迅速に叶える、GKE Autopilot によるユニバーサルモダン アーキテクチャの実践

March 1, 2024

株式会社スリーシェイク
横尾 杏之介

Google Cloud

Speaker Introduction

Annosuke Yokoo

3-shake, inc

（@866mfs）

前職では、AWS をメインとしたクラウドインフラ構築支援を実施する中で2023 Japan AWS Jr.Champions も受賞

スリーシェイクにSRE として Join 後は Google Cloud や Kubernetes を中心としたアプリケーション インフラの構築・運用を通して、カルチャー変革を含むCloud Native な SRE 支援を実践中

Google Cloud の公式 User Community である「 Jagu'e'r 」の運営リードも しています

本日お話しする内容

話すこと

- 3-shake が支援したお客様事例を元に、アーキテクチャを構築する際の軸を言語化
- GKE Autopilot をコアとするワケと Container Workload の信頼性を高める技術

話さないこと

- ソフトウェア アーキテクチャについて
- Kubernetes リソース内部の構成や、リリース方式 / アップグレード戦略等の運用サイクル

Contents

01
モダン アーキテクチャの再考

みなさんの組織では普段どのようなことを　軸としてアーキ テクチャを考えていますか？

SRE Roadmap ✖ Cloud Native of 3-shake

SRE チームの定義
組織にあったSRE を定義しよう

SRE の開始
まずは小さく。
効果が最大限に出るところ
からコツコツと始めよう

SRE 実践
Toil 削減（自動化推進）
SLI・SLO の設定

SRE の発展と継続
SRE の文化が浸透する
Error Budget を元に
業務をコントロールしていく

CNCF 曰く
Cloud Native とは？

“

クラウド ネイティブ技術は、パブリック クラウド、プライベート クラウド、ハイブリッド クラウドなどの近代的でダイナミックな環境において、スケーラブルなアプリケーションを構築および実行するための能力を組織にもたらします。このアプローチの代表例に、コンテナ、サービス メッシュ、マイクロサ ービス、イミュータブル インフラストラクチャ、および宣言型API があります。

これらの手法により、回復性、管理力、および可観測性のある疎結合システムが実現します。これらを堅牢な自動化と組み合わせることで、エンジニアはインパクトのある変更を最小限の労力で頻繁かつ予測どおりに行うことができます。

Cloud Native Computing Foundation は、オープンソースでベンダー中立プロジェクトのエコシステムを育成・維持して、このパラダイムの採用を促進したいと考えてます。私たちは最先端のパターンを民主化し、これらのイノベーションを誰もが利用できるようにします。

https://github.com/cncf/toc/blob/main/DEFINITION.md#%E6%97%A5%E6%9C%AC%E8%AA%9E%E7%89%88

CNCF曰く
Cloud Nativeとは？

Cloud Native Architecture

- 疎結合である
- 復元力がある
- 管理しやすい
- 可観測である
- 堅牢な自動化
- 最小限の労力で頻繁かつ大きな変更が可能

クラウドネイティブ技術は、パブリッククラウド、プライベートクラウド、ハイブリッドクラウドなどの近代的でダイナミックな環境において、スケーラブルなアプリケーションを構築および実行するための能力を組織にもたらします。このアプローチの代表例に、コンテナ、サービスメッシュ、マイクロサービス、イミュータブルインフラストラクチャ、および宣言型APIがあります。

これらの手法により、回復性、管理力、および可観測性のある疎結合システムが実現します。これらを堅牢な自動化と組み合わせることで、エンジニアは最小限の労力で頻繁かつ予測どおりに行うことができます。

Cloud Native Computing Foundationは、オープンソースでベンダー中立プロジェクトのエコシステムを育成・維持して、このパラダイムの採用を促進したいと考えてます。私たちは最先端のパターンを民主化し、これらのイノベーションを誰もが利用できるようにします。

アーキテクトにまつわるよくある現象

- **Cloud や「CloudNative な」技術領域にノウハウが少ないエンジニアが集まった時に起きる現象**
  - Cloud サービスや Managed サービスを十分に使いこなせない
  - Application Modernization が出来ない
  - 最悪のケースでは可用性・信頼性などに問題が生じる

- **強すぎるエンジニアが集まった時に起きる現象**
  - 要件に対して技術的にオーバーキルしてしまう問題
  - 楽しくなり、色々な技術を触りたくなってしまって逆に複雑化させてしまう
  - 受け入れる組織的土壌と合致していないのに Micro Service にしてしまう等

アーキテクトにまつわるよくある現象

- ➢ **Cloud や「CloudNative な」技術領域にノウハウが少ないエンジニアが集まった時に起きる現象**
  - Cloud サービスや Managed サービスを十分に使いこなせない
  - 楽しくなり、色々な技術を触りたくなってしまって逆に複雑化させてしまう
  - 受け入れる組織的土壌と合致していないのに Micro Service にしてしまう等

Modern で Open で Simple で様々な指標が可視化されて、
且つ構成変更が容易なアーキテクチャ構成が至高
非常に難しいが、苦闘しながらでもここを目指していきたい

なぜ Modern な開発が必要か？

- 「2022 ACCELERATE State of DevOps」の中でも実施された検証によると、ソフトウェア デリバリーにおいて　組織的に高パフォーマンスに分類されたチームは、**リリース速度と安定性が高い**
- 開発速度と高頻度のデプロイ サイクルを支えるのは、**低い運用コストで実現可能な信頼性のある基盤**
- 「SRE の探求」でも言及されているが、「SRE が最も高い生産性と価値を示すのは、ビジネスにおける成功の実現に集中する時」である

つまり、Modern なアプリケーション開発の目的とすることは、

ビジネスを市場へ展開するスピード高めることに加え、安定した信頼性のあるサービス運用を実現することが重要

なぜ Modern な開発が必要か？

- 「2022 ACCELERATE State of DevOps」の中でも実施された検証によると、ソフトウェアデリバリーにおいて組織的に高パフォーマンスに分類されたチームは、リリ
-
- 「SRE の探求」でも言及されているが、S のは、ビジネスにおける成功の実現に集中する時である

運用レスに限りなく近く
Cloud Native な特性を持つプロダクトを選択肢としたアーキテクチャを考えることが
実現の近道ではないか

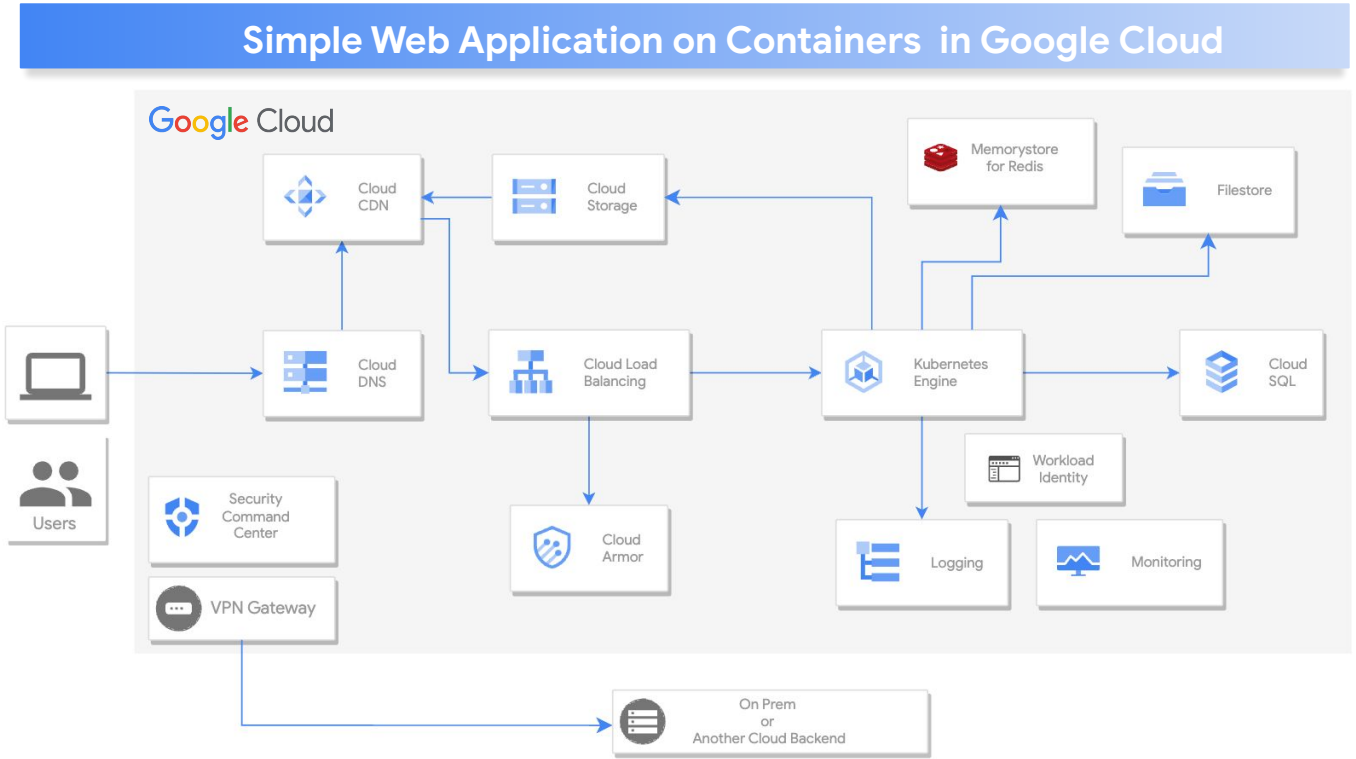
つまり、Modern な開発の目的とすることは、

ビジネスを市場へ展開するスピード高めることに加え、安定した信頼性のあるサービス運用を実現することが重要

アーキテクチャ ユースケース

- **Google Cloud 上に構築することをベースとする**
- **モノリシックなアプリケーション**
  - 迅速に実現するという観点から Micro Service ではなく、汎用的かつシンプルな（ユニバーサル） モノリシック Web アプリケーション アーキテクチャを想定
- **Cloud Native な特性を持つアーキテクチャを目指すべく、コアとする Workload には Contaier ベースのものを選択肢とする**
- **実現に至るプロセスを探るため、既存サービスインフラの改修ではなく、新規に構築する際のシナリオを想定**

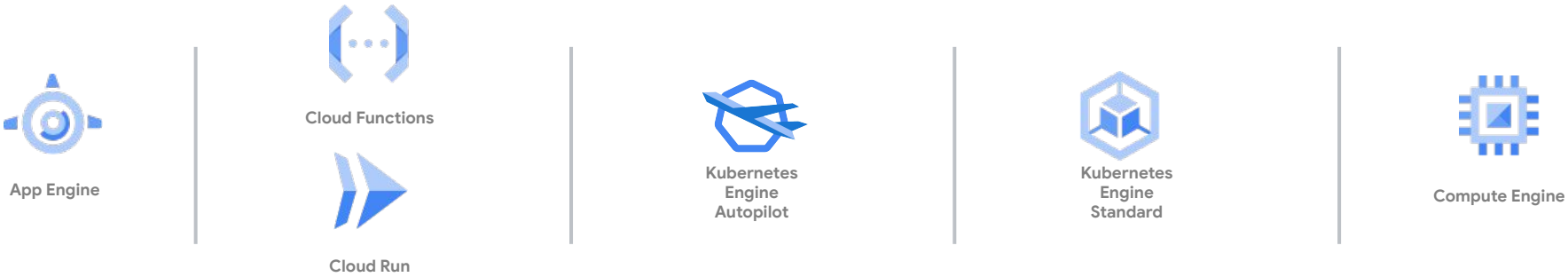
アーキテクチャ構成図


Simple Web Application on Containers in Google Cloud
Google Cloud
Cloud CDN
Cloud Storage
Memorystore for Redis
Filestore
Cloud DNS
Cloud Load Balancing
Kubernetes Engine
Cloud SQL
Users
Security Command Center
Cloud Armor
Workload Identity
Logging
Monitoring
VPN Gateway
On Prem or Another Cloud Backend

02
Container Workload
Google Kubernetes Engine - Autopilot

Container Workload

Google Cloud には多様な Container Workload 実行基盤の選択肢がありますが...



Managed — Unmanaged

機能要件及び非機能要件は十分に考慮に入れる必要がありますが、運用レスという観点から、Managed なサービスでの実現性を始めに検討した上で徐々に Unmanaged なサービスへ降りていくようなイメージで Workload を選定する

Container Workload

Google Cloud には多様な Container Workload 実行基盤の選択肢がありますが...

機能要件及び非機能要件と最大限向き合う事を前提にしつつも、
「**いかに楽を出来るか**」という観点から考える事も重要
それによりそもそもの Toil の生産量を減らすことが出来るかもしれない

Ex. 何でもかんでも Kubernetes が良いという訳ではない

Managed ← → Unmanaged

機能要件及び非機能要件は十分に考慮に入れる必要がありますが、運用レスという観点から、Manged なサービスでの実現性を始めに検討した上で徐々に Unmanaed なサービスへ降りていくようなイメージで Workload を選定する

02

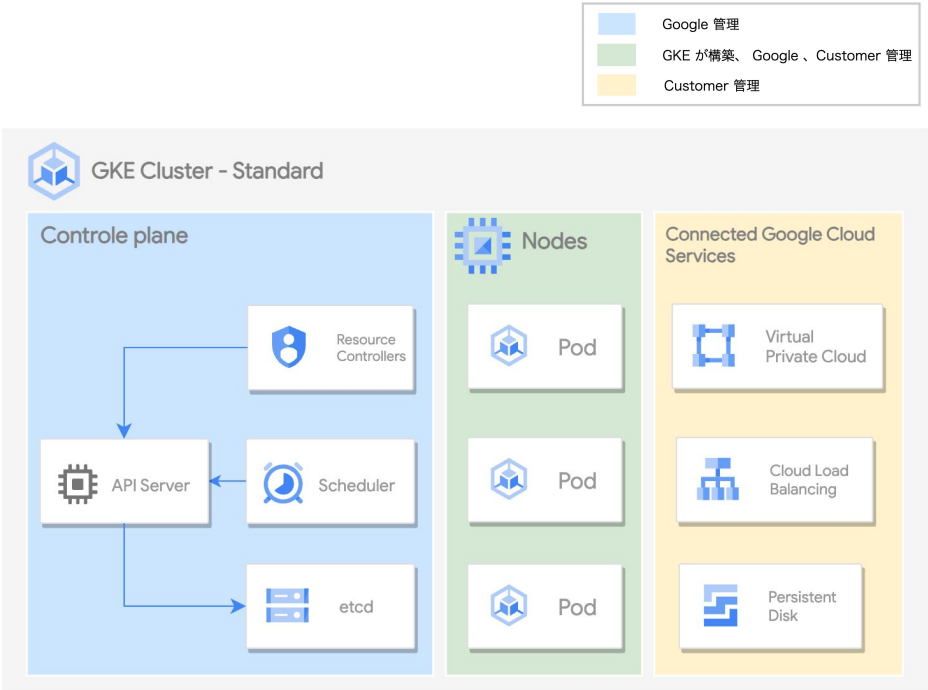
GKE Autopilot

- Introducing GKE Autopilot
- GKE Autopilot - Scaling workaround
- なぜ GKE Autopilot なのか

Google Kubernetes Engine - Standard

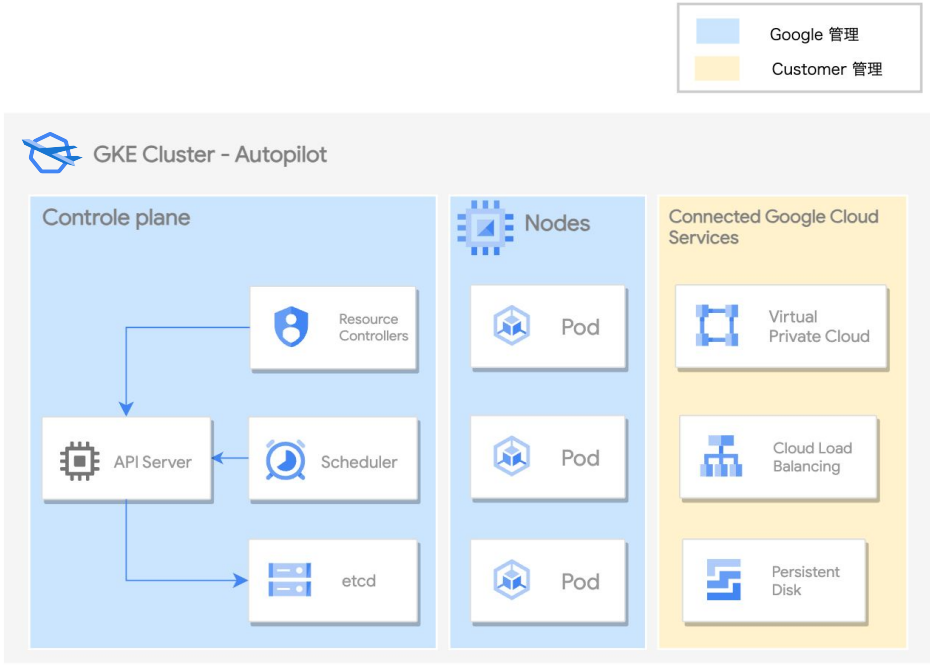
Google Cloud マネージドの Kubernetes 環境

- 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提供
- アップグレードやバックアップと復元など、完全に自動化されたクラスタ ライフサイクル管理
- ノードの自動プロビジョニング
- ワークロードを簡単に移行できる自動ツール
- **Google Cloud の各種サービスと統合**されている



Google Kubernetes Engine - Autopilot

- Control plane に加えて **Node も完全マネージド**な Kubernetes クラスタ
- Workload ベースの考え方（ Pod 単位の課金 / Pod 単位の SLA ）
- 本番ワークロードに適したベスト プラクティスがデフォルト構成
- GKE の良さを踏襲しつつ、**より運用レス**を実現できる

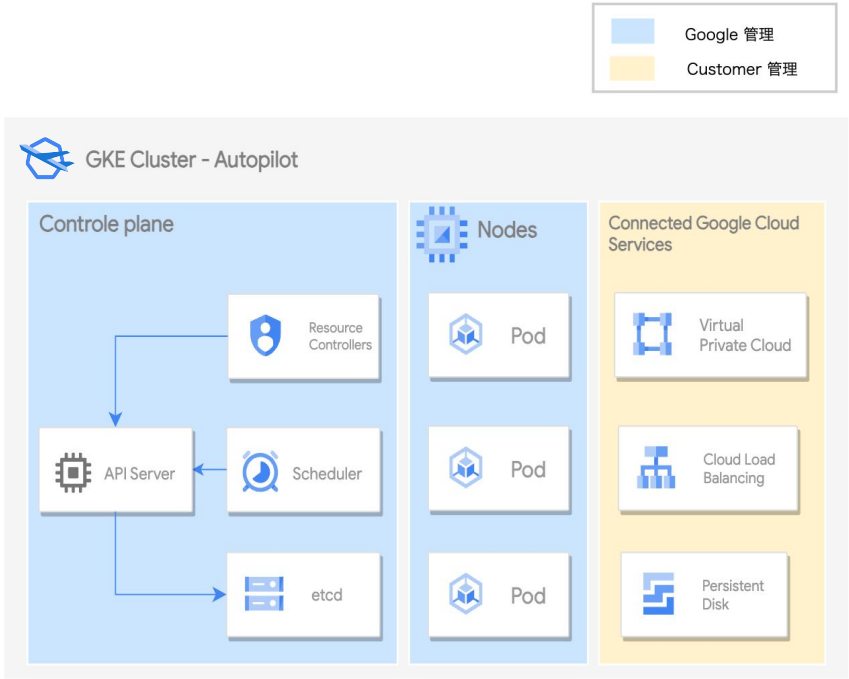


Node も完全マネージドな状態とは？

これまで行っていた Node の運用を意識しない

- ワークロードの Toleration の設定
- ワークロードに合わせたノードの プロビジョニング
- ノードのハードウェア定義からノード Taint の構成と適用
- Node の作成、更新、削除は自動
- バージョンアップグレードも自動

ユーザーは、より Kubernetes の中の世界に集中 することが可能に！



GKE Autopilot は制約もある

（組み込みのセキュリティが充実しているという言い方もできるが...）

- Privileged pod（特権コンテナ）は利用出来ない
- GKE がノードを管理するため hostNetwork は利用出来ない
- hostPath は /var/log 配下のみ許可
- Node の sysctl / kubelet のパラメーター チューニングは出来ない
- Node への SSH アクセスをブロック
- Mutating Admission webhook は一部使えない , etc.

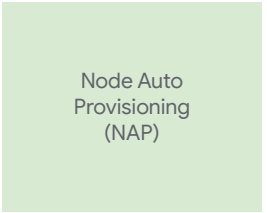
詳細は公式ドキュメント

Node がマネージドである仕組み

Node の作成、スケール、削除には従来の GKE が備えている、

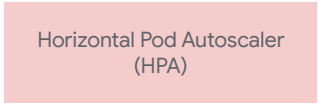
Cluster autoscaler (CA) と **Node auto provisioning (NAP)** を使用

詳しくは[こちら]で解説されている



Node の作成、削除は Pod のスケジューリングに依存している

- 余剰の Node を予め用意しておいてスパイクに備えるということは基本出来ない
- Node 上にスペースを作りにくいため、Pod の水平スケールと同時に Node のスケールが走りやすい
  - [特にバージョン 1.28 より前の GKE の場合、Node あたりの Pod 数の上限は 32 個]となっているため Node のスケールが多くなることも



GKE Autopilot - Scaling workaround

01 Balloon pods

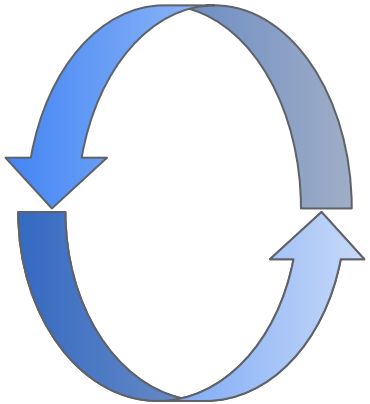
- 余剰のノードを予め用意しておいて、ノードのスケールを発生せずに　Pod をスケジュールする
  https://wdenniss.com/gke-autopilot-spare-capacity

02 イメージ ストリーミング

- コンテナ イメージをリモートのファイル システムから Lazy-pulling で　ダウンロードして Pod の起動時間を高速化する
  https://cloud.google.com/kubernetes-engine/docs/how-to/image-streaming?hl=ja

03 Pod や Node を事前にスケールする

- トラフィックが増えるタイミングが予測可能な場合、事前に Pod の　スケールを増やすことで Node のスケールを行う
  https://cloud.google.com/kubernetes-engine/docs/tutorials/reducing-costs-by-scaling-down-gke-off-hours?hl=ja



なぜ GKE Autopilot なのか

Kubernetes 運用やノードにおける課題の例

Kubernetes 運用の課題

- リソース管理 (見積り、調整作業)
- アップグレード
- Kubernetes エコシステムの運用
- セキュリティ対策 (脆弱性対応)
- 監視
- 障害対応

ノードのリソース管理における課題

- 事前に明確なリソース要件を把握できない
- オーバー プロビジョニングによるコスト増加
- taints , toleration , nodeAffinity など Pod の Node に対するスケジューリングを考える

手動による定期的なリソース見積もり・調整作業はサービスの成長に比例して運用負荷が増加する

より効率的に運用し作業負荷を軽減するために、各種自動化機能やマネージドサービスの活用が求められる

GKE Autopilot がもたらす価値

運用負荷低減

Node 管理をGoogle が担うことにより、ユーザーは Kubernetes ワークロードに集中できる

コスト最適化

Node 単位ではなく Pod単位の課金となるため、よりユーザー側の利用に即したコストへ最適化

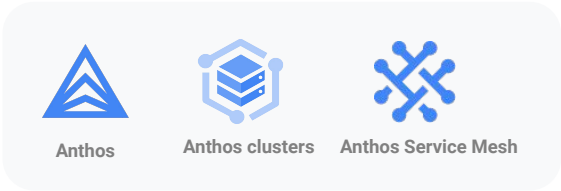
設計コスト削減

GKE に必要なベスト プラクティスが組み込みされているため、迅速にプロダクション レディなクラスタを利用可能

GKE Autopilot の組織適正とユースケース

- 費用対効果や自動化、セキュリティと信頼性の改善から、**基本的にはどの組織にも推奨**できると考える
- Kubernetes の特性をベスト プラクティスに沿った形を反映し、運用レスを実現できるのは最大のメリット
  - Google Cloud コンソールから新規にクラスタを作成する際のデフォルトは Autopilot モード
- **特に効果的なユースケース**
  - バッチ処理 / オートスケールの速さをそこまで求めない / 大量のリクエスト トラフィックを捌かない / 開発・テスト環境用途 / ステートレス

- 大規模サービスの構築や他パブリック クラウドとのマルチ クラスタ運用の場合には、フリートによる管理された**GKE Enterprise（Anthos）**という選択肢もある



GKE Autopilot 参考資料

Introducing GKE Autopilot

- 完全マネージドな k8s ! GKE Autopilot を解説する
- GKE Autopilot と GKE Standard の比較
- GKE Autopilot の性能検証
- Autopilot クラスタのトラブルシューティング

Security

- GKE Autopilot のセキュリティ機能

Cost

- GKE Autopilot の費用対効果と従来のマネージド Kubernetes との比較
- **autopilot-cost-calculator**

03
Architecture around Google Kubernete Engine

03

Architecture around GKE

- Security
- Observability
- CI/CD（Release Engineering）

Security

Kubernetes Security Posture Management（KSPM）

KSPM ツール

- Kubernetes クラスタの網羅的なセキュリティ スキャンと状態管理の機能
- ベスト プラクティスを適用するためのルールベースのセキュリティ対策
- OSS のスキャンツール + 手動管理 or 有償製品での統合管理

KSPM の運用

- まずはスキャンを実施、クラスタの状態を把握する
- 非準拠項目の対応要否、優先度を決定
    - リスクの大きさやコスト パフォーマンスの観点から
- 継続的スキャンによるポスチャー監視（設定変更時、定期スキャンなど）
- Admission Control に適用することで設定の強制も可能

https://speakerdeck.com/kyohmizu/an-quan-na-kubernetes-huan-jing-womu-zhi-site?slide=29

trivy k8s

Kubernetes クラスタの設定ミスを検出

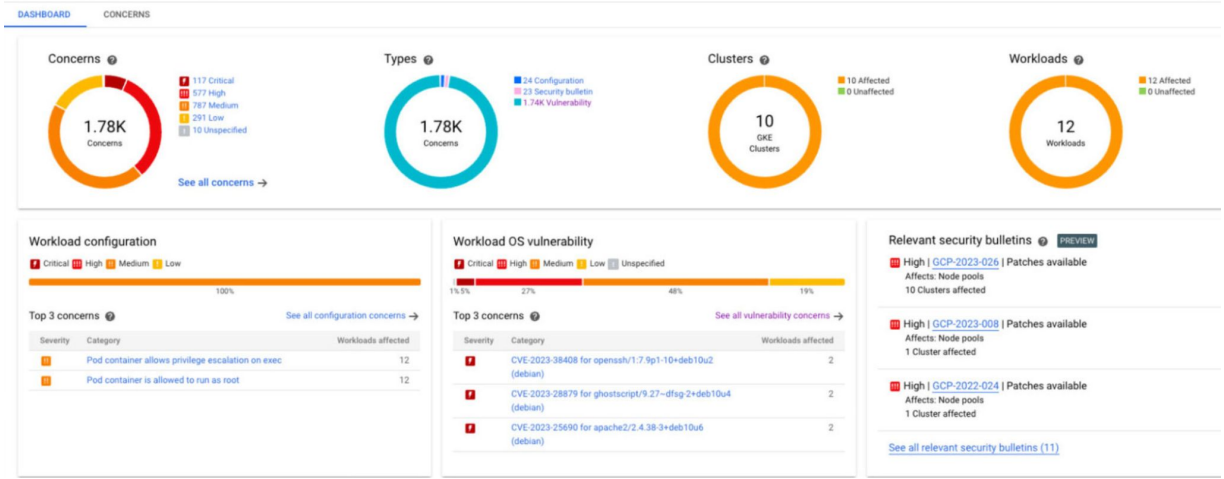
- クラス コンポーネントと、ワークロードの設定ミスや脆弱性のスキャン
- 複数データソースをベースにした独自の検出項目
  - https://avd.aquasec.com/misconfig/kubernetes
- Kubernetes Operator による継続的スキャン

```
$ trivy k8s cluster - - scanners misconfig - - report all - - components infra

$ trivy k8s cluster - - scanners vuln - - report summary

$ trivy k8s cluster - - compliance k8s-cis -- report all
```

GKE Security Posture

GKE クラスタのセキュリティ管理・運用ダッシュボード

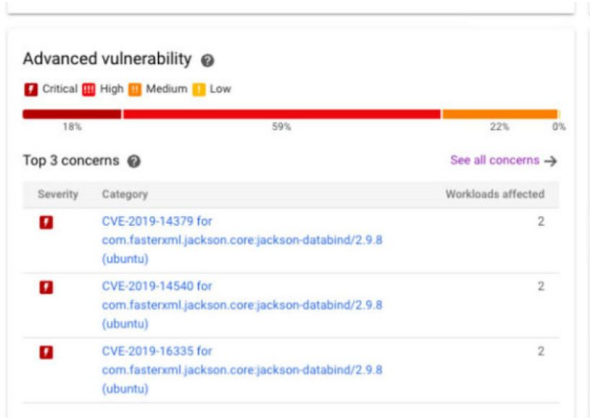
- クラスタとワークロードの構成チェック、イメージ脆弱性スキャン
- 無料で利用できるのは、Pod Security Standards (PSS) ベースの構成チェックとイメージの OS スキャンのみ
- 追加のセキュリティ ポリシーは、GKE Enterprise のみ利用可能



Advanced Vulnerability Insights for GKE

言語パッケージのスキャンと脆弱性を検出

- Java、Go、JavaScript、Python 等の言語パッケージのスキャンと脆弱性検出を提供
- ランタイムで検出された脆弱性の半分以上は言語パッケージで検出されてるため、OS レベルだけでは不足が生じる
  - Sysdig 2023 クラウドネイティブ セキュリティおよび使用状況レポート
- クラスタ単位での有効化が可能
- GKE Enterprise では標準的な組み込み機能



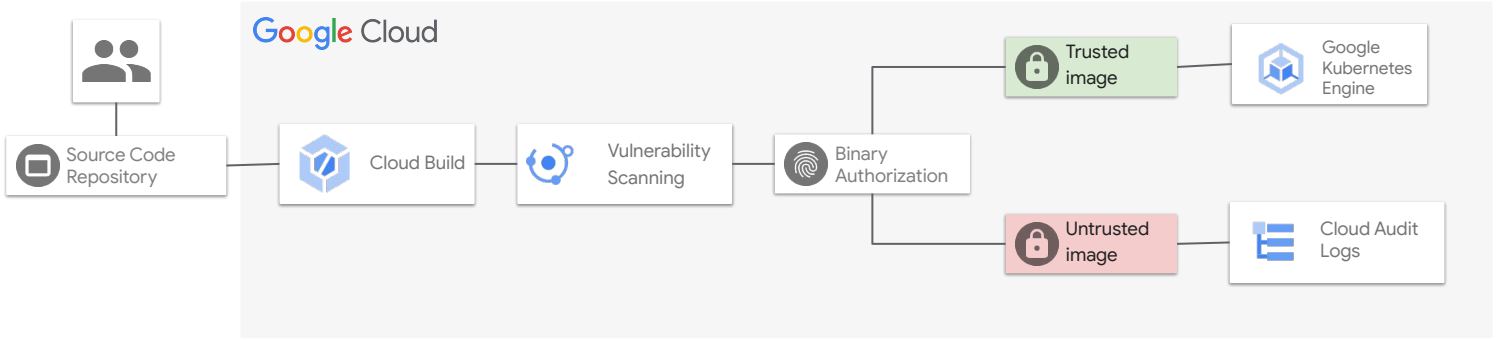

```
resource "google_container_cluster" "autopilot-cluster" {
...
   security_posture_config {
    mode = “BASIC”
      valnerability_mode =  “VULNERABILITY_ENTERPRISE”
  }
}
```

https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/container_cluster

Binary Authorization

コンテナベースのアプリケーションに ソフトウェア サプライチェーンのセキュリティを提供

- 信頼できる container image の署名とポリシー チェックを行い、信頼できる image のみがデプロイされることを保証
- 悪意のあるコードや脆弱性のあるイメージの使用を未然に防ぐ
- さらに詳しくはこちら
  - Demo - Gating Deployments with Binary Auth
  - Google Cloud Binary Authorization 徹底調査



Container Threat Detection

コンテナに対する脅威の検出

- Security Command Center のプレミアム ティアに組み込まれているサービス
- 不審なバイナリ実行, 不審なライブラリ読み込みを検出
- **イメージの脆弱性スキャン等だけでは検知・防ぎきれないような攻撃の検知を実現**
  - Security Command Center や Cloud Logging に連携しアラートを飛ばすことも可能
  - セキュアな GKE クラスタを構築するために知っておきたいポイント 2022 年夏

Observability

Kubernetes 上で observability を実現する際に立ちはだかること

アプリケーションの監視

アプリケーションとそのプロセスの状態を可視化し、エラーやパフォーマンスの情報を元にアプリケーションコードのデバッグを行う
ゴールデン シグナルなどに基づく、アプリケーションの指標を把握する

コンテナの監視

Kubernetes リソースの一般的な問題やコンテナ終了メッセージの管理、実行中のコンテナのデバッグを行う
Pod・Service・Persistent Volume などとコンテナ自体の状態を把握する

ホストの監視

アプリケーションが原因でない場合の、コントロール プレーン・ワーカーノードなどの Cluster の状態の確認、各ワークロードの状態を深く掘り下げるログの確認を行う
ノードの死活監視と **kube-apiserver・kube-scheduler・kubelet** の状態を把握する

Kubernetes 上で observability を実現する際に立ちはだかること

課題

- サービス成長に伴う監視すべきコンポーネントの増加
- Kubernetes 周辺エコシステムの継続的なメンテナンス
    - Prometheus , Istio, etc. アップグレード・セキュリティ対応 や可用性の担保、　　リソース管理などを継続的な実施が必要

Istio

Google
Kubernetes Engine

Kubernetes を導入したという既成事実から入った組織であるほど、これら運用は大変となり、
第2章として待つオペレーション業務、Kubernetesを保守する辛みを極力減らしたい

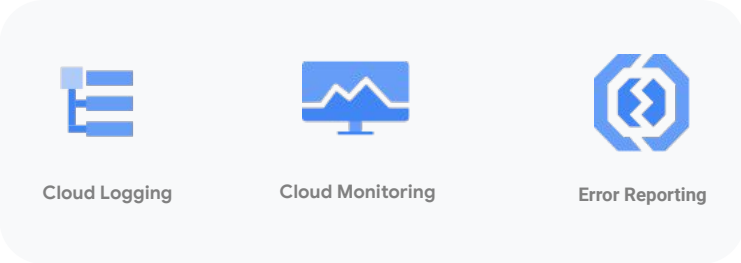
Container Workload と同様に Managed なサービスでの実現性を始めに検討した上で徐々に
Unmanaged なサービスへ降りていくようなイメージで observability を実現するサービス選定する

Google Cloud Managed Observability ①

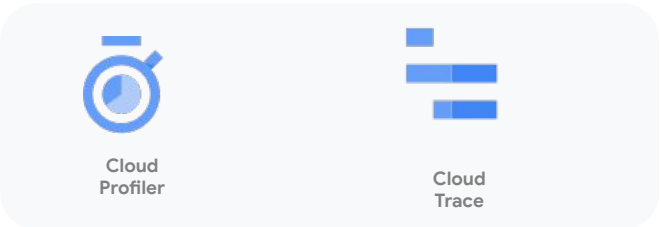
Observability (Cloud Operations) for GKE

- Google Cloud のあらゆるサービスと Default で連携　されているのが最大のメリット
- また、SaaS として監視基盤を利用する事が可能なので基盤運用を行う必要がなく、その分工数を減らせる

※ もちろん可用性要件やオペレーションによっては OSS や　他の SaaS の利用も検討に入れる必要あり



Application Performance Management（APM）



Google Cloud Managed Observability ①

Observability (Cloud Operations) for GKE

1. GKE クラスタを作成すると、デフォルトで Cloud Logging と Cloud Monitoring が有効となる
2. Metrics / Logging Agent が各ノードにデプロイされ、システムとワークロードの情報を収集できる

    - クラスタの重要な指標　: CPU 使用率、メモリ使用率、対応待ちのインシデント数
    - クラスタリソースの情報　　: インフラ ストラクチャ、ワークロード、サービス
    - コンテナリソースの情報　　: Namespace、ノード、ワークロード、Service、Pod
    - Pod とコンテナのログ情報　　: 時間の関数として指標を表示し、ログエントリを確認

3. Cloud Monitoring ではデフォルトで様々な GKE ダッシュボードを提供している

    - GKE Compute Resources　　: クラスター・ノード・ワークロード毎にリソースの利用状況を可視化
    - GKE Interactive Playbook　　: Crashlooping / Unschedulable Pods に対するプレイブックを確認できる
    - GKE Workloads at Risk　　: ワークロードの CPU / Memory の Request 未設定やリソース不足を可視化できる

Observability for GKE で利用されるテクノロジー:

https://speakerdeck.com/aoto/how-to-efficiently-run-gke-and-managed-services-with-datadog?slide=15

Google Cloud Managed Observability ②

[GKE Dataplane V2 Observability(Preview)]

eBPF を利用した Kubernetes データプレーン

1. Kubernetes ネットワーク用に最適化された GKE クラスタのデータプレーン
2. eBPF(Cilium) を使用して実装することでパケット内に Kubernetes 固有のメタデータを使用し、
   より効率的なパケットのルーティング・処理を行う
3. 従来の Observability for GKE に加えて、Hubble による Pod のネットワーク テレメトリを収集する

   Dataplane V2 Observability で利用されるテクノロジー:

4. さらに、アプリケーションのパフォーマンスを可視化したい場合は別の実装が必要

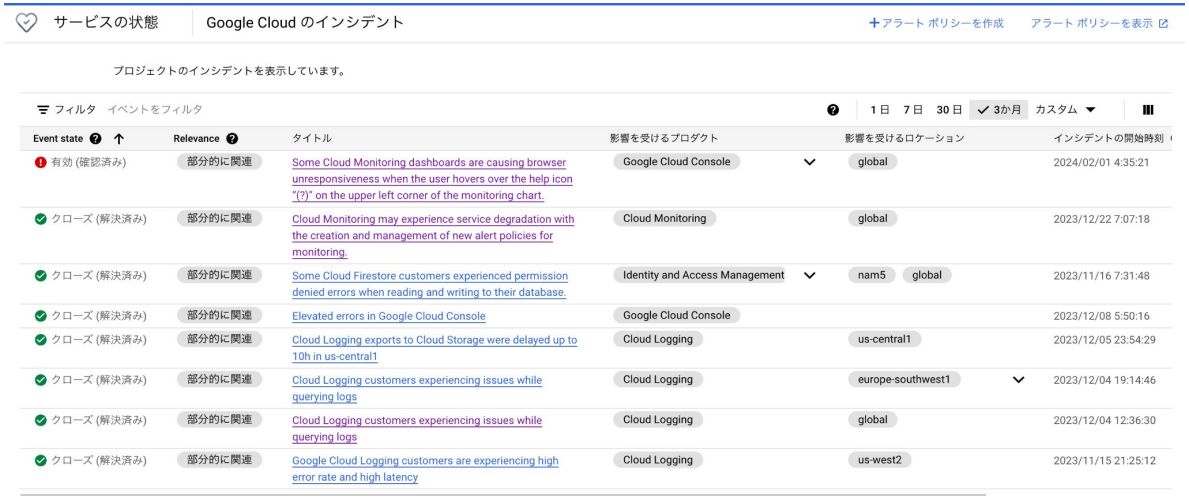
   アプリケーション パフォーマンスの可視化:

https://speakerdeck.com/aoto/how-to-efficiently-run-gke-and-managed-services-with-datadog?slide=15

Personalized Service Health

Google Cloud のインシデント管理・運用ダッシュボード

- Dashboard , Alert (Cloud Monitoring , Cloud Logging), API などのインシデント管理のワークフローとして　　組み込み可能な手法を提供
  - https://cloud.google.com/blog/products/devops-sre/personalized-service-health-is-now-generally-available/?hl=en



CI/CD (Release Engineering)

CI/CD を行う目的（モチベーション）とは?

CI

- テストを自動化したい
- コードの品質を高めたい
- Release Ready な状態にしたい

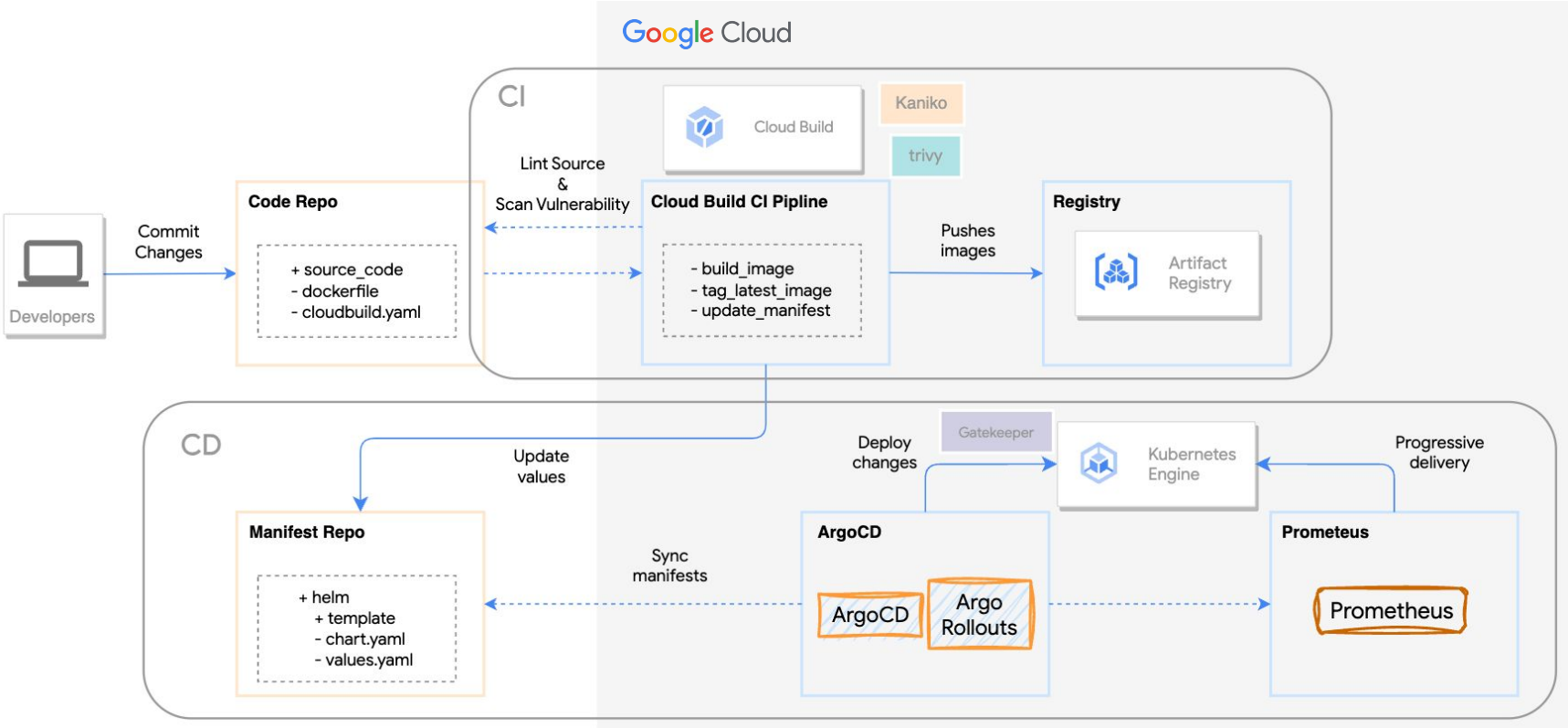
CD

- 簡単にデプロイ出来るようにしたい
- 何かあればすぐ切り戻したい
- 承認など段階を踏んだデプロイをしたい

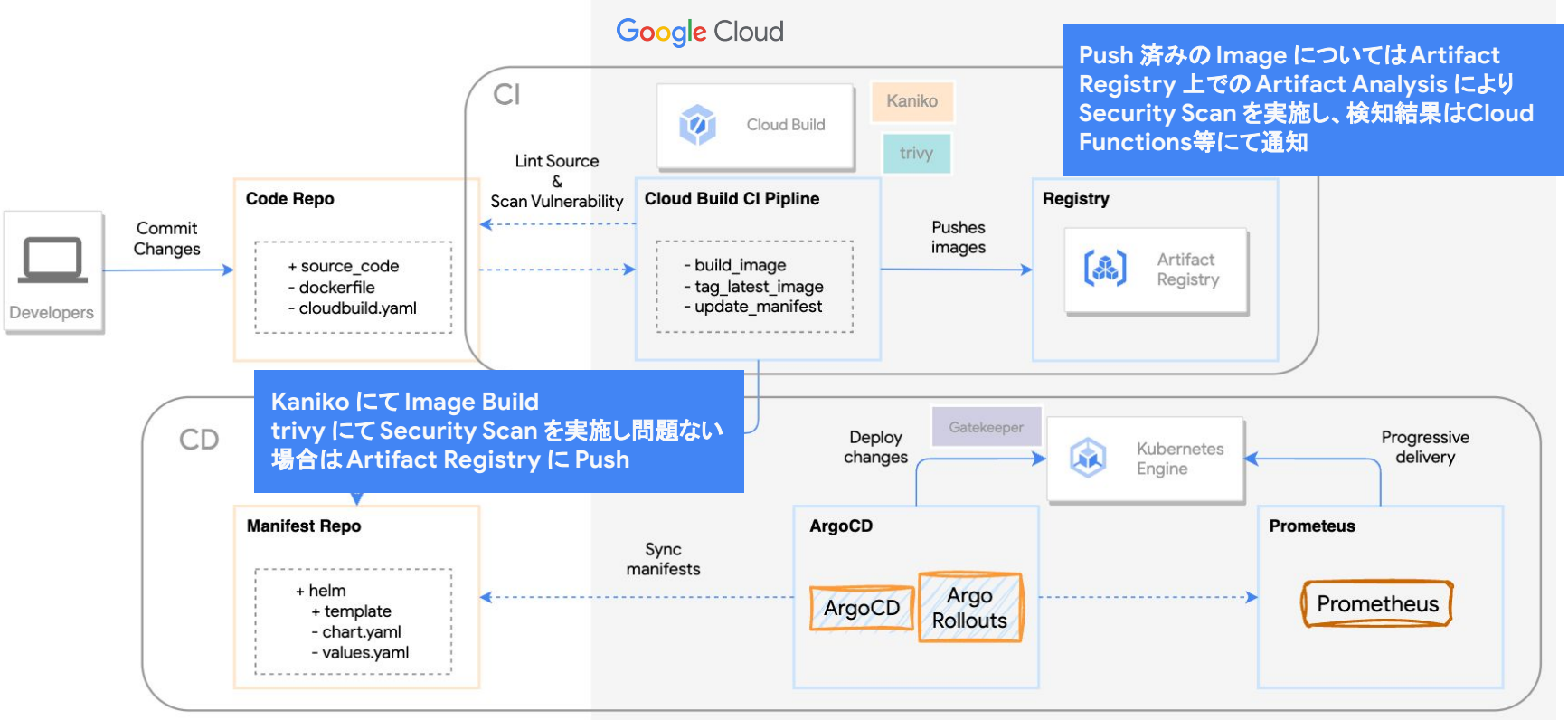
障害発生の多くは人間が手を加えたとき（新しいバージョンのリリース時）に起きると言われている

CI/CD は **リリース サイクルの向上と信頼性の両方を担保する**為のアプローチであるべき

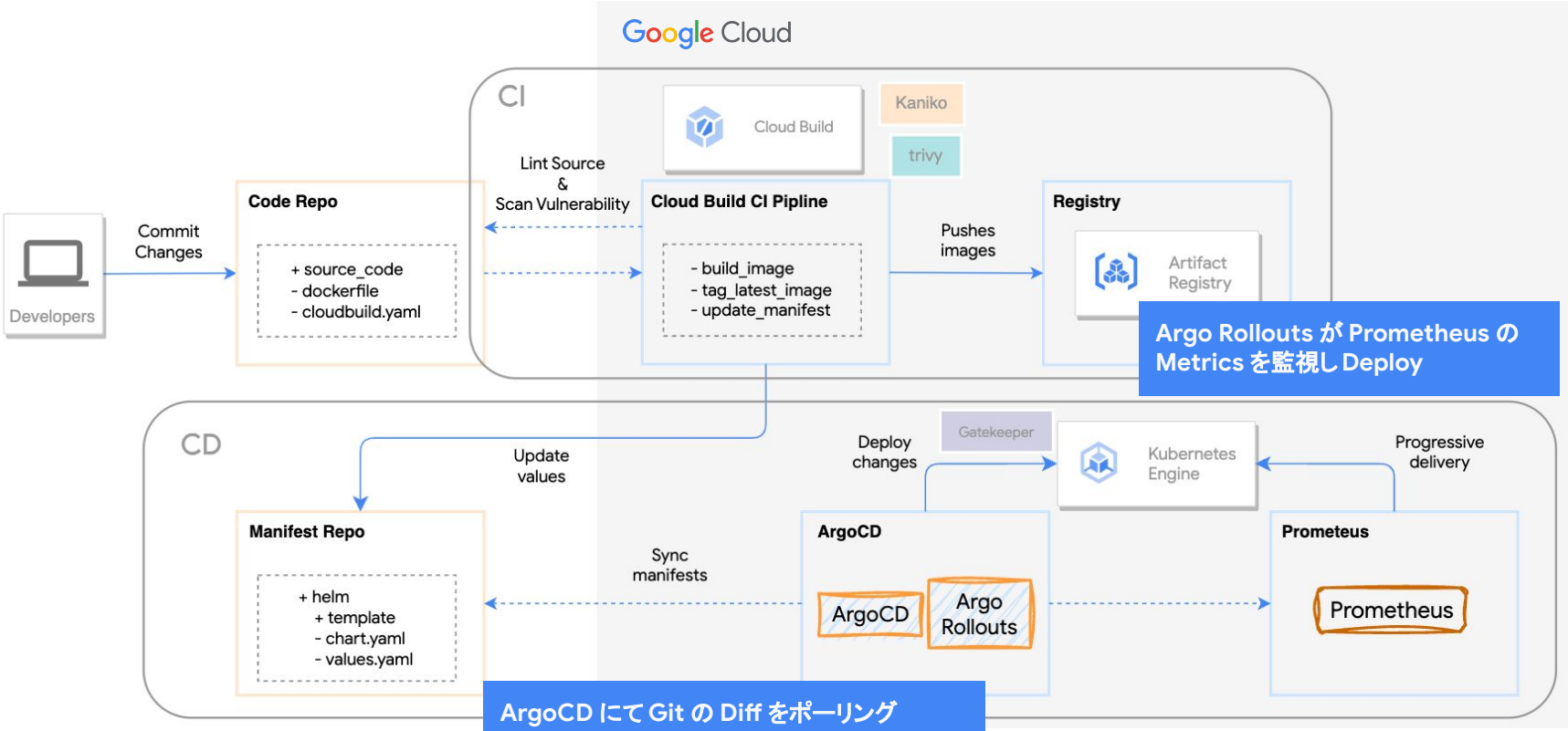
GKE でのよくある CI/CD の実践例



GKE でのよくある CI/CD の実践例



GKE でのよくある CI/CD の実践例



GKE でのよくある CI/CD の実践例

ベースとなる最低限の CI/CD であり、組織やサービス特性に応じて、

より信頼性のある構成を目指していくべき

まずはこれをベースに GitOps を始めてみると良いのではないか

CI/CD - Kubernetes / Terraform OSS

- CI/CD に関連するもの（下記掲載）や他にも日々新しい OSS が 開発されているのでご参考までに掲載
  https://ossinsight.io/collections/kubernetes-tooling/

Kubernetes

- conftest
- konstraint
- kubeconform
- pluto
- checkov
- polaris
- kube-bench
- gatekeeper
- kyverno

Terraform

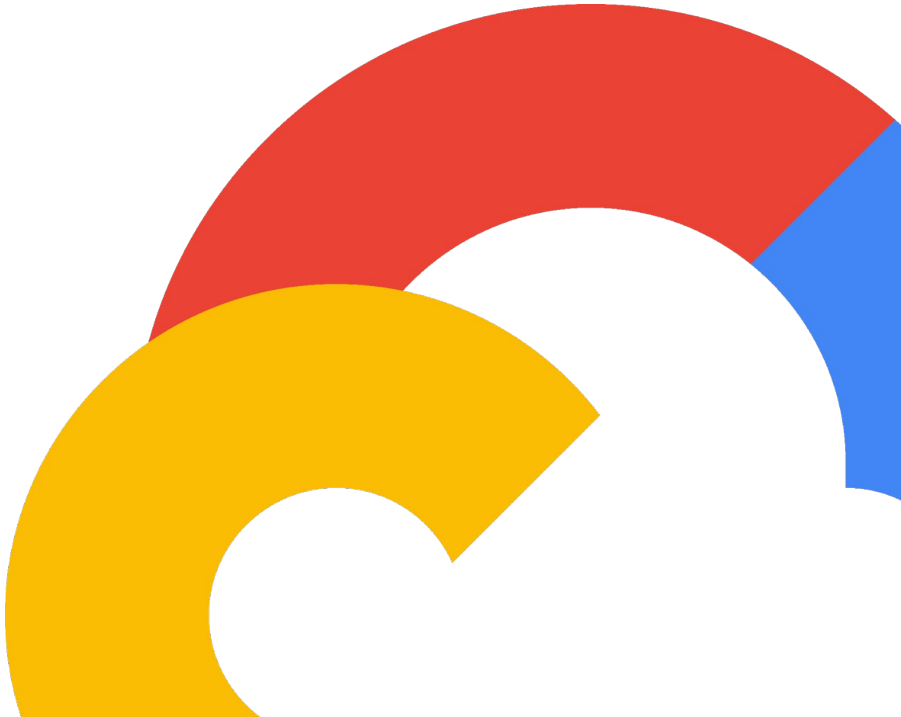
- tflint
  - tflint-ruleset-google
- tfsec
  - tfsec-pr-commenter-action
- tfcmt

04
まとめ

まとめ

- 運用レスに限りなく近く、Cloud Native な特性を持つプロダクトを選択肢としたアーキテクチャは、**安定した信頼性のあるサービス運用**に近づく
- GKE Autopilot は各種自動化機能やセキュリティなど推奨されるベスト プラクティスが組み込まれた**運用レスを実現できるクラスタ**である
- GKE や GKE と統合された周辺サービスにより、Kubernetes 本来のワークロードに集中できるようなアーキテクチャ構成が可能となる
- 迅速なビジネスの実現には、**Managed なサービスでの実現性を始めに検討**した上で徐々に Unmanaged なサービスへ選択肢を取っていく

Thank you



Google Cloud