KubeFest Tokyo 2020

FUJITSU
shaping tomorrow with you

Kubernetesトラブル原因特定 容易化に向けたロギング強化機能

2020/6/13

富士通株式会社 Linuxソフトウェア事業部
長谷 仁
小林 大介

Copyright 2020 FUJITSU LIMITED

アジェンダ

FUJITSU

- ロギング機能の概要
- ロギング機能の課題
- 解決策
- K8sコミュニティでの開発状況
- 残課題/今後の予定

Copyright 2020 FUJITSU LIMITED

自己紹介

FUJITSU

長谷 仁

- ハード/BIOS/OS/仮想化層の性能解析、チューニングが専門
  富士通サーバで仮想系ベンチマークで複数の性能世界記録を達成
- 2018末からK8s(AWSのEKS)を活用したシステム構築を担当
- 2019末からOSS開発チームへ、K8ロギング強化機能に取組み中

小林 大介

- OSSコミュニティで開発者として活動
- 昨年からK8sの開発に携わり、CKA取得やコミュニティへのコントリビュートの他、ロギング強化機能の共同開発者として活動
- 現在はK8s本体の他に、周辺のOSSの社内運用にも着手中

Copy right 2020 FUJITSU LIMITED

アジェンダ

FUJITSU

- ロギング機能の概要
- ロギング機能の課題
- 解決策
- K8sコミュニティでの開発状況
- 残課題/今後の予定

Copyright 2020 FUJITSU LIMITED

ロギングとは

FUJITSU

- システムで発生した事象を特定の形式で時系列で記録すること
  - 利用者/運用者の操作履歴
  - ソフト(OS、アプリなど)やハード(サーバー、通信機器など)の稼働状況
  - 通信記録など
- ロギングの重要性
  - 利用状況の把握、運用管理に使用
  - **トラブル(障害、不正アクセスなど)発生時の調査に使用**
    (エラーログ、アクセスログ、オペレーションログなどから原因を突き止める)

本セッションではトラブルシューティング観点でK8sのロギング機能を説明します。

Copyright 2020 FUJITSU LIMITED

Kubernetesにおけるロギング機能

FUJITSU

- K8sでは主に以下の種類のログを記録

ログの種類	概要	主な用途
Podログ	Pod上で動作するアプリの標準出力/エラー出力	アプリ側のトラブルシューティング
イベントログ	K8sクラスタで発生したイベント	K8sクラスタ環境側のトラブルシューティング
コンポーネントログ	K8sコンポーネントごとの個別ログ(kubelet、kube-apiserverなど)	イベントログより詳細な内部動作を追う場合に利用
Auditログ	kube-apiserverとやりとりするHTTPリクエスト/レスポンス	セキュリティ/監査など

- トラブルシューティングで最もよく扱うログはPodログとイベントログ
- この2つのログで原因を絞り込めない場合、コンポーネントログを見る場合あり

Copyright 2020 FUJITSU LIMITED

コンポーネントログを調査するケース例

FUJITSU

ケース1: PodへPVがアタッチされない場合

- 前提条件: PVが正常に作成できていることは確認済
- この場合、ストレージ側のボリューム生成はOK、Pod内のコンテナへのマウント処理NGの可能性あり
- 原因特定のため、ノード側のシスログやマウント関連の設定だけでなく
  コンポーネント(kubelet)のログを確認しながら、問題個所を探る手番が必要

ケース2: 不審なユーザー操作やクラスタ動作を検知した場合

- 把握していないPod操作や予期していないPodのWorker移動などを発見
- その事象の全体像や原因を把握したい
  (各コンポーネント(kube-schedulerやkube-apiserverなど)の動作内容、大元のリクエストなど)
- 調査の一環で、一連の動作を抑えるために、各コンポーネントの関連ログ精査が必要な場合あり

Copyright 2020 FUJITSU LIMITED

アジェンダ

FUJITSU

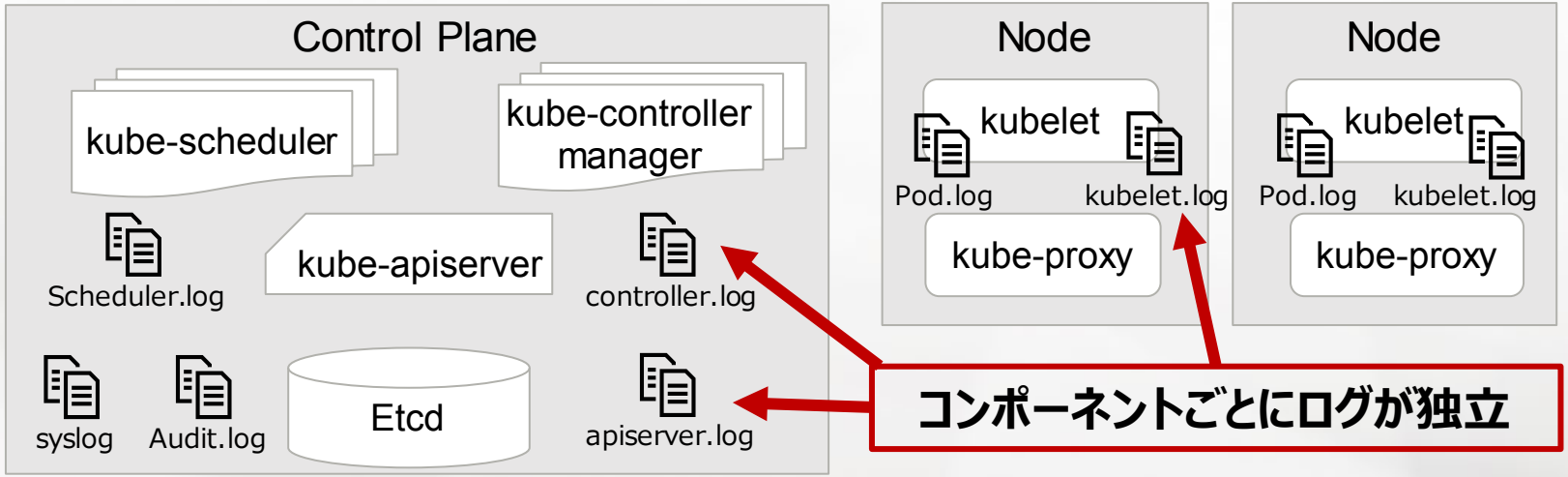
- ロギング機能の概要
- ロギング機能の課題
- 解決策
- K8sコミュニティでの開発状況
- 残課題/今後の予定

Copyright 2020 FUJITSU LIMITED

Kubernetesのロギング機能の課題

FUJITSU

- コンポーネント間を跨るログ調査が困難
  - K8sでは任意の操作(kubectl applyなど)の延長で、複数のコンポーネントが動作する
  - このため、クラスタ内部の一連の動作を追う場合は、コンポーネントを跨ったログ解析が必要
  - しかし、K8sのコンポーネントログは、**コンポーネントごとに独立して出力**される
  - このため、個々のコンポーネントのログから関連するログを抽出し、紐づける作業が必要

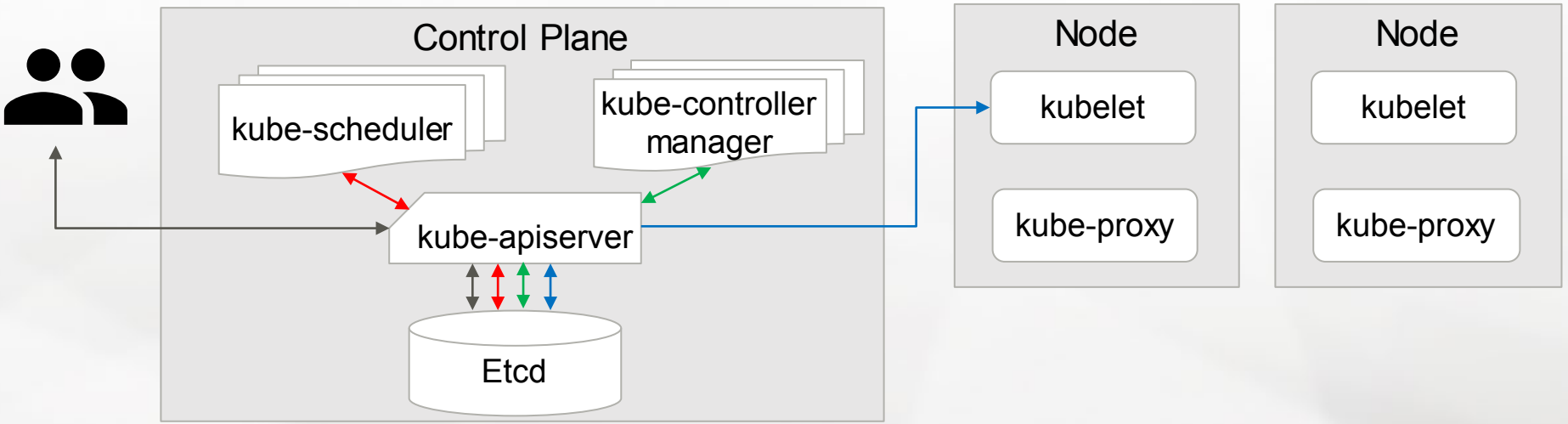


Copy right 2020 FUJITSU LIMITED

コンポーネントごとにログが独立している理由

FUJITSU

- K8sのアーキテクチャに起因
  - K8sではクラスタのDesiredな状態(Deployment数など)をEtcdに保存している
  - 各コンポーネントはapiserverと通信してEtcd内のDesiredな状態と現状を比較し、差分がある場合はDesiredな状態に保つように動作する
  - 各コンポーネントがDesiredに従い独立駆動する仕組みのため、ログも独立して出力される



Copy right 2020 FUJITSU LIMITED

コンポーネント間のログの紐づけ方法

FUJITSU

- 既存の手法
  - 各コンポーネントのログをリソース名でフィルタリング
    (例: Pod作成に関するログを紐づける場合は、Pod作成時の名前で各ログをクエリする)
  - 各コンポーネントのログをタイムスタンプで並べ、近い時間帯のログを精査して手作業で紐づける
- 既存手法の課題
  - リソース名のないログは抽出不可
  - 同じ時間帯に複数のAPIリクエストが投入された場合、仕分けに時間がかかる

Copyright 2020 FUJITSU LIMITED

アジェンダ

FUJITSU

- ロギング機能の概要
- ロギング機能の課題
- 解決策
- K8sコミュニティでの開発状況
- 残課題/今後の予定

Copyright 2020 FUJITSU LIMITED

解決策(Request ID)

FUJITSU

- 任意のユーザー(kubectl applyなど)の結果として各コンポーネントが出力するログに一意なID(以降Request IDと呼ぶ)を付与する
- Request IDのメリット
  - 一連のK8s内部の動作を追う場合に、各コンポーネントのログをRequest IDでフィルタリングするだけで関連するログを簡単に抽出可能
  - リソース名のないログも対応可能
  - タイムスタンプやログの内容精査により、関連するログを仕分ける作業も不要

Copyright 2020 FUJITSU LIMITED

ユースケース: トラブル原因特定短縮化

FUJITSU

- **PodへPVがアタッチされないケース**
  - 前提条件: PVが正常に作成できていることは確認済
  - この場合、ストレージ側のボリューム生成はOK、Pod内のコンテナへのマウント処理NGの可能性あり
  - 原因特定のため、ノード側のシスログやマウント関連の設定だけでなく
    コンポーネント(kubelet)のログを確認しながら、問題個所を探る手番が必要
- **Request IDを用いた場合**
  - K8s内のどの処理まで進んで、どの処理でエラーになったか即座に特定可能
  - そのため、実際に問題がある箇所をより高精度で絞り込むことができる
  - 結果として、調査範囲も狭まり、原因特定を容易化することができる

**エラー検知時に、元となったユーザー操作やどのような経緯を辿って、
そのエラーが出たかについて、関連するログを容易に抽出することができる**

Copyright 2020 FUJITSU LIMITED

ユースケース: ユーザー操作による影響範囲把握

FUJITSU

- **不審なユーザー操作やクラスタ動作を検知したケース**
  - 把握していないPod操作や予期していないPodのWorker移動などを発見
  - その事象の全体像や原因を把握したい
    (各コンポーネント(kube-schedulerやkube-apiserverなど)の動作ログ、大元のリクエストなど)
  - 調査の一環で、一連の動作を抑えるために、各コンポーネントの関連ログ精査が必要な場合あり
- **Request IDを用いた場合**
  - Request IDを用いると、1度のクエリ操作で関連するログを全て抽出可能
  - そこから何が起きているか、原因範囲絞り込みなど、事象把握に要する時間を短縮可能

任意のユーザー操作に対して、結果として出力されるログを簡単に抽出可能

Copyright 2020 FUJITSU LIMITED

実際のログ例(1/2)

- 例としてDeploymentの作成操作に関するログの抽出(一部を抜粋)
- ログの末尾に付与されている**reqIDの数値をキー**として、関連する処理を抽出可能

kube-apiserver.log

```
:90] verb="GET" URI="/apis/batch/v1/jobs?limit=500" latency=4.200904ms resp=200 UserAgent="kube-controller-manager/v1.19.0
n:serviceaccount:kube-system:cronjob-controller" srcIP="127.0.0.1:42298" reqID="000":
:90] verb="GET" URI="/apis/batch/v1beta1/cronjobs?limit=500" latency=2.232937ms resp=200 UserAgent="kube-controller-mana
ger/v1.19.0 (linux/amd64) kubernetes/ca7a906/system:serviceaccount:kube-system:cronjob-controller" srcIP="127.0.0.1:42298" reqID="000":
I0610 11:35:39.936636 23035 httplog.go:90] verb="GET" URI="/openapi/v2?timeout=32s" latency=719.637ms resp=304 UserAgent="kubectl/v1.18.2 (linux/amd64) kuberne
tes/52c56ce" srcIP="127.0.0.1:43628" reqID="111":
I0610 11:35:40.154162 23035 httplog.go:90] verb="GET" URI="/api/v1/namespaces/default/services/nginx" latency=2.00
9283ms resp=404 UserAgent="kubectl
I0610 11:35:40.187608 23035 httplog
/v1.19.0 (linux/amd64) kubernetes/ca7a
I0610 11:35:40.189286 23035 httplog
md64) kubernetes/52c56ce" srcIP="12
I0610 11:35:40.199000 23035 httplog
ments/nginx" latency=8.157038ms resp
I0610 11:35:40.199404 23035 httplog
ger/v1.19.0 (linux/amd64) kubernetes/ca
I0610 11:35:40.199604 23035 httplog
gent="kube-controller-manager/v1.19.0
```

Kube-controller-manager.log

```
I0610 11:35:31.415483 23452 request.go:571] Throttling request took 194.154901ms, request: GET:https://localhost:6443/apis/certificates.k8s.io/v1beta1?timeout=32s
I0 est took 243.777244ms, request: GET:https://localhost:6443/apis/batch/v1beta1?timeout=32s
I0 icas for ReplicaSet default/nginx-97499b967, need 3, creating 3 reqID="111"
I0 ference{Kind:"Deployment", Namespace:"default", Name:"nginx", UID:"76ef788a-b4c3-4cbe-9961-6c9f552d5
94c", APIVersion:"apps/v1", ResourceVersion:"717", FieldPath:""}): type: 'Normal' reason: 'ScalingReplicaSet' Scaled up replica set nginx-97499b967 to 3 reqID="111"
I0610 11:35:40.273453 23452 event.go:278] Event(v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"nginx-97499b967", UID:"cc728714-df1b-465e-aba0-
249f8aaf614a", APIVersion:"apps/v1", ResourceVersion:"718", FieldPath:""}): type: 'Normal' reason: 'SuccessfulCreate' Created pod: nginx-97499b967-6ztdx reqID="111"
I0610 11:35:40.273856
been modified; please ap
I0610 11:35:40.280390
9f8aaf614a", APIVersion
I0610 11:35:40.342805
9f8aaf614a", APIVersion
I0610 11:36:01.320381
```

kubelet.log

```
I0610 11:35:35.203862 23617 prober.go:126] Readiness probe for "kube-dns-c74654849-pvxm7_kube-system(1fd40a5a-934d-4c0d-bd41-593f29196c10):kubedns" succeeded
I0 factory.go:207] Factory "raw" can handle container "/system.slice/kubelet.service", but ignoring.
I0 kubelet.go:1837] SyncLoop (ADD, "api"): "nginx-97499b967-6ztdx_default(f7c403c0-5cca-4bf6-a789-3656738c1e96)"
I0 topology_manager.go:233] [topologymanager] Topology Admit Handler
I0 reflector.go:207] Starting reflector *v1.Secret (0s) from object-"default"/"default-token-gfz64" reqID="111"
I0610 11:35:40.345228 23617 reflector.go:243] Listing and watching *v1.Secret from object-"default"/"default-token-gfz64" reqID="111"
I0610 11:35:40.370523 23617 kubelet.go:1837] SyncLoop (ADD, "api"): "nginx-97499b967-xc5n8_default(0e00848d-902d-45b2-b5ab-1a4116b6bbd4)" reqID="111"
I0610 11:35:40.370595 23617 topology_manager.go:233] [topologymanager] Topology Admit Handler
I0610 11:35:40.370713 23617 kubelet_pods.go:1363] Generating status for "nginx-97499b967-6ztdx_default(f7c403c0-5cca-4bf6-a789-3656738c1e96)" reqID="111"
I0610 11:35:40.378012 23617 factory.go:210] Using factory "raw" for container "/kubepods.slice/kubepods-besteffort.slice/kubepods-besteffort-podf7c403c0_5cca_4bf6_a789
_3656738c1e96.slice"
I0610 11:35:40.379605 23617 manager.go:948] Added container: "/kubepods.slice/kubepods-besteffort.slice/kubepods-besteffort-podf7c403c0_5cca_4bf6_a789_3656738c1e9
6.slice" (aliases: [], namespace: "")
I0610 11:35:40.380129 23617 container.go:480] Start housekeeping for container "/kubepods.slice/kubepods-besteffort.slice/kubepods-besteffort-podf7c403c0_5cca_4bf6_a78
9_3656738c1e96.slice"
I0610 11:35:40.387846 23617 volume_manager.go:372] Waiting for volumes to attach and mount for pod "nginx-97499b967-6ztdx_default(f7c403c0-5cca-4bf6-a789-3656738c
1e96)" reqID="111"
I0610 11:35:40.395511 23617 eviction_manager.go:238] eviction manager: synchronize housekeeping
I0610 11:35:40.404504 23617 topology_manager.go:233] [topologymanager] Topology Admit Handler
```


Copy right 2020 FUJITSU LIMITED

実際のログ例(2/2)

- reqIDをキーにログを抽出して時系列順に整理することで、一連の処理の流れを把握可能
- エラーログからの原因調査の場合は、エラーログに付与されているreqIDをキーを用いる
- 実際のログの収集や解析はElasticsearchやKibanaなど他OSSの使用を想定

```
I0610 11:35:39.936636   23035 httplog.go:90] verb="GET" URI="/openapi/v2?timeout=32s" latency=719.637ms resp=304 UserAgent="kubectl/v1.18.2 (linux/amd64) kubernetes/52c56ce" srcIP=
“127.0.0.1:43628" reqID="111":
I0610 11:35:40.154162   23035 httplog.go:90] verb="GET" URI="/api/v1/namespaces/default/services/nginx" latency=2.009283ms resp=404 UserAgent="kubectl/v1.18.2 (linux/amd64) Kubernete
s/52c56ce" srcIP="127.0.0.1:43628" reqID="111":
I0610 11:35:40.189286   23035 httplog.go:90] verb="POST" URI="/api/v1/namespaces/default/services" latency=33.947136ms resp=201 UserAgent="kubectl/v1.18.2 (linux/amd64) kubernetes/5
2c56ce" srcIP="127.0.0.1:43628" reqID="111":
I0610 11:35:40.199000   23035 httplog.go:90] verb="GET" URI="/apis/apps/v1/namespaces/default/deployments/nginx" latency=8.157038ms resp=404 UserAgent="kubectl/v1.18.2 (linux/amd64
) kubernetes/52c56ce" srcIP="127.0.0.1:43628" reqID="111":
I0610 11:35:40.244334   23452 replica_set.go:562] Too few replicas for ReplicaSet default/nginx-97499b967, need 3, creating 3 reqID="111"
I0610 11:35:40.245614   23452 event.go:278] Event(v1.ObjectReference{Kind:"Deployment", Namespace:"default", Name:"nginx", UID:"76ef788a-b4c3-4cbe-9961-6c9f552d594c", APIVersion:"a
pps/v1", ResourceVersion:"717", FieldPath:""}): type: 'Normal' reason: 'ScalingReplicaSet' Scaled up replica set nginx-97499b967 to 3 reqID="111"
I0610 11:35:40.273453   23452 event.go:278] Event(v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"nginx-97499b967", UID:"cc728714-df1b-465e-aba0-249f8aaf614a", API
Version:“apps/v1", ResourceVersion:"718", FieldPath:""}): type: 'Normal' reason: 'SuccessfulCreate' Created pod: nginx-97499b967-6ztdx reqID="111"
I0610 11:35:40.273856   23452 deployment_controller.go:485] Error syncing deployment default/nginx: Operation cannot be fulfilled on deployments.apps “nginx”: the object has been modified; p
lease apply your changes to the latest version and try again reqID="111"
I0610 11:35:40.280390   23452 event.go:278] Event(v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"nginx-97499b967", UID:"cc728714-df1b-465e-aba0-249f8aaf614a", API
Version:"apps/v1", ResourceVersion:"718", FieldPath:""}): type: 'Normal' reason: 'SuccessfulCreate' Created pod: nginx-97499b967-xc5n8 reqID="111"
I0610 11:35:40.342805   23452 event.go:278] Event(v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"nginx-97499b967", UID:"cc728714-df1b-465e-aba0-249f8aaf614a", API
Version:"apps/v1", ResourceVersion:"718", FieldPath:""}): type: 'Normal' reason: 'SuccessfulCreate' Created pod: nginx-97499b967-sgfg2 reqID="111"
I0610 11:35:40.345157   23617 reflector.go:207] Starting reflector *v1.Secret (0s) from object-"default"/"default-token-gfz64“ reqID="111"
I0610 11:35:40.345228   23617 reflector.go:243] Listing and watching *v1.Secret from object-"default"/"default-token-gfz64“ reqID="111"
I0610 11:35:40.370523   23617 kubelet.go:1837] SyncLoop (ADD, "api"): "nginx-97499b967-xc5n8_default(0e00848d-902d-45b2-b5ab-1a4116b6bbd4)“ reqID="111"
I0610 11:35:40.370595   23617 topology_manager.go:233] [topologymanager] Topology Admit Handler
I0610 11:35:40.370713   23617 kubelet_pods.go:1363] Generating status for "nginx-97499b967-6ztdx_default(f7c403c0-5cca-4bf6-a789-3656738c1e96)“ reqID="111"
I0610 11:35:40.387846   23617 volume_manager.go:372] Waiting for volumes to attach and mount for pod "nginx-97499b967-6ztdx_default(f7c403c0-5cca-4bf6-a789-3656738c1e96)“ reqID="111"
```


Copy right 2020 FUJITSU LIMITED

アジェンダ

- ロギング機能の概要
- ロギング機能の課題
- 解決策
- K8sコミュニティでの開発状況
- 残課題/今後の予定

Copyright 2020 FUJITSU LIMITED

開発の現状

FUJITSU

- **コミュニティ状況**
  - 提案中の機能(https://github.com/kubernetes/enhancements/pull/1348)
  - 現在はProvisionalステータス(機能の設計段階)
- **提案中の主要機能**
  - Request IDフラグ: ログへのRequest ID付与をON/OFFするフラグ
  - Request ID付与対象: 主要なユーザー操作(kubectl apply/delete/createなど)
- **開発予定にない機能**
  - 使用頻度の低いユーザー操作に対するRequest ID付与
    (操作ごとに個別の修正が必要のため全操作の対応は時間/工数大。優先度を決めて対象を選定)
  - ログ収集/分析/可視化: ログの収集&Request IDでクエリ&可視化など
    (これらは既存OSS(Elasticsearch, Kibana, Fluentdなど)を利用)

Copyright 2020 FUJITSU LIMITED

Request IDフラグ

FUJITSU

- **Request ID付与の有効/無効を制御する機能**
  - 既存のログフォーマットを変えたくないユーザー向け
  - デフォルトはRequest ID付与は無効にする予定

- **有効無効の制御は以下の2つの方式を検討中(コミュニティで議論中)**
  - K8sの既存ログレベル機能(log verbosity)を活用 (-vの値が4以上で有効にする)
    メリット: ログ関係のフラグ管理の複雑化を避ける
    デメリット: Request IDを有効化と同時に出力されるログ情報も増える
  - Request ID専用の新規フラグを実装
- **どちらの方式も各コンポーネント起動時のlog初期化時に設定**
- **したがって、設定変更する場合はコンポーネントの再起動が必要**

Copyright 2020 FUJITSU LIMITED

設計上の課題1

FUJITSU

対象操作やリソースの選定

- Request IDを付与(ソース上で実装)する場合、ソースコード上の既存のログ出力(klogメソッド呼び出し)箇所を1つずつ書き換える
- K8sのログ出力箇所は1万以上あるため、Request IDを付与する対象を優先度に基づいて限定する
- 優先度は使用頻度の高いユーザー操作

優先度	操作内容	コマンド例
1	ユーザー利用が多いPod操作	kubectl create/delete/apply + Pod/Deployment
2	ユーザ資産を保存するPV / PVC操作	kubectl create/delete/apply + PV/PVC
3	オーケストレーション系で比較的トラブルの多いネットワーク操作	kubectl create/delete/apply + SVC

利用者の操作優先度を検討し、段階的に対象範囲を拡大する

Copyright 2020 FUJITSU LIMITED

設計上の課題2

FUJITSU

対象環境の選定

- ユーザー操作(Pod生成など)により呼び出されるklogは環境によって差分が出る
  例: コンテナランタイム(docker/cri-o)、ボリュームタイプ(hostpath/emptyDir/etc)
- Request IDを付与するklogは全ての構成パターンを対象にすると膨大な作業となる懸念あり
- 現状は代表的な以下の構成で呼び出されるklogを対象とする予定

構成要素	種別(左から優先度高)
コンテナランタイム	cri-o, docker
ボリュームタイプ	empydir, configMap, local, nfs, cinder, cephfs, iscsi
ネットワークプラグイン	multus, flannel

※2020/6/13時点の予定。コミュニティ状況により変更の可能性あり

Copyright 2020 FUJITSU LIMITED

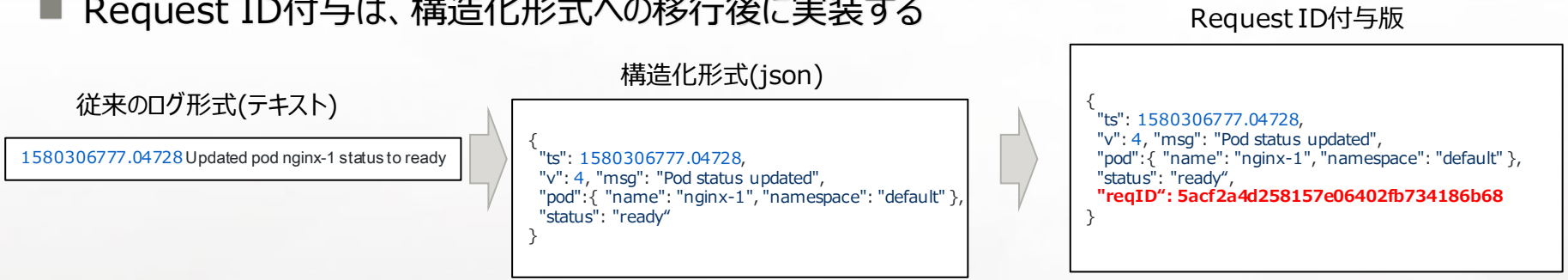
前提機能1

FUJITSU

Request IDは2つの前提機能が存在

Structured Logging(Googleが提案)(https://github.com/kubernetes/enhancements/pull/1367)

- K8sのコンポーネントログを従来のテキスト形式から構造化(json)形式に移行
- Request ID付与は、構造化形式への移行後に実装する



- ログがjson形式になることで、ログの管理やクエリの容易性が大幅に向上
- K8s 1.19で主要なログ呼出しが構造化形式に変更予定

Copyright 2020 FUJITSU LIMITED

補足(個々のklog呼び出しに手を加える理由)

FUJITSU

大元のklog定義の変更やWrapperの使用で対処しない理由

- **前提条件**
  - 前述の前提機能により、klogは従来のテキスト形式から構造化形式に移行
  - Request IDは構造化形式に準拠(Key/Value形式使用)
- **構造化への移行課題**
  - ソースレベルでは、個々のklog呼出しは、以下の例のようにテキストをKey/Valueに分解必要

```
従来(テキスト)          : klog.Infof("Pod name %s is ready", pod)
構造化版(Key/Value)     : klog.InfoS("Pod status updated", "pod", "pod", "status", "ready")
```

  - Wrapperなどで対処する場合、個々のテキストを適切なKeyとValueに自動選別するロジックが必要
  - しかし、K8s内の個々のklogはメッセージのフォーマットが統一されておらず、自動選別は困難
- **現状の対応**
  - 構造化への移行は当面は優先度を決めて、手作業での置換が計画されている
  - したがって、Request-IDも手作業で個々のklogを構造化形式に移行する作業が前提として必要

Copyright 2020 FUJITSU LIMITED

前提機能2

FUJITSU

Distributed Tracing(Googleが提案)(https://github.com/kubernetes/enhancements/pull/650)

- **概要**
  - ユーザー操作(kubectl)のトレースを実現する(--traceオプションを追加)
  - kubectlコマンドの延長で動作する各コンポーネントの処理を、関数レベルで処理時間を計測
- **Request IDの前提となる部分**
  - トレース情報はコンポーネントを跨り伝播する。この伝播機能をRequest IDに活用
  - 具体的にはTracing機能では、k8sのオブジェクトのAnnotationにトレース情報を格納する
  - 各コンポーネントの関数はオブジェクトを参照することで、情報の伝播が可能となる
- **開発状況**
  - コミュニティで設計の承認待ち。
  - 当初はk8s 1.19で実装される予定だったが、設計の承認が長引き1.21以降になる見込み

Copyright 2020 FUJITSU LIMITED

Tracing機能の遅延理由とRequest IDへの影響

FUJITSU

遅延理由

K8sのような独立駆動型アーキテクチャにおいて、コンポーネントを跨ったトレース方式の最適解がまだ確定していない(アイデアはあるが正式合意には至っていない)

- 例えば、RPCプロトコルに従うHTTP関連の伝播(kube-client～kube-apiserver～Etcd)のみであればヘッダに情報を付与する形でトレースは容易に実現可能。
- しかし、大半のコンポーネントは独立駆動型のため、上記の方法は使用不可
- オブジェクトのAnnotationを使用するアイデアはまだコミュニティで正式に承認が取れていないため、k8s　1.20ではHTTP周りの伝播のみを実装予定
- したがって、コンポーネント全体の伝播は1.21以降となる見込み

Request IDへの影響

- Request IDはTracing機能の伝播方式を利用する場合、1.21以降に実装となる
- これは大きなブロッカーとなるため、現在、Tracing機能に頼らない伝播方式を検討中

Copyright 2020 FUJITSU LIMITED

アジェンダ

- ロギング機能の概要
- ロギング機能の課題
- 解決策
- K8sコミュニティでの開発状況
- 残課題/今後の予定

Copyright 2020 FUJITSU LIMITED

残課題/ロードマップ

FUJITSU

残課題

- 設計の合意(Request ID有効/無効の制御方式、コンポーネント間の伝播方式)

ロードマップ

	位置づけ	リリース予定時期	Request IDを付与する 対象オペレーションとリソース
α版	基本機能の実装	K8s 1.21 (2021/2月)	- kubectl create/apply/delete + Pod/Deployment - kubectl drain node
β版	対応操作の拡充	K8s 1.22 (2021/5月)	- kubectl create/apply/delete + DaemonSet/PV/PVC/SVC - kubectl scale/rollout
GA版	障害対応、改善要望などの反映	K8s 1.23 (2021/8月)	上記に加えコミュニティ上のフィードバックを反映

※2020/6/13時点の予定。コミュニティ状況により変更の可能性あり

Copyright 2020 FUJITSU LIMITED

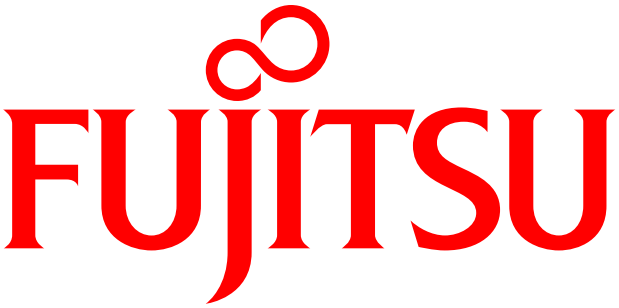
最後に

FUJITSU

- **運用者への期待**
  - トラブルシュート時のログで困ったことがある方はフィードバックお待ちしております。
  - 機能改善の要望は大歓迎です。
    (例: kubectlの別のサブコマンドやリソースも対象にしてほしいなど)
  - 様々な運用者のニーズを把握することで、より価値のある機能にすることが目的です。
- **開発者への期待**
  - Request IDの開発に興味を持たれた方は連絡お願いします。
  - 共同での開発(コンポーネントを跨る伝播方式に対するアイデアなど)は歓迎します。
  - K8sの日本での開発を盛り上げていきましょう。
- **連絡の際は、下記のいずれかにお願いします（日本語/英語可）。**

Github(Request IDのKEP): https://github.com/kubernetes/enhancements/pull/1348
K8sのslack ID: @hase1128
長谷のEmail: hase.jin@fujitsu.com

Copyright 2020 FUJITSU LIMITED



shaping tomorrow with you