Google Cloud

Google Kubernetes Engine / Cloud Run と Cloud Spanner で実現するスケーラブルなアプリケーション

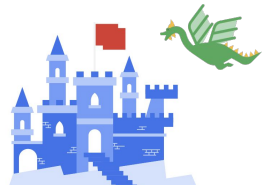
磯 賢大

Google Cloud Customer Engineer

スピーカー自己紹介

磯 賢大
Google Cloud
Customer Engineer

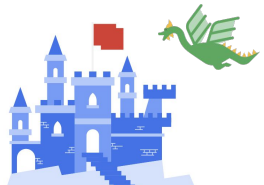
日系 SIer にて、システム エンジニアとしてエンジニアのキャリアを開始する。
キャリアの途中から Cloud Native な技術の楽しさに目覚める。
2020 年 12 月に Google Cloud にジョイン。
ゲーミング チームのカスタマー エンジニアとして、お客様の開発を支援。
OSS へのコントリビューションが最近の趣味。



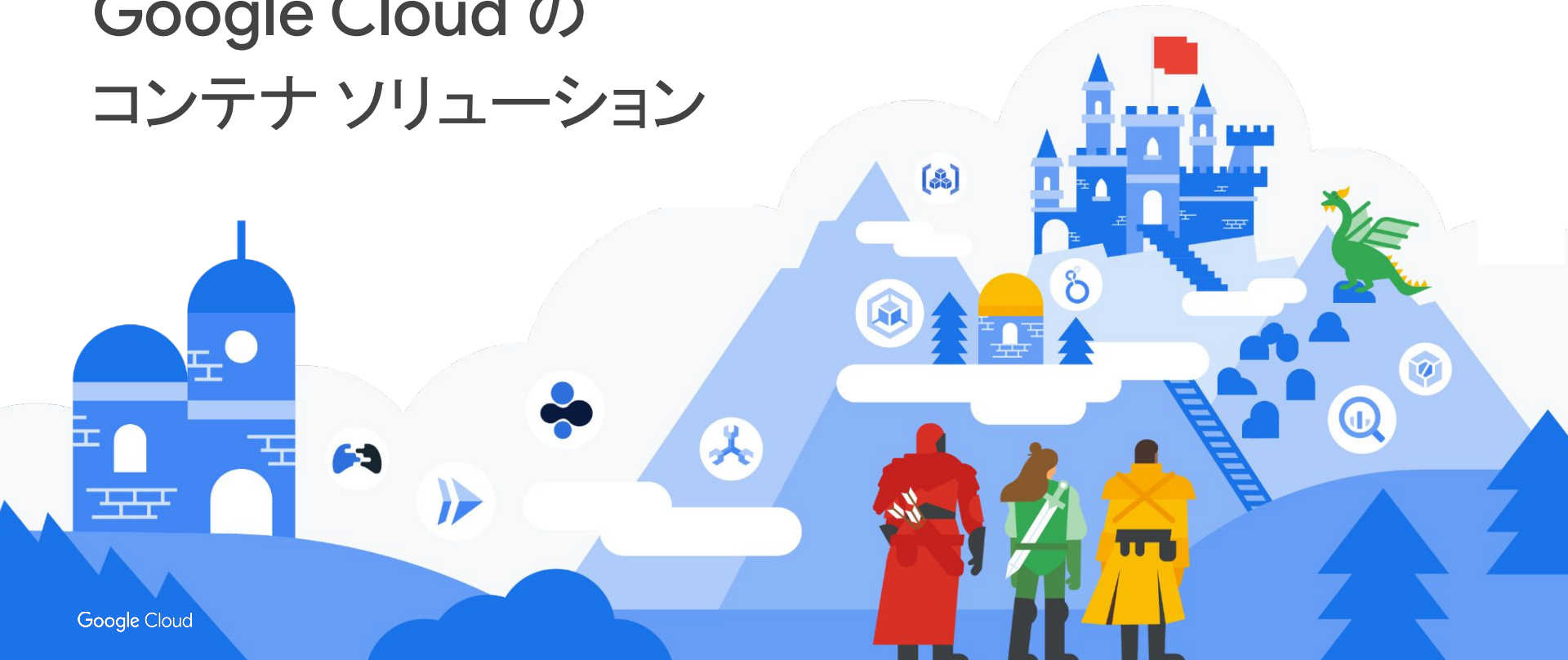
ゲームにおける考慮すべきシナリオ

- 新規ゲームのローンチ時における大量ユーザーの流入
- 毎日のピークタイムにおけるアクティブ ユーザーの一時的な増加
- ゲーム内イベントにおけるユーザー アクセスのスパイク
- ゲームが広く普及することによるユーザー アクセスの増加
- アクセスが落ち着くに合わせて縮退し、リソースやコストも最適化したい

予想されるプレーヤー数に基づいてシステムを設計しテストを行ったとしても、予期しない負荷急増への対応ができていない場合には、信頼性のあるサービスの提供はできなくなる可能性も



Google Cloud の
コンテナ ソリューション

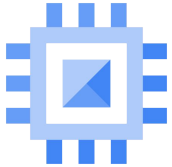


Google Cloud

Google Cloud における Compute の選択肢

マネージド

サーバーレス



Compute Engine

仮想マシン
既存のシステムや
既存の運用

Kubernetes Engine

コンテナ実行基盤のマネージドサービス
アプリケーションの
デプロイ、管理、更新を容易に

App Engine

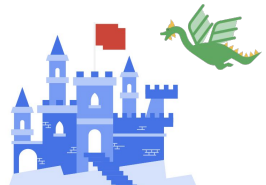
Web アプリ基盤
Web アプリや API 向け

Cloud Run

Web アプリ基盤
Web アプリや API 向け

Cloud Functions

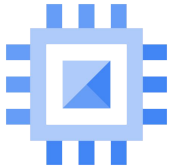
イベント駆動型
個別の機能・軽量の
ロジック



Google Cloud における Compute の選択肢

マネージド

サーバーレス



Compute Engine

仮想マシン
既存のシステムや
既存の運用

Kubernetes Engine

コンテナ実行基盤のマネージドサービス
アプリケーションの
デプロイ、管理、更新を容易に

App Engine

Web アプリ基盤
Web アプリや API 向け

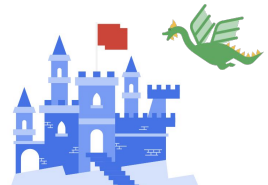
Cloud Run

Web アプリ基盤
Web アプリや API 向け

Cloud Functions

イベント駆動型
個別の機能・軽量の
ロジック

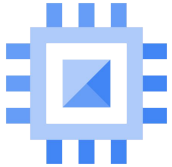
今回のテーマ



Google Cloud における Compute の選択肢

マネージド

サーバーレス



Compute Engine

仮想マシン
既存のシステムや
既存の運用

Kubernetes Engine

コンテナ実行基盤のマネージドサービス
アプリケーションの
デプロイ、管理、更新を容易に

App Engine

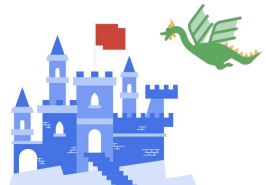
Web アプリ基盤
Web アプリや API 向け

Cloud Run

Web アプリ基盤
Web アプリや API 向け

Cloud Functions

イベント駆動型
個別の機能・軽量の
ロジック

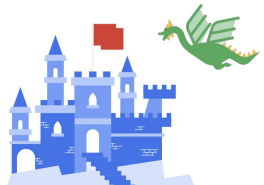


Google Cloud

ゲームにおける考慮すべきシナリオ

- 新規ゲームのローンチ時における大量ユーザーの流入
- 毎日のピークタイムにおけるアクティブ ユーザーの一時的な増加
- ゲーム内イベントにおけるユーザー アクセスのスパイク
- ゲームが広く普及することによるユーザー アクセスの増加
- アクセスが落ち着くに合わせて縮退し、リソースやコストも最適化したい

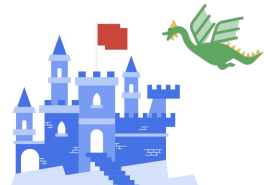
→ 信頼性や可用性を高めるために **スケーラビリティ** が求められる



Google Kubernetes Engine（GKE）

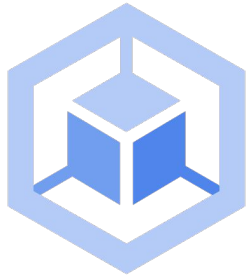
Google が提供するマネージドな Kubernetes サービス

- ワンクリックで Kubernetes を構築
- 運用の手間を軽減するオートヒーリング
  オートアップグレード、リリースチャネル
- ロギング、モニタリングなど Google Cloud の
  各種サービスとのネイティブな連携
- エンタープライズでの利用を前提とした SLA の設定、
  HIPAA、PCI-DSS といったセキュリティ基準への準拠

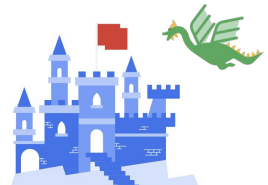


なぜゲーム業界で GKE が強みを持つのか

GKE は Kubernetes そのものが持つオートスケーリング /オートヒーリングなどの
機能の他にも様々な利点を持つ

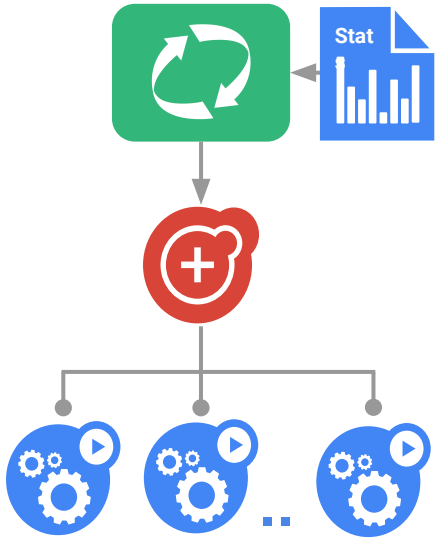


- Horizontal Pod Autoscaler/Cluster Autoscaler による素早い**オートスケーリング**
- LivenessProbe/ReadinessProbe による**オートヒーリング** /**オートサービスイン**
- **コンテナネイティブ 負荷分散** によるオーバヘッドの削減
- **MultiClusterService, MultiClusterIngress** を使った MultiCluster 連携
- **Gateway Controller** によるきめ細かいトラフィック制御 ※ 2022/06 時点で Preview
- **Gaming OSS** との親和性

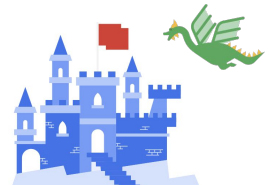


Horizontal Pod Autoscaler（HPA）によるオートスケール

HPA によって Pod の スケールアウト・スケールインを自動で実行する。
CPU やメモリーを閾値としたオートスケーリングの他にも、カスタム指標を使うことも可能。



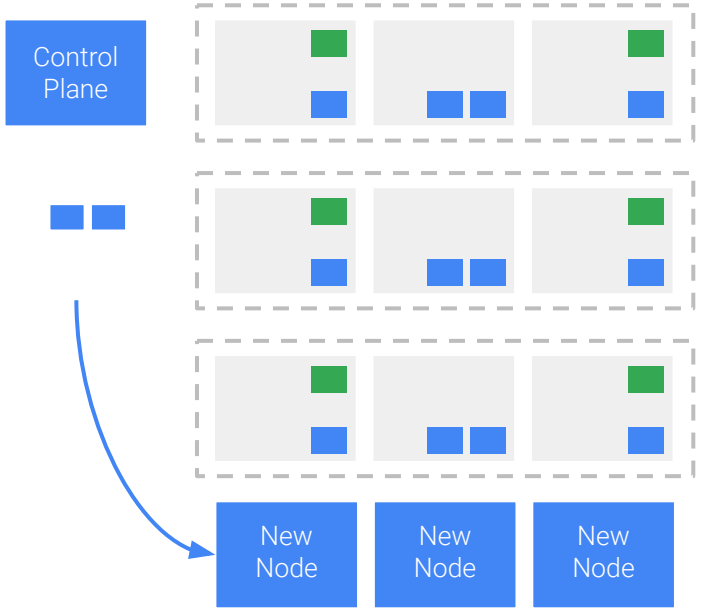

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: custom-metric-sd
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: custom-metric-sd
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Pods
    pods:
      metric:
        name: custom-metric
      target:
        type: AverageValue
        averageValue: 20
```

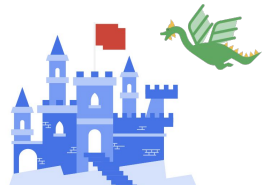


Cluster Autoscaler（CA）によるオートスケール

Cluster Autoscaler は、リソース（CPU /メモリ）が不足して Pod のスケジュールに失敗した場合にノードを追加する。

Kubernetes は requests フィールドを使用して Pod をスケジュールするため、スケーリングが発生します。

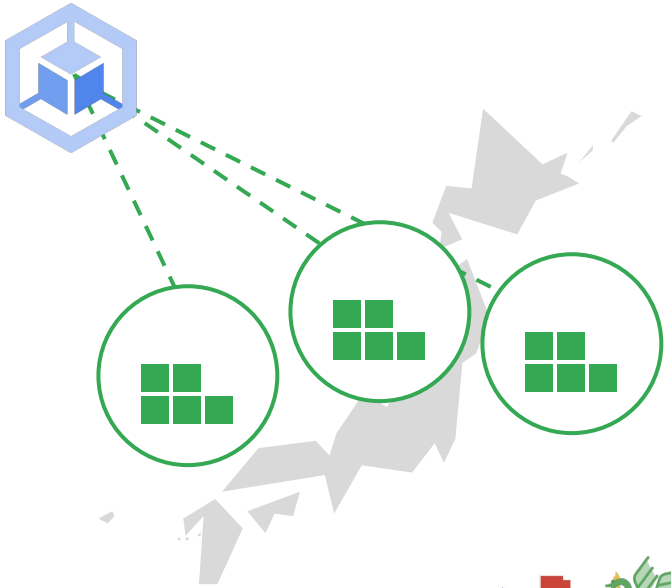


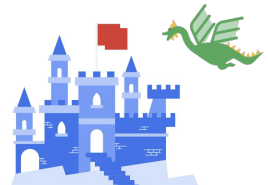


Cluster Autoscaler（CA）による Multi Zone スケール

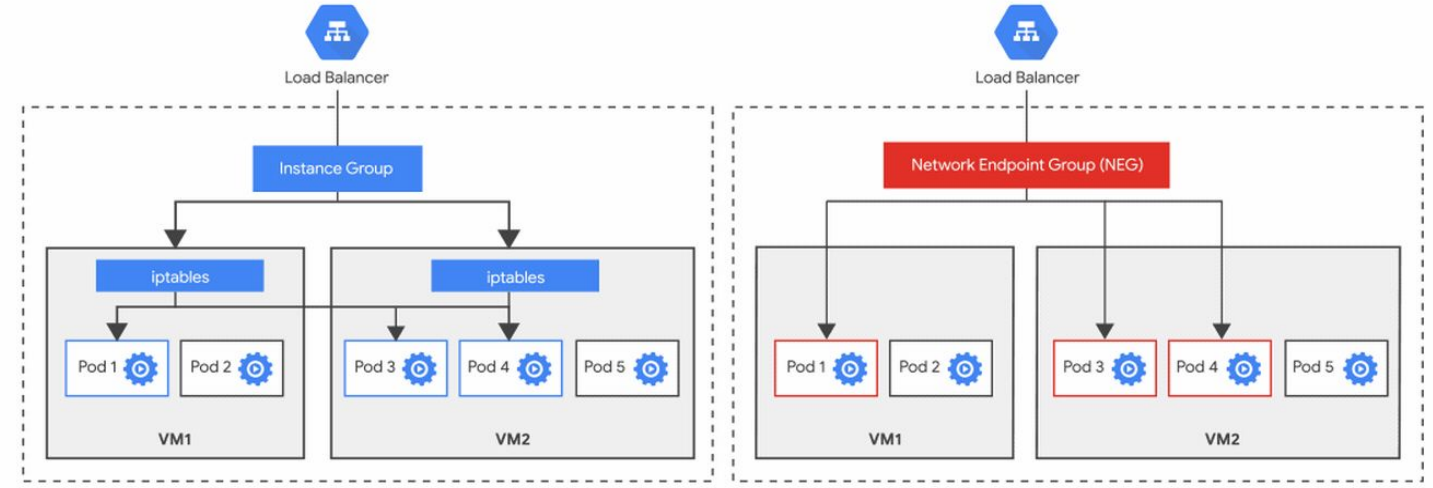
異なるゾーン間でスケールアップする場合、Cluster Autoscaler は各 NodePool のサイズのバランスをとる。

これにより、リージョン内の複数のゾーンにトラフィックを分散するときに、ノードの数が不均一になるのを防ぐことができる。



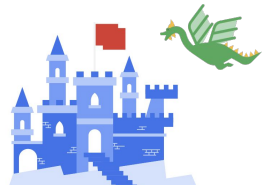


コンテナ ネイティブ負荷分散によるオーバヘッド削減

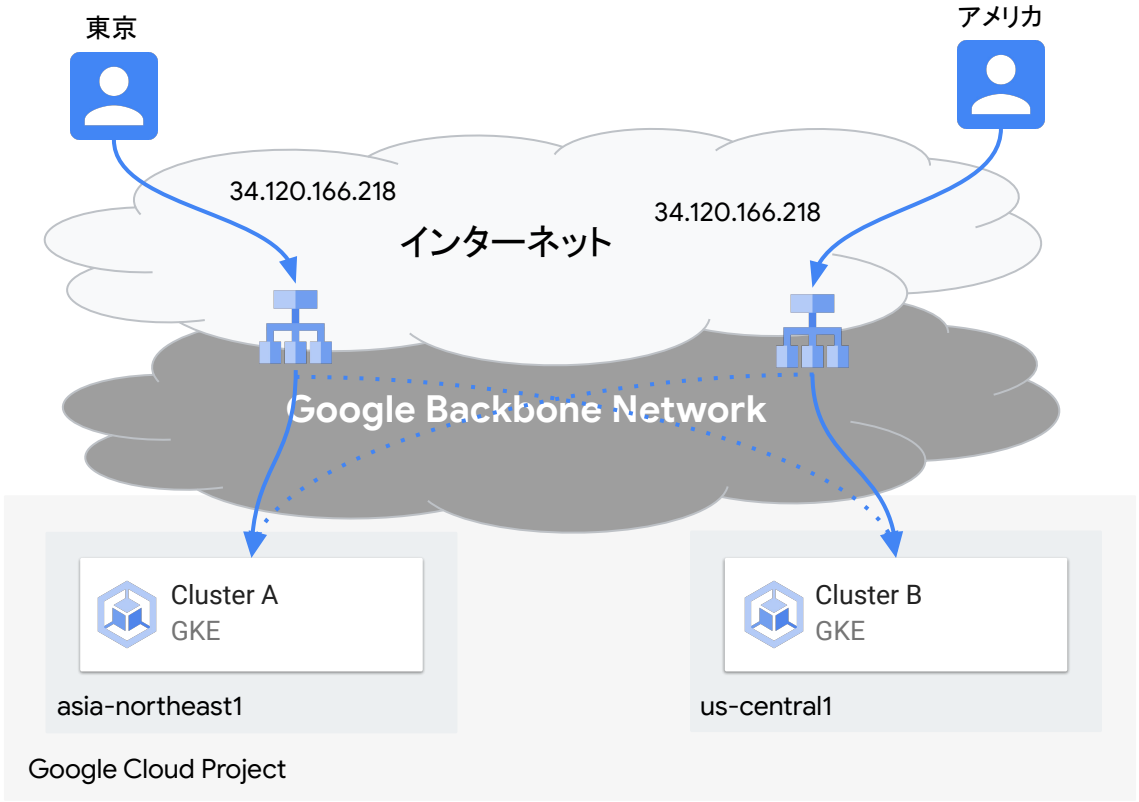



```
kind: Service
metadata:
  annotations: cloud.google.com/neg: "true"
```

※ デフォルトで有効



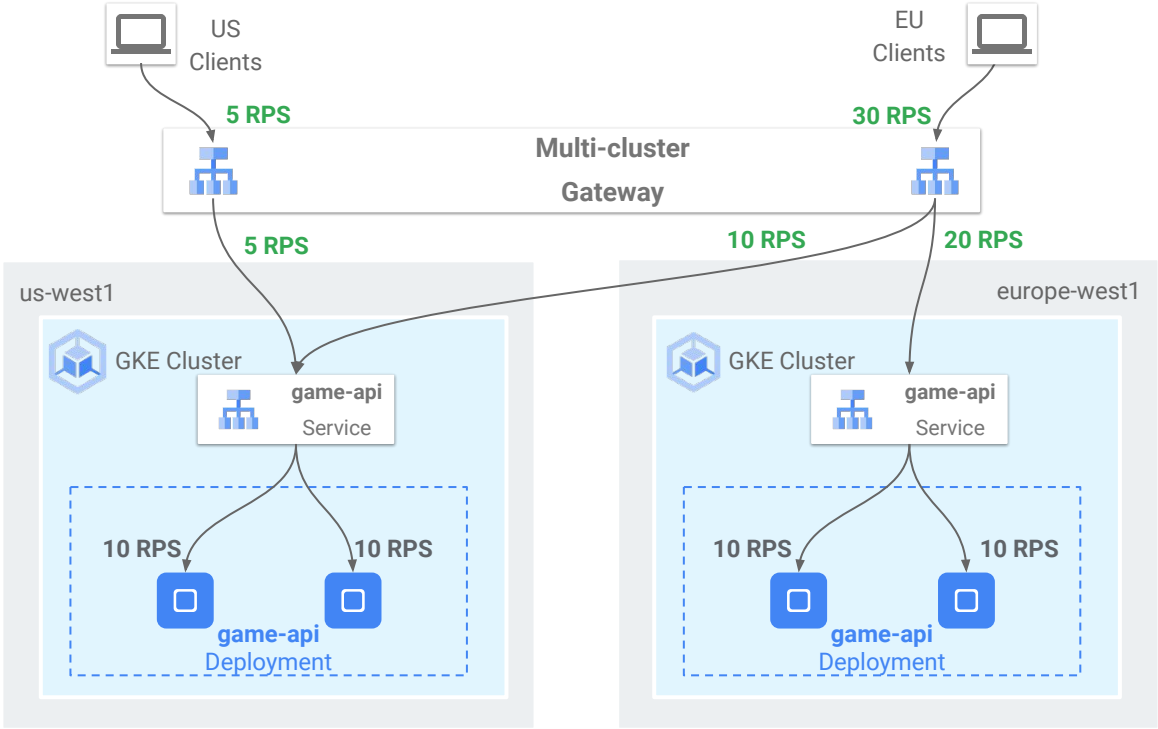
MultiClusterIngress によるクラスタ間での負荷分散



Public Preview

Gateway API Service Capacity

```
apiVersion: v1
kind: Service
metadata:
  name: store
  annotations:
    networking.gke.io/max-rate-per-endpoint: RATE_PER_SECOND
```

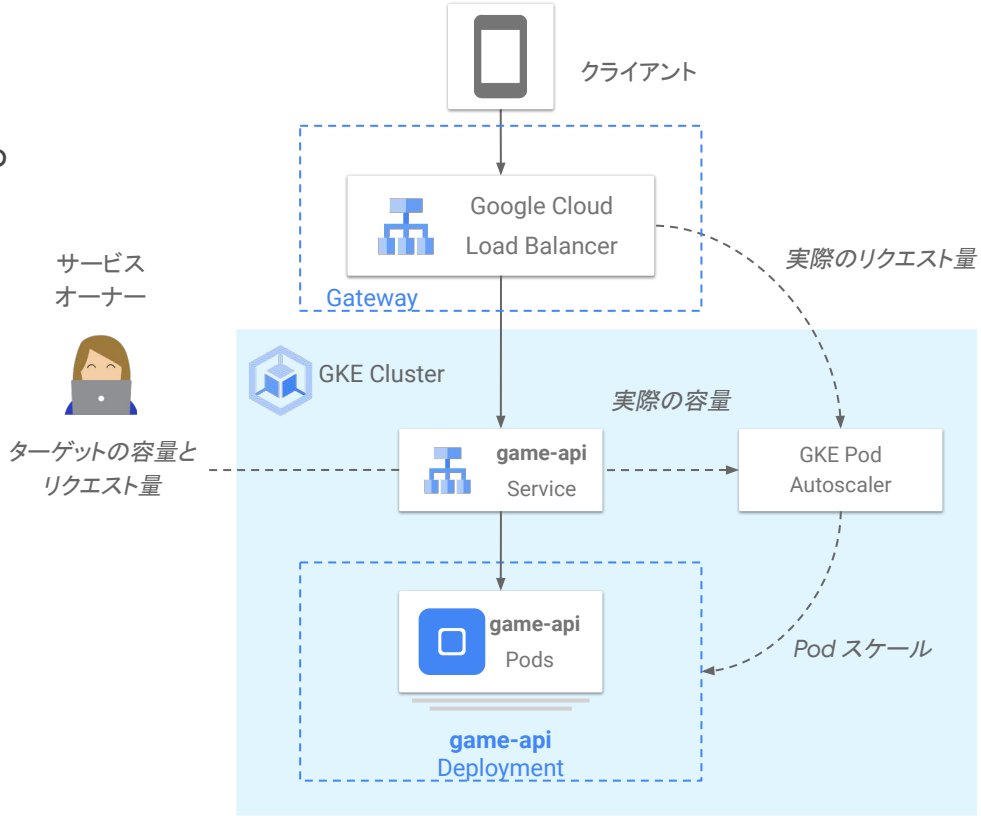


Public Preview

Gateway API Traffic Autoscaling

1 秒あたりのリクエスト数（RPS）を使って、HPA を設定できる

RPS を指標とすることで、DAU（Daily Active User）などのアプリの使用状況やビジネス指標に合わせてスケーリングできる



Gaming OSS と親和性も高い GKE

ゲームサーバー

マッチメイキング

agones

UBISOFT

open match

unity

ネットワーク

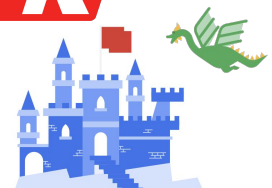
ストレージ

QUILKIN

embark

open saves

2K



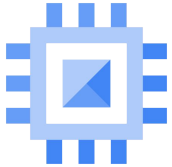
ゲーム会社様と共同で開発

Google Cloud

Google Cloud における Compute の選択肢

マネージド

サーバーレス



Compute Engine

仮想マシン
既存のシステムや
既存の運用

Kubernetes Engine

コンテナ実行基盤のマネージドサービス
アプリケーションの
デプロイ、管理、更新を容易に

App Engine

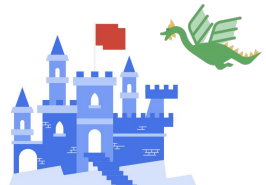
Web アプリ基盤
Web アプリや API 向け

Cloud Run

Web アプリ基盤
Web アプリや API 向け

Cloud Functions

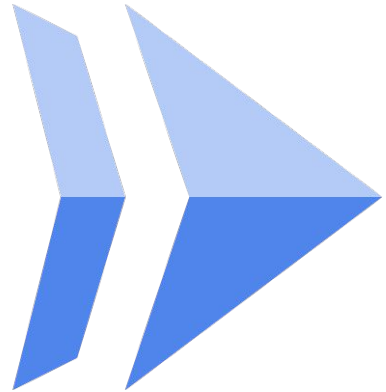
イベント駆動型
個別の機能・軽量の
ロジック

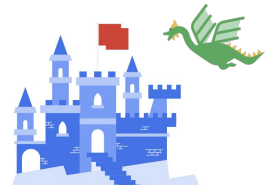


Cloud Run

Google が提供するマネージドなサーバーレスサービス

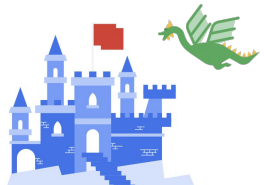
- 高速なデプロイ
  - ステートレスなコンテナ , 言語やライブラリの制約なし
  - 数秒でデプロイし URL を付与
- サーバーレス ネイティブ
  - 管理するサーバーはなく、コードに集中
  - 高速に 0 to N スケール
  - 効率的に使った分だけお支払い
- 高いポータビリティ
  - Knative の一貫した API の一貫性
  - ロックインの排除



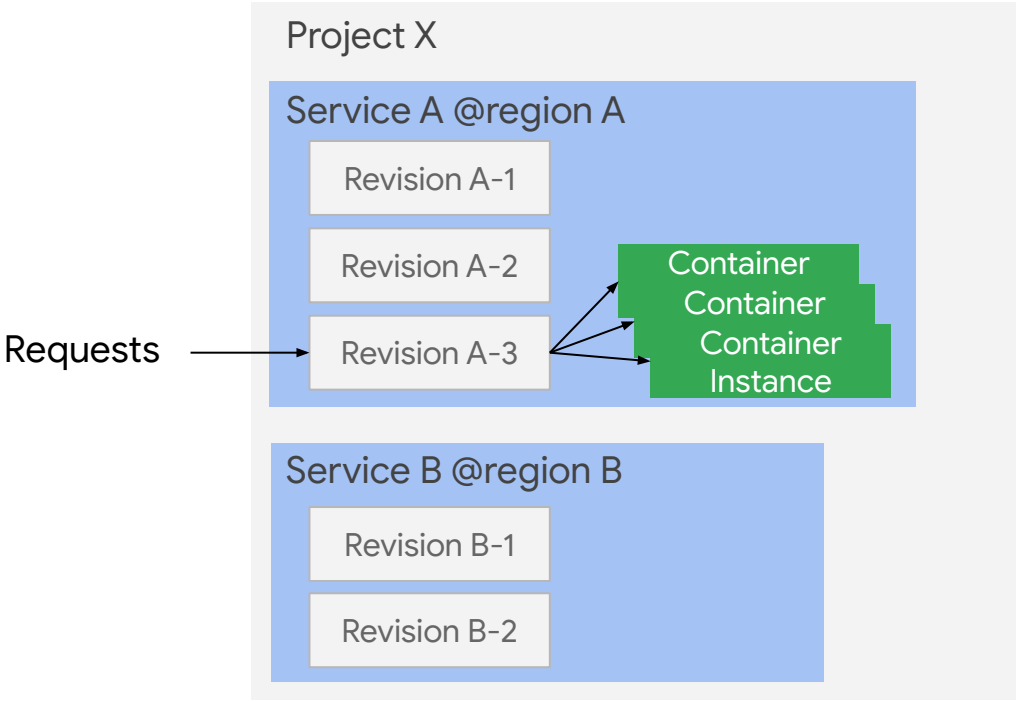


Cloud Run だとコンテナをシンプルに使える

Kubernetes	Cloud Run
コードを書く	**コードを書く**
$ docker build	$ docker build
$ docker push	$ docker push
$ kubectl apply -f deployment.yml	$ gcloud run deploy..
$ kubectl apply -f service.yaml	
$ kubectl apply -f hautoscal.yaml	
他にも監視・ロギングの設定など ...	



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

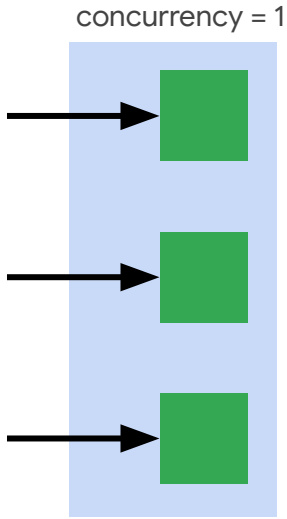
実際にリクエストを受けるコンテナ、
リクエスト の数に応じて自動的にスケール

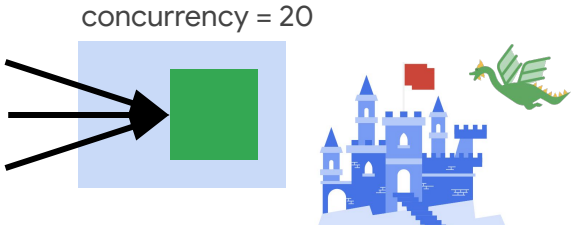
Cloud Run オートスケール

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

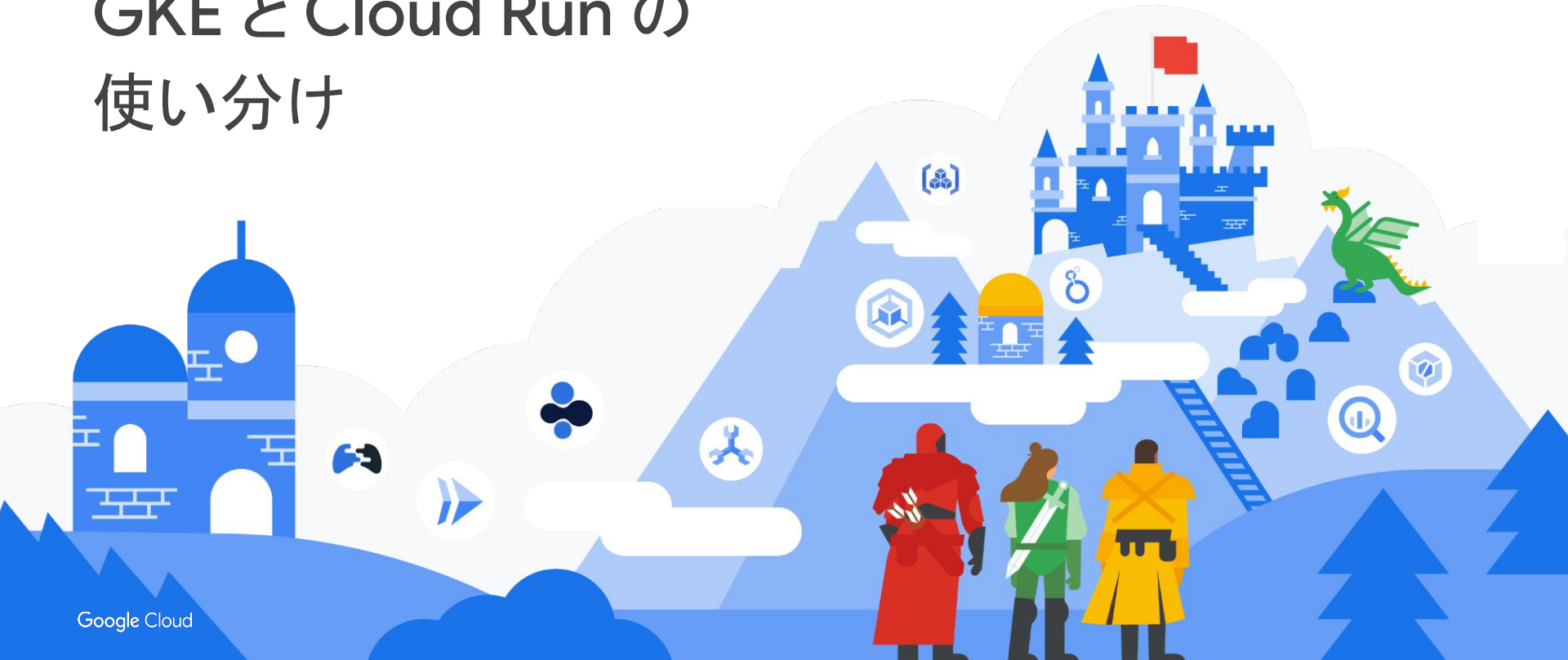
Cloud Run Concurrency（コンカレンシー）

- **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
  - 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
    （つまり"concurrency = 1"）
  - Cloud Run の場合、concurrency の値を 1 から 1000 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可能





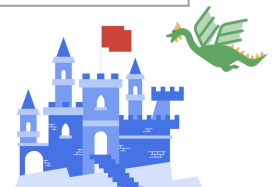
GKE と Cloud Run の使い分け



Google Cloud

コンテナソリューションの違い

	GKE Standard	GKE Autopilot	Cloud Run (managed)
概要	Kubernetes のマネージドサービス	Kubernetes のマネージドサービス	コンテナをサーバーレスに利用可能
コンテナの マシンスペック	特に制限なし (GCE の仕様に準拠)	最大 28 vCPU / 80 GiB RAM	最大 8 vCPU / 32 GB RAM 永続 Disk なし
互換性のあるOSS	Kubernetes	Kubernetes	Knative
インフラ抽象度	低	中	高
費用	Node (GCE の 仮想マシン) の インスタンス時間に応じた 従量課金とクラスタ管理手数料	Pod のインスタンス時間に 応じた従量課金と クラスタ管理手数料	リクエスト処理時間とリクエスト数に応じた従量課金 もしくは、コンテナインスタンス 時間に応じた従量課金



GKE と Cloud Run どちらを選択するか？

GKE を選択する場合

- 大規模なマイクロサービス
- カーネルレベルのパフォーマンス・チューニング
- 独自メトリクスを使ったスケーリング
- K8s や CNCF のエコシステムを活用したい
- K8s に習熟したインフラ人員がいる、
  もしくはこれから揃える

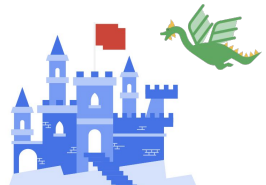
Cloud Run を選択する場合

- とにかくスモールスタートしたい
- パフォーマンスとかスケーリングとかは
  Cloud Run にお任せ
- アプリ開発者中心の小規模チーム
- K8s に習熟した専門インフラチームがいない、
  これから揃える予定もない

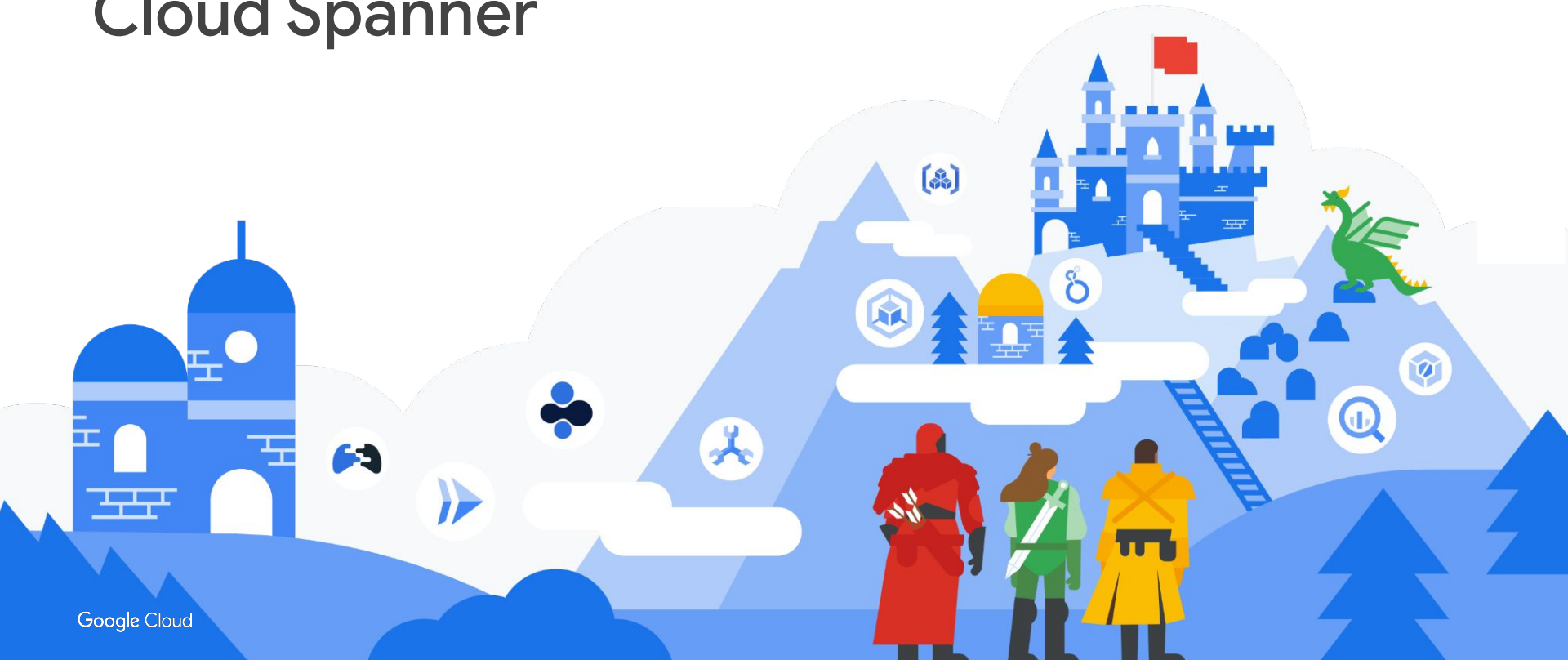
GKE Autopilot の使い所は？

K8s のマネージドサービスでありながら、GKE Standard よりシンプルに運用出来るのが強み。

K8s にチャレンジしたいが、K8s のクラスタ設計や運用にリソースを多く割けない場合や、
パフォーマンス・チューニング、高速なスケーリングなどが必要ない場合におすすめ！！



Cloud Spanner



Google Cloud

Google Cloud における Database の選択肢

移行に適した
クラウド環境と
ベアメタル環境

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

Bare Metal Solution

BYOL が可能で
コンプライアンス準拠した
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

Compute Engine

Cloud Memorystore

マネージド
Redis &
memcached

Cloud SQL

マネージド RDBMS
MySQL &
PostgreSQL &
SQL Server

AlloyDB

高可用 & ハイパフォーマンス
マネージド
PostgresSQL

Cloud Bigtable

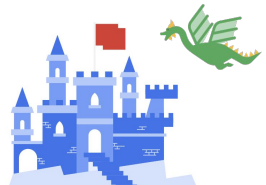
低レイテンシで
スケーラブルな
ワイドカラムストア

Cloud Spanner

スケーラブルで
可用性の高い
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



Google Cloud

Google Cloud における Database の選択肢

移行に適した
クラウド環境と
ベアメタル環境

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

Bare Metal Solution

BYOL が可能で
コンプライアンス準拠した
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

Compute Engine

Cloud Memorystore

マネージド
Redis &
memcached

Cloud SQL

マネージド RDBMS
MySQL &
PostgreSQL &
SQL Server

AlloyDB

高可用 & ハイパフォーマンス
マネージド
PostgresSQL

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

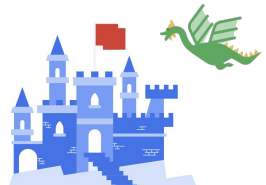
Cloud Spanner

スケーラブルで
可用性の高い
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

今回のテーマ



Google Cloud

運用観点でみる Database 分類

DB はユーザーによる管理

DB はフルマネージドで運用負担なし

サーバーレス

Bare Metal Solution

BYOL が可能で
コンプライアンス準拠した
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

Compute Engine

最低限の OS 環境まで用意されるので、そこから上は全てユーザー管理

Cloud Memorystore

マネージド
Redis &
memcached

Cloud SQL

マネージド RDBMS
MySQL &
PostgreSQL &
SQL Server

AlloyDB

高可用 & ハイパフォーマンス
マネージド
PostgresSQL

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
ミッション
クリティカル
RDBMS

ほとんどの運用は自動化されており、ノード追加といった構成変更作業は必要に応じてユーザーが実施

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

事前の構成なく利用可能
必要に応じてリソースの
見積もりをする

Google Cloud

サーバー構成要素でみる Database 分類

ノード単位

サーバーレス

Cloud Memorystore

Cloud SQL

AlloyDB

Cloud Bigtable

Cloud Spanner

Cloud Firestore

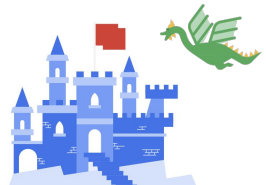
Redis
HA 構成のシングル

Memcached
複数ノードによる
シャーディング

HA 構成の
シングル DB
+
リードレプリカ

ノード追加による
スケールアウト

サーバーレスなのでサー
バー構成をユーザーが意
識する必要なくすぐ利用
開始できる



主な課金単位で見る Database 分類

ノード単位

サーバーレス

Cloud Memorystore

Cloud SQL

AlloyDB

Cloud Bigtable

Cloud Spanner

Cloud Firestore

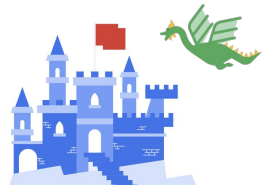
Redis
インスタンスの種類と合計メモリサイズ

Memcached
合計 vCPU サイズと合計メモリサイズ

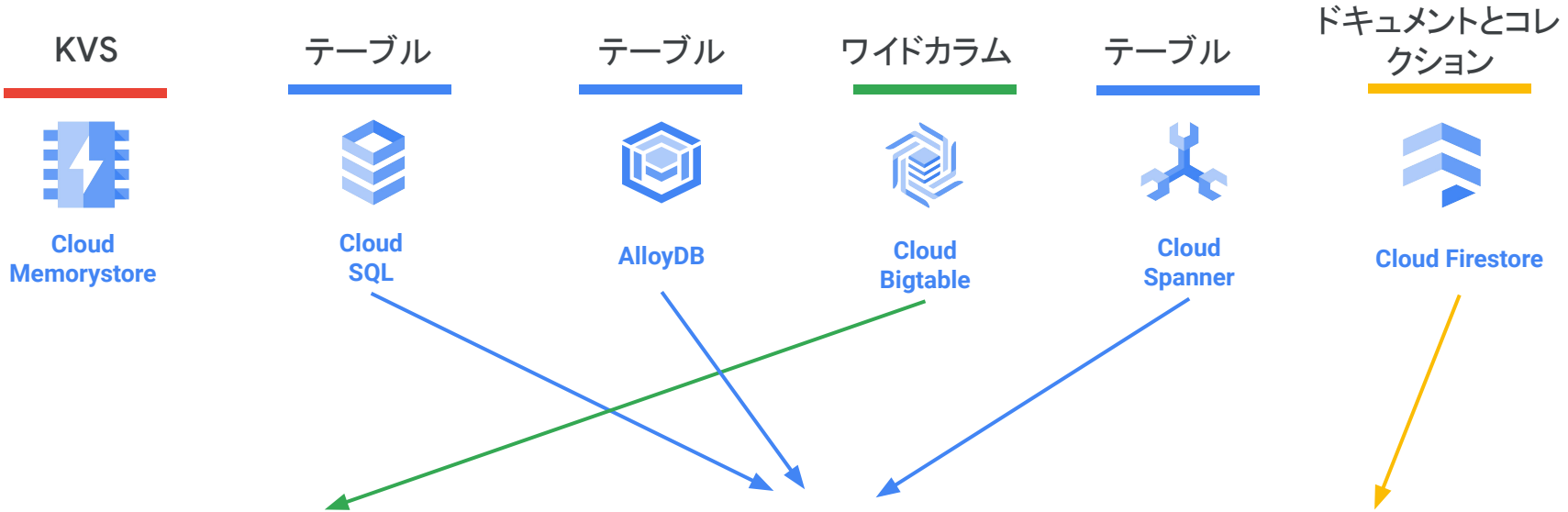
インスタンスの種類とノード数
+
ストレージ

ノード数
+
ストレージ

API コール数
+
ストレージ

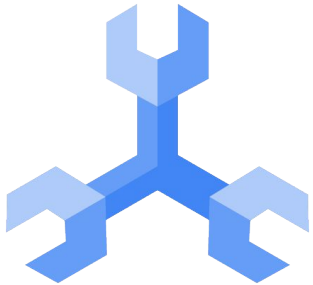


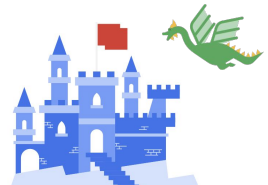
データモデルでみる Database 分類



Cloud Spanner

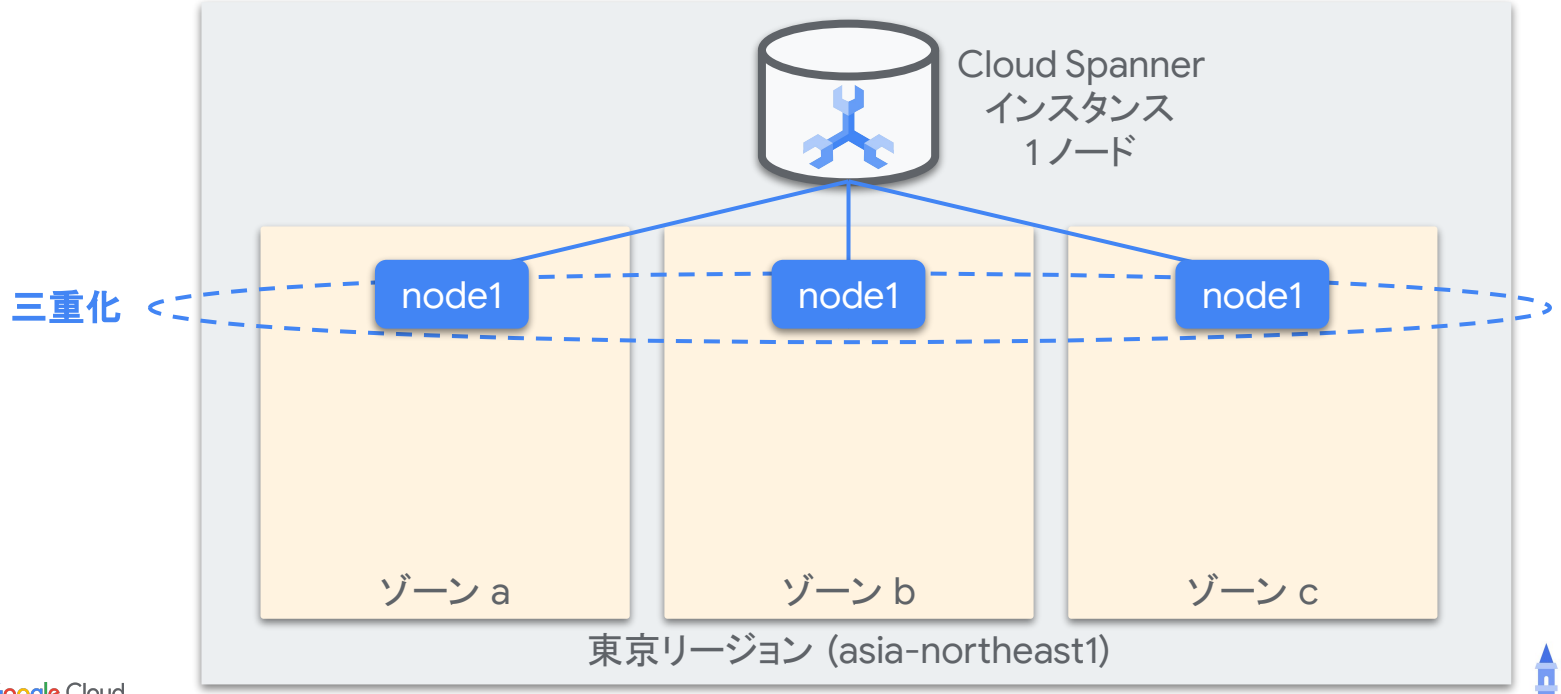
- Google のミッションクリティカル , スケーラブル , リレーショナル性 を持つ DB
- リレーショナル性
  - スキーマ、SQL 、 ACID トランザクション
  - RDB の使い勝手はそのままに
- 水平スケール
  - 99.999% SLA, フルマネージド , スケーラブル , マルチリージョン
  - DBA をシャード、レプリカ管理から開放





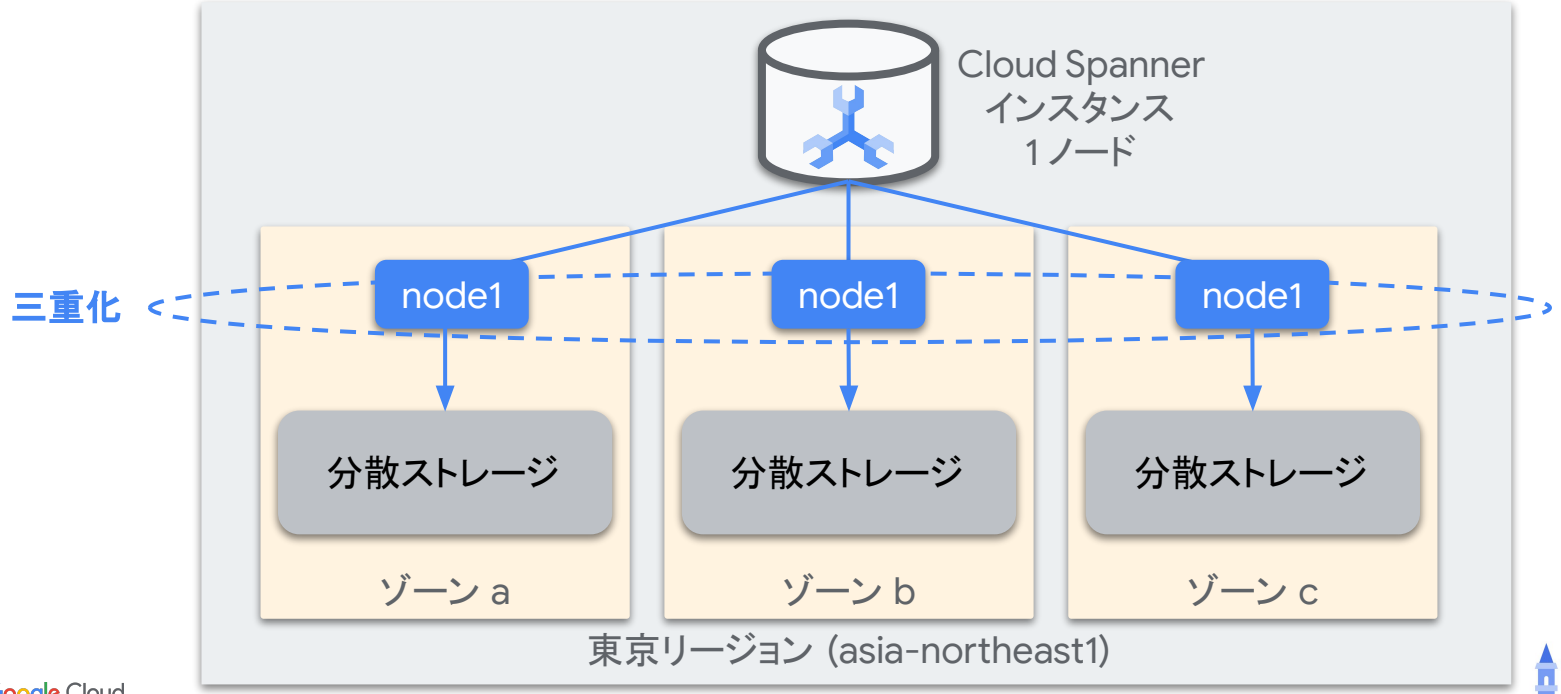
Cloud Spanner の冗長性

1 ノードの Cloud Spanner インスタンスとは、実際に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（ SPoF は無い）構成になっている。



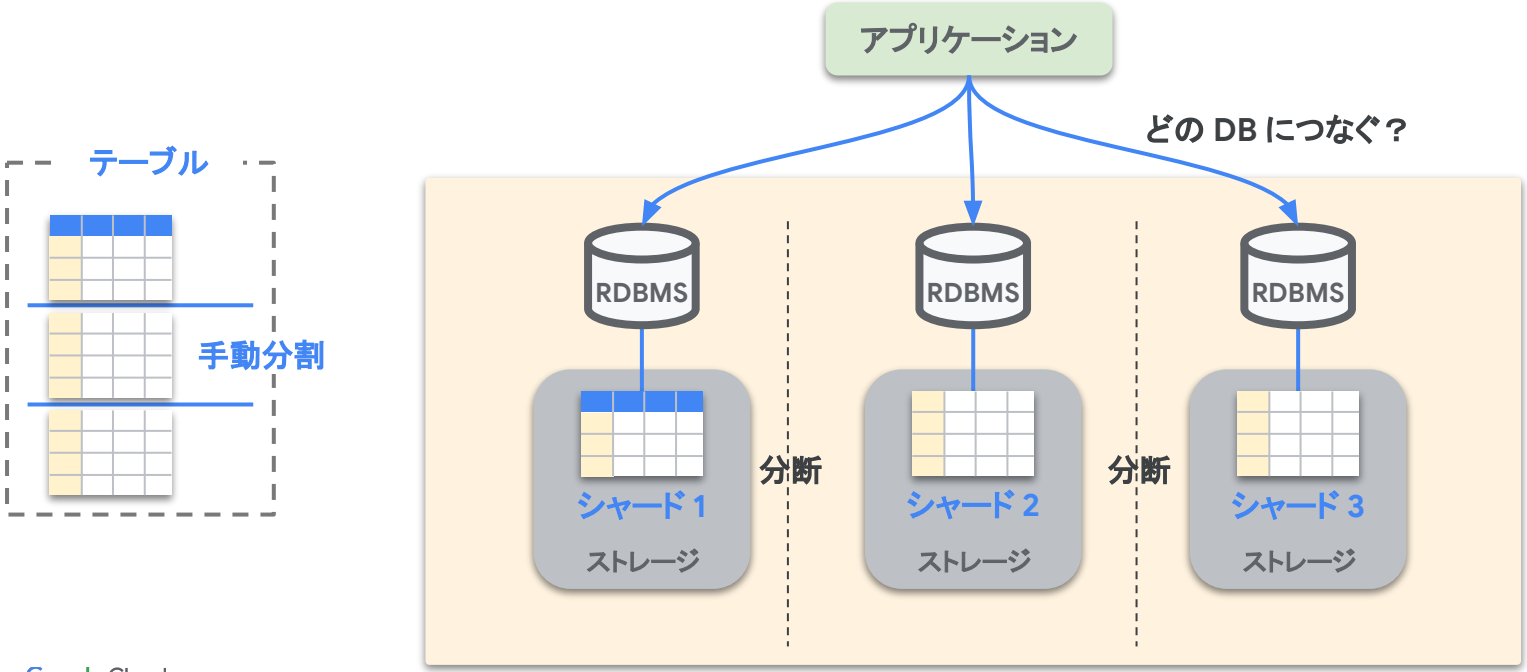
Cloud Spanner は 1 ノードであっても冗長化されている

実際のデータは分散ストレージに保存されるため、仮にノード（処理タスク）で障害が起こっても、データへ影響は無し。



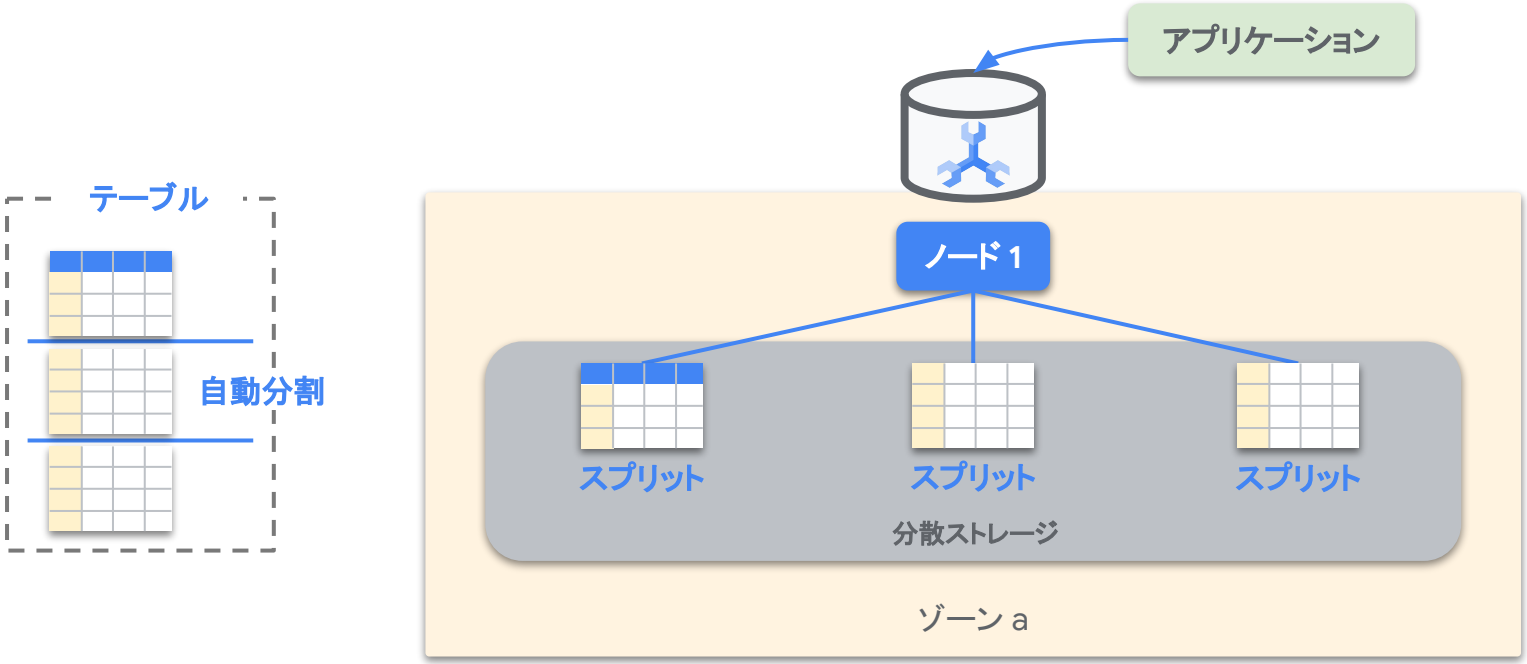
一般的な RDBMS の手動シャーディング

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



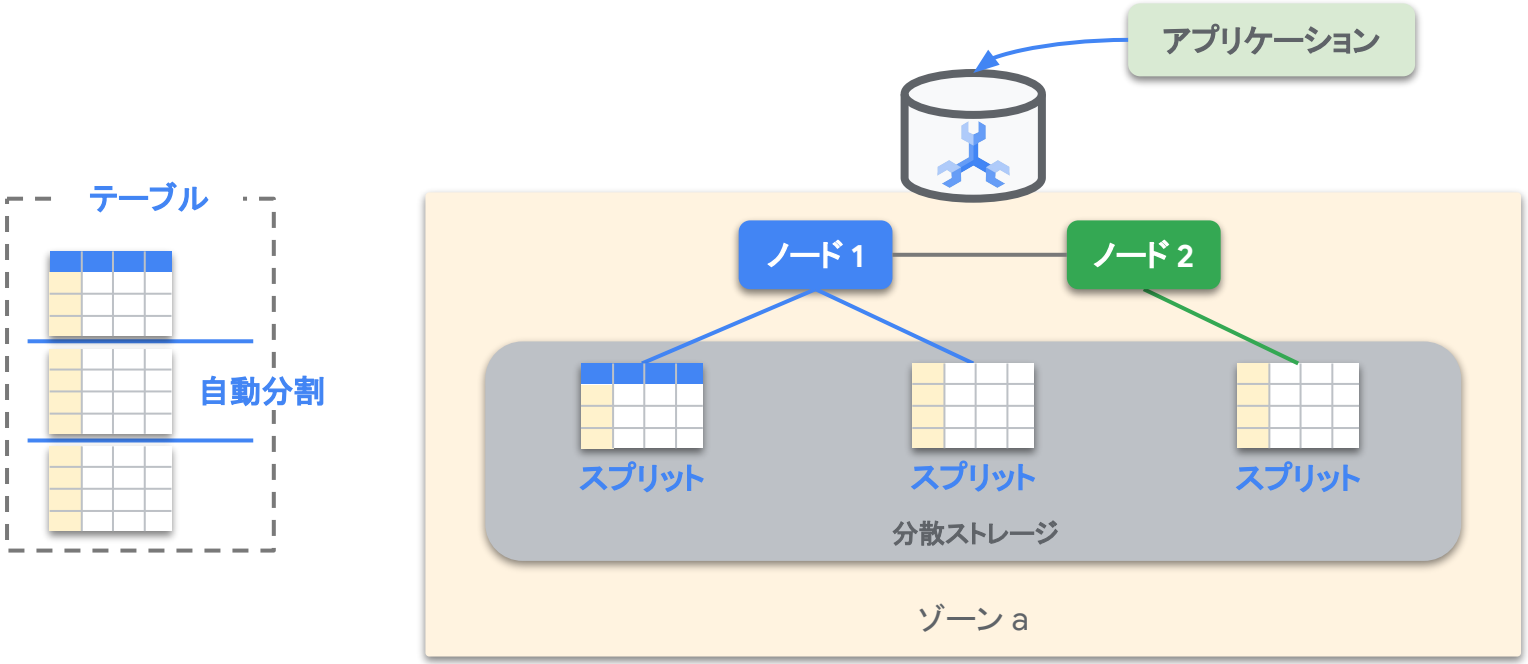
Cloud Spanner の自動シャーディング（1 ノード）

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。



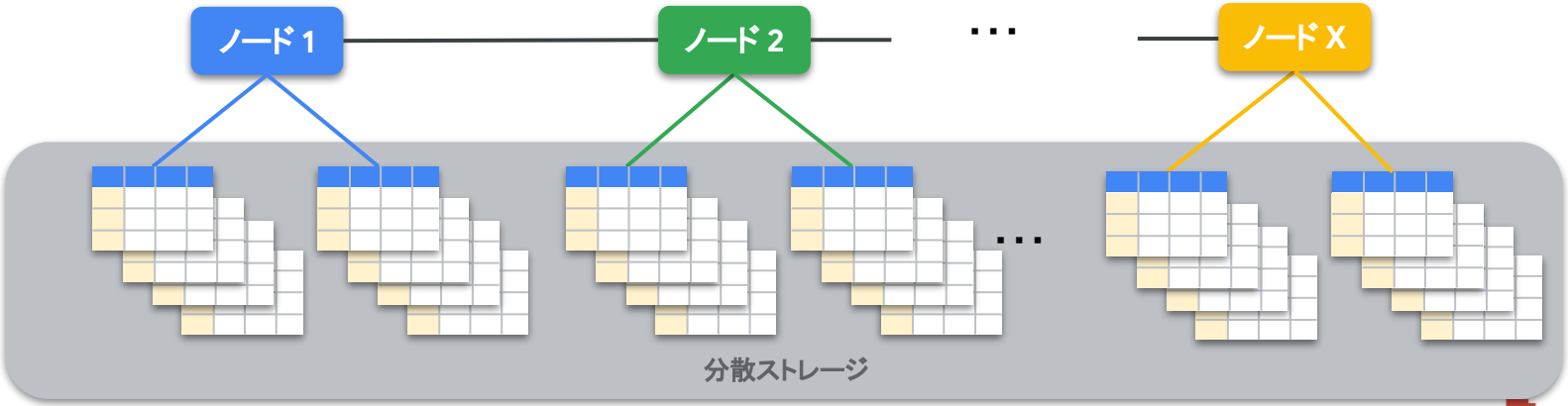
Cloud Spanner の自動シャーディング（2 ノード）

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。

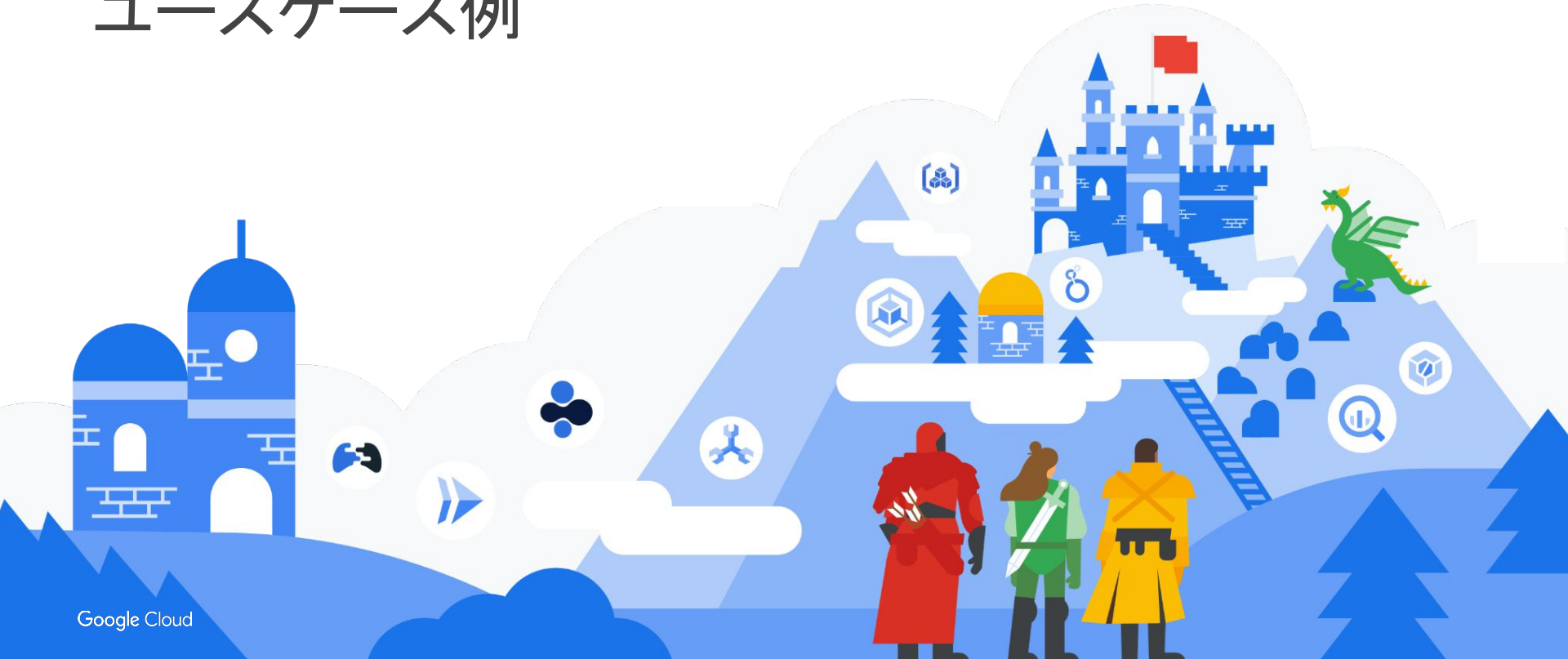


自在にスケール可能な Cloud Spanner

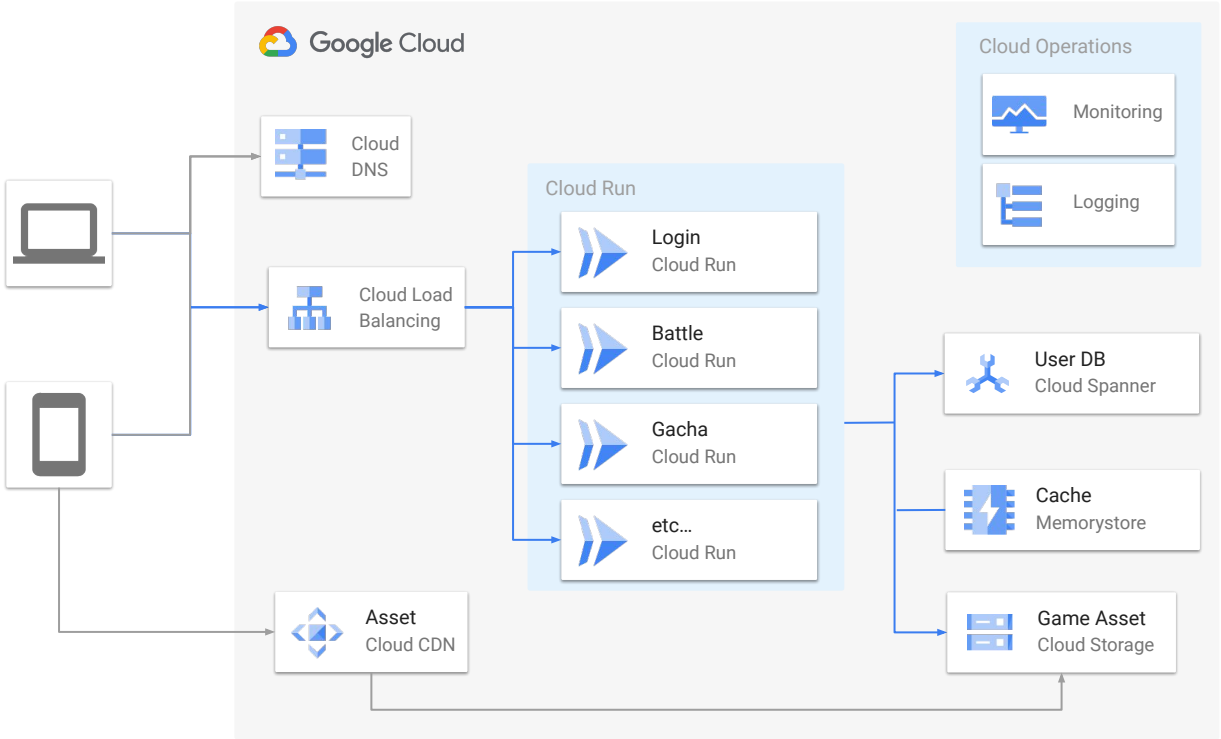
このようにして Cloud Spanner は、1 ノード未満からスモール スタートし、小規模構成から数百ノードもの大規模構成まで、柔軟にスケールさせることが可能。分散ストレージ内のスプリットは、データサイズや負荷状況に応じて分割や結合が行われる。



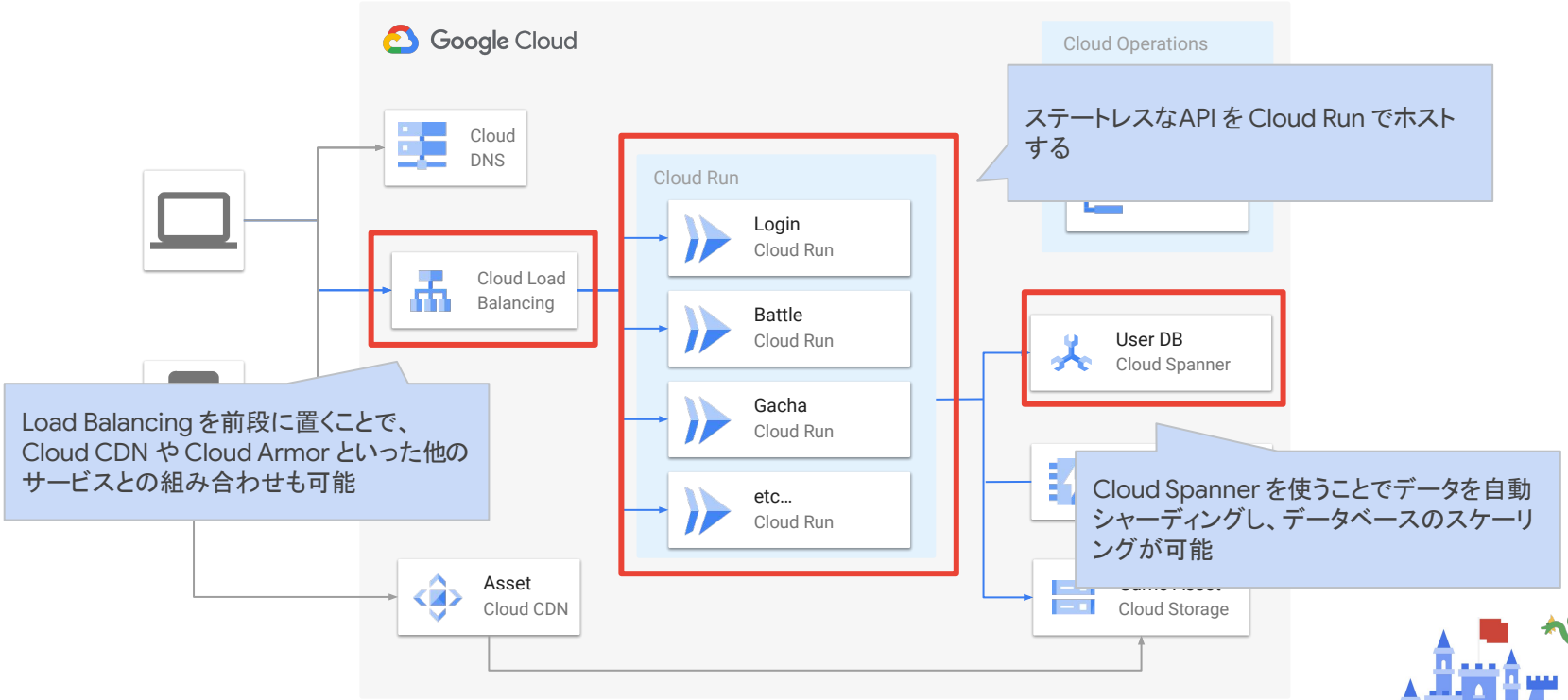
ユースケース例



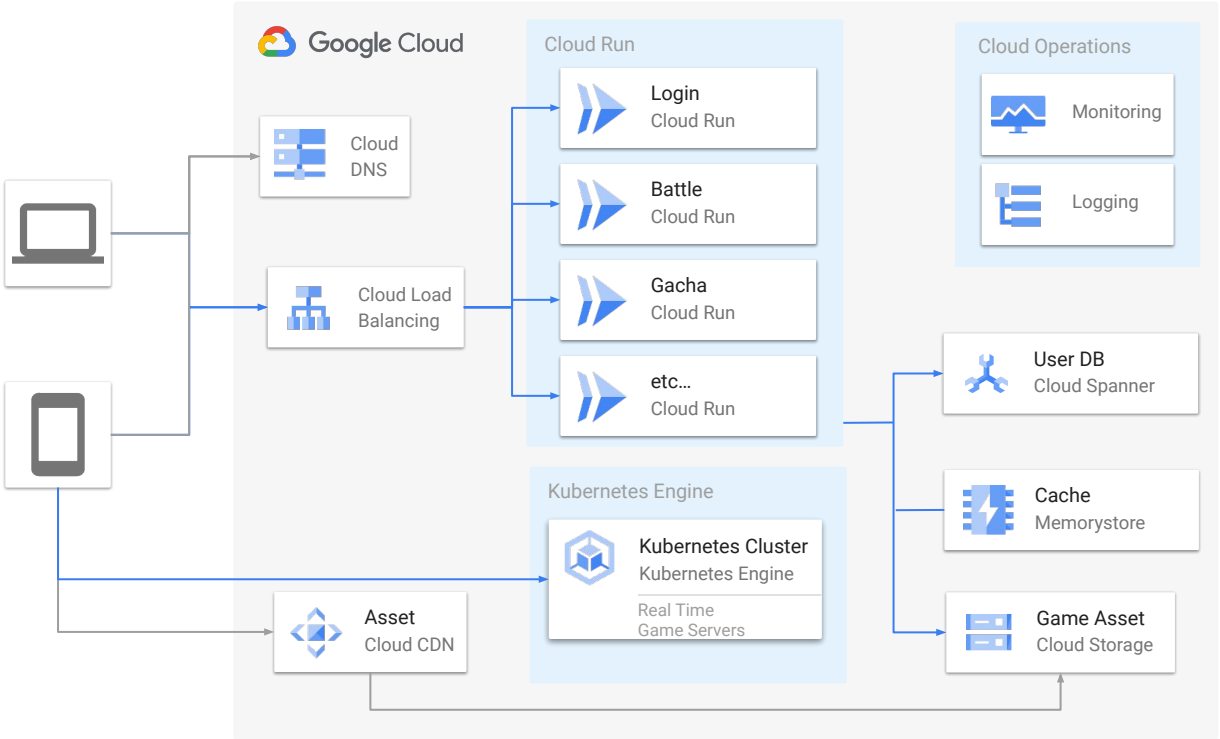
Cloud Run ユースケース: Game API Server



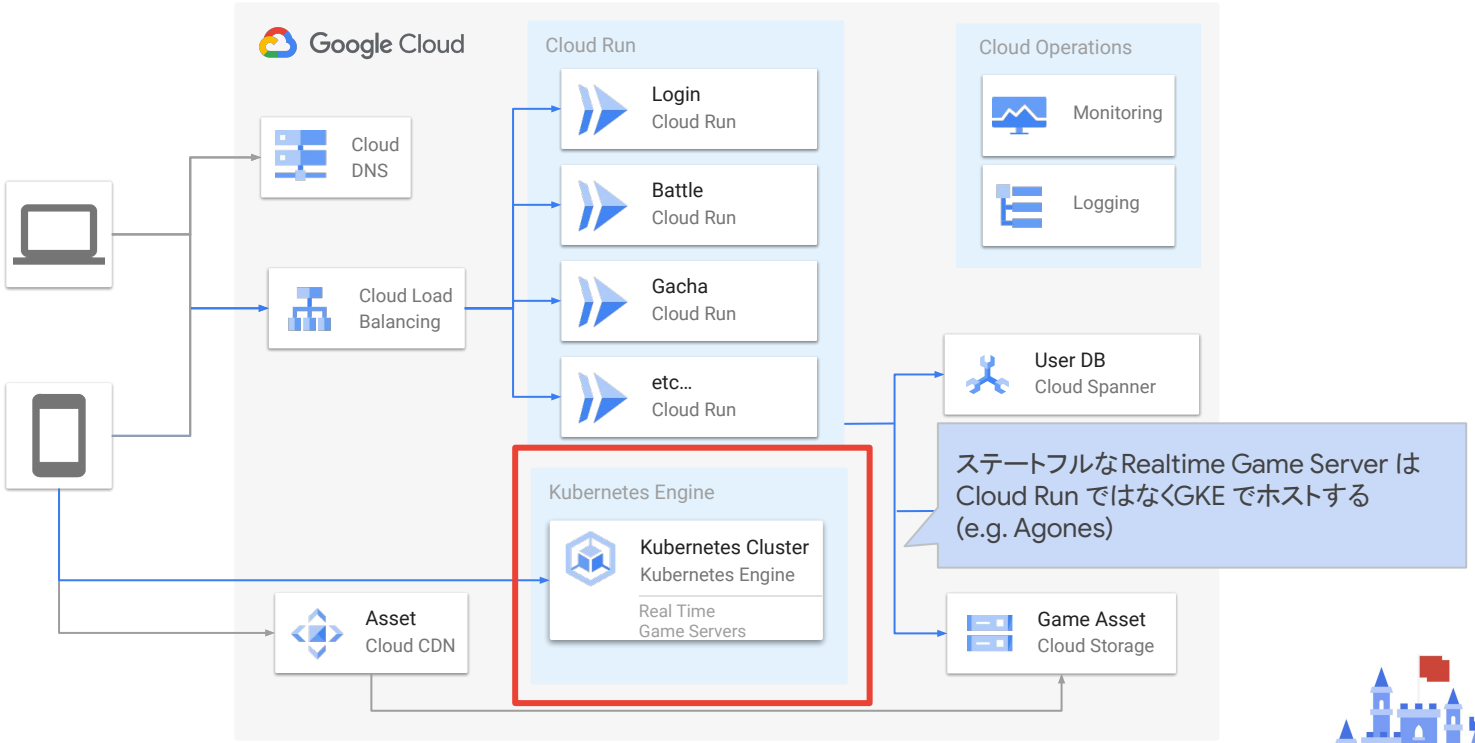
Cloud Run ユースケース: Game API Server



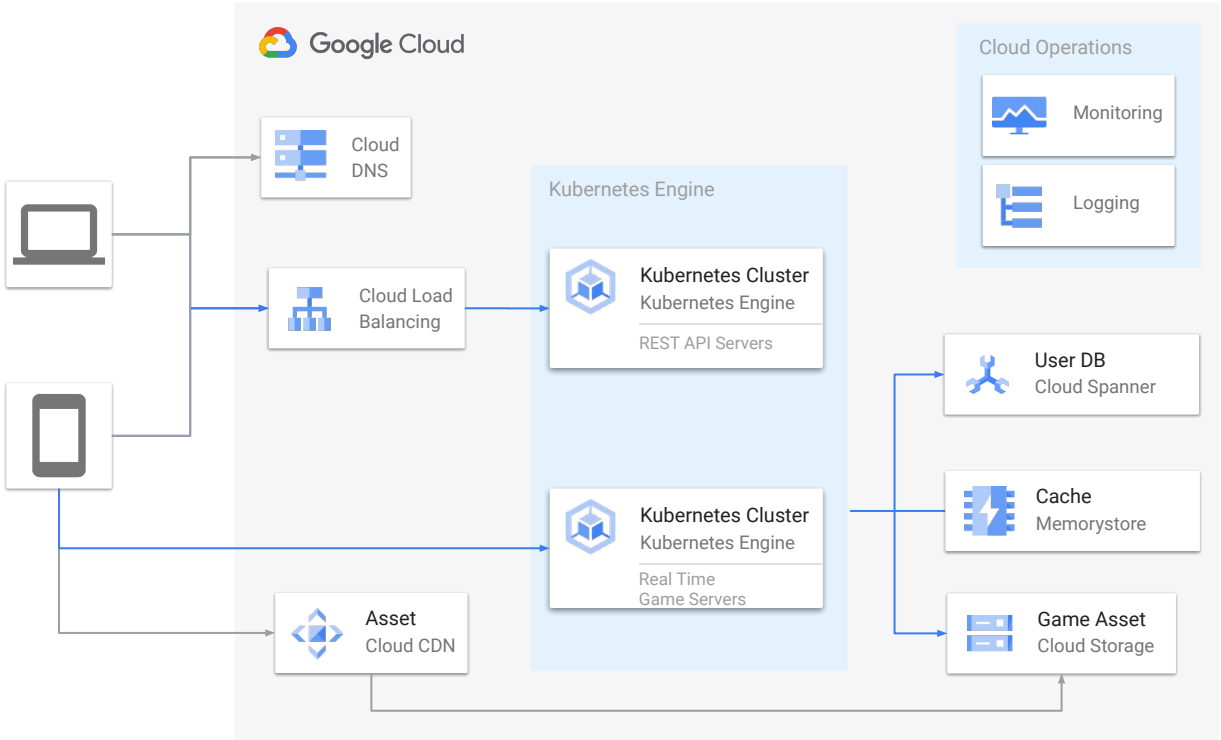
Cloud Run ユースケース: API Server + Realtime GameServer



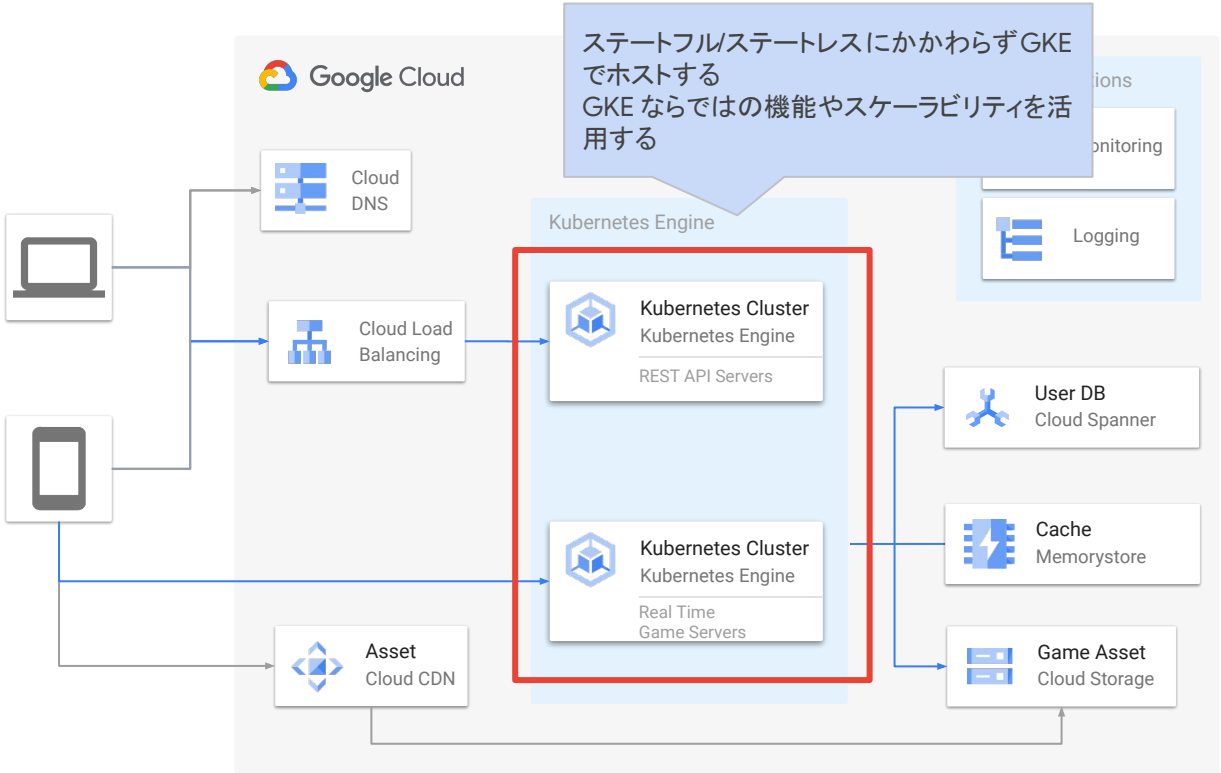
Cloud Run ユースケース: API Server + Realtime GameServer



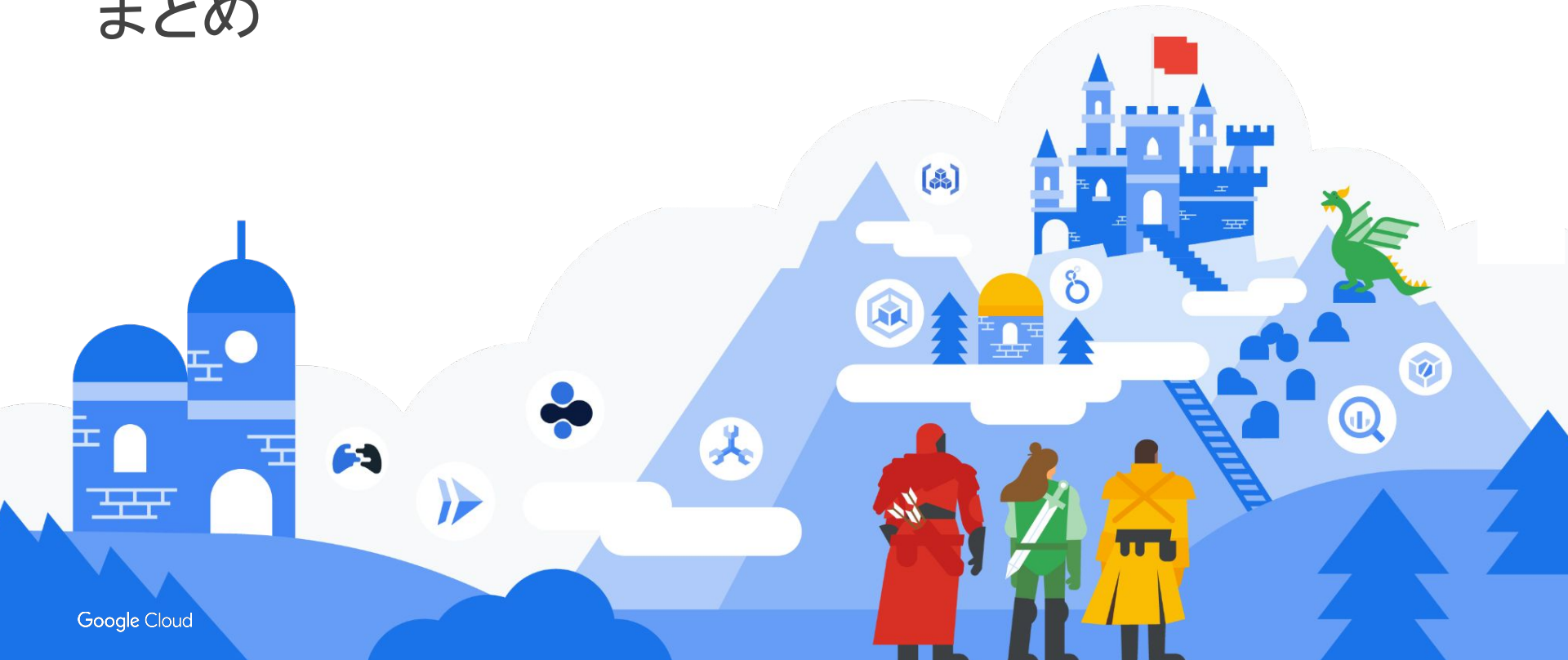
GKE ユースケース: API Server + Realtime GameServer



GKE ユースケース: API Server + Realtime GameServer

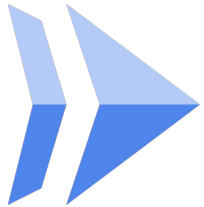


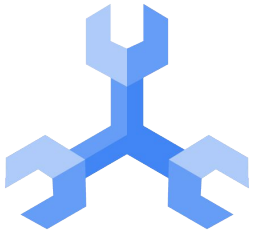
まとめ

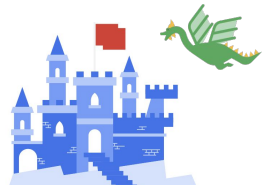


まとめ

- スケール可能なコンテナソリューションと Cloud Spanner を使うことで、スケーラブルなアプリケーションを構築可能
- Cloud Run と GKE の使い分けは、規模の大きさやメンバー構成などから考える
    - アーキテクチャによっては組み合わせもあり！
- Gaming OSS や Kubernetes/Cloud Spanner の運用などについては後続のセッションをぜひご視聴ください







Google Cloud

Thank you