Google Cloud

Google Kubernetes Engine (GKE) で実現する運用レスな世界

Google Cloud
アプリケーション モダナイゼーション スペシャリスト
内間 和季

Table of Contents

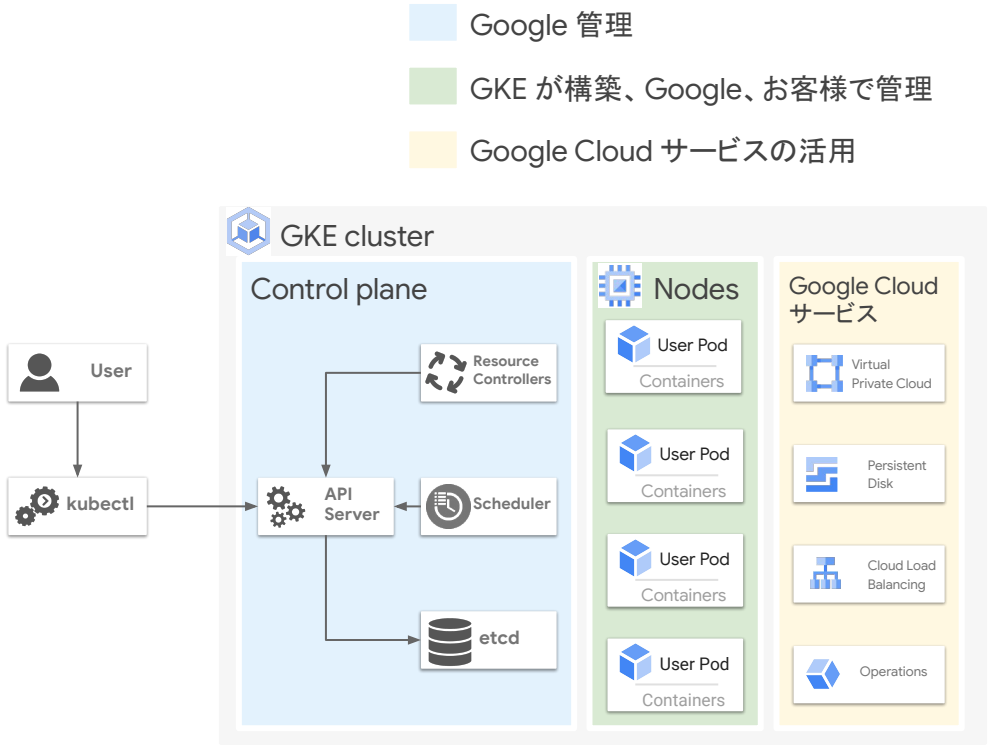
01

Google Kubernetes Engine (GKE)
概要

Google Kubernetes Engine - Standard

Google のマネージド Kubernetes 環境

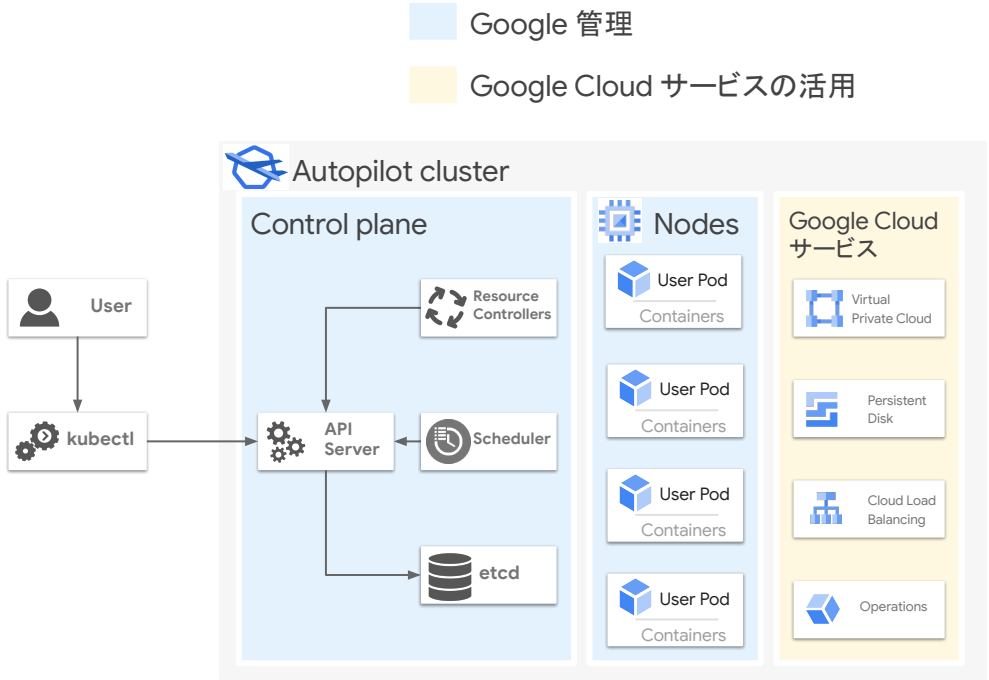
- **自動**でスケーリング、
  アップグレード、ノード修復
- Kubernetes 運用の**ベスト プラクティス**をマネージド サービスとして提供
- **セキュリティとコンプライアンス**
  - 業界をリードするセキュリティ機能群
  - HIPAA や PCI DSS など各種コンプラ イアンスに準拠



Google Kubernetes Engine - Autopilot

特長

- Control plane に加え
  Node も Google マネージド
- 本番ワークロードに適した
  ベスト プラクティスが適用済み
- GKE の利用体験は残しつつ利便性を高めたモード
  - Daemonset 等のリソースも利用可能
- Workload（Pod）ドリブン
  - Pod 単位での課金、Pod への SLA

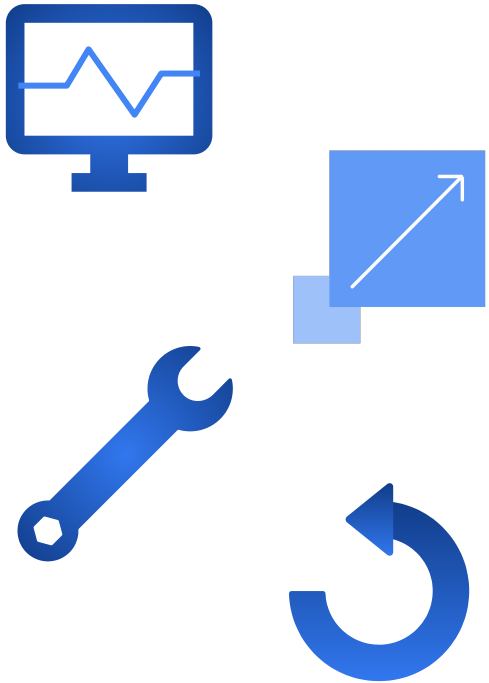


Kubernetes 運用における課題の例

Kubernetes 環境では数多くの運用作業が必要となる

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

本セッションでお話する観点



サービスの成長に比例して運用負荷は増加する

より効率的に運用し作業負荷を軽減するためには、**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

02

ワークロードやノードの
リソース管理

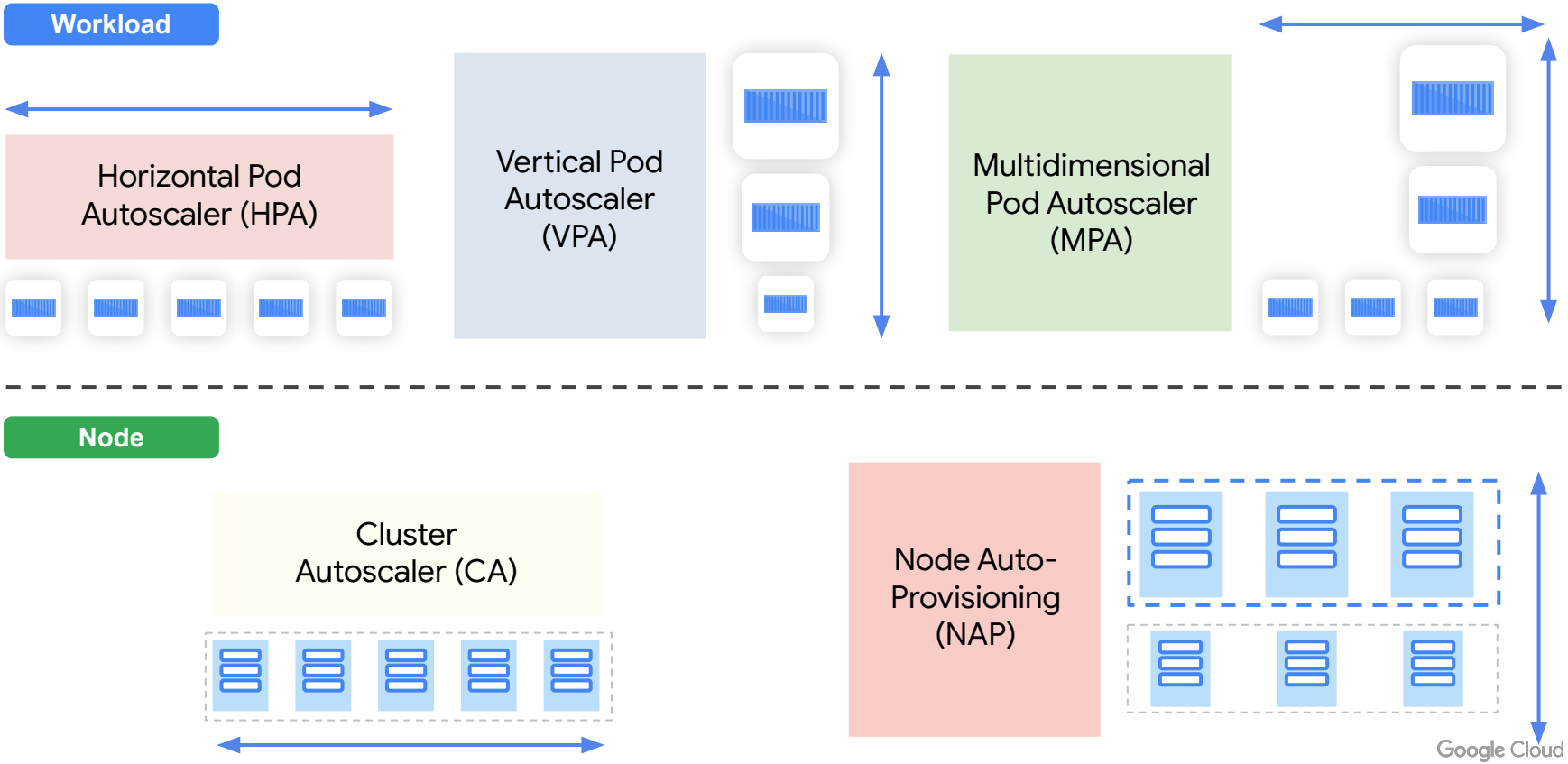
アプリケーションやノードのリソース管理における課題

新規ワークロードやバースト性があるワークロードなどは明確なリソース要件を事前に把握することが難しい

多めに見積もるとオーバープロビジョニングでコストが想定よりもかかってしまう可能性もある

Kubernetes および GKE ではワークロードやノードのリソース量を自動的に調整 / スケールさせる機能を提供しており、これらを活用することで **手動による定期的なリソース見積もり・調整作業の負荷を下げる** ことができる

GKE がサポートする自動スケール機能



Workload

HPA - Horizontal Pod Autoscaler

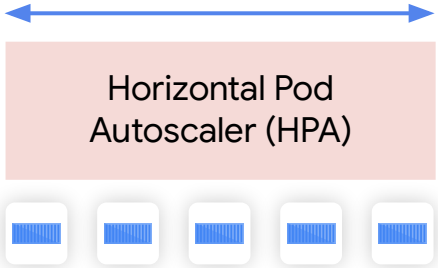
ワークロードの CPU やメモリの消費量等に応じて、自動的に Pod 数を増減させる機能

CPU / メモリ以外にも、カスタム指標や外部指標を使ったオートスケールもサポート

API version により、利用可能なメトリクスが異なる

- **autoscaling/v1**: CPU 利用率のみ
- **autoscaling/v2beta2**: カスタムメトリクスや外部メトリクスをサポート

複数のメトリクスを利用する場合、各メトリクスで算出されたレプリカのうち最大値を選択する



[ターゲットの決め方の例]

$$\text{Utilization target} = \frac{1 - \text{buffer}}{1 + \text{percent}}$$

10% CPU buffer

$$\frac{1 - 0.1}{1 + 0.3} = 0.69$$

Expect 30% traffic growth
in 2 or 3 minutes a Pod takes to start up
(consider node provisioning)

HPA - トラフィックベースの Autoscaling Preview

Workload

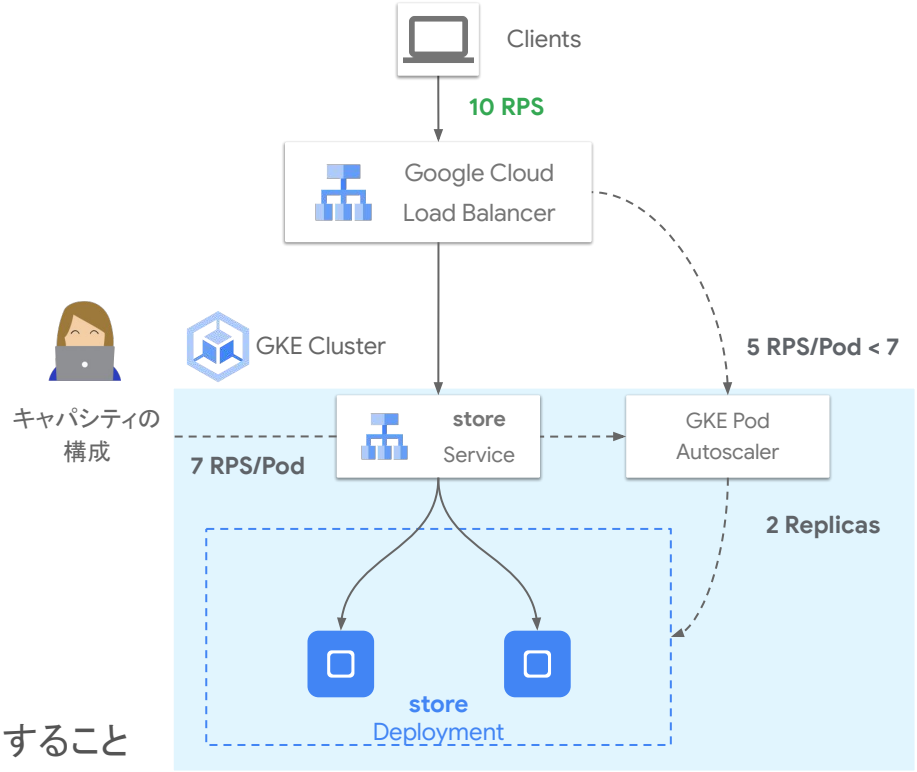
Gateway API Preview と組み合わせることにより、
トラフィック (RPS) ベースのオートスケールが可能

定義した Service Capacity (RPS) に対する Utilization
を基に HPA による Pod のオートスケールを実現

$$\text{replicas} = \text{ceiling}\left[\frac{\text{current traffic}}{\text{averageUtilization} \cdot \text{maxRatePerEndpoint}}\right] = 2$$

10rps (current traffic)
70% (averageUtilization)
10rps (maxRatePerEndpoint)

RPS だけでなく、CPU 等をセカンダリターゲットとして設定することを推奨



VPA - Vertical Pod Autoscaler

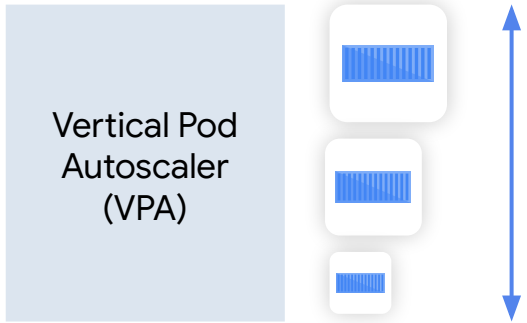
実行中のワークロードを分析し、CPU やメモリの requests / limits 値の推奨値算出やリソース値を自動更新する機能

VPA の更新モード (updateMode):

- **Off**: 推奨値を算出するのみ
- **Initial**: Pod の作成時に割り当て、既存 Pod の再起動無し
- **Auto**: 既存 Pod の再起動あり

急激なトラフィック増に対処する場合は、VPA ではなく HPA を利用することを推奨

VPA を使う場合はまず **updateMode: Off で推奨値の算出から始めてみる**



Workload

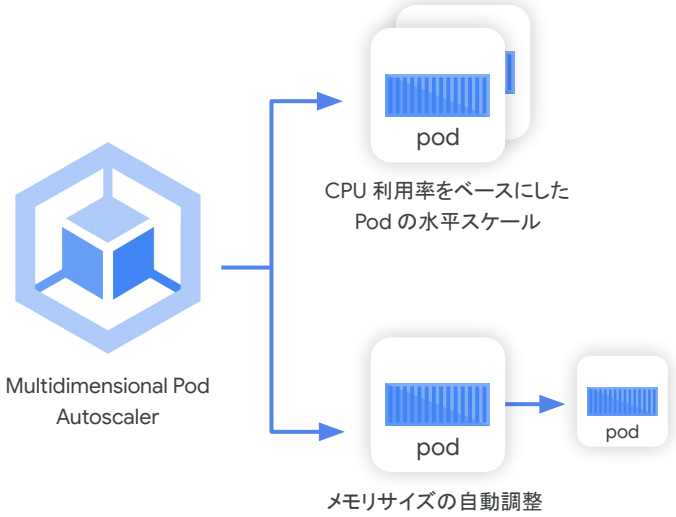
MPA - Multidimensional Pod Autoscaler Preview

HPA と VPA を併用したオートスケール 方式を提供

CPU ベースの HPA とメモリベースの VPA を組み合わせたオートスケールを 1 つのオブジェクト (MultidimPodAutoscaler) で実現

内部実装的には MPA コントローラが HPA と VPA のオブジェクトを作って管理している

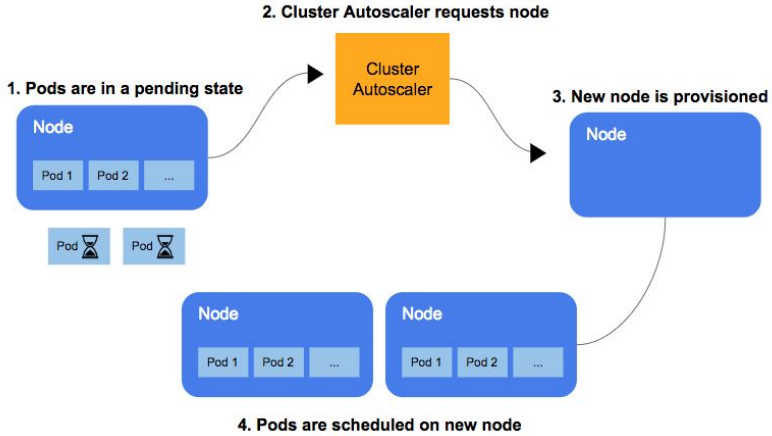
CPU の利用率ベースで Pod 数を調整しつつ OOM が発生しないよう VPA によるメモリの垂直スケールも併せて行う ことが可能に



CA - Cluster Autoscaler

ワークロードの需要に応じて Node pool 内の Node 数を自動で増減させる機能

- Node pool 内の Node 数が不足し Pod のスケジューリングができない場合 Node を追加
- Node の使用率が低く、Node pool 内 Node 数を少なくしてもすべての Pod のスケジューリングが可能な場合 Node を削除

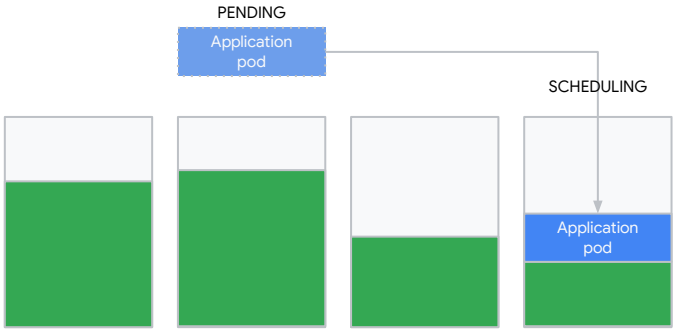


Node

CA - 自動スケーリング プロファイル

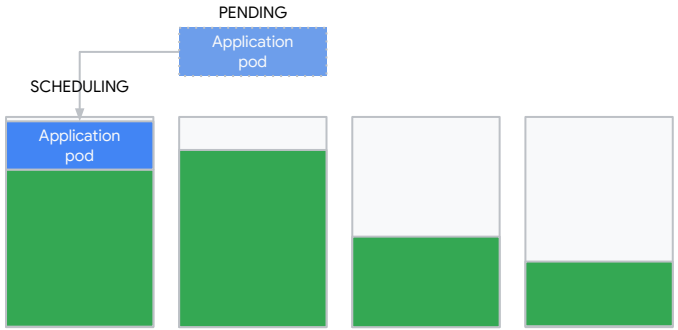
Balanced Profile

デフォルトのプロファイル
余剰リソースと使用率のバランスをとる



Optimize Utilization Profile

クラスタ内で余剰リソースを保持するよりも使用率の最適化を優先させる



Node

CA - 自動スケーリング プロファイル

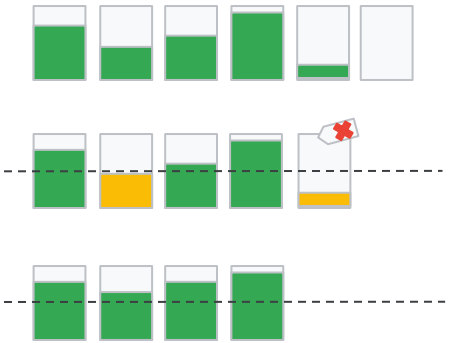
Balanced Profile

可用性の最大化

Runs every: **10m**
Scale down: **nodes < 50%** resource requests

Delete empty nodes and evict pods from nodes flagged as scale down

Delete empty nodes
--
Nothing else to do



Optimize Utilization Profile

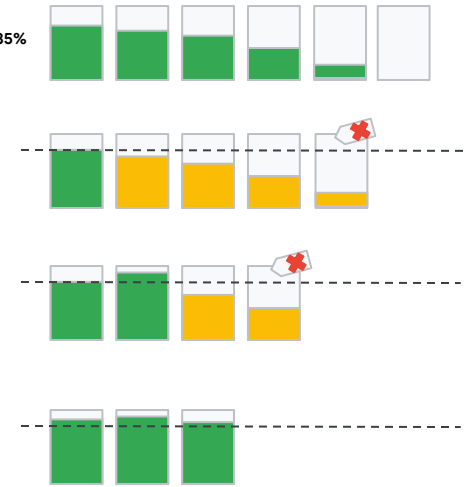
積極的なスケールイン

Runs every: **1m**
Scale down: **nodes < 85%** resource requests

Delete empty nodes and evict pods from nodes flagged as scale down

Delete empty nodes and evict pods from nodes flagged as scale down

Delete empty nodes
--
Nothing else to do



CA - Node がスケールインしないケース

以下のケースでは CA による Node の削除が実行されない

- PodDisruptionBudget (PDB) や Affinity / Anti-Affinity に違反している
- Pod がコントローラによって管理されていない
- Pod でローカルストレージを利用している（GKE 1.21 以下）
- Pod に *"cluster-autoscaler.kubernetes.io/safe-to-evict": "false"* アノテーションが付与されている
- PDB の設定されていないシステム系 Pod (metrics-server や kube-dns など)

期待通りにスケールインしない場合は、 *"cluster-autoscaler.kubernetes.io/safe-to-evict": "true"* アノテーションの付与や、PDB の適切な設定等を検討

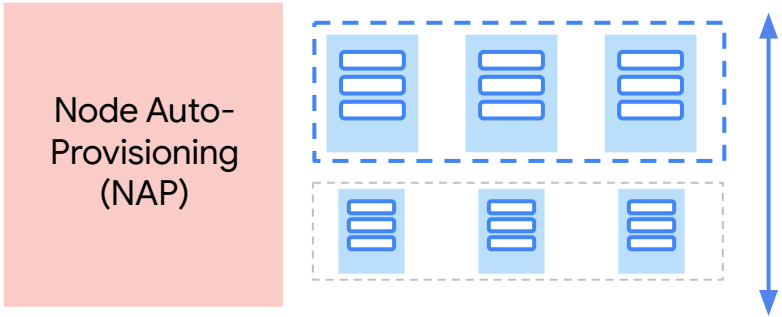
システム系とワークロード系の Pod でノードプールを分けるなど、ワークロードのスケール特性に応じて CA を有効化・無効化することで運用をシンプルにすることも

NAP - Node Auto Provisioning

ワークロードにフィットする Node Pool を自動的に作成・削除する

ノードのサイジングを含めたリソース管理を GKE に任せることができ、またワークロードに合ったサイズのマシンをプロビジョニングするため **インフラリソースを効率的に活用可能**

GKE Autopilot では NAP を利用し Pod に必要な Node を自動的にプロビジョニングしている



Node

ノードのスケールを高速化するために - イメージストリーミング

コンテナイメージのデータをストリーミングし、イメージの　Pull を高速化させる機能

イメージストリーミング機能を活用することで、CA / NAP 等によりノードが新しくプロビジョニングされた場合のイメージの Pull を高速に

Image streaming がない場合

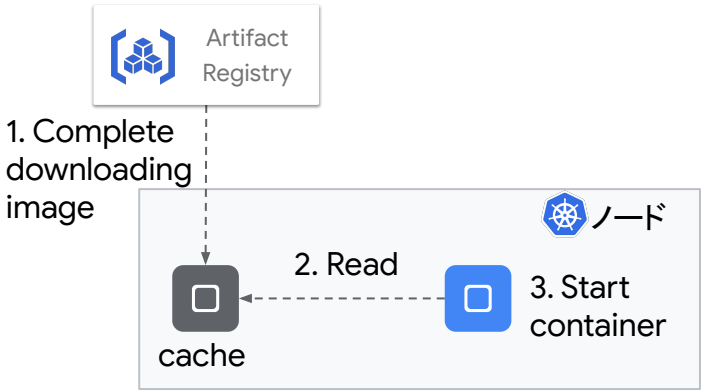
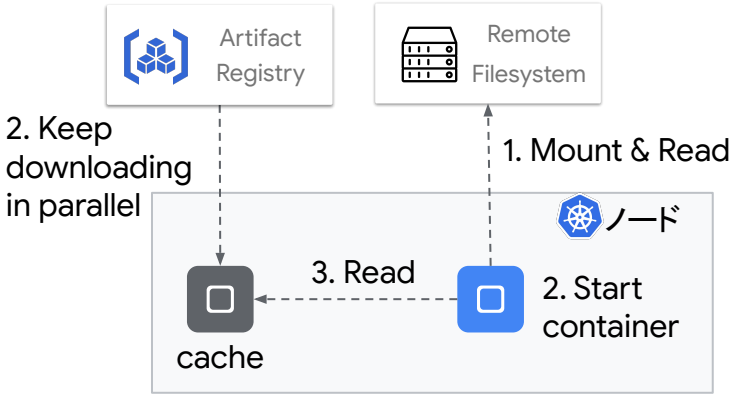


Image streaming がある場合



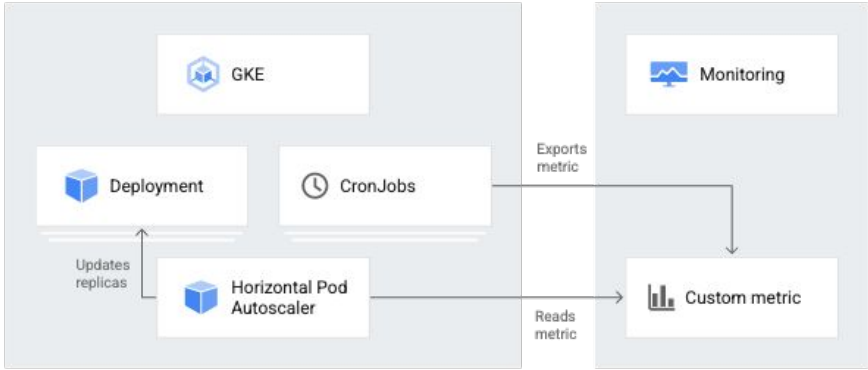
Workload Node

Pod や Node を事前にスケールする

HPA や CA を設定していたとしても急激なトラフィック増に対してスケーリングが間に合わないケースもある

ある程度**トラフィックが増えるタイミングが予測可能な場合**は事前に Pod や Node を増やすことでリソースを効率的に扱うことができる

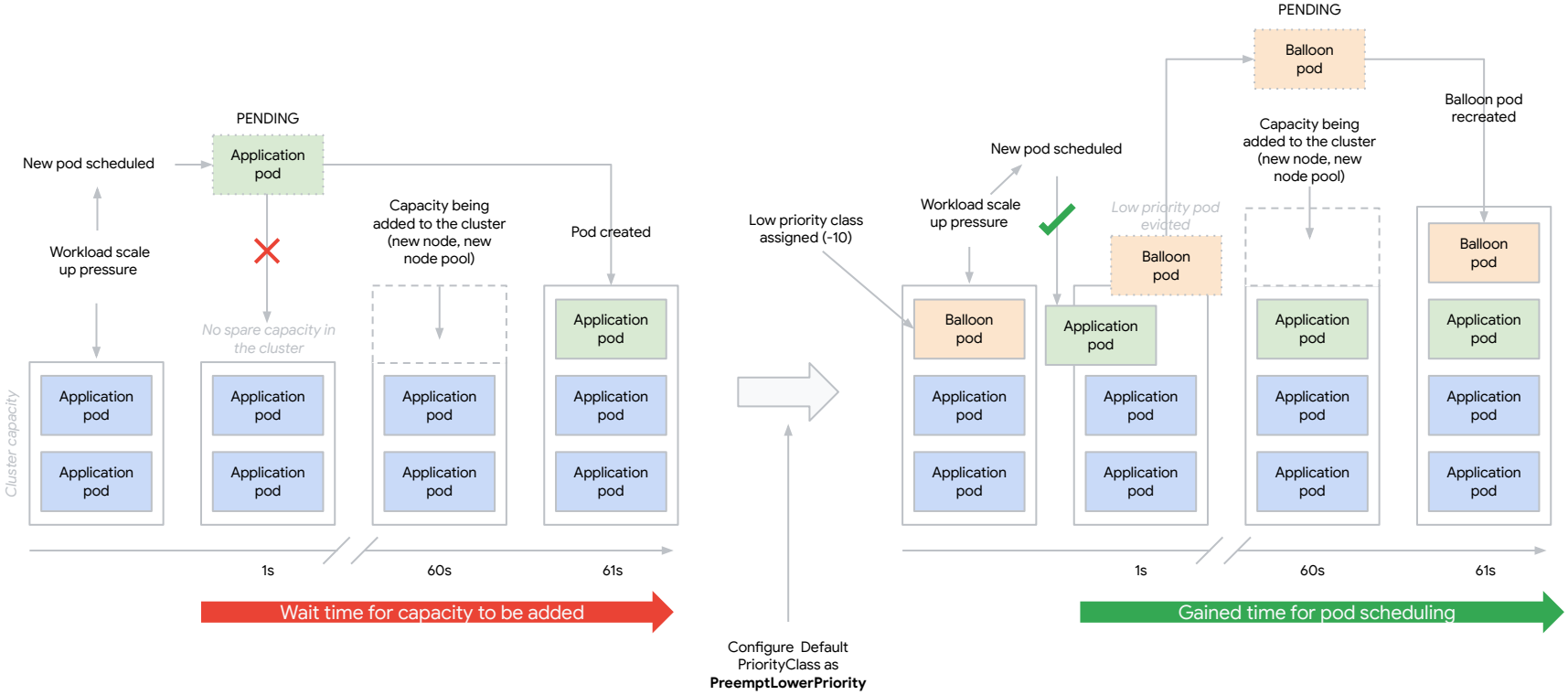
右の例では CronJob を使ってカスタムメトリクスをエクスポートし、HPA ではそのカスタムメトリクスをベースに Pod 数を調整している



オフピーク時に GKE クラスタをスケールダウンしてコストを削減する
https://cloud.google.com/architecture/reducing-costs-by-scaling-down-gke-off-hours

Workload

ノードのスケールを発生させずに Pod をスケールさせる - Balloon Pod の活用



Workload Node

メンテナンス時のワークロードの可用性を保持する

Readiness / liveness probe

kubelet は Probe を使用して、ポッドの実行、クラッシュ時の再起動、修復を行います

Pod disruption budget（PDB, Pod 停止予算）

メンテナンス中でも最低限必要となるレプリカの数を指定し、保護できます。PDB が構成されると、Kubernetes はポリシーに沿ってノードをドレインします

Affinity / Anti-Affinity

Pod の Affinity ルールと Anti-Affinity ルールを使用し、レプリカの配置を制御できます。例えばノード損失による停止は Pod Anti-Affinity で回避します

PreStop hook & terminationGracePeriodSeconds

やむを得ず Pod が停止する場合にも、プロセスがリソースを安全に解放し、重要なデータを残すためにはこれらを使います

03

クラスタのアップグレード / 脆弱性対応

アップグレードや脆弱性対応における課題

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

一般的にKubernetes クラスタのアップグレード作業は **運用負荷が高く**、また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

- 新バージョン・脆弱性情報の検知
- 変更内容や影響範囲の特定
- クラスタのアップグレード（各環境ごとに実施）
- 無影響確認、など

無理なく継続的にアップグレード・セキュリティパッチ適用を行っていくためには、**自動化された仕組み** が求められる

GKE クラスタのアップグレード

GKE では Control Plane 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が　Node については自動もしくは手動でのアップグレードを選択可能（Autopilot は自動のみ）

1. **リリースチャンネルを指定したクラスタ**
    - Control Plane: **自動**
    - Node: **自動**
    - GKE Autopilot はリリースチャンネルに登録される
2. **静的にバージョンを指定したクラスタ**
    - Control Plane: **自動**
    - Node: **自動** or **手動**
        - Node のアップグレードタイミングを自分でコントロールしたい場合は **手動**

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

ノードプールの手動アップグレード方法

ノードプールを手動アップグレードする場合、以下 2 種類からアップグレード戦略を選択する

1. **サージアップグレード**
    - デフォルトのアップグレード戦略
    - ノードを順次新しいバージョンにローリングアップデートする
2. **Blue / Green アップグレード**
    - 新旧 2 つのバージョンのノードを保持したままアップグレードを行う
    - サージアップグレードに比べてロールバックが高速だが、リソース消費が多い（一時的にノード数が 2 倍になる）

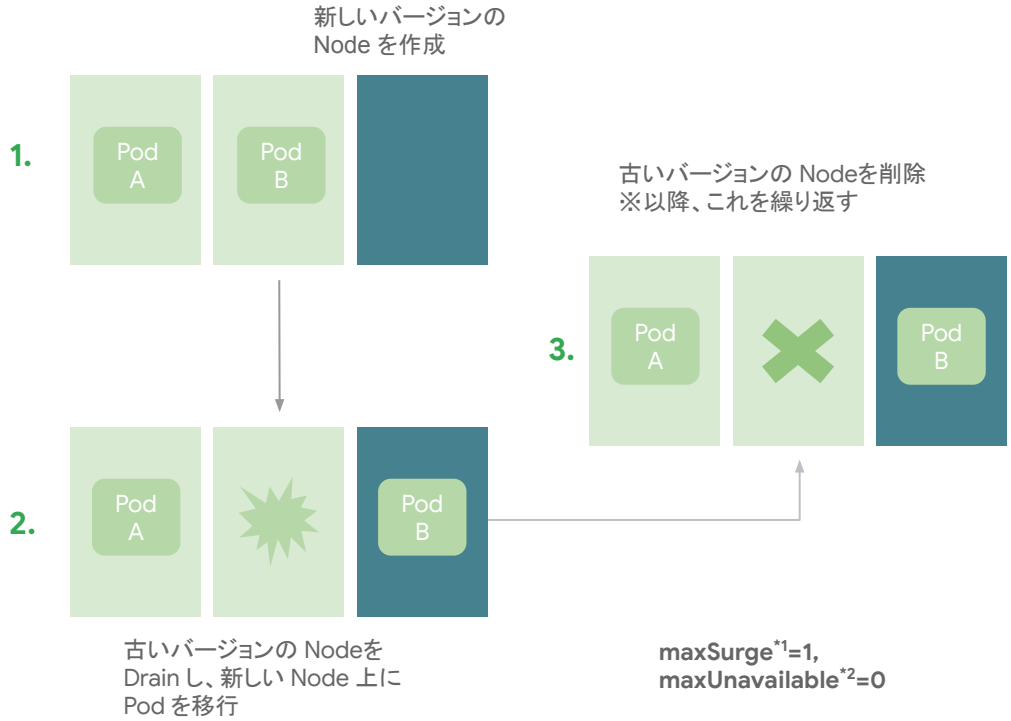
Autopilot 含め、ノードプールを自動アップグレードする場合は **サージアップグレード** が利用される

サージアップグレード

新しいバージョンの Node を追加し、ローリング方式でアップグレードする方式

Blue / Green に比べて、アップグレード時の追加リソースを少なく構成することが可能

max-surge-upgrade[*1] や *max-unavailable-upgrade*[*2] パラメータを設定することにより、アップグレードの同時実行数などを調整することもできる



*1 ... アップグレード中に追加可能な Node 数
*2 ... アップグレード中に使用不可になる Node 数

Blue / Green アップグレード

Node レベル Blue / Green アップグレードの一連の手順を自動的に実行

サージアップグレードに比べて、問題発生時のロールバックを迅速に行うことができる

一時的にノード数が 2 倍になるため、Quota 含め充分なリソースが確保できているか事前に確認が必要

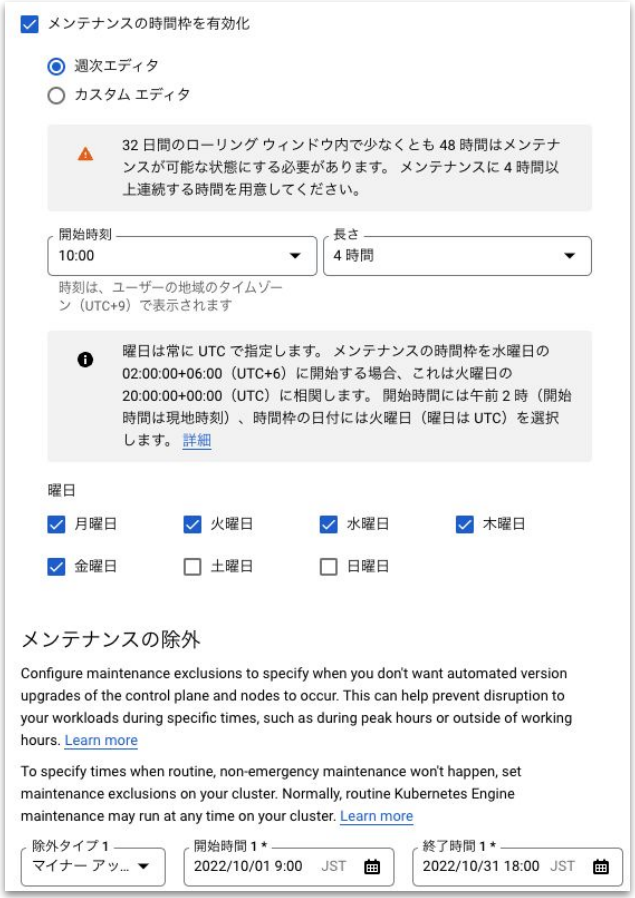
*1 デフォルト 1 時間 (3,600 秒)、最大 7 日間 (604,800 秒)

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

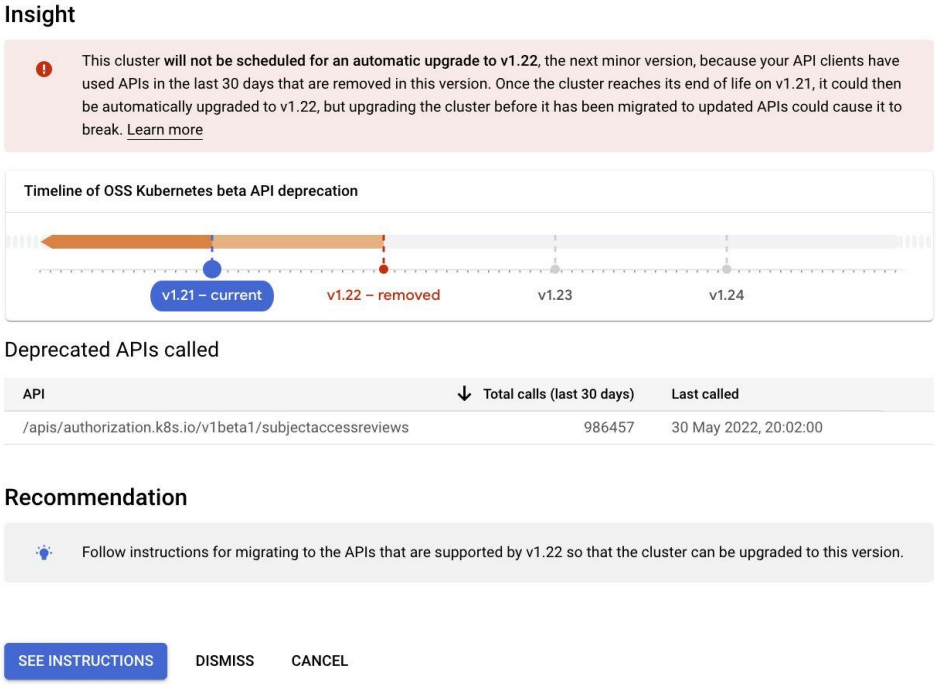
***1 マイナーバージョンの EOL を超過することはできない**

***2 リリースチャンネルを利用しているクラスタが対象**

Deprecated な API の自動検出 Preview

今後のマイナー バージョンで削除される Kubernetes 機能または API 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 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（手動でのアップグレードは可能）



クラスタのアップグレード / 脆弱性情報の通知

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）

New master version "1.19.9-gke.1400" is available for upgrade in the RAPID channel.

```
cluster_location: asia-northeast2
cluster_name: rapid-autopilot-an2
payload: {"version":"1.19.9-gke.1400", "resourceType":"MASTER", "releaseChannel":{"channel":"RAPID"}}
project_id: 605899591260
type_url: type.googleapis.com/google.container.v1beta1.UpgradeAvailableEvent
```

New master version "1.18.16-gke.2100" is available for upgrade in the REGULAR channel.

```
cluster_location: asia-northeast2
cluster_name: regular-autopilot-an2
payload: {"version":"1.18.16-gke.2100", "resourceType":"MASTER", "releaseChannel":{"channel":"REGULAR"}}
project_id: 605899591260
type_url: type.googleapis.com/google.container.v1beta1.UpgradeAvailableEvent
```

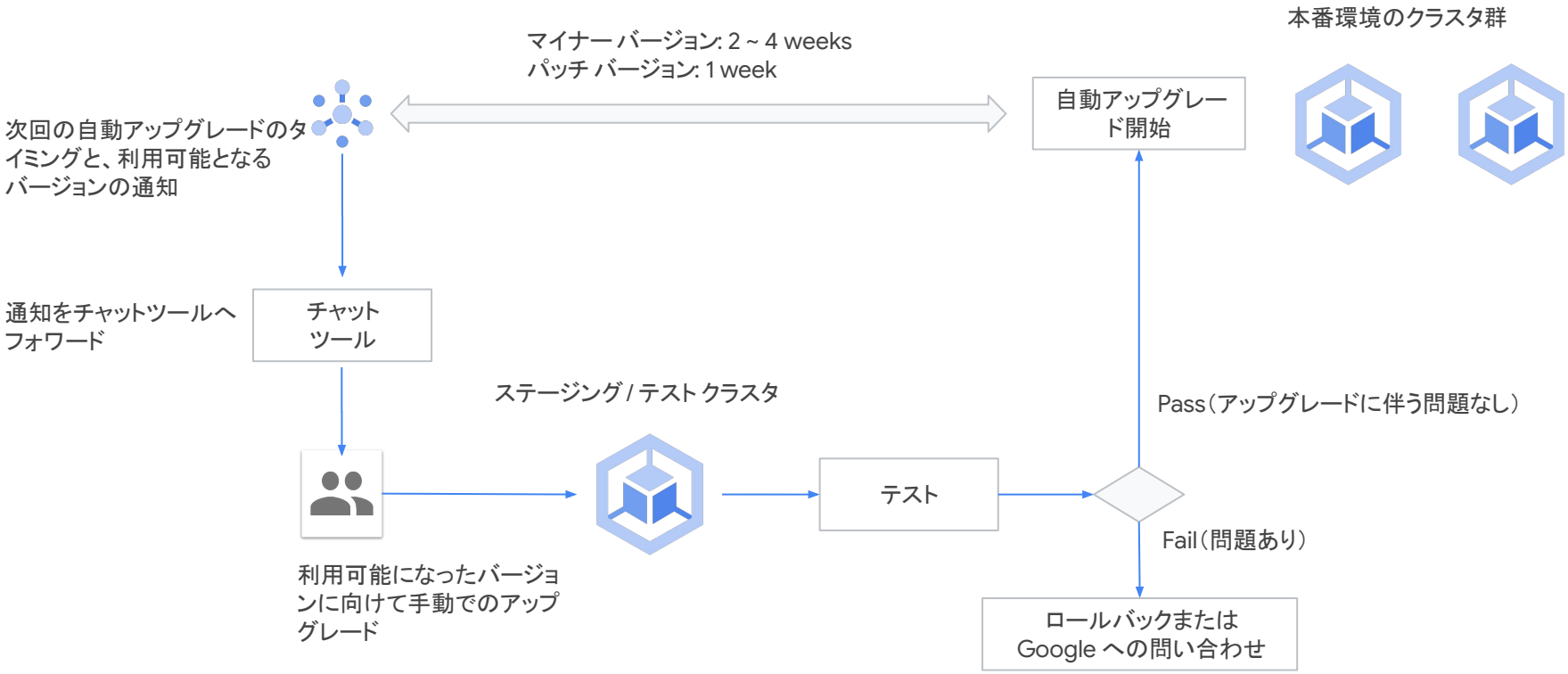
New node version "1.18.17-gke.1200" is available for upgrade.

```
cluster_location: asia-northeast2
cluster_name: static-standard-an2
payload: {"version":"1.18.17-gke.1200", "resourceType":"NODE_POOL", "resource":"projects/kzs-sandbox/locati
project_id: 605899591260
type_url: type.googleapis.com/google.container.v1beta1.UpgradeAvailableEvent
```

New master version "1.18.16-gke.2100" is available for upgrade in the STABLE channel.

```
cluster_location: asia-northeast2
cluster_name: stable-standard-an2
payload: {"version":"1.18.16-gke.2100", "resourceType":"MASTER", "releaseChannel":{"channel":"STABLE"}}
project_id: 605899591260
type_url: type.googleapis.com/google.container.v1beta1.UpgradeAvailableEvent
```

アップグレードの通知を使ったテスト ワークフロー例



04

エコシステムの運用

Kubernetes エコシステム運用の課題

Prometheus や Istio など Kuberentes エコシステムを利用していると、それらコンポーネントの
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**などを継続的に実施する必要がある

Google Cloud では **OSS をベースにしたマネージド サービス**を提供しており、
マネージドサービスを活用することにより Day 2 オペレーションの手間を省くことができ
運用負荷の軽減 が期待できる

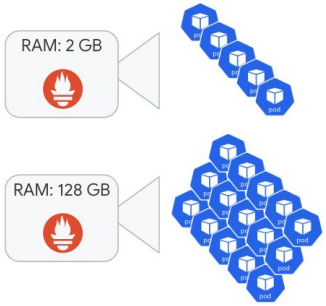
Kubernetes → Google Kubernetes Engine

Prometheus → Managed Service for Prometheus

Istio → Anthos Service Mesh

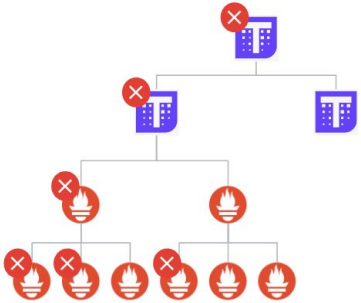
Prometheus 運用における課題の例

水平スケール



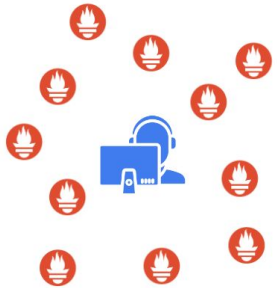
- 収集する対象や保管しているメトリクス量が増えるに従い、Promtheus の RAM / Storage の**スケールが必要**となる
- メトリクスの長期保管用にリモートストレージを利用するケースもあるが、**リモートストレージ側のスケーラビリティ**についても考慮する必要が出てくる

グローバル展開



- 複数の Kubernetes クラスタのメトリクスをグローバルレベルに集約するには Prometheus の Federation 機能を利用可能だが構成が複雑になりやすく、また **パフォーマンス** や**スケーラビリティ**、**信頼性**の面での課題がある

メンテナンストイル



- Prometheus や 3rd Party のリモートストレージ ソリューション自体のリソースの監視や Helm chart / マニフェスト等の **メンテナンス負荷** がかかる

Prometheus の管理は運用上

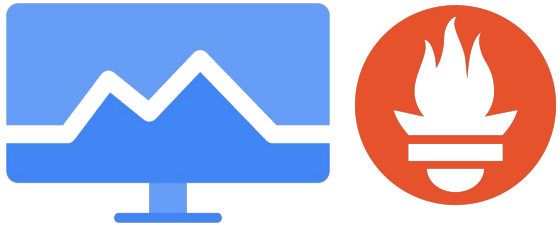
負荷が「かかり続ける」傾向にあり、

その大変さに見合うだけの感謝はされにくい・・

Google 規模の負荷と経験を活かした

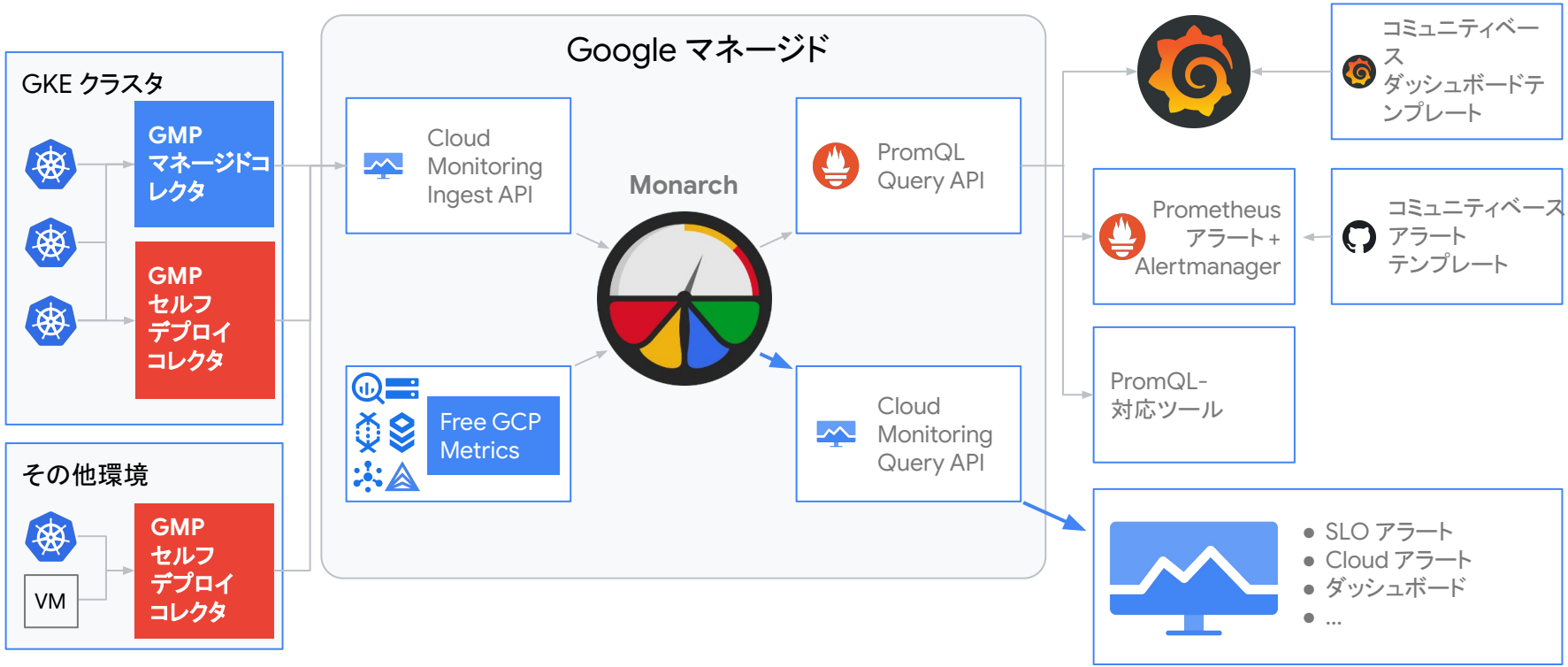
Prometheus と互換性をもつ Google Cloud マネージド サービス

Google Cloud Managed Service for Prometheus (GMP)



Google Cloud Managed Service for Prometheus | オペレーションスイート
https://cloud.google.com/stackdriver/docs/managed-prometheus?hl=ja

Managed Service for Prometheus (GMP) アーキテクチャ



Managed Service for Prometheus (GMP) の特長

運用負荷の低減

フルマネージドなデータストア / コレクタを活用することで**運用負荷を低減**

貴重なエンジニアリング リソースをコア業務に集中

複数環境の統合監視

複数クラスタ / プロジェクトを一元的にモニタリング

オンプレミス・他社クラウド上のワークロードからもメトリクスを収集可能

既存監視ツールの有効活用

Grafana / Alertmanager 等**既存の Prometheus エコシステムを活用可能**

既存の運用は変えずにストレージのみマネージドサービス側に移行

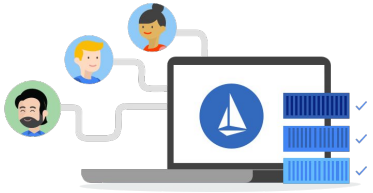
Anthos Service Mesh (ASM)

Google が提供するフルマネージドの
サービスメッシュ

Istio コントロールプレーン (istiod) や認証局を
Google が管理する

Cloud Monitoring / Logging とネイティブに連携

可観測性



トラフィックコント
ロール

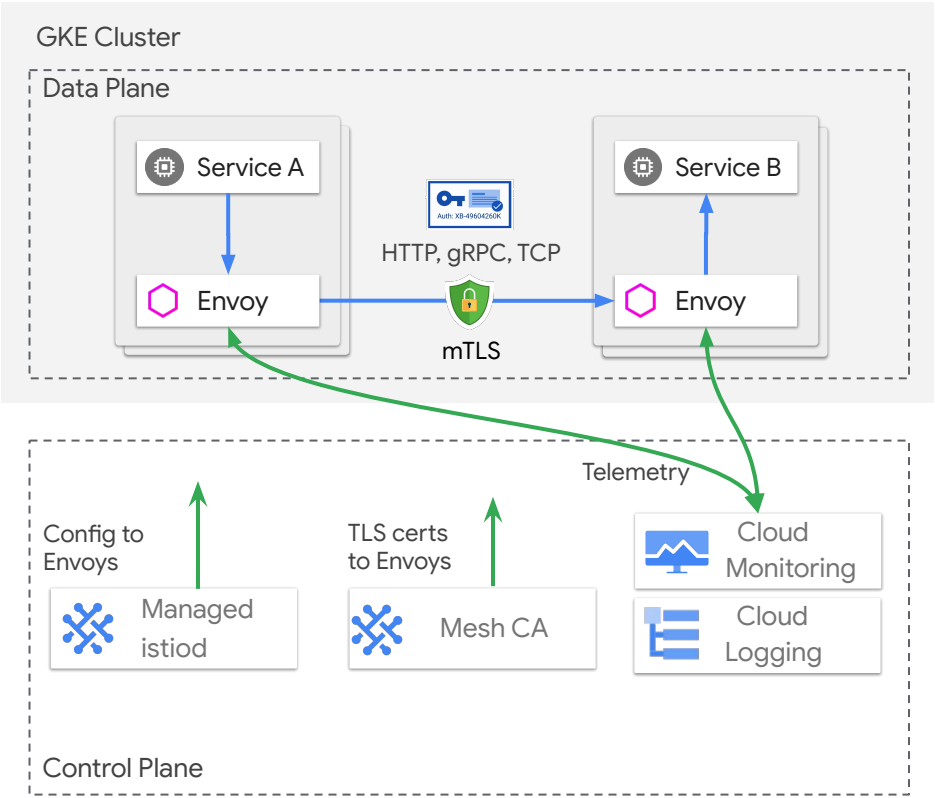
セキュリティ

ASM のアーキテクチャ - Managed Control Plane (MCP)

OSS 版 Istio と違い、コントロール プレーンがユーザー クラスタから分離され、Google 管理となる

- Istiod がマネージドに
- CA 機能は Mesh CA / CAS に
- メトリクス監視、ロギング監視は Cloud Logging / Monitoring に

Istiod や認証局 (CA) の**セキュリティ**や**スケーラビリティ**、**アップグレード作業**等を気にせず利用できる



ASM - 自動アップグレード

MCP は以下のリリースチャンネルに従って、自動的にアップグレードされる

利用するリリースチャンネルは Namespace に付与する istio.io/rev ラベルの値でコントロールする

チャンネル (istio.io/rev ラベルの値)	利用可能なバージョン	詳細
Rapid (asm-managed-rapid)	最新のASM バージョン	アップグレード頻度が高。最新の機能をなるべく早く 利用したい場合、テスト環境での利用を推奨
Regular (asm-managed)	Rapid を経た ASM バージョン	アップグレード頻度が中。安定性と最新機能のバランスを取る 場合に推奨
Stable (asm-managed-stable)	Regular を経た ASM バージョン	アップグレード頻度が低。安定性を最優先にする場合に推奨

Managed Data Plane (MDP) を有効にすることにより、Control Plane のアップグレードにあわせて

サイドカープロキシを自動的にリスタートさせアップグレードすることも可能

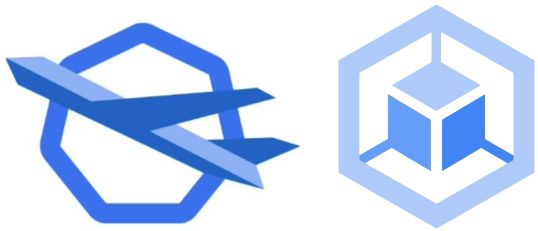
MDP のアップグレード通知 を設定することで、アップグレードのタイミングを事前に知ることができる

05

まとめ

セッションのまとめの前に...
簡単に**運用レスなクラスタ**を手に入れたいあなたにお勧めなのが

GKE Autopilot



Best Practices

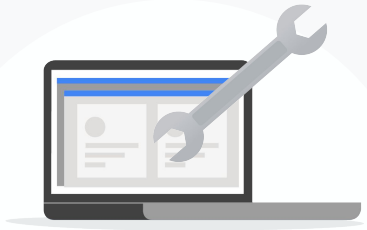
ワークロードやセキュリティに関するベストプラクティスが適用済み

Dynamic Workload

CA / NAP を使ったリソースの動的なプロビジョニング

No infra 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



まとめ - GKE を使って可能な限り運用負荷を軽減するために

- GKE の **自動スケール機能** や **自動アップグレード機能** を活用し、クラスタリソースの管理やアップグレードの負荷を軽減
- Prometheus や Istio 等の Kubernetes エコシステムを **マネージドサービスに置き換える** ことにより、運用をよりシンプルに
- **GKE Autopilot** は各種自動化機能やベストプラクティスが組み込みで適用されているので、 **これから Kubernetes を使い始める方にもおすすめ**
- Kubernetes やそのエコシステムの良さは享受しつつ、運用の Toil は Google Cloud に任せることで **貴重なエンジニアリングリソースをコア業務に集中** させることが可能に

Thank you.

Google Cloud