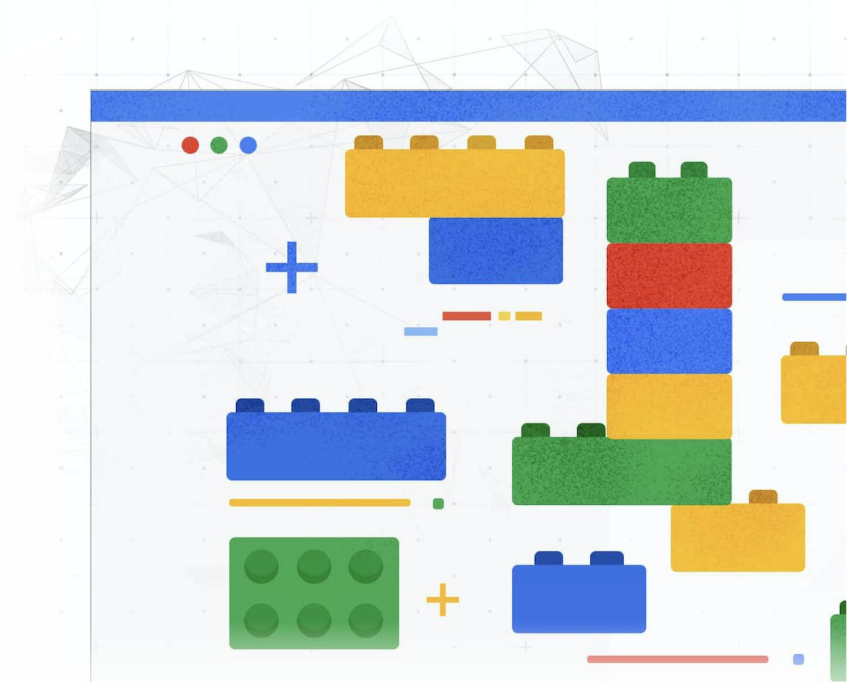
GitOps でインフラ構築
~ yaml で定義して
sync everything ~

グーグル・クラウド・ジャパン合同会社
アプリケーション モダナイゼーション スペシャリスト
頼兼 孝幸



アジェンダ

1. インフラ構築の自動化
2. Config Connector と Config Sync
3. Config Connector と Config Sync デモ
4. まとめ

インフラ構築の自動化

どのようにクラウドの環境を構築していますか

1. GUI でポチポチ
    a. 属人的。変更履歴が無い。流用できない。
2. gcloud コマンドを必要なだけ実行（シェル スクリプトなどで記述して実行）
    a. git 管理可能。しかし、パラメータが羅列されて可視性が悪い
    b. 適用済の state 管理まではされない
3. Terraform などを利用した Infrastructure as Code（IaC）で構築
    a. フォーマットされた形式で記述。git 管理もでき、可視性も良い。
    b. state をファイルで管理（GCS などに配置）して、差分適用も可

Infrastructure as Code

理想を定義したコードを書き、
インフラを管理する思想

一般的に CI 環境でコマンドを実行
e.g. *terraform plan, terraform apply*

プログラミング言語のように記述

内部的に API を call して構築

```
variable "region" {
  default = "us-west1"
}

variable "network_name" {
  default = "tf-gke-k8s"
}

provider "google" {
  region = "${var.region}"
}

resource "google_compute_network" "default" {
  name                    = "${var.network_name}"
  auto_create_subnetworks = false
}

resource "google_compute_subnetwork" "default" {
  name                     = "${var.network_name}"
  ip_cidr_range            = "10.127.0.0/20"
  network                  = "${google_compute_network.default.self_link}"
  region                   = "${var.region}"
  private_ip_google_access = true
}
```

Terraform 適用例

プラットフォーム管理者

構成のコード化

HCL（HashiCorp Configuration Language）で定義

コード = アクション
コード ≠ 状態

状態は別途 state ファイルを管理

CI / CD

terraform validate
terraform plan
による確認

terraform apply
で適用

依存関係が解決出来ずに失敗する場合もあるため、定義が必要

state file

state file

プロビジョニング

state ファイルと定義の差分が API で構築される

サービス管理者

サービス毎 CI 管理

管理者が行いたいのは「適用」することなのかを考える

- 多くのリソースは適用だけではなく、**管理、維持され続ける**ことも大事
- より高度な管理が求められる
- 例えば、定義したコードの内容を確実に適用させ続けたい（ Single Source of Truth）
- 他にも、ポリシーを適用してガバナンスを効かせたい
- インフラとアプリ基盤、両方のデプロイと管理を統合的に行うのに、理想としては
  同じ構成管理ツールの利用 が望ましい

Configuration as Data

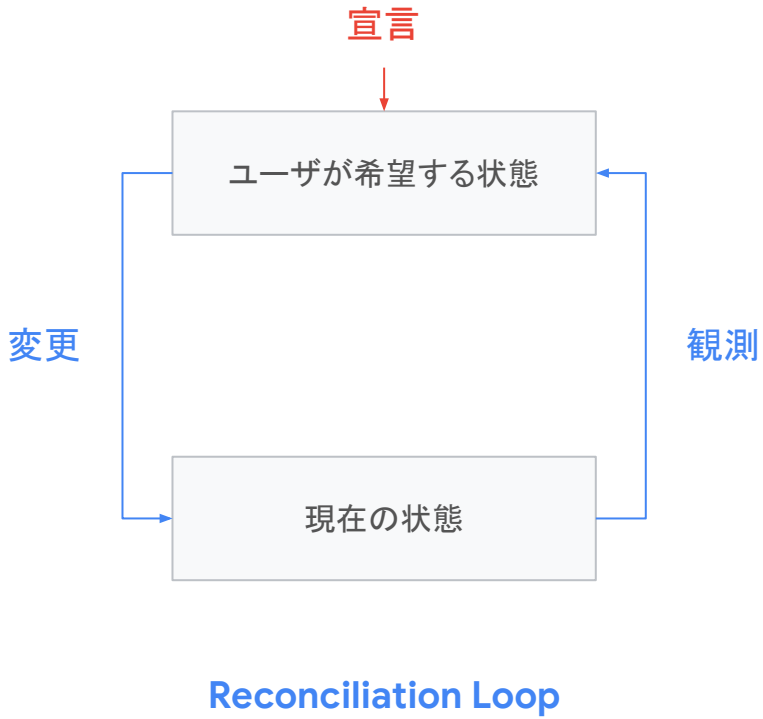
Kubernetes（K8s）で採用されている概念

操作や手順ではなく、状態を宣言する

K8s の Reconciliation Loop という特徴を利用

- 実際の状態を観測し、理想の状態（定義した内容）と比較
- 理想の状態になるよう変更を繰り返す

この手法をインフラ管理にも適用すると、継続的な適用が実現できる



Config Connector と Config Sync

02

Config Connector

Google Kubernetes Engine（GKE）クラスタ作成、更新時に、チェックボックスで有効化可能

Google Cloud のインフラ管理を CaD で実現

Kubernetes クラスタを基盤として利用する
（Kubernetes の知識が必要なわけではない）

Control Plane にある分散 KVS で、理想の状態を Kubernetes リソースとして管理

```yaml
apiVersion: iam.cnrm.cloud.google.com/v1beta1
kind: IAMServiceAccount
metadata:
  labels:
    label-one: "value-one"
  name: iamserviceaccount-sample
spec:
  displayName: Example Service Account
```

https://cloud.google.com/config-connector/

Config Connector を利用する場合

プラットフォーム
管理者

構成のデータ化

yaml で定義

コード = 状態

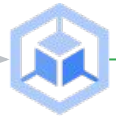
CI / CD

kubectl apply

で以下の両方を適用

- Google Cloud リソース

- Kubernetes リソース



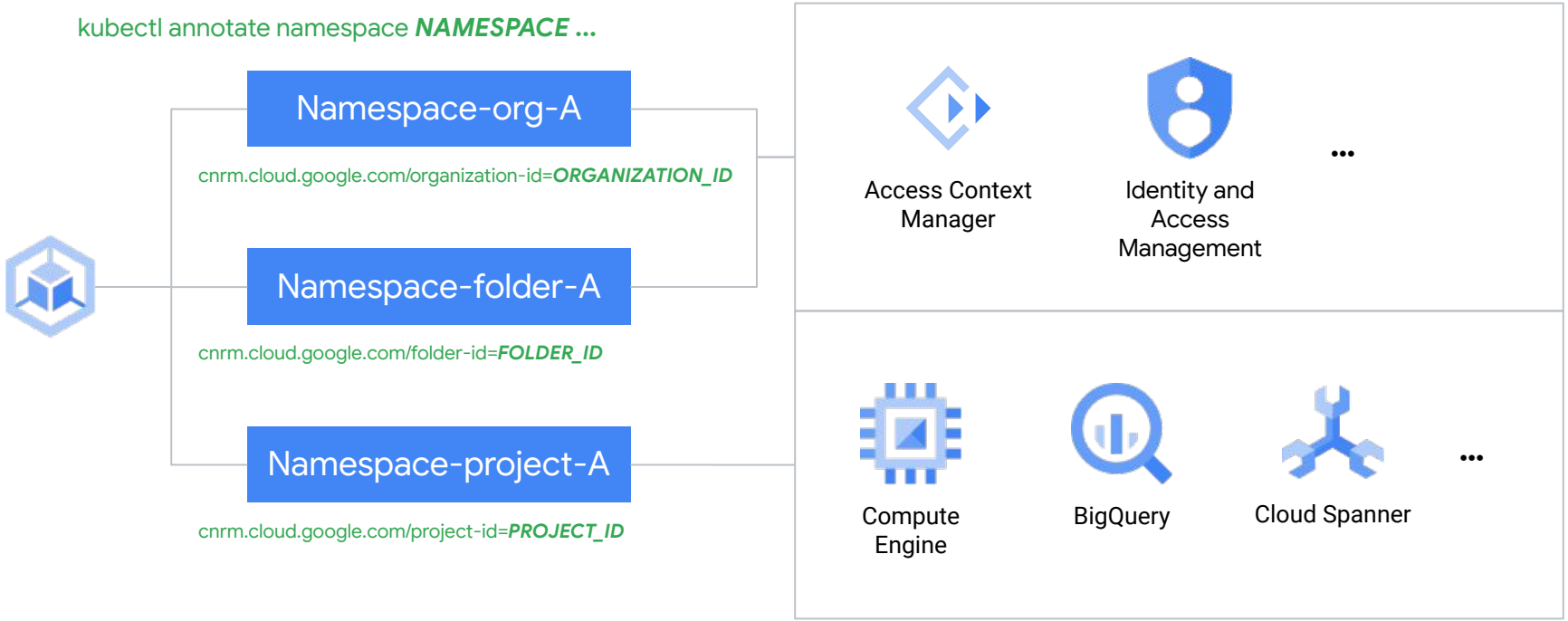
プロビジョニング

Reconciliation Loop
によって、理想の
状態になるよう変更さ
れる

サービス管理者

GKE が Google Cloud のコントロール プレーンとなる

Kubernetes Namespaces と Config Connector の適用範囲の関係性



Kubernetes manifest で定義する場合

リソースを必要に応じて、さまざまな
レベルのスコープで管理が可能

- プロジェクトレベル
- フォルダーレベル
- 組織レベル

```yaml
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    cnrm.cloud.google.com/project-id: PROJECT_ID
  name: NAMESPACE_NAME
```

```yaml
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    cnrm.cloud.google.com/folder-id: FOLDER_ID
  name: NAMESPACE_NAME
```

```yaml
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    cnrm.cloud.google.com/organization-id: ORGANIZATION_ID
  name: NAMESPACE_NAME
```

既存リソースの管理

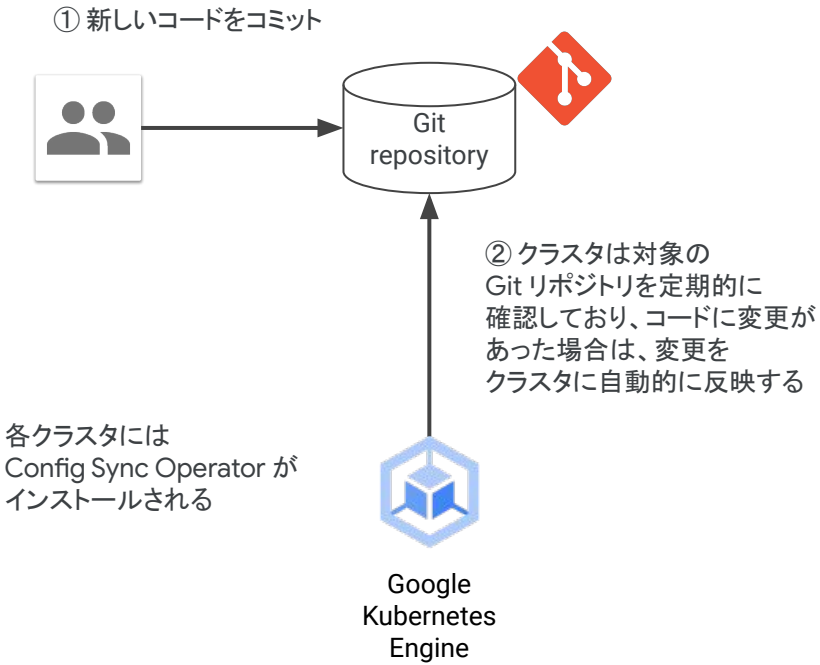
- Config Connector は Google Cloud リソースが存在しない場合は、新規リソースを作成
- **同じ名前**の Google Cloud リソースが存在する場合、そのリソースを取得して管理
- **resourceID** フィールドを使用して、リソースを管理することも可能

```
apiVersion: pubsub.cnrm.cloud.google.com/v1beta1
kind: PubSubTopic
metadata:
  name: pubsub-topic-sample
spec:
  resourceID: pubsub-topic-id
```

環境の数だけデプロイを管理するのは大変

- CaD のデプロイで実現したいこと
  - 状態を宣言し、その宣言した内容が自動的に各環境へ適用されている
  - Git に保存しているソースが Single Source of Truth となっている
- GitOps の適用フローを採用
  - 特定の Git ブランチへコミットすると、Google Cloud リソースの状態を管理する Kubernetes クラスタが、Pull 型でソースを取得して適用
  - ブランチ戦略を工夫することで、環境毎の適用管理が容易

Config Sync




```yaml
apiVersion: configsync.gke.io/v1beta1
kind: RootSync
metadata:
  name: root-sync
  namespace: config-management-system
spec:
  sourceFormat: FORMAT
  git:
    repo: REPO
    branch: BRANCH
    dir: "DIRECTORY"
    auth: TYPE
    secretRef:
      name: SECRET_NAME
```

Git リポジトリとの接続設定

ディレクトリ構造

nomos コマンドライン ツールを利用して構築

初期化

nomos init

構成と有効性を確認

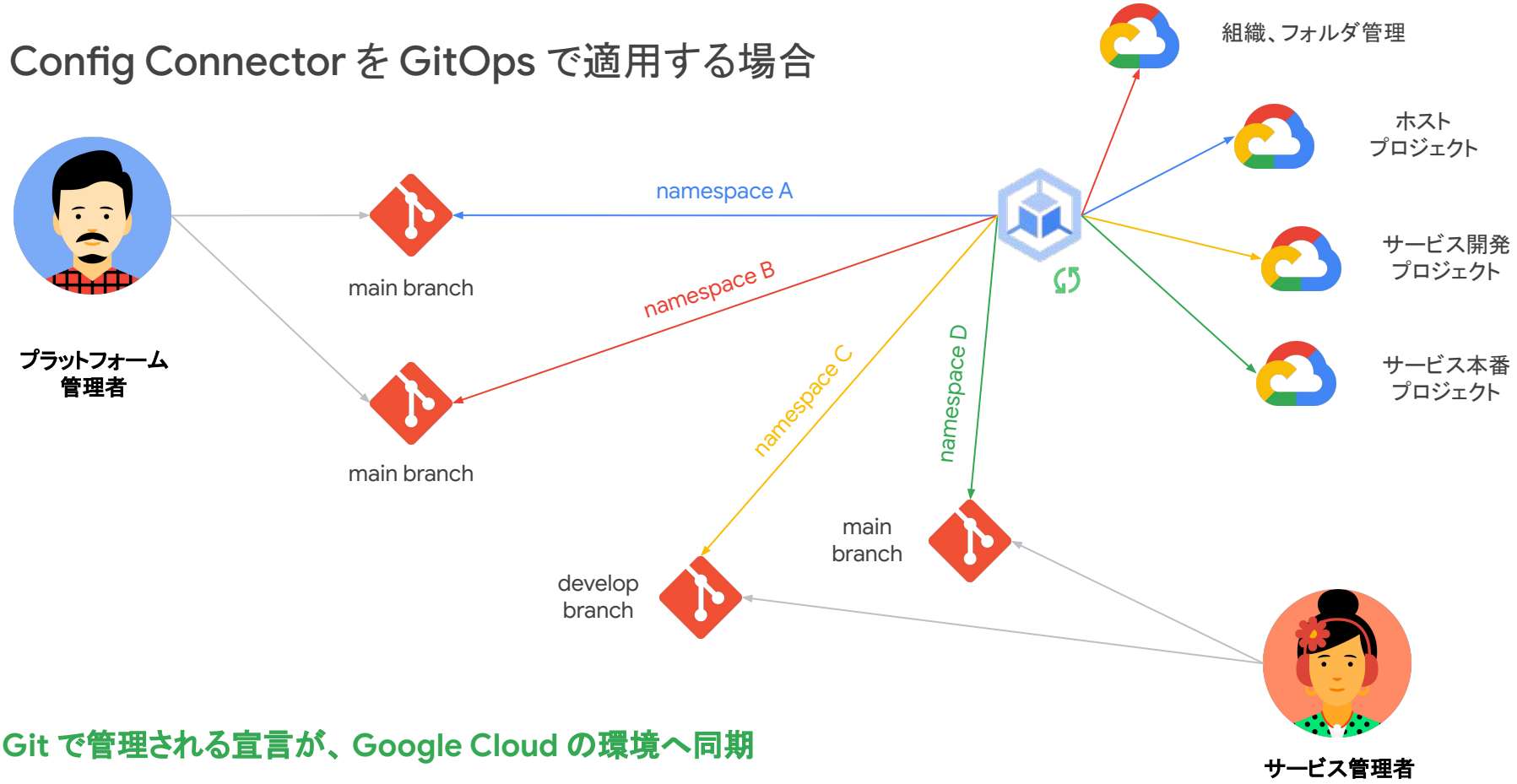
nomos vet

クラスタの状態確認

nomos status [--contexts *CONTEXT*]

```
$ tree
.
├── README.md
├── cluster
├── clusterregistry
├── namespaces
│   └── dev
│       └── dev-project-A
│           └── namespace.yaml
│           └── spanner.yaml
│       └── dev-project-B
│           └── namespace.yaml
│   └── prod
│       └── prod-project-A
│           └── namespace.yaml
│       └── prod-project-B
│           └── namespace.yaml
└── system
    ├── README.md
    └── repo.yaml
```

Config Connector を GitOps で適用する場合



Git で管理される宣言が、Google Cloud の環境へ同期

さらに、インフラの変更をポリシーで制御したい

- インフラ適用でいくつかのポリシーを決めて、制限をかけたい
  - Cloud Storage バケットで、公開バケットを作成してはいけない
  - オーナー権限を持つ Service Account を作成してはいけない
  - 特定のインスタンス タイプでしか Compute Engine を作成してはいけない

など

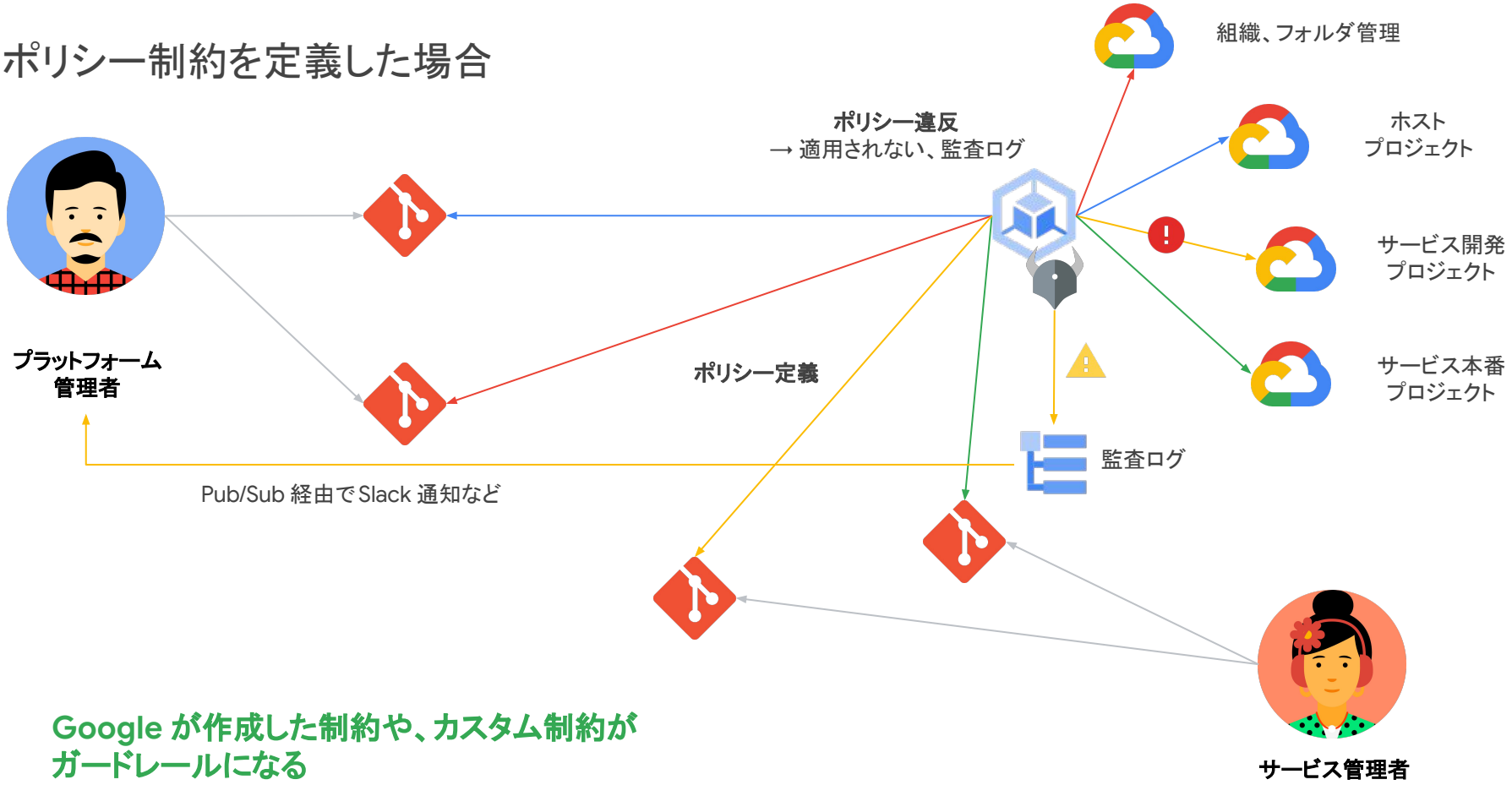
Policy Controller の利用

- GKE がコントロール プレーンとなるため、Kubernetes エコシステム（OSS）の利用が可能
- Open Policy Agent (OPA) Gatekeeper を利用すると、制約を使ってポリシーを定義することができ、デプロイの動作を許可、または拒否することが可能
- Rego と呼ばれる Native Query Language を使って制約テンプレートを記述（右図）
- Gatekeeper をベースにした、Policy Controller という、Anthos が提供する機能がある

```
  targets:
  - target: admission.k8s.gatekeeper.sh
    rego: |
      package gcpstoragelocationconstraintv1

      allowedLocation(reviewLocation) {
          locations := input.parameters.locations
          satisfied := [ good | location = locations[_]
                                good = lower(location) ==
lower(reviewLocation)]
          any(satisfied)
      }

      exempt(reviewName) {
          input.parameters.exemptions[_] == reviewName
      }

      violation[{"msg": msg}] {
          bucketName := input.review.object.metadata.name
          bucketLocation := input.review.object.spec.location
          not allowedLocation(bucketLocation)
          not exempt(bucketName)
          msg := sprintf("Cloud Storage bucket <%v> uses a
disallowed location <%v>, allowed locations are %v",
[bucketName, bucketLocation, input.parameters.locations])
      }
      ...
```

ポリシーを遵守したGoogle Cloud リソースを作成する
https://cloud.google.com/architecture/policy-compliant-resources

ポリシー制約を定義した場合



Google が作成した制約や、カスタム制約が
ガードレールになる

Config Connector と Config Sync デモ

デモの流れ

1. Config Connector の適用
2. Google Cloud リソースを Git リポジトリに定義
3. Config Sync の適用
4. 同期された Google Cloud リソースを確認

まとめ

04

まとめ

- インフラ構築の自動化には、IaC が使われることが多い
- CaD で実現する方法もあり、Kubernetes の機構が利用されることが特徴
  - Reconciliation Loop や、Gatekeeper によるポリシー制御など
  - yaml で一元管理することで、インフラとアプリ基盤のコード管理が統一される
- GitOps の手法でデプロイすることで、多くのメリットがある
  - Git 上の宣言が Single Source of Truth となり、各環境に自動的に同期される
  - Git リポジトリ、ブランチ、ディレクトリ階層を考えるだけで複数環境を管理可能

Thank you