Google Cloud

Kubernetes とエコシステムの継続的なバージョン更新

Tsubasa Nagasawa

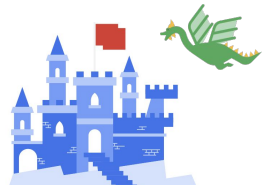
株式会社コロプラ インフラ エンジニア

自己紹介

新卒で SIer に就職。2018 年に CyberAgent に入社し、認証基盤/ポイント サービスの SRE を担当。
2020 年 3 月からコロプラで Kubernetes エンジニアとしてソーシャル ゲームの運用やインフラ基盤の改善に従事しています。

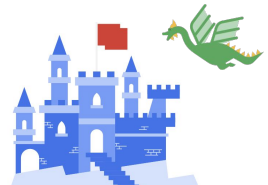
Tsubasa Nagasawa
(@toversus26)
株式会社コロプラ
技術基盤本部
インフラ ストラクチャ部
ゲームインフラ グループ

Google Cloud



本日の内容

- アーキテクチャの紹介と GKE のバージョン更新の戦略
- コロプラと Cloud Native / Kubernetes エコシステム
- GKE と Cloud Native / Kubernetes エコシステム
- Kubernetes への追従



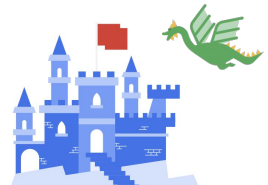
アーキテクチャの紹介と
GKE のバージョン更新の戦略

Google Cloud

コロプラのゲームの特徴

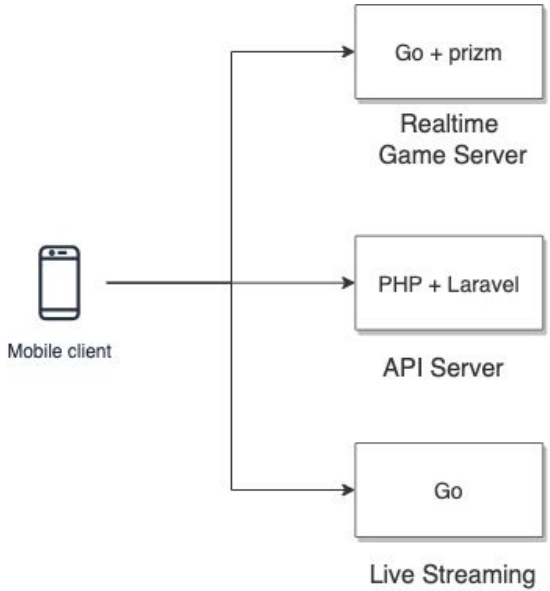
- 最新のテクノロジーと独創的なアイデア
  - 位置情報ゲームやライブ配信など新しい仕組み
- サービス停止を伴う定期メンテナンスを原則行わない
  - サービスを停止せずにアプリの更新や不具合修正
  - ゲームインフラの運用作業も対象
    - GKE クラスタのバージョン更新
    - GKE クラスタの新機能への移行
    - データストアの移行
  - 外部システムのメンテナンスを乗り切る仕組み
    - キューにタスクを積んで非同期処理など

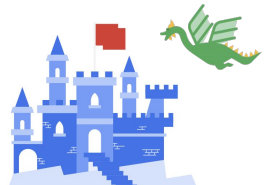
コロプラ



バックエンドの構成要素

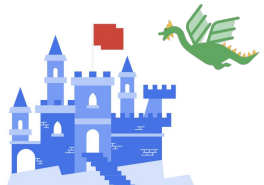
- **API Server**
  - ゲームロジックが実装された REST API サーバー
- **Realtime Game Server**
  - UDP/TCP でメッセージをやり取り
  - 対戦・協力・チャットルーム
  - Agones で Game Server を管理
  - リアルタイム通信の部分は内製フレームワーク（prizm）
- **Live Streaming**
  - 内製の UDP リレーサーバー



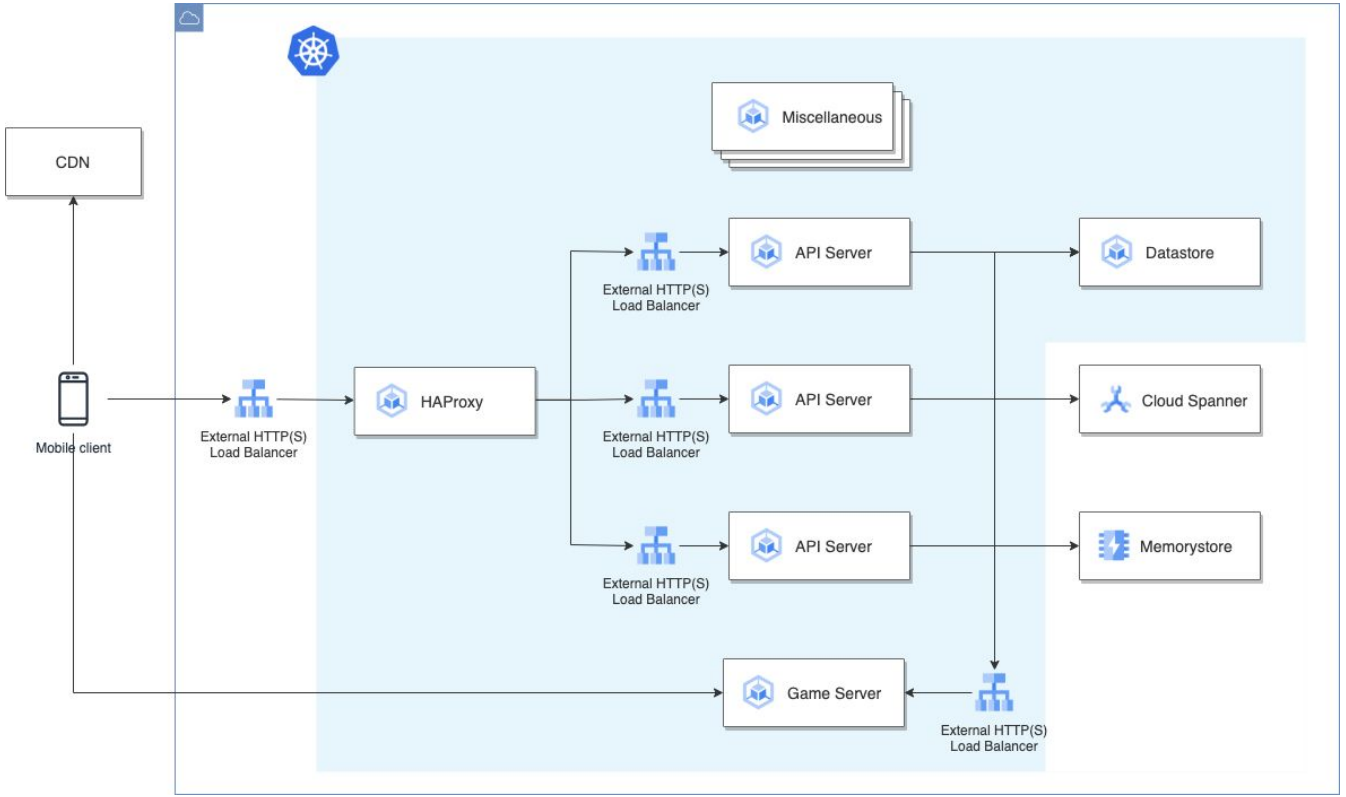


ゲームインフラの特徴

- ゲームタイトル毎に GCP アカウントを分離
  - Infrastructure as Code (IaC) で管理
- 環境がいくつか存在
  - 開発環境
    - 開発者やプランナーが日々利用
  - 検証環境
    - 本番相当の環境
  - 本番環境
    - RC (Release Candidate)
    - Canary



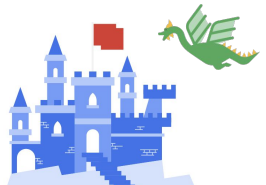
以前のアーキテクチャ


CDN
Mobile client
External HTTP(S)
Load Balancer
HAProxy
Miscellaneous
API Server
API Server
API Server
External HTTP(S)
Load Balancer
External HTTP(S)
Load Balancer
External HTTP(S)
Load Balancer
Datastore
Cloud Spanner
Memorystore
Game Server
External HTTP(S)
Load Balancer

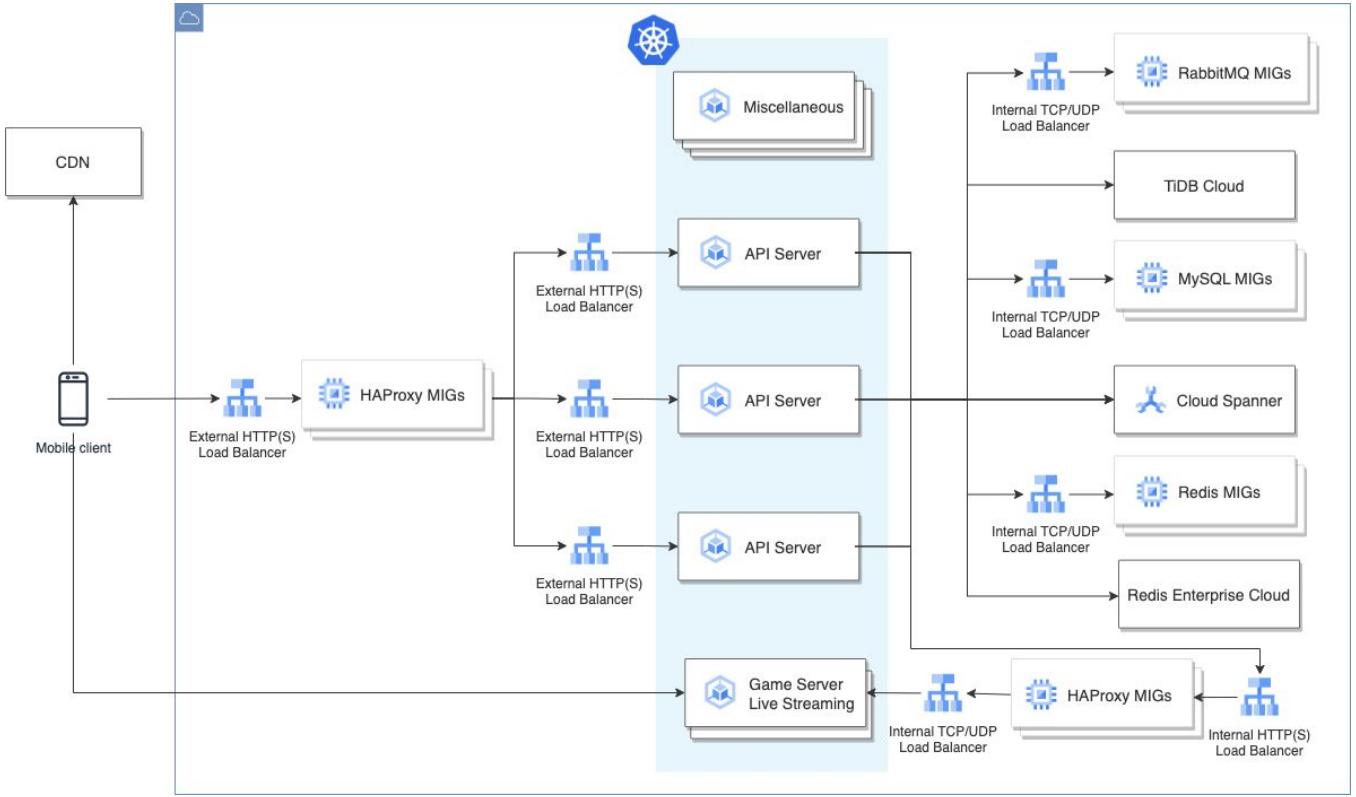
以前のアーキテクチャ

- GKE クラスタのバージョンはクラスタ切り替えで更新
  - 安全に切り替え可能だが工数が掛かる（2 人月程度）
    - HAProxy の DNS 切り替えに時間が掛かる
    - データストアの移行が面倒
- マルチクラスタ間の通信の複雑さ
  - API Server -> データストアのサービス検出が複雑
    - colopl/k8s-local-dns (*1) + colopl/expose-kubedns (*2) で独自実装 (*3)
  - 今は選択肢が増えている (*4)
    - Multi-cluster Services
    - Cloud DNS for GKE

*1: https://github.com/colopl/k8s-local-dns
*2: https://github.com/colopl/expose-kubedns
*3: https://speakerdeck.com/toversus/reliable-and-performant-dns-resolution-with-high-available-nodelocal-dnscache
*4: https://cloud.google.com/kubernetes-engine/docs/concepts/service-discovery#service_discovery_outside_a_single_cluster

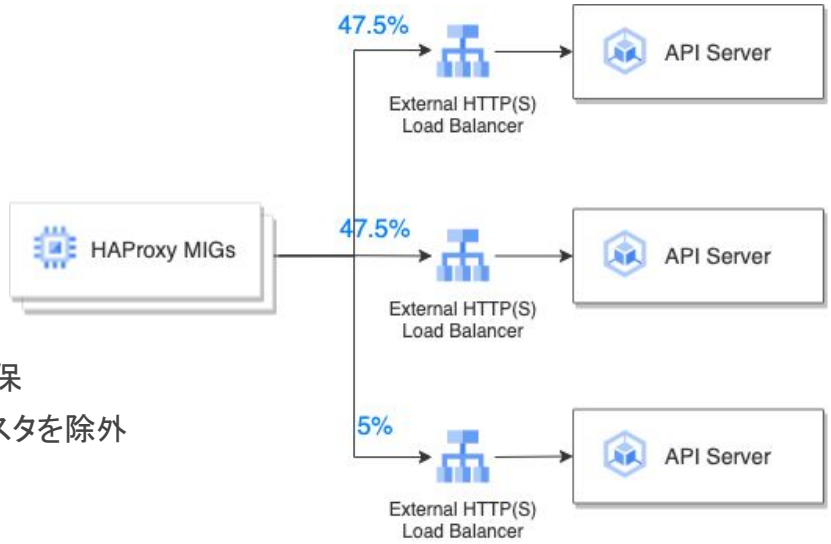


現在のアーキテクチャ


CDN
Mobile client
External HTTP(S) Load Balancer
HAProxy MIGs
External HTTP(S) Load Balancer
External HTTP(S) Load Balancer
External HTTP(S) Load Balancer
Miscellaneous
API Server
API Server
API Server
Internal TCP/UDP Load Balancer
RabbitMQ MIGs
TiDB Cloud
Internal TCP/UDP Load Balancer
MySQL MIGs
Cloud Spanner
Internal TCP/UDP Load Balancer
Redis MIGs
Redis Enterprise Cloud
Game Server Live Streaming
Internal TCP/UDP Load Balancer
HAProxy MIGs
Internal HTTP(S) Load Balancer

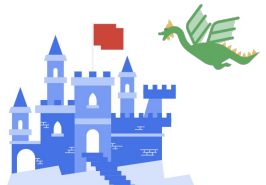
現在のアーキテクチャ

- GKE クラスタのバージョンは in-place 更新
  - 工数を大幅に削減 (2 人日)
  - GKE クラスタを順番に更新
    - GKE 更新の Workflow を実行
    - 流量の少ないクラスタから更新して安全性を担保
    - 問題が起きたら HAProxy 配下から手動でクラスタを除外
- HAProxy (Global Load Balancer) を GKE 外に移動
- マルチクラスタ間の通信の複雑さを排除
  - データストアを GKE 外に移動
    - マネージド サービスを積極的に活用



GKE 更新のタイミング

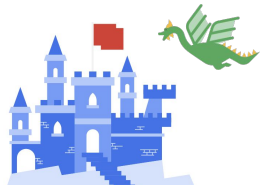
- No channel を利用
  - Stable channel のデフォルト バージョンの更新サイクルに従う
    - デフォルト バージョンが一番安定
  - クラスタの更新順序を制御したい
    - 流量の少ないクラスタから順番に更新したい
  - ノードプールの自動更新を無効化したい
    - コントロール プレーンのマイナー バージョン更新のタイミングで上げる
    - パッチバージョンは更新しない
      - CVE 対応や影響の大きなバグ修正は例外
  - コントロール プレーンは自動でどんどん上がる
    - ノードプールとのパッチ バージョンの差異は目を瞑る



GKE 更新

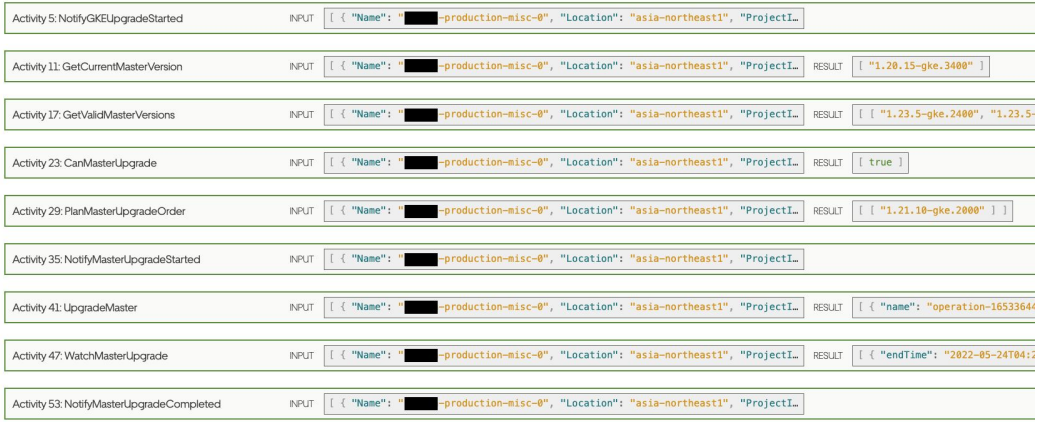
- Temporal (*1)
  - Uber の Cadence からスピンオフ
  - 耐障害性のある状態を持った Workflow エンジン
    - Amazon Simple Workflow, Azure Durable Functions の元開発者
  - ユースケース
    - Netflix が社内の Workflow 基盤として提供 (*2)
      - Spinnaker を Temporal ベースで書き換え
    - Hashicorp が Terraform Cloud のインフラ プロビジョニングに使用 (*3)
      - Temporal の Workflow の中で Terraform を実行
    - Datadog がデータベースの運用作業をコード化して実行 (*4)

*1: https://temporal.io/
*2: https://www.youtube.com/watch?v=LIiBP7YMGyA
*3: https://www.youtube.com/watch?v=vOoPxs9NHgc
*4: https://docs.temporal.io/blog/how-datadog-ensures-database-reliability-with-temporal

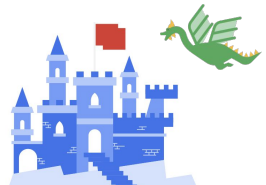


GKE 更新

- GKE 更新の Workflow を Temporal + Go SDK で実装
  - コントロール プレーン
    - バージョン更新
    - 更新完了待ち
  - ノードプール
    - Max surge を割合から動的計算
    - 自動修復機能の無効化
    - Surge upgrade の実行
    - Surge upgrade の完了待ち
    - 自動修復の有効化



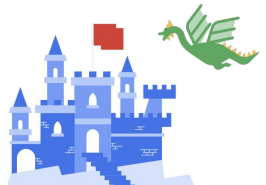
- 耐障害性があるので自身が動いている GKE クラスタも更新可能



GKE 新機能への対応

- GKE Dataplane v2
  - Cilium によるコンテナ ネットワークの置き換え
  - eBPF ベースのルーティング、ネットワーク ポリシーとロギング
  - さようなら kube-proxy
  - 今後の Cilium ベースの機能拡張への期待
    - Hubble (*1) + Tetragon (*2) 連携
    - Sidecar-free eBPF Service Mesh (*3)
  - クラスタ作成時のみ有効化可能
    - 既存クラスタからの切り替えが必須

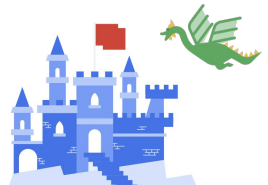
*1: https://github.com/cilium/hubble
*2: https://github.com/cilium/tetragon
*3: https://www.youtube.com/watch?v=mpwTkm53YTY



GKE 新機能への対応

- クラスタ切り替えの仕組みは必要
  - DNS ベース
  - Global Load Balancer
    - コロプラの場合は HAProxy MIGs
  - Multi-cluster Ingress (*1)
  - Multi-cluster Gateways (*2)
  - Traffic Director service routing APIs (*3)

*1: https://cloud.google.com/kubernetes-engine/docs/concepts/multi-cluster-ingress
*2: https://cloud.google.com/kubernetes-engine/docs/how-to/deploying-multi-cluster-gateways
*3: https://cloud.google.com/traffic-director/docs/service-routing-overview



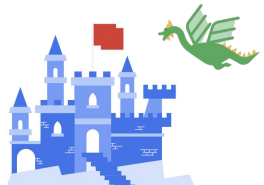
コロプラと
Cloud Native / Kubernetes エコシステム

Google Cloud

コロプラが利用しているエコシステム (*1)

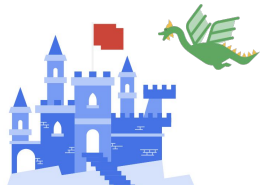
- Agones
- Argo CD
- Berglas
- ChartMuseum
- Datadog (APM)
- Dex
- ExternalDNS
- Falco
- Fluentd
- GitLab
- Grafana
- Helm
- Helmfile
- kube-state-metrics
- Nginx
- Prometheus stack
- Spinnaker
- Temporal (Cadence)
- Terraform
- Victoria Metrics

*1: アーキテクチャ図に明記したもの以外



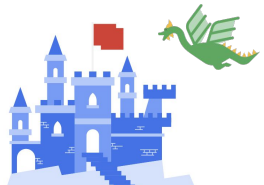
エコシステムから提供されているコンテナ イメージを

GKE 上でどう使っていますか？



エコシステムのコンテナ イメージ

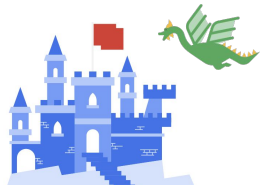
- コンテナ レジストリの使用状況
  - 全体的に DockerHub からの配布が根強い
  - Kubernetes SIG (Special Interest Group) 関連は GCR が多い
  - Quay や GitHub Container Registry から提供もちらほら
- なぜ気にするのか？
  - 外部コンテナ レジストリからのプル
    - プライベート クラスタだと Cloud NAT が必須
  - DockerHub のレート制限
    - kubelet は匿名ユーザーでイメージをプルする
      - 100 pulls per 6 hours per IP address



外部コンテナ レジストリのイメージ同期

- mirror.gcr.io (*1) の存在に依存
  - latest のタグしか使えない
  - 使用頻度の低いイメージはキャッシュから消える可能性がある
    - 開発環境で参照していた Debian イメージが消えたことがある
- Docker Registry Synchronizer (*2)
  - Python で書かれたスクリプトを定期実行して同期
    - どのイメージが更新されたか分からない
- dregsy - Docker Registry Sync (*3)
  - Golang で書かれた定期実行 or デーモンとして新規イメージタグを検知
    - イメージの同期処理は Docker デーモンや skopeo (*4) に投げる

*1: https://cloud.google.com/container-registry/docs/pulling-cached-images
*2: https://github.com/AliyunContainerService/sync-repo
*3: https://github.com/xelalexv/dregsy
*4: https://github.com/containers/skopeo



外部コンテナ レジストリのイメージ同期

- GitHub + Dependabot + Cloud Build で自動同期
  - 同期したい Docker イメージをモノレポで管理
  - Dependabot で Dockerfile のベースイメージのタグ更新を検知
    - 最新のバージョンを追従
    - Node.js のように LTS が最新でない場合
      - ignore (*1) を設定して頑張る
  - Dependabot が PR を作成
    - 特定のブランチ名で PR を作成
      - dependabot/docker/argoproj/argocd/argoproj/argocd-v2.3.4

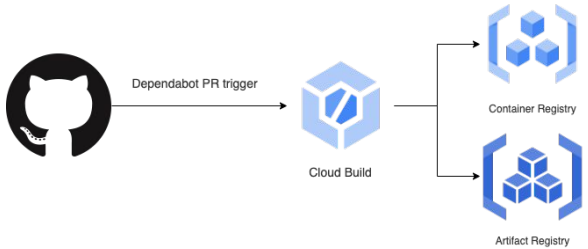
```
❯ tree -a
.
├── .dependabot
│   └── config.yml
├── .cloudbuild.yaml
├── .editorconfig
├── .dockerignore
├── argoproj
│   └── argocd
│       └── Dockerfile
└── chartmuseum
    └── chartmuseum
        └── Dockerfile
  (...)
```

```
❯ cat argoproj/argocd/Dockerfile

FROM argoproj/argocd:v2.3.4
```

*1:
https://docs.github.com/en/code-security/dependabot/dependabot-version-updates/configuration-options-for-the-dependabot.yml-file#ignore

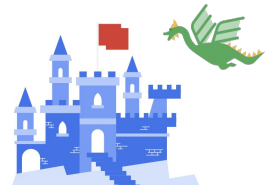
外部コンテナ レジストリのイメージ同期

- GitHub + Dependabot + Cloud Build
  - PR 作成をトリガーとして Cloud Build を実行
    - スクリプトで同期対象のイメージ名を特定
      - ブランチ名から Dockerfile のパスを特定
      - Dockerfile の中をパースして対象のイメージ名とタグを特定
        - argoproj/argocd:v2.3.4
    - crane (*1) で GCR / Artifact Registry にイメージをコピー
      - マルチ アーキテクチャ対応したイメージを同期可能




```
❯ tree -a
.
├── .dependabot
│   └── config.yml
├── .cloudbuild.yaml
├── .editorconfig
├── .dockerignore
├── argoproj
│   └── argocd
│       └── Dockerfile
└── chartmuseum
    └── chartmuseum
        └── Dockerfile
  (...)
```

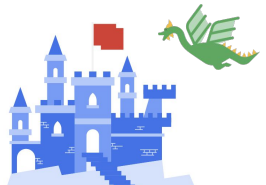
```
❯ cat argoproj/argocd/Dockerfile

FROM argoproj/argocd:v2.3.3
```



*1: https://github.com/google/go-containerregistry/tree/main/cmd/crane

エコシステムへの追従

- コンテナ イメージの更新駆動
  - 週に 1 回チーム内でリリースノートの読み合わせ
    - 脆弱性対応が必要か
    - 新機能が使えそうか
    - バグ修正の恩恵を受けられそうか
  - Helm chart の更新と動作確認
  - 各ゲームタイトルに横展開
    - 脆弱性修正以外は更新のタイミングを担当に任せる
    - 運用中のタイトルは GKE 更新のタイミングでやることが多い



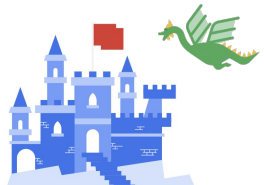
GKE と
Cloud Native / Kubernetes エコシステム

Google Cloud

GKE のコントロールプレーンで動作

- v1.21.10-gke.2000 を想定
- クラスタで有効化している機能によって一部異なる

- Cluster Autoscaler
  - https://github.com/kubernetes/autoscaler/tree/cluster-autoscaler-1.21.0
- Common Webhooks
- etcd
  - https://github.com/etcd-io/etcd/tree/v3.4.13
- GLBC (ingress-gce)
  - https://github.com/kubernetes/ingress-gce/tree/v1.13.4/cmd/glbc
- Konnectivity server
  - https://github.com/kubernetes-sigs/apiserver-network-proxy/tree/v0.0.24/cmd/server
- kube-addon-manager
  - https://github.com/kubernetes/kubernetes/tree/v1.21.10/cluster/addons/addon-manager
- kube-api-server
- kube-scheduler
- kube-controller-manager

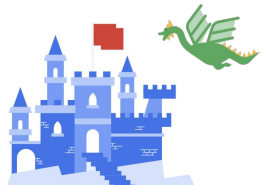


GKE のノード上で動作 or バイナリ提供

- v1.21.10-gke.2000 を想定
- クラスタで有効化している機能によって一部異なる

- cAdvisor (kubelet 同梱)
  - https://github.com/google/cadvisor/tree/v0.39.3
- CNI network plugins
  - https://github.com/containernetworking/plugins/tree/v1.0.1
- containerd
  - https://github.com/containerd/containerd/tree/v1.4.11
- containerd-shim-runc-v2
  - https://github.com/containerd/containerd/tree/v1.4.11/cmd/containerd-shim-runc-v2
- containerized_mounter (*1)
  - https://github.com/kubernetes/kubernetes/tree/v1.21.10/cluster/gce/gci/mounter
- crictl (cri-tools)
  - https://github.com/kubernetes-sigs/cri-tools/tree/v1.20.0
- ctr
  - https://github.com/containerd/containerd/tree/v1.4.11/cmd/ctr

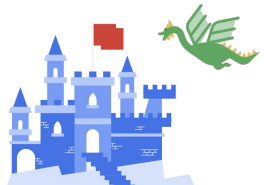
*1:
https://github.com/kubernetes/design-proposals-archive/blob/main/storage/containerized-mounter.md



GKE のノード上で動作 or バイナリ提供

- v1.21.10-gke.2000 を想定
- クラスタで有効化している機能によって一部異なる

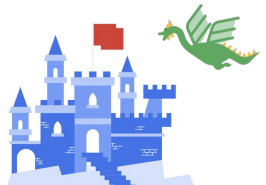
- gke-exec-auth-plugin
  - https://github.com/kubernetes/cloud-provider-gcp/tree/providers/v0.21.0/cmd/gke-exec-auth-plugin
- kubectl
- kubelet
- Node Problem Detector
  - https://github.com/kubernetes/node-problem-detector/tree/v0.8.7
- runc
  - https://github.com/opencontainers/runc/tree/v1.0.0-rc95



GKE の addons として動作

- v1.21.10-gke.2000 を想定
- クラスタで有効化している機能によって一部異なる

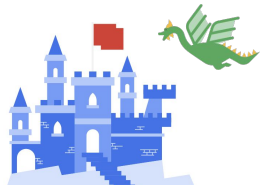
- event-exporter
  - https://github.com/GoogleCloudPlatform/k8s-stackdriver/tree/event-exporter-v0.3.5
- Fluent Bit
  - https://github.com/fluent/fluent-bit/tree/v1.5.7
- GCE Persistent Disk CSI Driver
  - https://github.com/kubernetes-sigs/gcp-compute-persistent-disk-csi-driver/tree/v1.3.6
- gke-metadata-server
- gke-metrics-agent (OpenTelemetry Collector)
- kube-proxy (static Pods)
  - https://github.com/kubernetes/kubernetes/tree/v1.20.10/cmd/kube-proxy
- kube-dns (Kubernetes DNS)
  - https://github.com/kubernetes/dns/tree/1.21.0
- kube-dns-autoscaler (Cluster Proportional Autoscaler)
  - https://github.com/kubernetes-sigs/cluster-proportional-autoscaler/tree/1.8.1



GKE の addons として動作

- v1.21.10-gke.2000 を想定
- クラスタで有効化している機能によって一部異なる

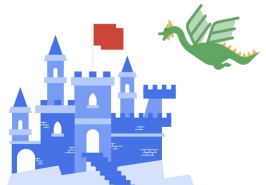
- Kubernetes Metrics Server
  - https://github.com/kubernetes-sigs/metrics-server/tree/v0.4.5
- Kubernetes Metrics Server sidecar (Addon-resizer)
  - https://github.com/kubernetes/autoscaler/tree/addon-resizer-1.8.13
- Konnectivity agent
  - https://github.com/kubernetes-sigs/apiserver-network-proxy/tree/v0.0.24/cmd/agent
- l7-default-backend (404-server-with-metrics)
  - https://github.com/kubernetes/ingress-gce/tree/v1.13.4/cmd/404-server-with-metrics
- netd
  - https://github.com/GoogleCloudPlatform/netd/tree/v0.2.13
- Node Driver Register
  - https://github.com/kubernetes-csi/node-driver-registrar/tree/v2.1.0
- prometheus-to-sd
  - https://github.com/GoogleCloudPlatform/k8s-stackdriver/tree/prom-to-sd-v0.10.0



GKE との付き合い方

- コントロール プレーン更新の影響
  - リージョン クラスタ (*1) だと更新中にサービス影響はない
  - kube-addon-manager で管理しているコンポーネントの更新
    - kube-system の namespace にいる GKE 管理のシステムの Pod が対象
    - 新しくコンポーネントが追加されたり削除されたりもする
  - マイナー バージョン更新後に API が削除されることがある
    - GKE 1.22 で削除される API 一覧 (*2)
    - 削除されても既存のオブジェクトに影響はない (*3)
      - 削除された API で新規にオブジェクトが作れなくなる
      - 削除された API で既存のオブジェクトを更新できなくなる

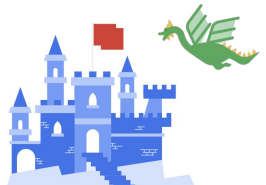
*1: https://cloud.google.com/kubernetes-engine/docs/concepts/regional-clusters
*2: https://cloud.google.com/kubernetes-engine/docs/deprecations/apis-1-22
*3: https://github.com/kubernetes/community/blob/master/contributors/devel/sig-architecture/api_changes.md



GKE との付き合い方

- ノードプール更新の影響
  - ノード上で動作している Pod が必ず再起動する
  - ノードのイメージに大きな変更がある場合は注意
    - GKE 1.20 で Ubuntu 18.04 から 20.04 に更新
      - NPD Service doesn't start properly on Ubuntu based GKE Nodes version 1.20.* and 1.21.* (*1)
    - Ubuntu のバージョンは GKE リリースノート (*2) で共有されなくなった
    - COS のバージョンと変更点は GKE リリースノートから確認可能

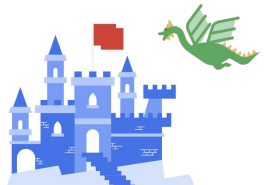
*1: https://issuetracker.google.com/issues/202338644
*2: https://cloud.google.com/kubernetes-engine/docs/release-notes



GKE との付き合い方

- GKE で利用しているエコシステムは GKE が管理
  - 問題が起きたらサポートに正確に報告する
    - 既知の問題か確認
      - Google IssueTracker (*1)
      - GCP のケース起票画面から見れる既知の問題 (*2)
      - GKE リリースノート (*3)
      - ドキュメントの既知の問題のセクション
      - Kubernetes や関連するエコシステムの Issue / PR
    - ログと再現手順を必ず添付
    - 既知の問題として追加してもらう

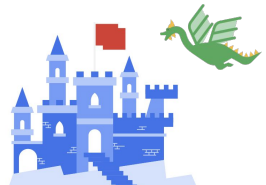
*1: https://issuetracker.google.com/issues
*2: https://cloud.google.com/support/docs/manage-cases#viewing_known_issues_with_services
*3: https://cloud.google.com/kubernetes-engine/docs/release-notes



Dataplane v2 の Pod IP リーク問題

- Pod が ContainerCreating 状態でスタック
- PodSandbox の作成に失敗
  - kubelet が Pod Sandbox の作成に失敗
  - Pod IP の割り当て可能範囲 (10.0.0.64/26) に空きがない

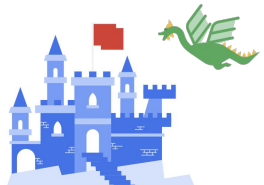
```
❯ kubectl describe pods -n kube-system metrics-server-v0.4.4-5698c9f6fc-sptpn
Name:                 metrics-server-v0.4.4-5698c9f6fc-sptpn
Namespace:            kube-system
(...)
Events:
 Type     Reason                  Age                   From                Message
 ----     ------                  ----                  ----                -------
 Warning  FailedCreatePodSandBox  81s (x722 over 158m)  kubelet, dummy-node  (combined from similar
events): Failed to create pod sandbox: rpc error: code = Unknown desc = failed to setup network for
sandbox "6a2d08a2a02fc8e491f9237c598b0d505e56eb7731d757f4d0c1d62eeee3012c": failed to allocate for range
0: no IP addresses available in range set: 10.0.0.65-10.0.0.126
```



Dataplane v2 の Pod IP リーク問題

- kubelet のログを追う
  - logName="projects/<project id>/logs/kubelet"
  - resource.labels.node_name="dummy-node"

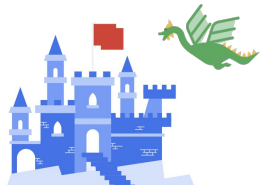
```
E0209 00:03:12.758449    2377 kuberuntime_sandbox.go:68] \"Failed to create sandbox for pod\" err=\"rpc error:
code = Unknown desc = failed to setup network for sandbox
\\\"372ea79037673d222c21d600302c83fbe139d4ba82adf61699aad8c102011ab4\\\": failed to find plugin
\\\"cilium-cni\\\" in path [/home/kubernetes/bin]\" pod=\"kube-system/metrics-server-v0.4.4-5698c9f6fc-sptpn\"
(...)
I0209 00:03:12.553946    2377 kuberuntime_manager.go:466] \"No sandbox for pod can be found. Need to start a
new one\" pod=\"kube-system/metrics-server-v0.4.4-5698c9f6fc-sptpn\"
(...)
E0209 00:03:33.803332    2377 kuberuntime_sandbox.go:68] \"Failed to create sandbox for pod\" err=\"rpc error:
code = Unknown desc = failed to setup network for sandbox
\\\"6a2d08a2a02fc8e491f9237c598b0d505e56eb7731d757f4d0c1d62eeee3012c\\\": failed to allocate for range 0: no IP
addresses available in range set: 10.0.0.65-10.0.0.126\"
pod=\"kube-system/metrics-server-v0.4.4-5698c9f6fc-sptpn\"
```



Dataplane v2 の Pod IP リーク問題

- container runtime (containerd) のログを追う
  - logName="projects/<project id>/logs/container-runtime"
  - resource.labels.node_name="dummy-node"

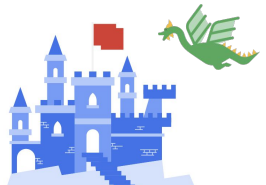
```
time=\"2022-02-09T00:03:12.752015398Z\" level=error msg=\"Failed to destroy network for sandbox
\\\"372ea79037673d222c21d600302c83fbe139d4ba82adf61699aad8c102011ab4\\\"\" error=\"failed to find plugin
\\\"cilium-cni\\\" in path [/home/kubernetes/bin]\"
(...)
time=\"2022-02-09T00:03:12.755882291Z\" level=error msg=\"RunPodSandbox for
&PodSandboxMetadata{Name:metrics-server-v0.4.4-5698c9f6fc-sptpn,Uid:935de484-be1b-4e9a-8681-8331f469d8cd,N
amespace:kube-system,Attempt:0,} failed, error\" error=\"failed to setup network for sandbox
\\\"372ea79037673d222c21d600302c83fbe139d4ba82adf61699aad8c102011ab4\\\": failed to find plugin
\\\"cilium-cni\\\" in path [/home/kubernetes/bin]\"
```



Dataplane v2 の Pod IP リーク問題

- Pod が余り載っていないノードで Pod IP が枯渇前に起動できることも確認
  - Endpoints 作成のレート制限に引っ掛かり起動が遅れる

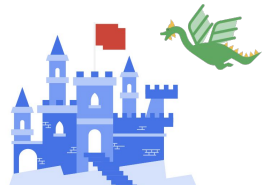
```
❯ kubectl describe pods -n default nginx
Name:                 nginx
Namespace:            default
(...)
Events:
 Type     Reason                  Age                  From                 Message
 ----     ------                  ----                 ----                 -------
 Warning  FailedCreatePodSandBox  5m25s                 kubelet, dummy-node  Failed to create pod
sandbox: rpc error: code = Unknown desc = failed to setup network for sandbox
"5be5d50aa4b186fb0a5c232968cd43996dc3ed7e77cba669b4e1c09a66784903": failed to find plugin "cilium-cni"
in path [/home/kubernetes/bin]
(...)
 Warning  FailedCreatePodSandBox  4m38s                 kubelet, dummy-node  Failed to create pod
sandbox: rpc error: code = Unknown desc = failed to setup network for sandbox
"0f654d461f604a0780b96be6a80d03b05f265eb5a07c64e1330c3f048366d9ed": unable to create endpoint: [PUT
/endpoint/{id}][429] putEndpointIdTooManyRequests
(...)
 Normal   Started                 31s                  kubelet, dummy-node  Started container
controller
```



Dataplane v2 の Pod IP リーク問題

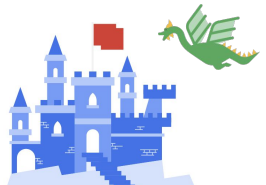
- CNI と GKE の実装の理解 (*1)
  - CNI は複数の plugin を組み合わせ (chain) 可能
  - netd が CNI の設定ファイルをホスト上に配置
  - GKE の IPAM (IP Address Management) は host-local
    - Node IPAM controller がノードに割り当てた Pod CIDR (10.0.0.64/26) から払い出し
- Dataplane v2 を有効化した時の挙動の変化
  - netd が CNI の設定ファイルに Cilium CNI の chain を追加して配置
  - anetd (cilium-agent) が cilium-cni のバイナリをホスト上に配置
    - PostStart Hook で cni-install.sh (*2) を実行
    - netd が CNI の設定を追加してから時間差がそれなりにある

*1: https://qiita.com/toVersus/items/4ff2525d562d8de4d530
*2: https://github.com/cilium/cilium/blob/v1.10.3/plugins/cilium-cni/cni-install.sh



Dataplane v2 の Pod IP リーク問題

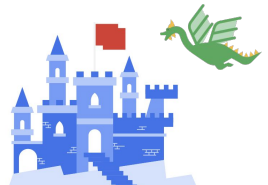
- 情報の整理とサポートケースの作成
  - Dataplane v2 を有効にしたクラスタでのみ発生
  - ノードの台数を 0 台からスケールさせた場合に発生しやすい
    - ノード作成直後に多くの Pod が起動する状態
    - ノードに Pod がそれほど載っていない場合は時間を置いて起動できる
  - 対象のノードの kubelet と container-runtime のログを添付
  - PodSandbox の作成に失敗した後に IPAM が払い出した IP を回収できてなさそう
    - Sandbox ネットワークの作成と削除に失敗
    - Sandbox を新たに作ろうとして延々と失敗して IP が枯渇



Dataplane v2 の Pod IP リーク問題

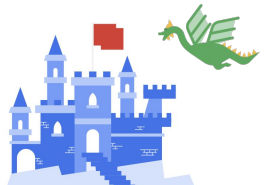
- containerd で Pod IP がリークする問題 (*1)
  - サポートから共有を受けた
  - Dataplane v2 (cilium-cni) に関係なく前から報告のあった問題
    - Pod ネットワークの初期化でタイムアウトする場合のみ解決済み
    - 他のエラー (e.g. cilium-cni のバイナリが存在しない) が発生した場合は未解決
- Dataplane v2 の既知の問題のセクションに追加して貰った (*2)
  - すぐに解決が難しそうだった
    - 他のユーザーへの共有
    - 解決したかを追跡したい

*1: https://github.com/containerd/containerd/issues/5768
*2: https://cloud.google.com/kubernetes-engine/docs/how-to/dataplane-v2#containerd-pod-ip-leak



Dataplane v2 の Pod IP リーク問題

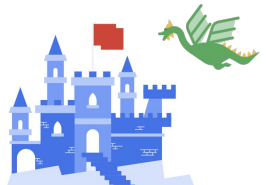
- Dataplane v2 の既知の問題のセクションで解決済みに
  - 以下のバージョンで修正済み
    - 1.23.4-gke.1600 or later
    - 1.22.8-gke.200 or later
    - 1.21.11-gke.1100 or later
    - 1.20.15-gke.5200 or later
  - 1.21.11-gke.1100 に上げて問題が発生しなくなったことを確認
    - cilium-cni のバイナリが見つからないエラーが出なくなった



Dataplane v2 の Pod IP リーク問題

- GKE はどう問題を解決したか？
  - containerd の IP 枯渇問題は根深く解決できていない
    - Google の人が PR を投げてはいるが議論中 (*1)
    - GKE 側が個別にパッチを適用した？
      - containerd のバージョンは COS のイメージに依存
      - バイナリのバージョンを確認したところ変更は見られず
  - cilium-cni のバイナリを事前に配置した？
    - anetd にも独自パッチの形跡が見られない
  - netd で cilium-cni 用の chain の設定追加を遅らせた (*2)
    - cilium-cni のバイナリが存在することが確認できてから設定を追加

*1: https://github.com/containerd/containerd/pull/5904
*2: https://github.com/GoogleCloudPlatform/netd/pull/121



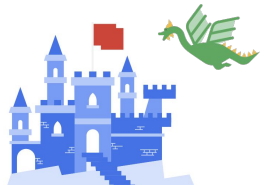
Kubernetes への追従

Kubernetes との付き合い方

- Kubernetes の大きな変更は事前に対応
  - 廃止予定の API の移行
    - マニフェストの更新
    - コントローラーなどが廃止予定の API を呼び出していないか確認 (*1)
    - マイナー バージョン更新の保護機能 (*2)
      - 完全に削除される API 呼び出しが過去 30 日間あると自動更新が止まる
  - デフォルトの挙動の変更
    - 新機能がベータに移行して有効化 (Kubernetes < 1.24)
    - バグ修正による挙動の変更

*1: https://cloud.google.com/kubernetes-engine/docs/deprecations/apis-1-22#locating_api_clients_writing_to_deprecated_apis

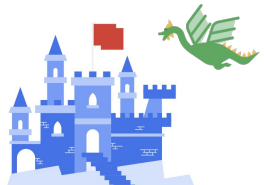
*2: https://cloud.google.com/kubernetes-engine/docs/deprecations#how_kubernetes_deprecations_work_with



Kubernetes との付き合い方

- Kubernetes (upstream) の変更を追いかけておくと楽
  - 全部追っていたら時間が足りないので自分の嗅覚を信じる
    - kubernetes/kubernetes
    - kubernetes/enhancements
  - リグレッションや潜在的なバグ報告の把握
    - リリース直後より利用者が増えたタイミングで報告が増える
  - 大きめのリファクタリングが入った後は危険
    - KEP 3138: Increasing Reliability Bar (*1)
      - テストの品質や網羅率の悪さを改善
        - 安全に機能追加やリファクタリングができるように
      - 新機能よりも安定性が大事

*1: https://github.com/kubernetes/enhancements/pull/3139



client-go / kubectl の認証方法の変更

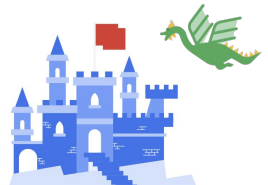
- KEP 541 : External credential providers (*1)
  - Kubernetes 1.22 で GA
  - client-go からクラウドプロバイダー固有の認証の仕組みを削除
    - クラウドプロバイダーが認証 plugin を提供
    - ExecCredential (*2) に指定して認証
  - Kubernetes 1.25 で gcp と azure の in-tree の認証機能が削除
    - GKE 1.25 までに kubectl 利用者は移行が必要 (*3)
      - gke-gcloud-auth-plugin (*4) を利用

*1: https://github.com/kubernetes/enhancements/tree/master/keps/sig-auth/541-external-credential-providers

*2: https://kubernetes.io/docs/reference/access-authn-authz/authentication/#configuration

*3: https://cloud.google.com/blog/products/containers-kubernetes/kubectl-auth-changes-in-gke

*4: https://github.com/kubernetes/cloud-provider-gcp/tree/master/cmd/gke-gcloud-auth-plugin

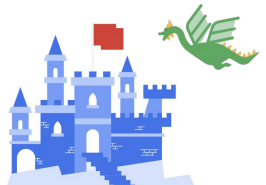


Kubernetes Service Account の仕様変更

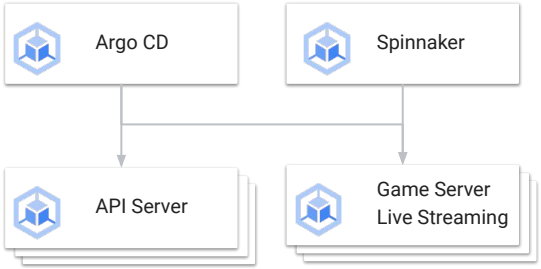
- KEP 1205 : Bound Service Access Token (*1)
  - Kubernetes 1.21 で Beta -> 1.22 で GA
  - 既存の Service Account Token の問題の解消
    - Service Account を作成すると Secret に JWT トークンが必ず保存される
    - JWT トークンにスコープが設定されていないので誰でも使える
    - JWT トークンに有効期限がない
- KEP 2799 : Reduction of Secret-based Service Account Tokens (*2)
  - Kubernetes 1.24 で Service Account Token の自動生成がデフォルトで無効化
    - TokenRequest API 経由で期限付きのトークンを手動生成する必要がある
      - kubectl create token <sa name> --namespace <namespace>
  - 将来的に Pod にマウントされていない自動生成されたトークンを削除

*1: https://github.com/kubernetes/enhancements/tree/master/keps/sig-auth/1205-bound-service-account-tokens

*2: https://github.com/kubernetes/enhancements/tree/master/keps/sig-auth/2799-reduction-of-secret-based-service-account-token

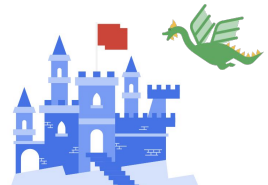


Kubernetes Service Account の仕様変更



- 何が問題か？
  - CI/CD の過程で Argo CD や Spinnaker を利用
    - マルチクラスタ構成のため外部クラスタのリソースを管理
    - cluster-admin 権限を紐付けた Service Account Token を読み込ませて使用
  - Secret ベースの Service Account Token が自動的に発行されなくなる
    - Secret ベースの無期限のトークンも手動発行可能 (*1) だが...
      - トークンがローテーションされないのはセキュリティ的に問題
      - 非推奨なため、今後削除される可能性もある
  - Bound Service Account Token を使う場合
    - クラスタを跨いで自動的にトークンをローテーションできない
    - 自己署名証明書のように有効期限を長めに設定して使いたくない

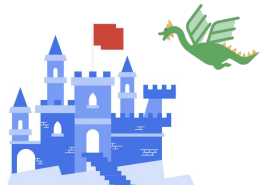
*1:
https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/#manually-create-a-service-account-api-token



Argo CD に対応を入れた話

- Argo CD のイメージの中に GKE 用の creadential plugin を追加 (*1)
  - Argo CD v2.4.0 から利用可能
  - カスタム イメージが不要に
    - gcloud や gke-gcloud-auth-plugin を upstream のイメージに同梱し辛い
      - イメージのサイズが増える
      - バイナリのバージョン管理が面倒
    - argocd-k8s-auth の仕組みに相乗り
      - AWS 以外のクラウドプロバイダーもサポートしやすい仕組みが最近入った
      - credential plugin の再実装
      - ライブラリとして吸収することで上記の問題を軽減
  - GCP Service Account の権限でリソースを操作可能
    - Workload Identity と組み合わせればサービス アカウントキーの発行も不要
    - ClusterRole で必要な権限に絞ることも可能

*1:
https://github.com/argoproj/argo-cd/pull/9190



Google Cloud

Thank you