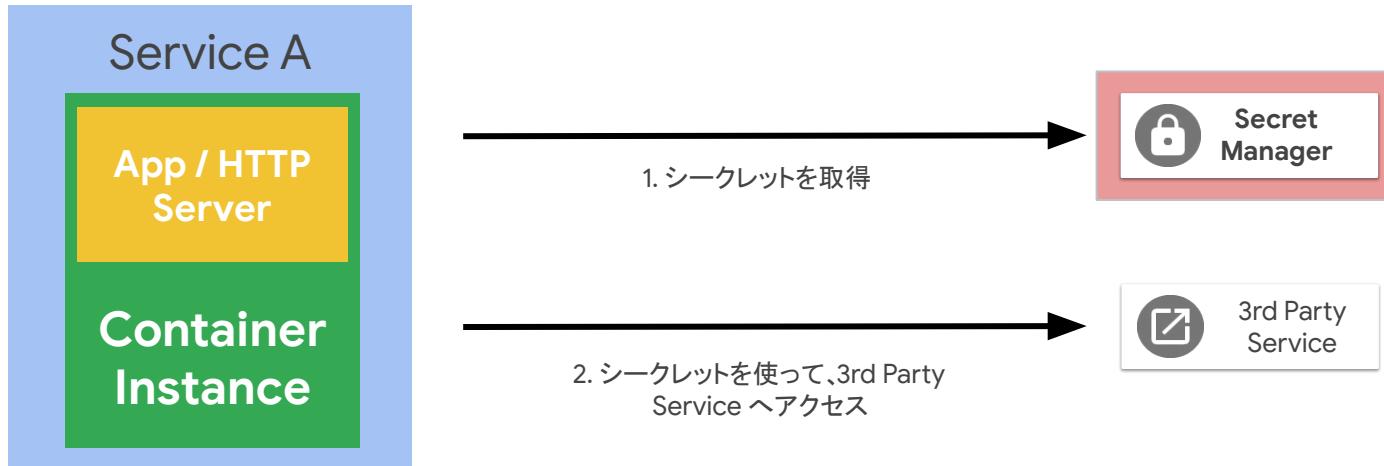
アクセスを制御する

GCP の IAM によるアクセス認証で不特定多数からのアクセスを防ぐ



3rd party サービスをアプリケーションから利用する

Cloud Run 上で動作するアプリケーションから **Secret manager** を使い、シークレットを安全かつ、従来の Cloud KMS や Berglas よりも簡単に扱う

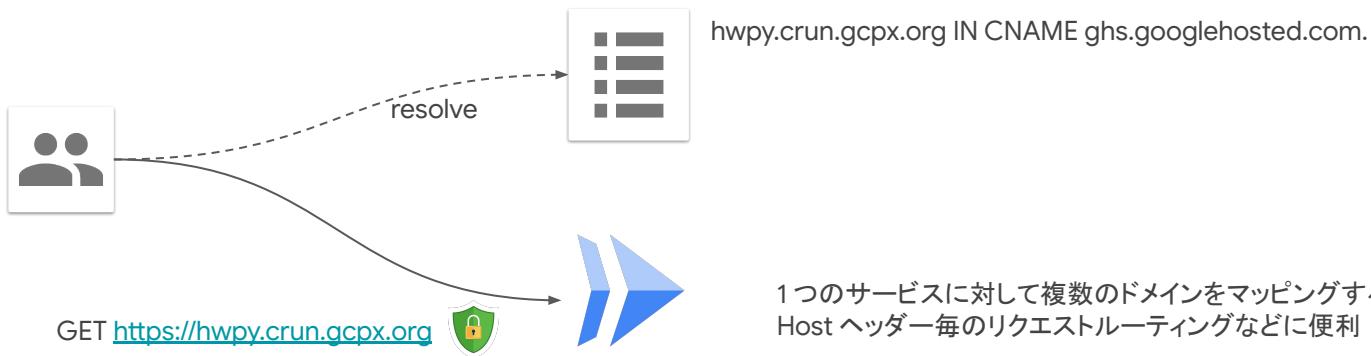


独自ドメインを使う

デフォルトでは `run.app` のサブドメインが割り当てられるが、

お客様自身のドメインを使うことも可能

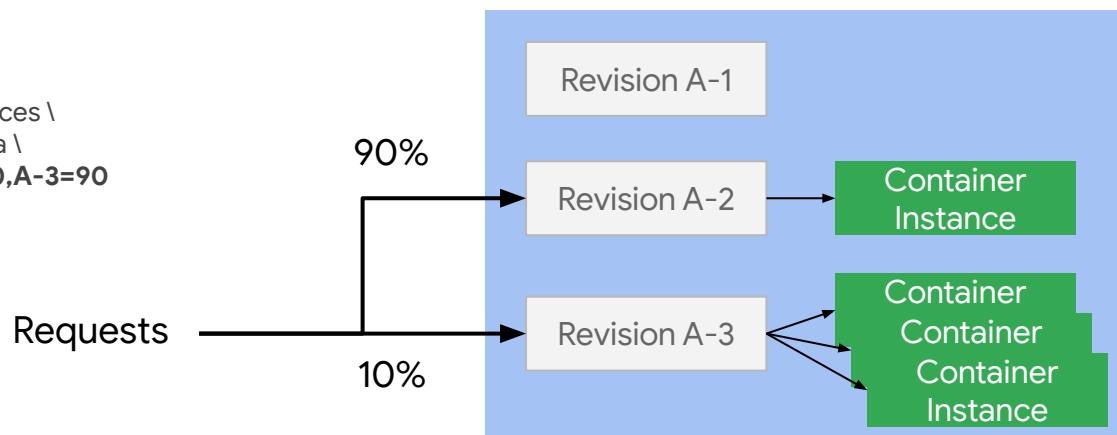
SSL 証明書は Let's Encrypt で自動的に払い出される(3 か月に一度、自動で更新)



トラフィックを制御する

Update traffic 機能を使うことで、Blue / Green Deployment や Canary release などが可能に

```
$ gcloud beta run services \  
update-traffic servicea \  
--to-revisions=A-2=10,A-3=90
```

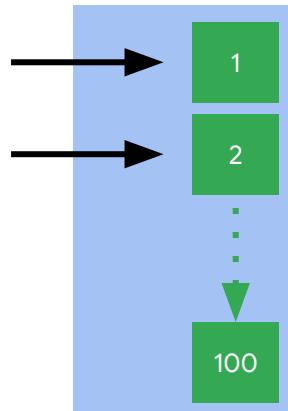


Container Instance が増えすぎないようにする

Max instances を指定することで課金の最適化と DB のコネクション枯渇などを防ぐ

```
$ gcloud run deploy servicea \  
--image gcr.io/cloudrun/hello \  
--concurrency 20 \  
--max-instances 100
```

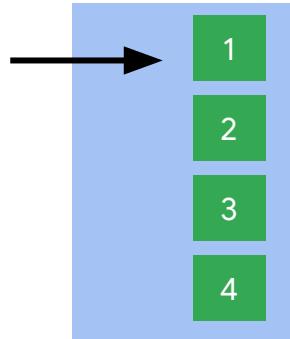
リクエスト数増加に伴い
最大 100 Container Instance まで
スケールアウト



Cold Start に対応する

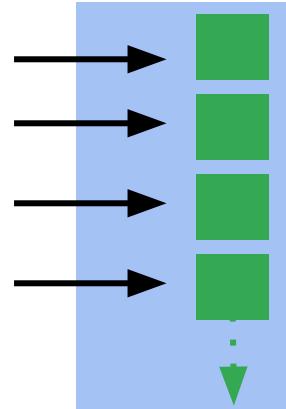
Minimum instances を指定することで Cold Start の影響を小さくすることができる

常時起動する Container Instance 数を
指定しておく



```
$ gcloud alpha run deploy servicea \  
--image gcr.io/cloudrun/hello  
--min-instances=4
```

リクエストがスパイクした時に
Cold Start の影響を小さくできる

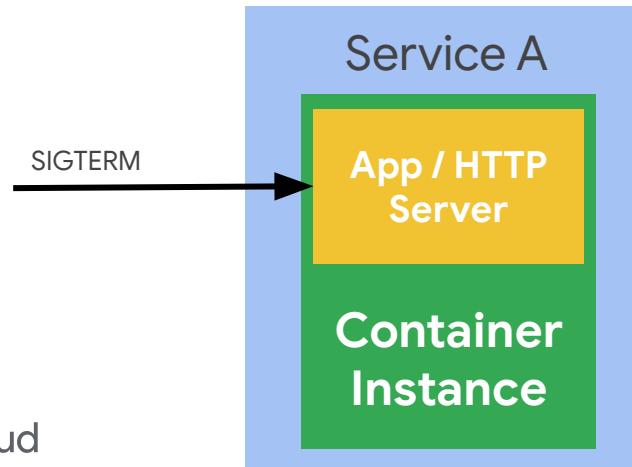


Container instance を graceful shutdown する

Cloud Run が Container Instance をシャットダウンする際、

SIGTERM が各 Container Instance へ送られ、**SIGKILL** されるまで **10 秒間の猶予**が与えられる。

アプリケーションで **SIGTERM Handler** を実装しておくことで Graceful shutdown が可能に。

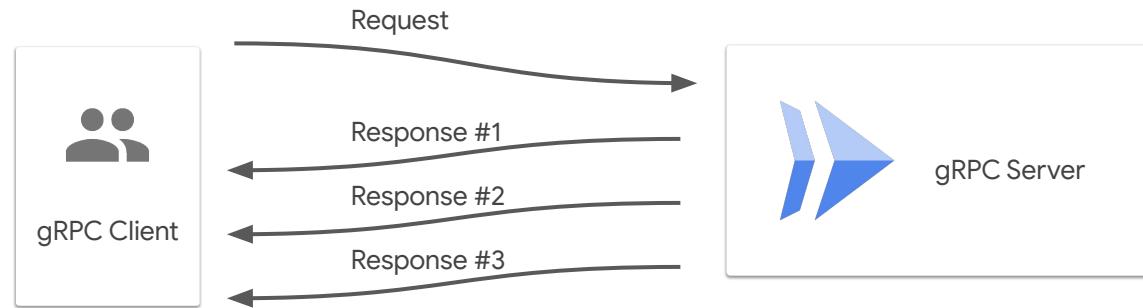


```
app.py  
import signal  
  
def handler(signal, frame):  
    # close db connections  
    # send any buffered telemetry data  
  
signal.signal(signal.SIGTERM, handler)
```

サーバー / クライアント間の長時間接続、ストリーミングを行う

Cloud Run では **gRPC Server-side streaming** 及び **HTTP/1.1 Chunked transfer encoding** をサポート。ビデオファイルなど大きめのファイルをストリーミングしたり、大きめ / 長めの計算処理の進捗を逐次クライアントに返していくアプリケーションを実行可能に。

※ gRPC は Unary もサポート



Cloud Run と他 GCP サービスの連携

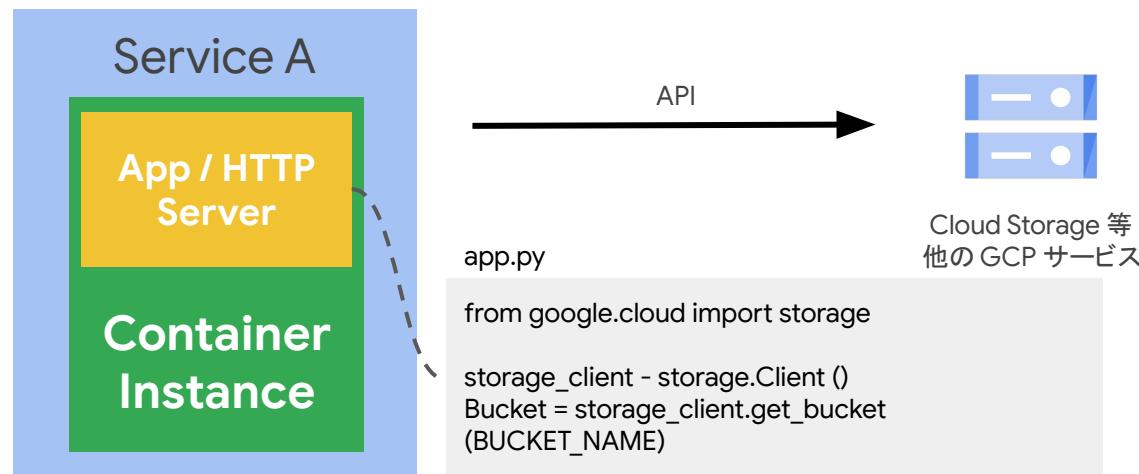
他の GCP サービスをアプリケーションから利用する

サービス アカウントを使って他の GCP サービスへアクセス

サービスのデプロイ時に任意のサービス アカウントを指定することが可能

```
$ gcloud run deploy servicea \
--image gcr.io/cloudrun/hello \
--service-account "run-storage@..."
```

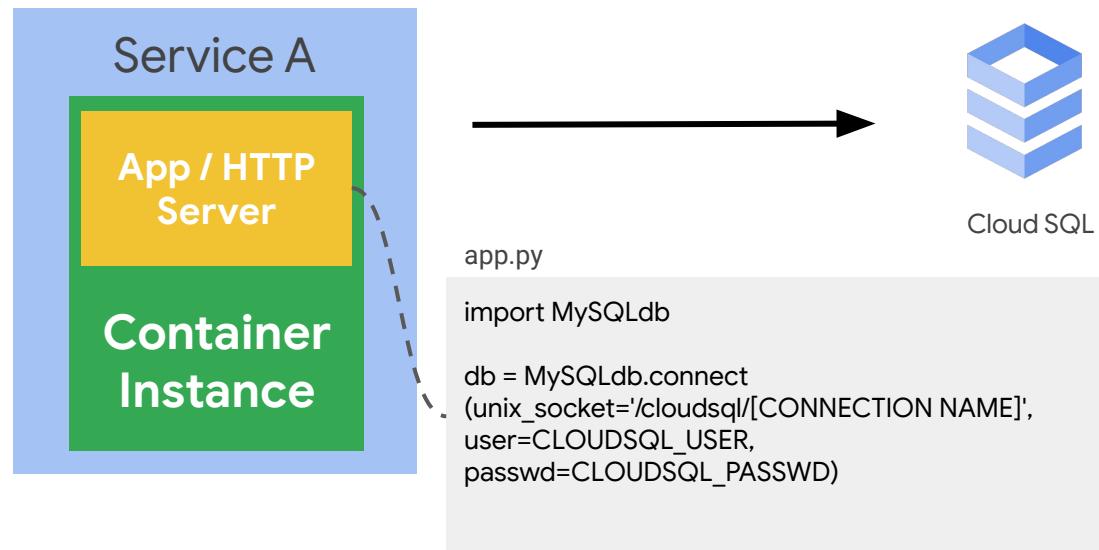
デフォルトのサービスアカウントは
Compute Engine のデフォルトサービス
アカウント



DB とセキュアに接続する

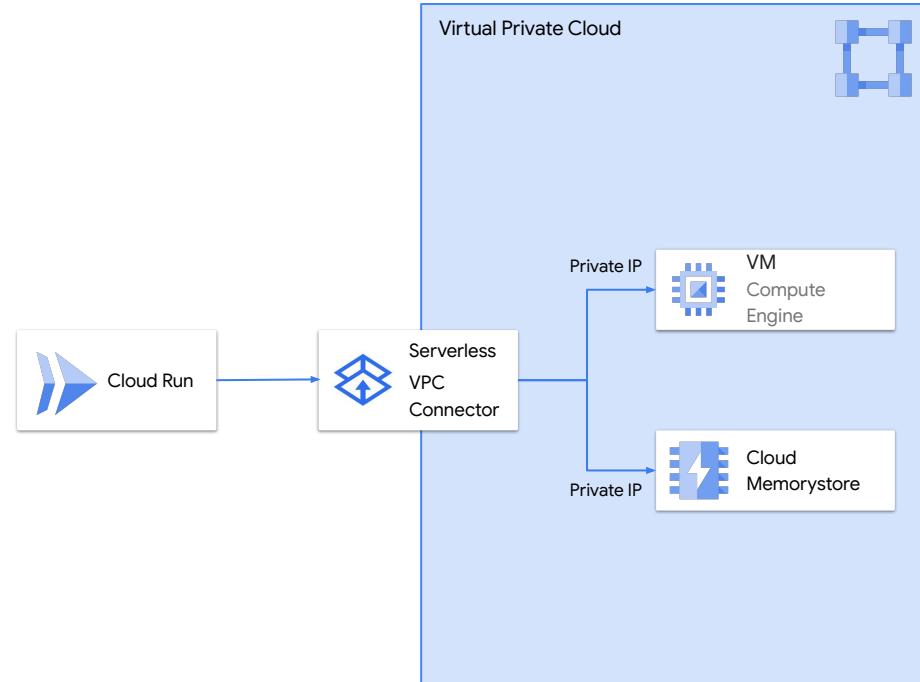
アプリケーションから UNIX ソケット経由で **Cloud SQL Proxy** へ接続可能
第 2 世代の MySQL インスタンス or PostgreSQL インスタンス を推奨

```
$ gcloud run deploy servicea \
--image gcr.io/cloudrun/hello \
--add-cloudsql-instances
INSTANCE-ID
```



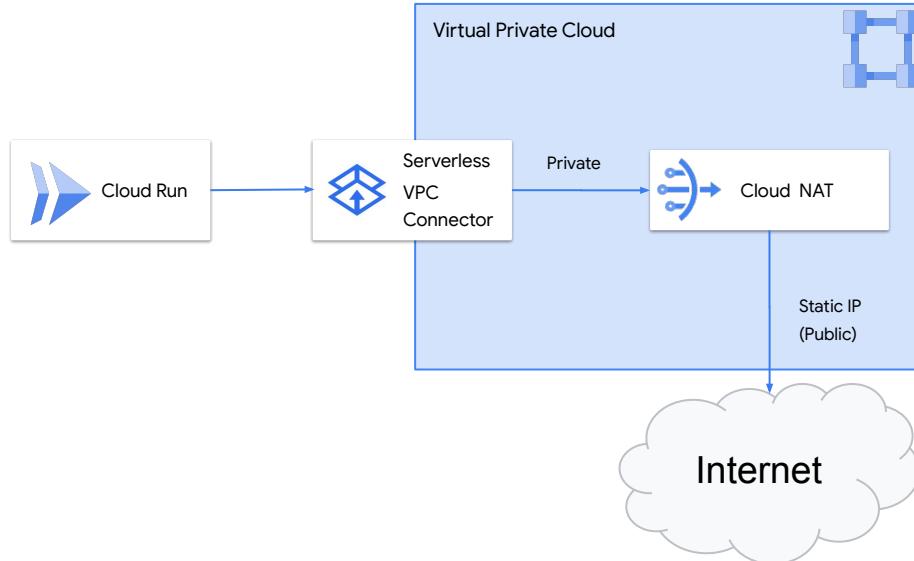
VPC 内のリソースへの接続

Serverless VPC Access が Cloud Run をサポート
Cloud Memorystore や Compute Engine
など VPC 内で動作するリソースへ接続出来るように
Shared VPC へのアクセスは可能



Cloud Run 発の通信の IP アドレスを固定する

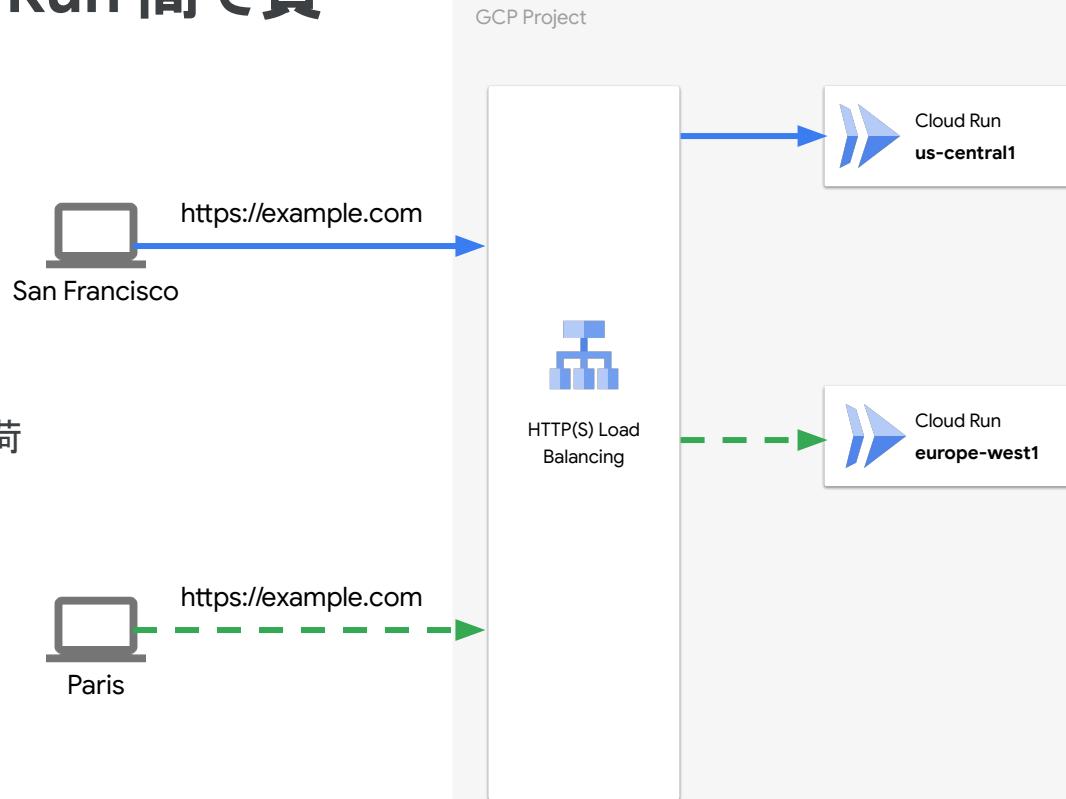
Cloud Run 発の全ての通信を Serverless VPC Access へ向けることが可能に。
本機能により Cloud NAT を使って Cloud Run 発の全ての通信の IP アドレスを固定することが可能に。



複数リージョンの Cloud Run 間で負荷分散

Serverless NEG により、
HTTP(S) Load balancing と連携して
以下を実現

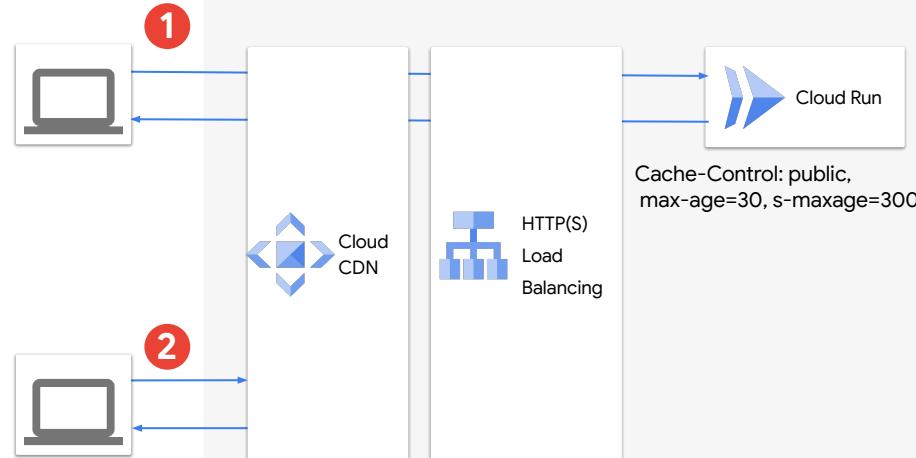
- 複数リージョン の Cloud Run 間での負荷分散 (GAE/GCF も含める事が可能)
- 持ち込みの SSL 証明書使用
- パスベースルーティング



CDN を使って静的コンテンツをキャッシュ

Serverless NEG により、
HTTP(S) Load balancing と連携することで、
Cloud Run と Cloud CDN が連携可能に。

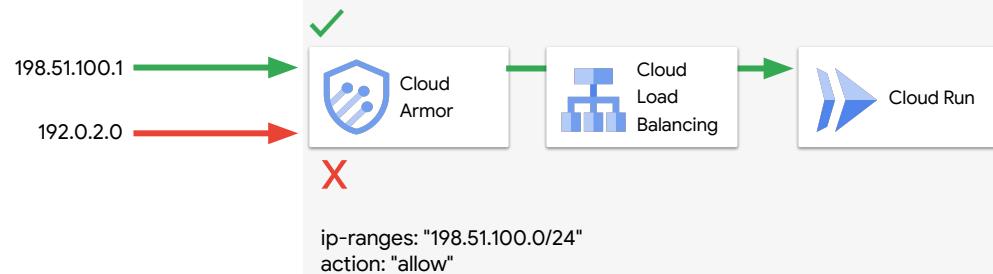
Cache-Control header を使い、
静的コンテンツを Cloud CDN にキャッシュ。
Cloud Run の負荷と Egress トラフィックを軽減し
つつ、レスポンスタイムを短縮。



DDOS 対策、ネットワークフィルタリング

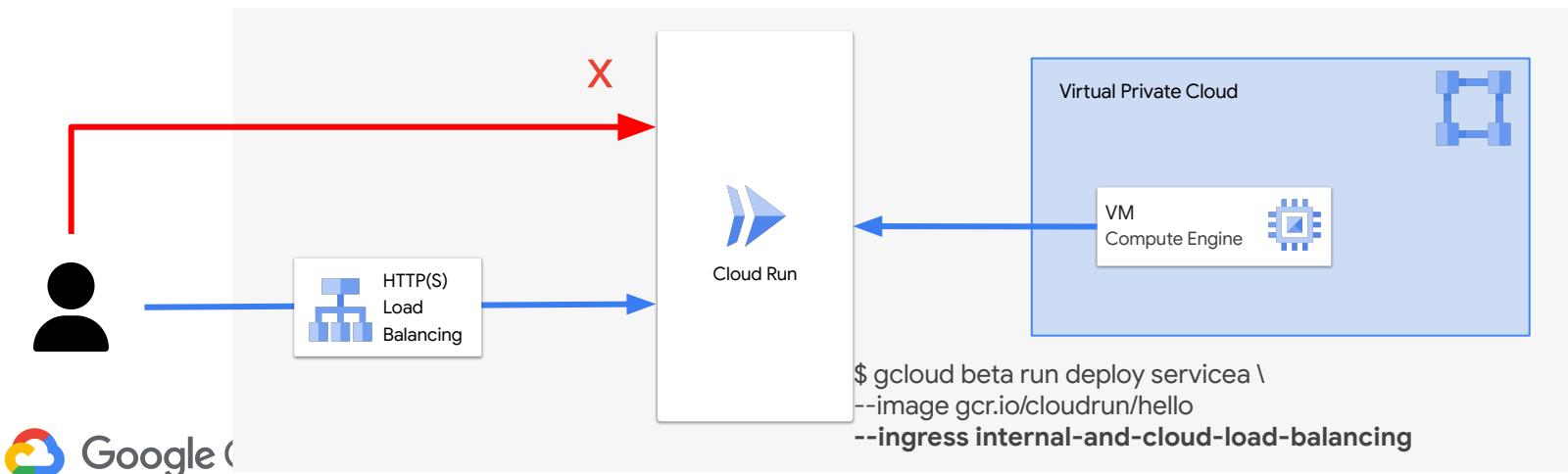
Serverless NEG により、
HTTP(S) Load balancing と連携することで、
Cloud Run と Cloud Armor が連携可能に。

- DDOS 防御
- IP アドレス / 地理情報をベースにした
フィルタリング



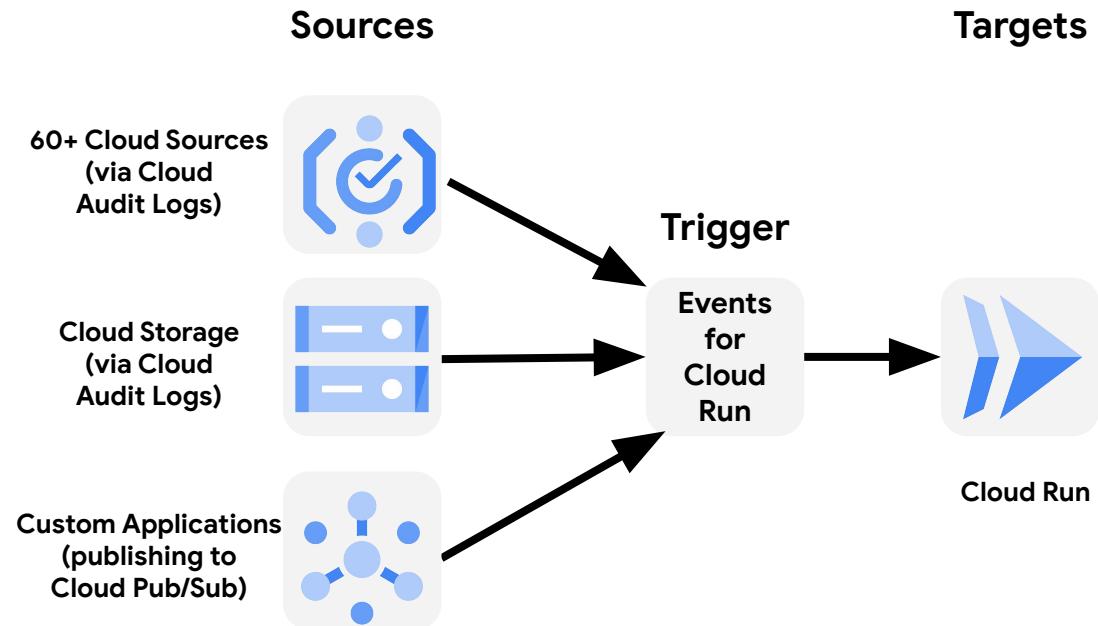
インバウンドリクエストの制御

-ingress オプションにより Cloud Run へのインバウンド リクエストを制御することが可能。e.g. HTTP(S) Load Balancing を経由したリクエストと同一プロジェクトの VPC からのみリクエストを受け付ける



イベントドリブン アーキテクチャ

Events for Cloud Run により、各 Event source と Cloud Run を繋いでイベントドリブンなアーキテクチャを構成可能

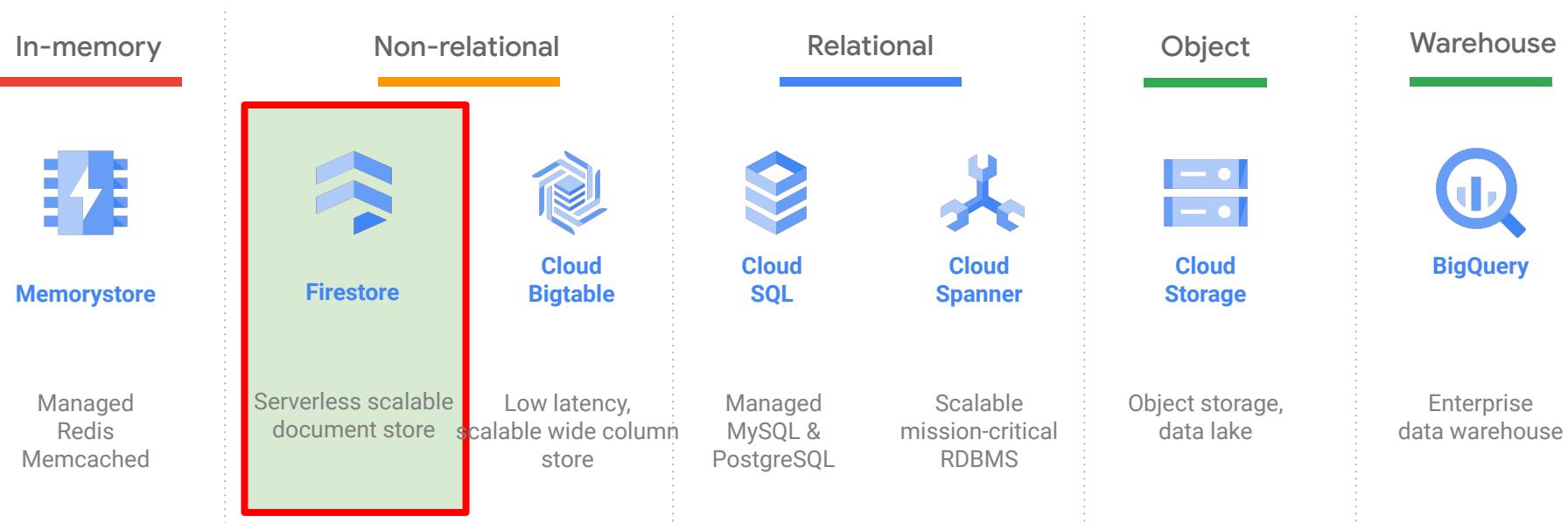


Cloud Run の特徴まとめ

- Cloud Run はコンテナをシンプルに使えるサービス
- 従来の PaaS / FaaS にあった言語ランタイム及びライブラリに制約がない
- 0 to N のスケーリングが特徴、完全な Pay per Use
- OSS の Knative がベース、ロックインなし
 - GCP だけではなくオンプレミスの Knative、他社のサービスでも！

Cloud Firestore

データベース ポートフォリオ



Firebase

フルマネージドでサーバレスな
ドキュメント指向 NoSQL データベース

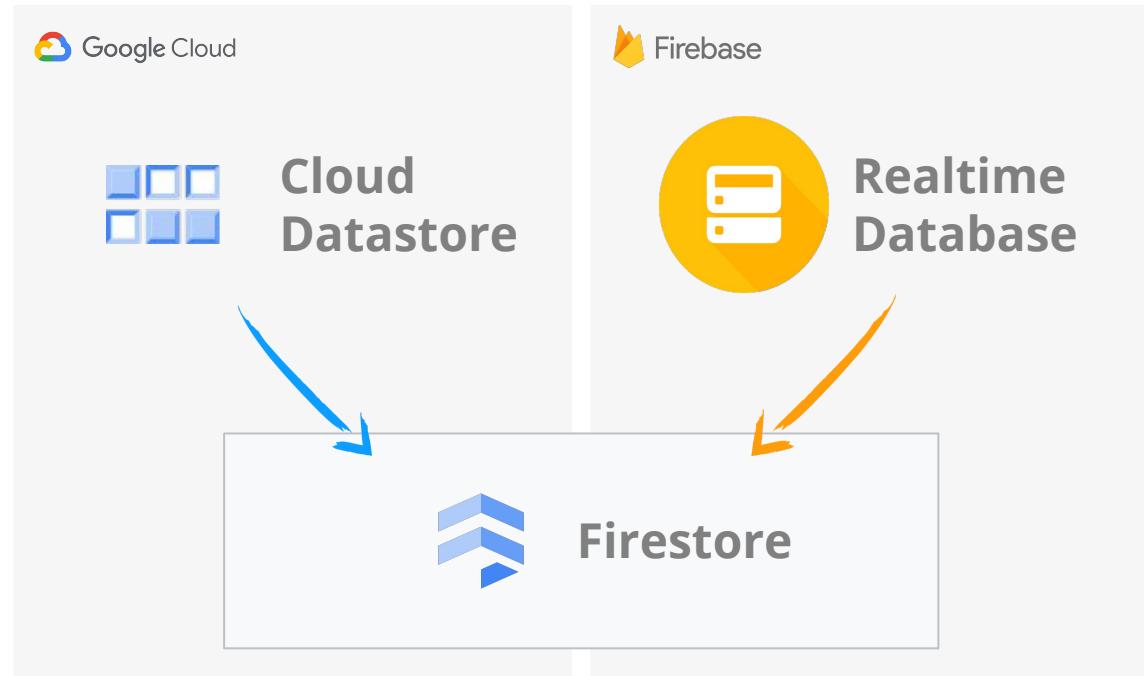
- ACID トランザクション
- リアルタイム同期
- オフラインアクセス
- Datastore との互換モード



Firebase の系譜

Cloud Datastore の特徴と
Realtime Database の特徴を
兼ね備える Firestore

Firebase には
Native モードと
Datastore モードという
2 種類のモードがある



Cloud Firestore の 2 つのモード

Datastore mode

- Datastore との後方互換性
- Datastore の一部制限を排除

Native mode

- 次期メジャーバージョン
- 数百万のクライアント同時実行
- リアルタイムアップデート

サーバサイドアクセス

ライブラリ

C#, Java, Go, Node.js, PHP, Python, Ruby

REST & gRPC API

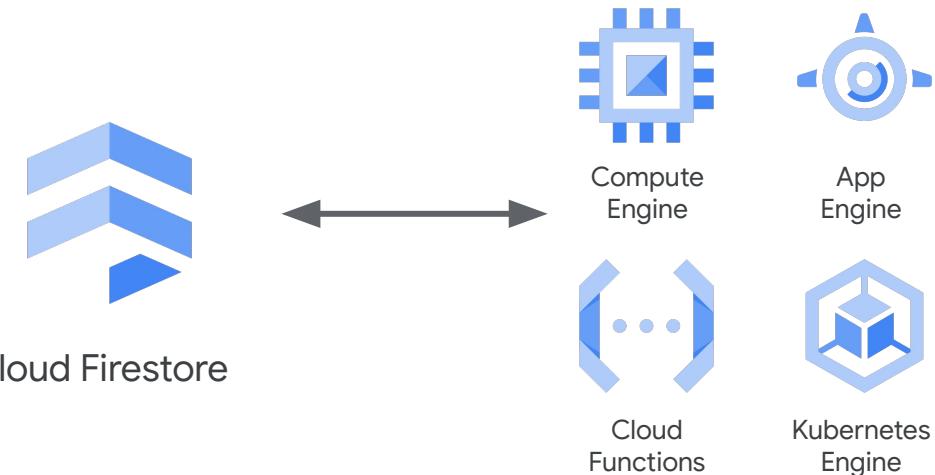
他システムと容易に連携

信頼性と可用性

マルチリージョン構成の SLA 99.999%

スケーラブル

トラフィックに応じて自動スケール



クライアントアクセス

Firebase SDK サポート

Web, iOS, Android 対応

Firebase Auth, Security Rules

データアクセス権限を制御

Cloud Functions

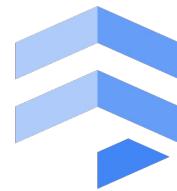
クラウドでロジックを実行

Firebase Hosting

静的ファイルをホスティング



App + Client SDKs



Cloud Firestore



基本データ構造

users

alovelace

first : "Ada"

last : "Lovelace"

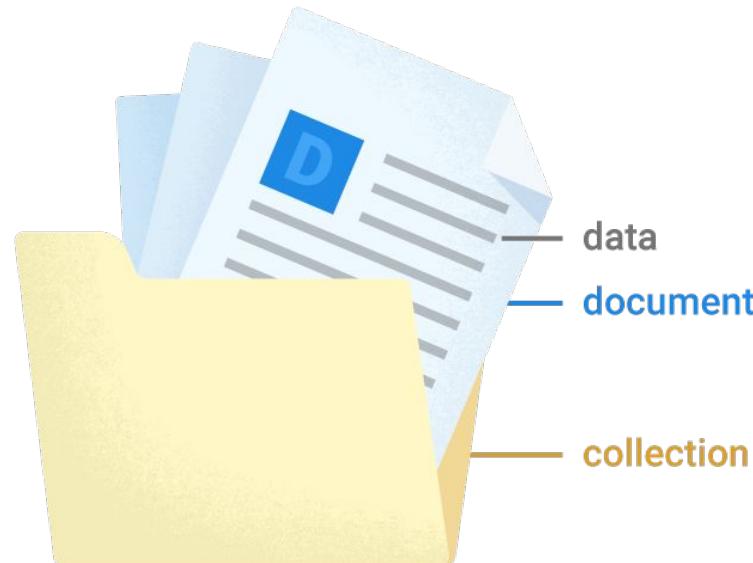
born : 1815

aturing

first : "Alan"

last : "Turing"

born : 1912



Firestore の特徴

- 強整合性
- スキーマレス
- インデックス
 - デフォルトで全てのドキュメントフィールドにインデックスが付与される
- クエリ
 - 1 クエリでフィルタをつなぎ合わせたり、フィルタと並べ替えを組み合わせ
 - ドキュメント全体あるいはサブコレクションのクエリが可能
- スケーラビリティ
 - 1 ドキュメントあたり 10,000 Write / sec までスケール
- インポート/エクスポート機能
- Pay per Use

その他 Firestore で覚えておくといいこと

- アトミックオペレーション
 - トランザクション
 - 1つ以上のドキュメントに対して Read/Write
 - バッチ書き込み
 - 1つ以上のドキュメントに対して Write
 - 1回のオペレーションでは、最大 500 ドキュメントまで書き込み可
- 1秒あたり同一ドキュメント 1書き込み
- ドキュメント ID は分散すべし
- 「500/50/5」ルール
 - 新しいコレクションのオペレーションは毎秒 500 回を上限とし、その後、5 分ごとにトラフィックを 50% 増やしていく
- ローカル エミュレーター

ハンズオン

ハンズオン

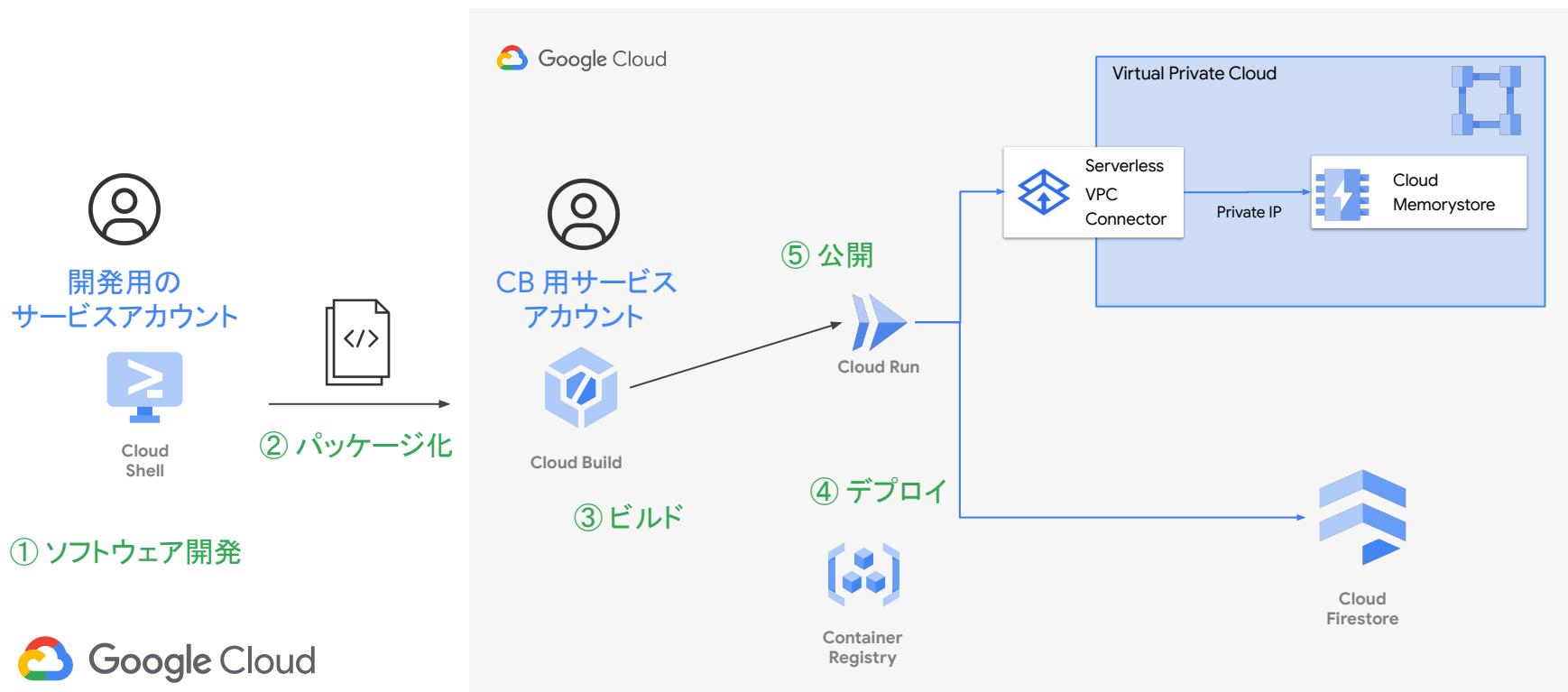
グーグル・クラウド・ジャパン合同会社
カスタマー エンジニア
Mourad El Azhari (ニックネーム : ムラディ)



ハンズオンの目的

GCP のネットワークやアクセス管理の基本コンセプトを理解しながら、ゲームインフラを設計する際の代表的なパターンの 1 つともいえる「Cloud Run」と「Cloud Firestore」をはじめとした各サービスを触ってみることで、GCP での Web アプリケーションの構築について学ぶハンズオンです。

ハンズオンのシステム構成 - GCP サービス完成図



はじめるまえに

- GCP アカウントをまだ開設されてない方は、[こちら](#)から、GCP コンソールでのサインアップ & 無料トライアルの申し込みを行なってください。(既に GCP アカウント開設済みの方は不要です。)
- クーポンコードを有効にするためには、GCP アカウントを無料トライアルから有料アカウントへアップグレードする必要があります。
- アカウントをアップグレードすると、クレジットを使い切った時点、またはクレジットが期限切れになった時点で自動的に請求が始まります。

Billing Account の設定確認

- GCP コンソールのメニュー「お支払い」→「請求先アカウントを管理」
- 「マイプロジェクト」を選択し、ハンズオンで利用するプロジェクトに Billing Account が紐付いていることを確認

組織を選択:	▼			
請求先アカウント		マイ プロジェクト		
名前	ID	請求先アカウント ▾	請求先アカウント ID	操作 ?
名前	ID	請求先アカウント	請求先アカウント ID	操作

(任意) Billing Alert の通知設定

- こちらの手順を参考に、Billing Alert を設定することで、利用料金がある金額に達すると通知を出すことが出来ます。
- 予算額を設定する際に「費用にクレジットを含める」を有効にすることで、クレジットをご利用の方もアラートを受け取ることができます。本プログラムは 3 か月のため、毎月 \$100 で通知を出すようにすれば \$300 を有効に使えると思います。

ハンズオンはじめ

- ハンズオンでは最初にここに書いてあることをしたらスライドは表示しないので、まずはここに書いてある手順で GCP コンソールを開いてください。
- コンソールで Cloud Shell を立ち上げ、必要なリソースを取得します。
- 以下に表示している [リンク](#)をクリックすると、Cloud Shell が開き、リポジトリがクローンされ、ディレクトリに移動します。
- チュートリアルが右側に開かれない場合は `teachme tutorial.md` を実行してください。

ハンズオンは、**14:50** 開始です
ハンズオンのご準備をしてお待ちください

うまくいかない場合

`git clone`

<https://github.com/GoogleCloudPlatform/gcp-getting-started-lab-jp>

`cd`

`gcp-getting-started-lab-jp/gaming/egg3-3`

`teachme tutorial.md`

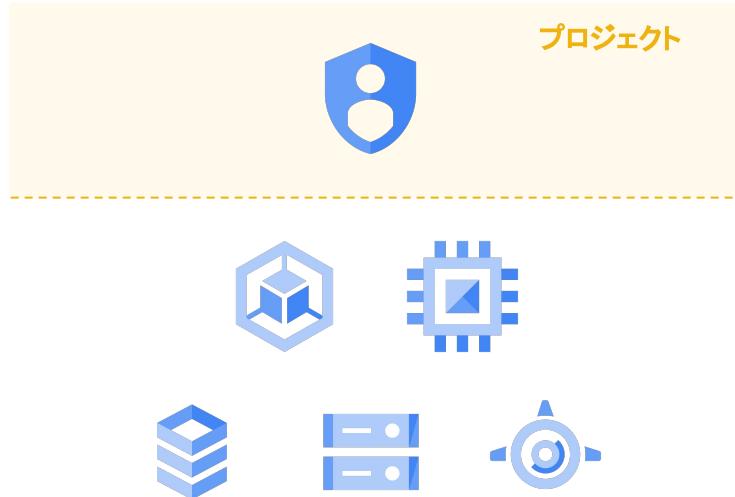
Appendix

ハンズオン 前提知識編

GCP プロジェクト

GCP プロジェクトとは

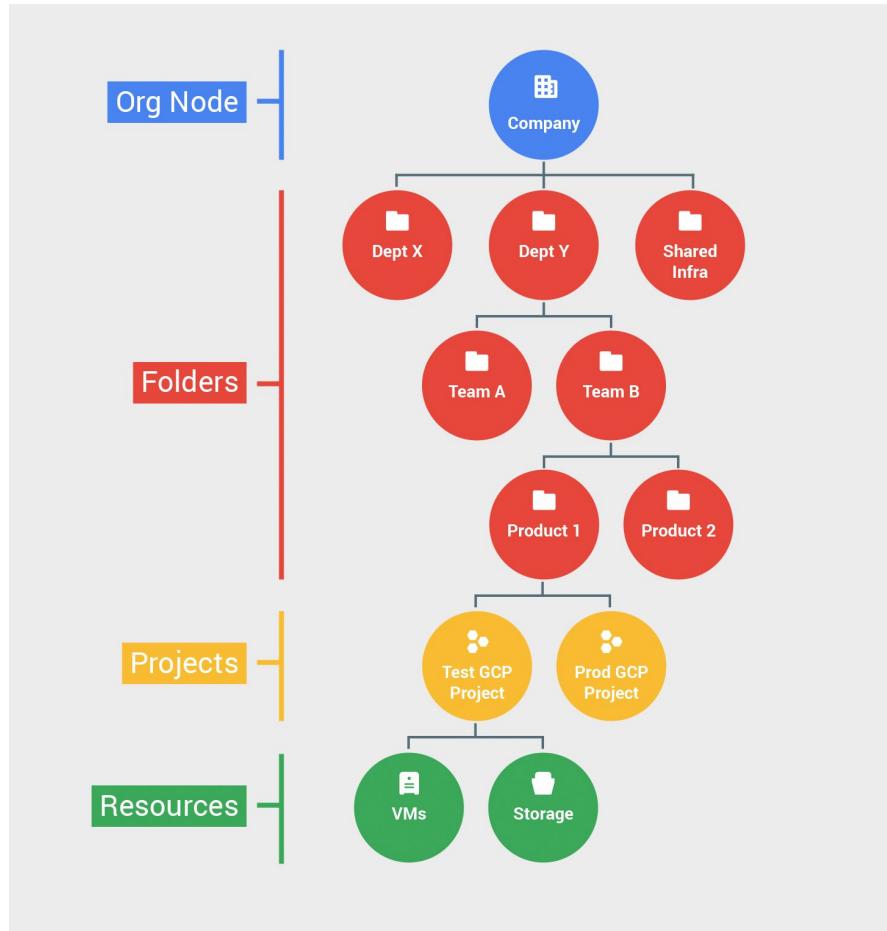
- GCP リソースの集合体
- IAM を適用することが可能
- 個々のプロジェクトは分離されている



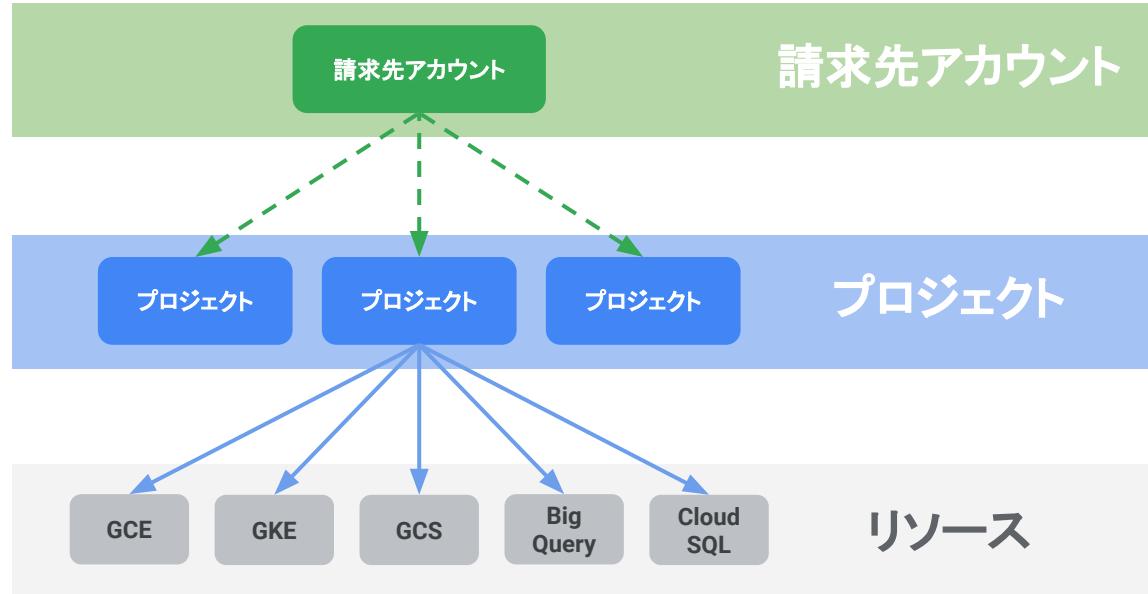
組織とリソース階層

- 組織
 - GCP リソース階層のルートノード
- フォルダー
 - 組織構造に合わせてリソースをグルーピング
- プロジェクト
 - リソースを分離する境界

IAM ポリシーは、下の階層に継承される



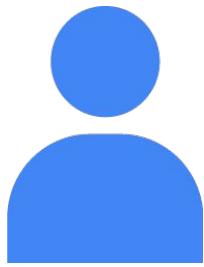
請求先アカウント、プロジェクト、リソースの関係



- プロジェクトには必ず 1 つの請求先アカウントが紐づく
- プロジェクトには複数のリソースが紐づく
- リソースの利用料はプロジェクト単位で請求される

IAM

Identity and Access Management (IAM)



どういう操作を

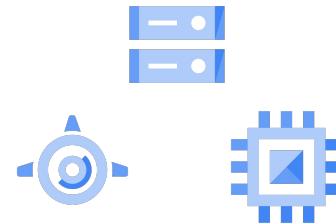
役割

制御
内容

誰が

具体例

Google アカウント
Google グループ
サービスアカウント



何に対して

組織 / フォルダ
プロジェクト
サービスリソース



どういう条件下で

IP アドレス
時間

IAM の役割 (ロール)

基本の役割

Primitive Roles

- 幅広いアクセス権
- 複数サービスに対する役割
- 3 つの役割:
 - オーナー
 - 編集者
 - 閲覧者

事前定義された役割

Predefined Roles

- 狹い範囲のアクセス権
- 一つのサービスに対する権限
- 多数の役割:
 - BigQuery 管理者
 - Compute インスタンス管理者
 - Compute ネットワーク管理者

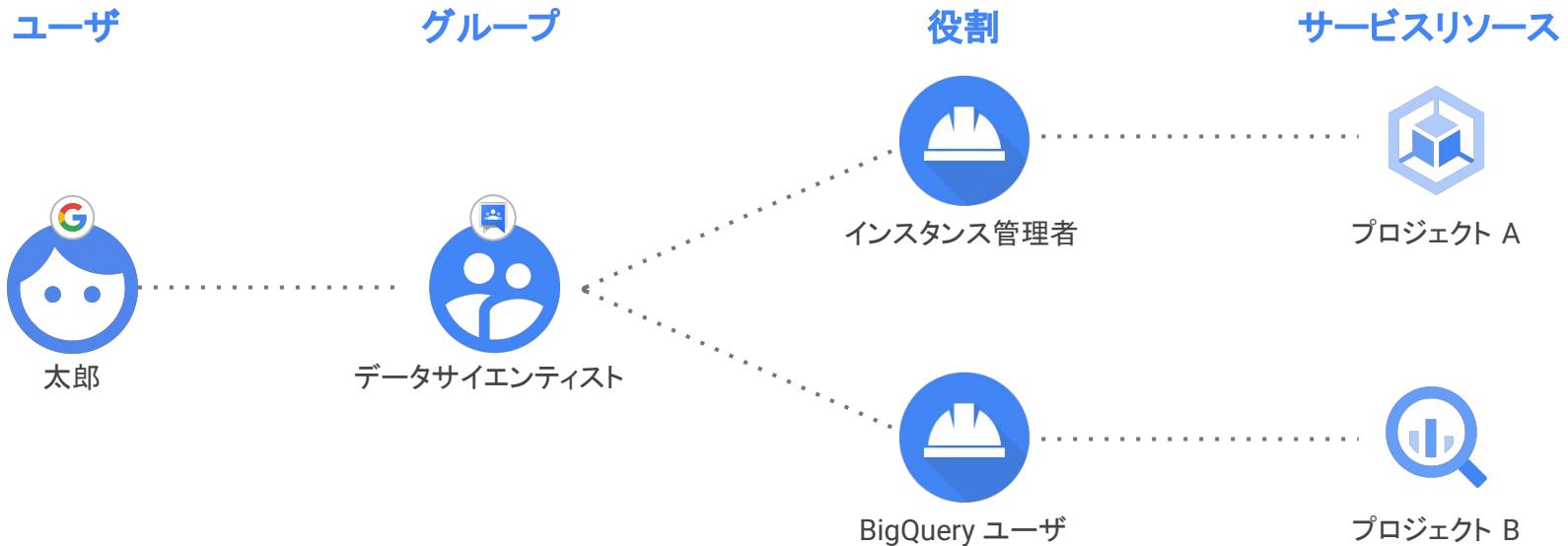
カスタムの役割

Custom Roles

- 最も詳細なアクセス権
- 最初から作成するか、既存の役割から編集して作成する
- 例:
 - 外部 IP アドレスを付与する権限を除外した Compute インスタンス管理者

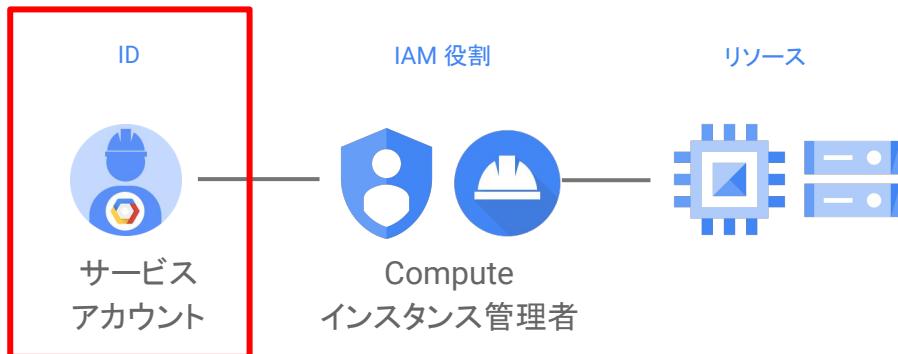
より詳細な権限設定が可能

IAM - グループに対して役割を割り当てる



サービスアカウント

個々のユーザーではなく、アプリケーションや
仮想マシン(VM)に付与する特別な Google アカウント



The screenshot shows the "Create Service Account" dialog with the following fields and options:

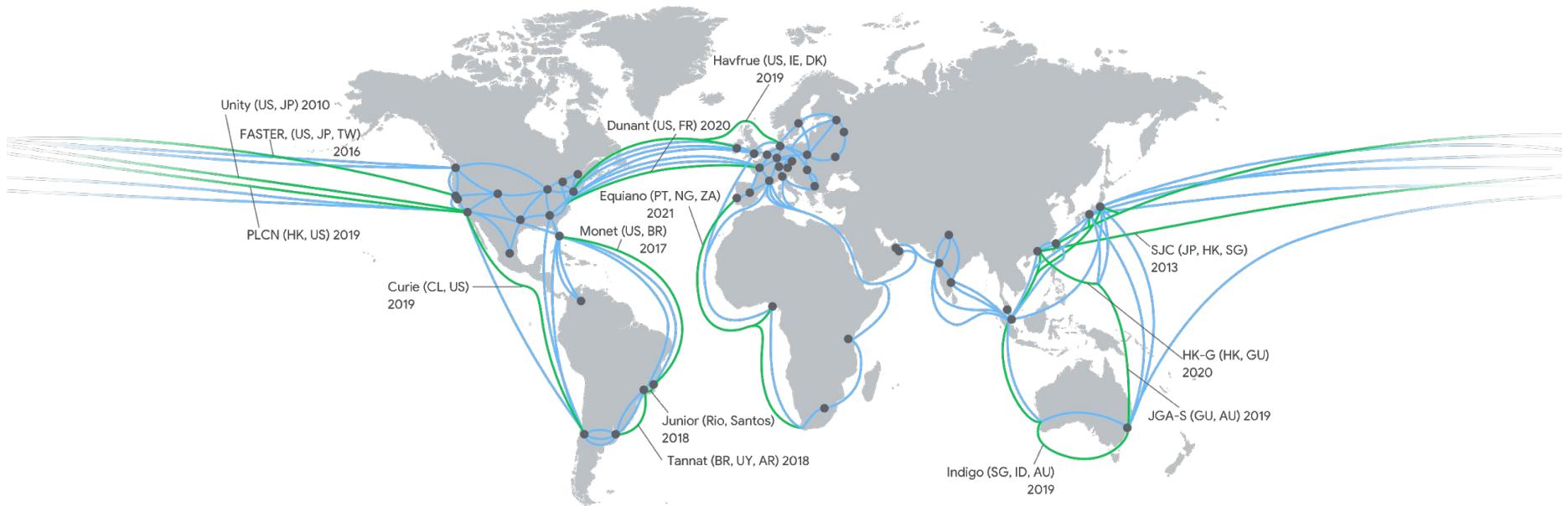
- サービスアカウント名を入力** (Input Service Account Name): A text input field where "service-account" is typed.
- 役割を選択** (Select Role): A button that opens a dropdown menu.
- サービスアカウントの作成** (Create Service Account): The main title of the dialog.
- サービスアカウント名** (Service Account Name): A placeholder text field.
- サービスアカウント ID** (Service Account ID): A placeholder text field ending in ".iam.gserviceaccount.com".
- 新しい秘密鍵を提供** (Provide new key): A checkbox that is unchecked.
- 新しい秘密鍵の提供** (Provide new key): A descriptive text explaining that a file containing the key will be downloaded.
- Project**: A dropdown menu showing a list of projects:
 - App Engine
 - BigQuery
 - Billing
 - Cloud Composer
 - Cloud DLP
 - Cloud Datajobs
 - Cloud Functions
 - Cloud IAP
 - Cloud IoT
 - Cloud KMS
 - Cloud SQL

リージョンとVPC

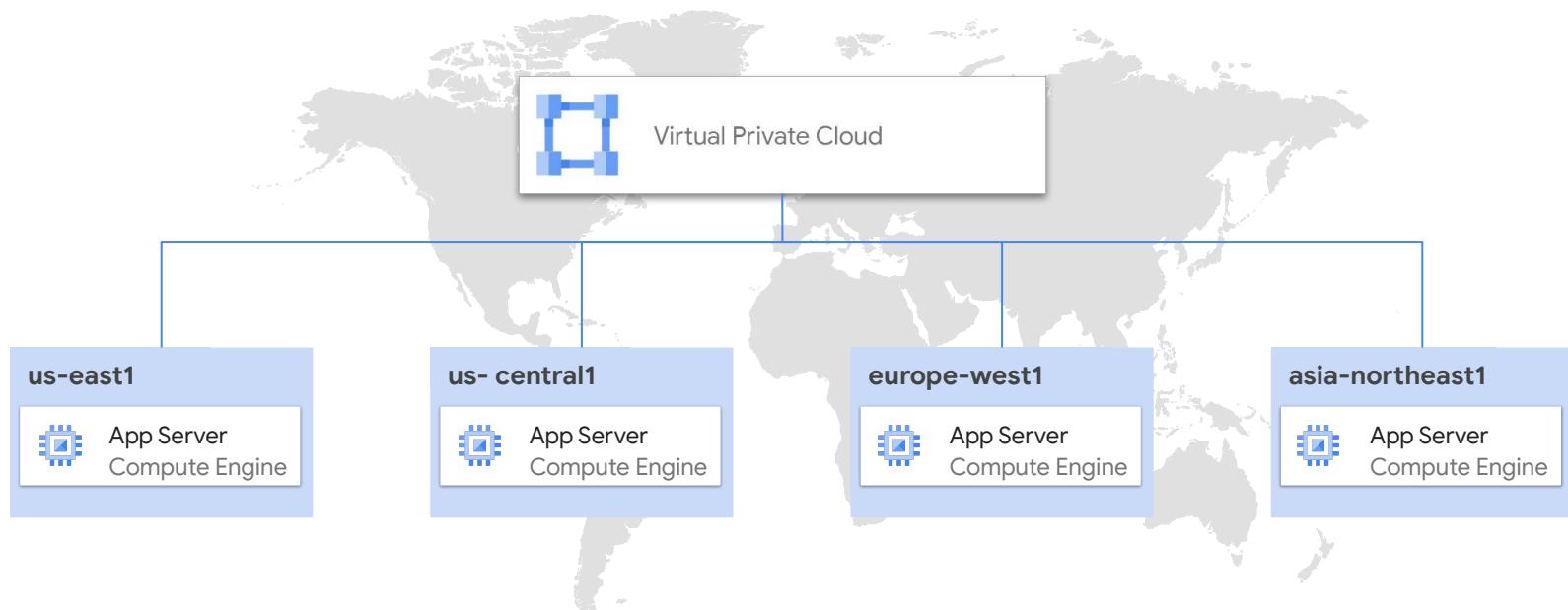
24 Regions, 73 Zones



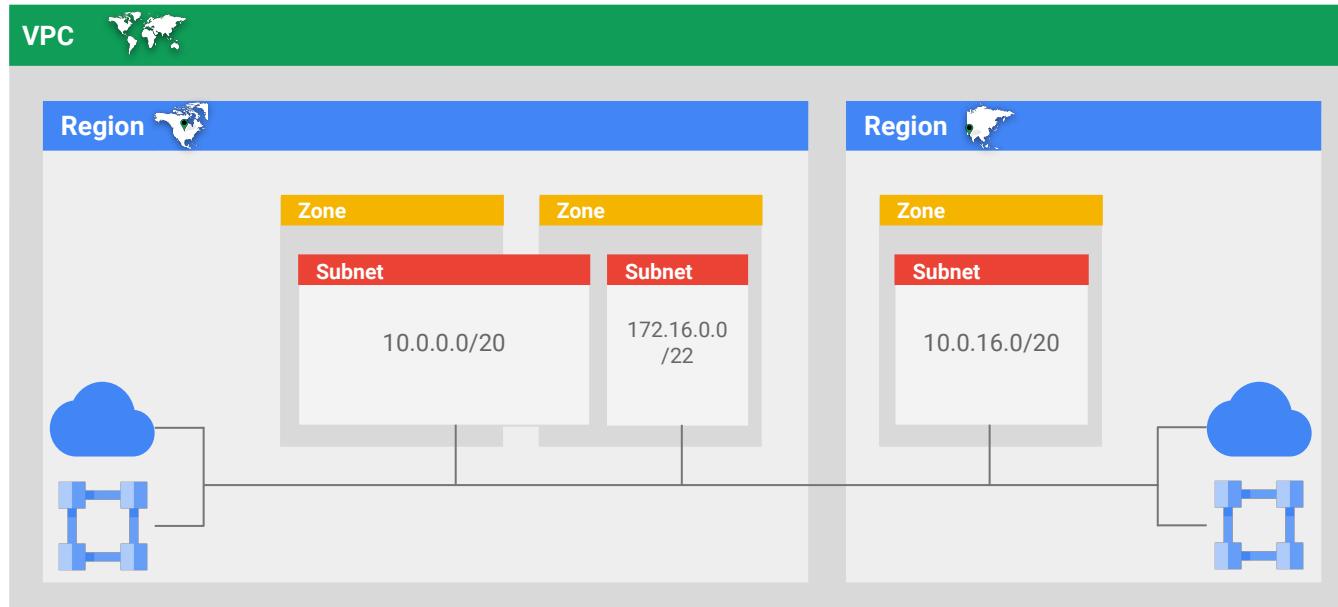
Google Cloud のネットワーク



グローバルなプライベート ネットワーク



簡略イメージ



Tips

Google Cloud Console へのアクセス

Google Cloud Platform を利用するには、Cloud Platform コンソールにアクセスします。

Cloud Platform コンソール:

<https://console.cloud.google.com/>

The screenshot shows the Google Cloud Platform dashboard for the project "yutah-catchup". The dashboard includes sections for Project info, Resources (App Engine, Compute Engine, Cloud Storage, BigQuery, Cloud SQL), Compute Engine metrics (CPU usage over time), Google Cloud Platform status (All services normal), Billing (total \$675.64), and Error Reporting.

Project info
yutah-catchup
Project ID: yutah-catchup
#210236062154

Compute Engine
CPU (%) -
Feb 14, 3:06 PM Feb 14, 4:06 PM
CPU: 7.83

Resources

- App Engine: 5 versions
- Compute Engine: 45 instances
- Cloud Storage: 8 buckets
- BigQuery: 3 datasets
- Cloud SQL: 2 instances

Compute Engine
CPU (%) -
Feb 14, 3:06 PM Feb 14, 4:06 PM
CPU: 7.83

Google Cloud Platform status
All services normal

Billing
\$675.64
Approximate charges so far this month

Error Reporting
No sign of any errors. Have you set up Error Reporting?



コマンドラインツールの選択肢

用途に応じた選択肢

- **Google Cloud Shell を利用**
 - ブラウザから手軽にコマンドラインツールを実行
 - ブラウザからインスタンスに SSH
- **gcloud SDK をローカルにインストール**
 - ローカルのマシンのターミナルから GCP リソースを操作
 - 開発を行う際、手元のターミナルを使う際に用いる

gcloud コマンド

GCP リソースを操作する CLI

- インストールされているコンポーネントを確認する
 - `$ gcloud components list`
- コンポーネントをアップデートする
 - `$ gcloud components update`
- 新規にコンポーネントをインストール(アップデート)する
 - `$ gcloud components update app-engine-python`
- コンポーネントを削除する
 - `$ gcloud components remove bq`

Operations

サーバーレスの統合監視プラットフォーム

Monitoring や Logging、APM など豊富な機能を一括で提供



Cloud Monitoring



Cloud Debugger



Cloud Trace



Cloud Logging



Error Reporting



Cloud Profiler

