



GKE ベストプラクティス

March 2021

自己紹介

Mourad El Azhari (ムラディ)
ゲーミング・カスタマー・エンジニア



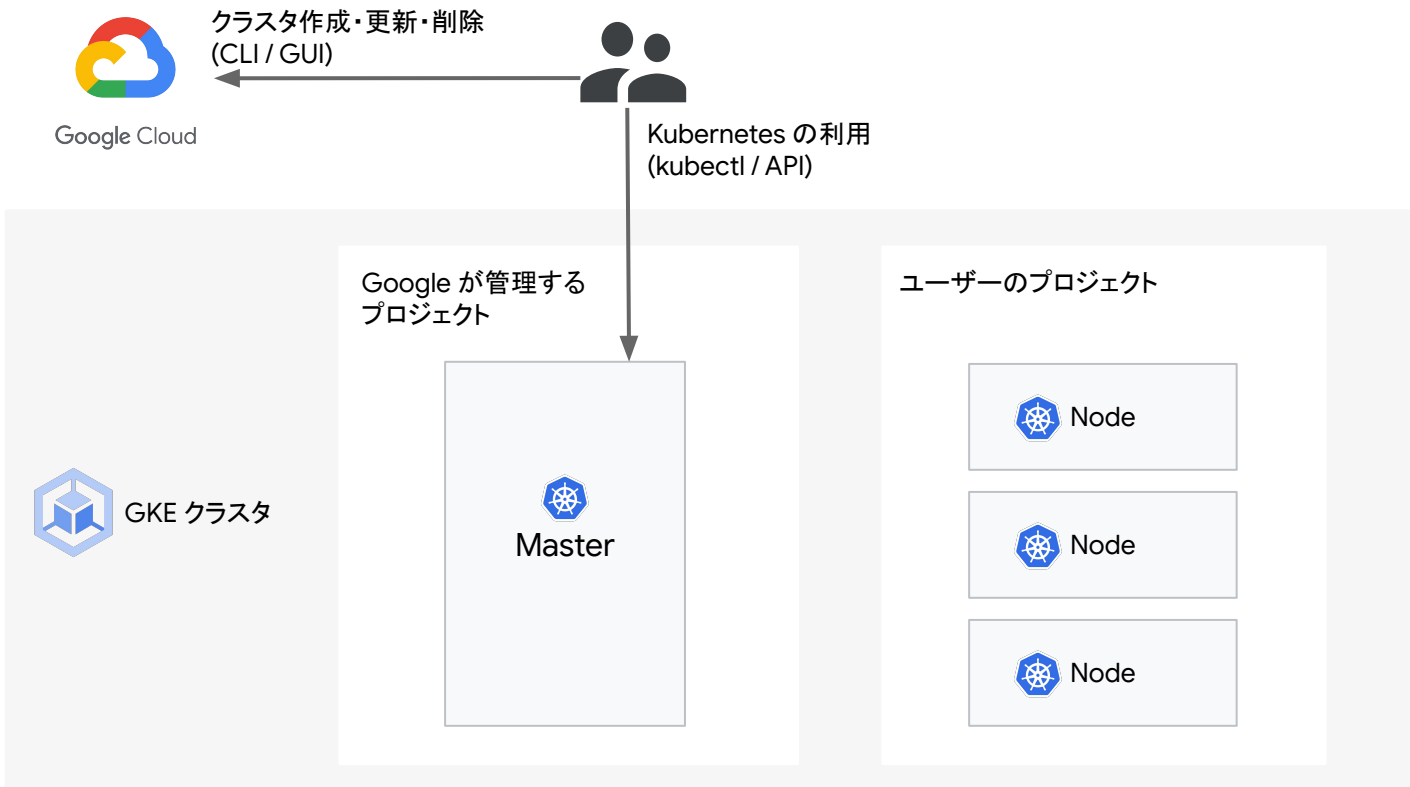
- 2005年3月にモロッコから留学生として来日
- 2013年4月に外資系大手ITベンダーに入社、デベロッパーのサポートエンジニア
- 2019年2月にカスタマーエンジニアとしてGoogle Cloudに入社

自慢話: 4.5ヶ国語が話せる

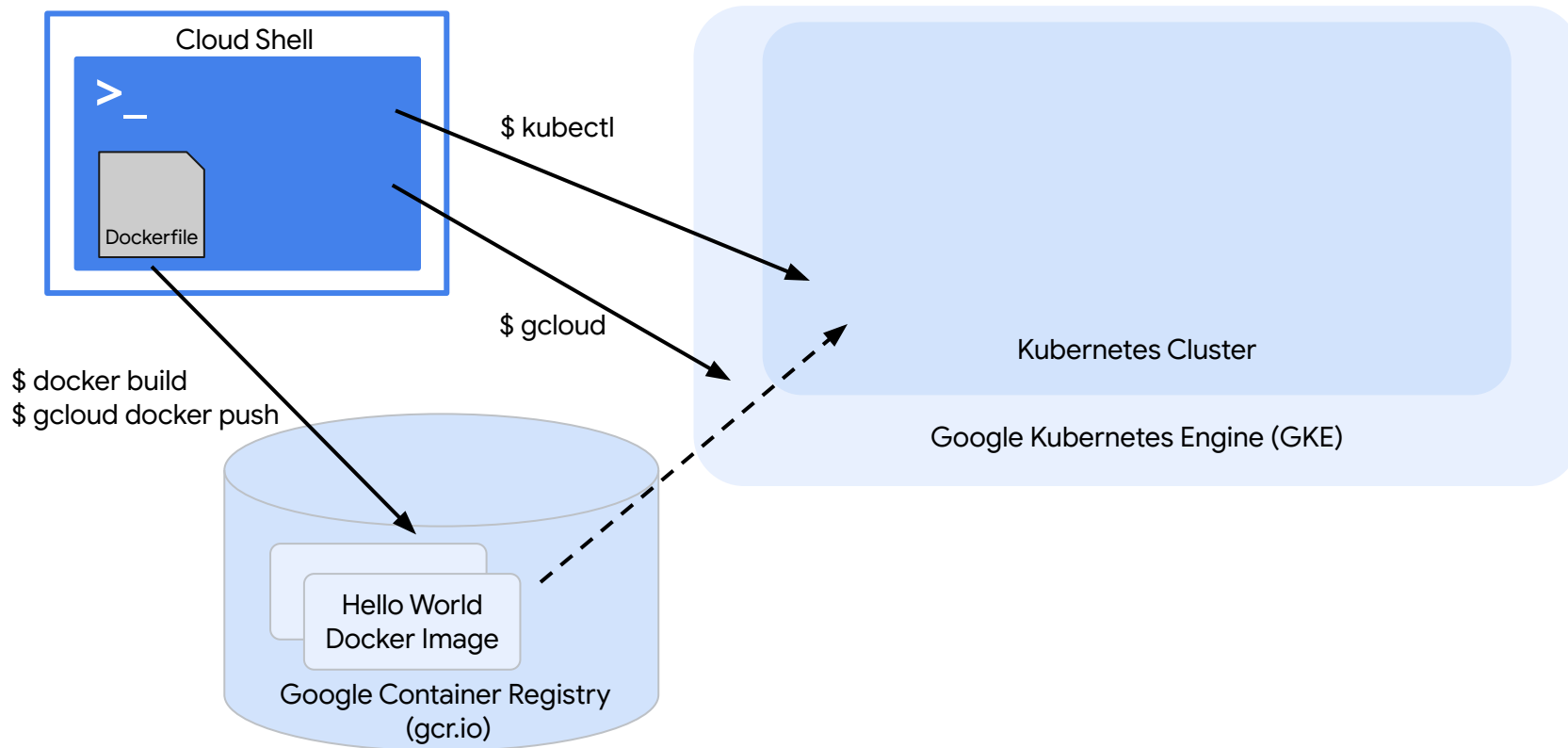
アラビア語・フランス語・英語・日本語
& 庄内弁

GKE のおさらい

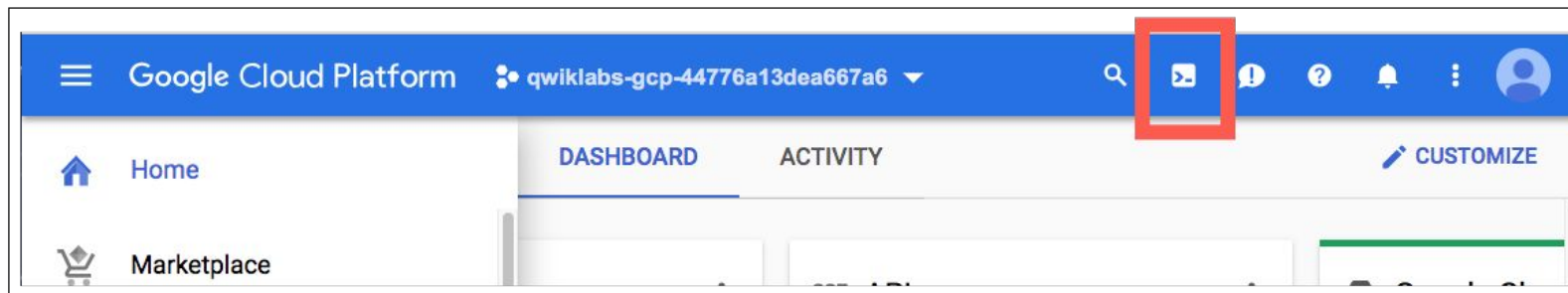
GKE の基本的なアーキテクチャ



コマンド整理



Cloud Shell



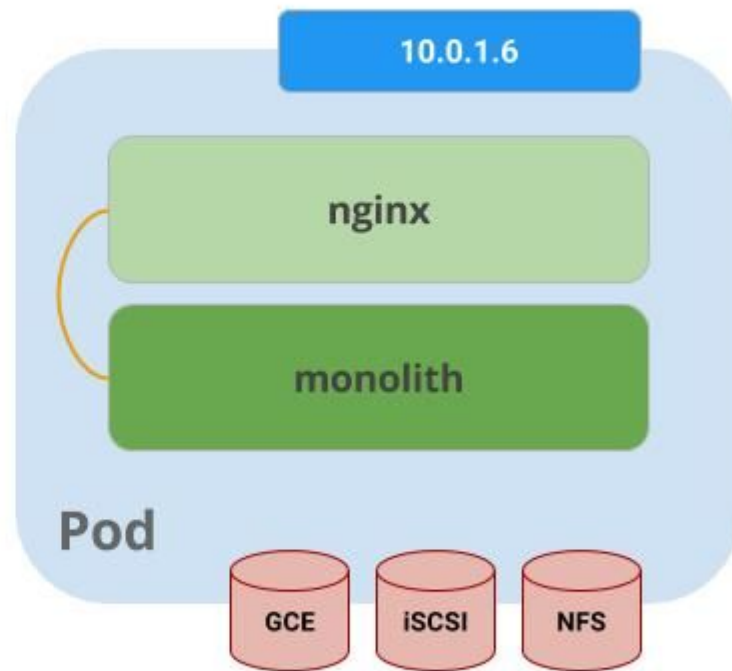
GKE クラスタの作成

- 次の gcloud コマンドを実行

```
$ gcloud container clusters create gke-demo \  
  --zone "asia-northeast1-c" \  
  --machine-type "n1-standard-2" \  
  --num-nodes "3"
```

Pod

- Kubernetes の中核
- 1つ以上のコンテナの集まり
- Podごとのボリューム
- Podごとに1つのIPアドレス



Pod を作成するマニフェスト

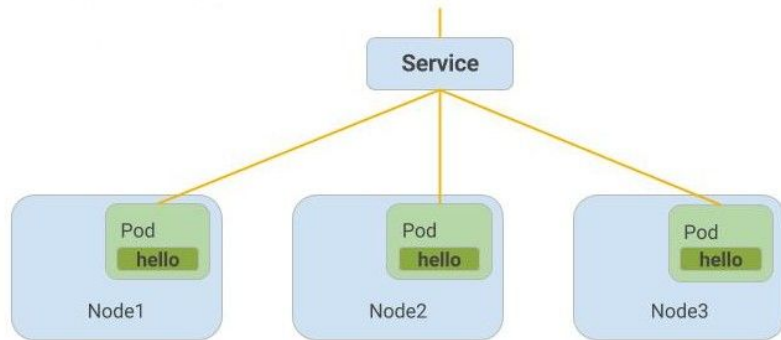
- Pod は1つのコンテナ(hello-app)で構成されている
- コンテナのイメージ
- トラフィック用にポート 8080 を開いている

```
apiVersion: v1
kind: Pod
metadata:
  name: helloweb
  labels:
    app: hello
spec:
  containers:
  - name: hello-app
    image: gcr.io/google-samples/hello-app:1.0
    ports:
      - containerPort: 8080
```

Service

Pod のための永続的なエンドポイント

- ラベルを使用して Pod を選択
- Service の種類
 - ClusterIP
 - NodePort
 - LoadBalancer



Service を作成するマニフェスト

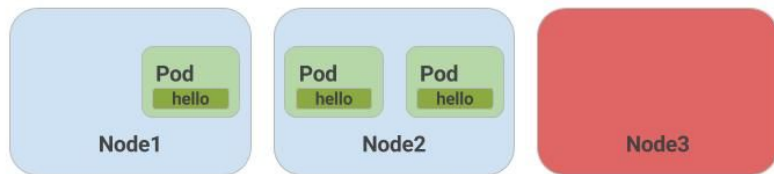
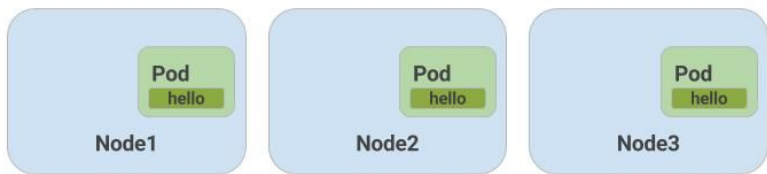
- ラベル「app=hello」を持つすべてのポッドを自動的に検出して公開するセレクトが存在する
- 外部トラフィックの受け付けにロードバランサーを使用している
- 外部トラフィックをロードバランサーのポート 80 で受け付け、Pod のポート(8080)に転送している

```
kind: Service
apiVersion: v1
metadata:
  name: "helloweb"
spec:
  selector:
    app: "hello"
  ports:
    - protocol: "TCP"
      port: 80
      targetPort: 8080
  type: LoadBalancer
```

Deployment

Deployment は実行中の Pod 数と構成ファイル定義された Pod 数を等しくする宣言的な方法

- 例)
 - app: hello
 - replicas: 3



Node3 が停止

→Deployment によって新しい Pod が
Node2で起動された

Deployment を作成するマニフェスト

- Deployment はレプリカ (Pod) を1つ作成している
- template 以下に作成する Pod の情報を持つ

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
        - name: hello-app
          image: gcr.io/google-samples/hello-app:1.0
          ports:
            - containerPort: 8080
```

GKE のベストプラクティス

GKE はコマンド一発で起動可能！
本番用クラスターは・・・？

```
$ gcloud container clusters create [CLUSTER_NAME] \  
  --zone "asia-northeast-1"
```

？

本セッションのゴールと前提条件

ゴール

GKE を本格的に利用する上で役に立つ知識やポイントを理解し、プロダクション レディな GKE クラスターの構築・運用を実現するイメージを持って頂くことをゴールにします。

アジェンダ

1. クラスタの可用性の選択
2. GKE のネットワークモード
3. プライベートクラスタ
4. アクセス管理
5. クラスタのバージョン体系
6. クラスタのバージョンアップ

クラスタの可用性の選択

シングルゾーン クラスタ、マルチゾーン クラスタ、リージョン クラスタ


シングルゾーン クラスタ

- 初期設定で選ばれるクラスタ
- Master と Node は同じゾーン
- Master の SLA
 - 99.5 % (stable release)
- Master バージョンアップ
 - Master 停止



シングルゾーンクラスタ

ゾーン A

 Master

 Node

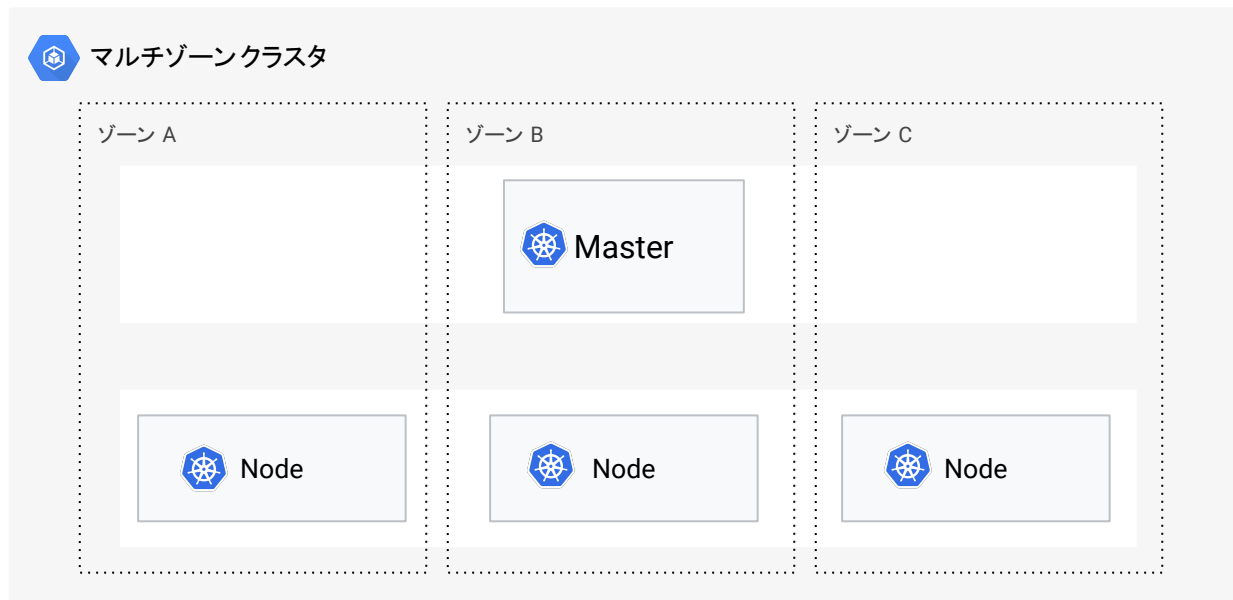
 Node

 Node

```
$ gcloud container clusters create [CLUSTER_NAME] \  
  --zone compute-zone
```

マルチゾーン クラスタ

- Master: 1つのゾーン
- Node: 複数ゾーンで冗長化
 - ゾーン指定可能
- Master の SLA
 - 99.5 % (stable release)
- Master バージョンアップ
 - Master 停止



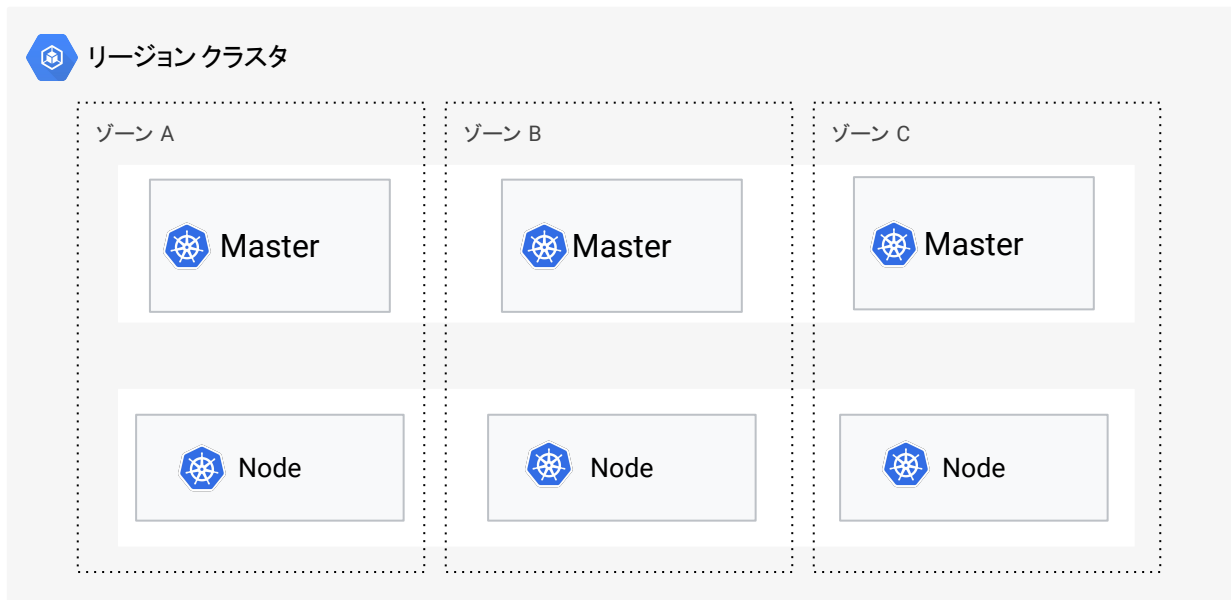
```
$ gcloud container clusters create [CLUSTER_NAME] \  
  --zone compute-zone \  
  --node-locations compute-zone,compute-zone,[...]
```

リージョン クラスタ

- Master: 複数ゾーンで冗長化
- Node: 複数ゾーンで冗長化
 - ゾーン指定可能
- Master の SLA
 - 99.95 % (stable release)
- Master バージョンアップ
 - 1 台ずつアップグレード

マスタの可用性を重視するなら

リージョン クラスタを選択



```
$ gcloud container clusters create [CLUSTER_NAME] \  
  --region compute-region
```

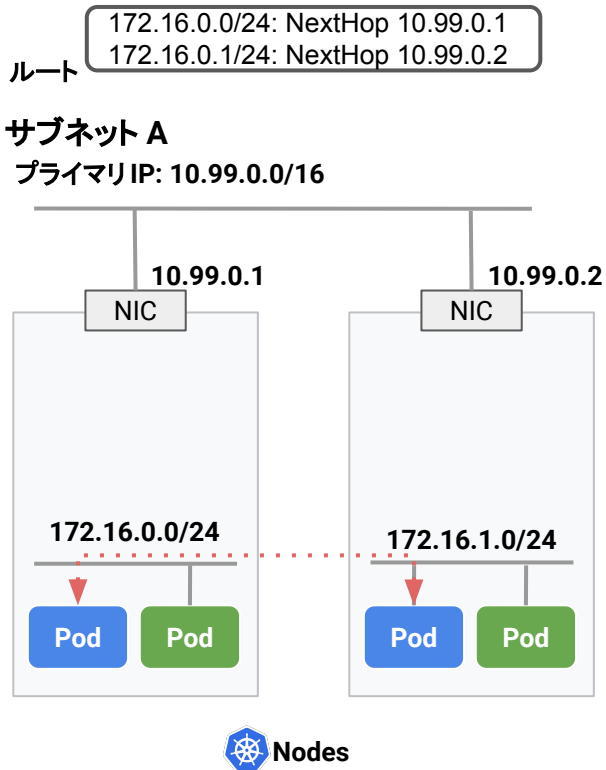
GKE のネットワークモード

ルートベース クラスタ、VPC ネイティブ クラスタ

ルートベース クラスタ

- クラスタ内部 (Service と Pod) のネットワークが VPC のネットワークから完全に独立
- クラスタ内部の IP アドレスは GCP 管理外
- 留意点
 - カスタム ルートの割当上限
 - クラスタ作成時のみ指定可能

```
$ gcloud container clusters create [CLUSTER_NAME]  
--no-enable-ip-alias
```



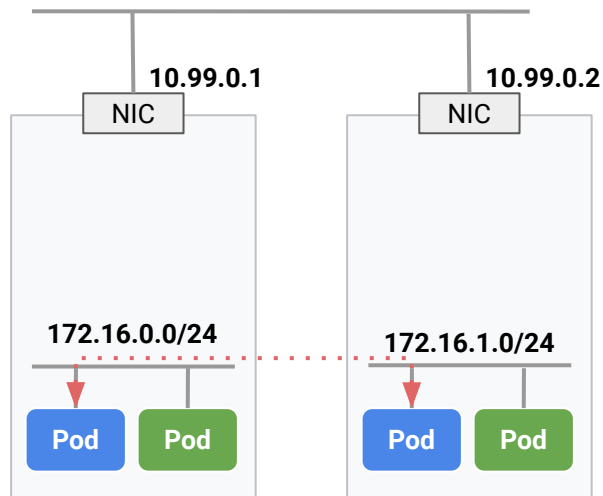
VPC ネイティブ クラスタ (推奨)

- Node が接続するサブネットのセカンダリ IP アドレスをクラスタ内部 (Service と Pod) で利用
- クラスタ内部のネットワークを GCP が管理
 - VPC 内でネイティブにルーティング
- 利点
 - GCP の他リソースとの柔軟な連携
 - 共有 VPC の使用
 - コンテナ ネイティブ負荷分散の利用 (後述)
- 留意点
 - クラスタ作成時のみ指定可能

サブネット A

プライマリ IP: 10.99.0.0/16

セカンダリ IP: 172.16.0.0/20



“default” VPC は使わないほうがいい？

Kubernetes は IP を沢山使用する

- Node, Pod, Service

“default” VPC の留意点

- 自動的に各ゾーンに /20 の Subnet を作成
- 他の Project の VPC や自社ネットワークと CIDR が重複する可能性

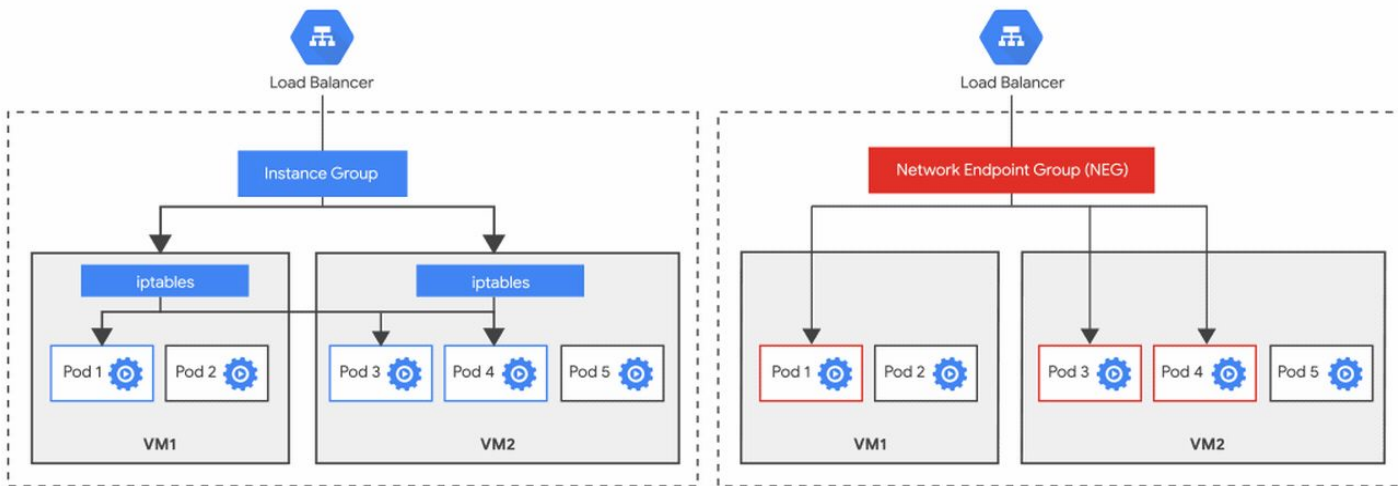
Tips

- カスタム VPC, Subnet を作成
- 1 Node あたりの Pod 最大数 (default: 110) を調整
 - IP Alias を節約 (例: 8 pods -> 16 IPs -> /28 block)
- [計算ツール](#)を活用

default	18	Auto ▼
us-central1	default	10.128.0.0/20,
europe-west1	default	10.132.0.0/20
us-west1	default	10.138.0.0/20
asia-east1	default	10.140.0.0/20
us-east1	default	10.142.0.0/20
asia-northeast1	default	10.146.0.0/20

```
$ gcloud container clusters create [CLUSTER_NAME] \  
  --region=region \  
  --enable-ip-alias \  
  --subnetwork=subnet-name \  
  --cluster-secondary-range-name=secondary-range-pods \  
  --services-secondary-range-name=secondary-range-services
```

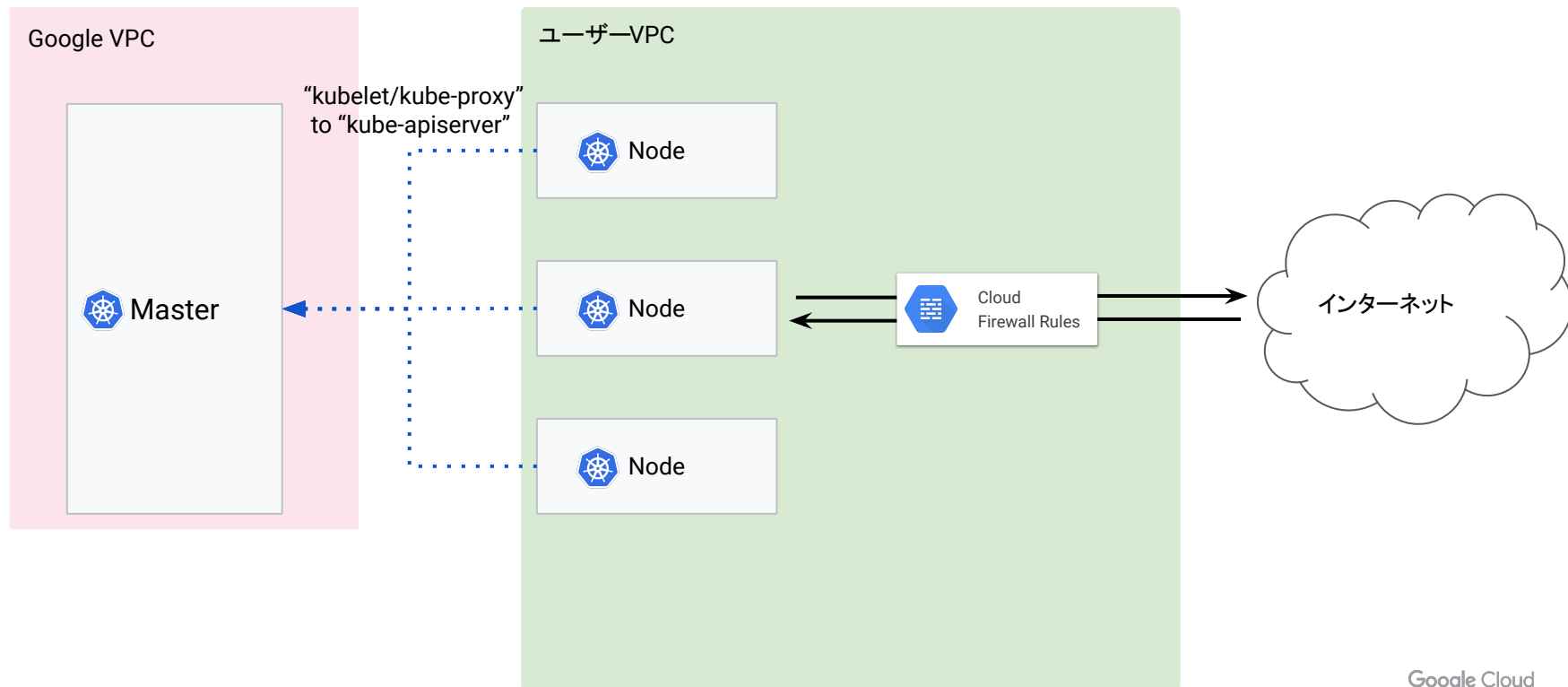
GKE Ingress から Pod へのトラフィック効率を向上させる コンテナ ネイティブ負荷分散



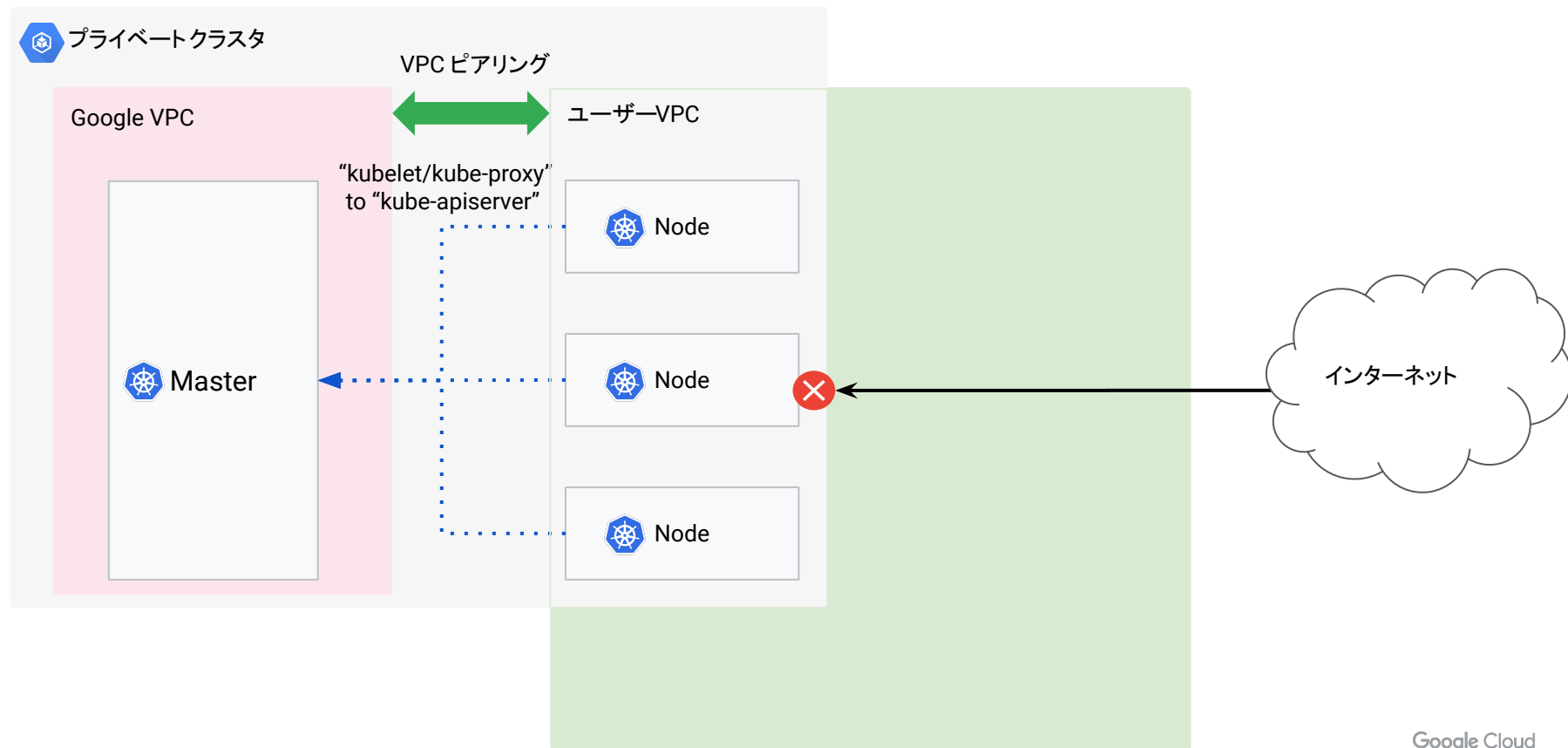
```
kind: Service
metadata:
  annotations: cloud.google.com/load-balancer-neg: "true"
```

プライベート クラスタ

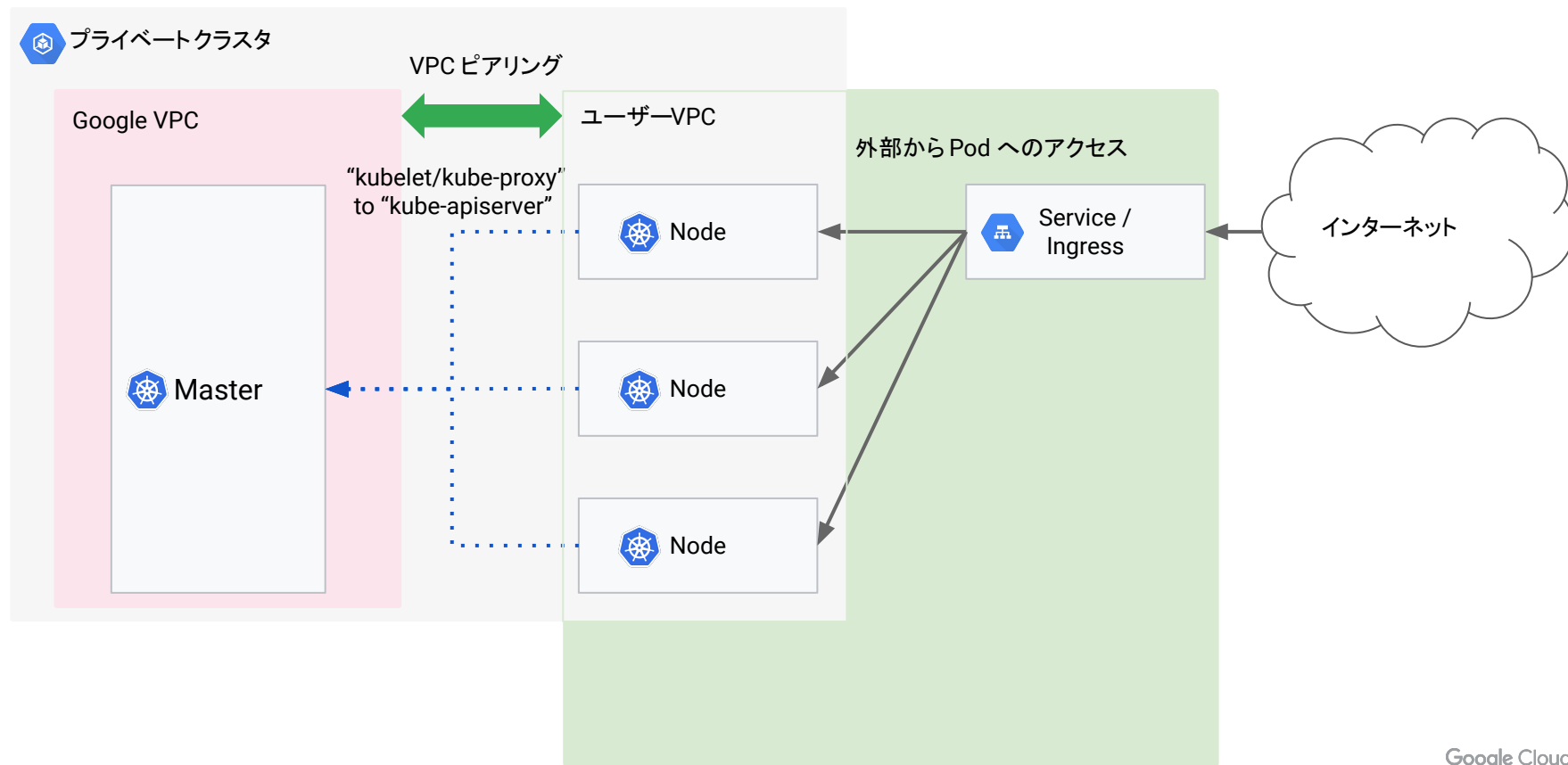
デフォルトの Node へのアクセス



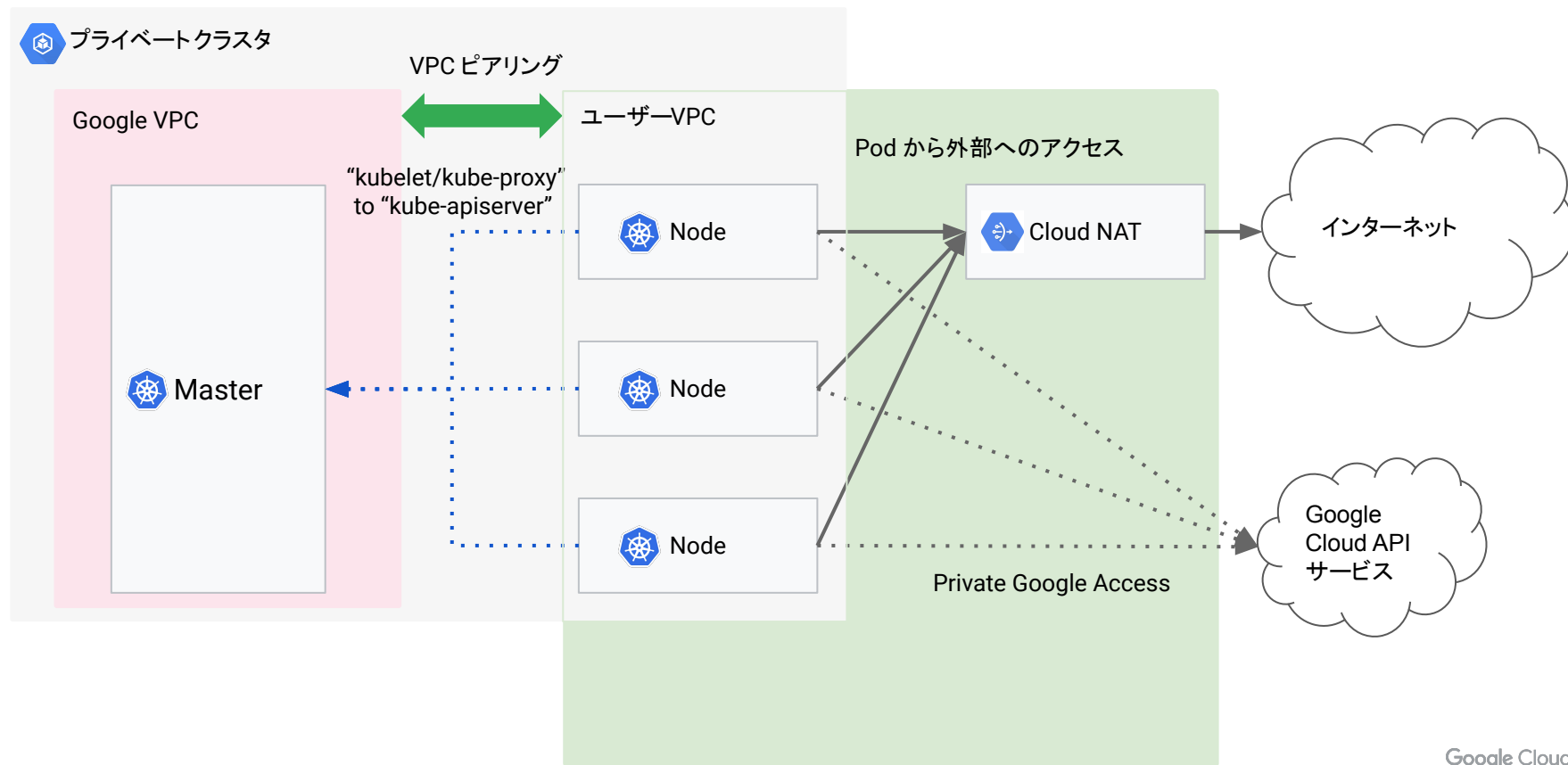
プライベート クラスタとは



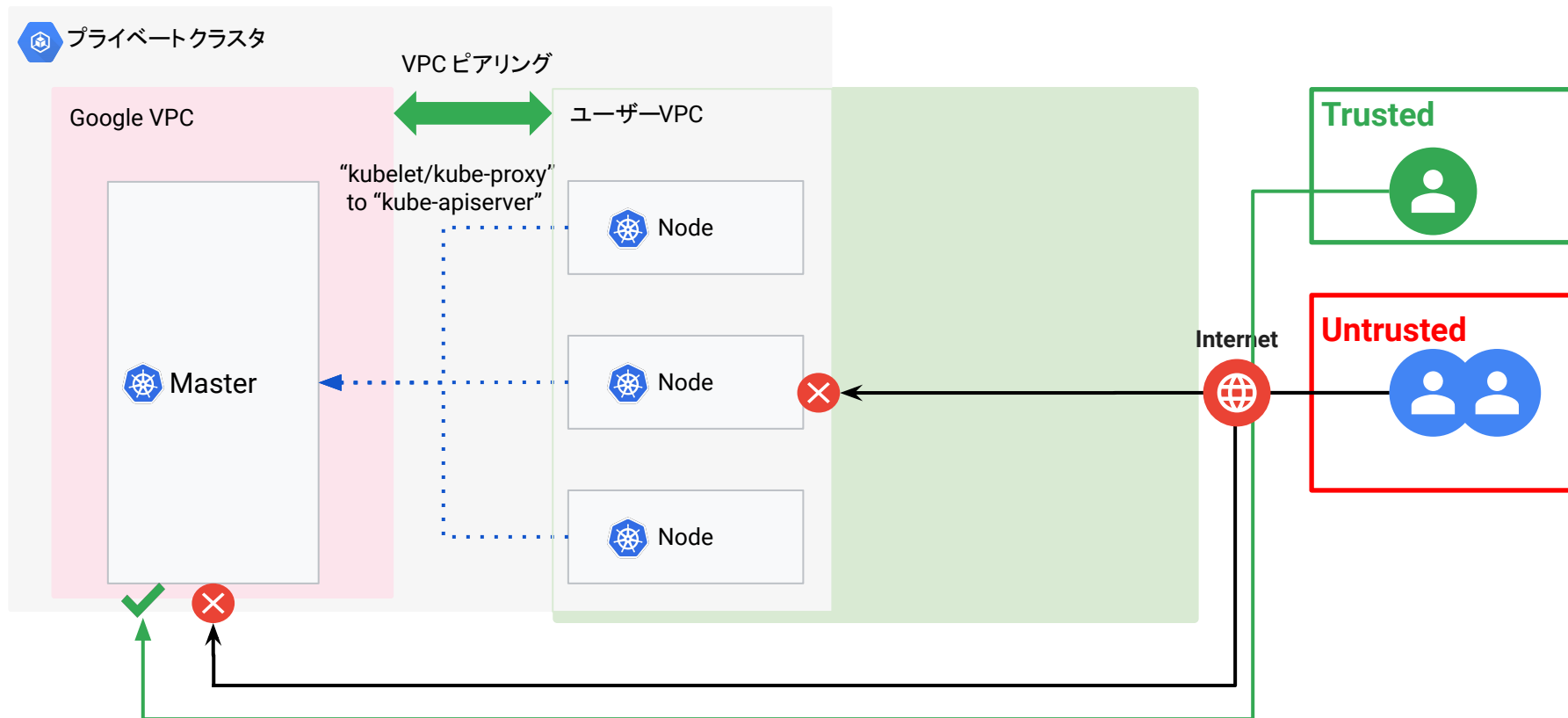
プライベート クラスタ - Inbound Traffic



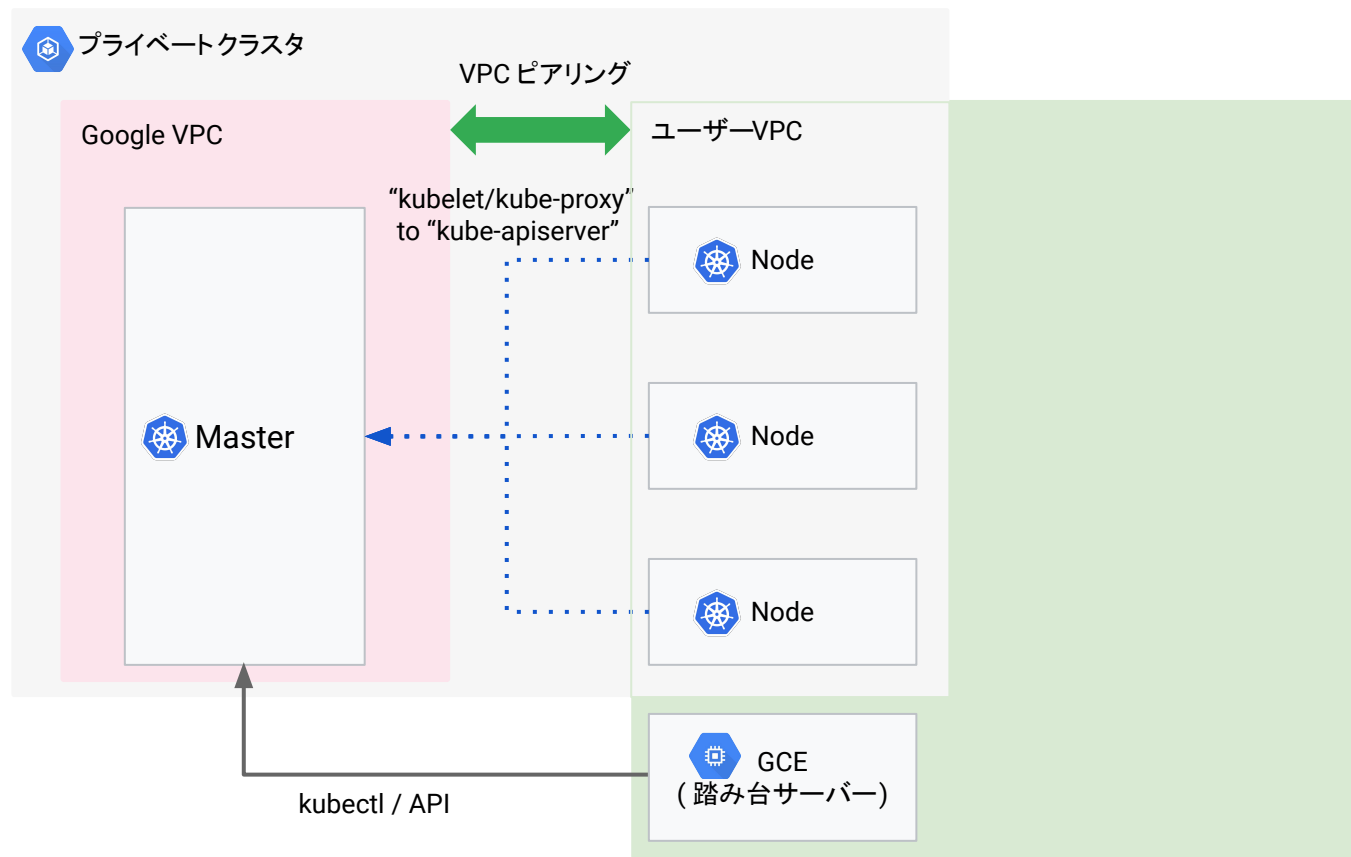
プライベート クラスタ - Outbound Traffic



マスタへのアクセス - Public Endpoint & Master Authorized Network



マスタへのアクセス - Private Endpoint



アクセス管理

GKE Node のサービス アカウント

Default の GCE サービスアカウントは強力な権限を持っている。

最小限の権限のみを持つサービスアカウントを作成して指定することを推奨。

Pod のアクセス権限は Workload Identity を活用 (後述)

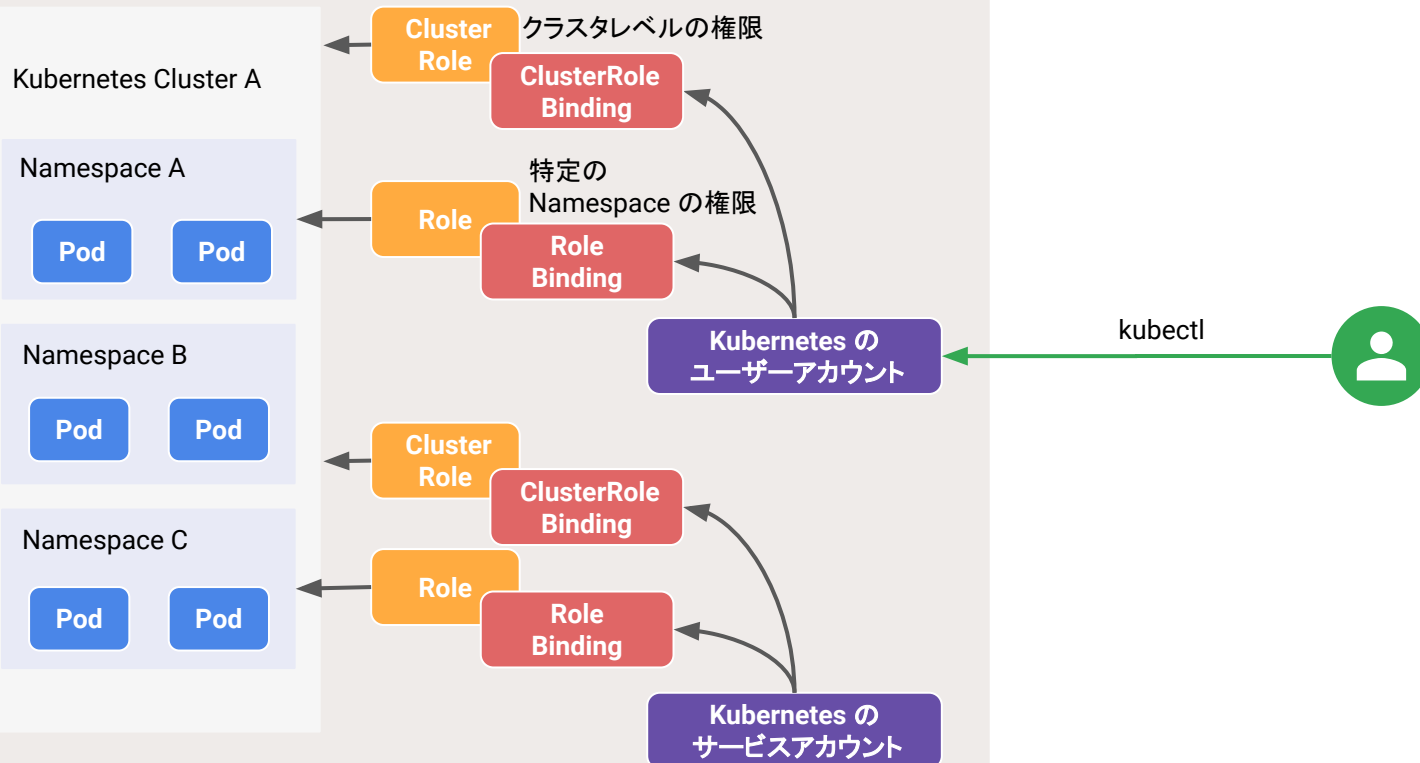
最小権限の例

- roles/monitoring.viewer
- roles/monitoring.metricWriter
- roles/logging.logWriter
- roles/storage.objectWriter
 - GCR 利用のため

Kubernetes におけるアカウントと RBAC



Kubernetes の RBAC (GKE クラスター内部で有効)



GKE におけるアカウントと RBAC の連携



Kubernetes の RBAC (GKE クラスタ内部で有効)

Kubernetes Cluster A

Namespace A

Pod

Pod

Namespace B

Pod

Pod

Namespace C

Pod

Pod

Cluster
Role

クラスタレベルの権限

ClusterRole
Binding

Role

特定の
Namespace の権限

Role
Binding

GCP の
アカウントを
利用する

Kubernetes の
ユーザーアカウント

Cluster
Role

ClusterRole
Binding

Role

Role
Binding

Kubernetes の
サービスアカウント



Cloud IAM (GCP の RBAC)

アカウント+ ロールの組み合わせで
各 GCP サービスへのアクセスを制御

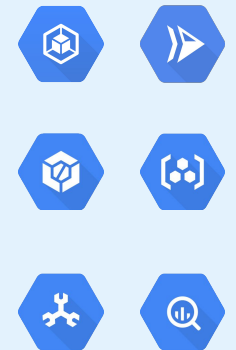
GCP の
ユーザーアカウント

ロール

GCP の
サービスアカウント

ロール

GCP の各サービス



GKE と Google グループの連携

- デフォルトでは Google Cloud のユーザー アカウントまたは Cloud IAM サービス アカウントにのみ GKE のロールを付与可能
- GKE 向け Google グループ(ベータ版)では、G Suite Google グループのメンバーにロールを付与可能
- 利点
 - クラスタ管理者がユーザーの詳細情報を維持する必要がない
 - 既存のユーザー アカウント管理機能と連携
 - 社員が退職したときに、そのユーザーのアクセス権を取り消しなど

```
gcloud beta container clusters create cluster-name \  
  --security-group="gke-security-groups@yourdomain.com"
```

Pod から GCP サービスへのアクセス管理

主な方法は 3 つ

1. Node のサービス アカウントを利用
 - Node 上全てに同等の権限が付与されるため、コンテナによっては必要以上の権限を与えてしまう
2. GCP サービスアカウントのキー (.json) を Pod 内にマウントして利用
 - キー管理の煩雑さがネック。キーの有効期限は 10 年
3. Workload Identity を利用 (推奨)

Workload Identity



Kubernetes の RBAC (GKE クラスター内部で有効)

Kubernetes Cluster A

Namespace A

Pod

Pod

Namespace B

Pod

Pod

Namespace C

Pod

Pod

Kubernetes の
サービスアカウント

Cloud IAM Policy Bind
による紐付け



Cloud IAM (GCP の RBAC)

アカウント+ ロールの組み合わせで
各 GCP サービスへのアクセスを制御

GCP の
サービスアカウント

ロール

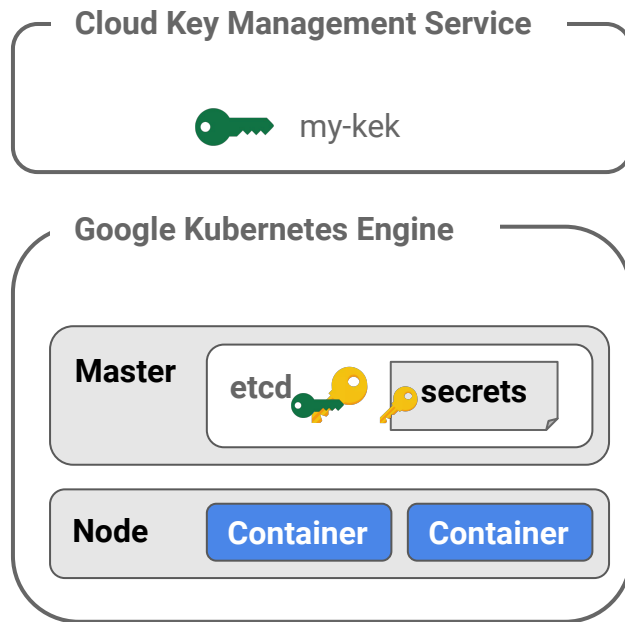
GCP の各サービス



Application layer secrets encryption

コンテナ上のアプリケーションが利用する Kubernetes Secret を
Cloud KMS で暗号化して更なるセキュリティ強化

- etcd のデータを DEK (data encryption key) で暗号化
- DEK を Cloud KMS の KEK (key encryption key) で暗号化
- DEK は 24 時間毎に再生成される



namespace は分離するとよい？

- RBAC によるリソース制御
- **GKE Usage Metering**
 - ワークロードが使用するリソースを namespace やラベル単位で解析
 - CPU, メモリ, GPU, PD, ネットワーク (egress)
 - GCP Billing Export と組み合わせる事でコストを可視化

GCP Project

Cluster

Node

Pod

Container

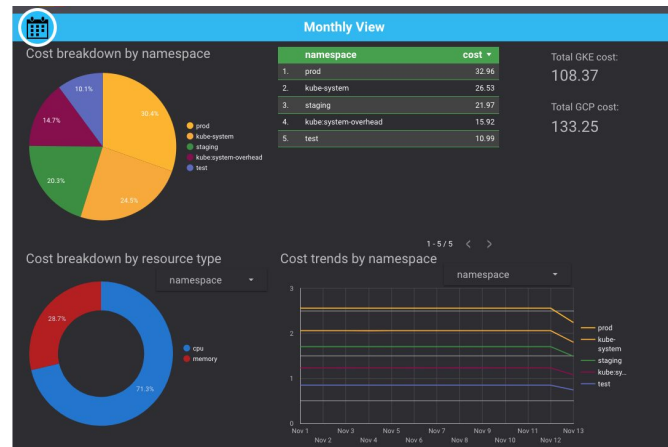
Container

Namespace

● **GCP プロジェクト:**
GCP レベルで完全に分離

● **GKE クラスタ:**
プロジェクト内で最も強い分離方法

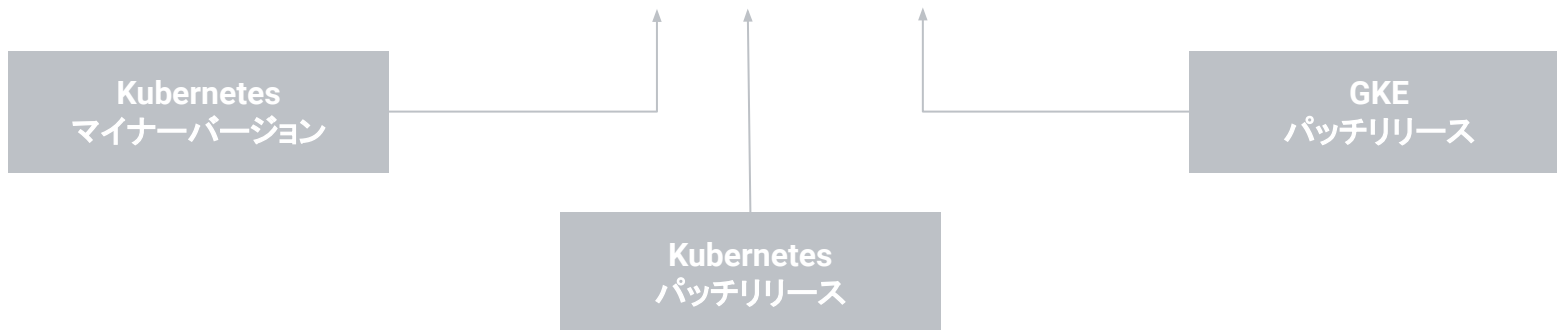
● **Namespace:**
クラスタのコントロールプレーンを分離
(ワークロードは分離されない)



クラスタのバージョン体系

GKE クラスターのバージョン体系

1.14.10-gke.36



```
gcloud container clusters create cluster-name \  
  --region compute-region \  
  --cluster-version version
```

リリース チャンネル

リリースチャンネルは自動更新の頻度とリスクのバランスを自身でコントロール

コンセプトは Chrome の自動更新と同じ

Stable のみ SLA 付与



Master version

Choose Release Channel to get automatic GKE upgrades as new versions are ready. Choose a static version to upgrade manually in the future. [Learn more.](#)

- ☒ Release channel
- ☐ Static version

Release channel

Rapid channel - 1.17.5-gke.9

Regular channel - 1.16.8-gke.15 (default)

Stable channel - 1.14.10-gke.36

```
gcloud container clusters create cluster-name \  
  --region compute-region \  
  --release-channel channel
```

アルファ クラスタ

- Kubernetes のアルファ API を検証したい方向け。
 - 通常の GKE は(どのリリースチャンネルにおいても)ベータ版か安定版のみ提供
- **プロダクション利用目的ではない**
- 留意点
 - SLA 対象外
 - 30 日後にクラスタ自動削除
 - アップグレード不可

```
$ gcloud container clusters create [CLUSTER_NAME] \  
  --enable-kubernetes-alpha \  
  --zone [COMPUTE_ZONE] \  
  --cluster-version [VERSION]
```

クラスタのバージョンアップ

バージョンアップグレードの基礎

Kubernetes は ~ 3 ヶ月毎に最新のバージョンをリリース、パッチは ~ 1 ヶ月毎にリリース

GKE は Kubernetes のリリースに加えて[セキュリティおよびバグ修正](#)を提供

Master Version

Kubernetes のマスタノードで使われるバージョン。Kubernetes クラスタで使われる大半のソフトウェアのバージョンを指す。

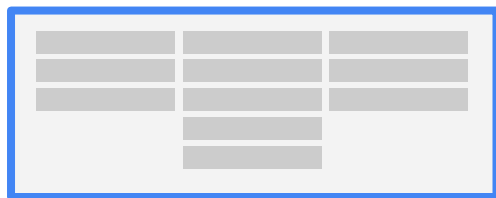
Node Version

Kubernetes のノードで使われるバージョン。主に OS イメージおよび Kubelet のバージョンを指す。

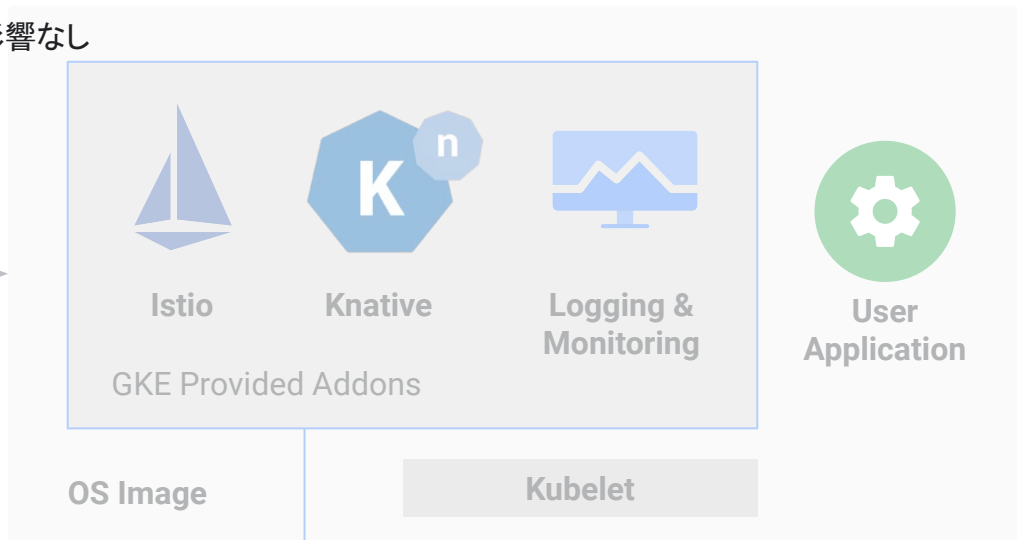
マスタのアップグレード

- 自動アップグレード
- ノードで動いているワークロードには直接影響なし
- Zonal マスタはアップグレード中に停止

Kubernetes master



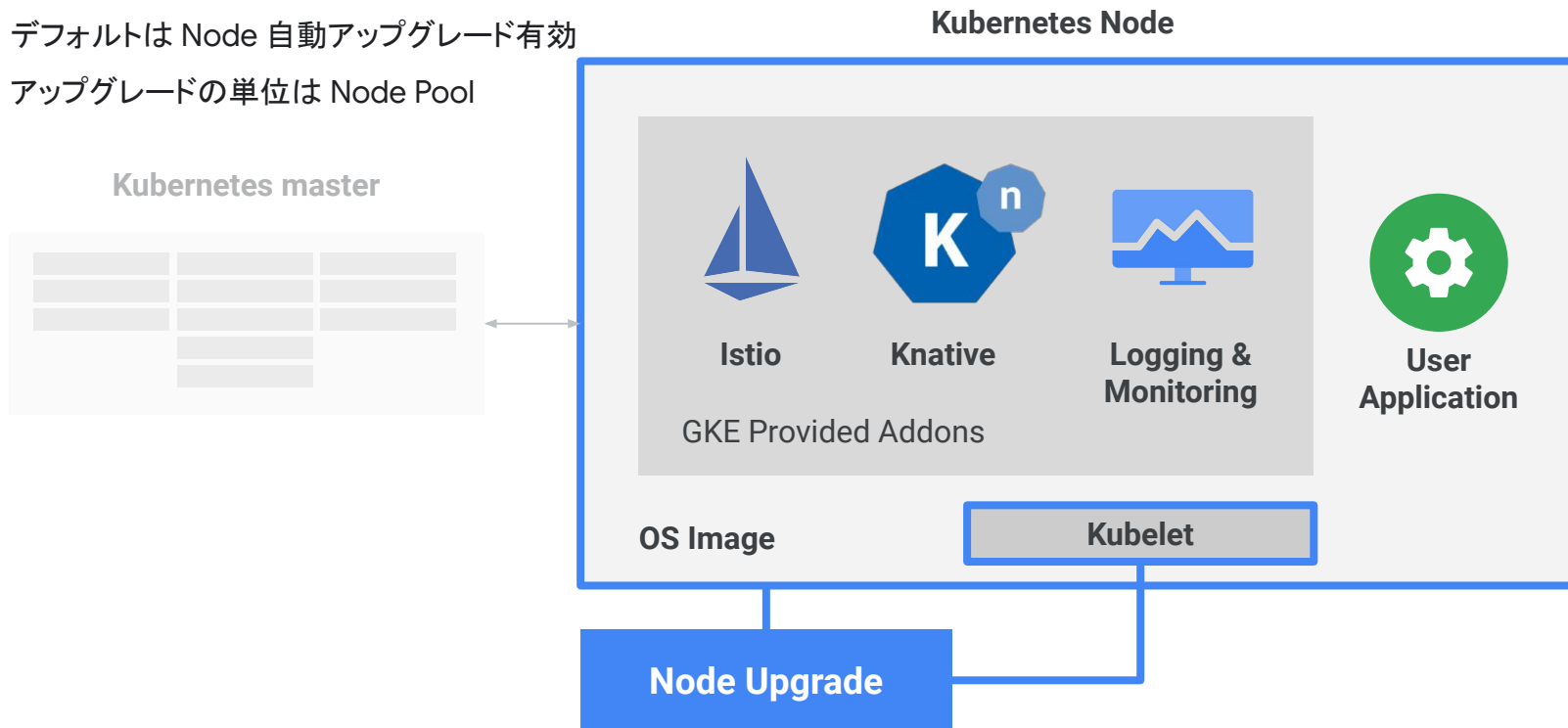
Kubernetes Node



Master Upgrade

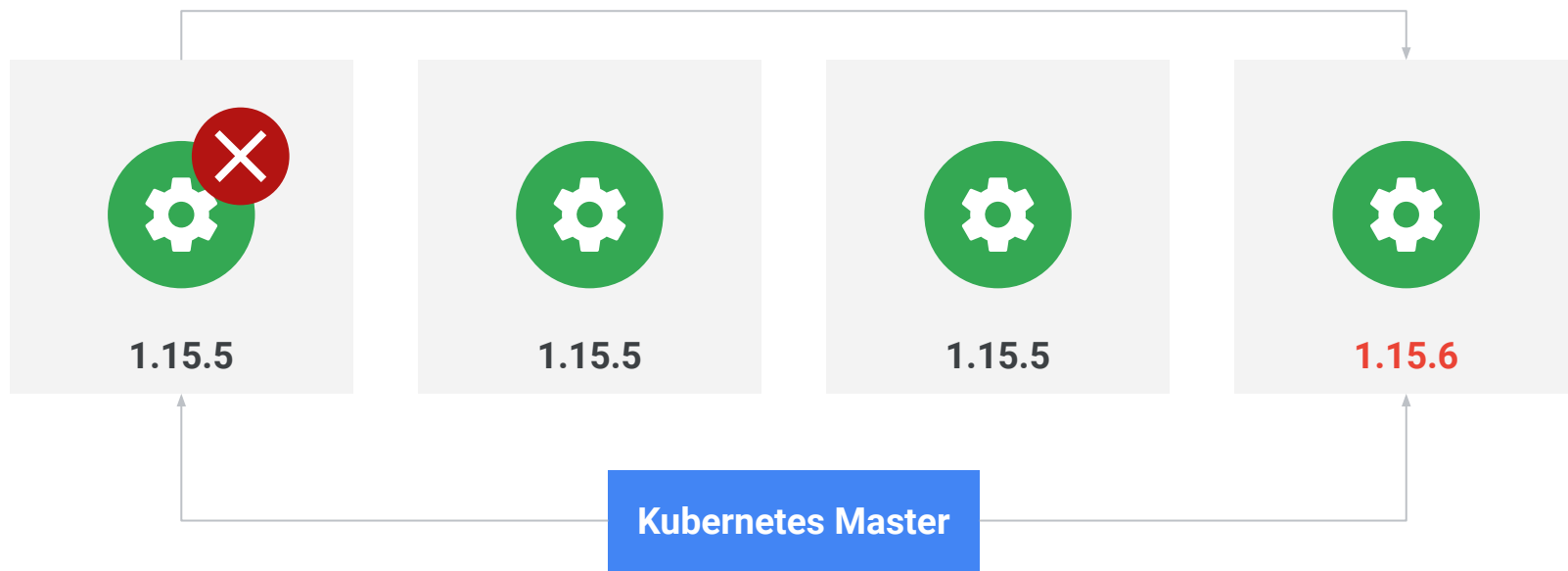
ノードのアップグレード

- デフォルトは Node 自動アップグレード有効
- アップグレードの単位は Node Pool



サージアップグレード

Cluster AutoScaler によって Node pool に新 Node を追加し、既存 Pod を移行していく手法
ストックアウト等のリソース不足による障害の可能性を低減



サージアップグレード Tips

1. サージ アップグレードのスピードを制御

Management

- ☒ Enable auto-upgrade ?
- ☒ Enable auto-repair ?
- ☒ Enable surge upgrade ?

Max surge 1	Max unavailable 0
----------------	----------------------

2. 旧 Node の Pod 削除時のふるまいを制御

```
kind: PodDisruptionBudget
metadata:
  name: km-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: example-app
```

Max surge (default: 1)

- アップグレード中に追加可能な Node 数

Max unavailable (default: 0)

- アップグレード中に停止可能な Node 数
- 空きリソースに Pod を退避
 - 空きがないとペンディングになるので注意

Pod Disruption Budget

- 計画作業時の Pod の並列削除数を抑制

(おまけの Tips)

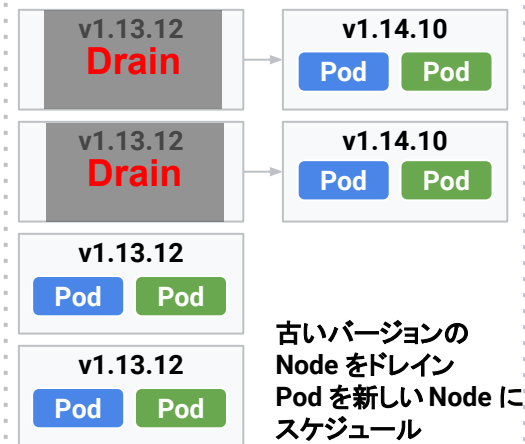
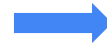
terminationGracePeriodSeconds で SIGKILL までの待機時間を指定
minReadySeconds で新 Pod が Ready になるまでの最低時間を指定



アップグレード開始



新しいバージョンの
Nodeを2つ追加



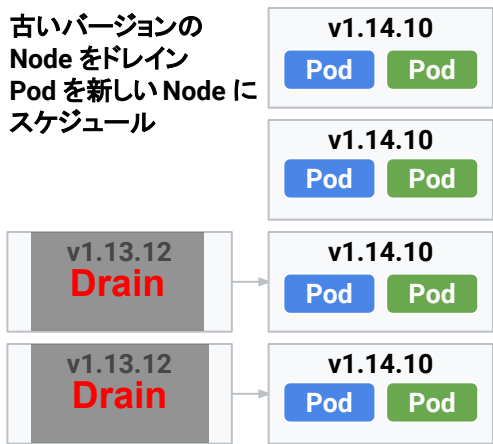
古いバージョンの
Nodeをドレイン
Podを新しいNodeに
スケジュール



Node pool A (計 4 Nodes)
> max_surge_upgrade: 2
> max_unavailable_upgrade: 0



アップグレード完了



古いバージョンの
Nodeをドレイン
Podを新しいNodeに
スケジュール



新しいバージョンの
Nodeを2つ追加

メンテナンスの「実行時間」と「除外時間」

メンテナンス時間枠 (Maintenance Windows)

- 自動メンテナンスの実行を許可する定期的な時間枠
 - 14 日周期で最低 24 時間の時間枠
 - 各時間枠は 4 時間以上の連続した時間

メンテナンス除外 (Maintenance Exclusion)

- 自動メンテナンスの実行を禁止する定期的な時間枠
- 1 クラスタ 3 つまで

v1 ▾ 🔍

Availability

Additional node locations ?
New nodes will be deployed for each zone selected based upon the node pools settings above.

☐ us-central1-b
☐ us-central1-c
☐ us-central1-f

Maintenance window (beta) ?

Days
☒ Monday ☒ Tuesday ☒ Wednesday ☒ Thursday
☒ Friday ☒ Saturday ☒ Sunday

Start time
Any Time ▾

Length
24h ▾

Hours shown in your local time zone
(UTC-7)

Switch to custom

Maintenance window exclusions

Start time
8/30/19, 6:28 AM ▾

End time
8/31/19, 6:28 AM ▾

✕

+ Add blackout date

まとめ

まとめ

おすすめ記事

[Google エンジニアと学ぶ GCP \[コンテナ\]](#)

- リージョン クラスタ
 - SLA は 99.95 %
 - マスタ アップグレードのリスクも極めて低い
- VPC ネイティブモード
 - コンテナ ネイティブ負荷分散を活用
 - カスタム VPC の利用、サブネットの作成を検討
- プライベート クラスタ
- アクセス管理
 - IAM、Workload Identity を活用して最小権限を付与
 - Cloud KMS で Secret のセキュリティを更に強化
- バージョンアップ
 - リリース チャンネル、サージアップグレード、メンテナンス ウィンドウを活用

Thank you!

Appendix

マシン イメージとコンテナ ランタイム

- Container-Optimized OS ※ おすすめ
 - Docker
 - containerd (version 1.14.3 以上)
- Ubuntu
 - Docker
 - containerd (version 1.14.3 以上)
- Windows

GKE が予約する Node のリソース

CPU

- 最初のコアの 6 %
- 次のコアの 1 % (最大 2 コア)
- 次の 2 コアの 0.5 % (最大 4 コア)
- 4 コアを超えるコアの 0.25 %

ストレージ

- 最大 100 GB (コンテナイメージの保存)

メモリ

- 255 MB (メモリ 1 GB 未満の VM)
- 最初の 4 GB のメモリの 25% (最大 8 GB)
- 次の 4 GB のメモリの 20 % (最大 8 GB)
- 次の 8 GB のメモリの 10 % (最大 16 GB)
- 次の 112 GB のメモリの 6 % (最大 128 GB)
- 128 GB を超えるメモリの 2%