



# OpenShift ハンズオン

## アプリケーションの運用

Kento Shirakawa  
Consultant  
Red Hat K.K.

# 最初に必要な情報

- ▶ 情報共有 (etherpad)
  - ・ トレーニング中に利用する情報の共有や QA に利用
- ▶ 本日のトレーニングの構成
  - ・ 説明と実習の組み合わせ。
  - ・ 質問がありましたら、適宜お声がけください。

# 講師 自己紹介

名前:白川 賢人

仕事:

2019.6～ Red Hat にてコンテナ・DevOps 関連の支援

2011.4～ Sler にてアプリケーションサーバの開発や  
CI/CD 導入支援、Kubernetes 基盤導入支援



# 本日のハンズオンのゴール

- ▶ アプリケーションをスケールアウト・インできる。
- ▶ OpenShift の Observability 機能 ( Monitoring, Logging ) を使い、アプリケーションの稼働状況を確認できる。

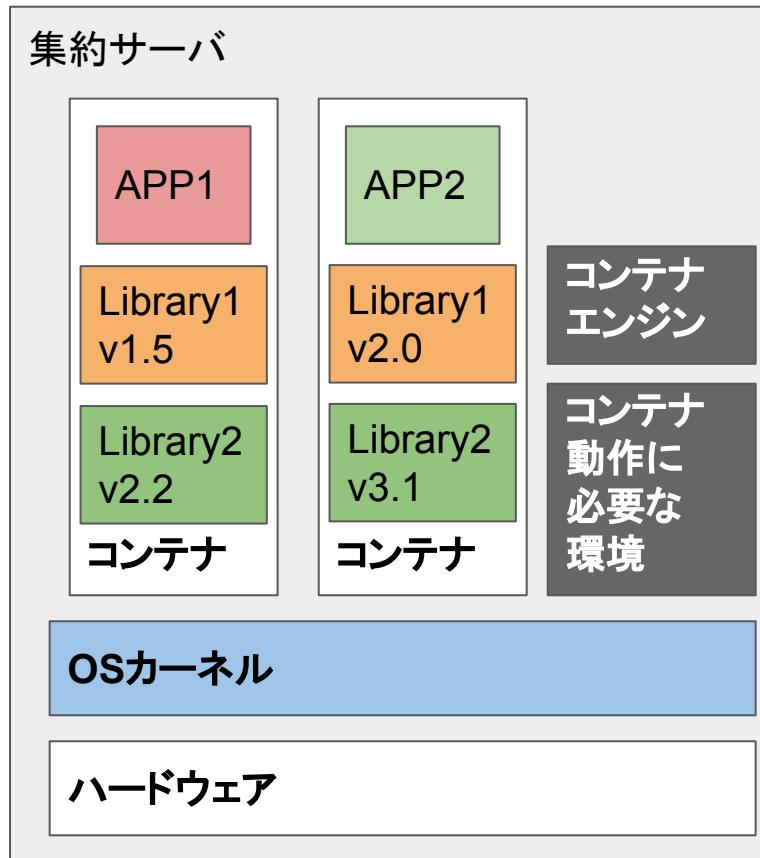
# アジェンダ

- ▶ 前回ハンズオンの振り返り
- ▶ アプリケーションのスケーリング
- ▶ Observability (Monitoring / Logging)

---

# 前回ハンズオンの振り返り

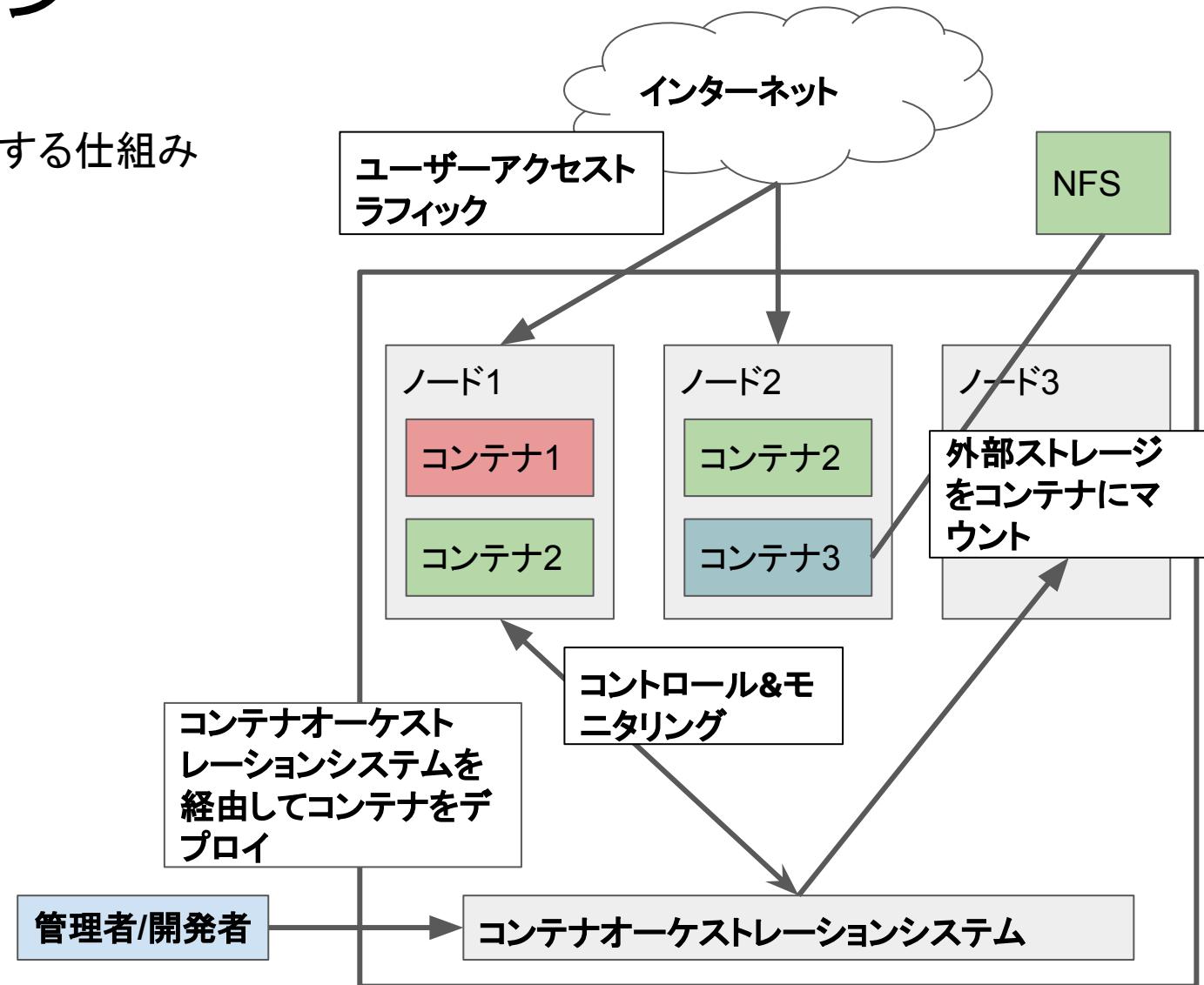
# コンテナの特徴



- アプリケーションのカプセル化
  - アプリケーション+必要なもの(カーネル以外)をカプセル化
    - カプセル化してパッケージングしたものをコンテナイメージ(または単にイメージ)と呼ぶ
- 再現性と可搬性
  - 環境非依存なパッケージングと、パッケージからの再現性
    - (OSカーネルに互換性があるなら)開発環境で正常に動作するものは運用環境でも正常に動作する
  - 配布と展開にかかる手間の軽減と高速化
- 軽い&速い
  - メインプロセスに必要なリソースのみ
  - 起動やスケールアウトも高速
- 実行方法の統一
  - アプリケーションごとの実行方法の差がない
  - デプロイ作業が単純化

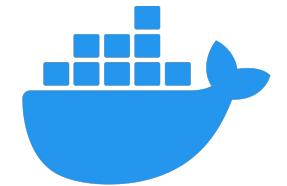
# コンテナオーケストレーション

- コンテナ群を適切に連携・隔離して実行・運用する仕組み
  - コンテナを動かす準備・設定
  - コンテナを起動し設定情報を伝達
  - コンテナにストレージを対応づけ
  - パスワード・秘密情報を付与
  - コンテナ間の連携対応づけ
  - ネットワーク設定
- コンテナの稼働制御
  - 稼働ノード(サーバ)にコンテナを展開
  - 可用性の維持
  - 負荷に合わせたスケールアウト
  - リソースの制限
  - クラスタの隔離
  - トラフィックの分散...etc

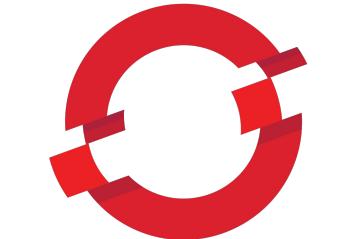


# コンテナオーケストレーション製品

- **Kubernetes (クバネティス/クバネテス/クーベネティス、K8s)**
  - Google 社内で使用されていたコンテナオーケストレーション「Borg」がベース
    - Google 社内で運用されていたコンテナ管理のノウハウが反映されている
  - 現在は CNCF(Cloud Native Computing Foundation) にて開発されている
  - さまざまなクラウドサービスで Kubernetes 基盤がサポートされている
  - 現在のコンテナオーケストレーションにおけるデファクトスタンダード
- **Docker Swarm(Swarmモード)**
  - Docker 社が開発したコンテナオーケストレーション
  - Kubernetes と比べると機能はシンプル
- **OpenShift**
  - Red Hat が開発している Kubernetes 拡張。Red Hatによるサポート
  - コンテナを複数ホストで稼働連携させるコンテナオーケストレーション機能に加え、監視、アプリケーションのライフサイクル、ビルド/デプロイ自動化、コンテナイメージレジストリ等を加えたコンテナ統合プラットフォーム



docker



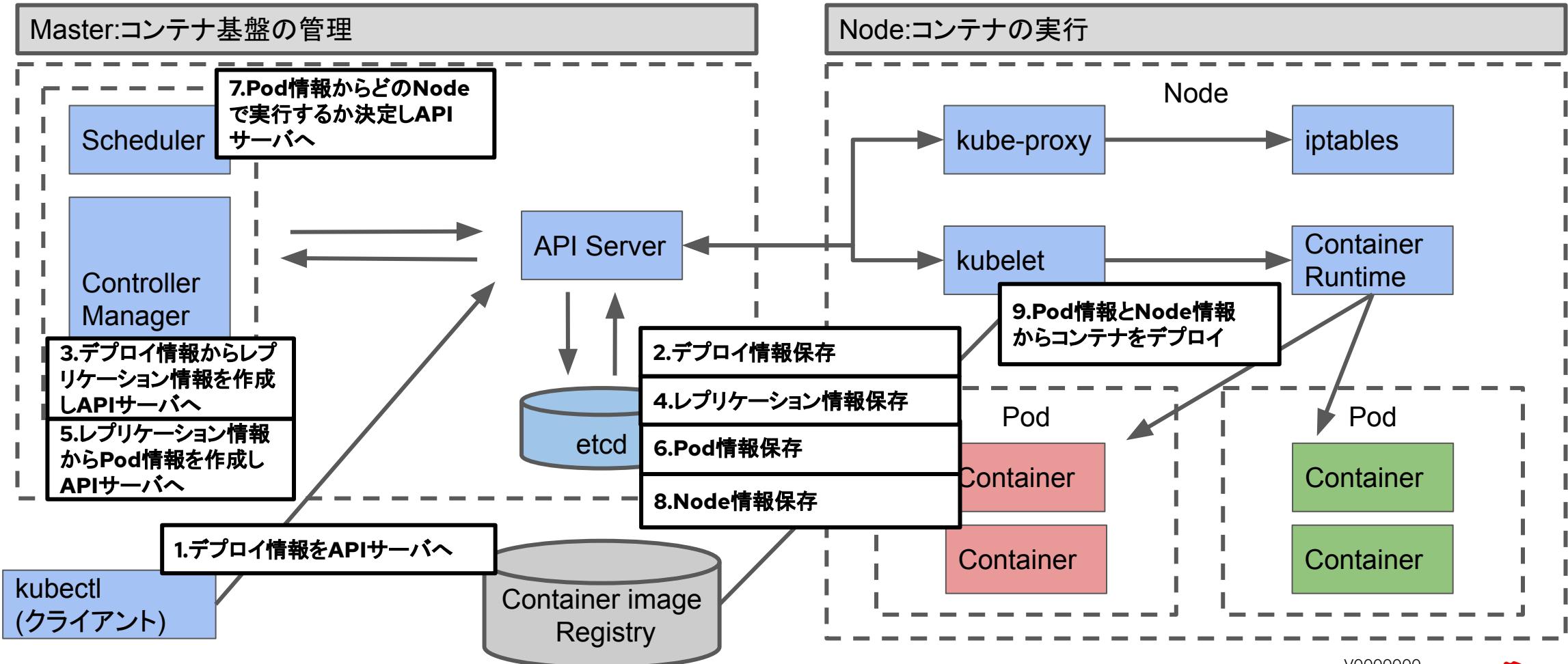
OPENSHIFT

V0000000



# Reconciliation Loop モデル

例: クラスタにコンテナをデプロイ



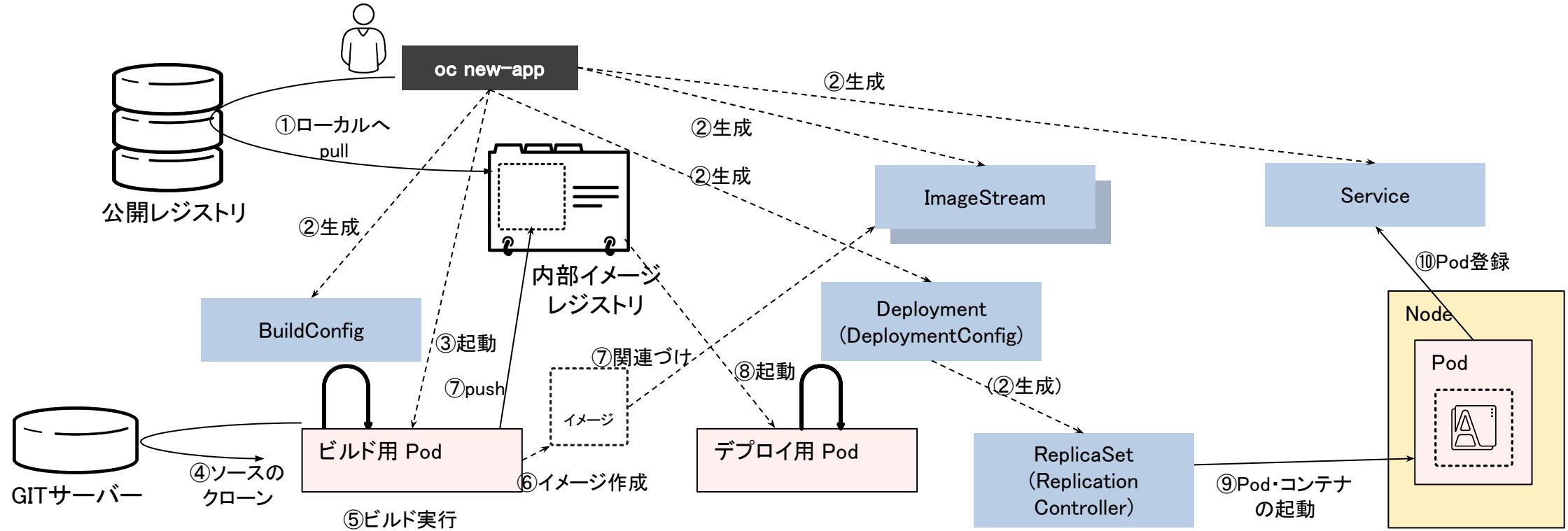
# Kubernetes の Reconciliation Loop モデル

- Kubernetes の継続的な制御ループモデル。
  - Control Loop とも呼ぶ。
  - Kubernetes 内部プロセス が API Server を通して取得・差分照合・実行を繰り返す



# application の作成時の動作

- ベースイメージの確保(図中①)
  - BuildConfig・Deployment (DeploymentConfig)・ImageStream・Service の生成(図中②)
  - ビルド実行によりコンテナイメージを作成し、イメージレジストリにpush、イメージストリームに関連づけ(図中③④⑤⑥⑦)
  - イメージストリームの変更(新規登録)を検知しデプロイ実行(図中⑧⑨)
  - デプロイされたPodはServiceに登録される(図中⑩)
- applicationとして、コンテナ上で稼働する機能のサービス実行が可能となる



# 前回の内容

- ▶ コンテナイメージの管理
- ▶ OpenShift クラスタ外からの Pod への通信
- ▶ デプロイ戦略
- ▶ コンテナのヘルスチェック
- ▶ Pod のコンピューティングリソース割当・上限
- ▶ アプリケーションの設定情報の管理
- ▶ 永続ボリュームの利用
- ▶ Template / Operator / Helm を用いたアプリケーションのデプロイ
- ▶ ServiceMesh

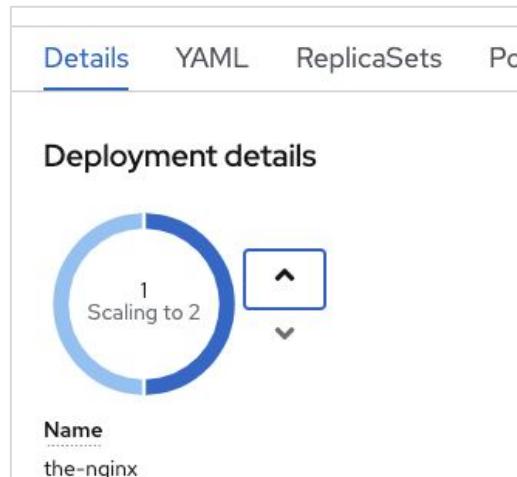
---

# アプリケーションのスケーリング

# Pod スケールアウト(手動)

Deployment (or DeploymentConfig) に指定したレプリカ数を変更することで、Podをスケールアウトさせる。

- マニフェストファイルを編集して `oc apply -f`
- `oc scale --replicas=レプリカ数 deployment/リソース名`  
→ マニフェストファイルに書かれたレプリカ数と乖離するため、GitOps的な運用を目指す場合は避ける。
- OpenShift Web コンソールにてレプリカ数を変更する。  
→ `oc scale`コマンドと同様



# Pod オートスケール

- Pod の負荷を計測し、規定の閾値を超えた場合にスケールアウト(イン)するオートスケールを設けることができる。
  - オートスケールは Horizontal Pod Autoscaler(HPA) による水平方向のスケーリングを提供。
  - 標準でオートスケールの基準とできる負荷は CPU・メモリの使用率のみ。
- Pod のリソース割当要求(request)を基準とした割合で閾値を設定する。
  - 直近1分間の使用状況が閾値を基準として、+10% を超えるとスケールアウト、-10% を下回るとスケールインされる。

# Pod のコンピューティングリソース割当・上限

- Pod が必要とする CPU・メモリのリソース要求を指定できる。
  - 割当要求(request)と上限(limit)として指定。
    - Pod または Pod template(DeploymentConfig 等の中) の定義内で記述する。

影響範囲	制限対象	設定	機能・動作
Pod	CPU Memory	request	Podに対してノードに確保されるCPU/Memory
		limit	Podが使用可能なCPU/Memoryの上限値
Container	CPU Memory	request	Containerに対してノードに確保されるCPU/Memory
		limit	Containerが使用可能なCPU/Memoryの上限値

- CPU の指定単位: コアまたはミリコア(m)
- メモリの指定単位: バイト(例: MB・MiB)
- メモリのlimitを越えた場合、OOM Killerによって停止される。
- Scheduler による Pod の配置時には、request で指定したリソースが使用可能なノードが選択される。

# OpenShift 実習 1

- Web Consoleへのアクセス確認
  - Web ブラウザ(Firefox・Chrome・等)を起動し、Web コンソール URL にアクセスし、割り当てられた開発者向けユーザ名 user# でログインする。
    - username: user#
    - password: openshift
  - ログインし、Developer Perspective の画面が開くことを確認する。



アカウントにログイン

ユーザー名 \*

パスワード \*

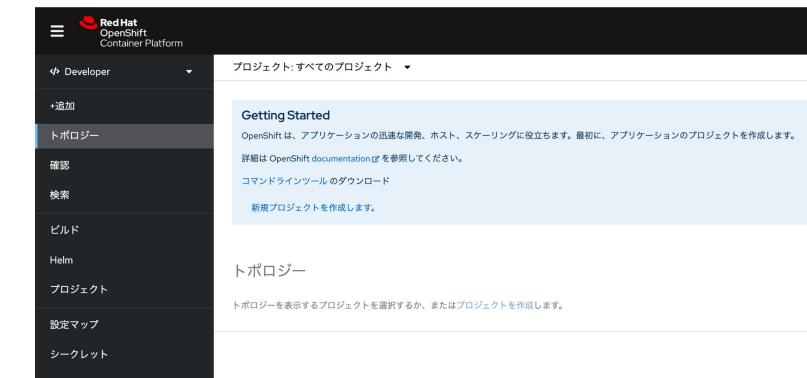
ログイン

## Developer パースペクティブへようこそ!

OpenShift 4.x の Developer パースペクティブの主な領域からスタートします。これは、ワークフローを実行し、生産性を高めるのに役立ちます。

ツアーをスキップ

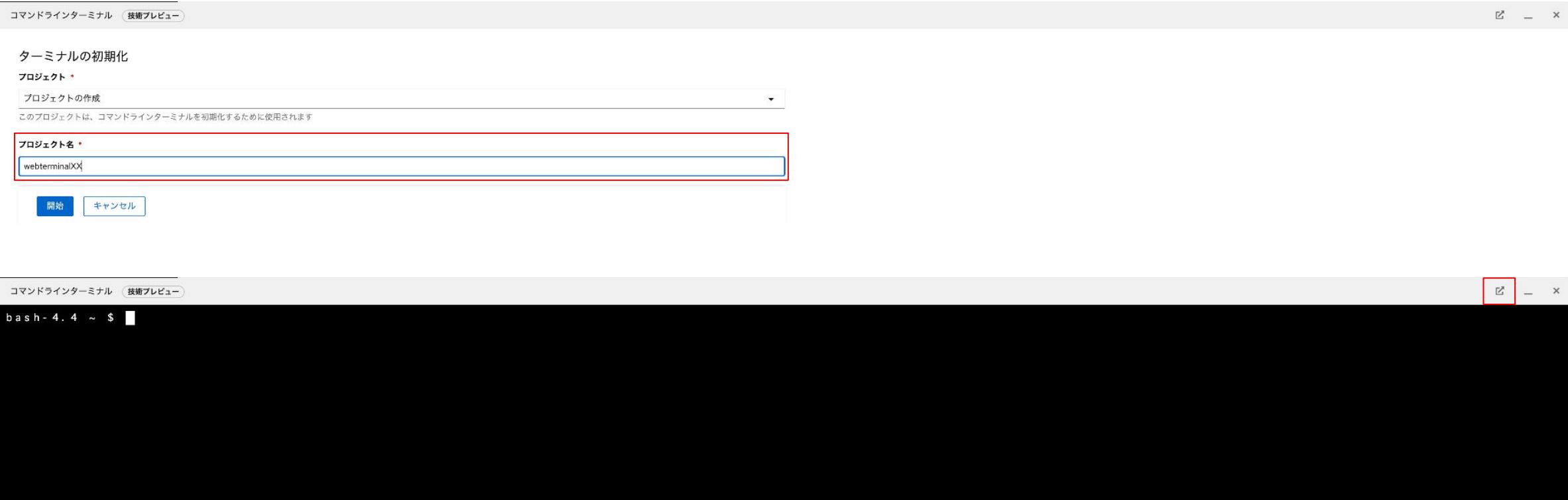
スタート



# OpenShift 実習 1

- Web Terminal へのアクセス確認

- 画面右上の  をクリックし Web Terminal を起動する
- プロジェクト名に webterminal#(開発者向け割り当てユーザ名の番号部分)を入力し開始をクリック
- 起動後右上の  から新規タブで Terminal を開くことができる



# OpenShift 実習 - 実習1

flaskアプリをデプロイする。

1. ソースコード、マニフェストファイルを取得する

```
git clone https://github.com/k-srkw/flask-example-app.git
cd flask-example-app/
```

2. プロジェクトを作る。

```
oc new-project flask-example-app-<開発者向け使用ユーザ番号>
→ user10 ならば flask-example-app-10
```

3. デプロイする。

```
oc apply -f openshift/flaskr.app.yaml
```

4. 公開する。

```
oc apply -f openshift/flaskr.route.yaml
```

5. アプリを開く。

```
oc get route flaskr
→ HOST/PORTのURLをブラウザで開く。
```

# OpenShift 実習1

HPAの挙動を確認する。

1. deployment/flaskr に対するHPAを設定する。

```
oc apply -f openshift/flaskr.hpa.yaml  
→ cpu 50ミリコアの50%を基準にスケールアウト・イン
```

2. 負荷をかけてみる。

- a. flaskrのPodに入る。

```
oc rsh deploy/flaskr またはWebコンソールからTerminal開く。
```

- b. abコマンドで負荷をかける。

```
ab -n 20000 -c 10 http://flaskr:8080/
```

3. HPAの状態を確認する(HPAが反応するまで数十秒かかります)。

```
oc get hpa またはWebコンソールからWorkloads > HorizontalPodAutoscalersを開く。
```

4. abコマンド終了後の挙動を確認する(負荷が止んでから5分程度かかります)。

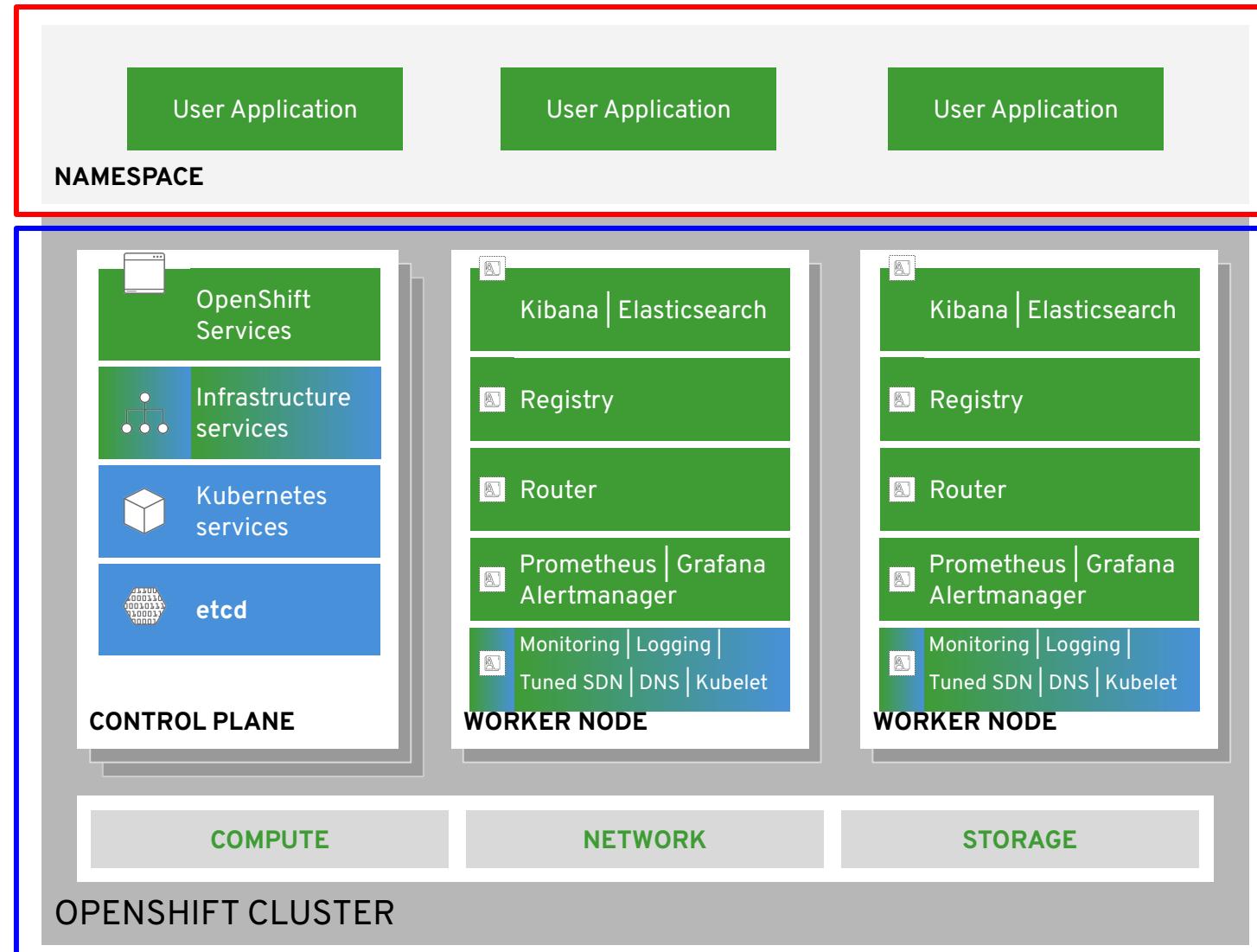
---

# Observability

# クラスタ管理とユーザアプリケーション管理

CONFIDENTIAL designator

監視・ロギングと一言で表すも、そのスコープは様々である。本資料では、大きくクラスタ管理に関する監視・ロギング(青枠)と、OpenShiftを利用するユーザのアプリケーションの監視・ロギング(赤枠)を分けて考えることができる。それぞれの利用者の観点で、どんな手法を用いることができるか注意してみよう。



V0000000



# モニタリングとはなにか

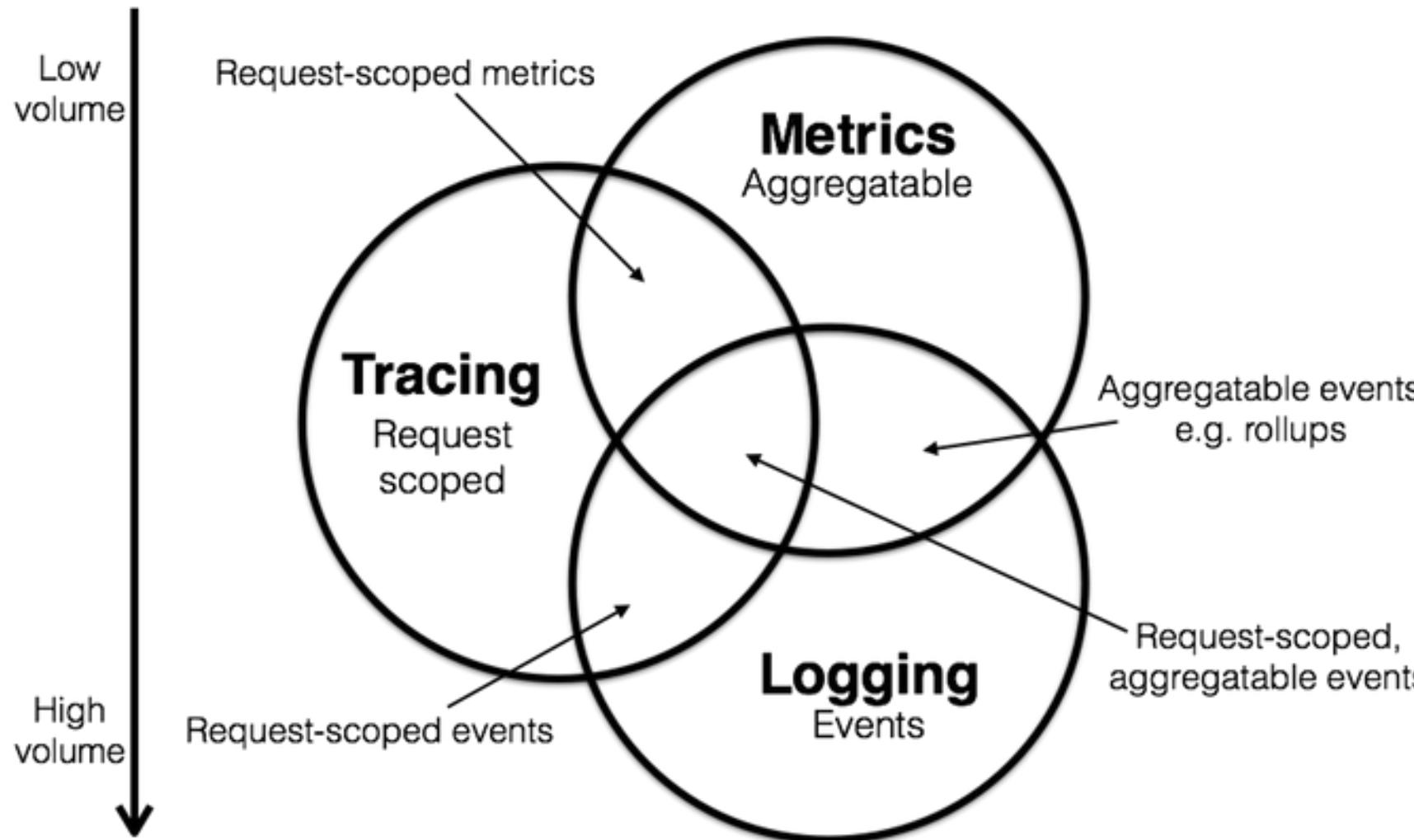
CONFIDENTIAL designator

本番環境でシステム(アプリケーション、ツール、データベース、ネットワークなど)の問題の発見と解決すること。以下のような項目がある。

- ▶ アラート
  - ・ システムが期待する状態でなくなったときの、運用者への通知。
- ▶ デバッグ情報の提供
  - ・ アラートを受け取った人が、問題の原因を特定するために必要な情報の提供。
- ▶ トレンド調査
  - ・ 通常、アラートやデバッグは分単位や時間単位で対応が必要であるが、一方で長期的な変化を捉えることが大切。
  - ・ 設計判断やキャパシティプランニングに活用
- ▶ 部品提供
  - ・ モニタリングで収集した情報の活用。
  - ・ オートスケーリング、AI Ops

# モニタリングの種類 (Observabilityの3本柱)

CONFIDENTIAL designator



# Metrics(メトリクス)

CONFIDENTIAL designator

メトリクスは特定の期間のデータの集計を表した測定値のこと。集計方法に、例えば average(平均)、total(合計)、minimum(最小)、maximum(最大)、sum-of-squares(平方和)などがある。

具体的なメトリクスに、CPU/メモリーの使用率や、レスポンスタイムなどがある。  
数値データのため閾値を設けることで、アラート発報が簡単にできる。

集計されたデータであるため、データ容量は少なく扱いが簡単な面もあるが、数値情報のためそのデータの背景や原因をもちあわせない。

# Logging(ロギング)

CONFIDENTIAL designator

ログはシステムが生成するいつどのように動いたか記録したテキスト行のこと。システムのトラブルシューティングに非常に役に立つ。というのもメトリクスと異なり、「異常な状態」となったときの原因となる行動の記録を持っている可能性があるため。

あらゆるシステムやミドルウェアがログを出力することが可能で、すべての動作をログに出力することは難しく、ログレベルでの出力を調整することや、保存したログから必要な情報をすばやく検索できることが重要。

また、クラウド・コンテナの環境においては、複数のプロセスがログを出力するため、アプリケーションのロール別に集約するなどの工夫が必要。

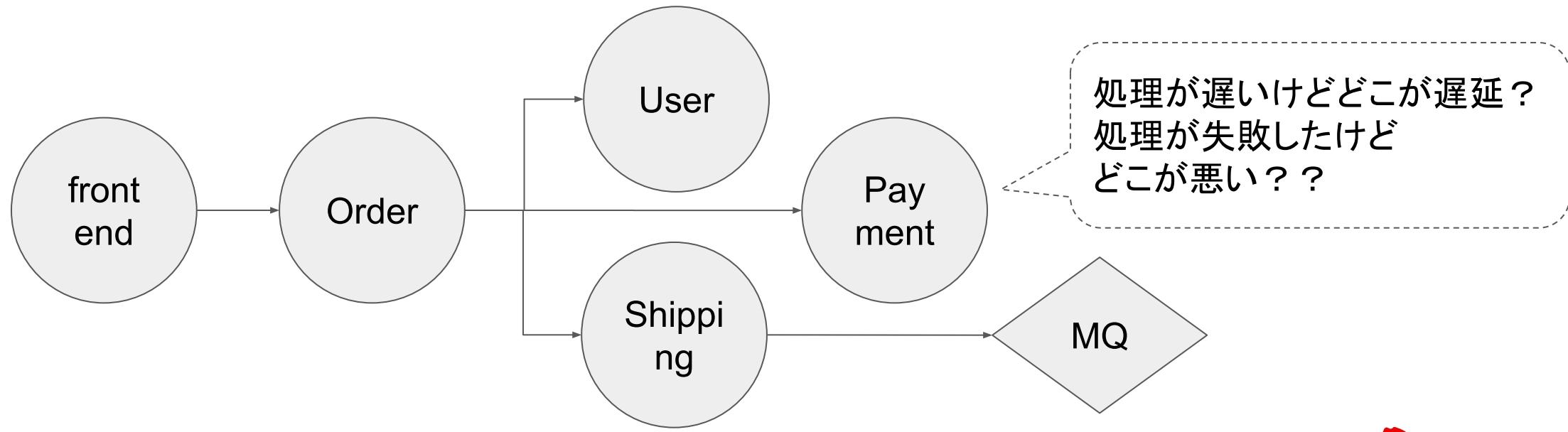
V0000000



# Tracing(トレーシング)

CONFIDENTIAL designator

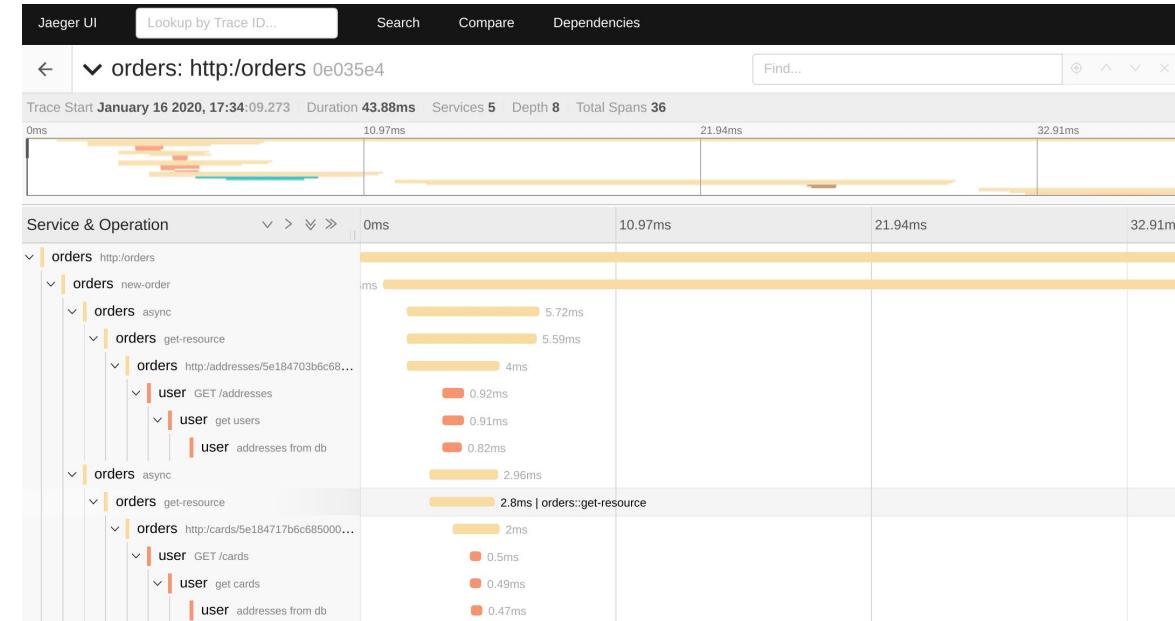
マイクロサービスのように、アプリケーションが複数のサービスで構成され構成が複雑になると、1つのリクエスト(例えばREST API)が内部的に複数のサービスを跨いで処理され。そのため、リクエスト全体の流れを分析することが難しくなり、トレーサビリティの低下を招くことになる。



# Tracing(トレーシング)

CONFIDENTIAL designator

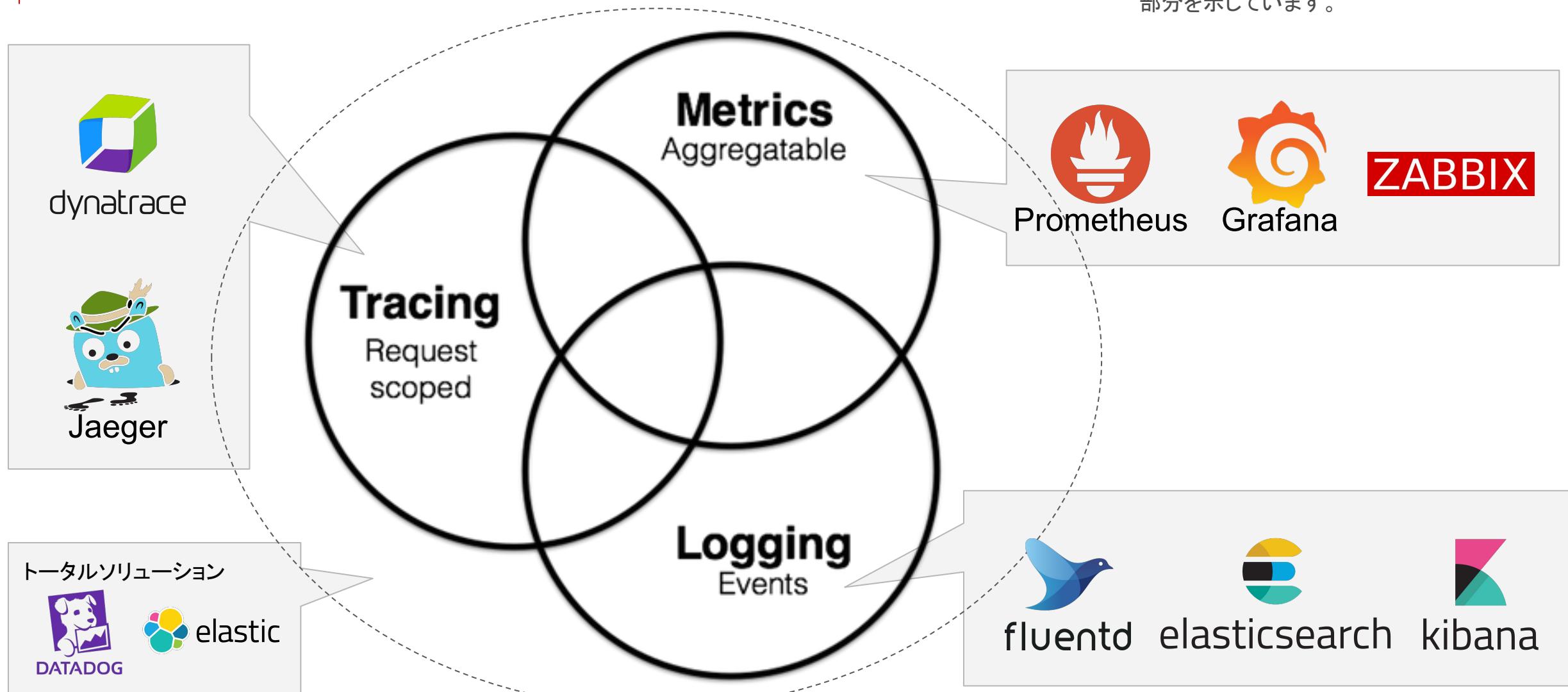
- ▶ 分散トレーシングではサービス間のリクエストの詳細な状態を記録する
  - ・ リクエストパスの追跡
  - ・ サービス間(ホップ)のレイテンション
  - ・ サービス間のエラー状況
- ▶ 複雑なマイクロサービス間の関係をビューとして確認することができる
  - ・ トラブルシューティングが容易
    - ・ どこのサービス間でエラーが起きているか、遅延が起きているか
- ▶ アプリケーションのパフォーマンスのボトルネック発見
  - ・ SQLのスロークエリなど



Metrics	<ol style="list-style-type: none"> <li>1. OpenShift Cluster Monitoring           <ul style="list-style-type: none"> <li>・Prometheus + Grafana 等で構成されたOCPサポートのモニタリング環境。</li> </ul> </li> <li>2. セルフホスト監視ツール           <ul style="list-style-type: none"> <li>・代表例として、Prometheus, Zabbixなど。動作環境に選択肢あり。</li> </ul> </li> <li>3. SaaS製品           <ul style="list-style-type: none"> <li>・代表例として、Datadog, Mackerelなど</li> </ul> </li> </ol>
Logging	<ol style="list-style-type: none"> <li>1. OpenShift Cluster Logging           <p>Fluentd + ElasticSearch + Kibanaで構成されたOCPサポートのクラスタロギング環境。</p> </li> <li>2. セルフホストLoggingツール           <ul style="list-style-type: none"> <li>・代表例として、ElasticSearchなど</li> </ul> </li> <li>3. SaaS製品           <ul style="list-style-type: none"> <li>・代表例として、Datadog, CloudWatch logsなど</li> </ul> </li> </ol>
Tracing	<ol style="list-style-type: none"> <li>1. OpenShift Jaeger (<a href="#">Link</a>)           <p>OpenShiftの機能としてJaeger Operatorを提供。</p> </li> <li>2. セルフホストトレーシングツール           <ul style="list-style-type: none"> <li>・代表例として、JaegerやZipkin, Elastic APMなど</li> </ul> </li> <li>3. SaaS製品           <ul style="list-style-type: none"> <li>・代表例として、Datadog, Dynatrace, NewRelicなど</li> </ul> </li> </ol> <div data-bbox="1868 1000 2470 1317" style="border: 1px dashed black; padding: 10px;"> <p>その他</p> <ul style="list-style-type: none"> <li>・Probe</li> <li>・Error Tracking</li> <li>・Kubernetes Eventの収集</li> <li>など</li> </ul> </div>

# モニタリングの種類 (Observabilityの3本柱)

※製品が完全にカテゴリに分類されるわけではありません。そのツールの特色が強い部分を示しています。



---

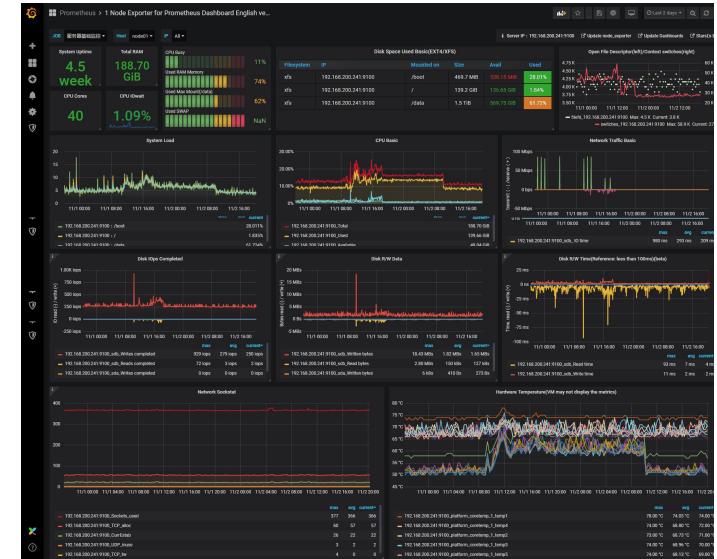
# メトリクス - OpenShift Monitoring

# Prometheus

CONFIDENTIAL designator



- ▶ Prometheusはオープンソースのメトリクスベースモニタリングシステム
  - ・ <https://github.com/prometheus/prometheus>
  - ・ CNCF(Cloud Native Computing Foundation)の2番目のメンバーでGraduated Projectのひとつ。
- ▶ Prometheusの特徴
  - ・ プル型のデータ取得アーキテクチャでスケールが容易
  - ・ サービスディスカバリ機能が充実。クラウドやコンテナ環境に最適化された仕組み
  - ・ 監視設定のコード化が容易
  - ・ 柔軟なクエリー(PromQL)
- ▶ 不向きなこと
  - ・ メトリクスベースのため、イベントログや個別のイベント情報の格納には不向き。



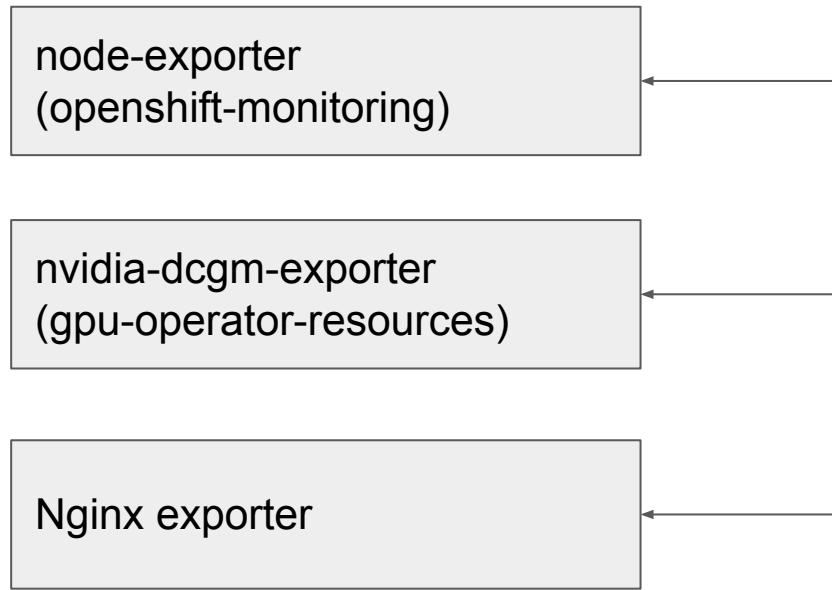
Grafanaと連携した可視化が容易

V00000000



# Prometheusとexporter, Grafana

exporterの例



Prometheusメトリクスの例

```
# HELP my_summary An example summary
# TYPE my_summary summary
my_summary_sum 0.6
my_summary_count 19
# HELP my_histogram An example histogram
# TYPE my_histogram histogram
latency_seconds_bucket{le="0.1"} 7
latency_seconds_bucket{le="0.2"} 18
latency_seconds_bucket{le="0.4"} 24
```

Alert manager



Prometheus

可視化



Grafana



ユーザ



- Cluster Monitoringを利用する
  - 裏側はPrometheus Operatorを活用
- Operatorを利用する
  - 構築が容易、イメージ管理不要
  - PrometheusやGrafanaの設定ファイル生成不要
  - 自由度に制限あり
- 自前で立てる
  - 最新バージョンが活用できる
  - 自由度は高いが、責任範囲も多い

v00000000

Red Hat

# OpenShift クラスタにおける監視対象 (1)

CONFIDENTIAL\_designator

ユーザのアプリケーション [1]



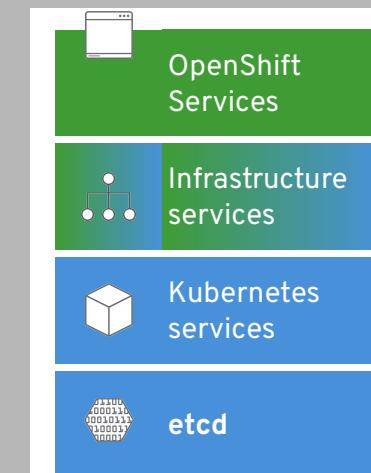
User Application

User Application

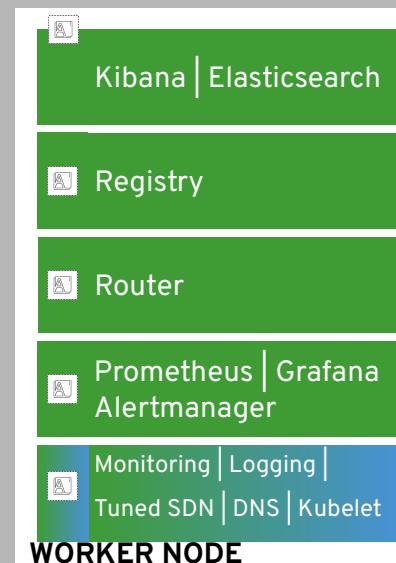
User Application

NAMESPACE

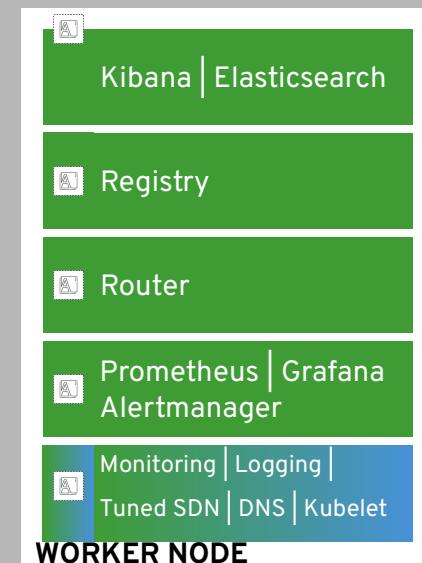
OpenShift クラスタ [2]



CONTROL PLANE



WORKER NODE



WORKER NODE

COMPUTE

NETWORK

STORAGE

OPENSIFT CLUSTER

[1] <https://docs.openshift.com/container-platform/4.9/monitoring/enabling-monitoring-for-user-defined-projects.html>

35

[2]

[https://docs.openshift.com/container-platform/4.9/monitoring/monitoring-overview.html#understanding-the-monitoring-stack\\_monitoring-overview](https://docs.openshift.com/container-platform/4.9/monitoring/monitoring-overview.html#understanding-the-monitoring-stack_monitoring-overview)

v0000000



# OpenShift クラスタにおける監視対象 (2)

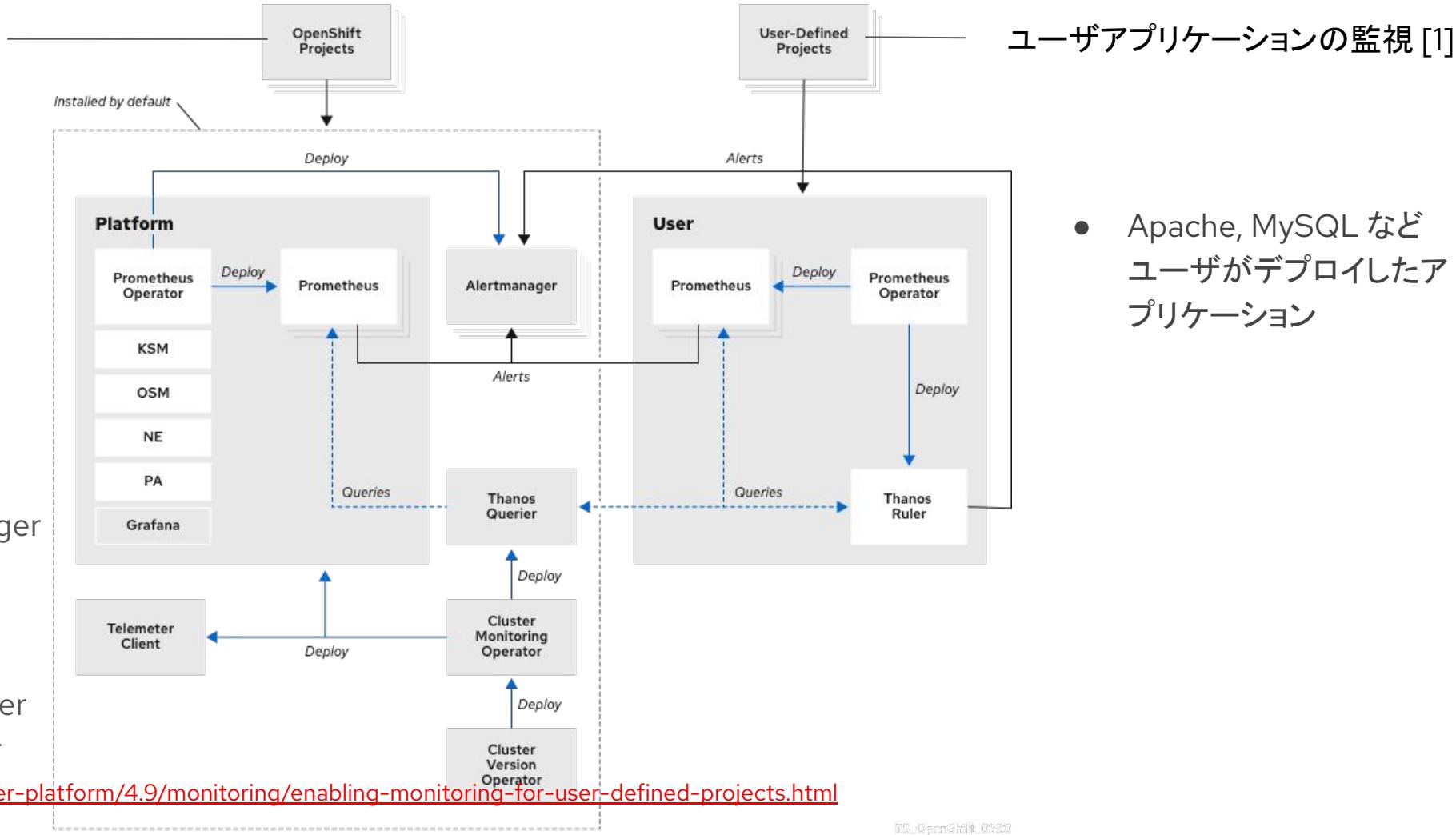
CONFIDENTIAL\_designator

## OpenShift クラスタの監視 [2]

- CoreDNS
- Elasticsearch, Fluentd [3]
- etcd
- HAProxy
- Image registry
- Kubelets
- Kubernetes apiserver
- Kubernetes controller manager
- Kubernetes scheduler
- Metering [3]
- OpenShift apiserver
- OpenShift controller manager
- Operator Lifecycle Manager

[(OLM)][docs.openshift.com/container-platform/4.9/monitoring/enabling-monitoring-for-user-defined-projects.html](https://docs.openshift.com/container-platform/4.9/monitoring/enabling-monitoring-for-user-defined-projects.html)

[2]

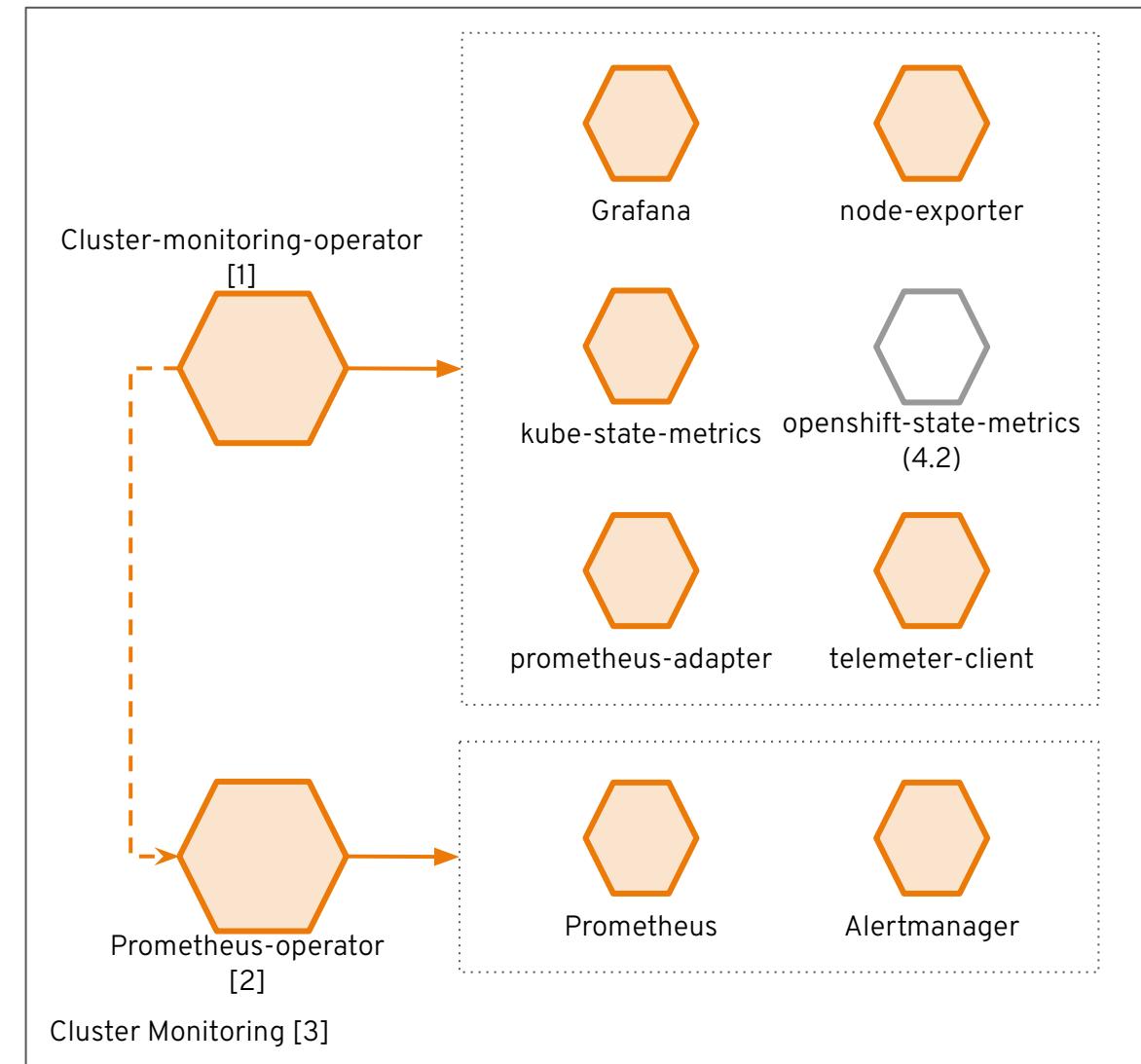
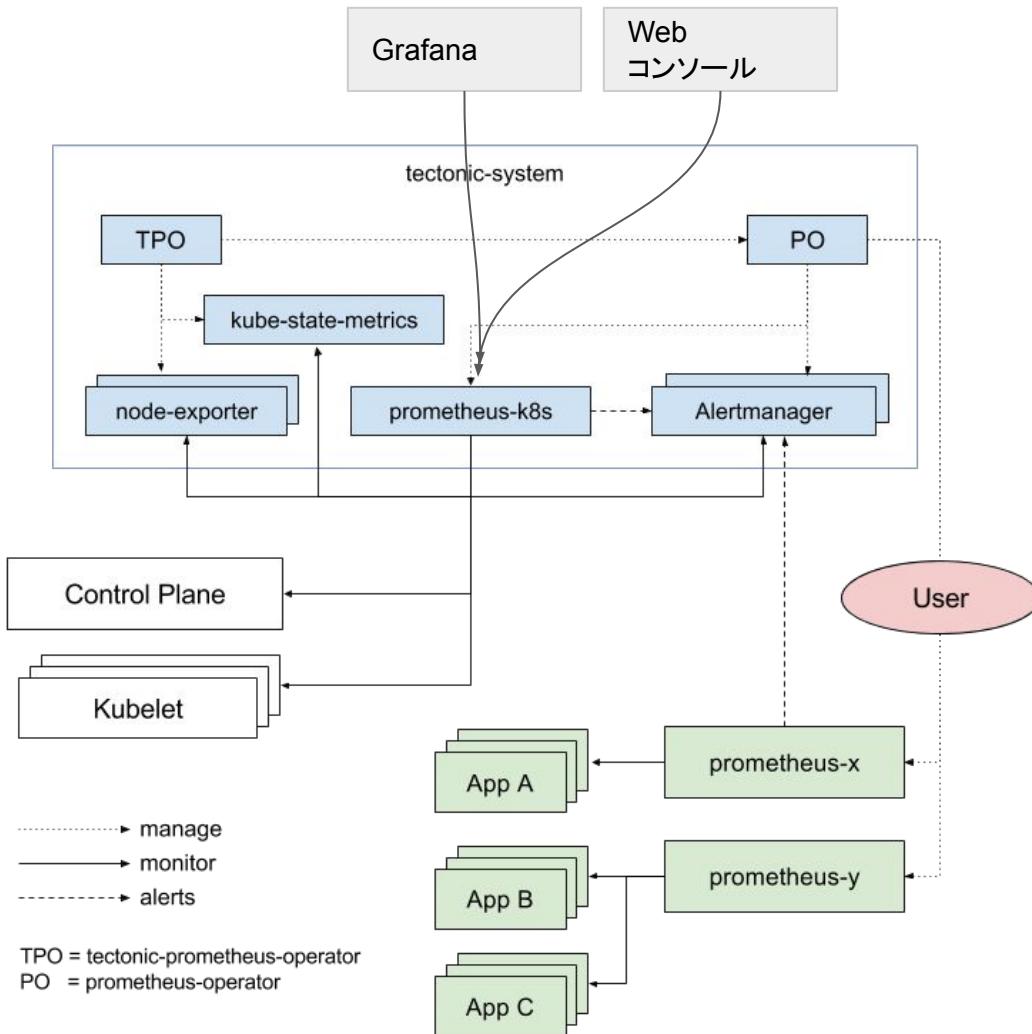


## ユーザアプリケーションの監視 [1]

- Apache, MySQL など  
ユーザがデプロイしたア  
プリケーション

# Cluster Monitoring のアーキテクチャ

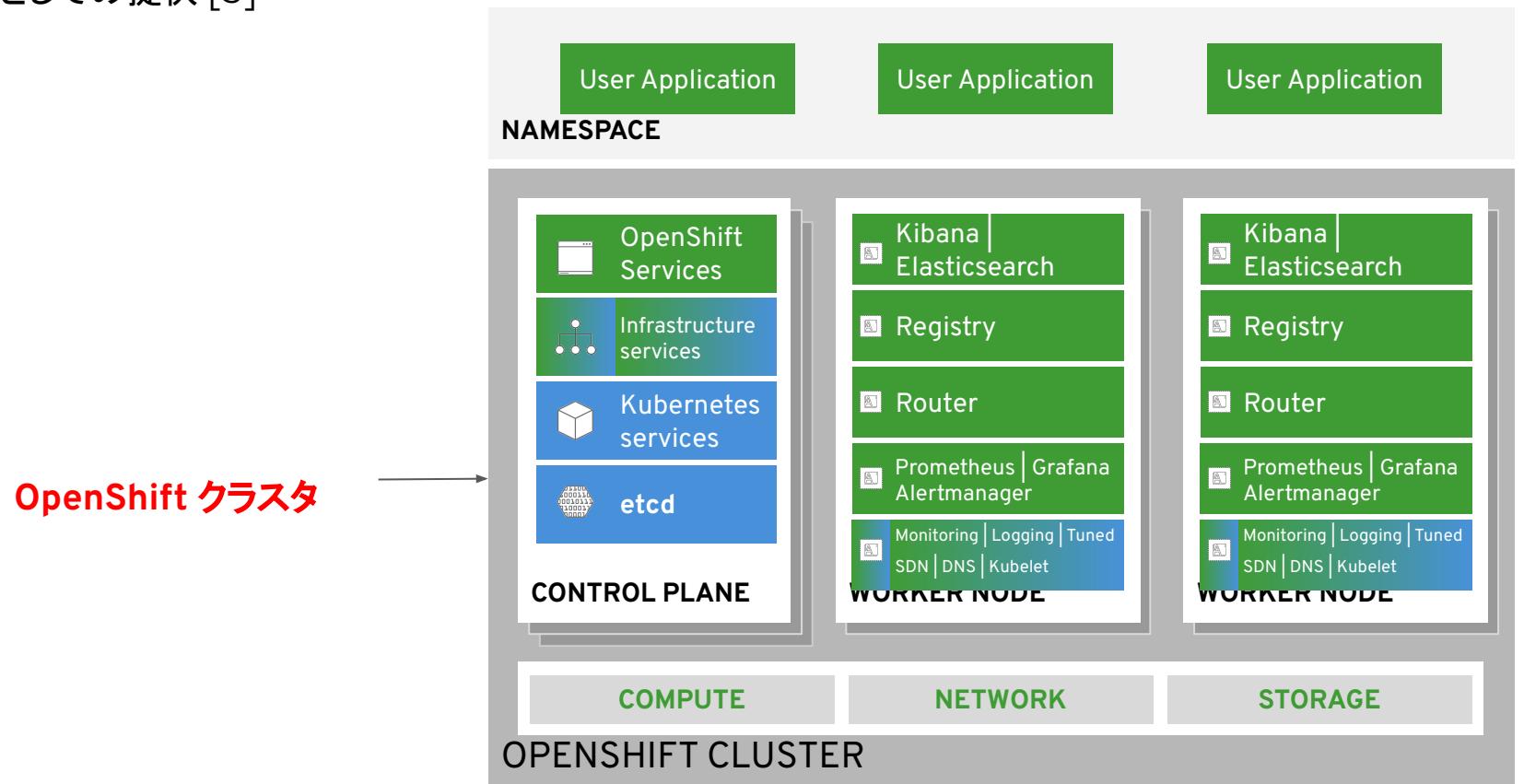
CONFIDENTIAL\_designator



# Cluster Monitoring の制約

CONFIDENTIAL\_designator

- ServiceMonitor の変更による収集対象の追加やアラートルールの追加または変更 [1]
- Grafana ダッシュボードのカスタマイズ [2]
- ダッシュボードのマルチテナントとしての提供 [3]



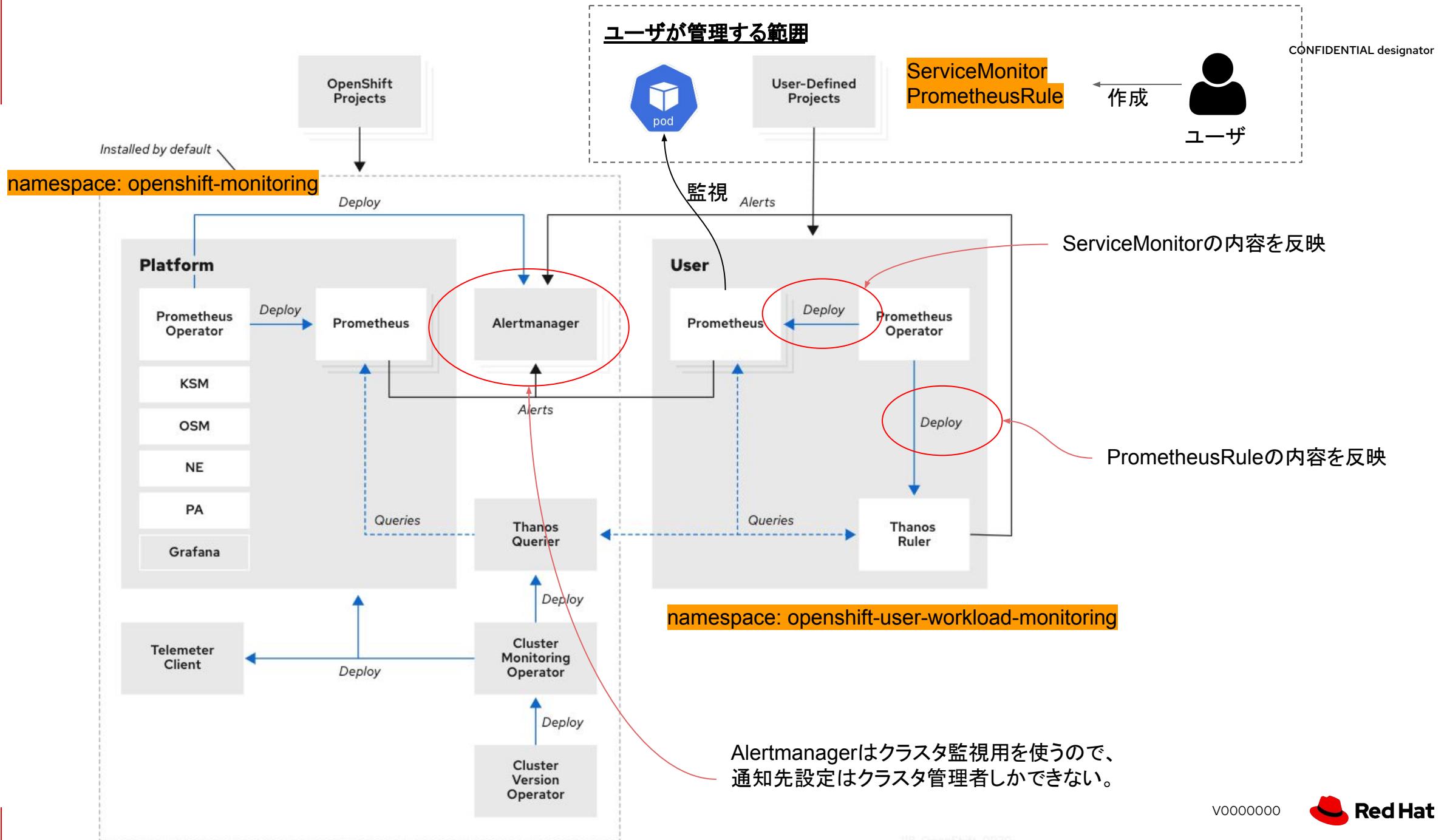
# Cluster Monitoring (monitor for user-defined project)

CONFIDENTIAL designator

OpenShift 4.6にて、Cluster Monitoringの拡張機能にてユーザ定義のプロジェクトの監視が可能になった。内部は、Prometheus Operatorで実装されており、ユーザ側で独自にPrometheus等を運用する必要がないマネージド機能として利用できる。

ユーザ側で対応が必要なことは以下の4つのみ。

1. 取得したい対象のアプリケーションでPrometheus metricsを出力させる。
  - a. 手法として、prometheus exporterをサイドカーコンテナで起動することや、アプリケーションのエンドポイントに埋め込むなどがある。
  - b. 代表例:nginx\_exporter, jmx\_exporter, mysql\_exporter ...
2. 監視対象のメトリクス取得のため“ServiceMonitor”リソースの作成を行う。
3. アラート発報のため、“PrometheusRule”リソースの作成を行う。
4. クラスタ管理者にアラート通知先の登録を依頼する



# 利用できるシーン

CONFIDENTIAL designator

Cluster Monitoringの拡張機能であるユーザ定義のプロジェクト監視について、次のような条件の場合に利用の検討をするとよい。

1. 監視サービス自身を管理することなく、手軽に監視を始めたい。
2. Grafanaダッシュボードは必要としていない。
  - a. Grafanaダッシュボードを用いた運用が必要な場合は、Prometheus Operator, Grafana Operatorなどを用いて開発チーム専用の監視環境をセットアップする必要がある。
3. アラート通知先を開発チーム側で管理する必要がない。

# OpenShift 実習2

OpenShiftでのコンテナアプリケーションのメトリクスの取得方法を理解する

<https://github.com/k-srkw/openshift-monitoring-hands-on/blob/main/monitoring-hands-on.md>

---

# ロギング - OpenShift Logging

# コンテナのログの確認

コンテナのログは、標準出力・標準エラー出力している場合、oc logsコマンドか、Webコンソールの”Logs”から確認することができる。単一のコンテナのログが確認でき、複数レプリカで動作している場合はそれぞれのコンテナを確認する必要がある。

## Webコンソール

The screenshot shows the OpenShift Web Console interface. At the top, there's a navigation bar with icons for dashboard, projects, and user admin. Below that is a dropdown for the project 'openshift-monitoring'. The main area shows a 'Pods' list with one item: 'grafana-7674648c6d-2bf7p' which is 'Running'. Below the list are tabs for 'Details', 'YAML', 'Environment', 'Logs' (which is selected), 'Events', and 'Terminal'. Under the 'Logs' tab, there's a button to 'Log streaming...' and a dropdown set to 'grafana'. On the right, there are links for 'Raw', 'Download', and 'Expand'. The log output itself starts with '324 lines' and continues with several lines of log entries from April 27, 2021, at 02:26:58, showing various info and error messages related to Grafana's startup and configuration.

## OCコマンド

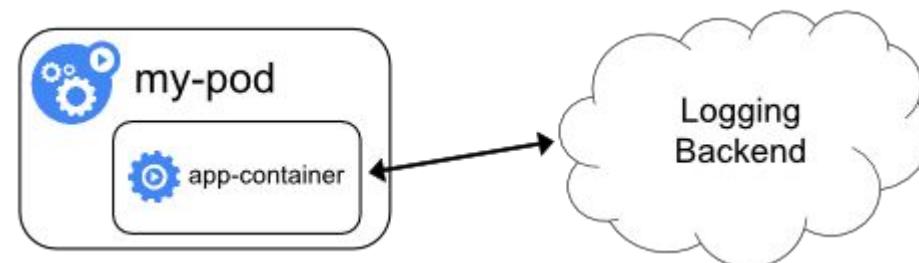
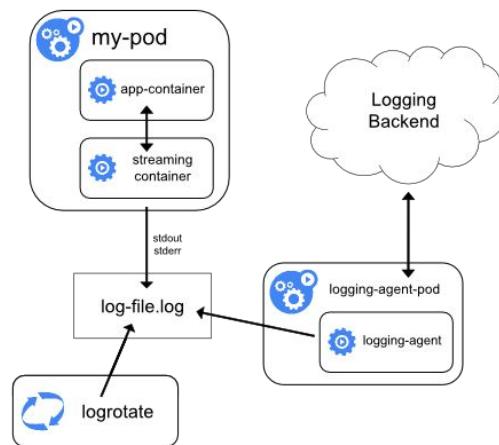
```
% oc logs -f debug-nginx-7968cd6c65-kbgtw
10.129.0.1 - - [09/Jun/2020:11:51:32 +0000] "GET / HTTP/1.1" 200 612 "-" "kube-probe/1.16+" "-"
10.129.0.1 - - [09/Jun/2020:11:51:33 +0000] "GET / HTTP/1.1" 200 612 "-" "kube-probe/1.16+" "-"
10.130.0.14 - - [09/Jun/2020:11:51:34 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.64.1" "14.3.84.47"
10.129.0.1 - - [09/Jun/2020:11:51:37 +0000] "GET / HTTP/1.1" 200 612 "-" "kube-probe/1.16+" "-"
10.129.0.1 - - [09/Jun/2020:11:51:38 +0000] "GET / HTTP/1.1" 200 612 "-" "kube-probe/1.16+" "-"
10.130.0.14 - - [09/Jun/2020:11:51:38 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.64.1" "14.3.84.47"
10.130.0.14 - - [09/Jun/2020:11:51:40 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.64.1" "14.3.84.47"
10.129.0.1 - - [09/Jun/2020:11:51:42 +0000] "GET / HTTP/1.1" 200 612 "-" "kube-probe/1.16+" "-"
10.129.0.1 - - [09/Jun/2020:11:51:43 +0000] "GET / HTTP/1.1" 200 612 "-" "kube-probe/1.16+" "-"
10.129.0.2 - - [09/Jun/2020:11:51:45 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.64.1" "14.3.84.47"
10.129.0.1 - - [09/Jun/2020:11:51:47 +0000] "GET / HTTP/1.1" 200 612 "-" "kube-probe/1.16+" "-"
10.129.0.1 - - [09/Jun/2020:11:51:48 +0000] "GET / HTTP/1.1" 200 612 "-" "kube-probe/1.16+" "-"
10.129.0.2 - - [09/Jun/2020:11:51:49 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.64.1" "14.3.84.47"
10.129.0.2 - - [09/Jun/2020:11:51:51 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.64.1" "14.3.84.47"
```

# Kubernetes のロギングパターン

CONFIDENTIAL designator

Kubernetesの公式ドキュメント[1]にてロギングパターンを公開している。大きく分けて3つの方法があり、適材適所で使い分けていくことが求められる。

1. Using a node logging agent
2. Using a sidecar container with the logging agent
  - a. Streaming sidecar container
  - b. Sidecar container with a logging agent
3. Exposing logs directly from the application



[1] <https://kubernetes.io/docs/concepts/cluster-administration/logging/>

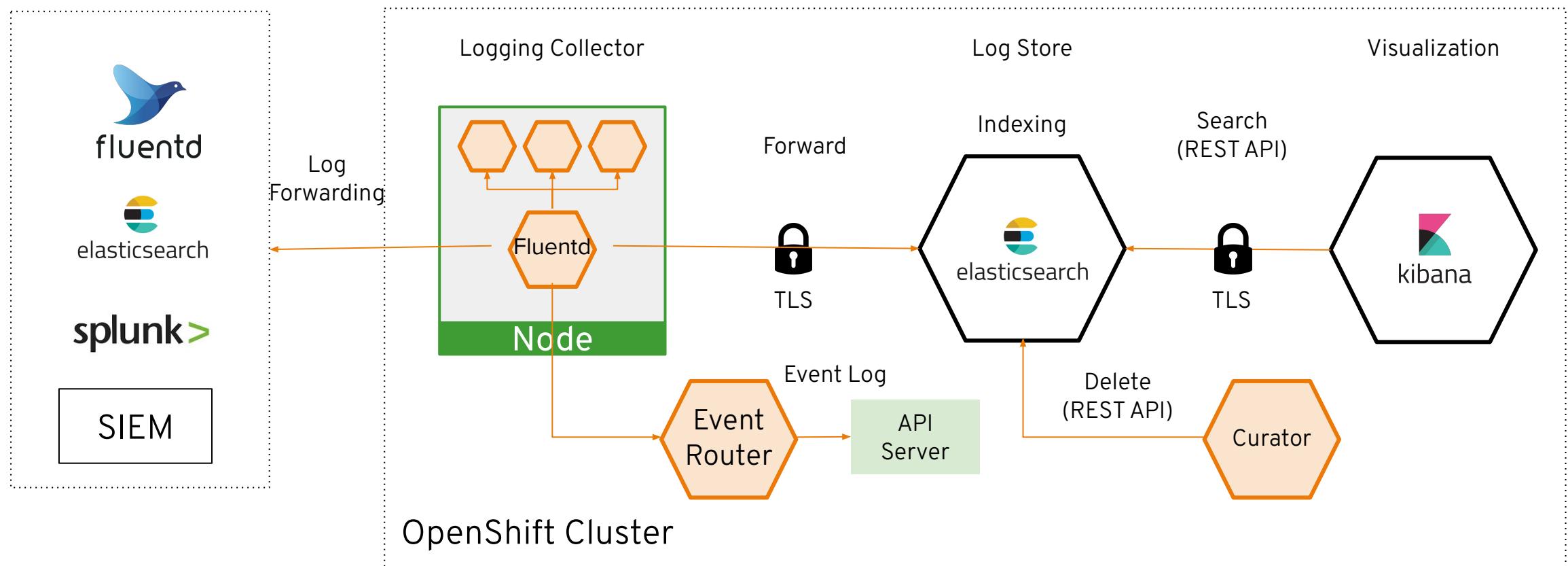
V0000000



# OpenShift Cluster Logging

CONFIDENTIAL designator

- Cluster Logging は Logging Collector (Fluentd), Log Store (Elasticsearch), Visualization (Kibana), Curator から構成される [1]
- Event Router は API Server から Event Log を取得し、他のコンテナと同様に Fluentd がログを転送する
- OpenShift 外部の Elasticsearch や Splunk などへのログの転送をサポートする [2]



[1] <https://docs.openshift.com/container-platform/4.6/logging/cluster-logging.html>

[2] <https://docs.openshift.com/container-platform/4.6/logging/cluster-logging-external.html>

[3] vRAN 基盤の場合 Elastic 社の Elastic Cloud on Kubernetes (ECK) を利用する想定 <https://www.elastic.co/guide/en/cloud-on-k8s/current/index.html>

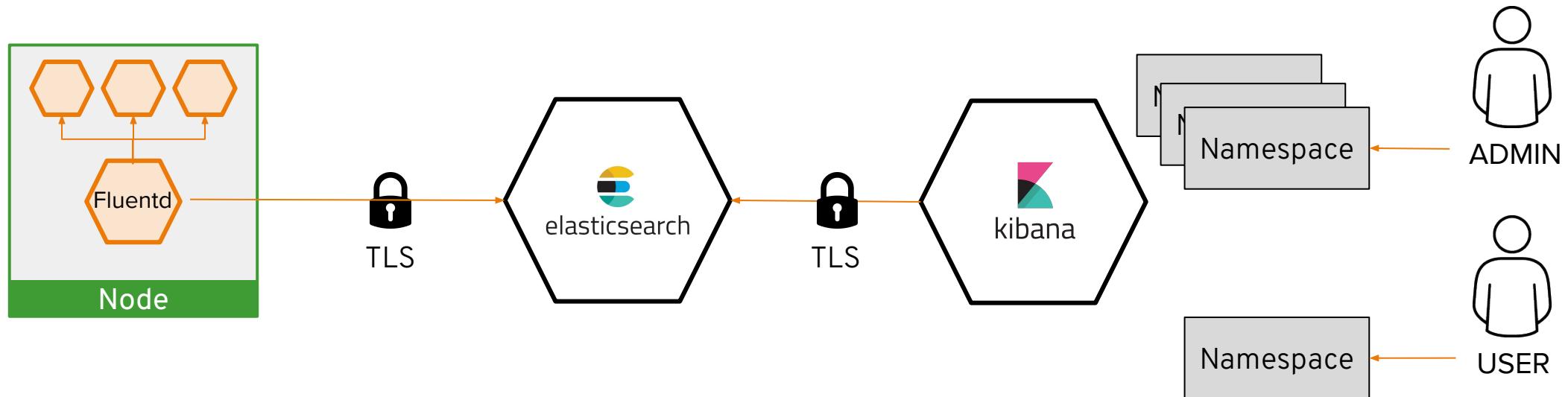
v0000000

 Red Hat

# Cluster Logging とマルチテナント

CONFIDENTIAL\_designator

- ログへのアクセス管理は Role-based access control (RBAC) を用いて Namespace 単位で適用される [1]
- クラスタ管理者 (cluster-admin) は全ての Namespace を横断して全てのログを参照することができ、ユーザは権限が与えられた Namespace に限定される



# OpenShift 実習4

Kibana でアプリログを見る。

1. Web コンソール URL にアクセスし、割り当てられた開発者向けユーザ名 user# でログインする。
2. Webコンソール > 画面右上から「Logging↑」
3. Management > Index Patterns > Create index pattern  
Index pattern: app-\*  
Time Filter field name: @timestamp
4. Discover 画面でログを見る。
  - a. Available fields から message を Add
  - b. 検索ボックスから全文検索
  - c. 「Add a filter +」でフィールド値で絞り込み
  - d. 画面右上で対象期間指定
5. Kibanaの詳細な使い方は、Elastic社のマニュアル参照  
<https://www.elastic.co/guide/en/kibana/current/index.html>



# Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.



[linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)



[facebook.com/redhatinc](https://www.facebook.com/redhatinc)



[youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)



[twitter.com/RedHat](https://twitter.com/RedHat)