コンテナ研修

コンテナを Kubernetes にデプロイしてみよう

浅野 大我

アジェンダ

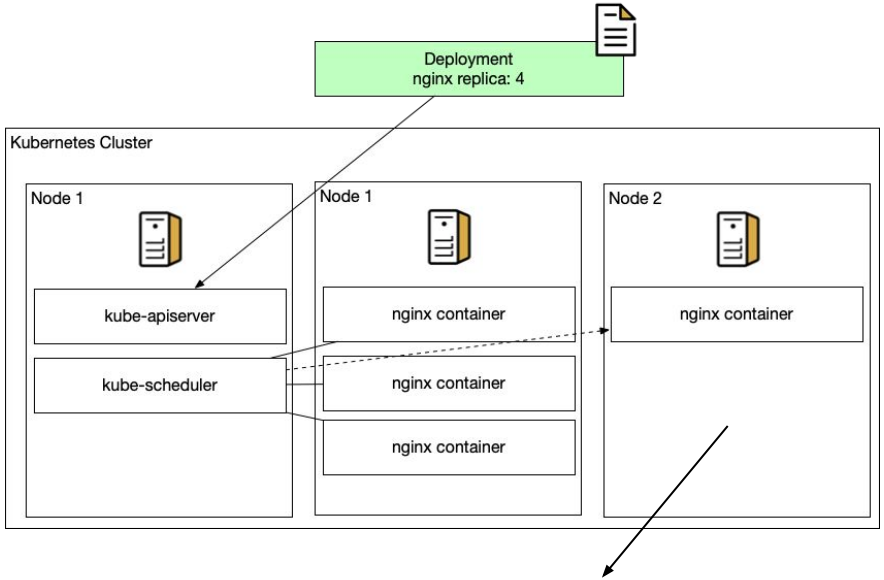
- コンテナオーケストレーション / Kubernetes とは
- Google Kubernetes Engine (GKE) とは
- パブリッククラウドにおける コンテナオーケストレーションツール
- Kubernetes コントローラーを理解しよう
- Kubernetes リソース（オブジェクト）を理解しよう
- Kubernetes にアプリをデプロイしよう
- カスタムコントローラー / Managed Prometheus を使って見よう

MIXI

コンテナオーケストレーション / Kubernetes とは

Kubernetes とは？

- Google の Borg と呼ばれる実行基盤がある
  - これを元にした OSS で、Google や CNCF といった団体が管理、メンテナンスしている
- Borg はコンテナオーケストレーションツール
  - ホストマシンを意識せずに、効率よくコンテナ（アプリケーション）を配置したい
  - 耐障害性を考えると、ホストマシンを意識しないようにしたい
  - Google ではその考えを元にBorgが開発され、運用されている
- Google の技術が OSS になり、一般的に使われるようになった
  - その他: Colossus、Andromeda、Zanzibar...



どうしてコンテナオーケストレーションツール？

- Borgの論文によるメリットは以下
  - リソース管理、障害への処理を隠してエンジニアをアプリケーション開発に集中させる
  - 高い信頼性と可用性を保つ
  - 数万台オーダーのマシンでワークロードを効率的に実行させる
- リソース管理を隠す
  - YAML ファイルを使ってあらゆるリソースを表現することができる
  - Kubernetes は kube-scheduler を使ってコンテナを指定したノードへ効率よく配置することができる
- 障害への処理
  - ノードに障害が発生した際、コンテナは自動的に回復する
- 信頼性、可用性
  - Kubernetes は etcd と呼ばれる分散ストレージを使って状態を管理している
  - マルチマスター構成にすることでSPOF（単一障害点）を減らすことができる
    - Borgは99.999%の可用性を維持している
- 数万台オーダーのマシンでワークロードを効率的に実行させる
  - スケジューラーの実装次第で効率よく配置できる
    - Gmail、Google Docs、検索、BigTableなどで使われている（らしい）

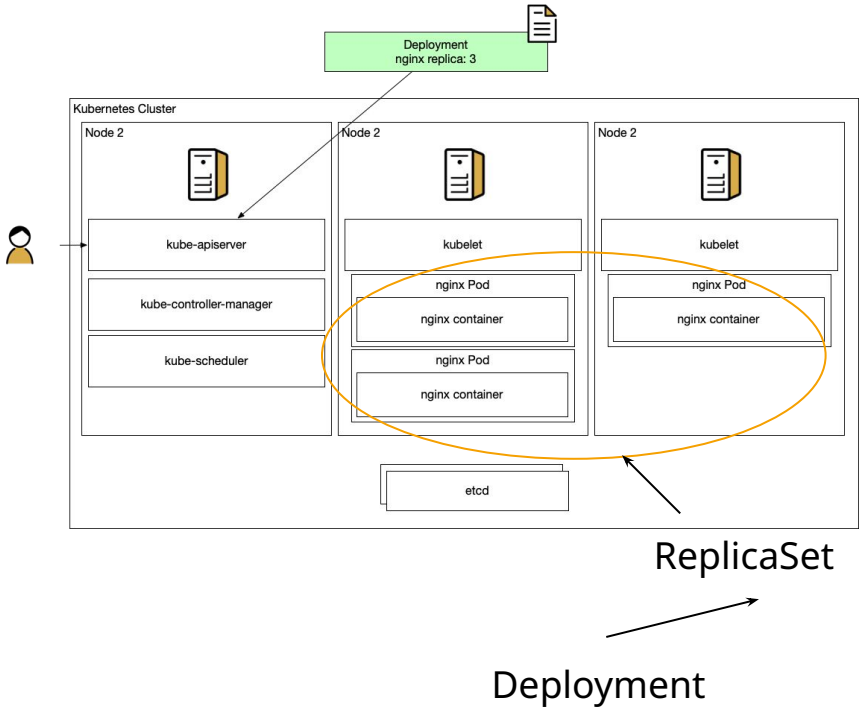
Google Kubernetes Engine (GKE) とは

Google Kubernetes Engine (GKE) とは

- Google Kuberntes Engine (GKE) はマネージド Kubernetes サービスの一つ
- Kubernetes のコントロールプレーン部分の定期的なアップデートや、コントローラーを提供してくれる
  - ノードの可用性などを気にせずにKubernetesの恩恵を受けることができる
- クラスタネットワーキングや、ノードのカスタマイズがある程度柔軟ではあるが・・・
  - アップデートなどの面倒を見る必要はある
  - ワーカーノードの効率的な活用は自分たちでやる必要がある
- 最近ではそのレベルの面倒を見てくれる GKE Autopliot もある
  - しかし構成が柔軟に変更できないなどの制約もある
  - Pod単位の課金になる
- 今回は Standard と呼ばれる方を扱います

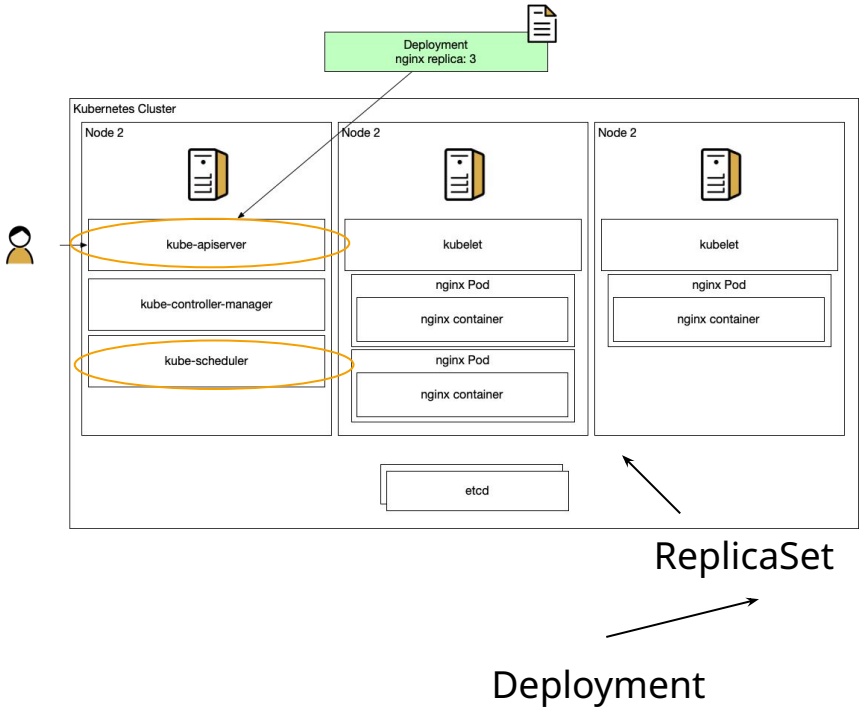
Kubernetes を簡単に理解する

- nginx コンテナを3つ起動する...とする
- Pod
  - コンテナをまとめた単位
  - Pod の中で複数のコンテナが起動することもある
- ReplicaSet
  - Pod の集合体
  - 規定された Pod の数を保つように管理している
- Deployment
  - ReplicaSet を管理している
  - 指定された構成や ReplicaSet 自体のアップデートを担う
- リソースをどこに作って、誰がコンテナを作る？



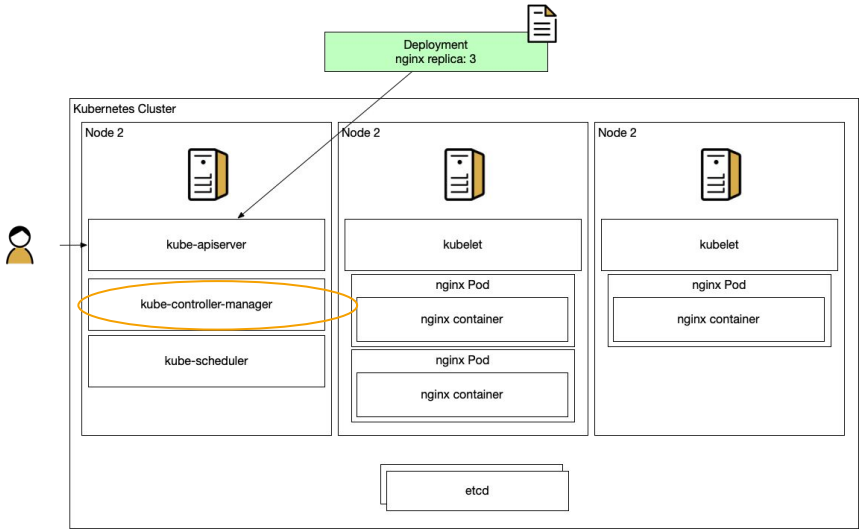
Kubernetes を簡単に理解する

- Node に対して Master / Worker という役割を与える
  - Node: 実際のKubernetesクラスタが動作するマシン
  - Master: Kubernetesを司るコントロールプレーンとして指定されたノード
- kube-apiserver は Kubernetes を制御するAPIサーバー
  - リソースの登録や状態の登録を行う
- kube-scheduler
  - Node に対してPod（コンテナ類）を割り当てるスケジューラー
  - Node の空き状況を見ていい感じに Pod をスケジュールする



Kubernetes を簡単に理解する

- kube-controller-manager
  - Kubernetes にビルトインされたリソースを監視、管理するコントローラー
  - 例えば、
    - コンテナ（**Deployment, Pod, ReplicaSet**）
    - ネットワーク（**Service**）
    - ボリューム（**Volume**）
  - これらが登録された際に、状態を見て指定されたオブジェクトを作る（＝リコンサイルする）
- コントローラーが実際にオブジェクトを作るフローを見てみる

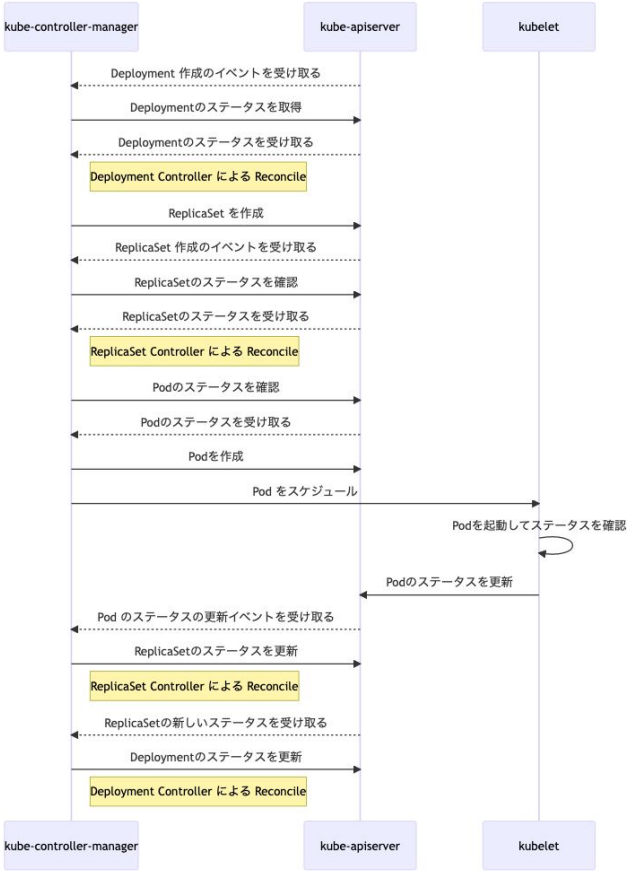


MIXI

Kubernetes コントローラーを理解しよう

Kubernetes コントローラーを理解する

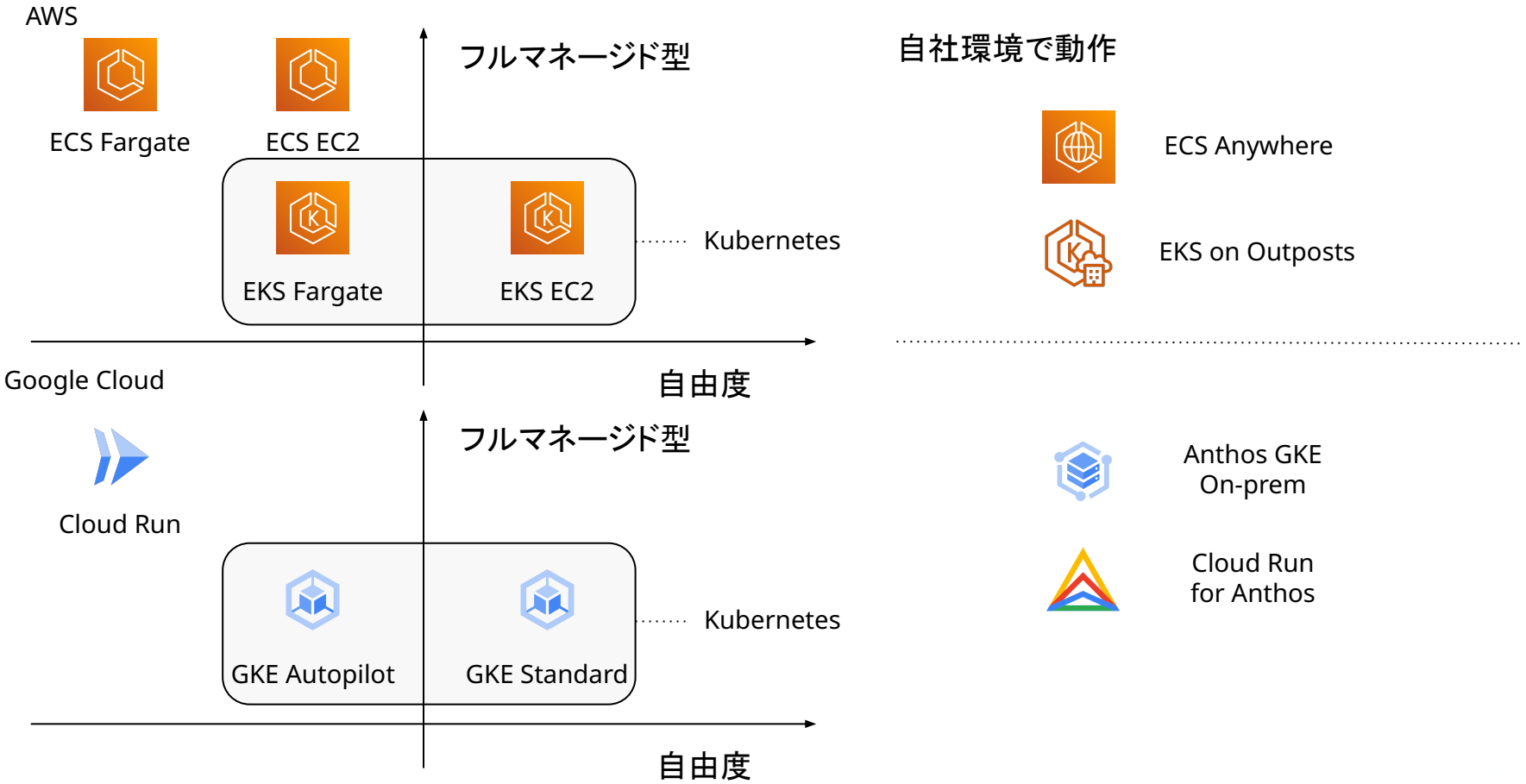
- Deployment Controller が Deployment リソースの作成を検知
  - Deployment Controller が ReplicaSet リソースを作成
- ReplicaSet Controller が ReplicaSet リソースの作成を検知
  - ReplicaSet Controller が Pod リソースを作成
- 削除時は逆
- オブジェクトの追加や削除を Reconcile Loop で行う



MIXI

パブリッククラウドにおける
コンテナオーケストレーションツール

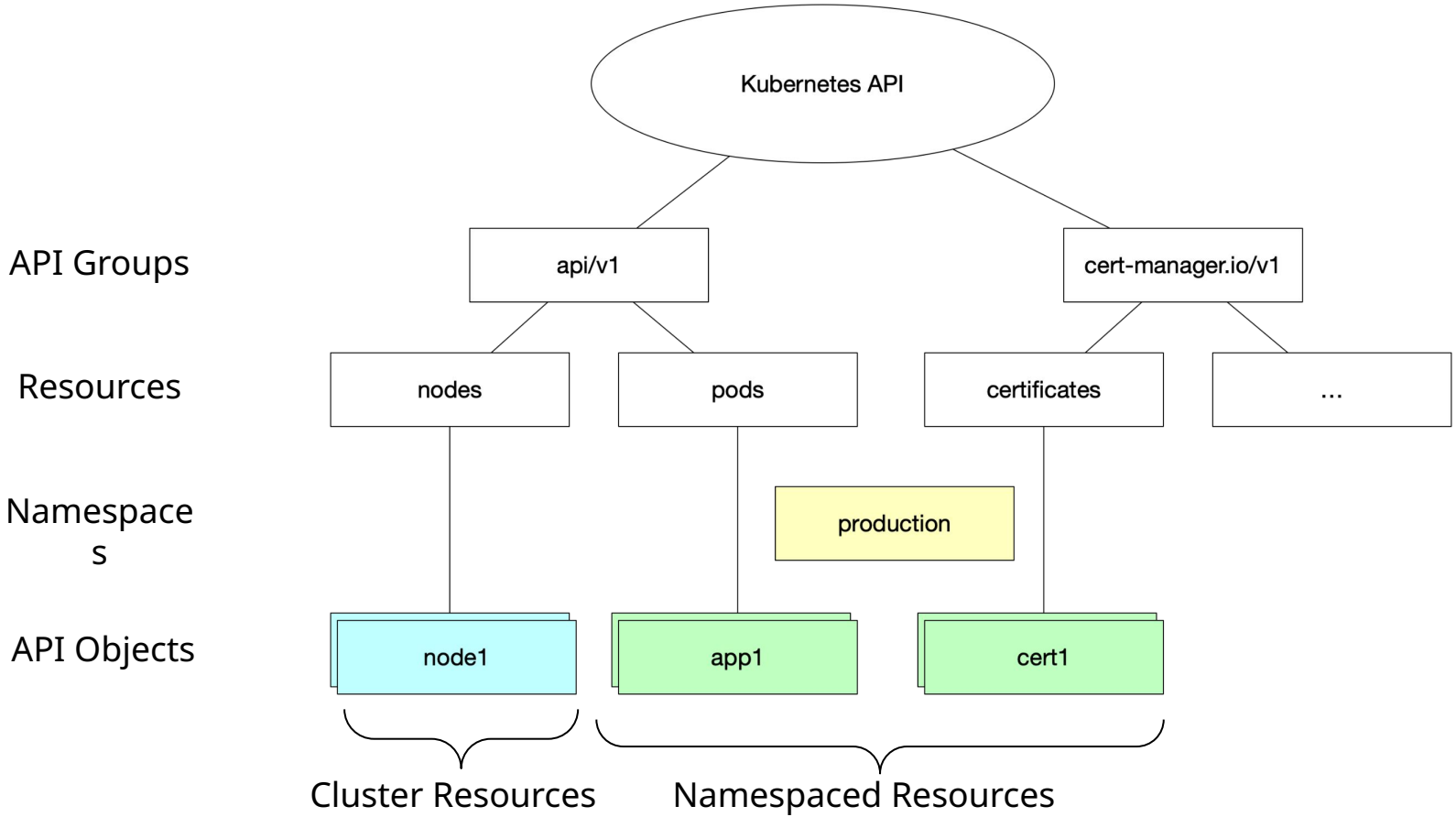
コンテナオーケストレーションツール



MIXI

Kubernetes リソース（オブジェクト）を理解しよう

今回の流れ


Kubernetes API
API Groups
api/v1
cert-manager.io/v1
Resources
nodes
pods
certificates
...
Namespaces
production
API Objects
node1
app1
cert1
Cluster Resources
Namespaced Resources

MIXI

実際に見てみよう

$ kubectl get node

Kubernetesリソースを理解しよう

Deployment Controller	PodとReplicaSetを理想的な状態に保つ

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

実際に見てみよう

$ kubectl get pod --all-namespaces

どんなPodがデプロイされているのか？

```
NAMESPACE     NAME                                                READY   STATUS    RESTARTS   AGE
kube-system   anetd-4zvx5                                         1/1     Running   0          3h48m
kube-system   anetd-8s48b                                         1/1     Running   0          3h48m
kube-system   anetd-sknbr                                         1/1     Running   0          3h48m
kube-system   antrea-controller-horizontal-autoscaler-6fb4bf7847-hdfjc 1/1     Running   0          3h53m
kube-system   event-exporter-gke-857959888b-2nlk6                 2/2     Running   0          3h53m
kube-system   fluentbit-gke-8mfwg                                 2/2     Running   0          3h48m
kube-system   fluentbit-gke-cj8q7                                 2/2     Running   0          3h48m
kube-system   fluentbit-gke-fwr8t                                 2/2     Running   0          3h48m
kube-system   gke-metadata-server-dk8kd                           1/1     Running   0          3h48m
kube-system   gke-metadata-server-l8qql                           1/1     Running   0          3h48m
kube-system   gke-metadata-server-v5ppl                           1/1     Running   0          3h48m
kube-system   gke-metrics-agent-bb9tj                             1/1     Running   0          3h48m
kube-system   gke-metrics-agent-brb7t                             1/1     Running   0          3h48m
kube-system   gke-metrics-agent-qdvtz                             1/1     Running   0          3h48m
kube-system   konnectivity-agent-579787d677-c77vn                 1/1     Running   0          3h47m
kube-system   konnectivity-agent-579787d677-jxftl                 1/1     Running   0          3h47m
kube-system   konnectivity-agent-579787d677-ltnf4                 1/1     Running   0          3h53m
kube-system   konnectivity-agent-autoscaler-7d9fbfd578-bltmc      1/1     Running   0          3h53m
kube-system   kube-dns-7d5998784c-sx2tx                           4/4     Running   0          3h53m
kube-system   kube-dns-7d5998784c-xmcl9                           4/4     Running   0          3h53m
kube-system   kube-dns-autoscaler-9f89698b6-llcs8                 1/1     Running   0          3h53m
kube-system   l7-default-backend-6dc845c45d-fn7qc                 1/1     Running   0          3h53m
kube-system   metrics-server-v0.5.2-6bf845b67f-fmfqk              2/2     Running   0          3h47m
kube-system   netd-5vz4m                                          1/1     Running   0          3h48m
kube-system   netd-9j52l                                          1/1     Running   0          3h48m
kube-system   netd-qjk8b                                          1/1     Running   0          3h48m
kube-system   pdcsi-node-c56tf                                    2/2     Running   0          3h48m
kube-system   pdcsi-node-qzkkw                                    2/2     Running   0          3h48m
kube-system   pdcsi-node-x42ns                                    2/2     Running   0          3h48m
```

Kubernetesリソースを理解しよう

Service Controller	Podを外部に公開するためのコントローラー

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

実際に見てみよう

$ kubectl get svc --all-namespaces

どんなServiceがデプロイされているのか？

```
$ kubectl get svc --all-namespaces
NAMESPACE     NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
default       kubernetes             ClusterIP   10.112.0.1      <none>        443/TCP
23h
kube-system   default-http-backend   NodePort    10.112.12.179   <none>        80:31114/TCP
23h
kube-system   kube-dns               ClusterIP   10.112.0.10     <none>        53/UDP,53/TCP
23h
kube-system   metrics-server         ClusterIP   10.112.2.112    <none>        443/TCP
23h
```

Kubernetesリソース（オブジェクト）を理解しよう

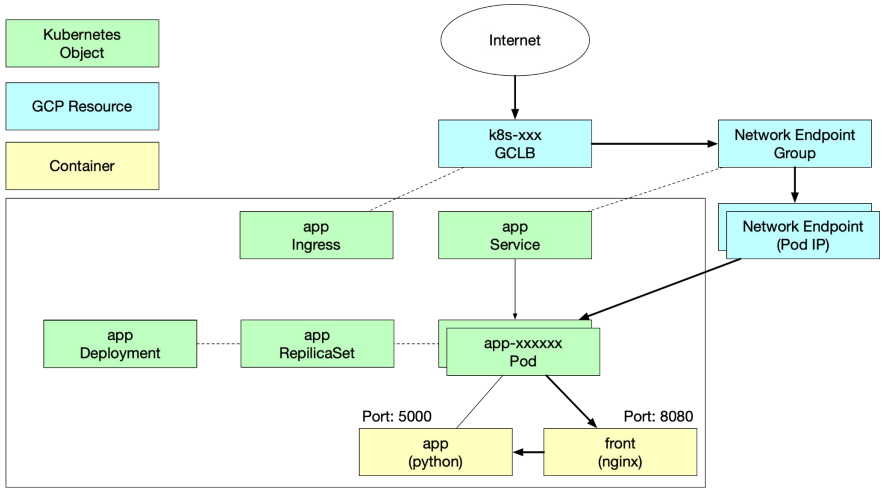
- Kubernetesのリソース、オブジェクト、APIの分類について理解した
  - Cluster Resource と Namespaced Resource
- Kubernetes コントローラーの役割について、コマンドを使いながら理解した
  - Deployment
  - Service

MIXI

Kubernetes にアプリをデプロイしよう

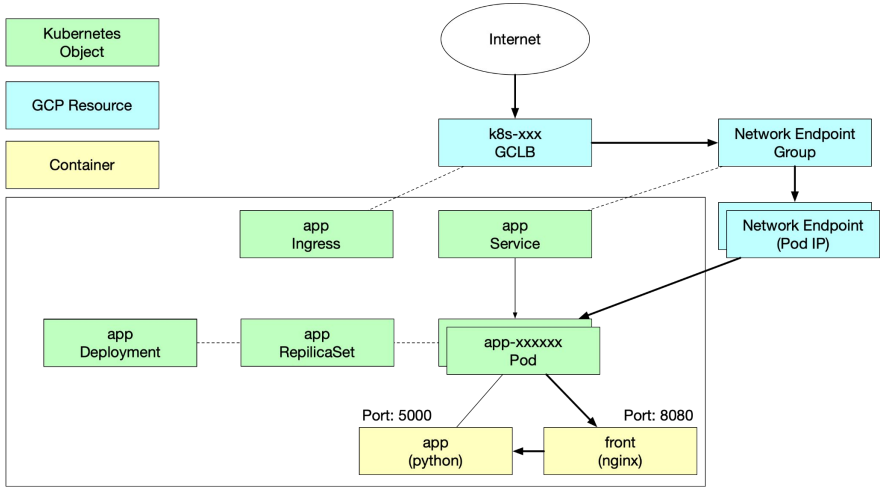
Kubernetesリソース（オブジェクト）を理解しよう

- 前半でやったアプリケーションを GKE 上にデプロイしてみる
- 前半のアプリケーションに nginx を挟んで Basic 認証を付ける
- 複数のコンテナを Pod として扱うことも可能



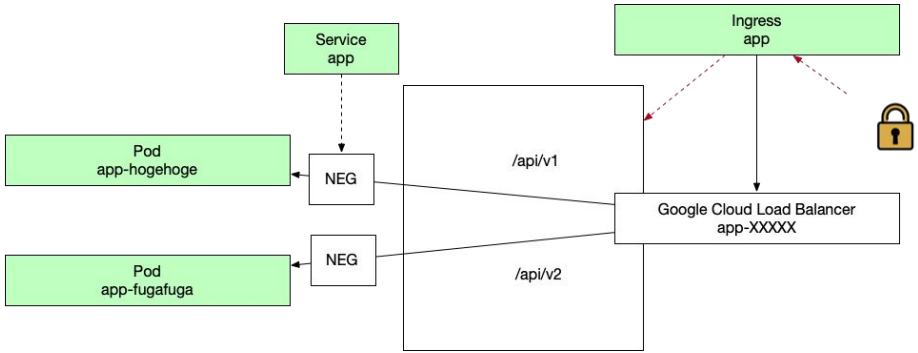
Kubernetesリソース（オブジェクト）を理解しよう

- GKE -> Kubernetes
- Network Endpoint = Pod
- Network Endpoint Group = Service
- 負荷分散 (LB / ロードバランサ) = Ingress
  - https://medium.com/google-cloud-jp/neg-%E3%81%A8%E3%81%AF%E4%BD%95%E3%81%8B-cc1e2bbc979e



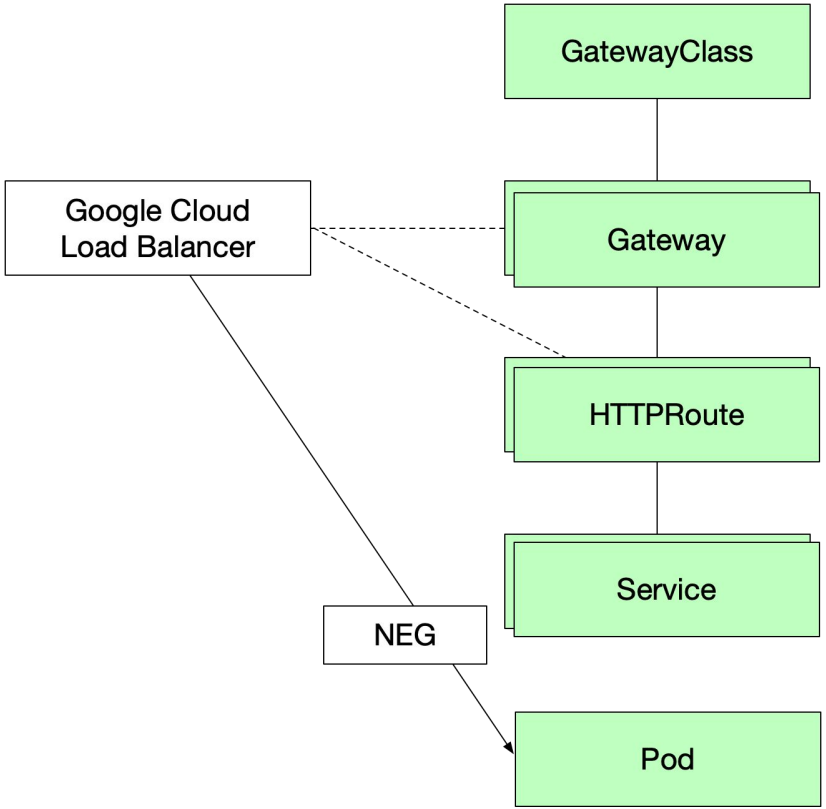
Kubernetes のリソース: Ingress

- Service に対して、さらに外部からのアクセスを管理するリソース
  - Serviceそのものは Kubernetes クラスタ内部のネットワークを管理するもの
    - type: LoadBalancer を使うと Pod へ L2 でアクセスさせることができる
  - Ingress は L7 をカバーする
- ロードバランサーの設定などができる（ざっくり）
  - パスベースでどこへルーティングする
  - TLS 証明書を設定する



Kubernetes のリソース: Gateway (Beta)

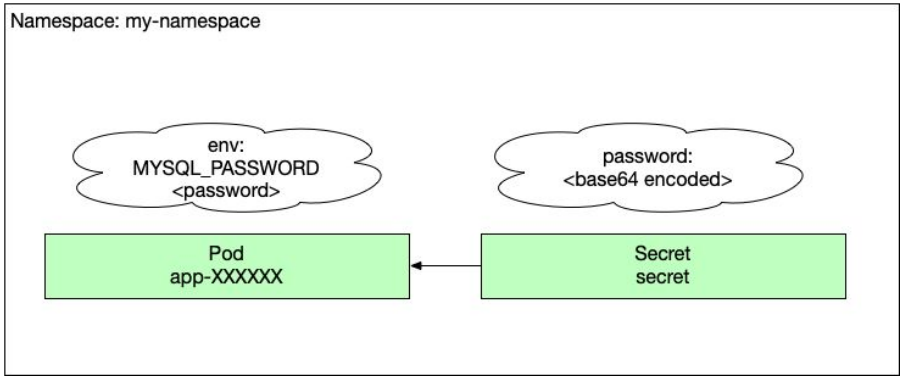
- Ingress をさらに拡張したリソース
- Ingress ではアノテーションのラベルベースで
  やっていたことを API レベルでサポート
  - 証明書の設定
  - HTTP 以外のプロトコルの柔軟なサポート
    - TCP / UDP から HTTP / HTTPS まで
  - Kubernetes RBAC を利用した柔軟な権限管理
- Google Kubernetes Engine でのサポート
  - シングルクラスタ、マルチクラスタのサポート
  - HTTPRoute のサポート
    - TCPRoute などは未実装
  - マネージド証明書のサポート（一部）
  - フルで利用可能になるのはまだ時間がかかる



https://gateway-api.sigs.k8s.io/

Kubernetesのリソース: Secret

- パスワードなどを管理するオブジェクトのこと
  - Podの環境変数や、コンテナイメージに機密情報を書き込まないようにするためのもの
- Secretを作ってコンテナの環境変数として参照させる（他もできる）
- 初期状態では暗号化はされていない
  - Kubernetes RBACを使ってアクセス制御をしたり…
  - Secretをクラスタ/マニフェスト単位で暗号化するツールを使う
  - Secret管理を外部のキーマネジメントシステムにやらせる
- 今回は暗号化されていないまま使います



実際にデプロイしてみよう

$ kubectl apply -f namespace.yaml

$ kubectl apply -f secret.yaml

$ kubectl apply -f deployment.yaml

$ kubectl get deployment -n namespace

$ kubectl get pod -n my-namespace

実際にデプロイしてみよう

$ kubectl apply -f service.yaml

$ kubectl apply -f ingress.yaml

Ingress がロードバランサとしてデプロイされていることを確認する

MIXI

← ロードバランサの詳細　編集　削除　→ネットワーク トポロジで表示

k8s2-um-qujkihr3-my-namespace-ingress-bhrwqf7e

Cloud CDN と Cloud Armor を使用したウェブ パフォーマンスの高速化とウェブ保護の向上。詳細

詳細　モニタリング　キャッシュ

フロントエンド

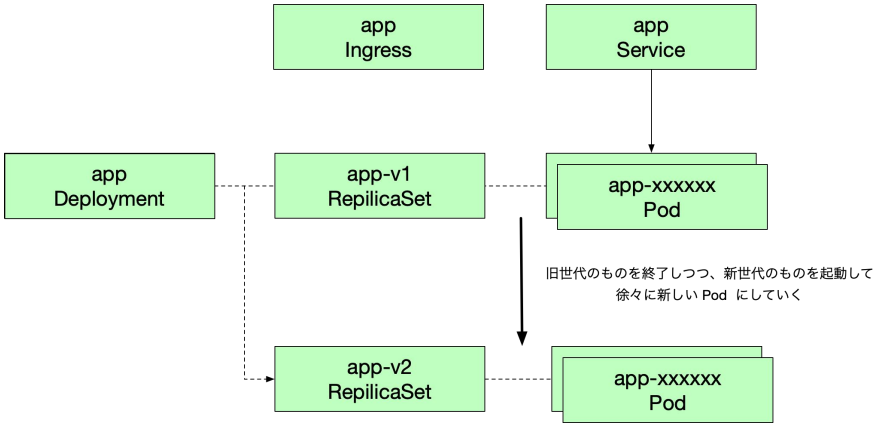
プロトコル ↑	IP:ポート	証明書	SSL ポリシー	ネットワーク階層 ?
HTTP	35.241.16.152:80	-		プレミアム

ホストとパスのルール

ホスト ↑	パス	バックエンド
不一致すべて（デフォルト）	不一致すべて（デフォルト）	k8s-be-30723--734b8d51cc0a5900
*	/*	k8s1-734b8d51-my-namespace-app-8080-d1e99d3e
*	/*	k8s-be-30723--734b8d51cc0a5900

Kubernetesリソース（オブジェクト）を理解しよう

- Deploymentを変更したとき、どのように ReplicaSet, Podが作成されるのかを観察してみる
- Deploymentのデプロイ戦略
  - RollingUpdate (デフォルト)
    - ローリングアップデートされていく
  - Recreate
  - Podを全部停止して、作り直す
- https://kubernetes.io/ja/docs/concepts/workloads/controllers/deployment/



MIXI

Exercise:
環境変数を足してDeploymentが新しいReplicaSetを作る
ところを見てみよう

環境変数を足してDeploymentが新しいReplicaSetを作るところを見る

1. Deploymentに環境変数を足す

```
--- a/kubernetes/deployment.yaml
+++ b/kubernetes/deployment.yaml
@@ -37,6 +37,8 @@ spec:
                  secretKeyRef:
                    name: secret
                    key: password
+             - name: HELLO
+               value: HOGEHOGE
            livenessProbe:
              httpGet:
                path: /liveness
```

2. applyする

   kubectl apply -f deployment.yaml

   このとき、別窓で kubectl get replicaset -w -n my-namespace しておく

Kubernetesリソース（オブジェクト）を理解しよう

- kubectlを使ってアプリケーションをデプロイした
  - Deployment
  - Service
  - Ingress
- Exercise
  - Deploymentのアップデート戦略について調べる
  - Deployment以外のPodのコントローラーについて調べてみる
  - Serviceのタイプについて詳しく調べてみる

MIXI

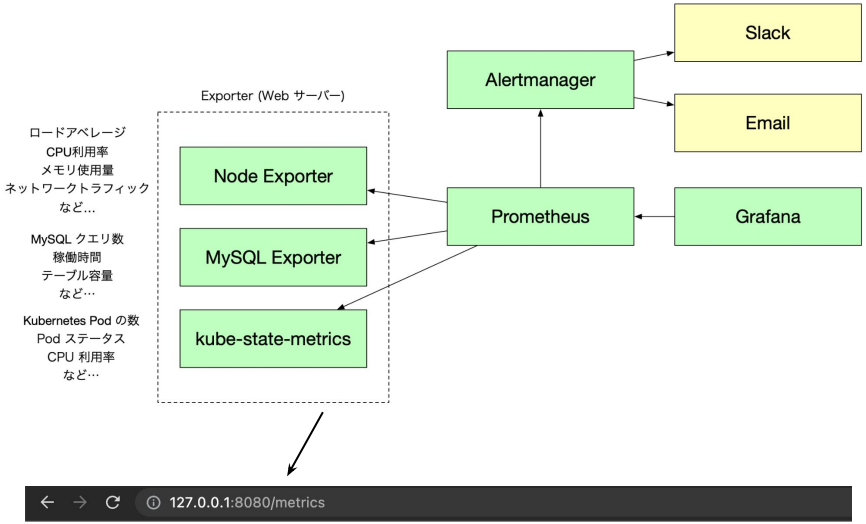
カスタムコントローラー
Managed Prometheus を使って見よう

いろいろなOperator / Custom Controller

- cert-manager
  - Kubernetes リソースとして証明書の発行や管理が出来る Custom Controller
- Prometheus Operator
  - メトリクス収集に必要な Prometheus のデプロイや Kubernetes リソースと監視の紐付けが出来る Operator
- **Google Cloud Managed Service for Prometheus**
  - **Google Cloud で利用できるマネージド Prometheus**
- Grafana Operator
  - Grafanaのデプロイが簡単にできるOperator
- Config Connector (Google Kubernetes Engine)
  - Kubernetes リソースとして Google Cloud のリソースを管理出来る Operator

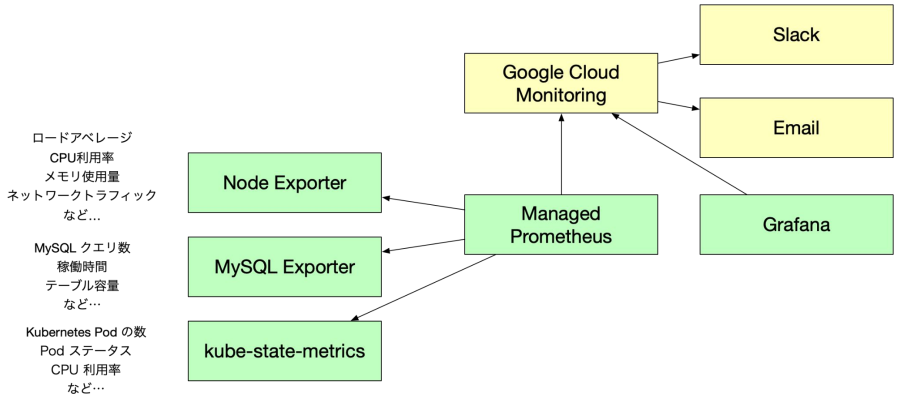
Prometheus とは

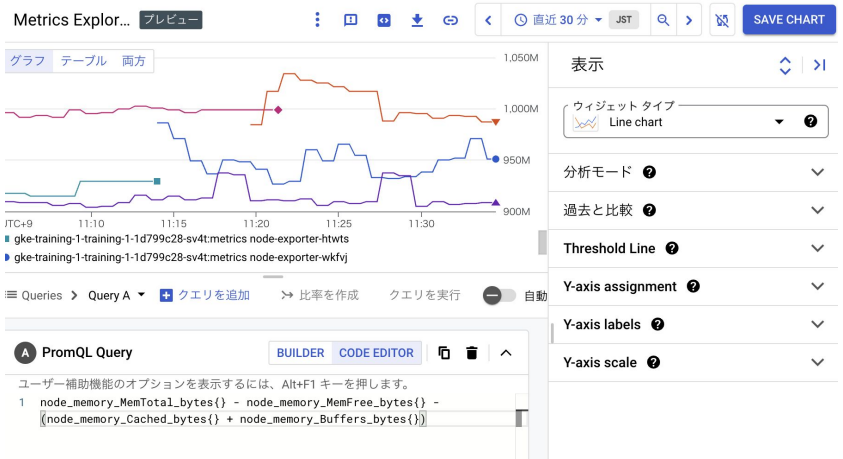
- Prometheus は色々なメトリクスを収集するアプリケーション
  - Exporter というメトリクスを書き出すサーバーから、定期的にメトリクスを収集
  - Prometheus の持つストレージか、好きなストレージに書き出してメトリクスを保存
  - PromQL という言語を使ってクエリして、アラートやグラフを書き出す
- 時系列 DB をスケール/マネージドにさせるために、色々な派生プロダクトが存在する
  - VictoriaMetrics
  - New Relic Remote Write
  - Grafana Cloud
- https://prometheus.io/



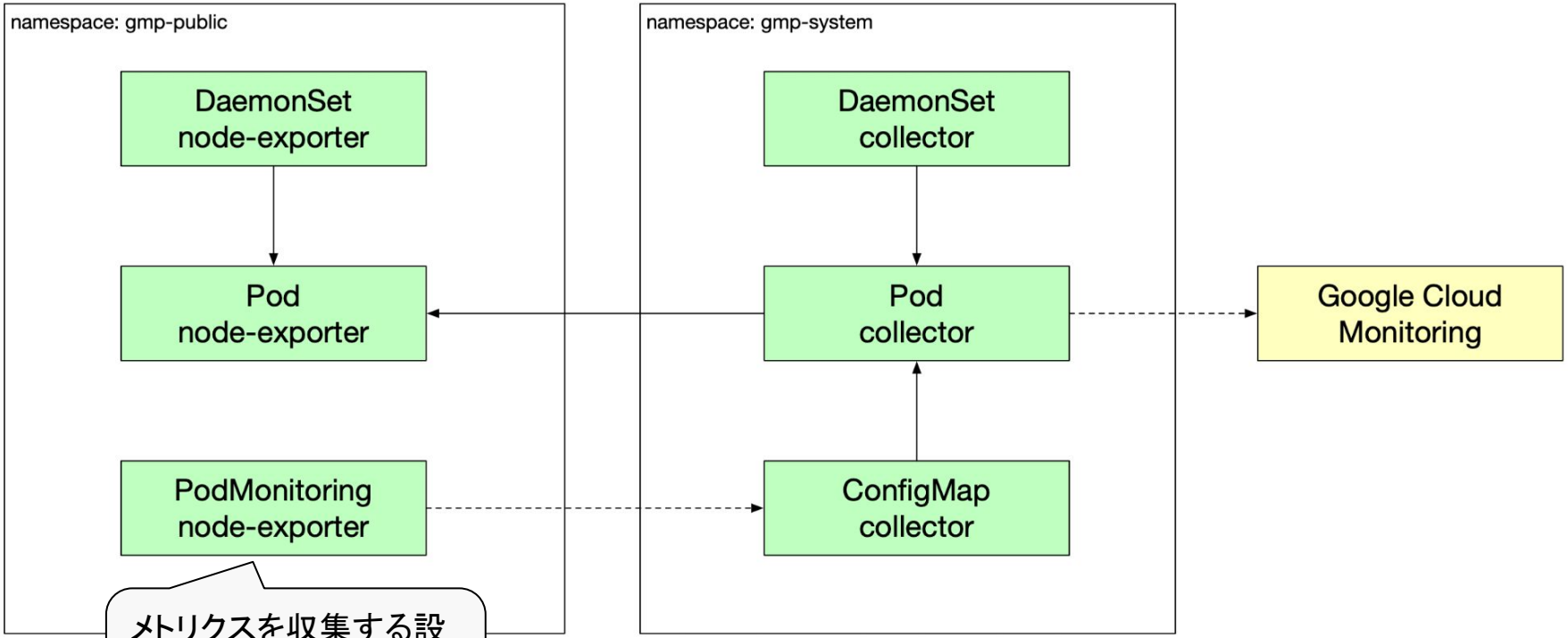
Google Cloud Managed Service for Prometheus

- Prometheus を実用するには考えることがたくさんある
  - Prometheus の管理
  - データの保存先（可用性）
  - 保存期間
  - Exporter の設定管理
- Google Cloud がマネージドにしてくれるのが Google Cloud Managed Service for Prometheus
- インフラ管理者はマネージド Prometheus を GKE クラスタで有効にするだけで良い
- アプリケーション開発者は Prometheus メトリクスを出すことに集中すれば良い
- ダッシュボードも Google Cloud で見れる
- 料金に注意





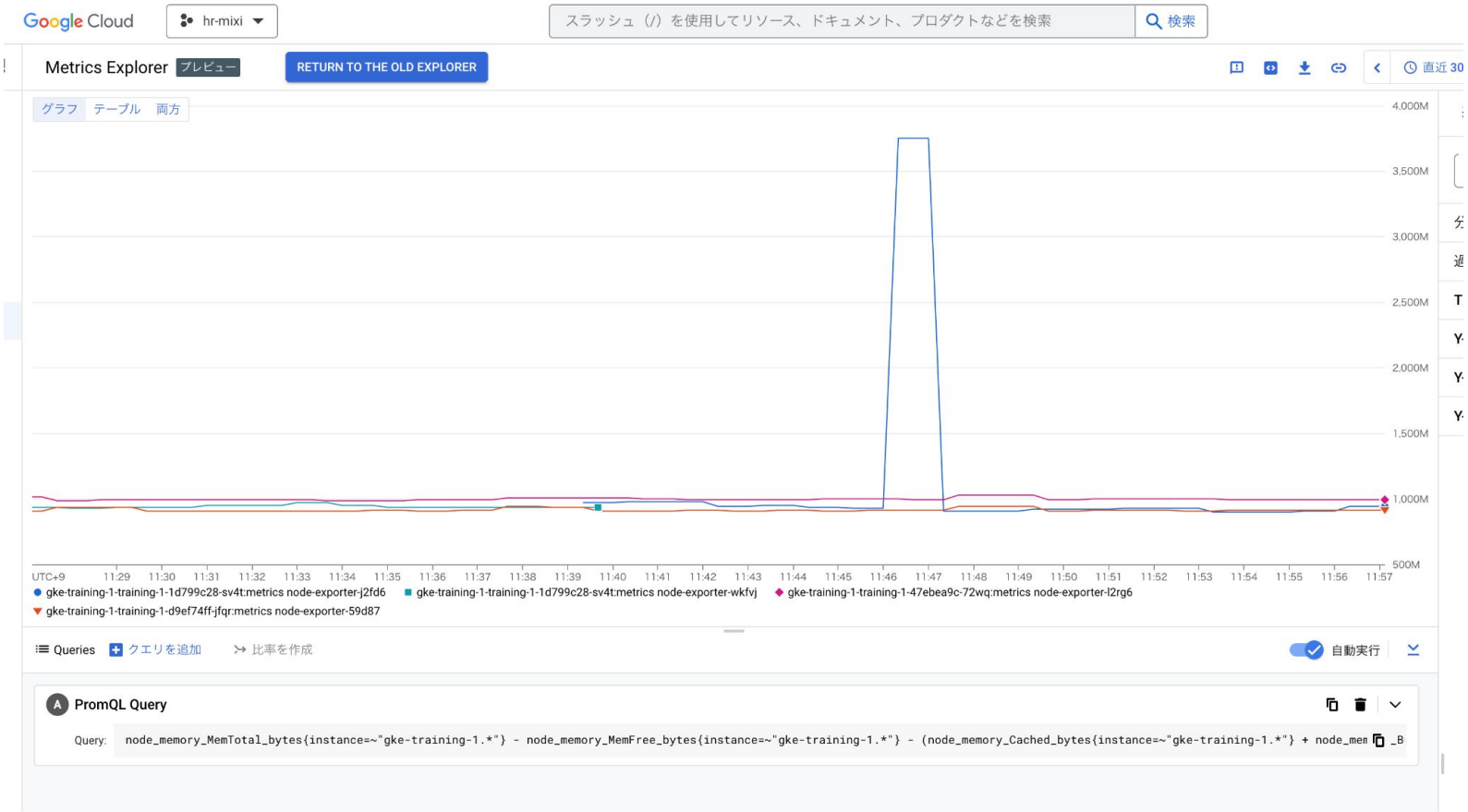
マネージド Prometheus を GKE で使う



実際にデプロイしてみよう

$ kubectl apply -f all-in-one.yaml

$ kubectl get -n gmp-public podmonitoring


Google Cloud
hr-mixi
スラッシュ（/）を使用してリソース、ドキュメント、プロダクトなどを検索
検索
Metrics Explorer
プレビュー
RETURN TO THE OLD EXPLORER
直近 30
グラフ
テーブル
両方
4,000M
3,500M
3,000M
2,500M
2,000M
1,500M
1,000M
500M
UTC+9
11:29 11:30 11:31 11:32 11:33 11:34 11:35 11:36 11:37 11:38 11:39 11:40 11:41 11:42 11:43 11:44 11:45 11:46 11:47 11:48 11:49 11:50 11:51 11:52 11:53 11:54 11:55 11:56 11:57
gke-training-1-training-1-1d799c28-sv4t:metrics node-exporter-j2fd6
gke-training-1-training-1-1d799c28-sv4t:metrics node-exporter-wkfvj
gke-training-1-training-1-47ebea9c-72wq:metrics node-exporter-l2rg6
gke-training-1-training-1-d9ef74ff-jfqr:metrics node-exporter-59d87
Queries
クエリを追加
比率を作成
自動実行
A
PromQL Query
Query:
node_memory_MemTotal_bytes{instance=~"gke-training-1.*"} - node_memory_MemFree_bytes{instance=~"gke-training-1.*"} - (node_memory_Cached_bytes{instance=~"gke-training-1.*"} + node_mem
_B

カスタムコントローラー Managed Prometheus を使って見よう

- サービスのメトリクスやアラートを行う Prometheus というアプリケーションについて理解した
- Google Cloud Managed Service for Prometheus を GKE クラスタで使ってみた
  - ノードのメトリクスを収集する Node Exporter を使って、GKE クラスタのメモリ利用率を Google Cloud Monitoring で見た
- Exercise
  - クラスタのノードのメモリ利用率を取る PromQL を書いてみよう
  - プロダクトのアプリケーションのメトリクスを Prometheus で取る方法を調べてみよう

MIXI

クラスタの様子を見てみよう

metrics-server

- Kubernetes の Pod や Node のメトリクスを収集できる API サーバー
  - https://github.com/kubernetes-sigs/metrics-server
- kube-apiserver には搭載されていないので自前で導入する必要がある
  - Google Kubernetes Engine では標準でインストールされている
  - Cloud Monitoring との連携もオプションで可能（今回は有効済み）
  - オンプレなどではこれを Prometheus などで取得してGrafanaで見る、みたいなことをする
- kubectl top などを叩くと metrics-server からの情報を得られる

MIXI

実際にPodのCPU利用率、メモリ使用率を見てみよう

$ kubectl top pod --all-namespaces

まとめ

- 手元で作ったコンテナを実際にコンテナオーケストレーションツールを使ってデプロイした
- Google Kubernetes Engine (GKE)、Google Cloud Managed Service for Prometheus などを使って Kubernetes クラスタの運用できるようになった