

# Serverless に関連するアーキテクチャの話

2021. 6. 10



# Agenda

1. Cloud Run の Cold Start への対応
2. Database の選択と接続
3. Google Cloud Load Balancing ( GCLB )と組み合わせて利用する
4. Serverless batch architecture
5. GKE Autopilot

# Agenda

1. Cloud Run の Cold Start への対応
2. Database の選択と接続
3. Google Cloud Load Balancing ( GCLB )と組み合わせて利用する
4. Serverless batch architecture
5. GKE Autopilot

## Cloud Run の設定で回避できること

- 最小インスタンスの設定を使用する。
- Warm up されている状態を維持することで、コールドスタートを回避することが可能。

※ ただし、サービスがリクエストを処理していない場合にも**コストが発生する**ので注意が必要。

## アプリケーションの作り自体を最適化すること

- 遅延初期化を行う、利用頻度の低いオブジェクトは利用するタイミングで初期化する。
- 起動時に不要なライブラリの読み込みは行わない。
- データベースへの接続に利用する Connection pool などは Global 変数を活用し、不用意にリクエストごとに初期化しない。

## アーキテクチャで最適化できること

- 小さなベースイメージを使用する。
- Memorystore を活用し、データのキャッシュをオフロードする。

## 参考リンク

Cloud Run の応答時間を最適化する 3 つの方法

<https://cloud.google.com/blog/ja/products/gcp/3-ways-optimize-cloud-run-response-times>

最小インスタンスの使用

<https://cloud.google.com/run/docs/configuring/min-instances>

全般的な開発のヒント

[https://cloud.google.com/run/docs/tips/general#using\\_minimum\\_instances\\_to\\_reduce\\_cold\\_starts](https://cloud.google.com/run/docs/tips/general#using_minimum_instances_to_reduce_cold_starts)

# Agenda

1. Cloud Run の Cold Start への対応
2. Database の選択と接続
3. Google Cloud Load Balancing ( GCLB )と組み合わせて利用する
4. Serverless batch architecture
5. GKE Autopilot



# Google Cloud のデータベース 関連ソリューション

移行に適した  
クラウド環境と  
ベアメタル環境



Bare Metal  
Solution



Compute  
Engine

クラウド上の仮想マシンと  
オンプレミスのベアメタル環境

キャッシュ



Cloud  
Memorystore

マネージド  
Redis &  
memcached

移行に適した  
データベース



Cloud  
SQL

マネージド  
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  
(Apache HBase 互換)

モダナイズに適した  
クラウドネイティブ DB



Cloud  
Spanner

スケーラブルで  
可用性の高い  
ミッション  
クリティカル  
RDBMS



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

OSS パートナー  
マネージド サービス



redis



mongoDB.



influxdb



elastic



DATASTAX



neo4j

confluent



Database  
Migration Service

オンプレミス、他クラウド、  
on GCE からの DB 移行サービス

# Google Cloud のデータベース 関連ソリューション

移行に適した  
クラウド環境と  
ベアメタル環境



Bare Metal  
Solution



Compute  
Engine

クラウド上の仮想マシンと  
オンプレミスのベアメタル環境

キャッシュ



Cloud  
Memorystore

マネージド  
Redis &  
memcached

移行に適した  
データベース



Cloud  
SQL

マネージド  
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  
(Apache HBase 互換)

モダナイズに適した  
クラウドネイティブ DB



Cloud  
Spanner

スケーラブルで  
可用性の高い  
ミッション  
クリティカル  
RDBMS



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

データ  
ウェアハウス

OSS パートナー  
マネージド サービス



redis



mongoDB.



influxdb



elastic



DATASTAX



neo4j



confluent



Database  
Migration Service

オンプレミス、他クラウド、  
on GCE からの DB 移行サービス

# Cloud Firestore

## フルマネージドなサーバーレスドキュメント データベース

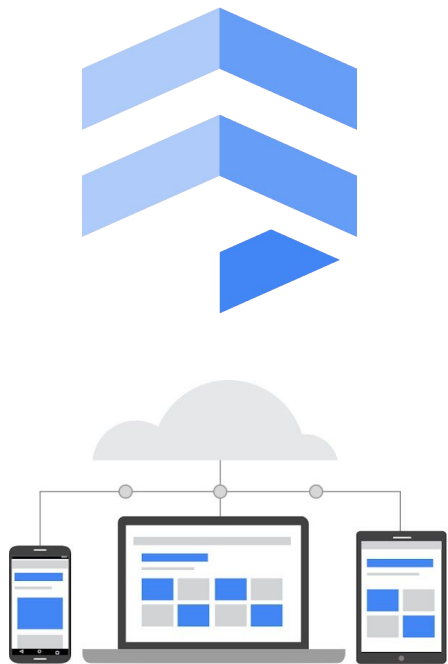
モバイルアプリ、ウェブアプリ、IoT アプリのデータを、ドキュメントとコレクションを組み合わせた構造で保存

## サーバーレスで開発期間を短縮

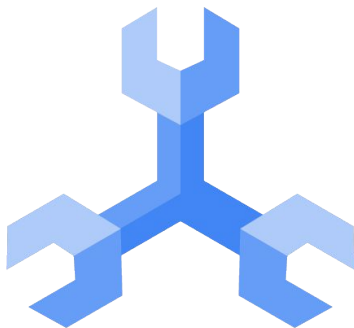
モバイル クライアントまたはウェブクライアントから Cloud Firestore に直接通信可能

## リアルタイム同期とオフライン対応

mBaaS である Firebase と親和性が高く、バックエンドのデータが変更されたとき、アプリケーションをほぼリアルタイム自動更新したり、アプリケーションのオフライン更新と復帰後の自動同期機能も提供



# Cloud Spanner



## フルマネージドなリレーショナルデータベース

リレーショナル データベースのようテーブル構造に対してSQL とトランザクションをサポートし、NoSQL のようなスケーラビリティを持つ

## エンタープライズ向けミッションクリティカル

複数ゾーンにまたがって分散が可能で、最大99.999% の可用性 SLA を提供し、メンテナンスなどによる計画ダウンタイムも一切なし



性能



SQL



スケーラビリティ



運用管理

# Cloud SQL



**MySQL**  
**PostgreSQL**  
**SQL Server**  
をサポート

## フルマネージドなリレーショナルデータベース

高可用性、バックアップ、暗号化、パッチ適用、容量増加などが組み込まれた RDBMS のマネージドサービス

## Google Cloud の各種サービスとの容易な統合

クライアントやドライバーを用いた接続だけでなく  
Google Cloud の各種サービスと容易に連携可能

## 透過的なホストメンテナンス

Compute Engine のライブマイグレーション技術により  
ホストメンテナンスが透過的に行われ、  
プロバイダ側メンテナンスによるお客様負荷を軽減

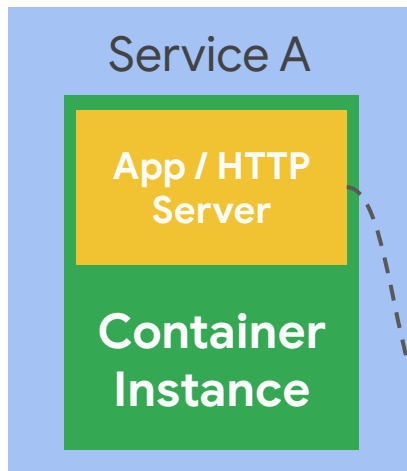
# Cloud Run から Cloud SQL を利用する

- Cloud SQL Auth Proxy を利用して、Cloud SQL に接続する。
- Serverless VPC Access を利用して、Private IP で Cloud SQL に接続する。

# 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 Auth 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
```



Cloud SQL

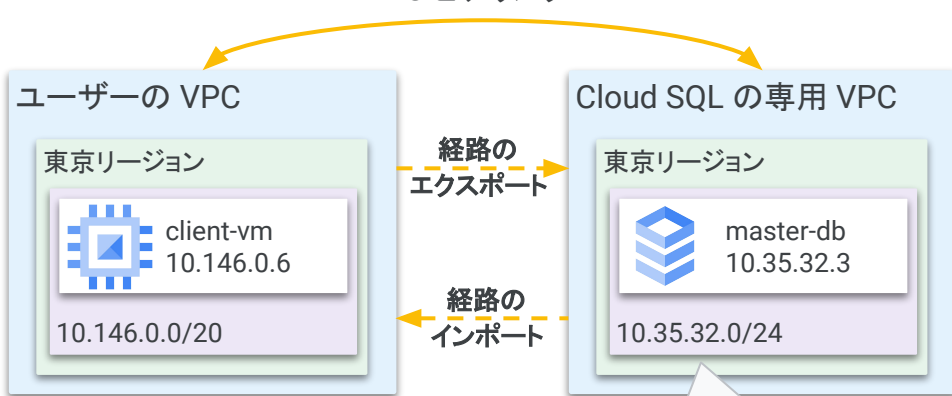
app.py

```
import MySQLdb

db = MySQLdb.connect
(unix_socket='/cloudsql/[CONNECTION NAME]',
user=CLOUDSQL_USER,
passwd=CLOUDSQL_PASSWD)
```

# Cloud SQL のプライベート IP 接続の仕組み

VPC ピアリング



Cloud SQL のインスタンスは、専用の VPC 内に構築される

VPC ピアリングにより、ユーザーの VPC と Cloud SQL 専用 VPC でプライベート IP の **ルートの交換(経路交換)**がされている

プライベート IP のレンジは初回構成時に自動生成またはユーザーが指定可能

VPC ネットワーク ピアリング [+ ピアリング接続を作成](#) [更新](#)

表をフィルタリング

<input type="checkbox"/>	名前 ↑	VPC ネットワーク	ピアリングした VPC ネットワーク	ピアリングしたプロジェクト ID	ステータス
<input type="checkbox"/>	cloudsql-mysql-googleapis-com	default	cloud-sql-network-201242915176-9b353d612b7c7fb0	speckle-umbrella-53	有効
<input type="checkbox"/>	cloudsql-postgres-googleapis-com	default	cloud-sql-network-201242915176-ca89a34bba3b05b2	speckle-umbrella-pg-11	有効
<input type="checkbox"/>	redis-peer-459920808437	default	default-redis-e619463a-57d4-4053-a70d-b8c275ad4d49	n531500bb3a070714-tp	有効

VPC ネットワークピアリングの画面には Cloud SQL の VPC が表示されている



# Cloud SQL のプライベート IP 接続の仕組み 2

## 設定オプション

### 1 接続

データベース インスタンスへの接続方法を選択します。

セキュリティを強化するには、Cloud SQL Proxy を使用してインスタンスに（作成後に）接続することを確認してください。 [詳細](#)

#### ☒ プライベート IP

プライベート IP 接続に追加の API と権限が必要です。この機能を有効にする、または使用するには、組織の管理者への依頼が必要になる場合があります。現在は、プライベート IP 接続を有効にすると、削除できなくなります。

#### 関連付けられたネットワーキング

非公開接続を作成するネットワークを選択します

default

#### マネージド サービス ネットワーク接続 <sup>②</sup>

割り当てられた IP 範囲を指定してサービス接続を作成します。

##### ☐ IP 範囲の選択

割り当てられた範囲がありません。新しいカスタム IP 範囲を割り当ててください。 [詳細](#)

##### ☒ 自動的に割り当てられた IP 範囲を使用する

Google Cloud Platform は接頭辞長 20 の IP 範囲を自動的に割り当て、**google-managed-services-default** という名前を使用します。

割り当てて接続

キャンセル

☐パブリック IP

Cloud SQL で初めてプライベートIPを設定しようとした時の画面

← VPC ネットワークの詳細 [編集](#) [VPC ネットワークを削除](#)

default

説明  
Default network for the project

サブネット作成モード  
自動サブネット

動的ルーティングモード  
グローバル

DNS サーバー ポリシー  
なし

サブネット 静的な内部 IP アドレス ファイアウォール ルール VPC ネットワーク ピアリング [プライベート サービス接続](#)

[サービスに割り当てられた IP 範囲](#) [サービスへのプライベート接続](#)

IP 範囲の割り当て [解放](#)

<input type="checkbox"/> 名前	内部 IP 範囲	サービス プロデューサー	接続名
<input type="checkbox"/> google-managed-services-default	10.35.32.0/20	Google Cloud Platform	servicenetworking-googleapis-com cloudsql-postgres-googleapis-com cloudsql-mysql-googleapis-com

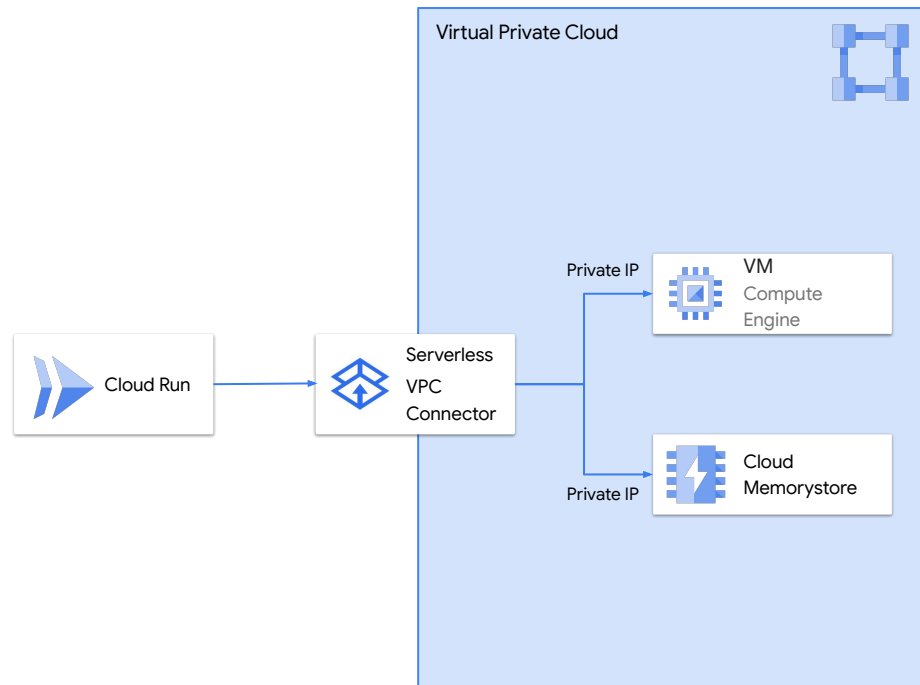
VPC ネットワークの詳細からレンジの確認または任意のレンジを作成可能

初回設定時に自動または手動でレンジの割り当てが可能

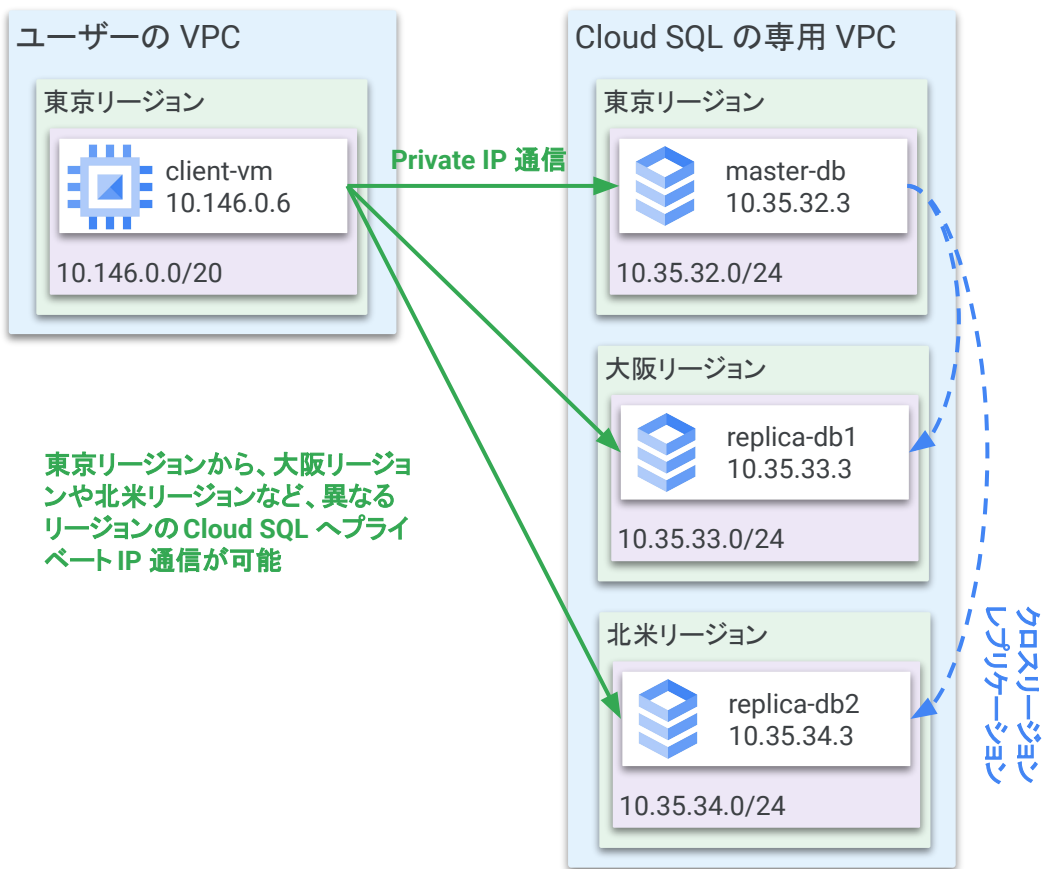
オンプレミスから Cloud SQL へ接続する際など、プライベート IP アドレスの重複を避ける必要がある場合は注意

# 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



# リージョン間のプライベート IP 接続



Cloud SQL において、異なるリージョン間でのプライベート IP による通信ができるようになった

そのため、利便性や可用性がさらに向上した

## 参考リンク

Cloud Run から Cloud SQL への接続

<https://cloud.google.com/sql/docs/mysql/connect-run>

プライベート IP を使用してインスタンスに接続する

[https://cloud.google.com/sql/docs/mysql/configure-private-ip#connecting\\_to\\_a\\_n\\_instance\\_using\\_its\\_private\\_ip](https://cloud.google.com/sql/docs/mysql/configure-private-ip#connecting_to_a_n_instance_using_its_private_ip)

サーバーレス VPC アクセス

<https://cloud.google.com/vpc/docs/serverless-vpc-access?hl=ja>

# Agenda

1. Cloud Run の Cold Start への対応
2. Database の選択と接続
3. Google Cloud Load Balancing ( GCLB )と組み合わせて利用する
4. Serverless batch architecture
5. GKE Autopilot

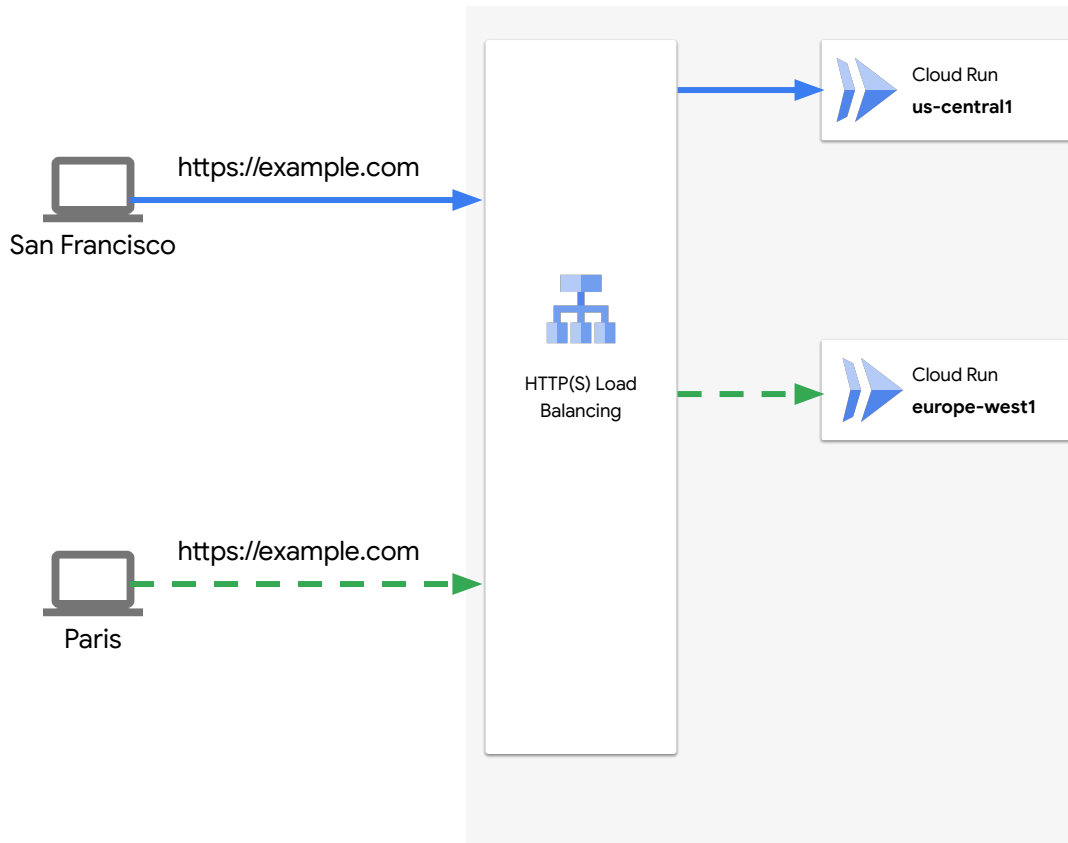
# 複数リージョンの Cloud Run 間で負荷分散

Serverless NEG により、

HTTP(S) Load balancing と連携して

以下を実現

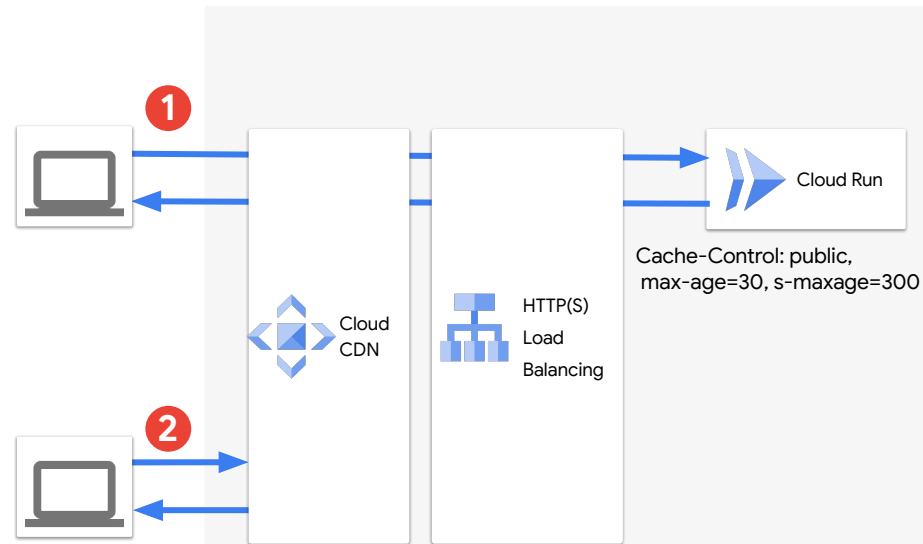
- 複数リージョン の Cloud Run 間での負荷分散
- 最寄りリージョンヘルーティング
- 持ち込みの SSL 証明書使用可能
- パスベースルーティング対応



# 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。



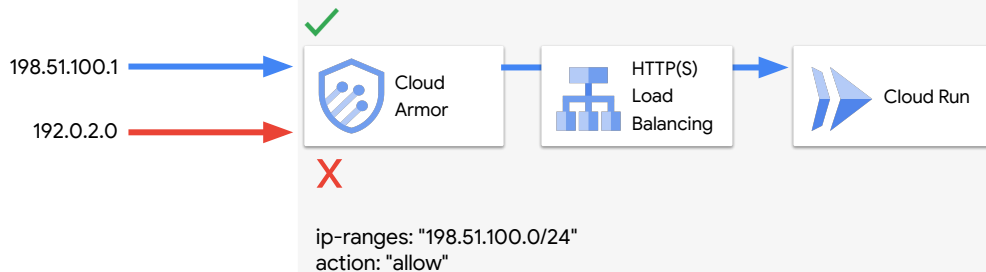
# DDOS対策、ネットワークフィルタリング

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



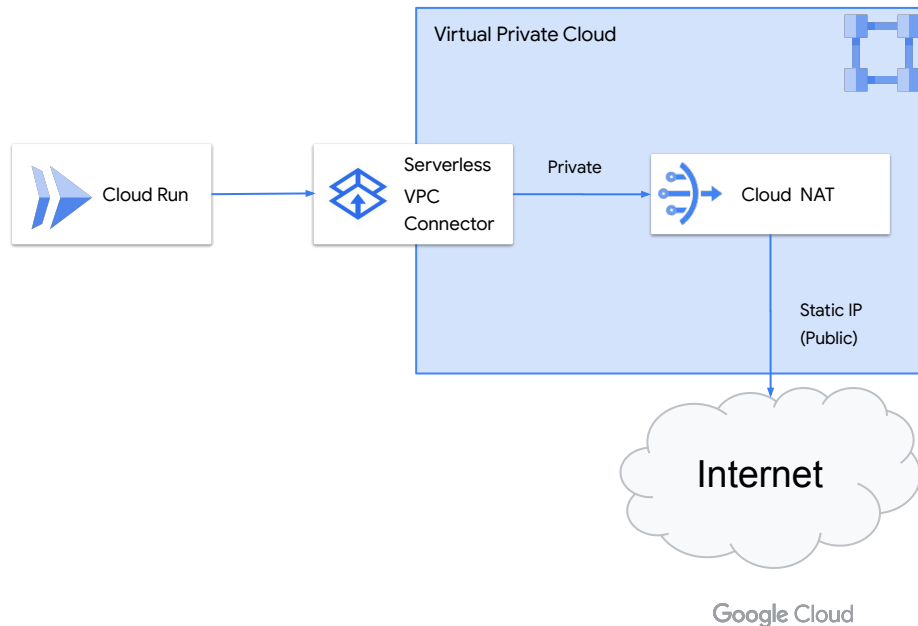


補足 ちょっと本題とはずれますが...

## 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。



## 参考リンク

マルチリージョンの負荷分散を設定する

[https://cloud.google.com/load-balancing/docs/https/setting-up-https-serverless#multi\\_region\\_lb](https://cloud.google.com/load-balancing/docs/https/setting-up-https-serverless#multi_region_lb)

Cloud CDN を有効にする

<https://cloud.google.com/load-balancing/docs/https/setting-up-https-serverless#enabling>

Google Cloud Armor を有効にする

<https://cloud.google.com/load-balancing/docs/https/setting-up-https-serverless#enabling>

外向きの静的 IP アドレス

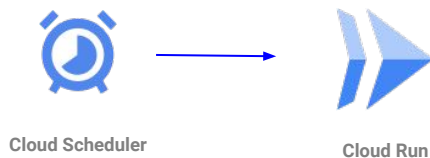
<https://cloud.google.com/run/docs/configuring/static-outbound-ip>

# Agenda

1. Cloud Run の Cold Start への対応
2. Database の選択と接続
3. Google Cloud Load Balancing ( GCLB ) と組み合わせて利用する
4. **Serverless batch architecture**
5. GKE Autopilot

# バッチ処理の定期実行

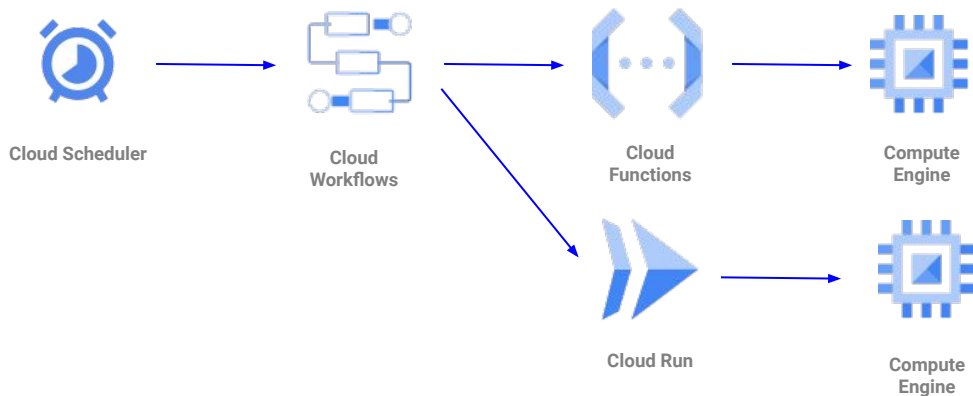
## [ シンプルな処理の実行 ]



※ ポイント

マネージドの Cloud Scheduler を利用する。

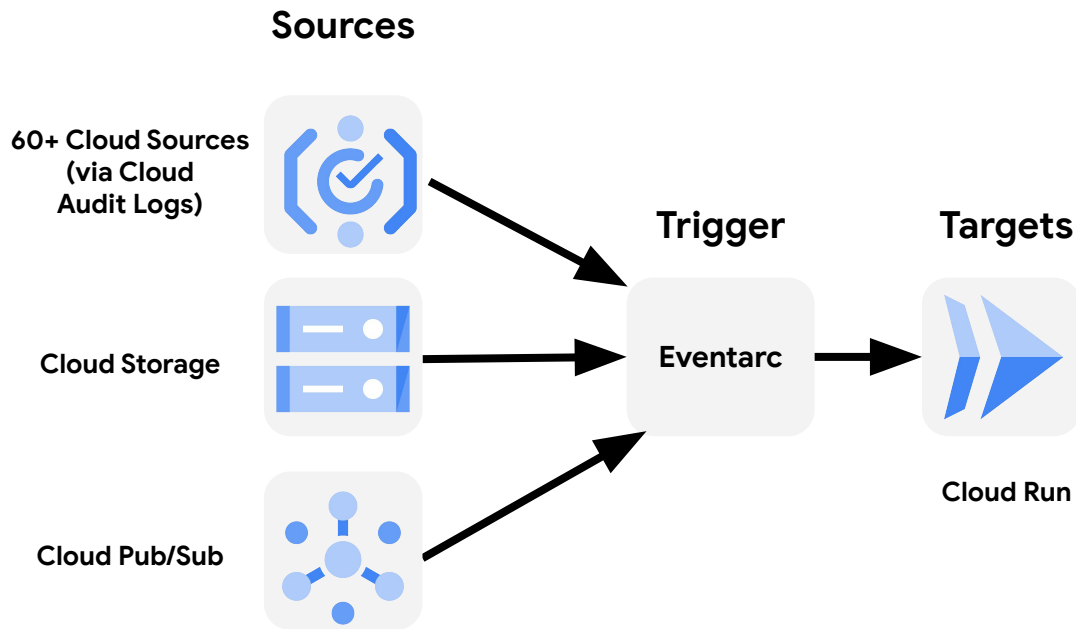
## [ 複数処理を組み合わせたワークフローの実行が可能。 ]



# イベントドリブン アーキテクチャ

Eventarc 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に。



## 参考リンク

スケジュールに従ってサービスを実行する

<https://cloud.google.com/run/docs/triggering/using-scheduler>

Workflows を使用したサービスの接続

<https://cloud.google.com/run/docs/tutorials/workflows>

Serverless workflows in Google Cloud ( Youtube 音声あり)  
<https://www.youtube.com/watch?v=Uz8G8fTwwXs>

Pub/Sub イベントの受信

<https://cloud.google.com/run/docs/events/pubsub>

# Agenda

1. Cloud Run の Cold Start への対応
2. Database の選択と接続
3. Google Cloud Load Balancing ( GCLB )と組み合わせて利用する
4. Serverless batch architecture
5. GKE Autopilot

# Autopilot

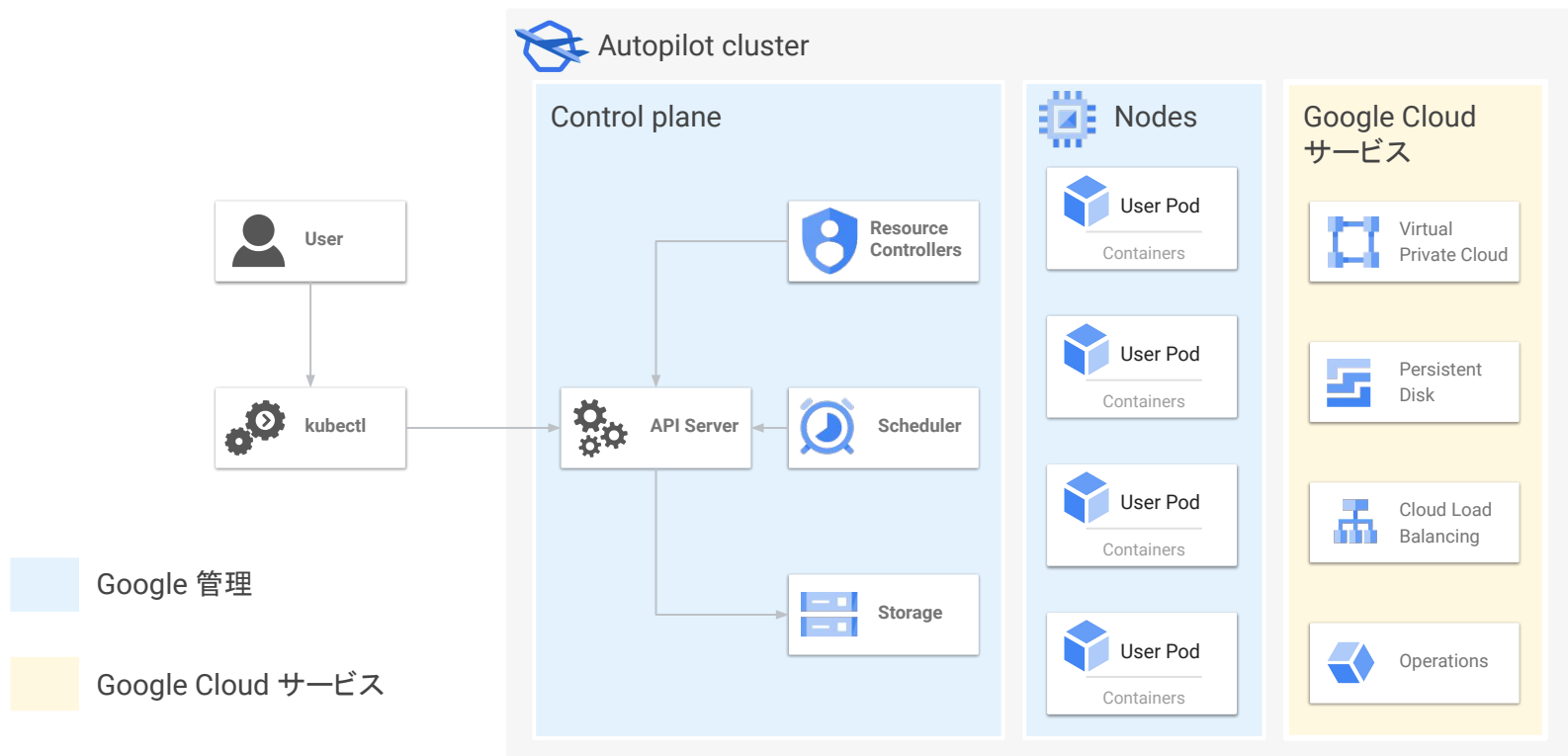
Autopilot = **GKE の新しい運用オプション** ≠ 新プロダクト

- Control plane に加え **Node も Google マネージド**に
- **本番ワークロードに適したベストプラクティス**が適用済み
  - セキュリティ
  - ワークロード
  - 各種設定
- Workload (Pod)ドリブンな世界へ
  - **Pod 単位での課金、Pod への SLA**
- 従来の運用オプションは **Standard** となった





# Autopilot のハイレベル アーキテクチャ



# Autopilot アーキテクチャのポイント

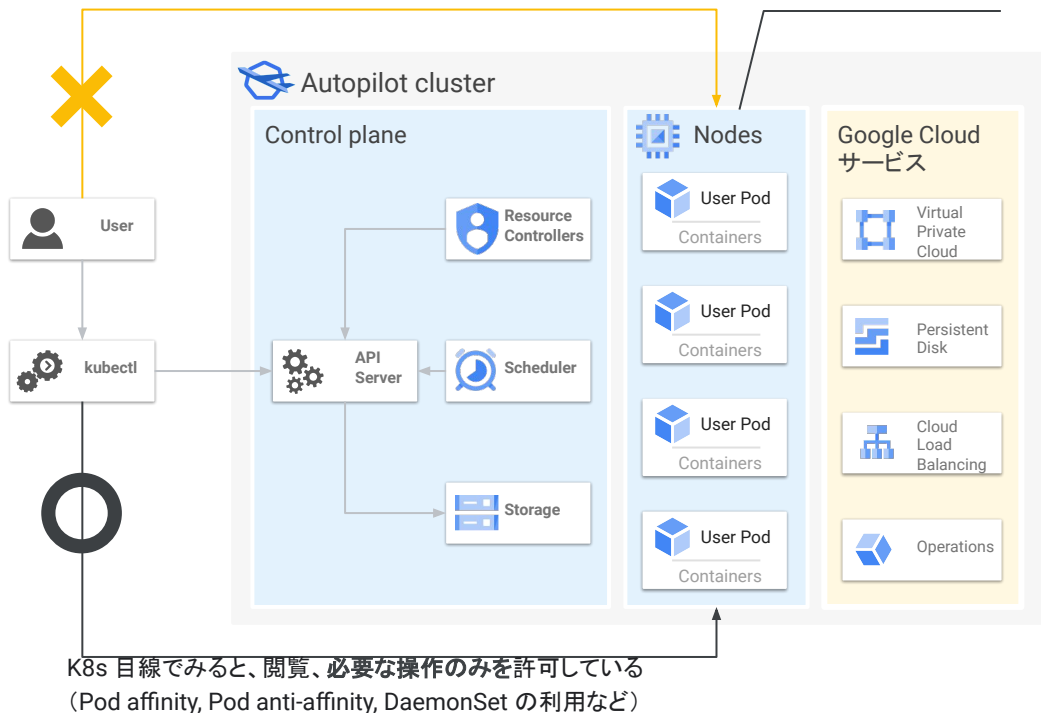
Node へは**直接のアクセス**(SSH、コンソールからの閲覧、操作など)はできないようになっている

Node が存在し  
お客様の VPC に作成される  
さらに各種プラクティスが  
事前に適用済み

## Autopilot は GKE (K8s) の 良いところをそのまま残しつつ

- 運用負荷の低減
- ベストプラクティスの適用
- セキュリティの向上

を実現している




# 参考: Autopilot にはなぜ Node があるのか

## GKE であることを守る

- GKE の利用体験はできる限り維持
  - Node もそのまま利用する
- 透明性を確保  
(✗ ブラックボックス)

## シンプル

- ワークロード (Pod) を中心に
  - 課金、SLA も同様
- ベストプラクティスを最初から組み込み



GKE のユーザー体験を  
もっとシンプルに  
もっとマネージドに  
もっと自動で

## マネージド

- Control plane に加えて Node もマネージドに
  - Google の SRE が運用、管理
- セキュリティも対象

## 自動

- 自動でスケーリング、アップグレード、ヒーリング
- 自動で適切なリソースを確保

GKE であるメリットを最大限に活かすため  
アーキテクチャも **Node を利用した構成を採用**

# Autopilot のもたらす価値

運用が  
より簡単に


Node の管理を Google に任せることで  
利用者は**ワークロードに集中**することが可能に

コストの  
最適化


Node の課金から Pod の課金に変わること  
で**利用リソースに即したコスト**へ最適化

# 例: アプリケーションを本番稼働させるまで

## Standard

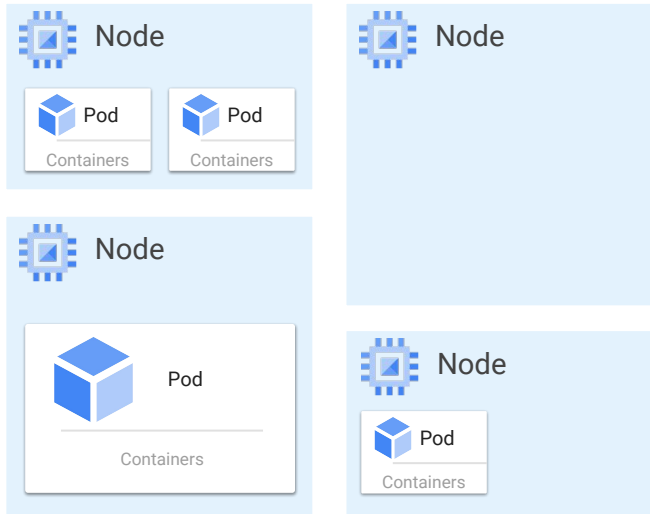
1. 負荷試験
  2. 必要ポッド数の見積もり
  3. 必要 Node の見積もり
  4. K8s 設定の作成
  5. Node のスケーリング設定
  6. デプロイ
- 

## Autopilot

1. 負荷試験
  2. 必要ポッド数の見積もり
  3. K8s 設定の作成
  4. デプロイ
- 

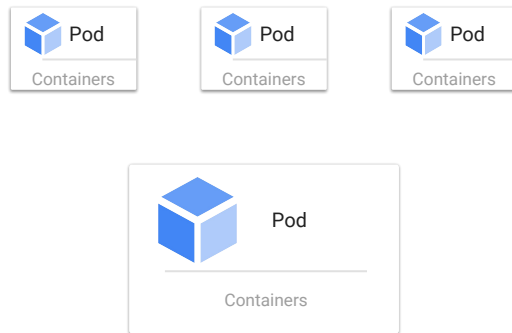
# 例：適切に構成できていない GKE Standard

## Standard



リソースが空いていても  
稼働 Node 分の課金が発生

## Autopilot



稼働 Pod 分のみの課金が発生

# Autopilot の主要な制約、制限

GKE standard と比べ**制約、制限**も存在する

- Node への SSH アクセス不可
- Node の OS は  
Container-Optimized OS with Containerd のみ
- 特権コンテナ利用不可
- Pod に付与する CPU 対 Memory の制限
- いくつかのセキュリティ機能、Add-on が未サポート

その他の制約、制限は[こちら](#)をご参照ください



柔軟な構成は  
**Standard** を選択

# Autopilot に適するユースケース



処理時間がかかっても中断されず  
利用したリソースのみの課金で済むため



Node のスケールアウトには時間がかかるためス  
パイク的な負荷があるシステムは適さない

## Node のチューニングがいないシステム

Node のチューニングはできないため、  
それが必須なシステムは載せることが難しい



自動アップグレードが必須なため、影響が  
小さいステートレスなアプリが適している



# Thank you

