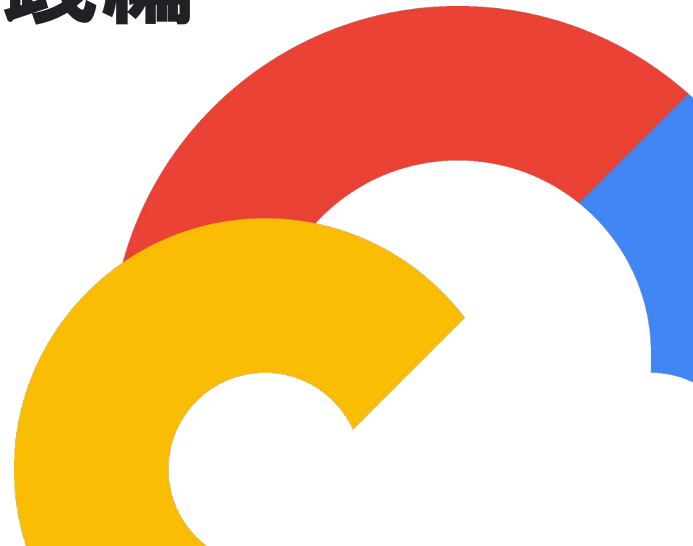
Google Cloud

実践 Google Cloud ハンズオン セミナー

Google Cloud で始める Platform Engineering ~実践編~



2024 年 11 月 8日

はじめるまえに

準備 - Qwiklabs アカウントの作成

事前に作成済の方は実施不要です

1. https://explore.qwiklabs.com/ にア

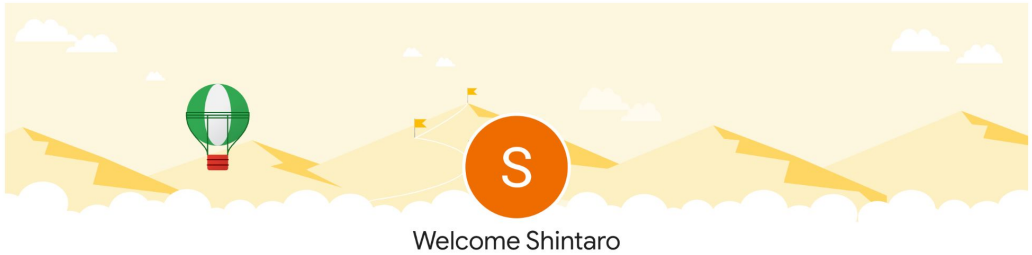
準備 - Qwiklabs アカウントの作成

事前に作成済の方は実施不要です

2. 氏名・メールアドレス・会社とパスワードを入力して [**アカウントを作成**] をクリック
   - ⚠️重要!! メールアドレスは 本ハンズオン 登録時に入力いただいた Qwiklabs メールアドレスを入力してください
3. *Qwiklabs へようこそ* という件名のメールが届いたら、本文中の [**メールアドレスを確認**] をクリック

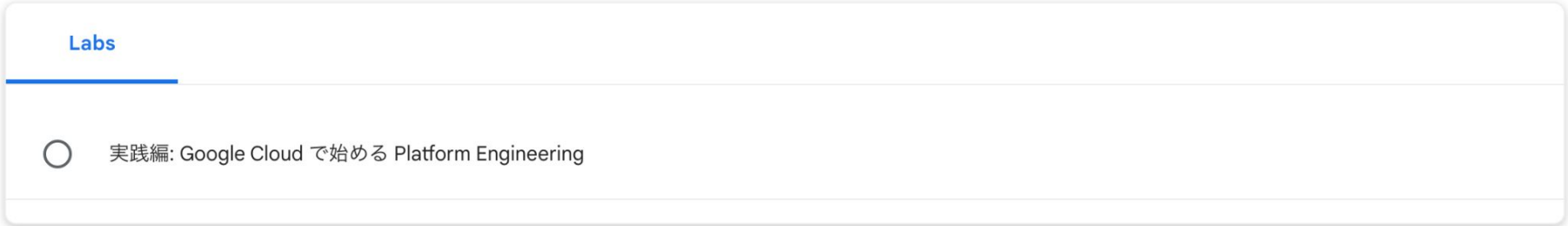
準備 - Lab の開始

4. https://explore.qwiklabs.com/ にアクセス、ログインし、[**実践編:Google Cloud で始める Platform Engineering** をクリック



準備 - Lab の開始

5. Labs タブで **実践編:Google Cloud で始める Platform Engineering** をクリック



準備 - Lab の開始

6. [**ラボを開始**] or [Start Lab] をクリックし、ラボが起動し表示が変わるのを待つ

※表示時間は異なる可能性がございます。

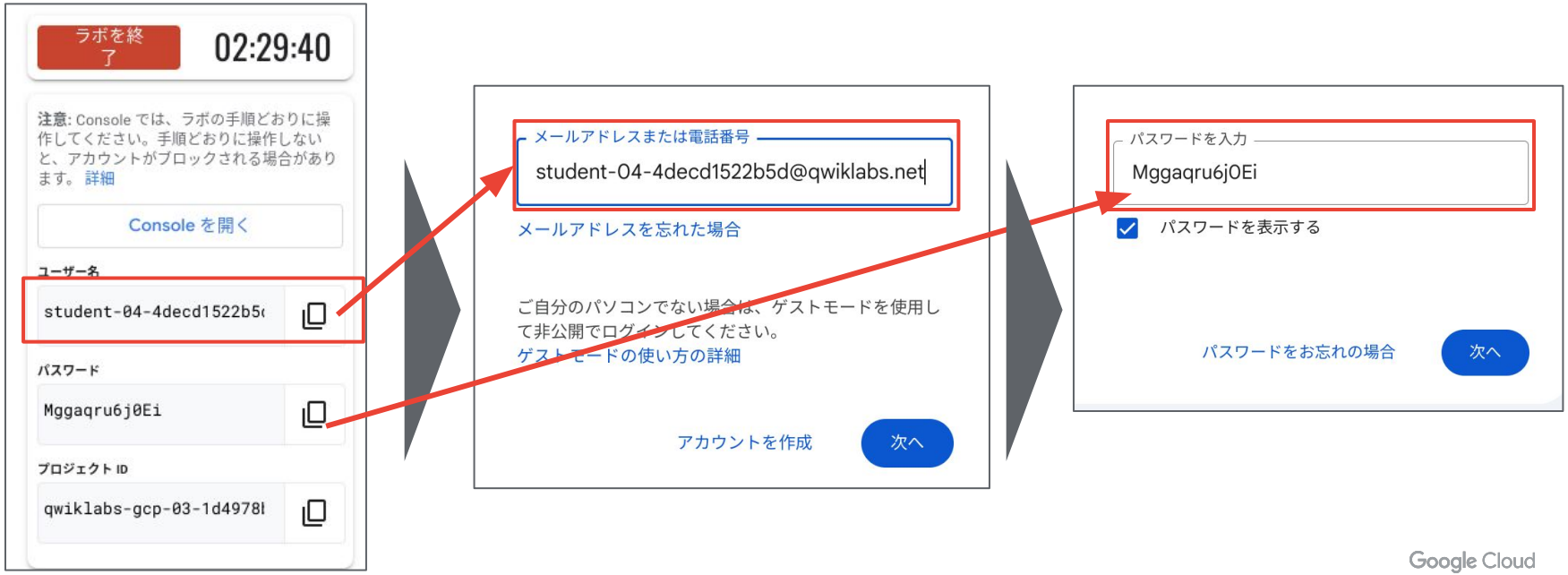
ラボ起動後の表示

準備 - Lab の開始

ブラウザのゲストモード
またはシークレットウィンドウで開きます

4. [**Console を開く**] を右クリックし、[**シークレットウィンドウで開く**] を選択

5. ユーザー名 と パスワード を入力し、[**次へ**] をクリック



準備 - ハンズオンの開始

6. [**Cloud Shell をアクティブにする**] をクリックし、画面下部に Cloud Shell のターミナルが開くのを待つ

準備 - ハンズオンの開始

7. 以下のコマンドをコピーし、Cloud Shell ターミナルで実行してチュートリアルを開く

```
cd ~; git clone https://github.com/GoogleCloudPlatform/gcp-getting-started-lab-jp
cd gcp-getting-started-lab-jp/pfe-adv-sep
teachme tutorial.md
```

事前準備：Lab0. GKE クラスタのデプロイ

GKE クラスタのデプロイに時間がかかるため、事前にデプロイしておきます

チュートリアル **Step 5/9** の「**Lab0. GKE クラスタのデプロイ**」まで実施してください

ハンズオンを 始める

事前準備：Lab0. GKE クラスタのデプロイ

GKE クラスタのデプロイに時間がかかるため、事前にデプロイしておきます

チュートリアル **Step 5/10** の「**Lab0. GKE クラスタのデプロイ** 」まで実施してください

チュートリアルを再度起動する

1. チュートリアル リソースがあるディレクトリに移動する

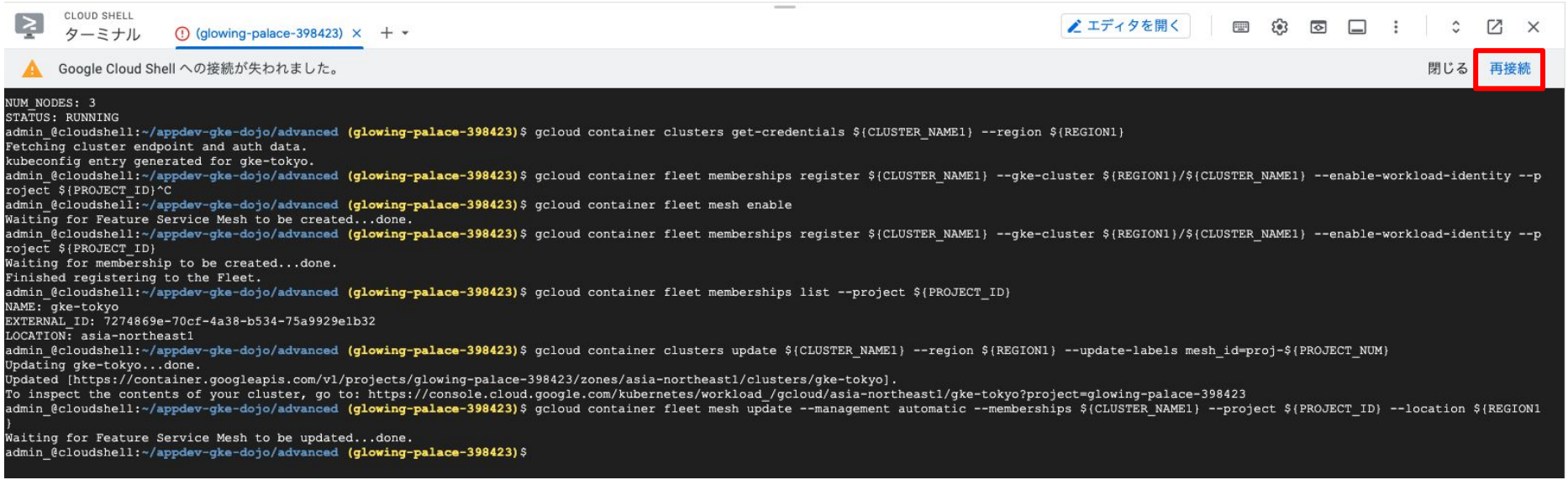
```
cd /$HOME/gcp-getting-started-lab-jp/pfe-adv-sep
```

2. チュートリアルを開く

```
teachme tutorial.md
```

3. ステップ 4/9 の “**参考: Cloud Shell の接続が途切れてしまったときは ?**” まで進み
手順 3, 4 を実行する

Cloud Shell への再接続



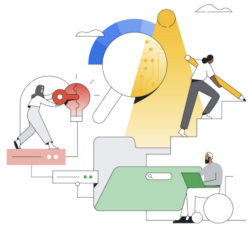
内部開発者プラットフォーム で CI/CD パイプラインを提供する

開発生産性を高めるためのプラットフォーム

プロビジョニングの自動化

インフラを抽象化して自動化することで、プラットフォーム利用者の認知負荷を軽減

アプリケーション ライブラリ



新しいアプリケーションを素早くブートストラップするために、テンプレートのライブラリを用意

セルフサービス UX

新しいアプリや機能の導入は、ツール、ポータルUI、またはその両方を通じたセルフサービス

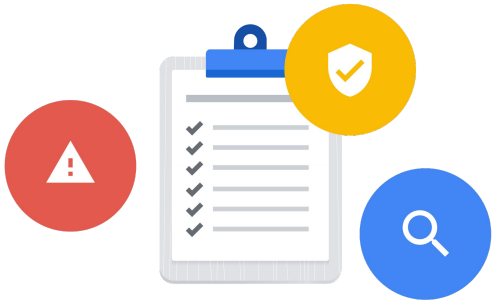
ガバナンスとガードレール

コスト管理、セキュリティ、ガバナンスがプラットフォームに標準組み込み

CI と CD の分離

昨今の **新しい継続的デリバリー (CD) の概念・手法** は
継続的インテグレーション (CI) と CD を分けて考え、実装することで
既存課題の解決策、または継続的改善の礎になりえるものです。

- ロールアウト / ロールバックにかかる**リードタイムの短縮**
- より柔軟で一貫性のある **成果物管理**
- **要件に応じ、柔軟に** 選べるデプロイ手段
- リリース履歴など **DevOps 指標の可視化**
- 適切な粒度 & 最小権限の原則による**権限・監査設定**



CI/CD とは

CI（Continuous Integration、継続的インテグレーション）と CD（Continuous Delivery/Continuous Deployment、継続的デリバリー/継続的デプロイメント）は、ソフトウェア開発のプラクティスであり、特に Kubernetes ベースの開発において重要である。

CI（継続的インテグレーション）

開発プロセスにおいて、チームメンバーが頻繁に（一日に数回以上）コード変更をメイン リポジトリに統合するプラクティス。主な目的は、ソフトウェアの品質を向上させ、リリースプロセスを加速すること。CI のプロセスでは、コードの変更がリポジトリにプッシュされるたびに自動化されたビルドとテストが行われ、問題を早期に発見し修正することが可能となる。

CD（継続的デリバリー、Continuous Delivery）

ソフトウェアをリリース可能な状態に保ち、リリース プロセスを自動化するプラクティス。継続的デリバリーでは、ビルド、テスト、リリース プロセスが自動化され、最終的なリリースの決定は人間が行うもの。

継続的デプロイメント（Continuous Deployment）

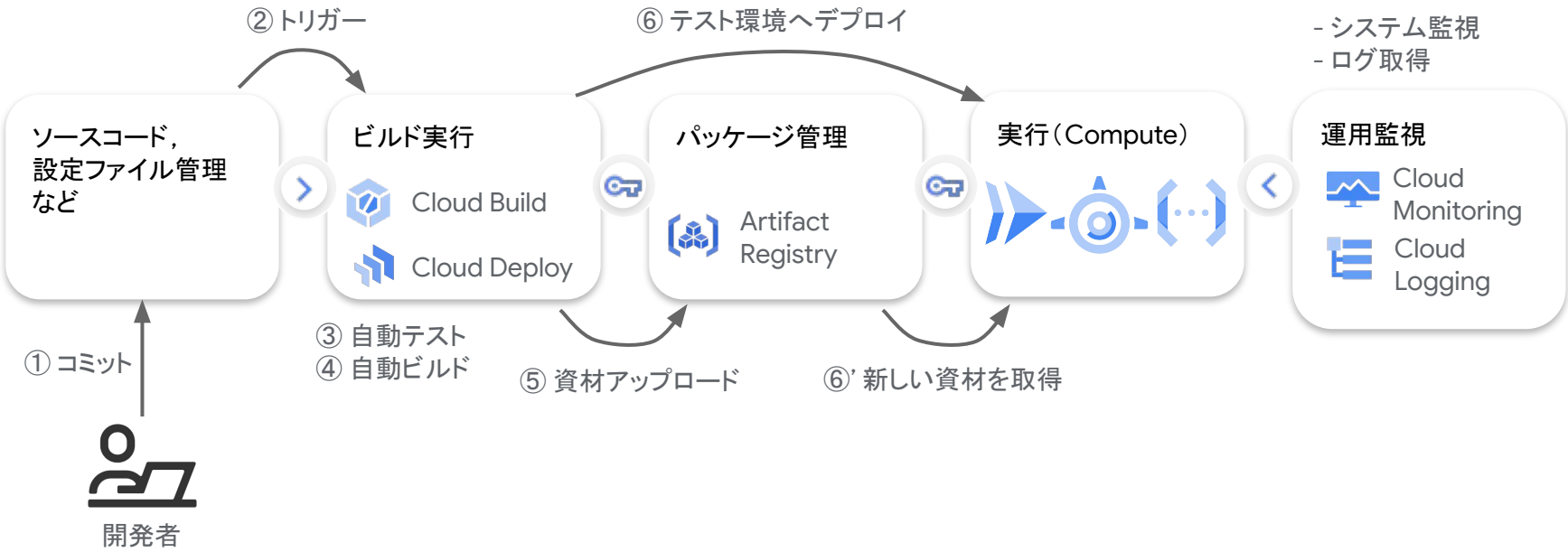
継続的デリバリーをさらに一歩進めたもので、変更が自動的に本番環境にデプロイされるプロセス。この場合、テストをパスしたビルドは、人間の介入なしに自動的に本番環境にリリースされる。

CI/CD の各プロセス

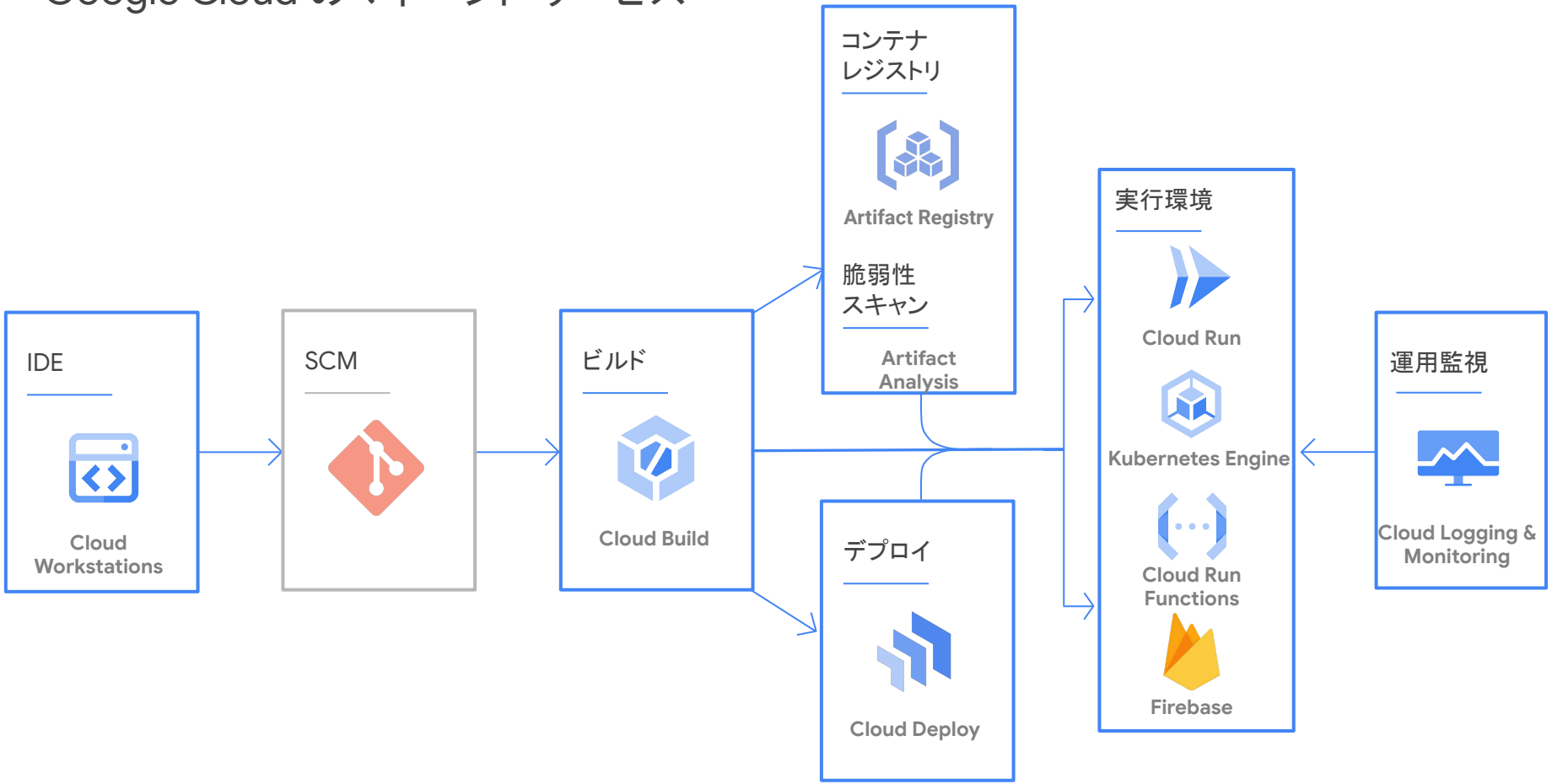
- コードコミット
  - 開発者が変更をリポジトリにコミット
- ビルド自動化
  - コミットされたコードの自動ビルド
- 自動テスト
  - ビルドに対する自動テストの実行
- フィードバック
  - 開発者への実行結果の送信

- リリース準備
  - ビルドをステージング環境に自動移行
- 継続的デリバリー
  - 手動での本番環境へのデプロイ承認
- 継続的デプロイメント
  - 変更を本番環境に自動デプロイ
- フィードバック
  - 本番環境での結果からの学習と改善

Google Cloud における CI / CD パイプラインの例



Google Cloud のマネージド サービス



Cloud Build

デベロッパーフレンドリー

CSR、GitHub または Bitbucket での変更をトリガーに

柔軟なビルドステップ

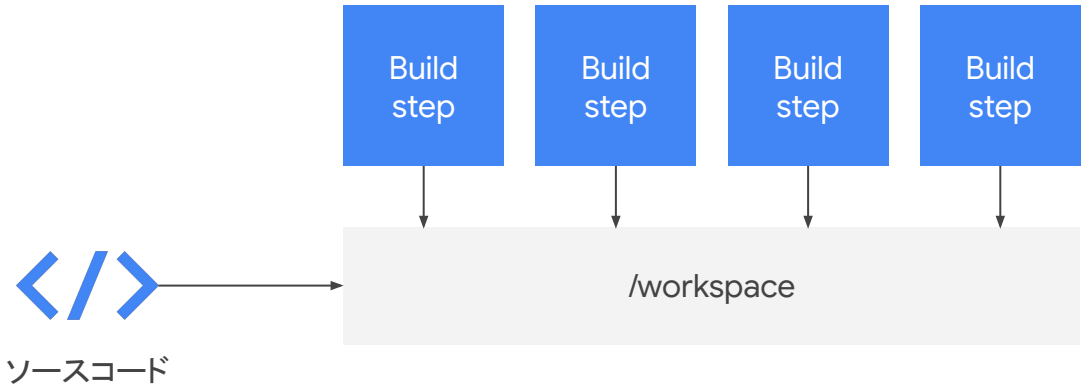
あらゆる CLI ツールをビルドステップとして
組み込むことが可能

フルマネージド CI プラットフォーム

お客様が VM を用意したりキャパシティの管理をする必要はない

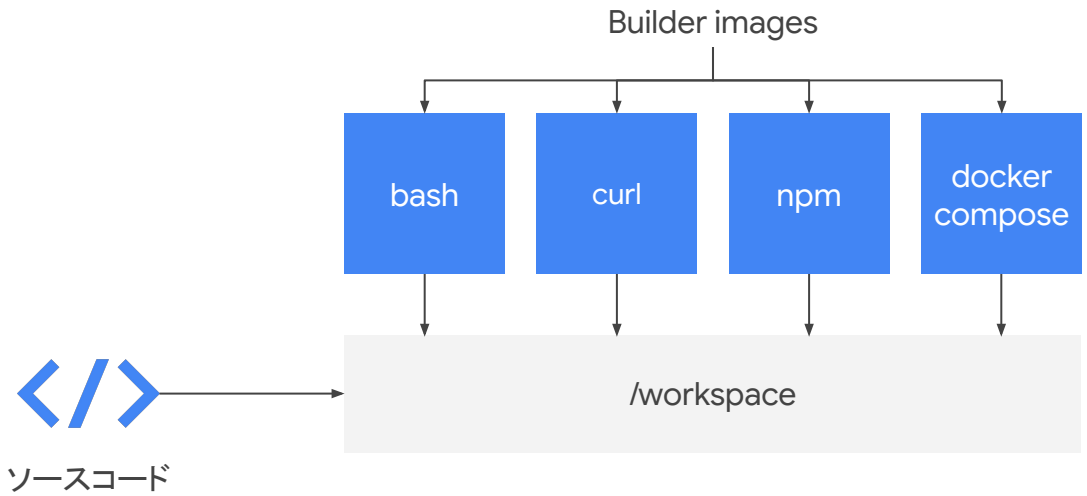
Cloud Build

ビルドステップとして テスト・ビルドなど
柔軟に CI / CD プロセスを実行可能



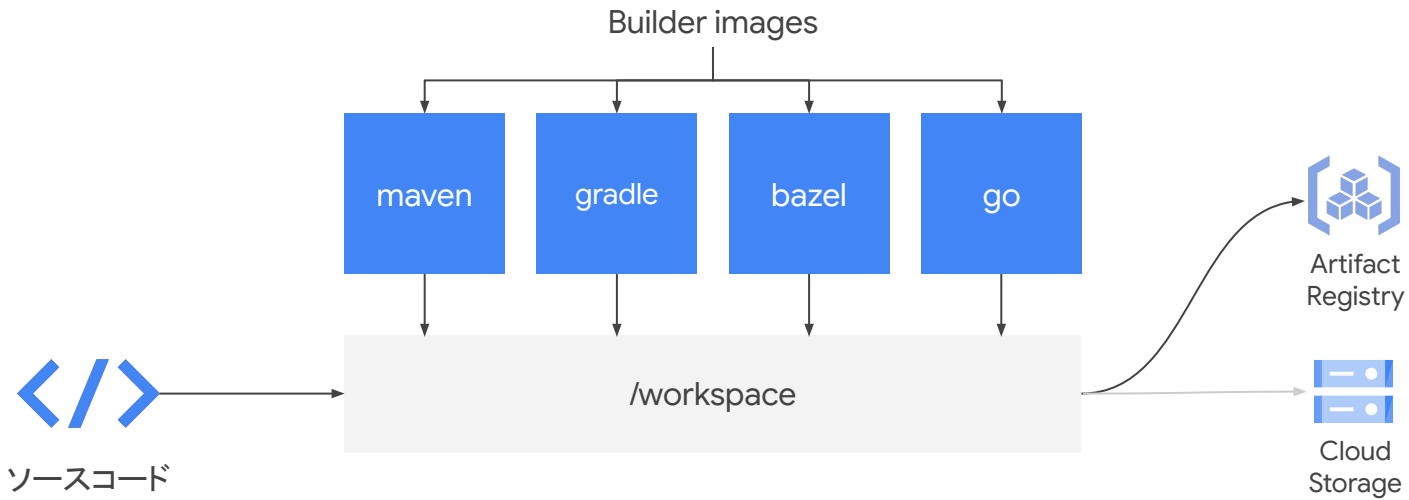
Cloud Build

テストしたり



Cloud Build

ビルドや成果物の保存ができます



Cloud Deploy

継続的デリバリー (CD) に特化

CD のための各種機能をフルマネージドでご提供します
CI はこれまでのパイプラインで実施、本機能はデプロイのみを担当

成果物の厳密な管理

リリース コンテンツを事前にまとめ、
環境依存のない一貫性ある成果物管理

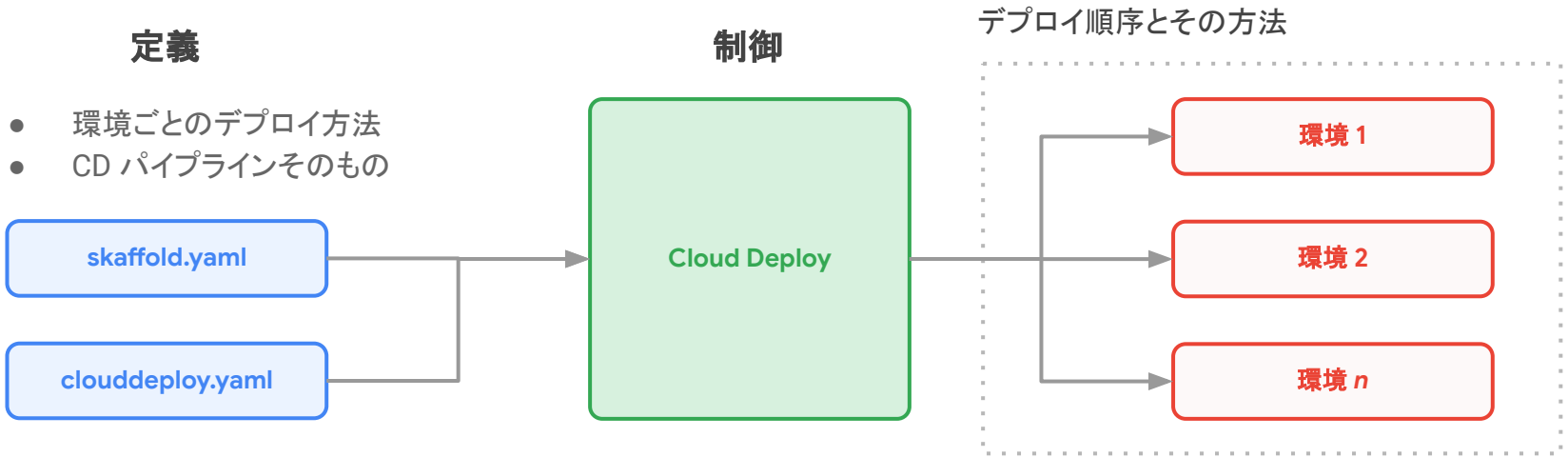
重要指標の可視化

CI / CD プロセスそのものの改善を促す指標を可視化
DORA 4 指標の “デプロイ頻度” と “デプロイ時失敗率” を組込み

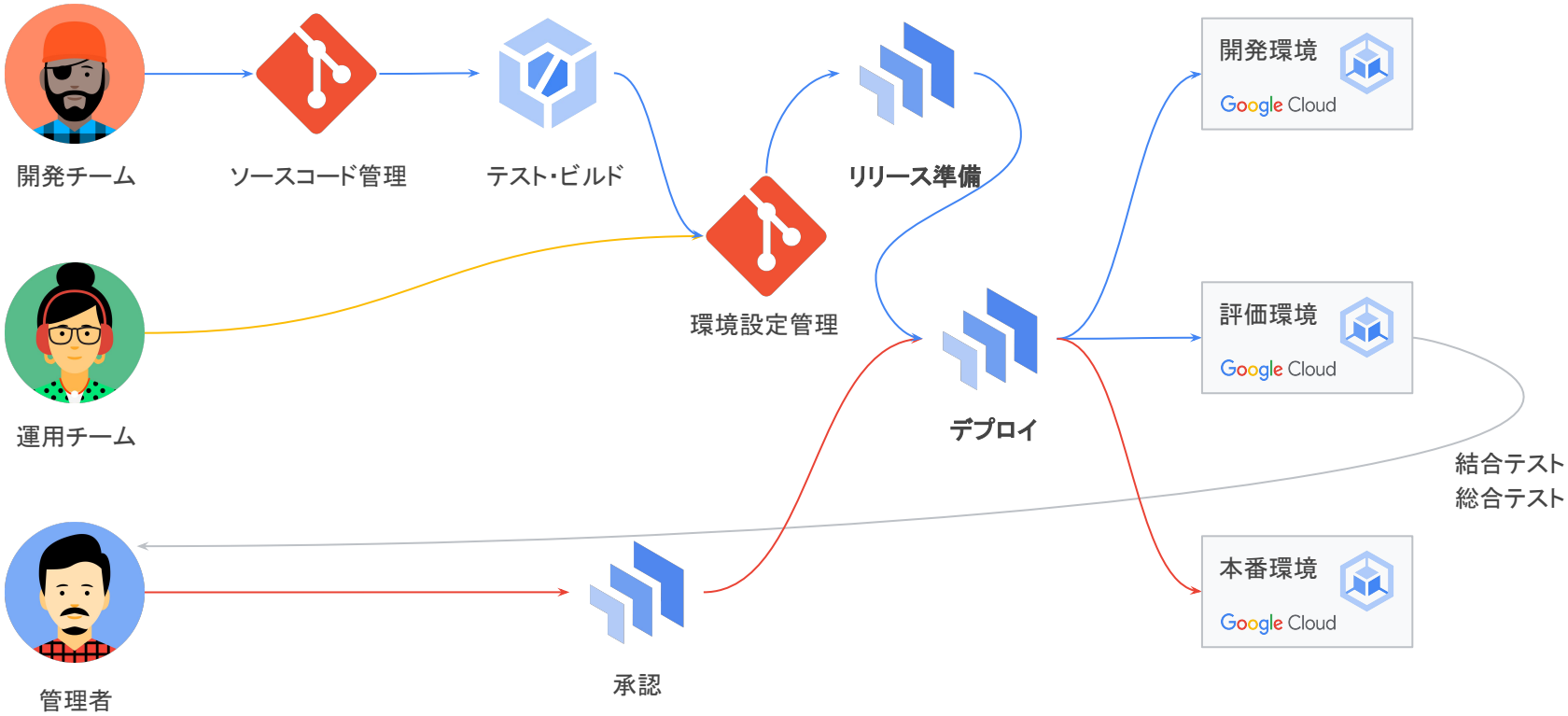
Cloud Deploy | Google Cloud
https://cloud.google.com/deploy?hl=ja

Cloud Deploy のアーキテクチャ

デプロイしたい単位（製品やサービス）ごとに
デプロイ先の環境、デプロイ順序、デプロイ方法が制御できます。



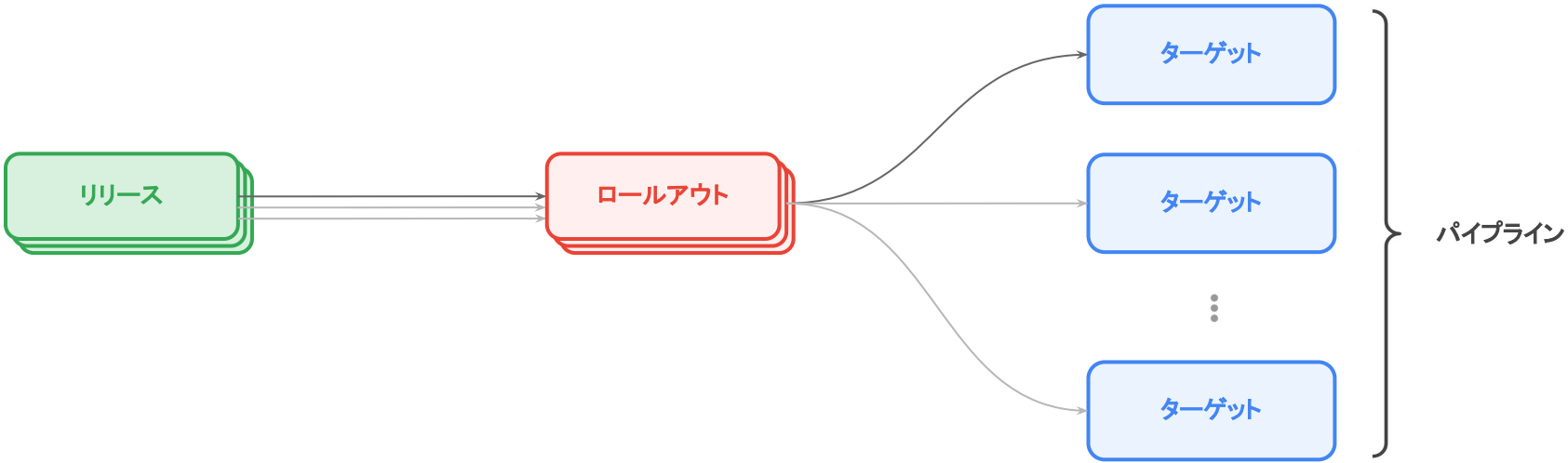
Cloud Build と Cloud Deploy で作る CI / CD 環境例



Cloud Deploy によるデプロイの流れ

予め **“パイプライン”** として定義した各 **“ターゲット”** に対し

成果物を **“リリース”** としてまとめ、順々に **“ロールアウト”** していく。

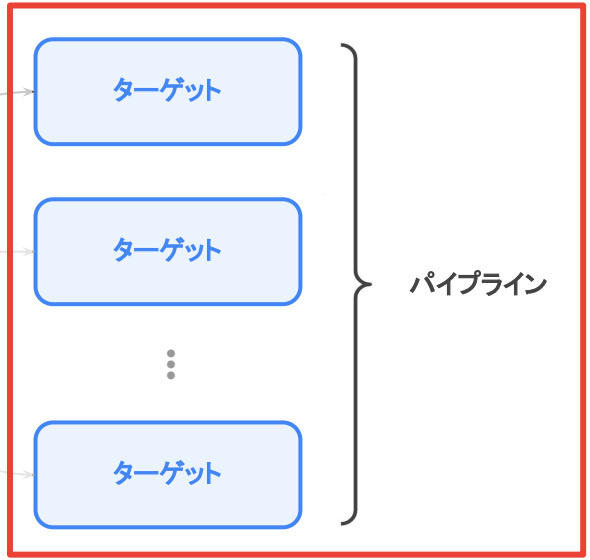


パイプラインとターゲット

デプロイ先の環境を “ターゲット”、
各ターゲットに、どういう順序でどうデプロイするかを “パイプライン” として定義する。

```
# パイプライン

apiVersion: cloudDeploy/beta1
kind: delivery-pipeline
name: <name>
description: <description>
serialPipeline:
   stages:
   - targetId: <id>
     profiles:
        - <skaffold profile(s)>
   ...
   - targetId: <id>
     Profiles:
        - <skaffold profile(s)>
```

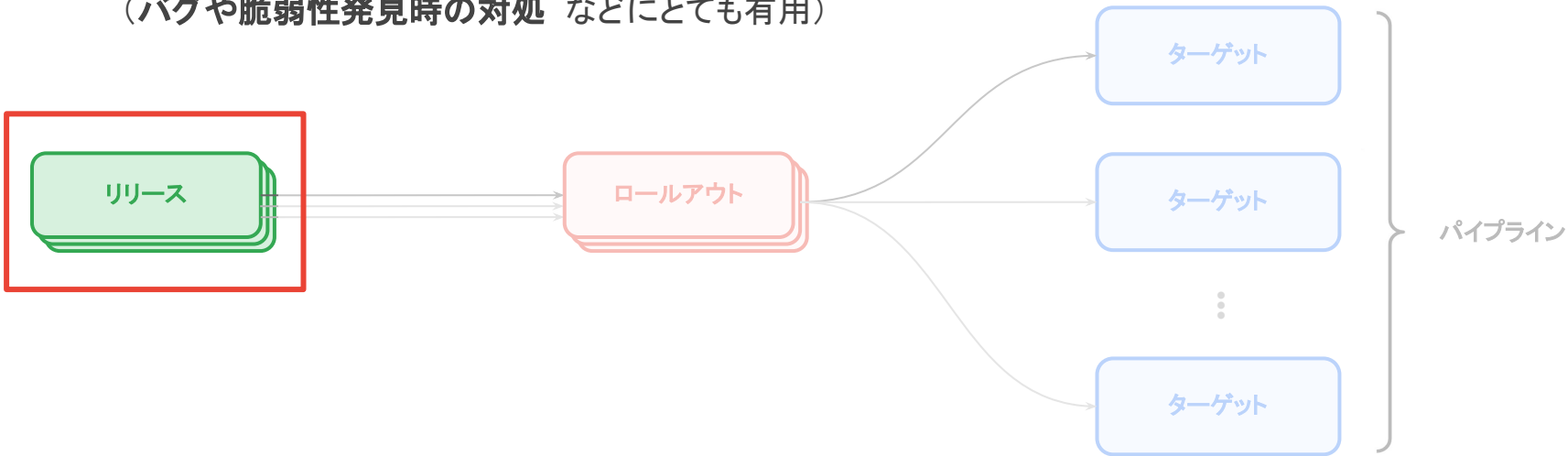
```
# ターゲット

apiVersion: cloudDeploy/beta1
kind: target
name: <name>
delivery-pipeline: <parent>
description: <description>

gkeCluster:
   project: <project name>
   cluster: <cluster name>
   location: <location>
```



リリース

ビルドしたイメージを、**一緒にデプロイしたい単位** “リリース” にまとめます。結果として、

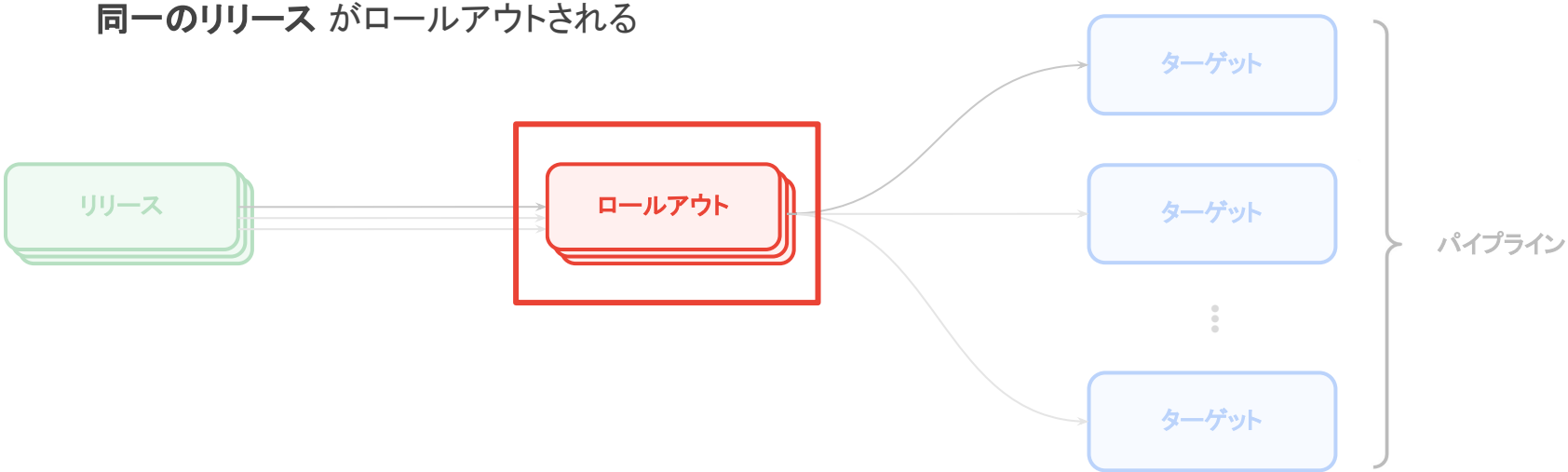
- 各環境では、どのリリース バージョンが動いているかがひと目で確認できます
- 逆に「このリリース バージョンはどの環境で稼働中か」もわかります
  （**バグや脆弱性発見時の対処** などにとても有用）



ロールアウト

リリースを、ターゲットにロールアウトする度に作られるリソース。

- 初回リリース作成時には、すぐに最初のターゲットにロールアウトされる
- 2 つ目以降のターゲットに対しては、**“プロモーション ”** することで
  同一のリリース がロールアウトされる



Cloud Deploy でデプロイするまでの流れ *Step 1 / 6*

skaffold を使い、気軽にホット リロード & デバッグでローカル開発


```
$ git clone git@github.com:your-org/your-prj
$ skaffold dev
```

Skaffold https://skaffold.dev/

Cloud Deploy でデプロイするまでの流れ *Step 2 / 6*

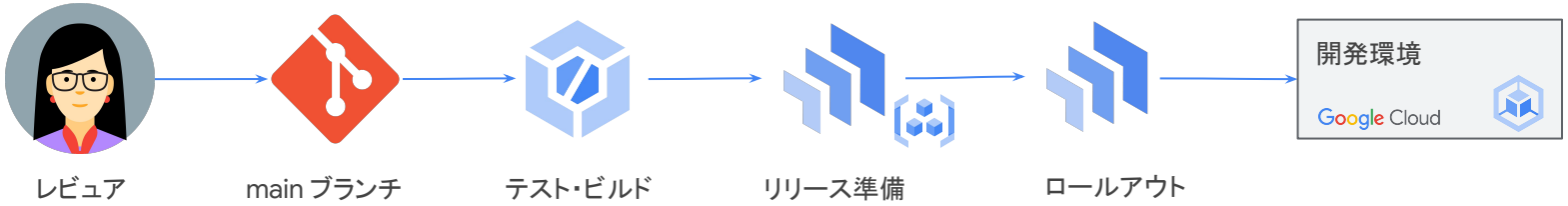
feature ブランチへ push、PR を作り、自動テスト結果とともにコードレビュー依頼


```
$ git push
```

Cloud Deploy でデプロイするまでの流れ *Step 3 / 6*

PR が承認され、コードが main ブランチへマージされたら

自動でコンテナを build & push、Cloud Deploy に**リリース対象** をまとめ、開発環境へ **ロールアウト**

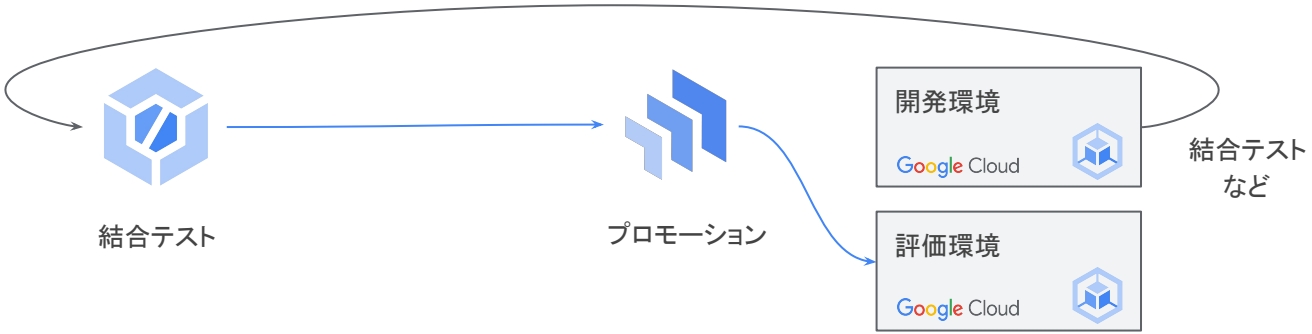



```
$ gcloud deploy releases create <release-name> \
    --delivery-pipeline <pipeline-name> \
    --image=<image-name>:digest
```

Cloud Deploy でデプロイするまでの流れ *Step 4 / 6*

開発環境での（自動）テストが問題なければ（ Pub / Sub 経由のトリガ起動で）

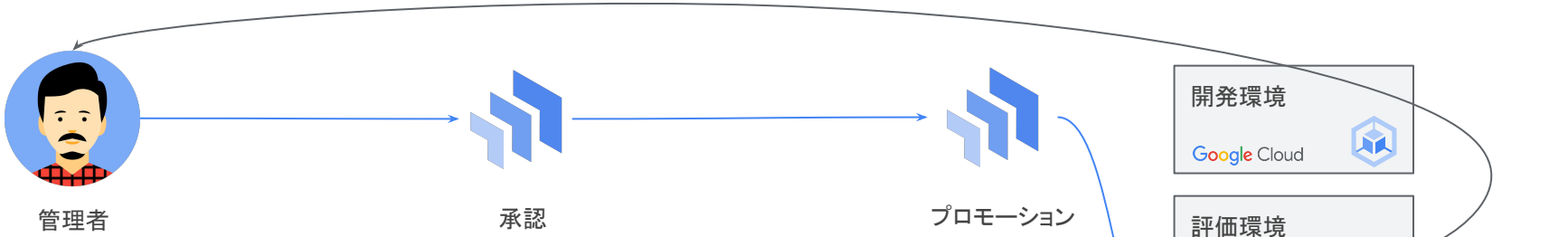
そのまま自動的に、評価環境へ “開発環境と同一のリリース対象 ” を**プロモーション**




```
$ gcloud deploy releases promote \
    --delivery-pipeline <pipeline-name> \
    --release <release-name>
```

Cloud Deploy でデプロイするまでの流れ *Step 5 / 6*

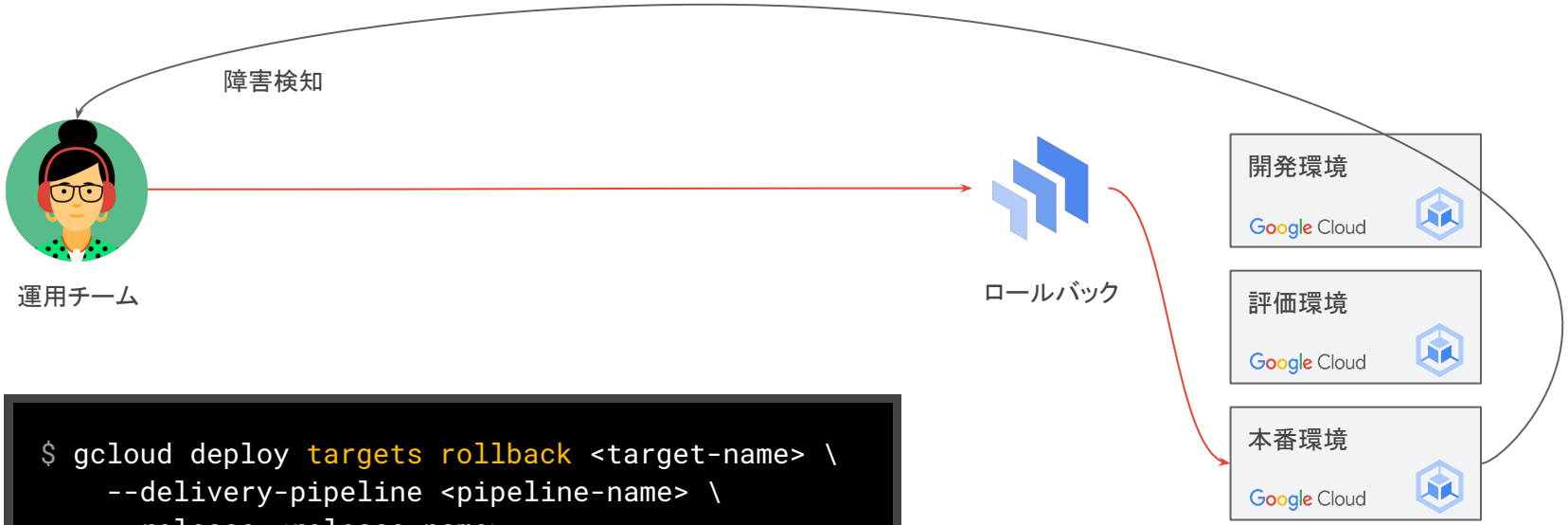
評価環境でも問題なければ、管理者の **承認** を得て、本番環境へプロモーション




```
$ gcloud deploy rollouts approve <rollout-name> \
    --delivery-pipeline <pipeline-name>
```

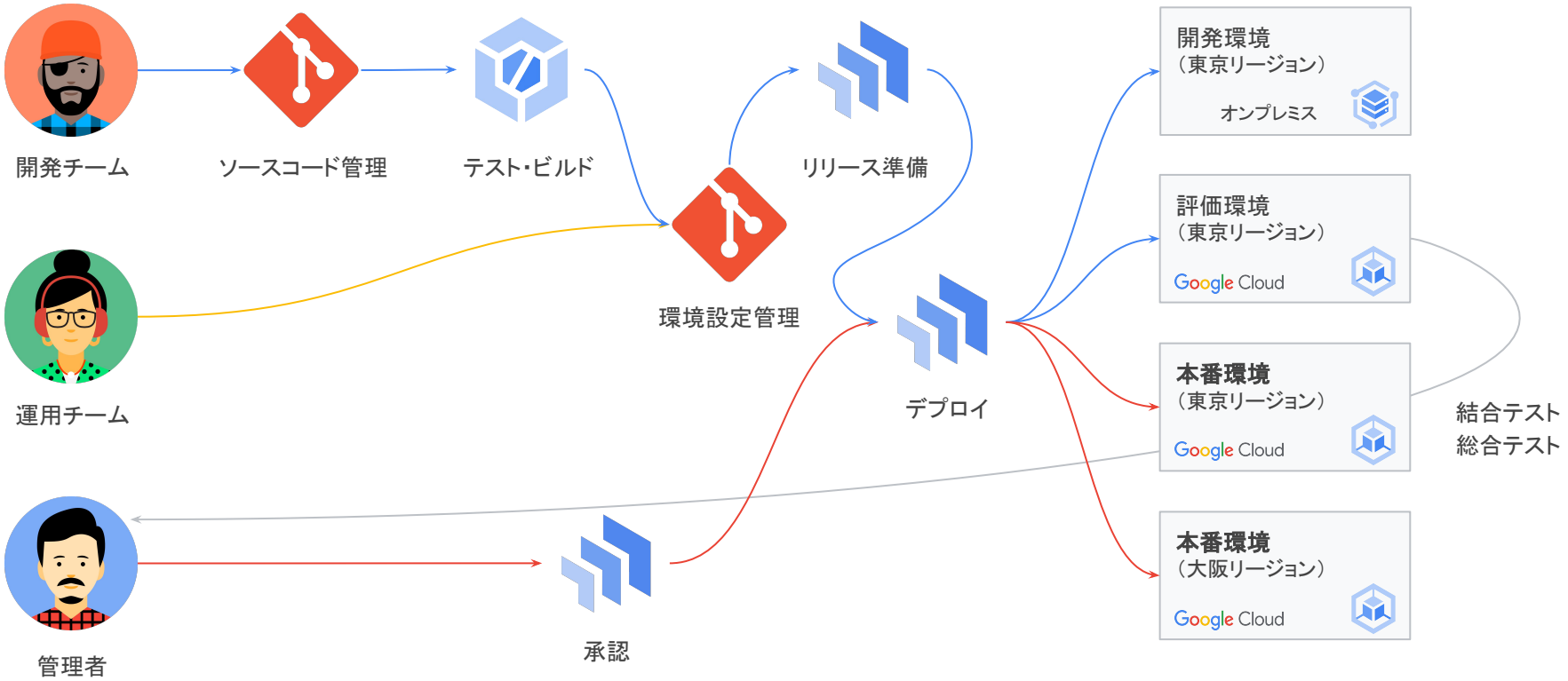
Cloud Deploy でデプロイするまでの流れ *Step 6 / 6*

本番環境から何らかの不具合が通知されたら、直前のリリースを **ロールバック**




```
$ gcloud deploy targets rollback <target-name> \
    --delivery-pipeline <pipeline-name> \
    --release <release-name>
```

モダンな CI / CD パイプライン（CI と CD の分離）



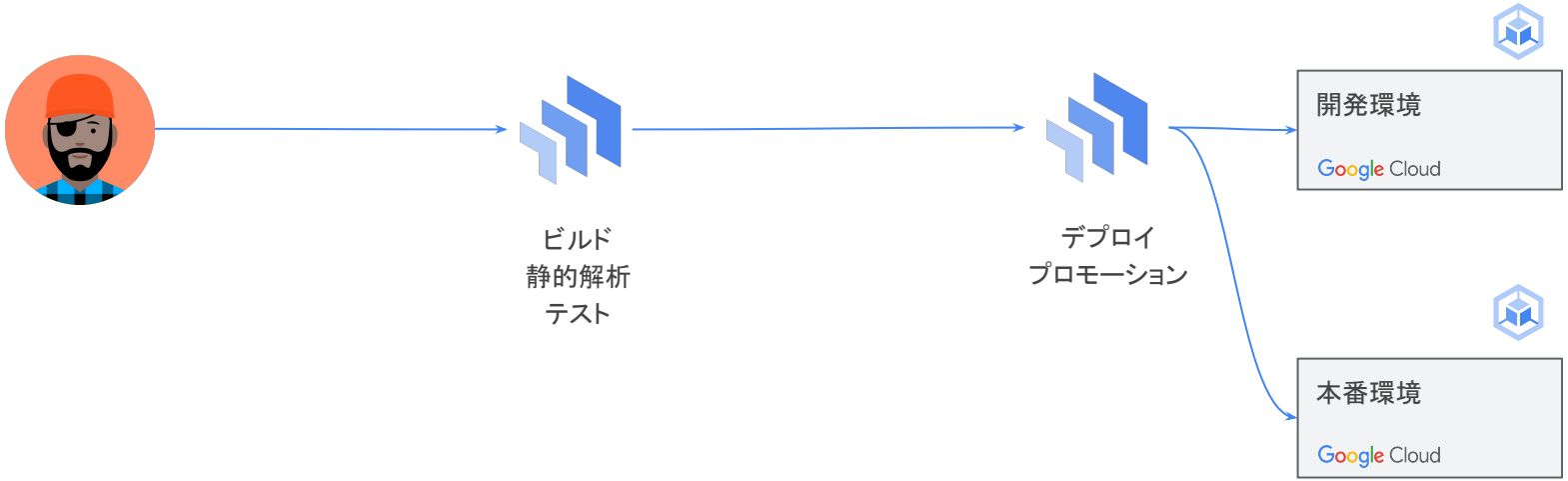
その他、Cloud Deploy でできること

- カナリーリリース
  - トラフィックを段階的に新しいバージョンに移行するデプロイ戦略
- カスタム ターゲット
  - GKE (Enterprise), Cloud Run 以外のランタイムに対してデプロイ先を定義する
- デプロイフック
  - デプロイの事前事後に特定のアクションを実行する
- デプロイ後のテスト
  - デプロイ後に任意の簡単なテストを実行する

Lab-01 Google Cloud での CI/CD

チュートリアル **Step 6/9** の「**Lab-01 Google Cloud での CI/CD**」まで実施してください

Python のサンプルアプリケーションを CI/CD パイプラインでデプロイするまでの流れを体感します。



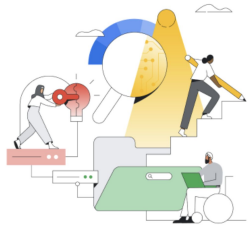
開発者プラットフォームにおける
ガードレール

開発生産性を高めるためのプラットフォーム

プロビジョニングの自動化

インフラを抽象化して自動化することで、プラットフォーム利用者の認知負荷を軽減

アプリケーション ライブラリ



新しいアプリケーションを素早くブートストラップするために、テンプレートのライブラリを用意

セルフサービス UX

新しいアプリや機能の導入は、ツール、ポータルUI、またはその両方を通じたセルフサービス

ガバナンスとガードレール

コスト管理、セキュリティ、ガバナンスがプラットフォームに標準組み込み

クラウド活用における ガードレールの必要性

クラウド活用が進み環境の規模が大きく複雑になってくると、すべての環境で適切な **セキュリティを担保** することが難しくなっていきます

開発における自由度やスケーラビリティを落とさず、セキュリティリスクを低減するためには、マニュアルによる運用ではなく **自動化された仕組みが必要** です

開発者に求められる安全なプラットフォーム

セキュリティは重要だがセキュアな環境を維持するためには様々なレイヤーでの考慮が必要となり、開発者の認知負荷の増大に繋がる可能性がある
また、必要以上に制限の強い環境は開発者の生産性を下げる要因となる

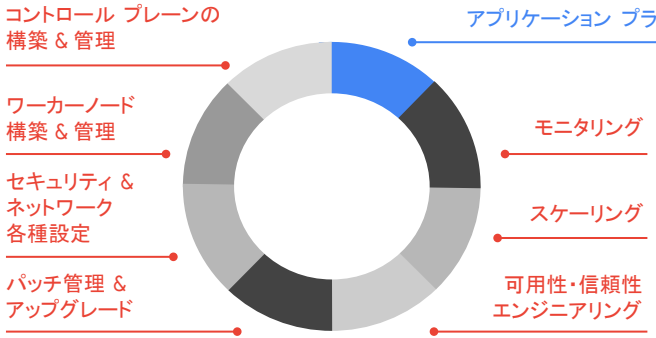
開発者の生産性を妨げない形で、**このプラットフォームを使えば簡単に自社要件に合った セキュリティレベルが保たれる** という状況を作ることが理想

プラットフォーム開発者は自ら提供するプラットフォームのセキュリティ リスクを適切に把握したうえで要件に合った対策をする必要がある

Google Kubernetes Engine (GKE)

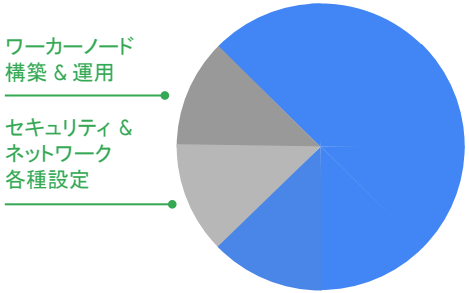
Kubernetes 運用のベストプラクティスを、マネージドサービスとしてご提供

DIY Kubernetes



Google Kubernetes Engine (GKE)

アプリケーション プラットフォーム



Standard モード

設定の柔軟性を兼ね備えた
マネージド Kubernetes

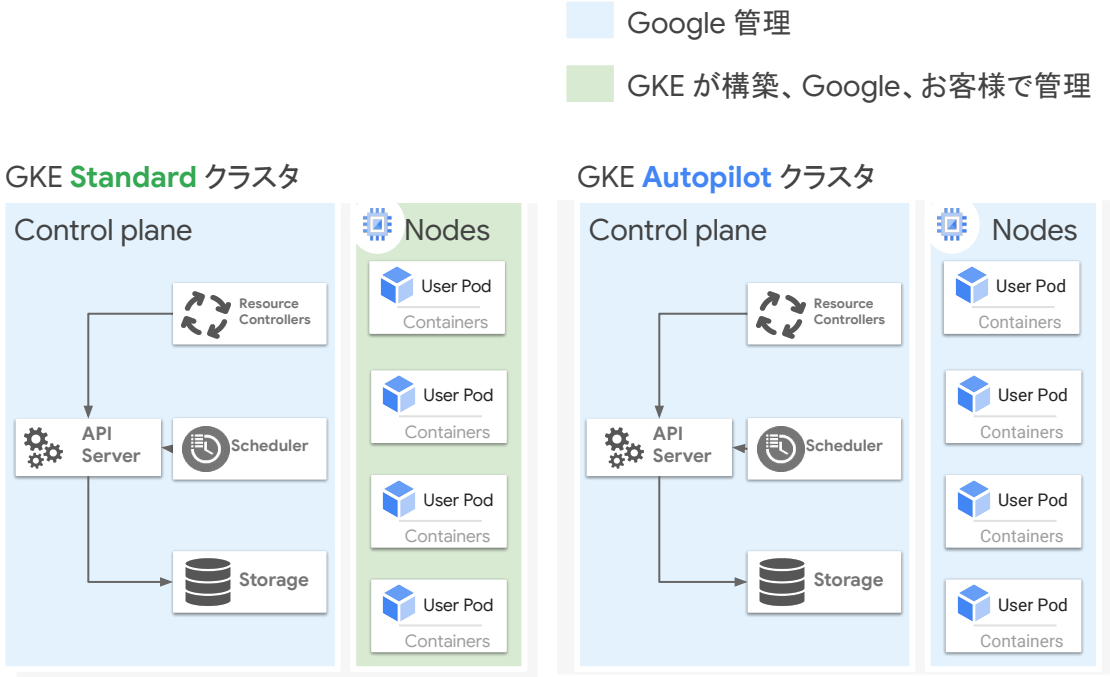
Autopilot モード

より運用負荷が軽減されるよう
最適化されたマネージド
Kubernetes

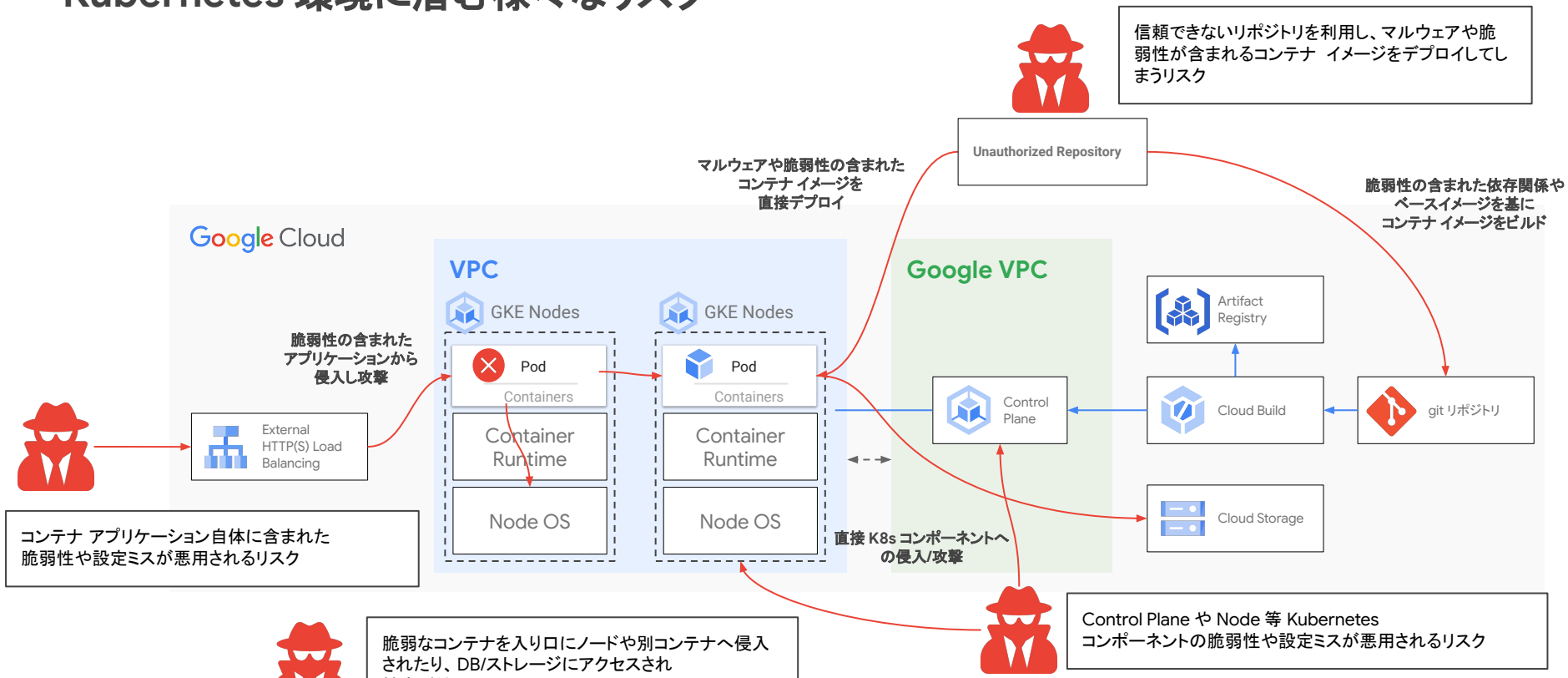
GKE のアーキテクチャ

GKE はクラスタという単位で管理され、
1 つの GKE クラスタは **Control Plane** と
Node というコンポーネントから
構成される

Node は **Node Pool** という単位で
グルーピングされ管理される

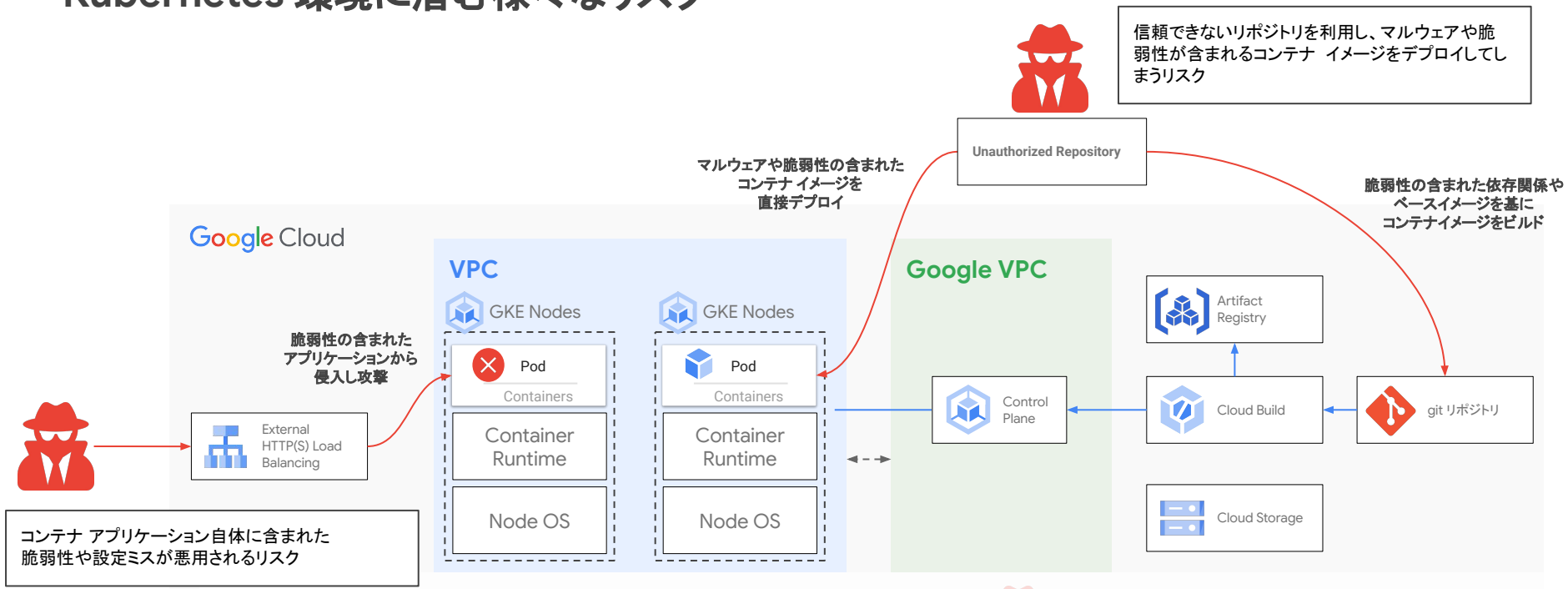


Kubernetes 環境に潜む様々なリスク



などなど…

Kubernetes 環境に潜む様々なリスク



アプリケーション やコンテナ イメージ 脆弱性への対策

アプリケーションやコンテナ イメージ脆弱性への対策

アプリケーションの脆弱性や、コンテナ イメージに含まれる言語 / OS パッケージの脆弱性が放置されていると脆弱性が起点となり攻撃される可能性があるため、リスクを低減するための対策が必要

- **アプリケーション脆弱性への対策**
  - WAF や静的解析ツールなどの導入
- **セキュアな CI/CD パイプラインの提供**
  - CI/CD パイプライン内でのコンテナ イメージのスキャン、静的解析の実行
- **実行環境上での継続的な脆弱性スキャン**
  - CI/CD パイプラインを経由せずデプロイされたアプリケーションの脆弱性スキャン
  - デプロイ後に発見された脆弱性の検知
- **信頼できるアーティファクトのデプロイを保証**
  - CI/CD をバイパスさせない、信頼できないコンテナ イメージをデプロイさせない

Google Cloud Armor

エッジ防御: DDoS & WAF

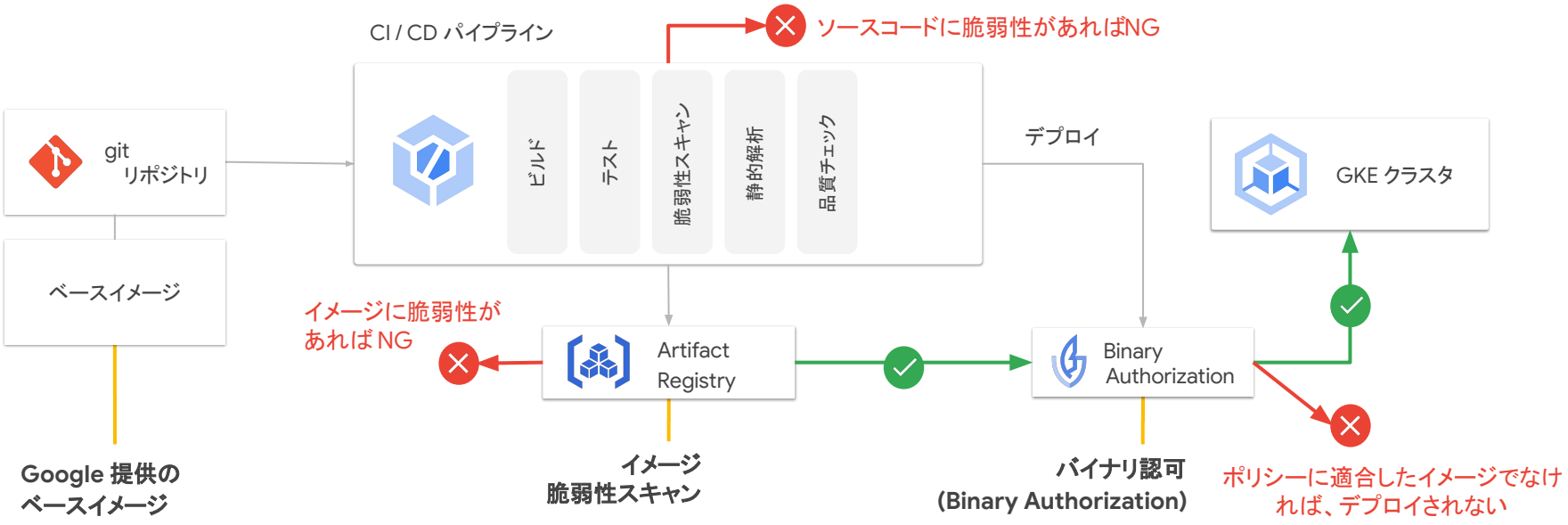
DDoS 攻撃からのインフラ防御
Global HTTP(S) Load Balancing にて（TCP SYN フラッド、増幅攻撃、IP フラグメント攻撃、他）

トラフィックの許可・ブロック
IP アドレス、地域、カスタム パラメータマッチ（L3-L7 他）

アプリケーション レイヤ攻撃からの防御
（SQLi、XSS 他）IAP との組み合わせ

テレメトリ
各種判断結果、メトリクスは Cloud Logging/Monitoring に記録

セキュアなソフトウェア　サプライ チェーンの例



Google 提供のベースイメージ

- **定期的な脆弱性スキャン** が実行され、**最新のセキュリティ パッチが自動的に適用** されたベースイメージを提供
- 利用可能な OS ディストリビューション:

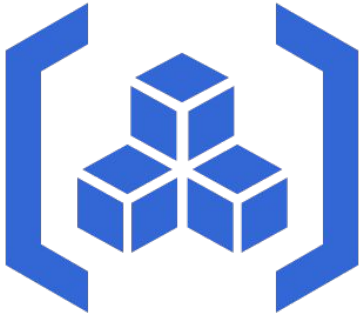
OS	ソース	リポジトリのパス	Google Cloud Marketplace リスティング
CentOS 7	GitHub	marketplace.gcr.io/google/centos7	Google Cloud Marketplace
CentOS 8	GitHub	marketplace.gcr.io/google/centos8	Google Cloud Marketplace
Debian 9 "Stretch"	GitHub	marketplace.gcr.io/google/debian9	Google Cloud Marketplace
Debian 10 "Buster"	GitHub	marketplace.gcr.io/google/debian10	Google Cloud Marketplace
Debian 11「BullsEye」	GitHub	marketplace.gcr.io/google/debian11	Google Cloud Marketplace
Ubuntu 18.04	GitHub	marketplace.gcr.io/google/ubuntu1804	Google Cloud Marketplace
Ubuntu 20.04	GitHub	marketplace.gcr.io/google/ubuntu2004	Google Cloud Marketplace

- また、distroless のような軽量イメージを利用することで **アタック サーフェスを小さくする** ことも可能
  https://github.com/GoogleContainerTools/distroless

Artifact Registry

フルマネージドな
成果物の保存、管理、保護 サービス

- Cloud IAM によるアクセス制御
- コンテナ、Maven、Go の**脆弱性スキャン (Container Analysis)**
- ソフトウェア部品表 (SBOM) の生成



https://cloud.google.com/artifact-registry?hl=ja

Container Analysis

- Common Vulnerabilities and Exposures（CVE）データを取得し、コンテナ イメージの脆弱性の重大度（5 段階）を判定

- **リポジトリへのイメージ　Push 時**や**オンデマンド** での脆弱性スキャンをサポート

- GKE 上のワークロードの継続的スキャンもサポートし、**デプロイ後に発見された脆弱性** や **CI/CD やリポジトリを経由しないコンテナ イメージの脆弱性** も検知
  - OS パッケージ スキャン
  - 言語パッケージ スキャン (GKE Enterprise)

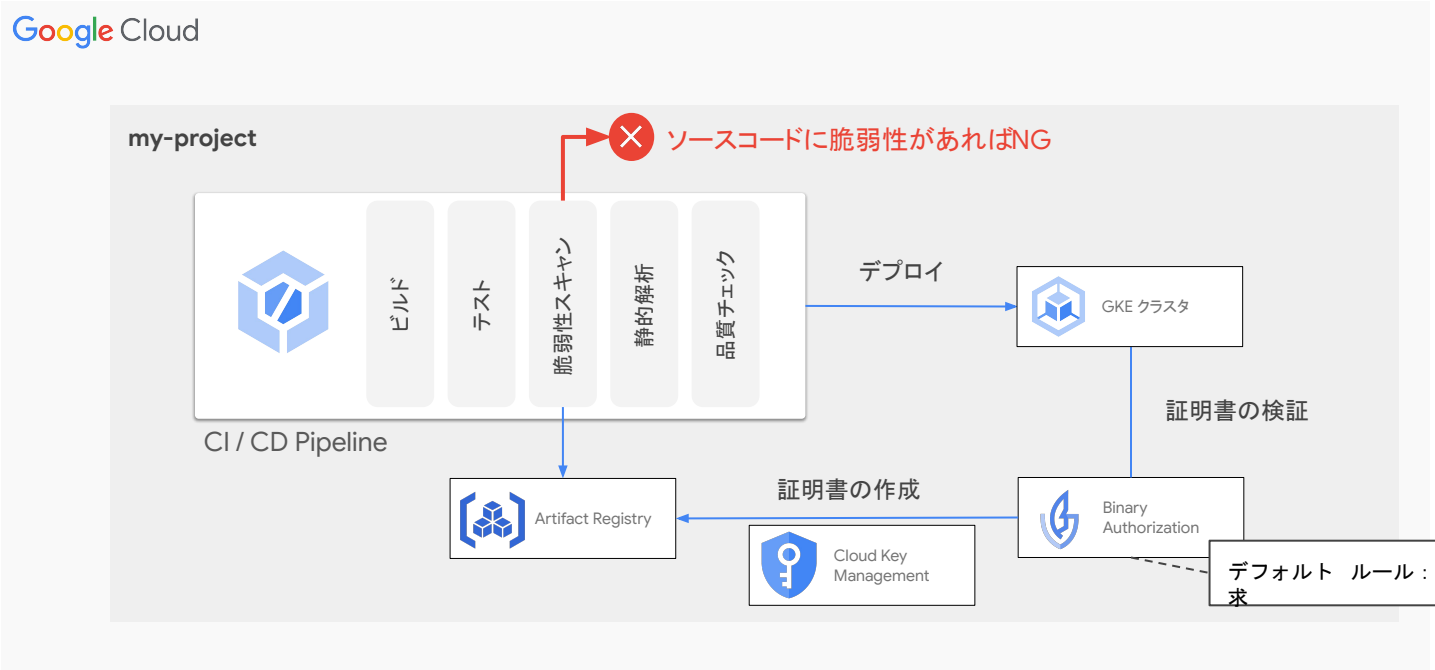
Binary Authorization

信頼できるコンテナ イメージのみ が GKE クラスタにデプロイされることを保証する

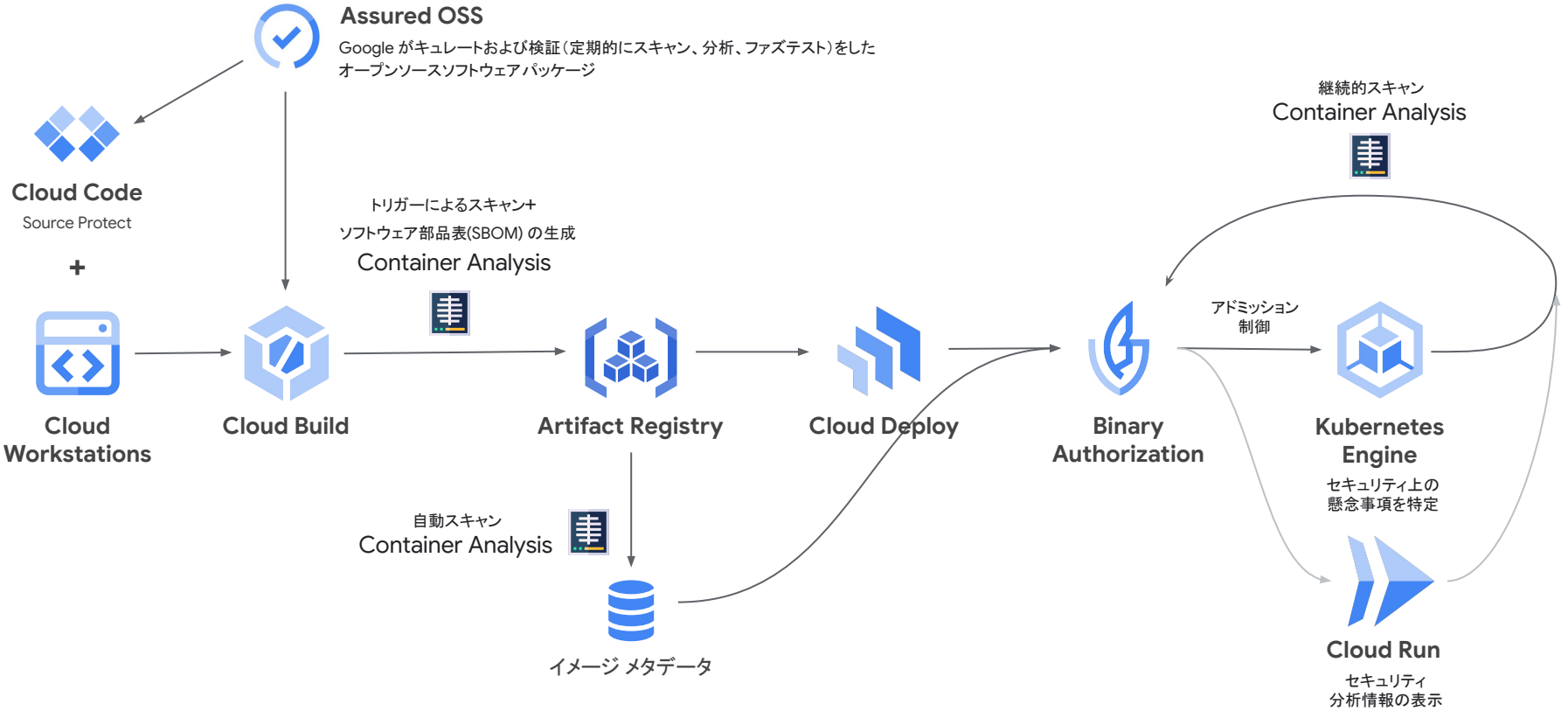
以下の条件に当てはまるコンテナ イメージのみデプロイを許可するよう制御が可能に

- イメージに対する認証者 (Attestors) による証明書 (Attestation) が存在する
- またはイメージ名が許可リストにマッチする

Binary Authorization の利用例
脆弱性スキャンが実行されたイメージのみデプロイを許可



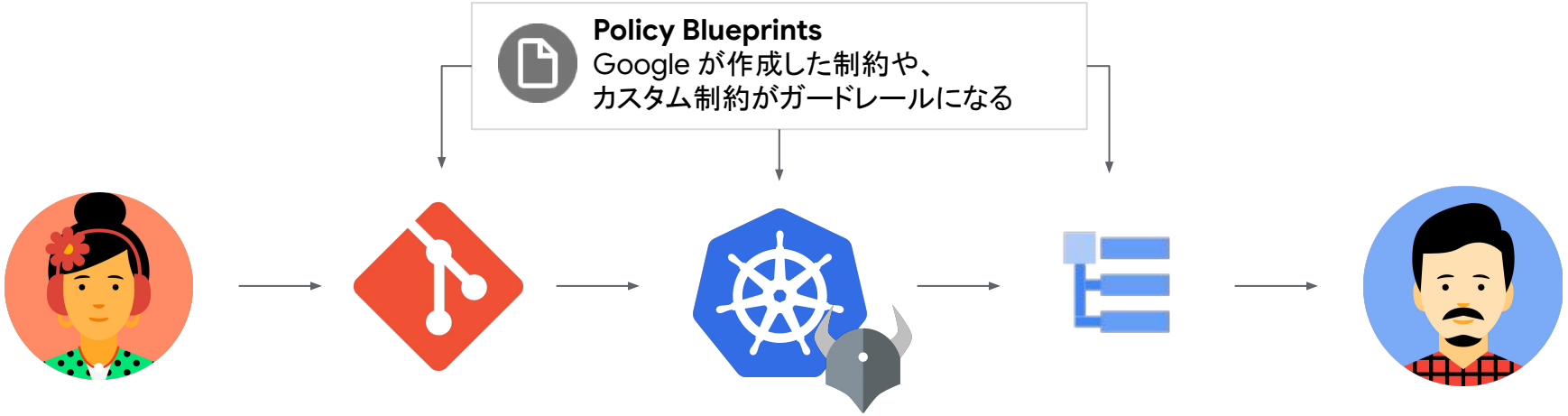
Software Delivery Shield を構成する Google Cloud のサービス群



Policy Controller

- マネージド版 Open Policy Agent (OPA) Gatekeeper
- Constraints を用いて**クラスタのコンプライアンスを強化**
- Kubernetes や Istio のベスト プラクティスに沿った**事前定義ポリシーを提供**
    - PodSecurityPolicy の代替としても活用可能
- 簡単なインストール / アップグレード
- Cloud Console や Cloud Monitoring との統合

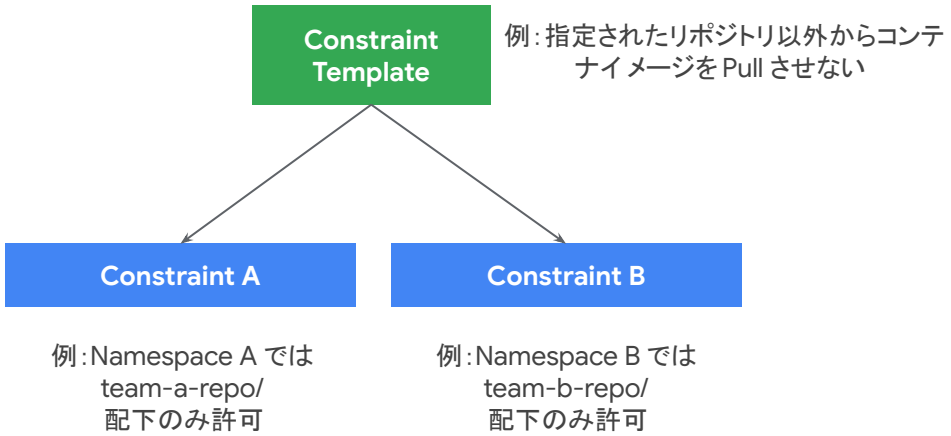
Policy Journey



Policy Controller 制約の構成要素

- Constraint Template (制約テンプレート)
  - 制約のスキーマとロジックを定義するリソース
  - ロジックは Rego で書かれる

- Constraint (制約)
  - 実際の制約を定義するリソース
  - 制約の対象となる API や Kind, Namespace 等を設定
  - 制約で利用するパラメータを設定



制約の構成要素

- Constraint Template (制約テンプレート)
  - 制約のスキーマとロジックを定義するリソース
  - ロジックは Rego で書かれる
- Constraint (制約)
  - 実際の制約を定義するリソース
  - 制約の対象となる API や Kind, Namespace 等を設定
  - 制約で利用するパラメータを設定

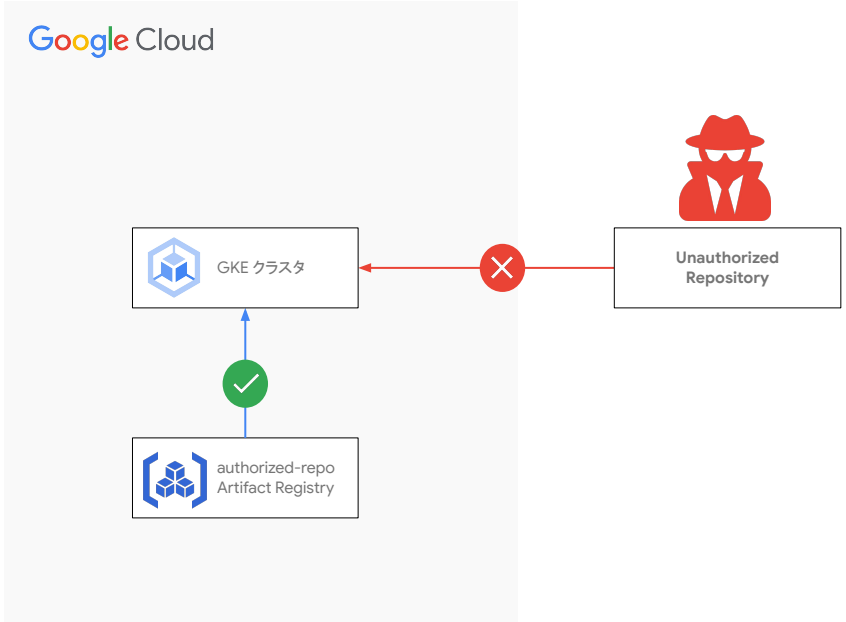
```
apiVersion: templates.gatekeeper.sh/v1beta1
kind: ConstraintTemplate
metadata:
  name: gcpstoragelocationconstraintv1
spec:
  crd:
    spec:
      names:
        kind: GCPStorageLocationConstraintV1
      validation:
        openAPIV3Schema:
          <スキーマの定義 >
  targets:
  - target: admission.k8s.gatekeeper.sh
    rego: |
      <ロジックの記述 >
```

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: GCPStorageLocationConstraintV1
metadata:
  name: tokyo-and-osaka-only
spec:
  enforcementAction: deny
  match:
    kinds:
    - apiGroups:
      - storage.cnrm.cloud.google.com
      kinds:
      - StorageBucket
    namespaces:
    - ns-japan
  parameters:
    <パラメータの定義 >
```

Policy Controller による制約の適用例

特定のコンテナ リポジトリ上のイメージのみデプロイを許可する（K8sAllowedRepos）

parameters で指定された文字列から始まるリポジトリ上のイメージのみデプロイを許可することで、不正なイメージのデプロイを防ぐ

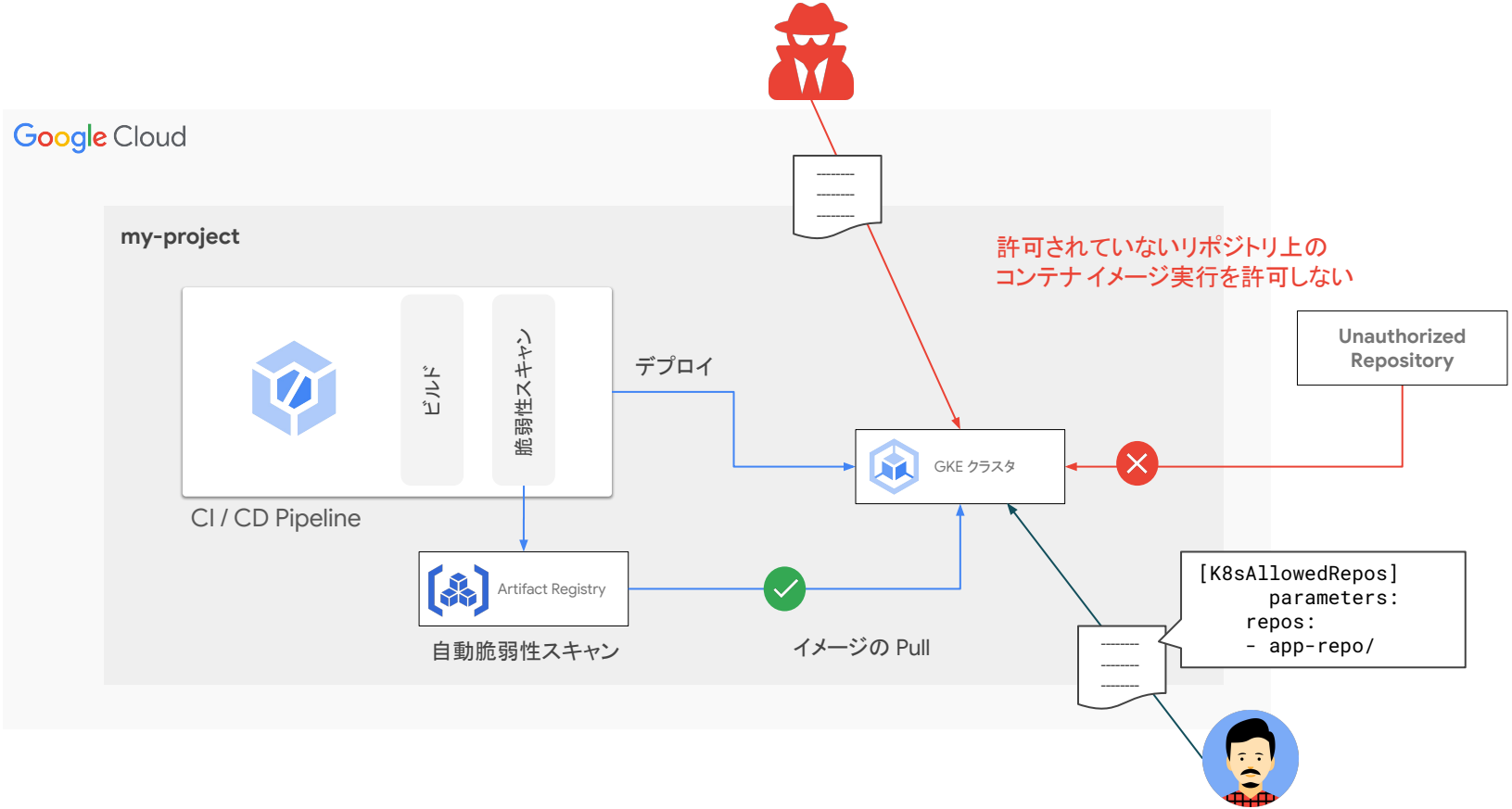


Lab-02 CI/CD セキュリティ ハンズオン

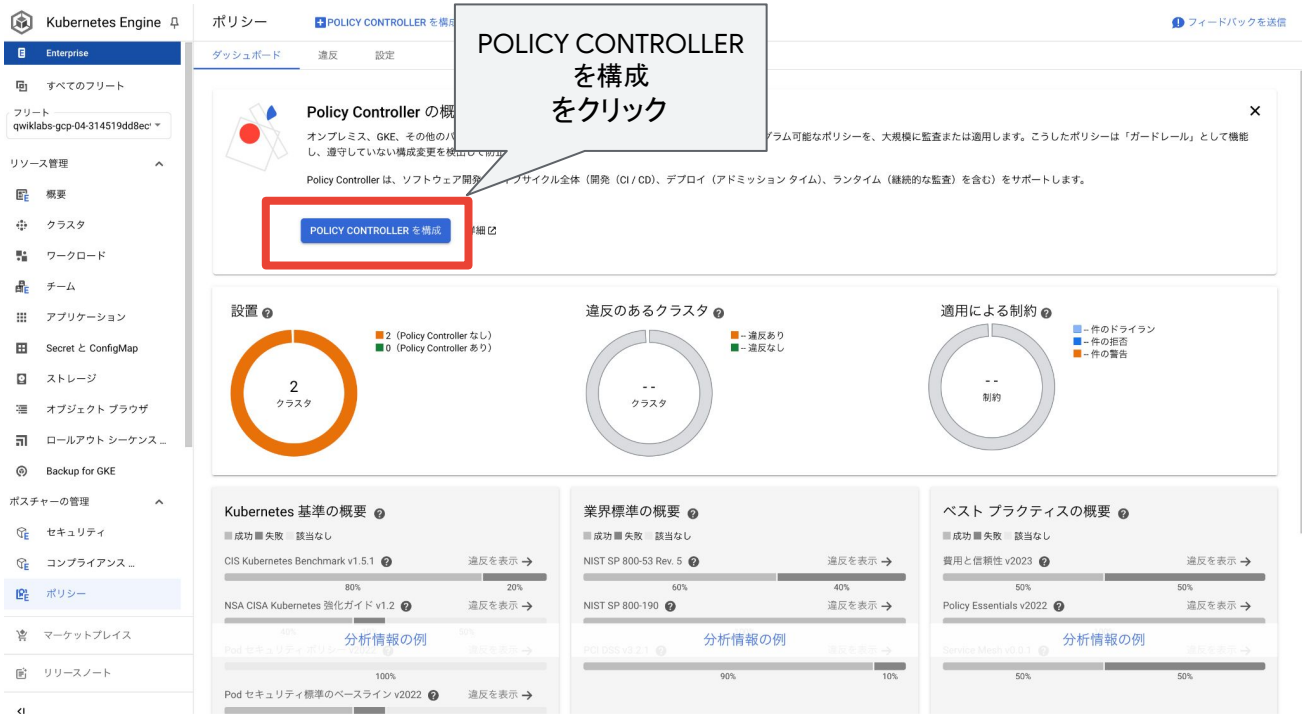
チュートリアル **Step 7/9** の「**Lab-02 CI/CD セキュリティ**」から再開してください

ハンズオン : 継続的脆弱性スキャン
GKE 上で実行されているイメージの脆弱性スキャン

ハンズオン : Policy Controller
許可されたリポジトリ上のイメージのみデプロイを許可



Lab-02-04 画面イメージ



Lab-02-04 画面イメージ

Lab-02-04 画面イメージ

Lab-02-04 画面イメージ

Lab-02-04 画面イメージ

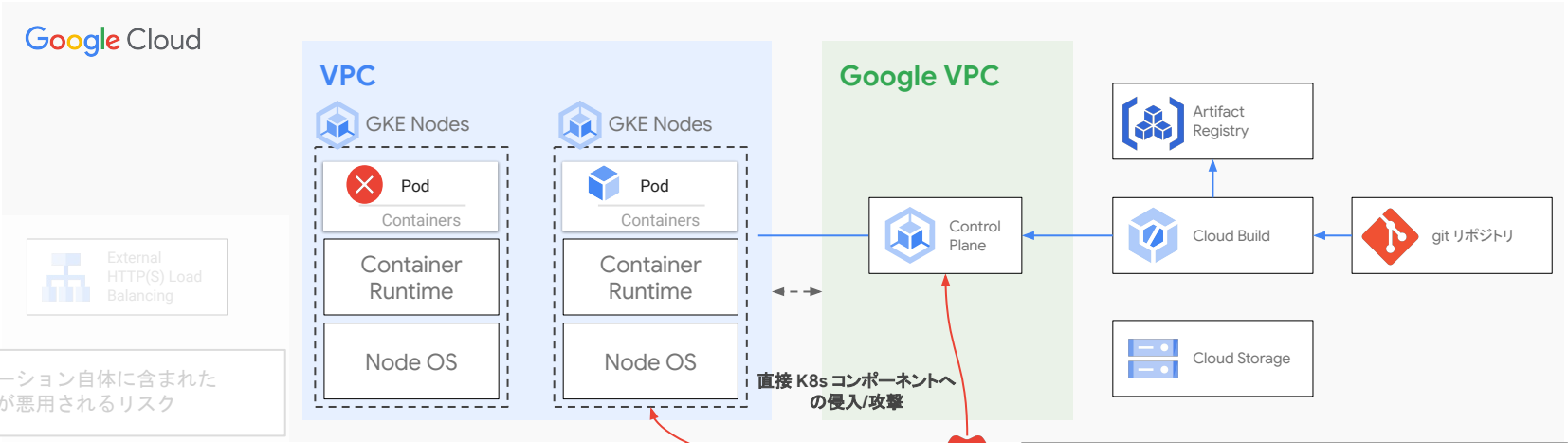
Lab-02-04 画面イメージ

GKE クラスタ コンポーネント
脆弱性への対策

Kubernetes 環境に潜む様々なリスク

信頼できないリポジトリを利用し、マルウェアや脆弱性が含まれるコンテナイメージをデプロイしてしまうリスク

Unauthorized Repository



External HTTP(S) Load Balancing

コンテナアプリケーション自体に含まれた脆弱性や設定ミスが悪用されるリスク

脆弱なコンテナを入り口にノードや別コンテナへ侵入されたり、DB/ストレージにアクセスされ被害が拡大してしまうリスク

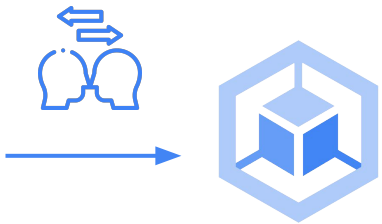
Control Plane や Node 等 Kubernetes コンポーネントの脆弱性や設定ミスが悪用されるリスク

などなど…

GKE クラスタ コンポーネントへの攻撃リスクを低減するために

GKE クラスタ コンポーネントへのアクセスを制限することにより **攻撃の機会を減らす** ことができる

また、アップデートの自動適用による脆弱性への迅速な対処や、サービス アカウント キーなど機密情報を外部に保管しない運用を実践することも効果的



- **アクセス経路の制御**
  - コントロール プレーンやノードへの不要なアクセス経路を閉じる
- **アップデートの自動化**
  - アップグレードやセキュリティパッチ適用の自動化
- **Google Cloud サービスアカウントの保護**
  - サービス アカウント キーを利用しない運用の実践

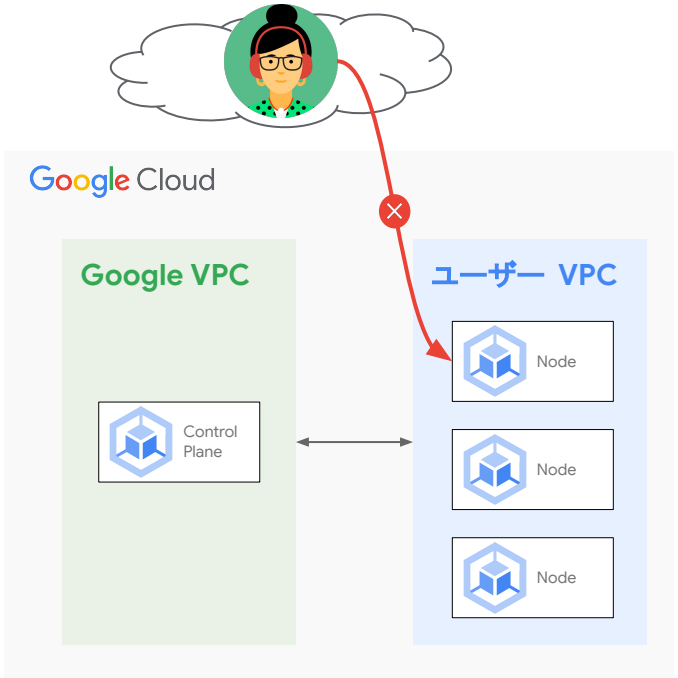
限定公開クラスタ(プライベート　クラスタ)

Node に Public IP が付与されていない クラスタ

外部ネットワークから Node に対する**直接アクセスのリスクを低減**

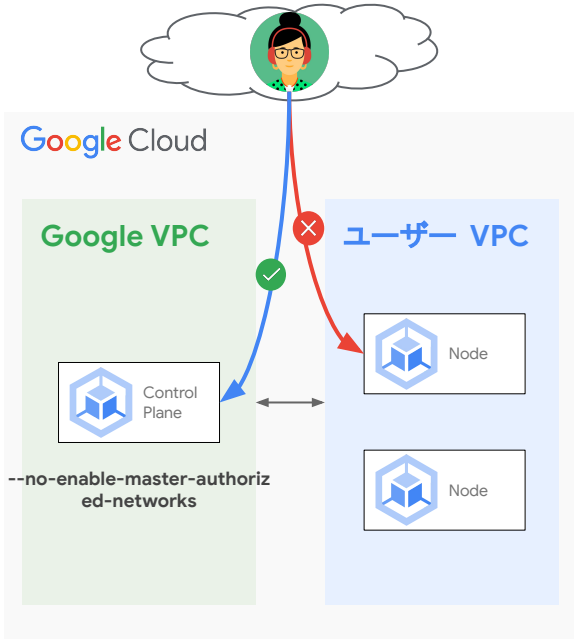
限定公開クラスタはコントロール プレーンの公開設定により、さらに以下3パターンに分類可能

1. パブリック エンドポイント有効
2. パブリック エンドポイント有効 + 承認済みネットワーク
3. パブリック エンドポイント無効

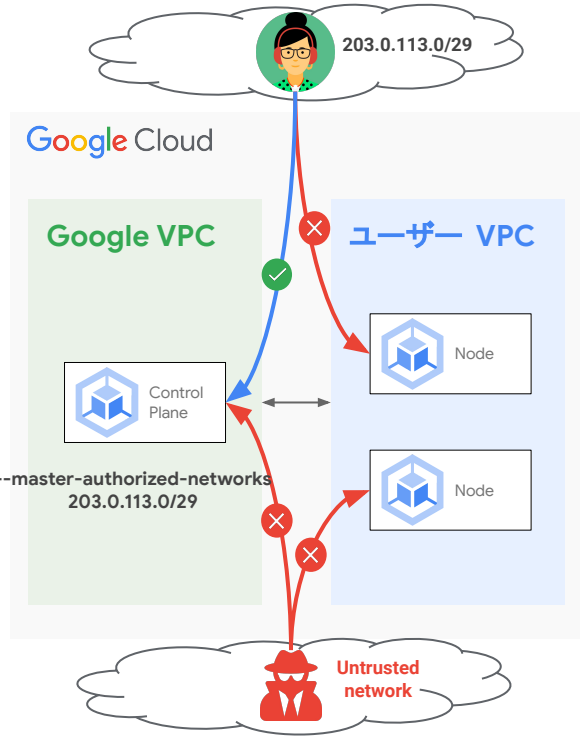


限定公開クラスタ(プライベート　クラスタ)の種類

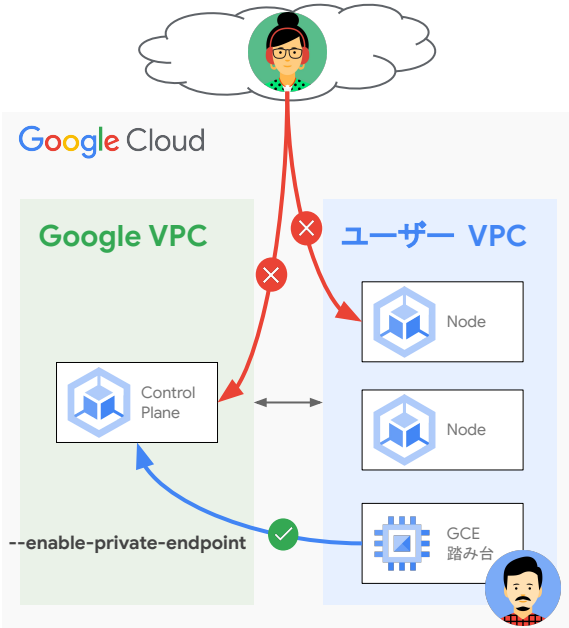
①パブリック　エンドポイント有効



②パブリック　エンドポイント有効＋承認済みネットワーク



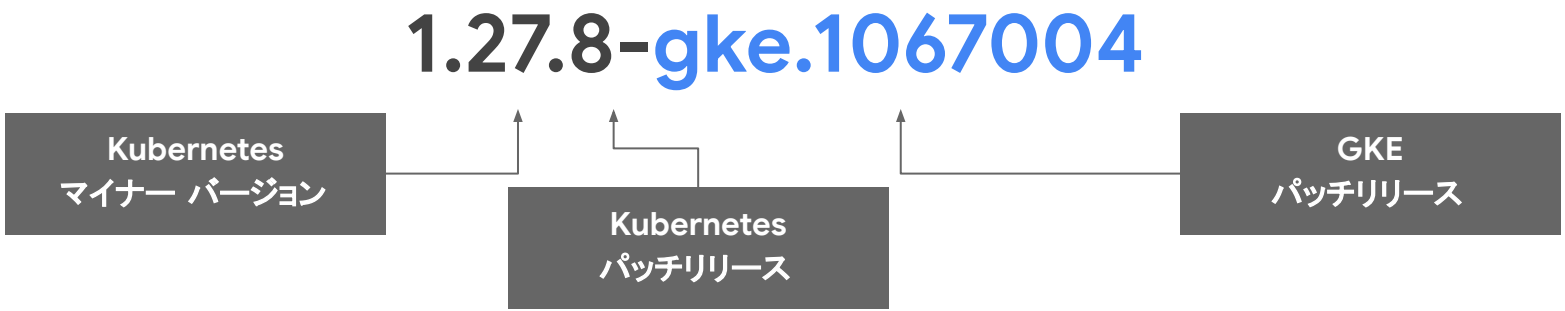
③パブリック　エンドポイント無効



クラスタのアップグレード

Kubernetes / GKE では脆弱性の対応や既知問題の修正、新機能の導入のためにアップデートを頻繁にリリースしている
特にセキュリティの観点では、**いかに迅速に脆弱性に対応できるか**　が重要となっており、継続的なアップグレード戦略を検討する必要がある

GKE ではコントロール プレーンを Google が管理しており、**自動的にパッチ適用・アップグレード**　される（無効化不可）がノードについては **自動** もしくは **手動** でアップグレードを行う必要がある



バージョン指定方法

1. 静的にバージョンを指定

- 特定のパッチ バージョンまで指定したい
- バージョンを指定しない場合は GKE の デフォルト バージョンが選択される
- デフォルト バージョンは安定性と パフォーマンスに基づき 定期的に変更される

```
$ gcloud container clusters \
    create <CLUSTER_NAME> \
    --cluster-version <VERSION>
```

2. リリース チャンネルを利用

推 奨

- Google が推奨するバージョンで クラスタが作成される
- コントロール プレーンとノードの アップグレードは Google が行う

```
$ gcloud container clusters \
    create <CLUSTER_NAME> \
    --release-channel <CHANNEL>
```

リリース チャンネル

バージョニングとアップグレードを行う際のベスト プラクティスを提供する仕組み

リリース チャンネルに新しいクラスタを登録すると、**Google により**

Control Plane と Node のバージョンとアップグレード サイクルが自動的に管理される

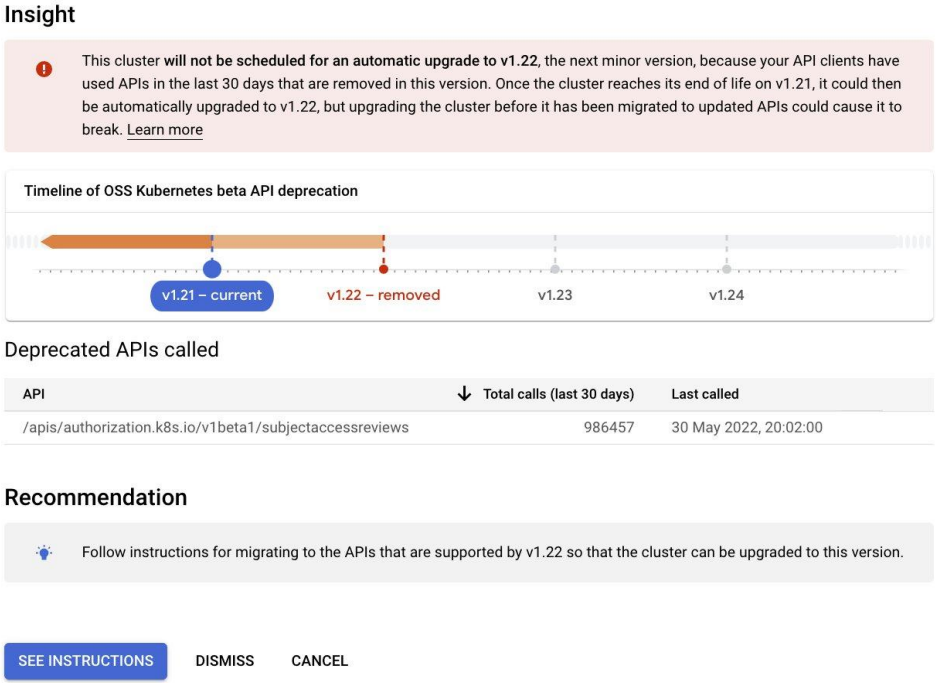
利用できる機能と更新頻度の異なる、以下 3 つのチャンネルがある

- **Rapid** ... 最新のバージョンが利用可能。検証目的での利用を推奨 (SLA 対象外)
- **Regular** ... 機能の可用性とリリースの安定性のバランス
- **Stable** ... 新機能よりも安定性を優先する場合

非推奨 API / 構成の自動検出

今後のマイナー バージョンで削除される Kubernetes API や構成がクラスタで使用されていることを自動的に検出し通知する機能

削除予定の API / 構成の利用が検出されると、GKE の自動アップグレードが一時停止されるため、**自動アップグレードによる互換性の問題を回避**可能（手動でのアップグレードは可能）



メンテナンスの時間枠と除外

- **メンテナンスの時間枠**
  自動メンテナンスを **許可する** 時間枠
  - 32 日周期で最低 48 時間必要
  - 各時間枠は 4 時間以上連続した時間
- **メンテナンスの除外**
  自動メンテナンスを **禁止する** 時間枠
  - 詳細は後続資料で説明

使い方の例:

- 週末を避ける
- 大型セールやイベント期間中を避ける
- 一時的にアップグレードを延期する

メンテナンスの除外設定

メンテナンスの除外設定により自動アップグレードを禁止する期間をコントロール

Scope	Control plane			Node pools		
	Minor upgrade	Patch upgrade	VM disruption due to GKE maintenance	Minor upgrade	Patch upgrade	VM disruption due to GKE maintenance
No upgrades (default)	No	No	No	No	No	No
No minor upgrades	No	Yes	Yes	No	Yes	Yes
No minor or node upgrades	No	Yes	Yes	No	No	No

- 完全にアップグレードを止める場合 (No upgrades) はメンテナンス除外期間を **最大 30 日間** *1まで設定可能
- Control Plane や Node のマイナー バージョン アップグレードを止める *2場合 (No minor upgrades / No minor or node upgrades) はメンテナンス除外期間を **最大 180 日間** *1まで設定可能

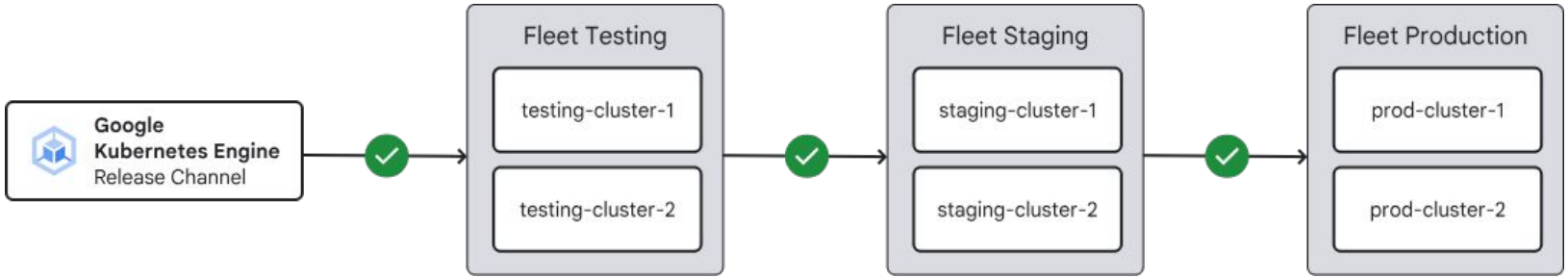
***1 マイナー バージョンの EOL を超過することはできない**

***2 リリース チャンネルを利用しているクラスタが対象**

Rollout Sequencing

クラスタ間のアップグレード順序を制御する機能

アップグレードするクラスタを環境毎にグルーピングし依存関係を持たせることで
段階的な自動アップグレードをすることが可能に（例：開発環境 →ステージング環境 →本番環境）



クラスタのアップグレード / 脆弱性情報の自動通知

任意の Pub/Sub トピックに対し、
アップグレードや脆弱性に関する通知メッセージを送信

- **SecurityBulletinEvent**

  クラスタに影響のある脆弱性情報を通知

- **UpgradeAvailableEvent**

  新バージョンが利用可能になった時に通知

  マイナー バージョン: 2 - 4 週間前
  パッチ バージョン: 1 週間前

- **UpgradeEvent**

  アップグレードが開始されると通知（自動、手動問わず）

https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-notifications#notification-types

Google Cloud サービス アカウントの保護

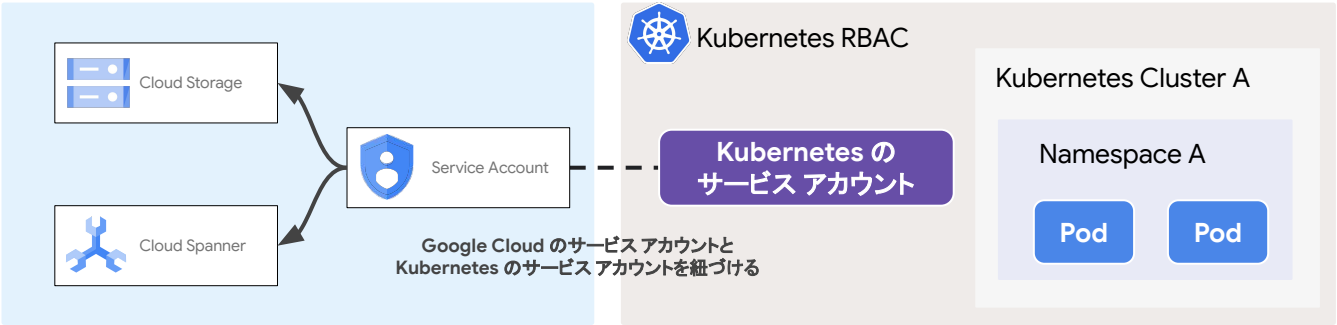
サービス アカウント キーの漏洩は GKE に限らず、Google Cloud リソース全体への不正アクセスに繋がる可能性があるため、不用意にサービス アカウント キーが利用されないようにする

- **組織ポリシーによるサービスアカウントキー利用の制限**
  - *iam.disableServiceAccountKeyCreation* や *iam.disableServiceAccountKeyUpload* 等の組織ポリシーを有効にし、サービス アカウント キーの利用を原則禁止とする
- **Workload Identity Federation の活用**
  - Google Cloud 外や GKE クラスタ内からのサービス アカウント利用時に Workload Identity Federation を利用し、サービス アカウント キーのローカル保存を不要にする
- **VPC Service Controls の活用**
  - サービス アカウント キーが漏洩してしまった場合も、特定の境界外からの Google API アクセスを防止する

Workload Identity Federation for GKE

Google Cloud のサービス アカウントと Kubernetes のサービス アカウントを紐づけ、サービス アカウント キー無しで Pod から Google Cloud サービスへのアクセスを実現

サービス アカウント キーをクラスタ内で保管する必要がなくなる ため、セキュリティの観点から利用を推奨

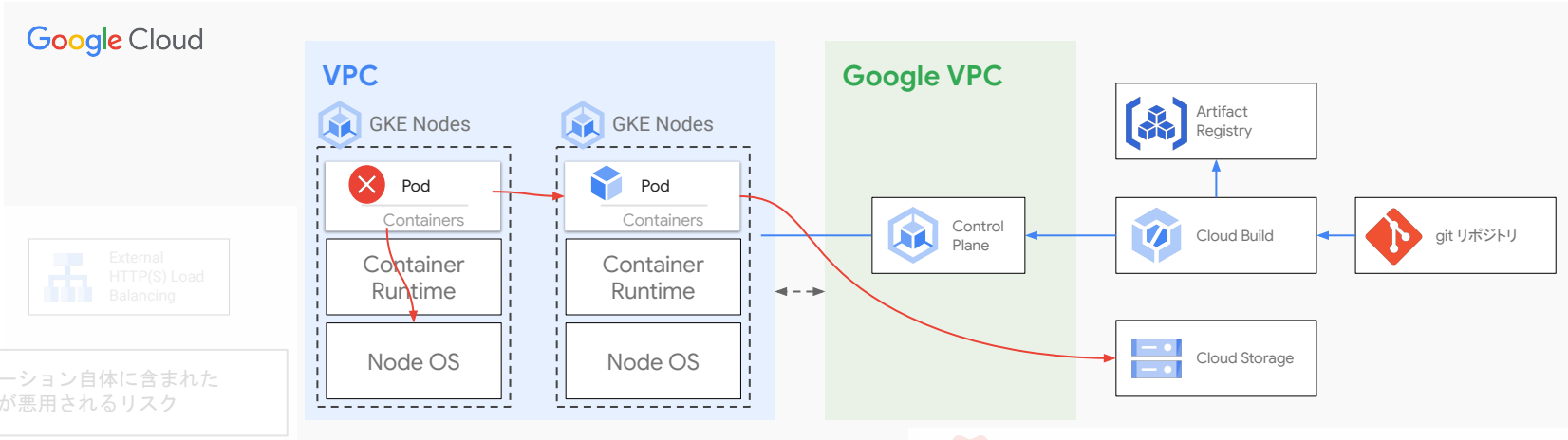


攻撃の影響範囲を小さくする ための対策

Kubernetes 環境に潜む様々なリスク

信頼できないリポジトリを利用し、マルウェアや脆弱性が含まれるコンテナイメージをデプロイしてしまうリスク

Unauthorized Repository



External HTTP(S) Load Balancing

コンテナアプリケーション自体に含まれた脆弱性や設定ミスが悪用されるリスク

脆弱なコンテナを入り口にノードや別コンテナへ侵入されたり、DB/ストレージにアクセスされ被害が拡大してしまうリスク

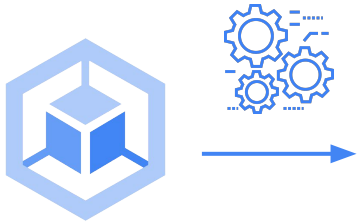
Control Plane や Node 等 Kubernetes コンポーネントの脆弱性や設定ミスが悪用されるリスク

などなど...

GKE クラスタのアクセス制御

コンテナの脆弱性等により侵入を許してしまった場合も、適切な権限設定や環境分離を行うことで影響範囲を小さくすることが可能

また、ノードレベルでのセキュリティ対策や脆弱な Kubernetes 設定を避けることでコンテナ エスケープのリスクを低減できる



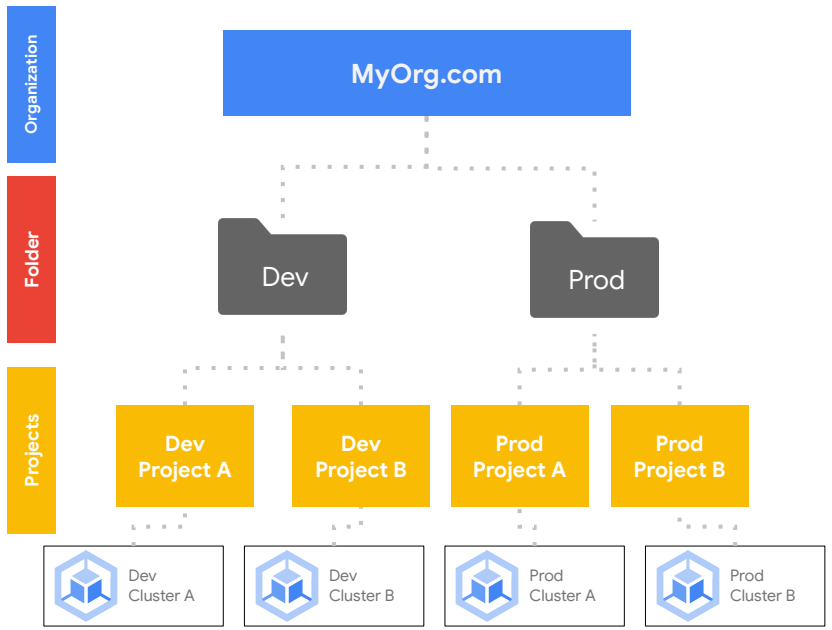
- **適切な権限制御**
  - 環境の分離、最小権限の付与
  - グループ単位での IAM や RBAC の設定
- **脆弱な Kubernetes 設定の検出 /防止**
  - 構成スキャン、ポリシーの構成
- **ノードレベルでのセキュリティ強化**
  - ホストOS、コンテナ ランタイム レベルでのセキュリティ強化

IAM による適切な認可の設定

Google Cloud では **IAM** の仕組みで各リソースに対する認可を制御

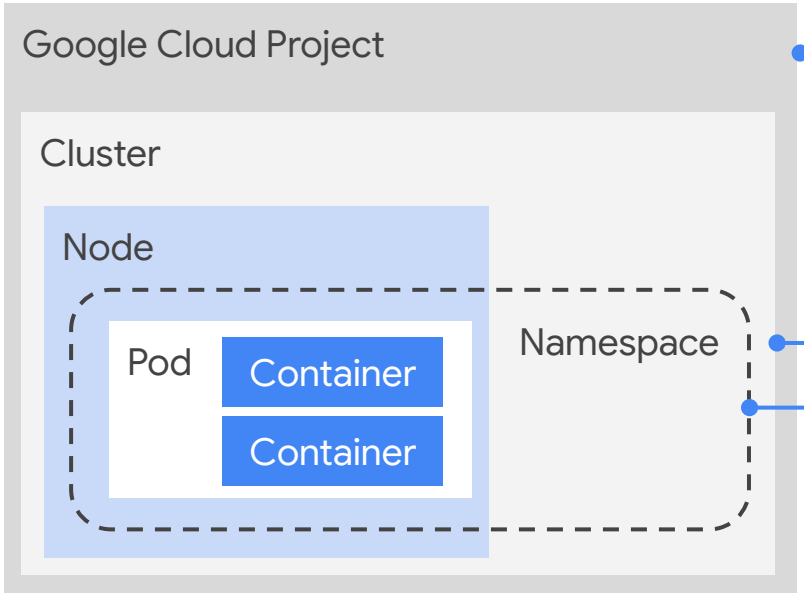
また、**フォルダ** 機能を活用することにより環境やチーム単位でプロジェクトをまとめることができる

- 権限分離やインシデント発生時の影響軽減の観点から、少なくとも**環境ごとにフォルダ・プロジェクトは分離** する
- 最小権限の原則に則り、**必要最低限の権限** を付与する（基本ロールではなく、事前定義ロールでの付与を検討する）
- 権限管理をシンプルにするために、個々のユーザーではなく**グループに対してロールを付与** する



本番環境での Google Kubernetes Engine の準備
https://cloud.google.com/architecture/prep-kubernetes-engine-for-prod

IAM と Kubernetes RBAC



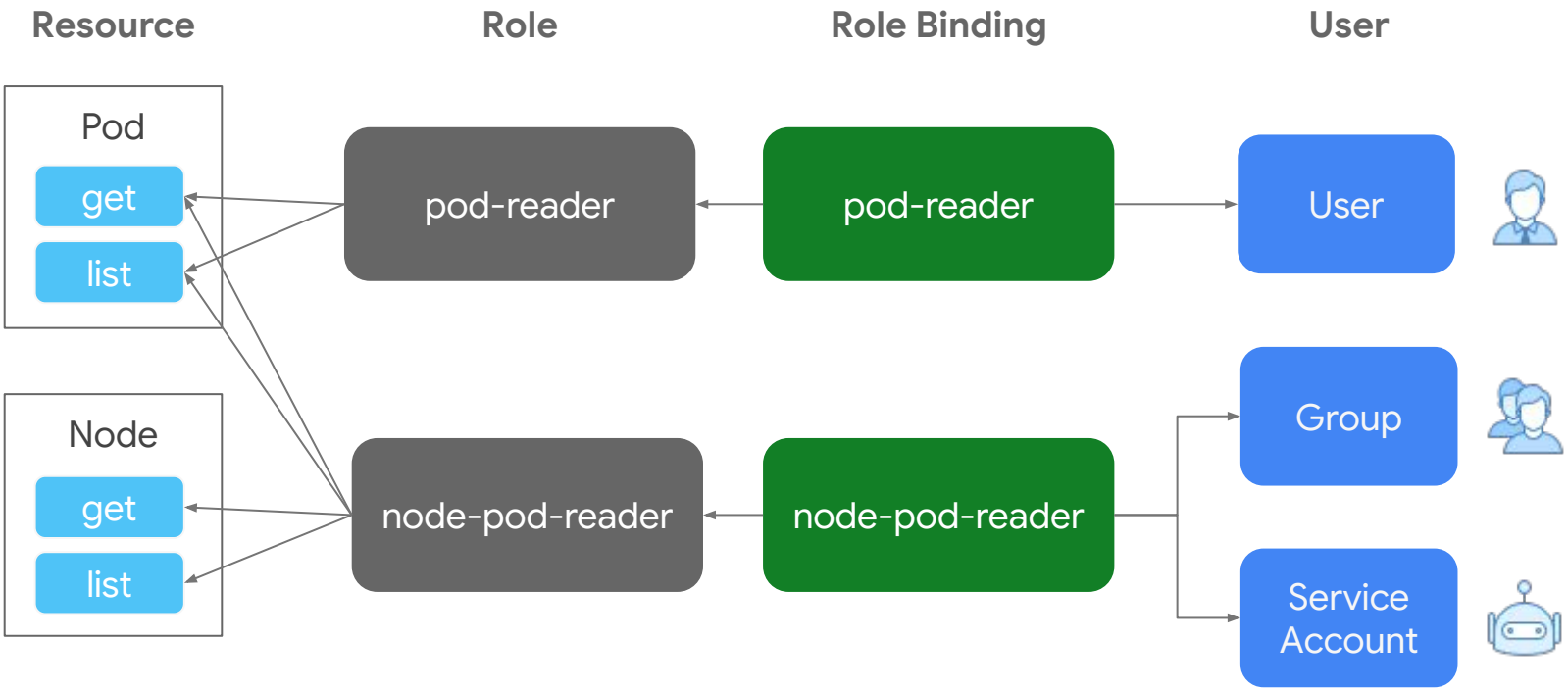
IAM: プロジェクトレベル

- クラスタ管理者: GKE クラスタの管理
- コンテナ開発者: クラスタ内の API アクセス

RBAC: Kubernetes クラスタ / Namespace レベル

クラスタや Namespace 内での個々の許可設定

Kubernetes RBAC



Kubernetes RBAC

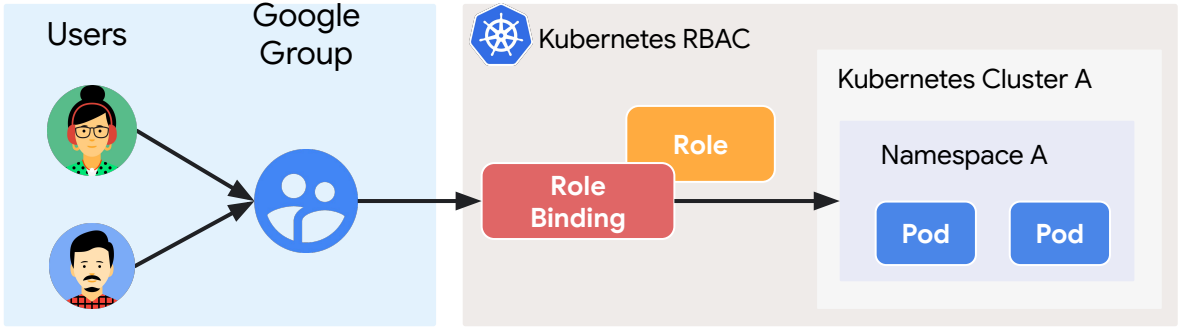
```
$ cat blue-binding.yaml

kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: blue-dev-binding
  namespace: blue
subjects:
  - kind: User
    name: blue-team-dev@kube-pw.iam.gsa.com
roleRef:
  kind: ClusterRole
  name: admin
  apiGroup: rbac.authorization.k8s.io
```

Google Groups for RBAC

Google グループに対する RBAC の設定をサポート

グループに対して権限を割り当てることにより、ユーザーアカウント個別の権限管理が不要に

グループに対して**必要最小限の権限を付与**することにより、認証情報の流出や悪用による影響の極小化に繋がる

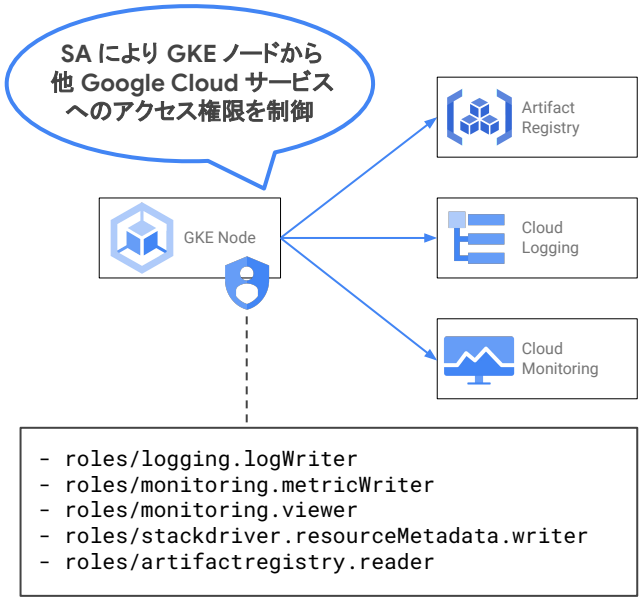


GKE ノードのサービス アカウントによる権限制御

GKE ではノードプール単位でサービス アカウントを設定可能

GKE ノードのデフォルト サービス アカウント (Compute Engine のデフォルト サービス アカウント) は強い権限を持っているため、**最小権限が付与されたサービス アカウントを作成・設定** することを推奨

アプリケーション Pod から各 Google Cloud へのサービスへのアクセス制御は先述の Workload Identity Federation for GKE の利用を推奨



Network Policy によるネットワーク レベルでの制御

- Network Policy により、GKE クラスタ内外の通信をネットワーク レベルで制御
- Namespace 間でのトラフィック制御し、意図しないもしくは悪意のある通信を遮断することも
- GKE Enteprise では Namespace 単位でなく、GKE Cluster 全体に適用する Cluster-wide Network Policy もサポート

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all
  namespace: tenant-a
spec:
  podSelector:
    matchLabels:

  ingress:
  - from:
    - podSelector: {}
```

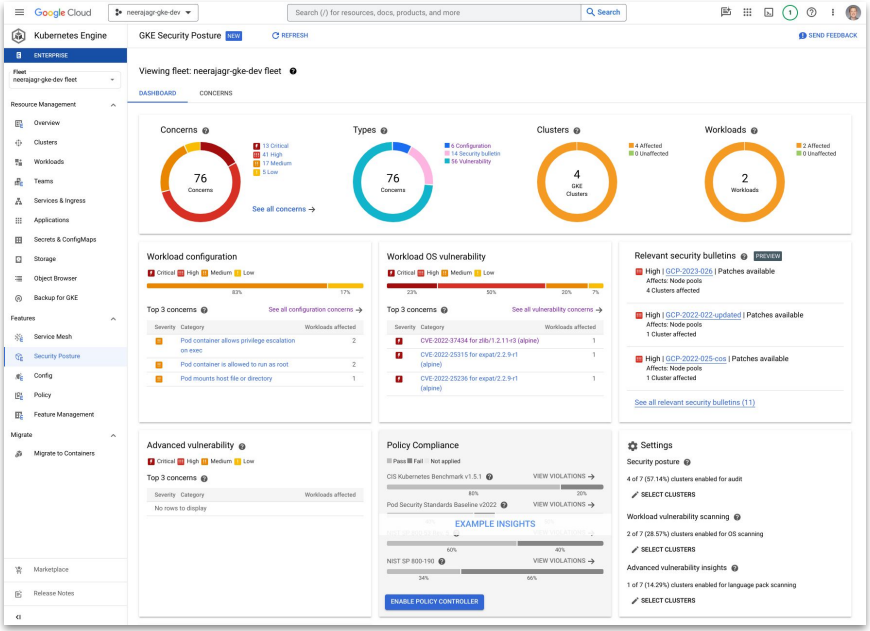
ワークロード構成スキャン

ワークロードの構成を自動的にスキャンし、**Kubernetes のベストプラクティスに沿っていない、潜在的なリスクがあるワークロードを報告**

発見された懸念事項は Cloud Console ダッシュボードや Cloud Logging のログエントリーから確認可能

検知するリスクの例:

- ホストのネームスペースの共有
- 特権コンテナの利用
- runAsNonRoot が有効になっていない
- 権限昇格が可能になっている



Policy Controller のデフォルト ポリシー ライブラリ

- 65+ 制約テンプレート ライブラリが Policy Controller のデフォルトでインストール
- ライブラリは Policy Controller チームにより**継続的にメンテナンス** され、拡張される

Pod による特権付きコンテナの実行を防ぐ

クライアント / サービス間での相互 TLS 通信を強制

きめ細かいサービス認可管理がなされていることを要求

コスト管理のために必要なタグが必須とする制限

認証を不要とするサービスへのアクセス許可を認めない

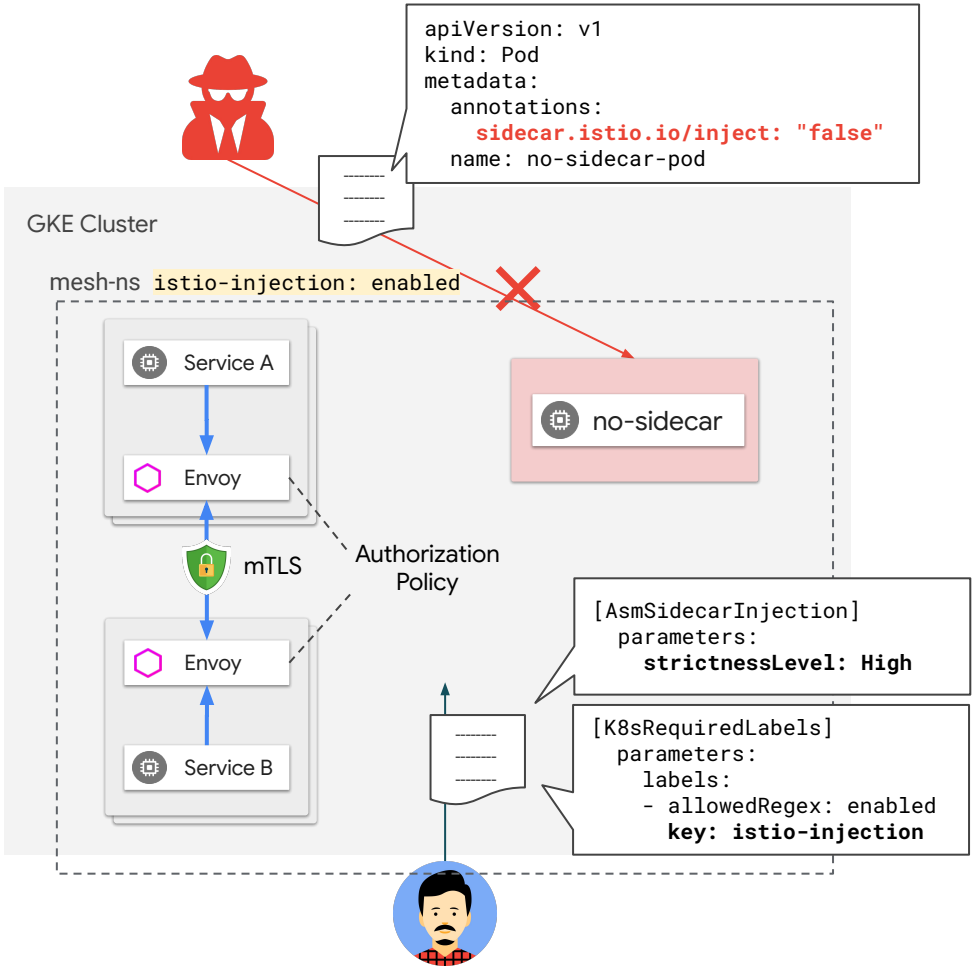
クラスタやサービスへのアクセスログ有効化を必須に

制約テンプレートライブラリ | Anthos Config Management
https://cloud.google.com/anthos-config-management/docs/reference/constraint-template-library

Policy Controller による制約の適用例

サイドカーが injection されないことにより、 Istio / ASM のセキュリティ ポリシーがバイパスされることを防ぐ

- **K8sRequiredLabels**
  - 特定の Namespace にサイドカー injection ラベルを設定する
- **AsmSidecarInjection**
  - メッシュ内の Pod が Istio プロキシ サイドカーの挿入をバイパスすることを禁止する



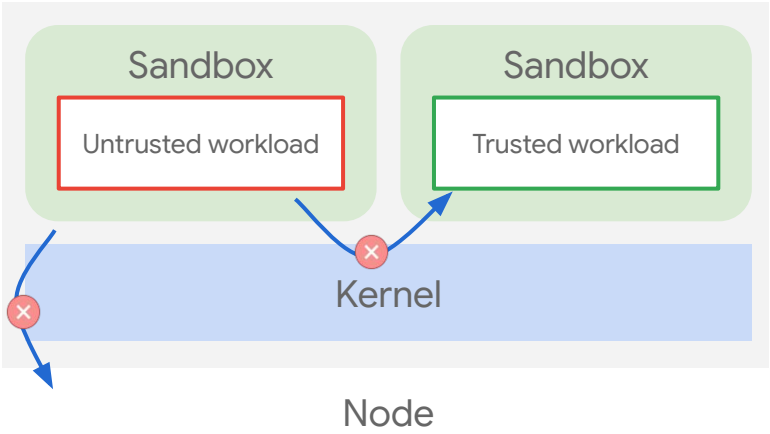
GKE Sandbox

gVisor (軽量な user-space kernel) による強い Isolation により

信頼できないワークロードからホストカーネルを保護

Container からの syscall をインターセプトし、
ホストカーネルへの直接アクセスを制限する

ホストアクセスを強く制限することにより
ホストへのエスケープ リスクを低減し、
他コンテナ/テナント環境での影響を極小化



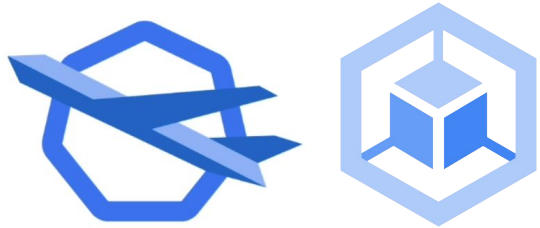
Container-Optimized OS (COS)

コンテナ実行に最適化されたセキュア　/ 軽量な OS

- 高速な起動
- 最低限のプロセスのみ動作
- 読み取り専用のルート ファイル システム
- セキュリティが強化されたカーネル
- サービス Listen ポートの最小化 , etc.

COS を利用することにより、**アタック サーフェスを小さくする** ことが可能に

GKE Autopilot



Secure by default

デフォルトで各種セキュリティ機能を適用

Locked down Nodes

ノードに対するアクセスを制限し、セキュリティに関するコストを削減

No infrastructure security tasks

自動的なセキュリティパッチ適用・バージョン アップグレード

Autopilot - Node レベルのセキュリティ対策

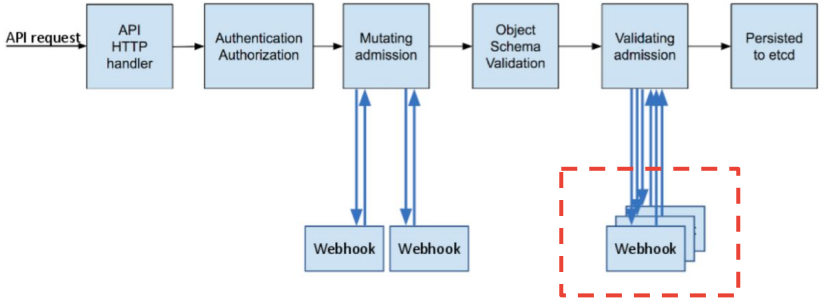
Shielded Nodes	攻撃者がノードのブートストラップ認証情報を抜き取った場合等でも、 攻撃者によるノードになりすましから保護する
Container Optimized OS	コンテナ実行に最適化されたセキュア / 軽量な OS COS を利用することにより、アタックサーフェスを小さくすることが可能に
Node への SSH	Node への SSH 不可
アップグレード	自動アップグレードにより、運用負荷を下げつつ迅速に脆弱性に対応
Node の変更	kubectl 等による Node リソースの変更不可

Autopilot - Pod レベルのセキュリティ対策

特権コンテナ	特権コンテナの実行を阻止し、攻撃 /侵入による Node へのアクセスリスクを低減
Linux Capability	利用可能な Linux Capability を制限 (NET_ADMIN 等、一部例外的に利用可能にできる)
Host Options	hostNetwork の利用不可、hostPath は /var/log/ 配下のみ許可
Security Profiles	全ての Pod に RuntimeDefault の seccomp プロファイルが適用される
Mutating Admission Webhook	kube-system 等の管理された Namespace 内リソースや Node, PV などに対する Mutating Webhook による変更を禁止する

GKE Warden

GKE Autopilot の各種制約を適用するための Admission Webhook



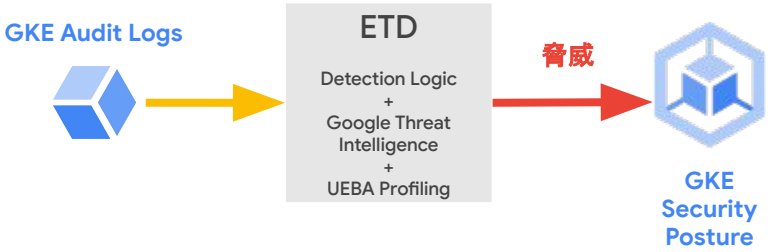

```
$ linkerd install | kubectl apply -f -
Error from server (GKE Warden constraints violations): error when creating "STDIN":
admission webhook "warden-validating.common-webhooks.networking.gke.io" denied the request:
GKE Warden rejected the request because it violates one or more constraints.
Violations details: {"[denied by autogke-default-linux-capabilities]":["linux capability
'NET_ADMIN' on container 'linkerd-init' not allowed; Autopilot only allows the capabilities:
'AUDIT_WRITE,CHOWN,DAC_OVERRIDE,FOWNER,FSETID,KILL,MKNOD,NET_BIND_SERVICE,NET_RAW,SETFCAP,SE
TGID,SETPCAP,SETUID,SYS_CHROOT,SYS_PTRACE'."]}
Requested by user: '<>@gmail.com', groups: 'system:authenticated'.
```

異常な挙動を検知 する

GKE Threat Detection Public Preview

GKE Audit Log を基に、GKE クラスタ上の脅威を検出

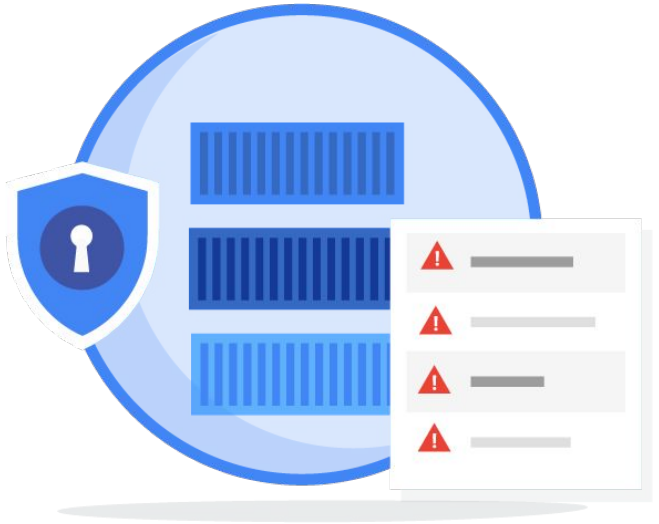
- 複数クラスタ環境に継続的にスキャンを実行し
  アクティブな脅威を迅速に検出
  - RBAC の変更や特権コンテナのデプロイ等のイベントを検出
- 検出された脅威は MITRE ATT&CK フレームワークにより分類される
- Security Command Center の Premium Tier を
  利用している場合は、SCC でも脅威情報を確認可能



Security Command Center: Container Threat Detection

実行中システムへの脅威を検出

- よくある脅威を広く検知
- 頻出の攻撃に対しては検出器をプリセット
  - Added Binary Executed
  - New Library Linked
  - Reverse Shell ..
- Google Cloud サービスとの統合
  - セキュリティコマンドセンターに結果を表示
  - 各種イベントはOperations へ

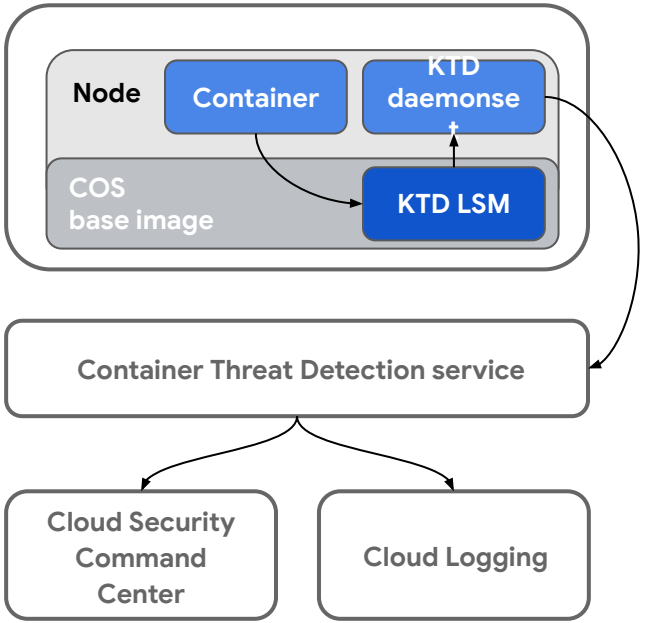


https://cloud.google.com/security-command-center/docs/concepts-container-threat-detection-overview?hl=ja

Security Command Center: Container Threat Detection

Container Threat Detection は複数の検出機能 / 分析ツール および API で構成

- 動作情報（COSのカネールモジュールに関する変更）を収集するためには GKE クラスタのノードには最新の COS が必要
- データとコンテナを識別し、DaemonSet を経由してデータが送られる
- カーネルと Detector Service のすべてのデータはエフェメラルで永続化されない
- 検出されたものは Security Command Center と Cloud Logging へ自動的に書き込まれる



*LSM: LInux Security Module: Linux Kernel への変更を検知
*KTD: Container Threat Detection

セキュリティ ポリシー ハンズオン

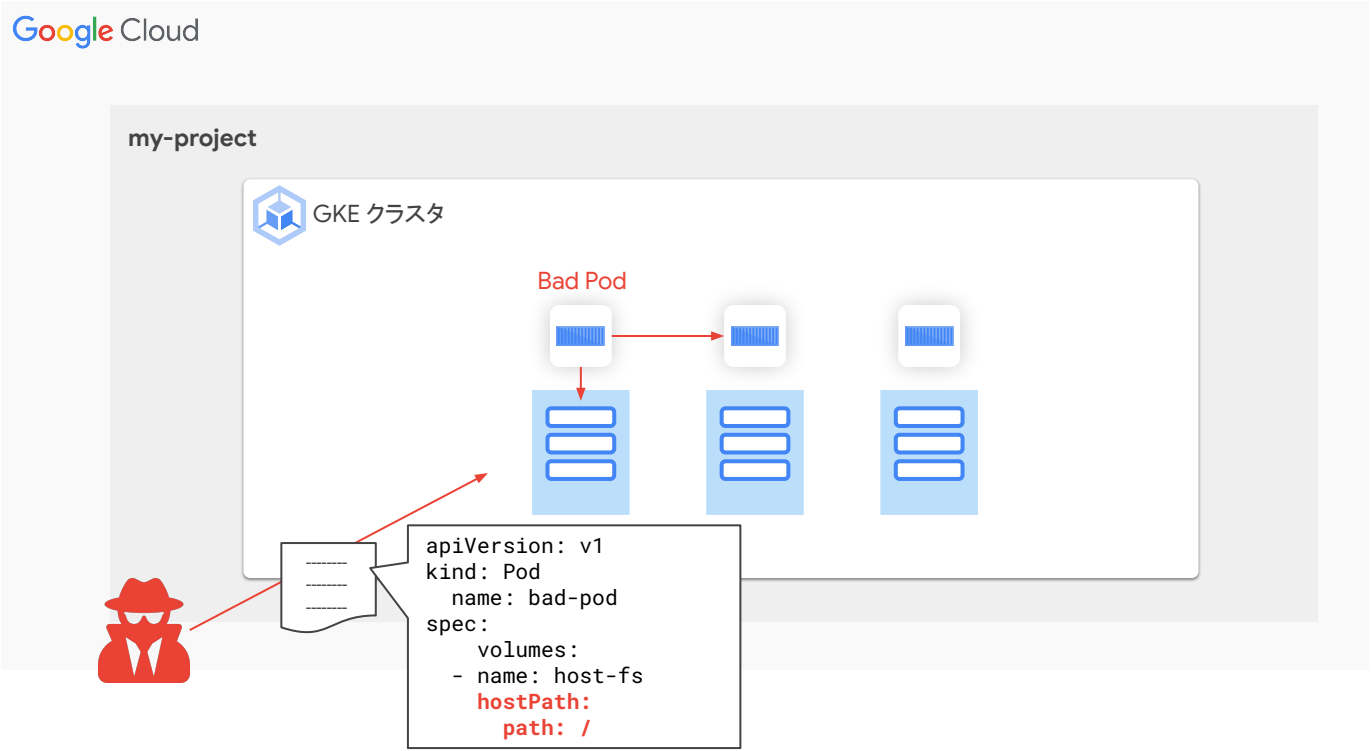
セキュリティ ポリシーのハンズオンでは、以下セキュリティ機能を体験します

- 不適切な設定の Pod の検知
- Policy Controller による不適切な設定のコンテナ デプロイの防止

チュートリアル **Step 8/9** の「**Lab-03 セキュリティ ポリシー**」から再開してください

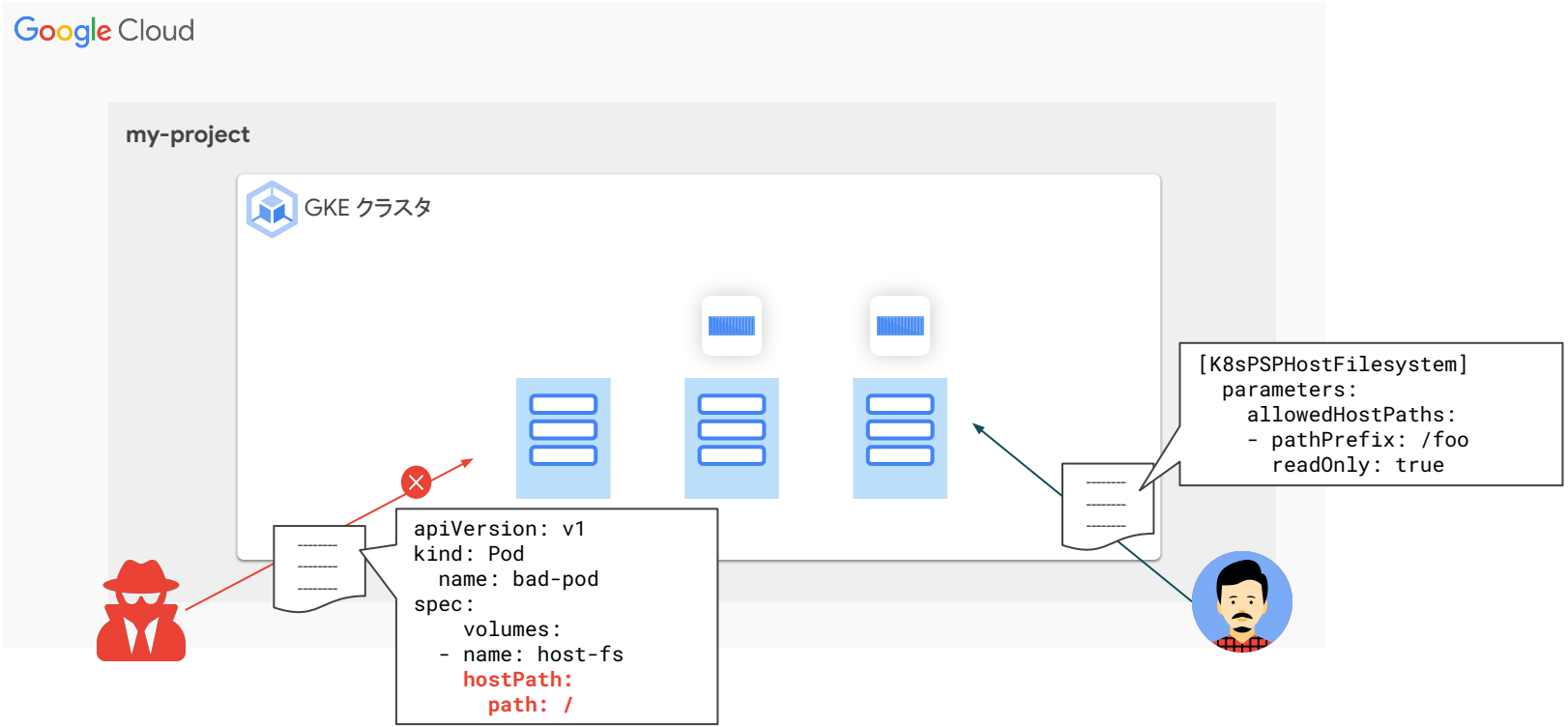
ハンズオン : 不適切な設定の Pod の検知

不適切な設定の Pod をデプロイし、GKE Security Posture 機能で検知できることを確認



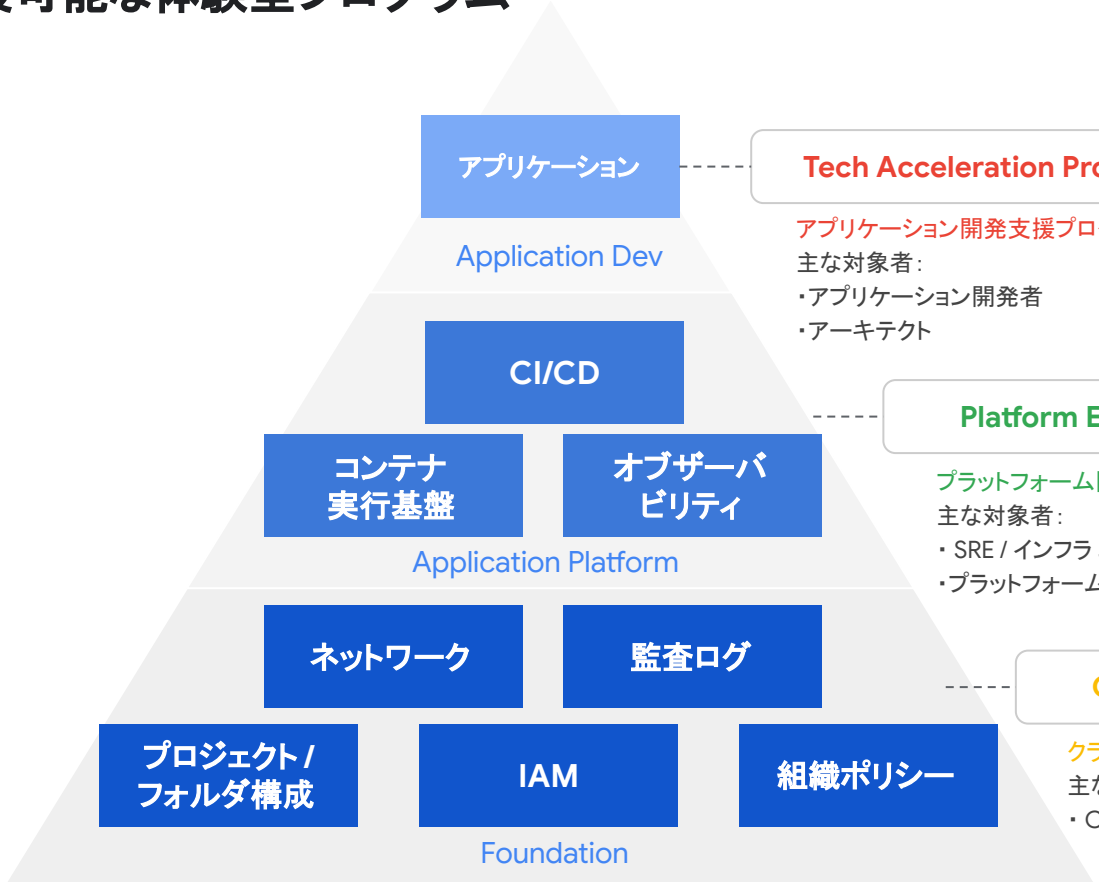
ハンズオン : Policy Controller による不適切な設定のコンテナ　デプロイの防止

クラスタ全体にセキュリティ　ガードレールを設定し不適切な設定のコンテナ　　デプロイを防止する



体験型プログラムのご紹介

ご支援可能な体験型プログラム



Tech Acceleration Program (TAP)

アプリケーション開発支援プログラム
主な対象者:
・アプリケーション開発者
・アーキテクト

Platform Engineering Jumpstart

プラットフォーム開発支援プログラム
主な対象者:
・SRE / インフラ エンジニア / DevOps エンジニア
・プラットフォーム アーキテクト

Cloud Governance as Code

クラウド ガードレール策定・構築支援プログラム
主な対象者:
・Cloud CoE / クラウド標準化チーム

Platform Engineering Jumpstart

Google Kubernetes Engine (GKE) をベースにした 開発者向けプラットフォームのプロトタイプ構築を Google Cloud のエンジニアがサポートします。

開発者向けプラットフォームを構築することによって、Developer Experience (DevEx) と開発生産性の向上が見込まれます。

本ワークショップを通じて、Kubernetes の基礎から実践的なプラットフォームの設計・構築方法について知見を得ることができます。

利用予定のプロダクト：GKE, Cloud Run, Cloud SQL, Service Mesh, Cloud Build, Cloud Deploy 等

Typically 2-3 days

対象者、アウトプット、アジェンダ

対象者

- SRE, インフラ エンジニア, DevOps エンジニア
- プラットフォーム アーキテクト
- アプリケーション開発者

アウトプット想定

- GKE をベースとした開発者向けプラットフォームのアーキテクチャ図
- 上記を実現するシステム (プロトタイプ)

アジェンダ

- プラットフォームの目指すべき姿やスコープの確認
- Kubernetes / GKE 概要説明
- プラットフォームのアーキテクチャ議論
- プラットフォームの構築・実装 (プロトタイプ)
- 継続的な支援についての議論

実施スケジュールの例

Day 0

Planning

本ワークショップで構築するプラットフォームの目指すべき姿やスコープの確認

必要に応じて Google Cloud や Kubernetes に関する概要説明・勉強会を事前に実施

Day 1

Knowledge Transfer

プロトタイピングに向けて必要となるナレッジのトランスファー

Customer Engineer による GKE や CI/CD, Operations プロダクトの各機能の概要説明

Day 2

Design & Architecting

開発者向けプラットフォームのベースとなる GKE クラスタや CI/CD, Operations の設計・アーキテクティングを実施

プロトタイプの開発に着手できる状態を作る

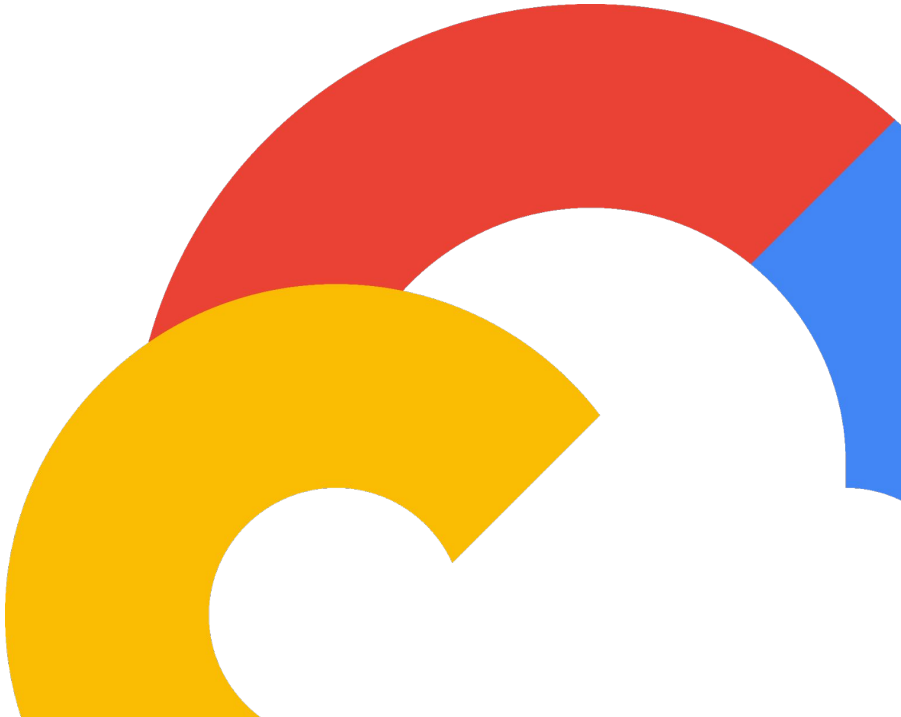
Day 3

Develop Prototype

ここまでにデザインした　プラットフォームを Customer Engineer が支援をしながら実装

必要となる情報、課題などをその場で解決。PoC の検証可能なプロトタイプを作り上げる

Thank you



Google Cloud