

GKE を使ってやってみた！ 少人数にて実践した マイクロ サービス

株式会社10ANTZ

開発部 リードエンジニア
芳賀 幸一郎

会社紹介

VISION

ひらめきとモノづくりで、世界中に新たな感動を

MISSION

ファンとアーティストの“ありがとう”をつなぐ

会社紹介

モバイルゲーム事業



コンテンツ プランニング事業



オンラインコミュニケーションアプリ

Agenda

- マネージドサービスの活用
- マイクロサービス
- Google Cloud を選択して良かったこと

マネージドサービスの活用

開発時間を捻出

- 複数のゲームタイトルを運用中
 - エンジニア リソースをそこまで割くことができない
- 多くのサービスを開発する予定がある
 - 環境構築の手間は減らしたい
- マイクロサービスのノウハウを蓄積する時間を確保したい
 - 10ANTZ では初めてのマイクロサービスということもあり、試行錯誤する時間が欲しい

開発時間を捻出

● Compute

- Kubernetes Engine



● Database

- Cloud SQL(MySQL)
- Cloud Memorystore(Redis)
- Cloud Firestore(native mode)



● Operations

- Cloud Logging
- Cloud Monitoring



laC は Deployment Manager を採用

デプロイメント + DEPLOY MARKETPLACE SOLUTION ■ STOP 🗑️ 削除

☰ デプロイをフィルタ ?

<input type="checkbox"/>	●	名前 ↑	作成日時	最終更新	ラベル
<input type="checkbox"/>	✓	dev-gke	17 分前	11 分前	None
<input type="checkbox"/>	✓	dev-mysql	28 分前	25 分前	None
<input type="checkbox"/>	✓	dev-network	30 分前	29 分前	None
<input type="checkbox"/>	✓	dev-redis	25 分前	18 分前	None

laC は Deployment Manager を採用

The screenshot displays the Google Cloud Platform console interface for a deployment named 'dev-redis'. It is divided into three main sections:

- Left Panel (Navigation):** Shows a breadcrumb trail: Overview - dev-redis > dev-redis (manifest-1607567055192) > dev-redis (gcp-types/redis-v1:projects.locations.instances). A '削除' (Delete) button is visible at the top.
- Middle Panel (Overview - dev-redis):** Displays the 'デプロイ プロパティ' (Deploy Properties) table:

ID	2151912852376270880
作成日	2020-12-10 (11:24:15)
マニフェスト名	manifest-1607567055192
構成	表示
インポート	../templates/redis.jinja
レイアウト	表示
拡張構成	表示
- Right Panel (拡張構成 - Expansion Configuration):** Shows the JSON configuration for the resource:

```
resources:  
- name: dev-redis  
  properties:  
    authorizedNetwork: projects/[redacted]/global/networks/dev-network  
    instanceId: dev-redis  
    memorySizeGb: 1  
    parent: projects/[redacted]/locations/asia-northeast1  
    redisVersion: REDIS_5_0  
    tier: BASIC  
  type: gcp-types/redis-v1:projects.locations.instances
```

laC は Deployment Manager を採用

The screenshot shows the Google Cloud Platform console interface. On the left, a navigation pane shows the 'dev-redis' deployment under a project. The main area is divided into three panels: a left sidebar with a success message 'dev-redis をデプロイしました', an 'Overview - dev-redis' section, and an '拡張構成' (Configuration) section showing resource details.

Overview - dev-redis

デプロイ プロパティ

ID	2151912852376270880
作成日	2020-12-10 (11:24:15)
マニフェスト名	manifest-1607567055192
構成	表示
インポート	// templates/redis.iinja

拡張構成

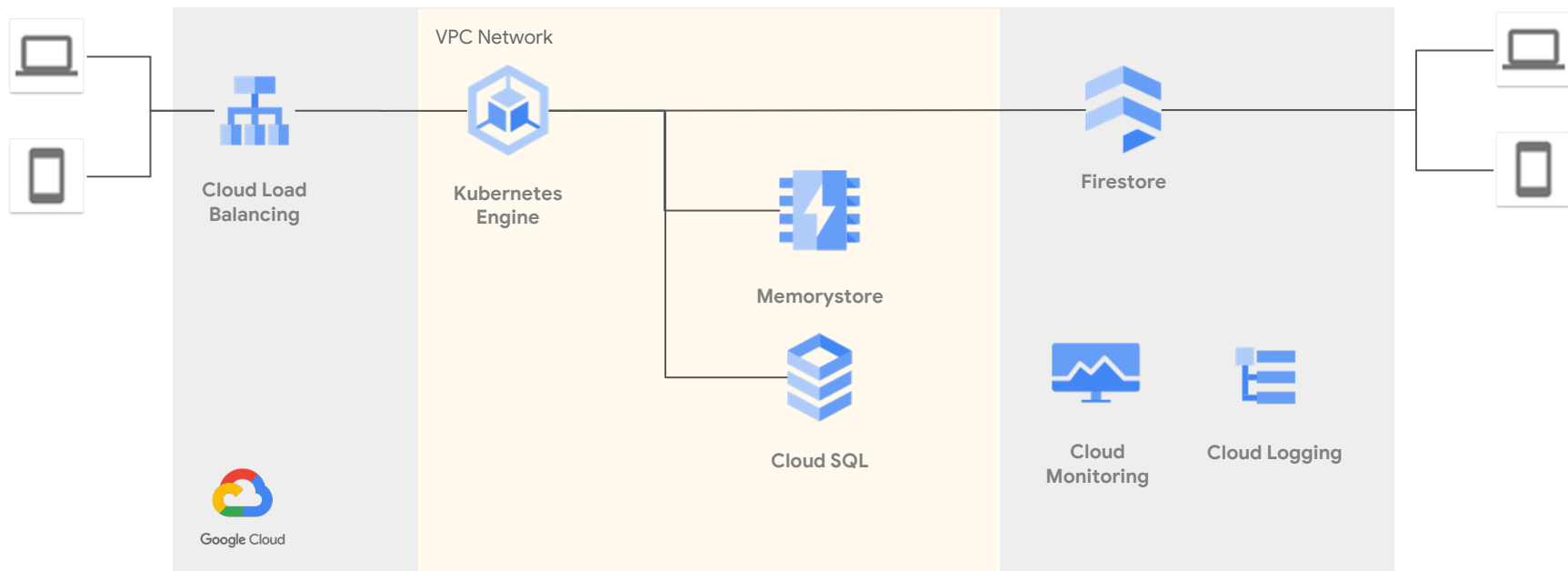
```
resources:  
- name: dev-redis  
  properties:  
    authorizedNetwork: projects/.../global/networks/dev-network  
    instanceId: dev-redis  
    memorySizeGb: 1  
    parent: projects/.../locations/asia-northeast1  
    redisVersion: REDIS_5_0  
    tier: BASIC
```

デプロイの削除

- dev-redis と、これによって作成されたすべてのリソース（VM、ロードバランサ、ディスクなど）を削除します
- dev-redis を削除しますが、これによって作成されたリソースは保持します

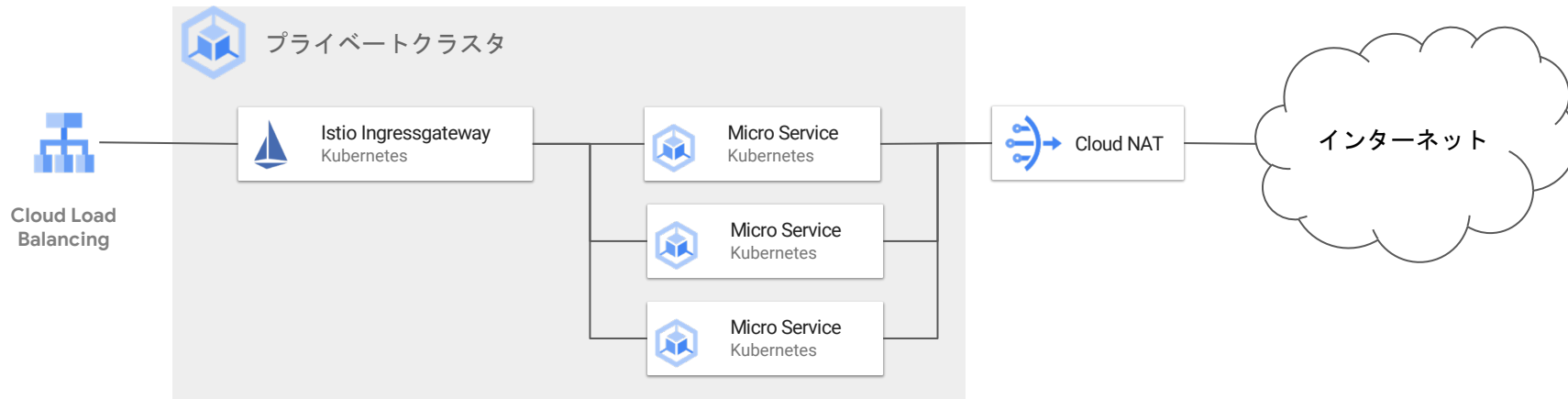
CANCEL

サーバ構成



マイクロサービス

Istio を利用した Service Mesh



Istio を利用した Service Mesh

← ロードバランサの詳細 [編集](#) [削除](#)

ab6be0ad932034c3584fed3e70a6440b

フロントエンド

[プロトコル](#) ^ IP:ポート [ネットワーク階層](#) ?

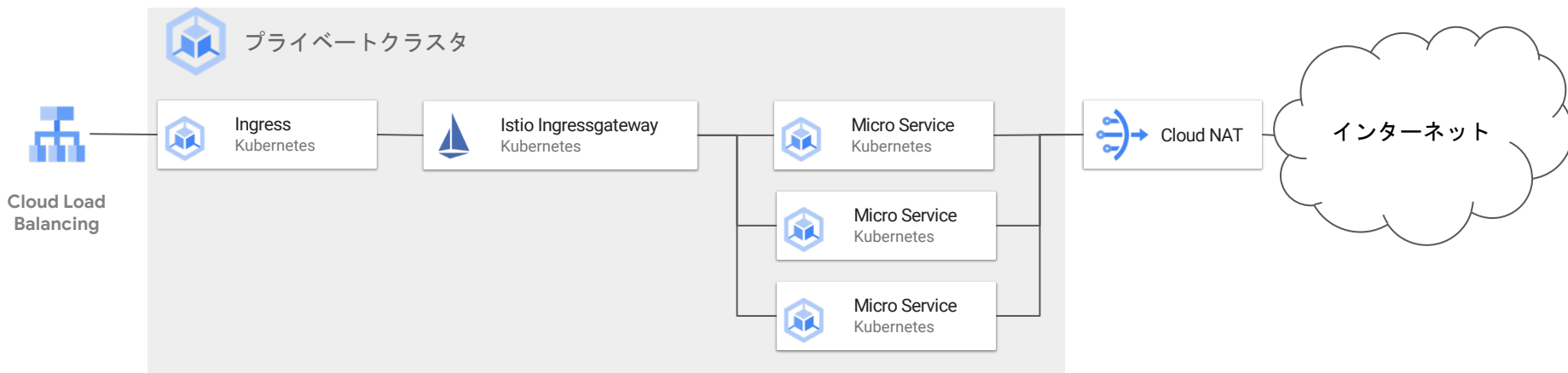
TCP  [プレミアム](#)

バックエンド

名前: ab6be0ad932034c3584fed3e70a6440b リージョン: asia-northeast1 セッションアフィニティ: なし ヘルスチェック: k8s-232fbf0d809455e2-node

```
> $ kubectl get svc --selector=app=istio-ingressgateway \  
...> $ -n istio-system \  
...> $ -o jsonpath='{.items[0].status.loadBalancer.ingress[0].ip}'
```

L7レイヤーの外部ロードバランサーを構築する



L7レイヤーの外部ロードバランサーを構築する

- ManagedCertificate

```
apiVersion: networking.gke.io/v1
kind: ManagedCertificate
metadata:
  name: certificate-name
  namespace: istio-system
spec:
  domains:
    - bar.10antz.co.jp
```

```
> $ kubectl describe managedcertificate -n istio-system
Name: [REDACTED]
Namespace: istio-system
Labels: <none>
Annotations: <none>
API Version: networking.gke.io/v1
Kind: ManagedCertificate
Metadata:
  Creation Timestamp: 2021-01-13T05:34:02Z
  Generation: 3
  Resource Version: 449947
  Self Link: /apis/networking.gke.io/v1/namespaces/istio-system/managedcertificates/[REDACTED]
  UID: 28881cf6-d83a-4907-ad9c-520c2e430d2f
Spec:
  Domains:
    [REDACTED]
Status:
  Certificate Name: mcrt-323c6cc0-9cfe-4039-8189-11405a58ba5e
  Certificate Status: Active
  Domain Status:
    Domain: [REDACTED]
    Status: Active
  Expire Time: 2021-04-12T21:43:30.000-07:00
Events: <none>
```


L7レイヤーの外部ロードバランサーを構築する

● BackendConfig

```
apiVersion: cloud.google.com/v1
kind: BackendConfig
metadata:
  name: ingress-backendconfig
  namespace: istio-system
spec:
  healthCheck:
    requestPath: /healthz/ready
    port: 15021
    type: HTTP
  securityPolicy:
    name: cloud-armor-policy-name
```

<input type="checkbox"/> アクション	タイプ	一致	説明	優先度 ^	
<input type="checkbox"/> <input checked="" type="checkbox"/> 許可	IP アドレス / 範囲		ip list of 10antz office	0	
<input checked="" type="checkbox"/> 拒否 (403)	IP アドレス / 範囲	* (すべての IP アドレス)	デフォルトのルール。優先度が高いとオーバーライドされます	2,147,483,647	

L7レイヤーの外部ロードバランサーを構築する

- Ingress

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: gke-ingress-name
  namespace: istio-system
  annotations:
    networking.gke.io/managed-certificates: certificate-name
    kubernetes.io/ingress.allow-http: "false"
spec:
  backend:
    serviceName: istio-ingressgateway
    servicePort: 80
```

L7レイヤーの外部ロードバランサーを構築する

← ロードバランサの詳細 [編集](#) [削除](#)

k8s2-um-tc46n8ut-istio-system-
15fsfxog

[詳細](#) [モニタリング](#) [キャッシュ](#)

フロントエンド

プロトコル ^	IP:ポート	認定証	ネットワーク階層 ?
HTTPS	:443	✔ mcr-323c6cc0-9cfe-4039-8189-11405a58ba5e	プレミアム

ホストとパスのルール

ホスト ^	パス	バックエンド
不一致すべて (デフォルト)	不一致すべて (デフォルト)	k8s1-5f65a919-istio-system-istio-ingressgateway-80-df1bdb7e

```
> $ kubectl get ingress \  
...> $ -n istio-system \  
...> $ -o jsonpath='{.items[0].status.loadBalancer.ingress[0].ip}'
```

マルチテナント構成

- Google Cloud プロジェクトを分けるか？
 - サーバ費用はオンラインコミュニケーションアプリとは別で計上したい
 - 一部機能はデータ含め共通化したい

結論: Google Cloud プロジェクトを分ける

マルチテナント構成

● どうやって一部機能を共通化するか

○ 同じコンテナをデプロイする？

- Container Registry は Cloud Storage の Read 権限を持ったサービス アカウント を利用すれば問題ない
- デプロイ先の管理コストが増える

○ Cloud SQL / Cloud Memorystore への接続は？

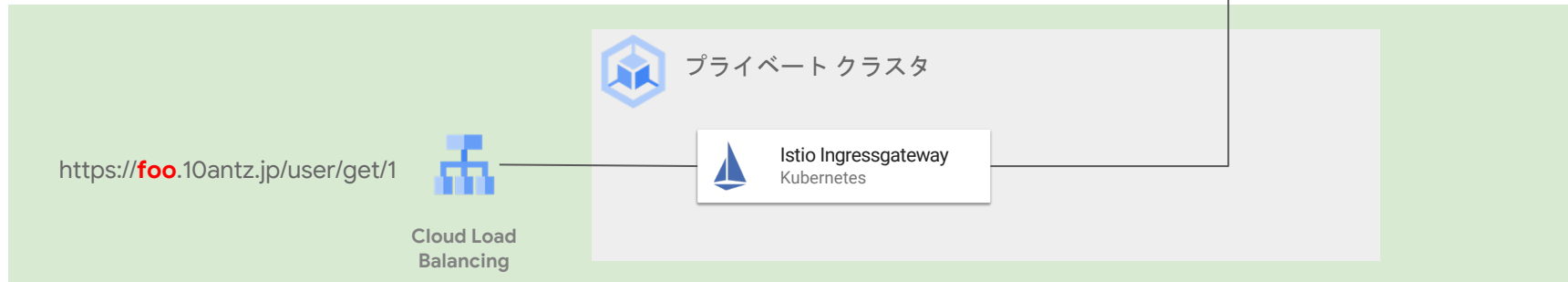
- VPC ネットワークが異なる状態

マルチテナント構成 (理想)

プロジェクト A



プロジェクト B



マルチクラスタ サービス メッシュ

ホーム > ソリューション

評価とクチコミ  

[フィードバックを送信](#)

レプリケートされたコントロールプレーン アーキテクチャを使用して GKE でマルチク ラスタ サービス メッシュを構築する

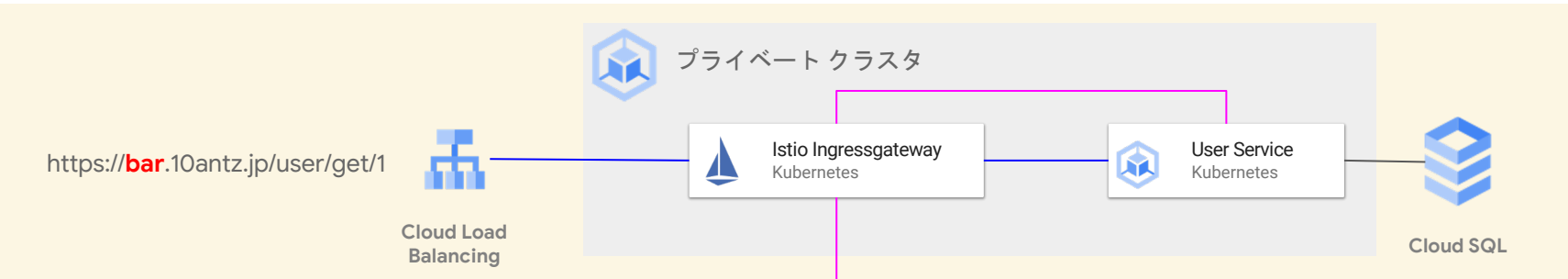
このチュートリアルでは、[Istio マルチクラスタ サービス メッシュ](#) を使用して、複数の Kubernetes クラスタにアプリケーションをデプロイする方法について説明します。Istio マルチクラスタ サービス メッシュを使用すると、複数の Kubernetes クラスタ上で実行されているサービス間で安全に通信を行うことができます。Kubernetes クラスタはさまざまな場所で実行されています。Google Cloud で実行される Google Kubernetes Engine (GKE) クラスタや、オンプレミスのデータセンターで実行される Kubernetes クラスタなど、異なるクラウド プラットフォームで実行されている場合もあります。

このチュートリアルでは、複数のネットワークにまたがる複数の GKE クラスタにサービス メッシュを作成する方法を説明します。このチュートリアルでは、Deployment、Service、Ingress など、Kubernetes の基本的な知識が必要になります。Istio の基礎知識は必要ありませんが、あったほうが内容を理解しやすくなります。

[Istio](#) は、Kubernetes クラスタで実行される Service の検出、動的ルーティング、安全な接続を行うためのオープンソースの [サービス メッシュ](#) です。Istio は、メッシュ内でルーティング、負荷分散、スロットリング、テレメトリ、サーキットブレイク、認証、サービス呼び出しの認可を行うためのポリシーベースのフレームワークを提供します。これにより、アプリケーションのコードをほとんど変更せずに、これらの機能を実現できます。Istio を Kubernetes クラスタに

マルチクラスタ サービス メッシュ

プロジェクト A



プロジェクト B



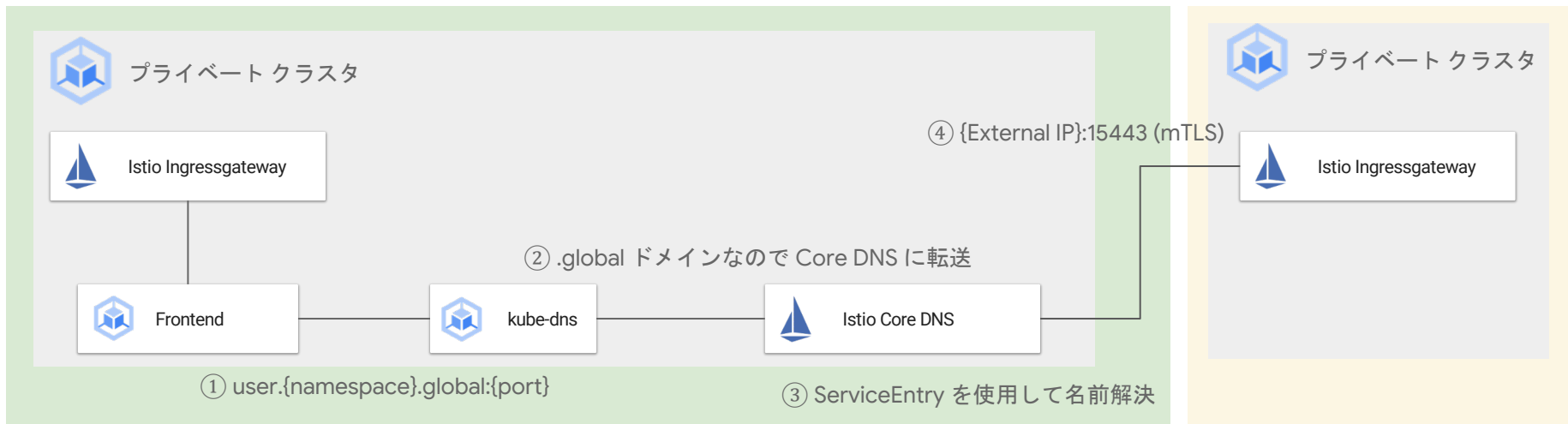
マルチクラスタ サービス メッシュ



プロジェクト B



プロジェクト A



マルチクラスタ サービス メッシュ

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-dns
  namespace: kube-system
data:
  stubDomains: |
    {"global": ["10.10.10.10"]}
```

Istio CoreDNS の Cluster IP

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: user-service-entry
spec:
  hosts:
  - user.10.10.10.10.global
  location: MESH_INTERNAL
  ports:
  - name: grpc
    number: 15443
    protocol: GRPC
  resolution: DNS
  addresses:
  - 10.10.10.10
  endpoints:
  - address: 10.10.10.10
    ports:
      grpc: 15443 # Do not change this port value
```

注意点

- Istio v1.7 系からこのソリューションが使えなくなっていた
 - <https://github.com/istio/istio/issues/26990>
 - Istio v1.7 で、このソリューションのテストが十分にされていなかったみたい

Google Cloud を選択して良かったこと

簡単に利用できるマネージド サービス

- マネージド サービスを利用して簡単にマイクロ サービスの環境を作れる
 - 少ない人数でもインフラ構築に費やすコストを削減することができた
 - モニタリング、ロギングが Cloud Monitoring / Cloud Logging に統合されているため開発時の不具合調査が楽だった
 - Deployment Manager を利用すれば同じ環境をすぐに作成できた

結果的に、開発者はサービス開発に時間を費やすことができた

デベロッパーに親切

- デベロッパーに寄り添ってくれる

- Google Cloud はドキュメントが豊富
- アーキテクチャ相談や Google Cloud の製品についての質問にすぐに回答してくれる
- 事例やブログとあわせて、「E.G.G. Japan (ゲーム業界向け特別トレーニング) への招待」

- 積極的にセミナーや勉強会に招待してくれる

- パートナー企業が開催してるハンズオントレーニングを紹介してくれる
- 今回のアーキテクチャで参考になる過去の Cloud OnAir を探してくれた

営業の方やカスタマーエンジニアの方が親身になって相談に乗ってくれる

まとめ

● 開発に専念できる環境作りを行い、時間を確保した

- Cloud SQL / Cloud Memorystore といったマネージドサービスを利用
 - インフラ管理コストの削減
- Cloud Monitoring / Cloud Logging の活用
 - 運用・監視ツールの導入コスト削減
- Deployment Manager を利用した環境構築
 - インフラ環境構築の手間を削減

● GKE を利用したマイクロサービス

- GKE と OSS の Istio を利用するだけで、簡単にマイクロサービスを体感できる
- GKEには Network Endpoint Group があり、External LB から直接 pod にバランシング可能

今後の展望

● Anthos Service Mesh の利用

- 今回 Istio を利用していたが、Anthos Service Mesh に切り替えることを検討
 - コントロール プレーンが分離されて Google の管理となる
 - Istio のバージョンアップでマルチクラスタ設定が使えなくなる、
といった事がなくなるので期待
 - Service Mesh の可視化

● Cloud Spanner の利用

- 書き込み分散
- node のスケーラビリティ

We are Hiring!!

