Kubernetes クラスタ運用に必要なこと

～Public Cloud and Private Cloud～

@オープンクラウドカンファレンス 2019

Masaya Aoyama

CyberAgent

amsy810

@amsy810

(Introduction CloudNative +)
Kubernetes クラスタ運用に必要なこと

～Public Cloud and Private Cloud～

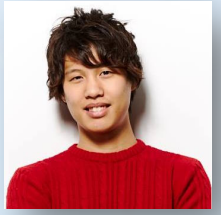
@オープンクラウドカンファレンス 2019

Masaya Aoyama

CyberAgent

amsy810

@amsy810



Masaya Aoyama (@amsy810)

Infrastructure Engineer

CyberAgent

+ CREATIONLINE - 技術アドバイザ

+ SAKURA Internet Research Center – 客員研究員

主業務:
K8s as a Service の実装
K8s / CloudNative 関連のアーキテクト

AKE
Adtech Container Engine.

CERTIFIED kubernetes ADMINISTRATOR
CERTIFIED kubernetes APPLICATION DEVELOPER

Publicity（一部抜粋）

書籍『Kubernetes 完全ガイド』『みんなの Docker/K8s』

基調講演『Japan Container Days v18.04』『Google Cloud K8s Day』

招待講演『情報処理学会 コンピュータシステムシンポジウム』
『AWS Dev Day Tokyo』『IBM Think Japan』『JEITA 委員会』

登壇『KubeCon + CNCon China 2019』『Open Source Summit 2019』等

資格『CKAD #2』『CKA #138』

Community

Co-chair 『Cloud Native Days Tokyo（旧 Japan Container Days）』

Organizer『Cloud Native Meetup Tokyo』
『Kubernetes Meetup Tokyo』
『KubeCon Japanese exchange meeting』

Contribute to OpenStack and Kubernetes

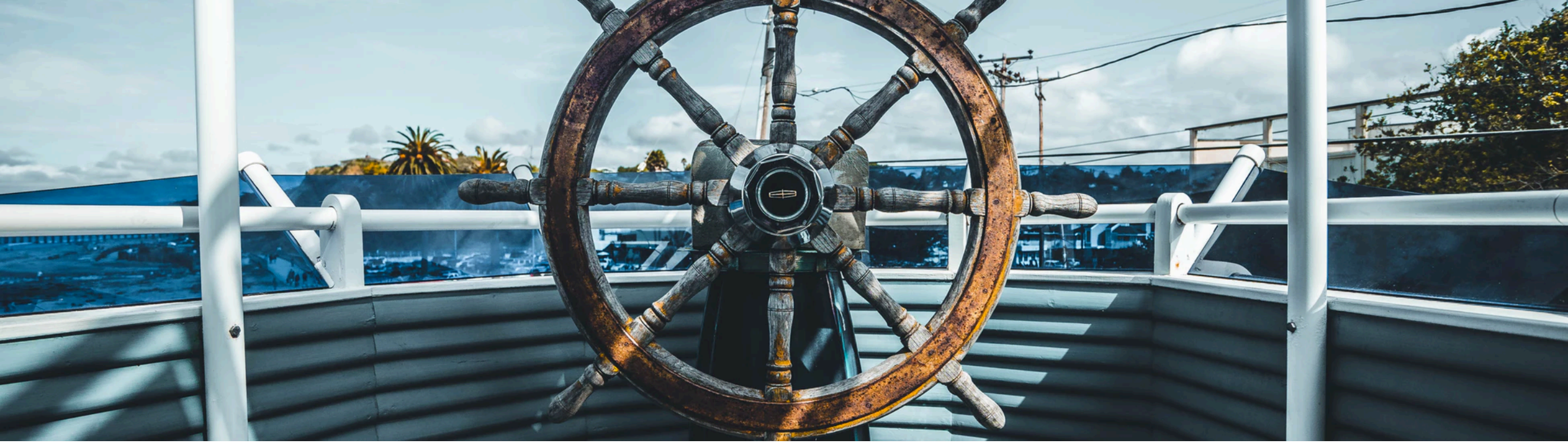
CLOUDNATIVEDAYS

CLOUD NATIVE Meetup Tokyo

tokyo

Agenda

- Cloud Native and Kubernetes
- Microservice and Service Mesh
- Container and Docker
- Kubernetes Overview
- Managing Kubernetes Cluster



Cloud Native and Kubernetes

What is Kubernetes? What is Cloud Naitve?

CNCF and The Linux Foundation

- KubernetesはThe Linux Foundationのサブプロジェクトであるの CNCFによってホスト
- その他にも多くのプロジェクトをホストしている

Cloud Nativeとは？

CNCF Cloud Native Defenition v1.0, CNCF, 2018-11-28
(https://github.com/cncf/toc/blob/master/DEFINITION.md)

Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

The Cloud Native Computing Foundation seeks to drive adoption of this paradigm by fostering and sustaining an ecosystem of open source, vendor-neutral projects. We democratize state-of-the-art patterns to make these innovations accessible for everyone.

Cloud Nativeとは？

CNCF Cloud Native Defenition v1.0, CNCF, 2018-11-28
(https://github.com/cncf/toc/blob/master/DEFINITION.md)

Cloud native te alable
applications in e, and hybrid
clouds. Contai tructure, and
declarative AP

These techniq manageable,
and observabl ers to make
high-impact ch

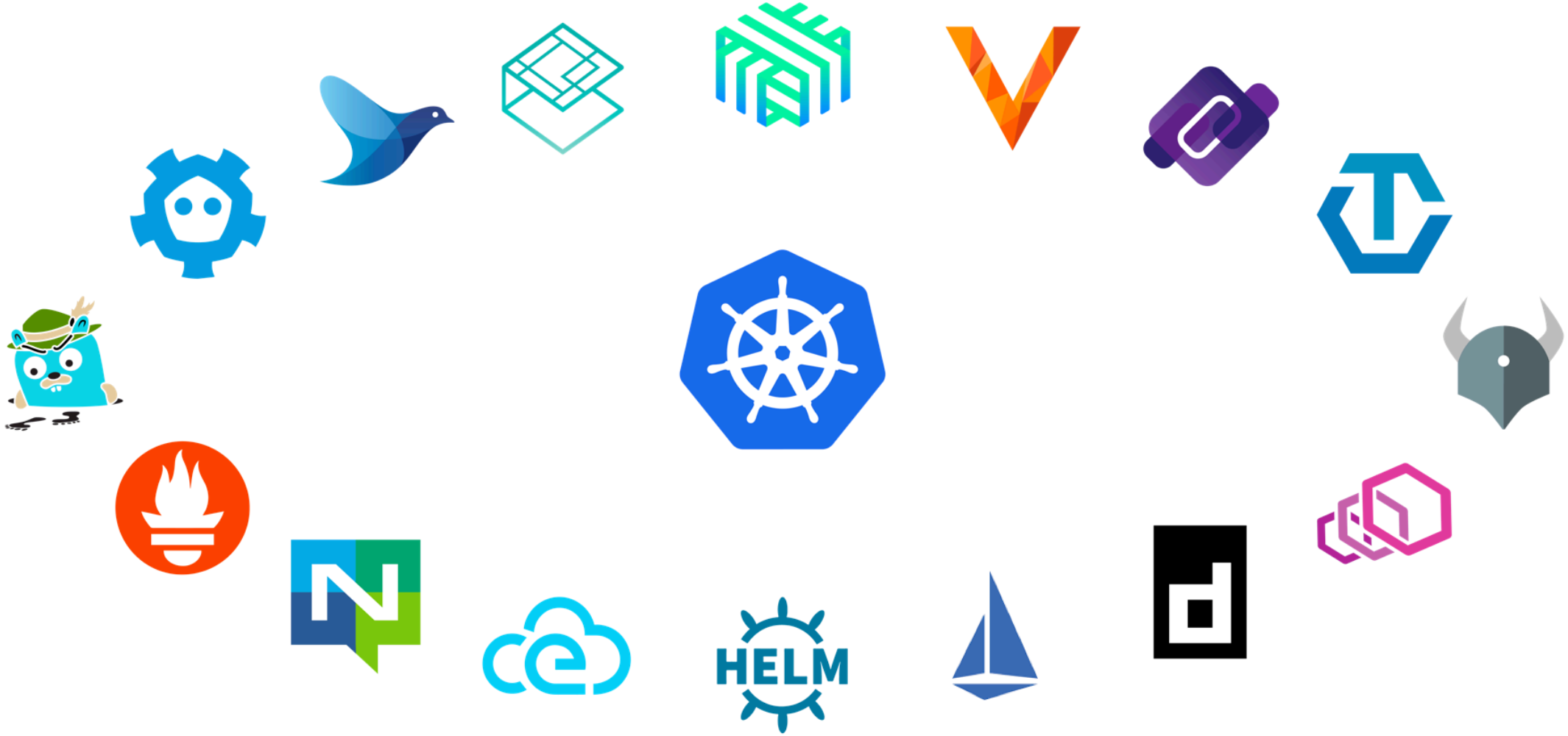
The Cloud Nat his paradigm by
fostering and sustaining an ecosystem o pen source, vendor-neutral projects. We
democratize state-of-the-art patterns to e these innovations accessible for
everyone.

- 疎結合なシステム
- 復元力がある
- 管理しやすい
- 可観測である
- 堅牢な自動化により、頻繁かつ期待通りに最小限の労力で大きな変更が可能

OpenかつScalableなシステムを実現

Cloud Native Ecosystem

Kubernetes is key product for Cloud Native Computing



Cloud Native Landscape
2018-11-25T06:41:03Z dedc385

See the interactive landscape at l.cncf.io

Greyed logos are not open source

App Definition and Development

Database
Vitess (CNCF Incubating)

Streaming & Messaging
NATS (CNCF Incubating)

Application Definition & Image Build
HELM (CNCF Incubating)

Continuous Integration & Delivery

Orchestration & Management

Scheduling & Orchestration
kubernetes (CNCF Graduated)

Coordination & Service Discovery
CoreDNS (CNCF Incubating)

Remote Procedure Call
GRPC (CNCF Incubating)

Service Proxy
envoy (CNCF Incubating)

API Gateway

Service Mesh
LINKERD (CNCF Incubating)

Runtime

Cloud-Native Storage
ROOK (CNCF Incubating)

Container Runtime
CNI (CNCF Incubating)

Cloud-Native Network

Provisioning

Automation & Configuration

Container Registry
HARBOR (CNCF Incubating)

Security & Compliance
Notary (CNCF Incubating)
TUF (CNCF Incubating)

Key Management

Cloud

Public

CLOUD NATIVE Landscape
l.cncf.io

Special

Kubernetes Certified Service Provider

Kubernetes Training Partner

Platform

Certified Kubernetes - Distribution

Certified Kubernetes - Hosted

Certified Kubernetes - Installer

PaaS/Container Service

Observability and Analysis

Monitoring
Prometheus (CNCF Graduated)

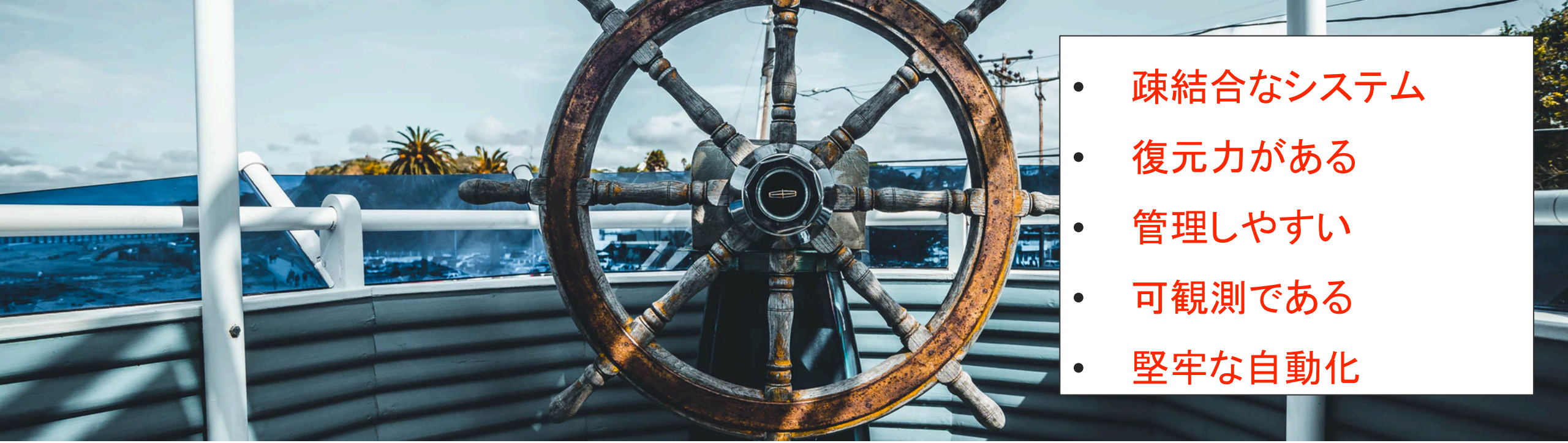
Logging
fluentd (CNCF Incubating)

Tracing
JAEGER (CNCF Incubating)
OPENTRACING (CNCF Incubating)

Chaos Engineering

Serverless

Cloud Native に適したアーキテクチャ？
大規模な開発 に適したアーキテクチャ？

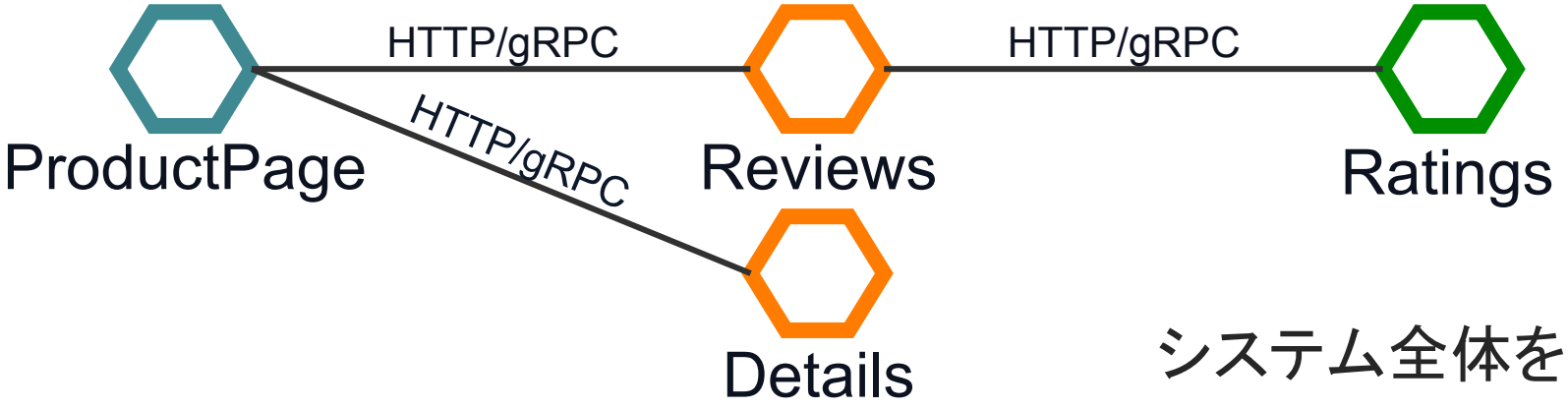


- 疎結合なシステム
- 復元力がある
- 管理しやすい
- 可観測である
- 堅牢な自動化

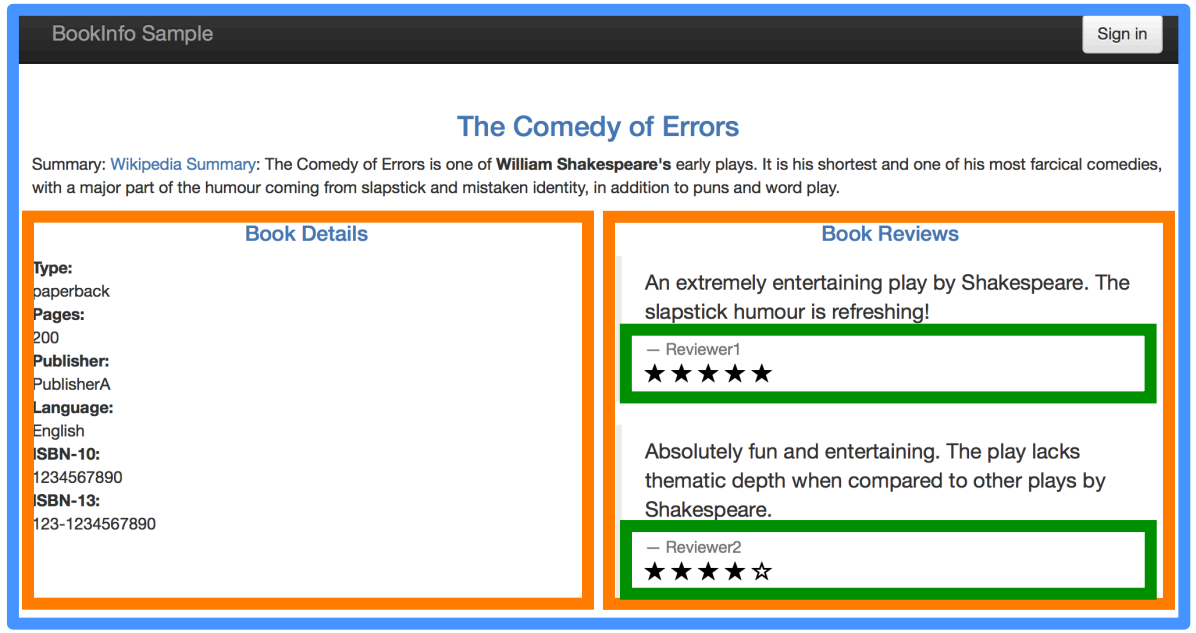
Microservice and Service Mesh

Benefit of Microservice and Service Mesh

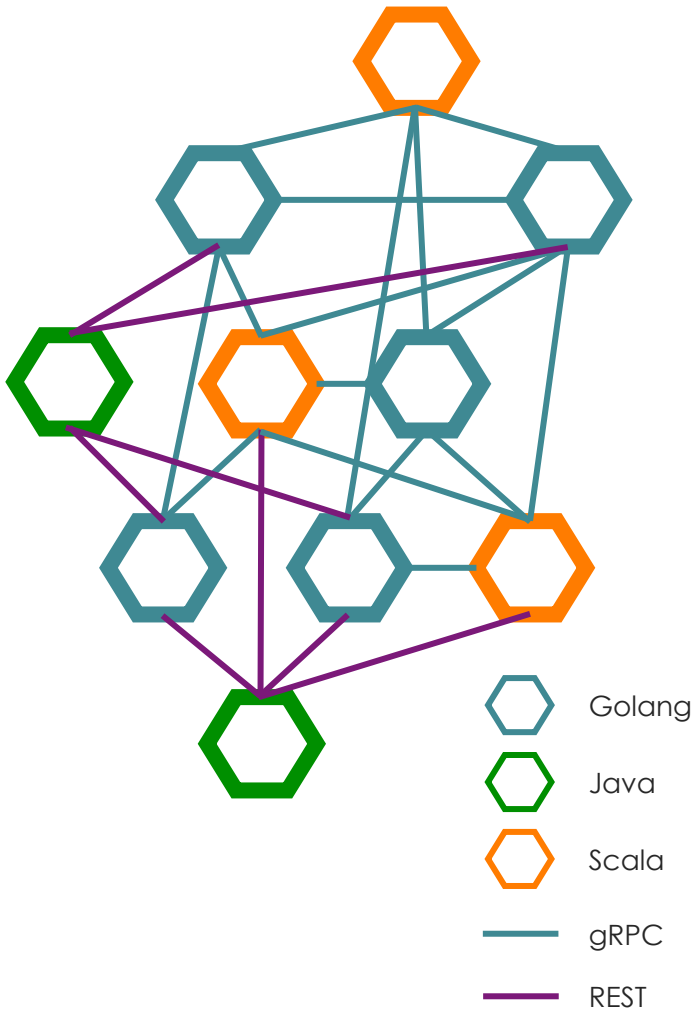
Microservice Architecture



システム全体を機能ごとに分離



Benefit of Microservice



マイクロサービス毎に技術選定可能

gRPC, REST 等でマイクロサービス間を繋ぐことで、
各サービスの技術選定の自由度が高い

大規模な開発を加速させる

各部門が特定のマイクロサービスを開発することで全体が成立
機能追加も比較的しやすい

デプロイが容易

小さい機能単位でデプロイが可能なため、
高速かつ影響範囲が少ない

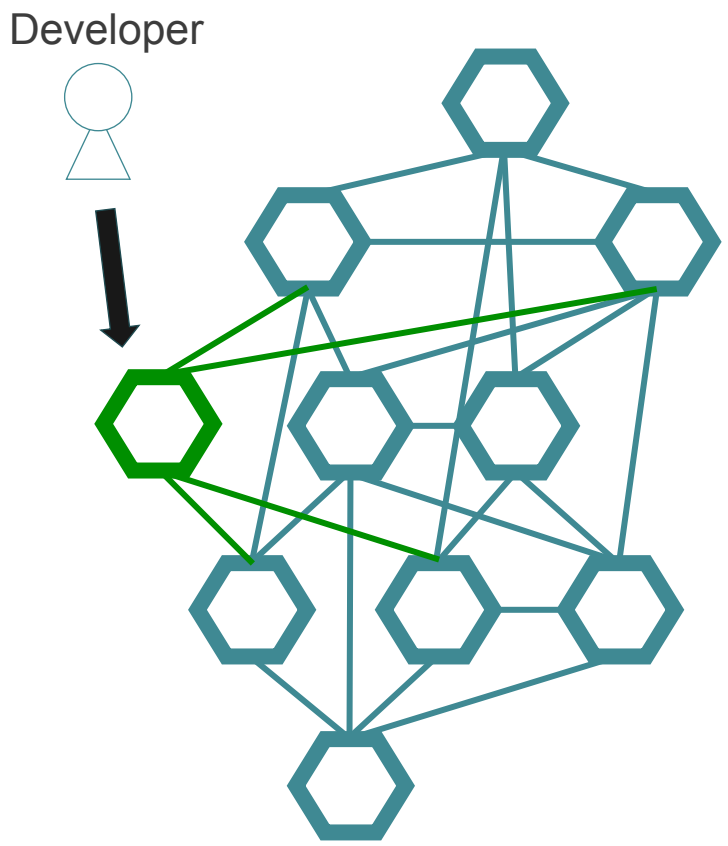
スケーリングが容易

特定の機能だけスケーリングさせるため効率が良い

障害が全体に波及しづらい

特定の機能に障害が起きた場合でも、
縮退した状態でサービス継続を行いやすい

Benefit of Microservice



マイクロサービス毎に技術選定可能

gRPC, REST 等でマイクロサービス間を繋ぐことで、
各サービスの技術選定の自由度が高い

大規模な開発を加速させる

各部門が特定のマイクロサービスを開発することで全体が成立
機能追加も比較的しやすい

デプロイが容易

小さい機能単位でデプロイが可能なため、
高速かつ影響範囲が少ない

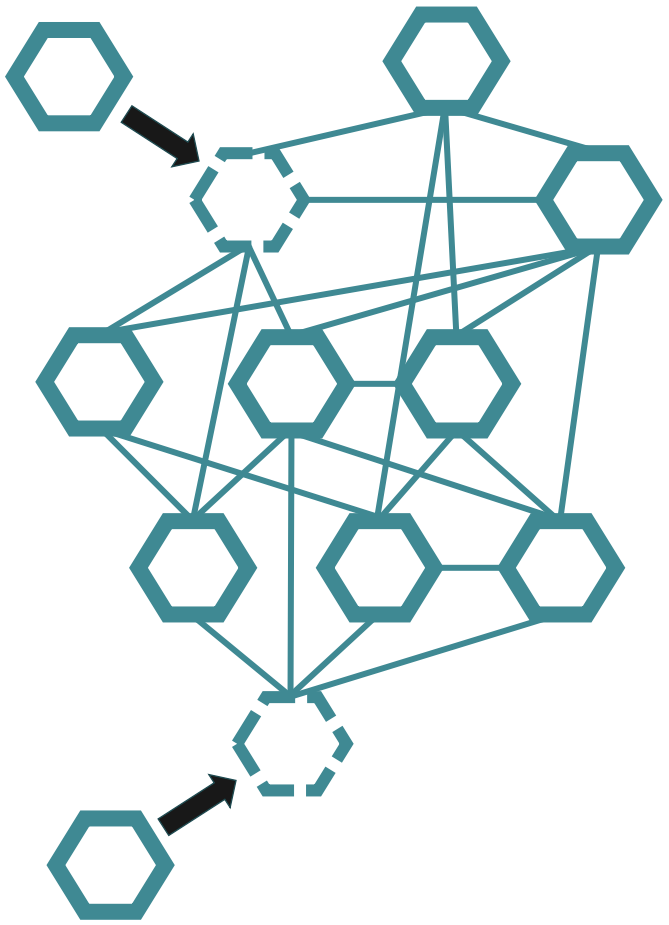
スケーリングが容易

特定の機能だけスケーリングさせるため効率が良い

障害が全体に波及しづらい

特定の機能に障害が起きた場合でも、
縮退した状態でサービス継続を行いやすい

Benefit of Microservice



マイクロサービス毎に技術選定可能

gRPC, REST 等でマイクロサービス間を繋ぐことで、
各サービスの技術選定の自由度が高い

大規模な開発を加速させる

各部門が特定のマイクロサービスを開発することで全体が成立
機能追加も比較的しやすい

デプロイが容易

小さい機能単位でデプロイが可能なため、
高速かつ影響範囲が少ない

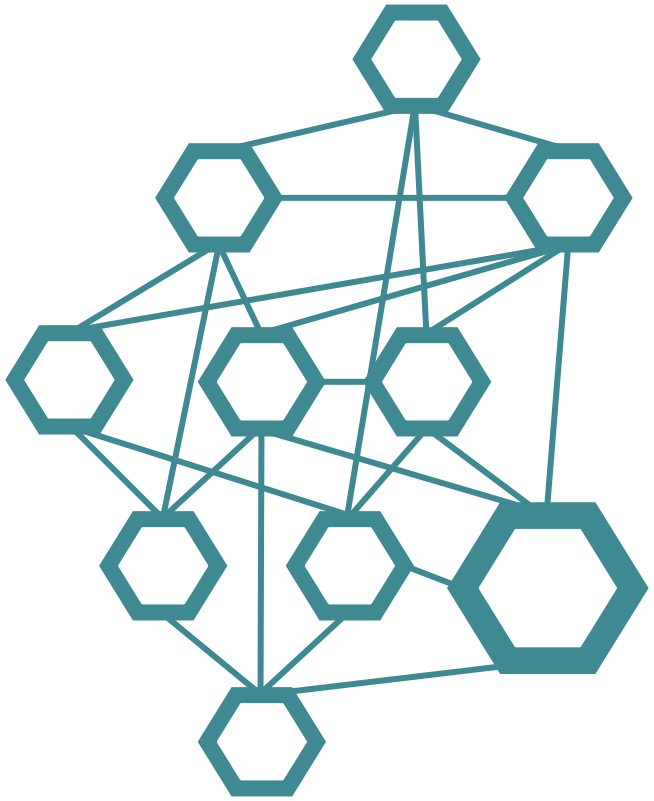
スケーリングが容易

特定の機能だけスケーリングさせるため効率が良い

障害が全体に波及しづらい

特定の機能に障害が起きた場合でも、
縮退した状態でサービス継続を行いやすい

Benefit of Microservice



マイクロサービス毎に技術選定可能

gRPC, REST 等でマイクロサービス間を繋ぐことで、
各サービスの技術選定の自由度が高い

大規模な開発を加速させる

各部門が特定のマイクロサービスを開発することで全体が成立
機能追加も比較的しやすい

デプロイが容易

小さい機能単位でデプロイが可能なため、
高速かつ影響範囲が少ない

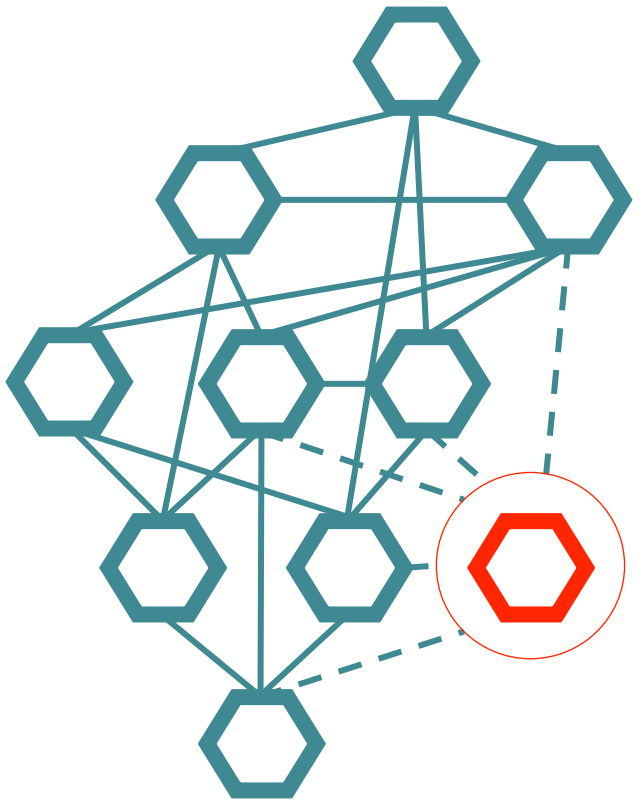
スケーリングが容易

特定の機能だけスケーリングさせるため効率が良い

障害が全体に波及しづらい

特定の機能に障害が起きた場合でも、
縮退した状態でサービス継続を行いやすい

Benefit of Microservice



マイクロサービス毎に技術選定可能

gRPC, REST 等でマイクロサービス間を繋ぐことで、
各サービスの技術選定の自由度が高い

大規模な開発を加速させる

各部門が特定のマイクロサービスを開発することで全体が成立
機能追加も比較的しやすい

デプロイが容易

小さい機能単位でデプロイが可能なため、
高速かつ影響範囲が少ない

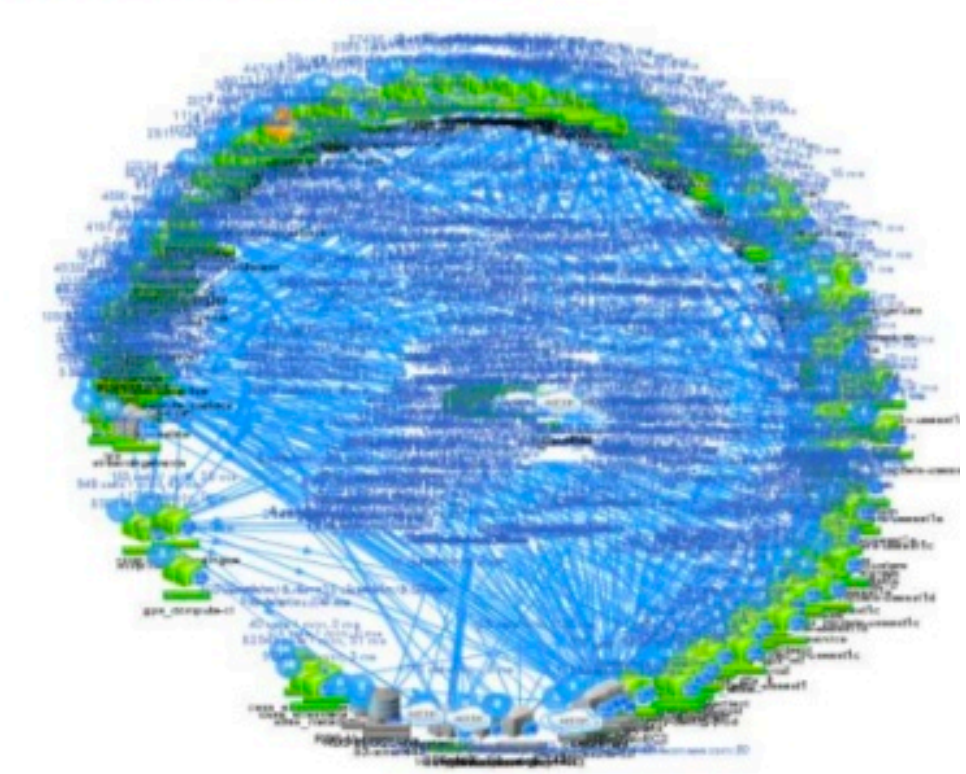
スケーリングが容易

特定の機能だけスケーリングさせるため効率が良い

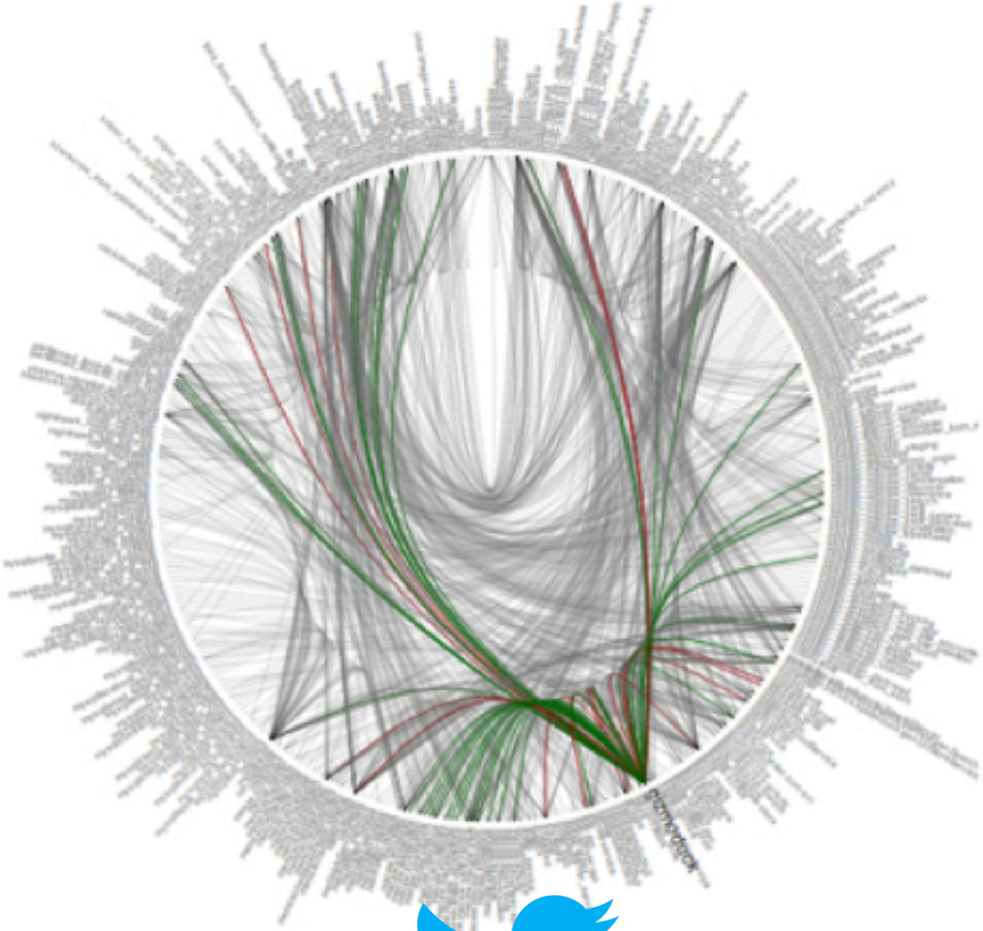
障害が全体に波及しづらい

特定の機能に障害が起きた場合でも、
縮退した状態でサービス継続を行いやすい

500+ microservices

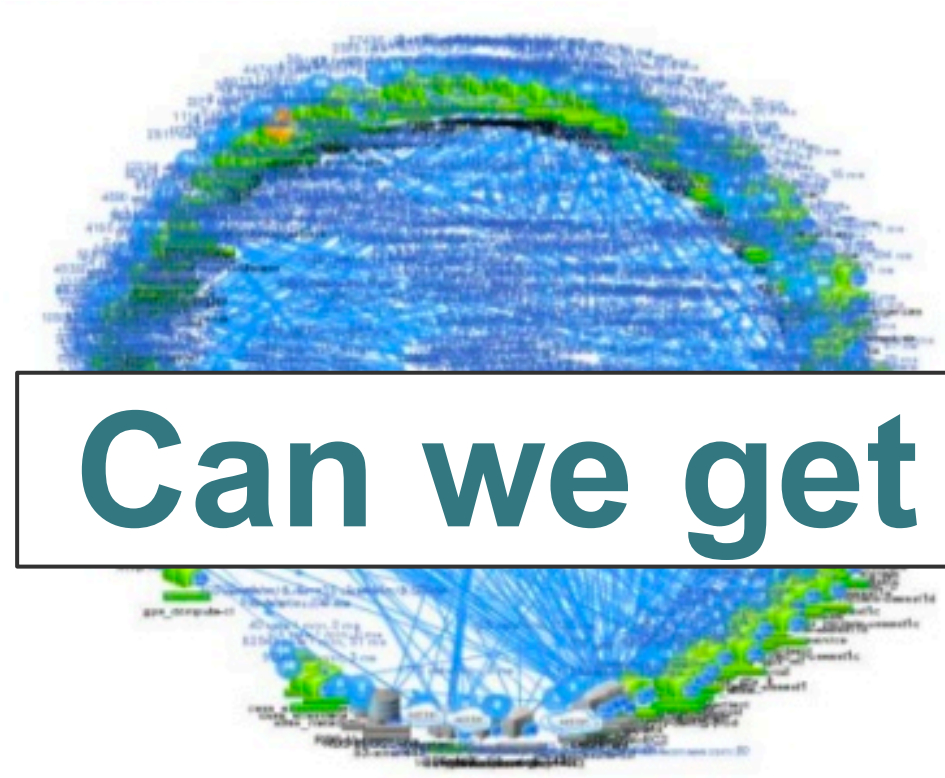


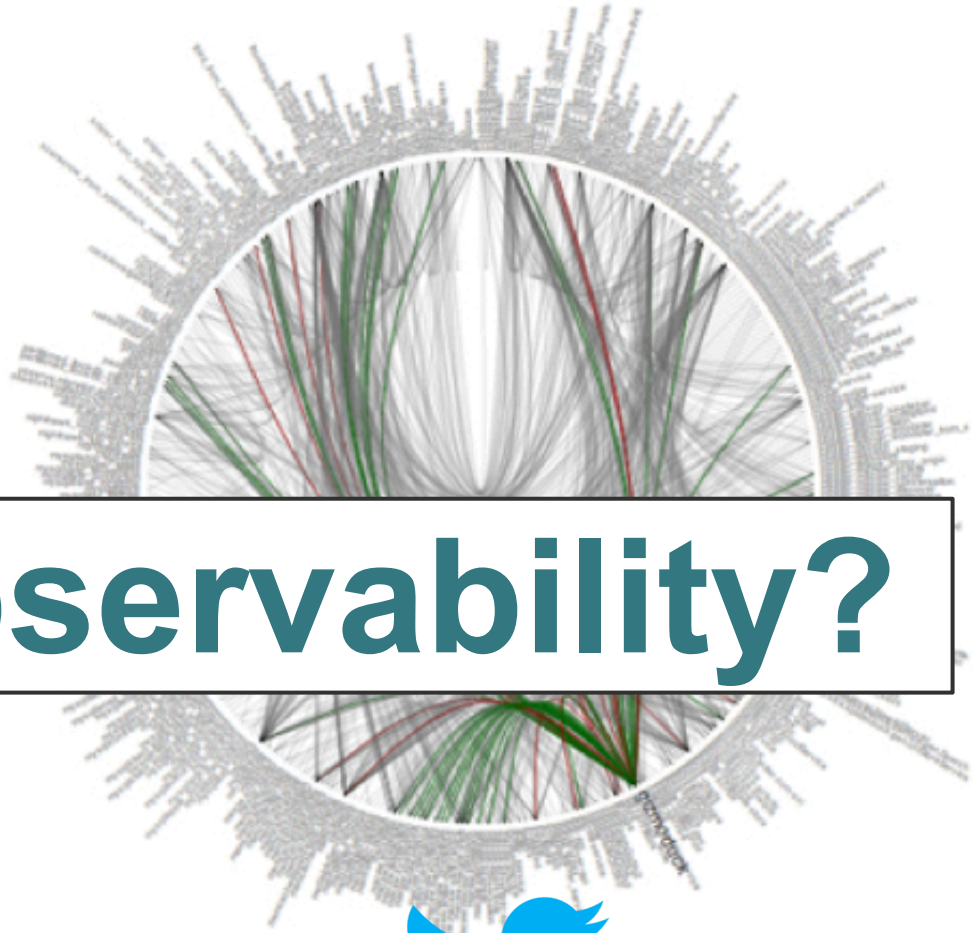
NETFLIX



The Case of Chaos, Bruce Wong, 2014-12-19, (https://www.slideshare.net/BruceWong3/the-case-for-chaos)
The History of the Service Mesh, The New Stack, 2018-2-13, (https://thenewstack.io/history-service-mesh/)

500+ microservices





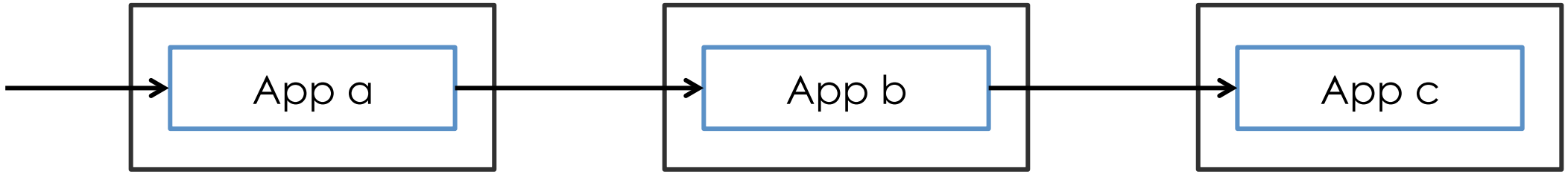
Can we get Observability?

NETFLIX

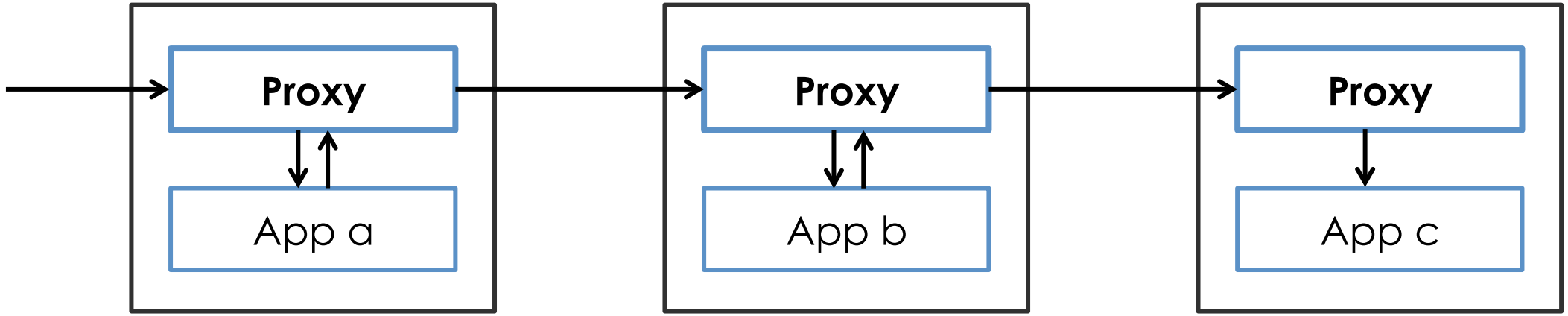
The Case of Chaos, Bruce Wong, 2014-12-19, (https://www.slideshare.net/BruceWong3/the-case-for-chaos)
The History of the Service Mesh, The New Stack, 2018-2-13, (https://thenewstack.io/history-service-mesh/)

Service Mesh Architecture (generally)

- 通常のMicroservice

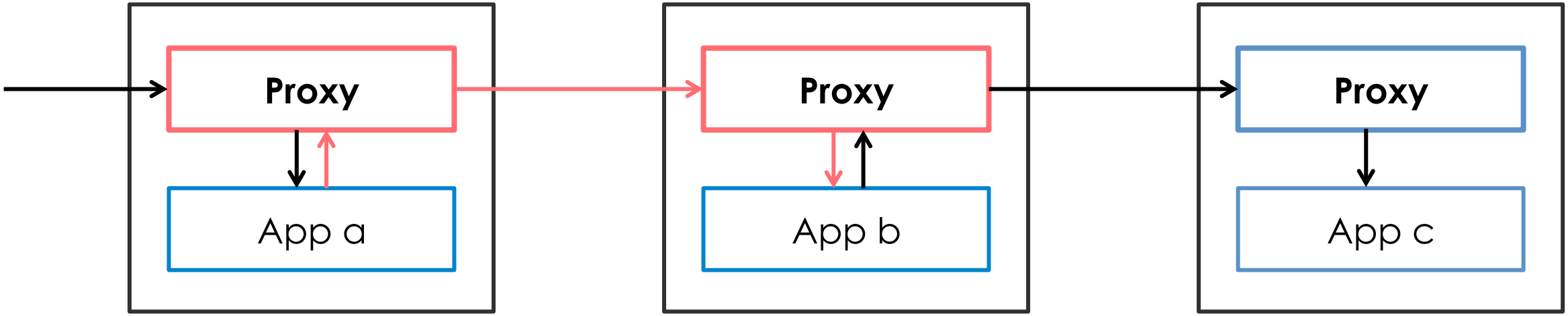


- Microservice with Service Mesh



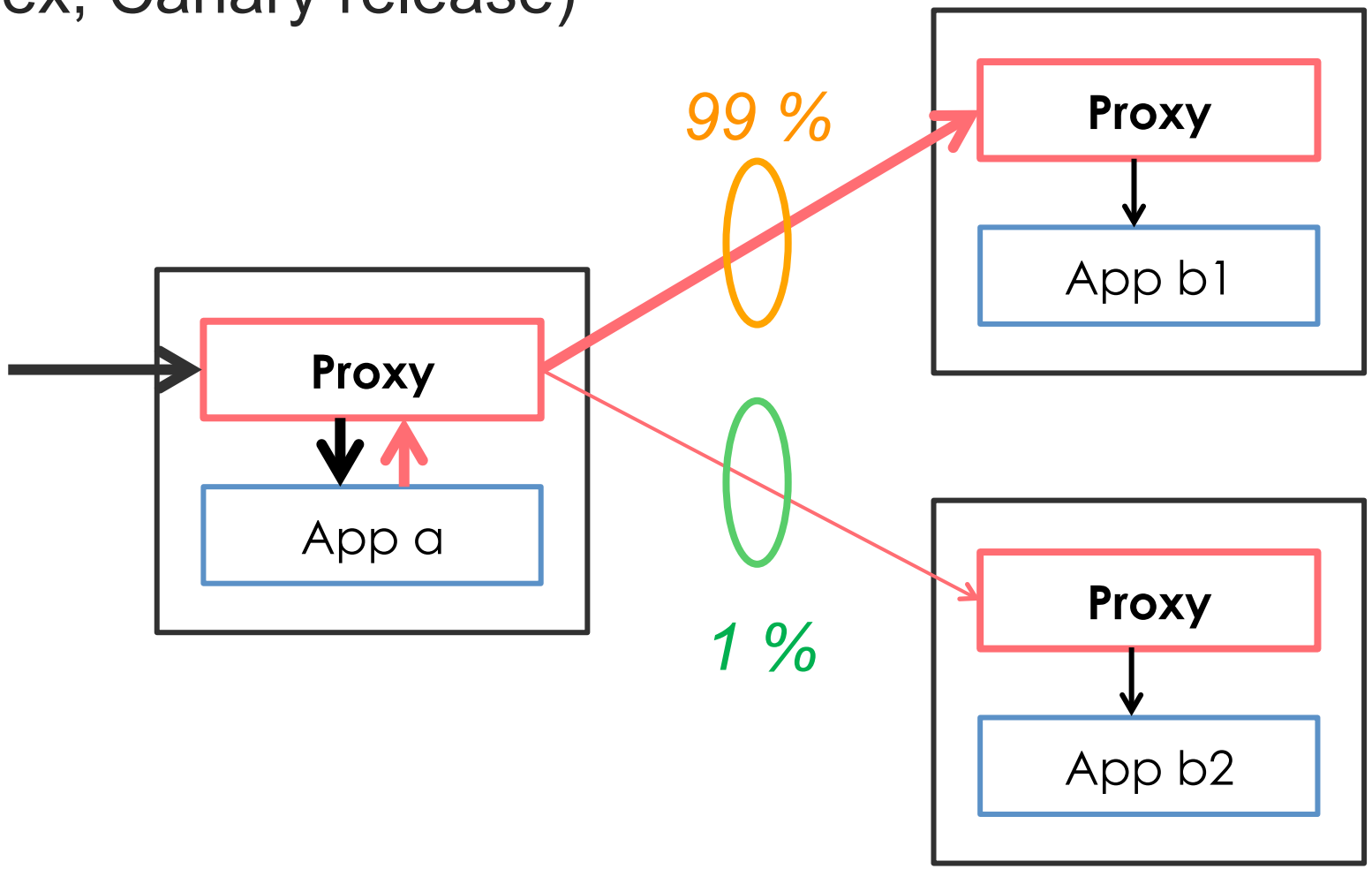
Service Mesh にできること

- Microservice間通信のモニタリング（メトリクス収集）



Service Mesh にできること

- Traffic Shifting (ex, Canary release)

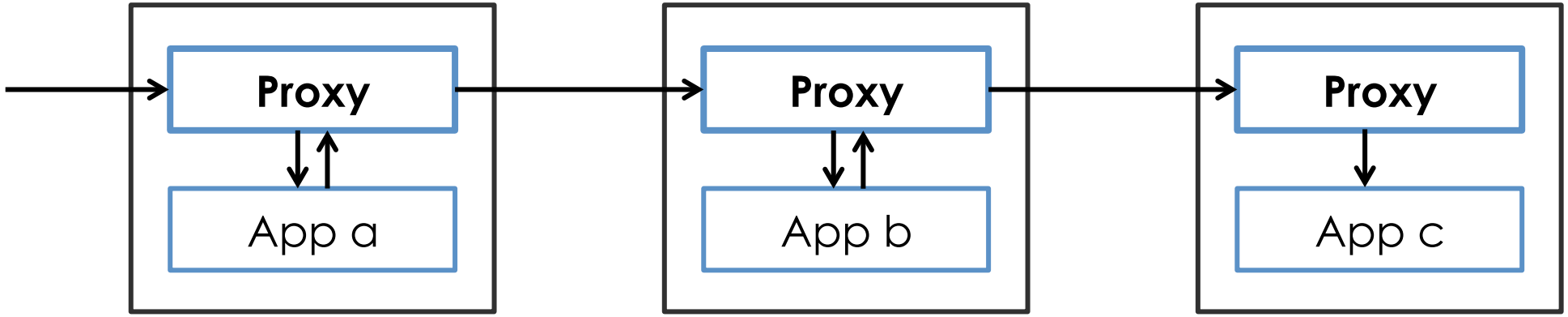


Service Mesh にできること

- Circuit Break
- Fault Injection
- Rate Limit
- Retry
- mTLS

従来アプリケーション側で制御していた機能を分離

開発者はアプリケーションのロジックにだけ集中することが可能



Microservice / Service Mesh まとめ

- 疎結合なシステム
- 復元力がある
- 管理しやすい
- 可観測である
- 堅牢な自動化

Microserviceにより大規模開発時の効率向上

影響の局所化が可能（スケール、デプロイ、障害）

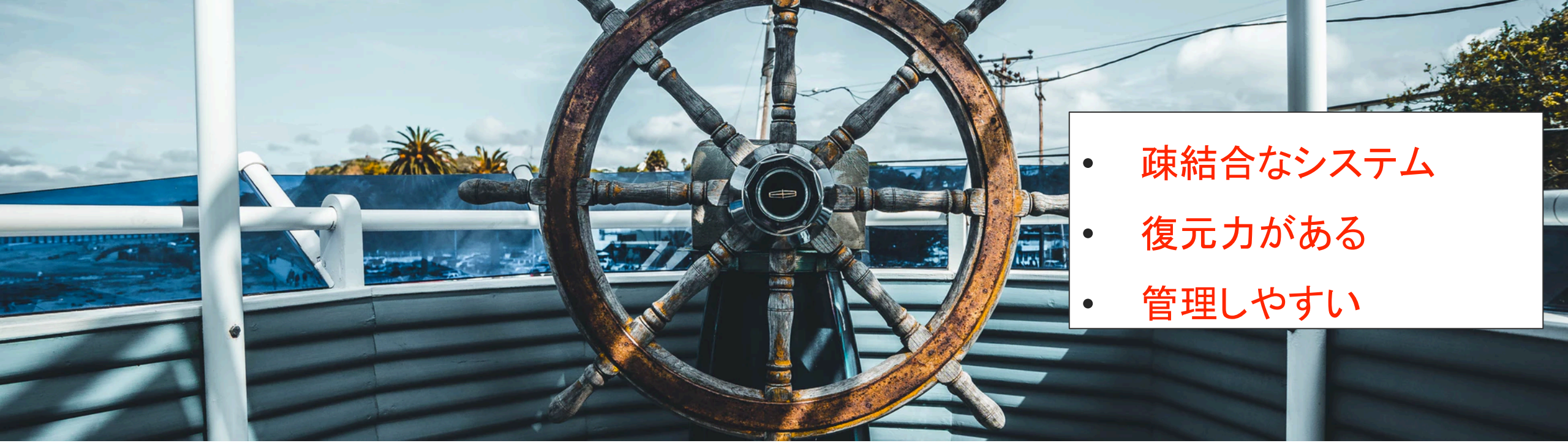
Service Mesh を利用することで…

- Microservice 間の 可観測性を確保

  柔軟な通信の制御を実現

  暗号化を実現

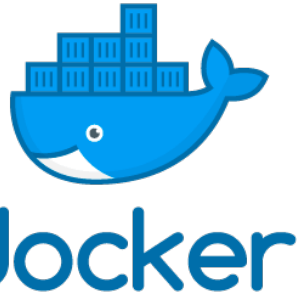
Micro Service 良さそう。
（小さめの）VMでもこれはできるけど…。



- 疎結合なシステム
- 復元力がある
- 管理しやすい

Container / Docker

Benefit of Container / Docker



Linux Containers

Container

System Container

- KVMやXenの代替のように利用する
- 親プロセスが init

Application Container

- アプリケーションプロセスのみを起動する
- 親プロセスはアプリケーション
- 環境とプロセスをパッケージング

Docker

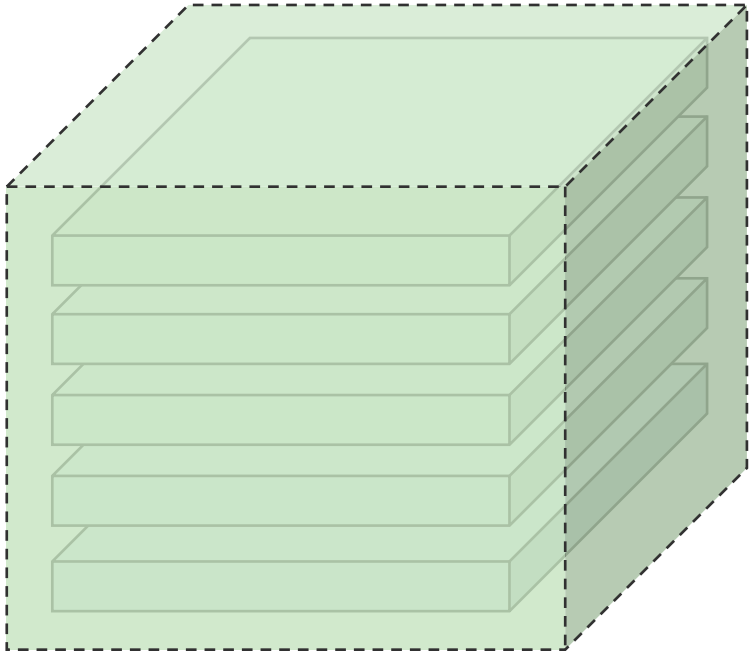
- Docker社が作ったApplication Containerの実行環境やその周辺ツール
- コンテナがDevOps/インフラ界隈で注目されるに至った功労者
- 現在はMobyプロジェクトとして、内部のコアランタイムはcontainerdとして切り出された

Dockerfile と Docker Image

Dockerfile

```
FROM centos:7

RUN yum -y install epel-release

RUN yum -y install nginx

COPY nginx.conf /etc/nginx/

ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

Docker Image



イメージのビルド

Build Once, Run Anywhere

（参考） Part of Docker Container Best Practice

- コンテナで起動するプロセスは1つ
- 一時的なコンテナを作成する（Immutable Infrastructure）
- 開発環境 / 本番環境で同一のイメージを利用する
- 不必要なパッケージは入れない
- 汎用化できる設定値は環境変数などで後から渡す
- レイヤはなるべく減らす
- etc

Ref: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
Ref: https://12factor.net/

Container / Docker まとめ

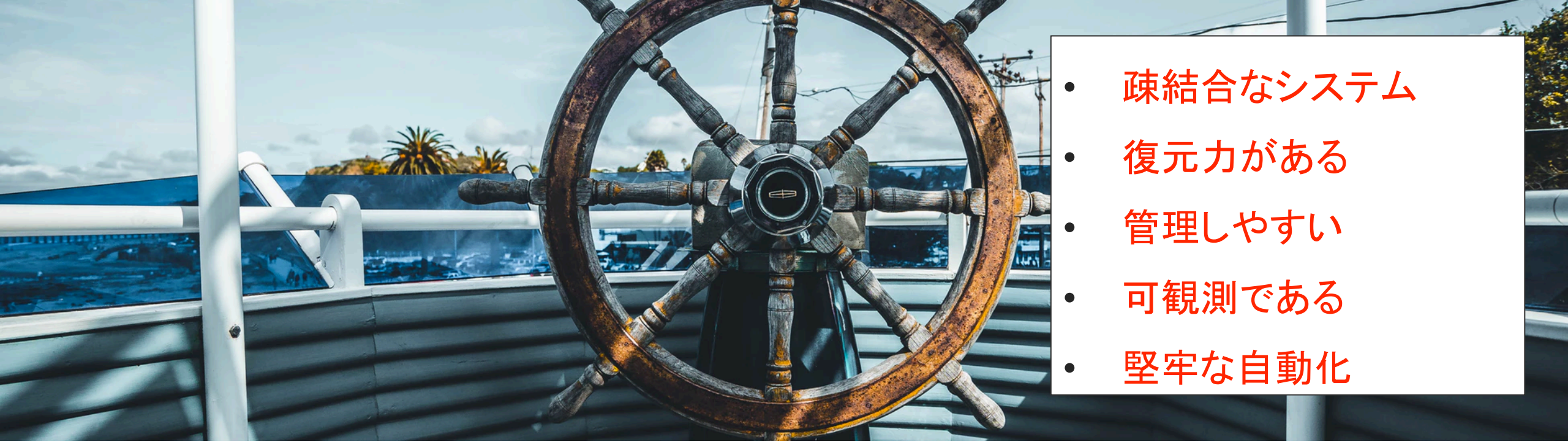
VMと比べて…

- イメージ化が容易
- 起動が高速
- オーバーヘッドが少ない

- 機能ごとに分離（1コンテナ＝1プロセス）
- Immutable Infrastructure を実現しやすい

- 疎結合なシステム
- 復元力がある
- 管理しやすい

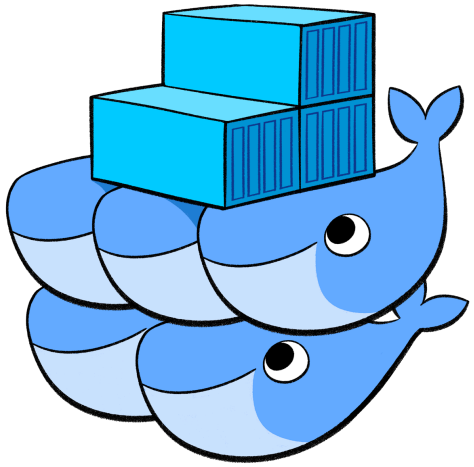
複数台のコンテナを管理する？
どうやって自動化する？



- 疎結合なシステム
- 復元力がある
- 管理しやすい
- 可観測である
- 堅牢な自動化

Kubernetes Overview

Promote Cloud Native



Docker Swarm mode

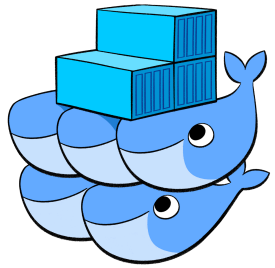
kubernetes

MESOS

Container Orchestration Engine

Container Orchestration Engine が必要な理由

- 複数のDockerホストの管理
- コンテナのスケジューリング
- スケーリング / オートスケーリング
- ローリングアップデート
- コンテナの死活監視
- 障害時のセルフヒーリング
- サービスディスカバリ
- ロードバランシング
- データや機密情報の管理

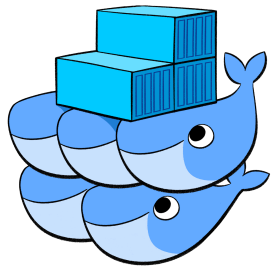


kubernetes

MESOS

Container Orchestration Engine が必要な理由

- 疎結合なシステム
- 復元力がある
- 管理しやすい
- 可観測である
- 堅牢な自動化により、頻繁かつ期待通りに最小限の労力で大きな変更が可能



kubernetes

MESOS

CNCFとStandardization

01. OCI v1.0

コンテナランタイム
コンテナイメージフォーマット
コンテナイメージの配布 (策定中)

02. CRI

オーケストレーションエンジンと
コンテナランタイム間の
インターフェース

03. CSI

コンテナとストレージ間の
インターフェース

04. CNI

コンテナとネットワーク間の
インターフェース

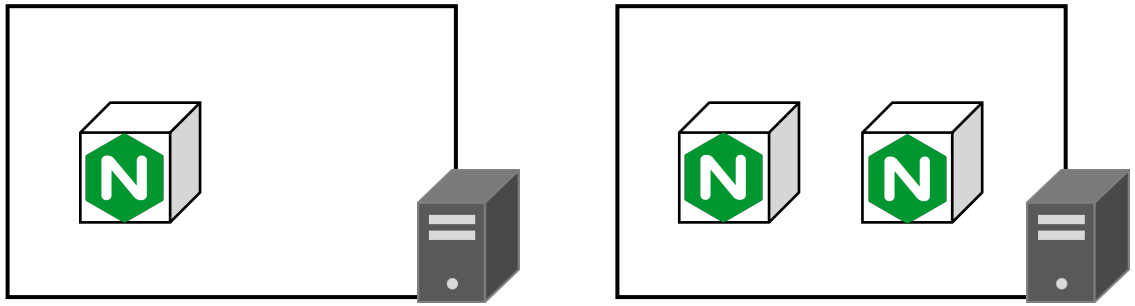
What is kubernetes

- 『コンテナ/アプリケーション実行基盤』としての Kubernetes
- 『X as a Service 基盤』としての Kubernetes
- 『分散システムフレームワーク』としての Kubernetes

『コンテナ/アプリケーション実行基盤』としての Kubernetes

Google Borg をベースとした洗練された基盤

- セルフヒーリング
- ローリングアップデート
- オートスケール
- ロードバランサとの自動連携（抽象化）
- 様々なワークロードへの対応
- データの管理



```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: sample-rs
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      containers:
        - name: nginx-container
          image: nginx:1.12
```

『X as a Service 基盤』としての Kubernetes

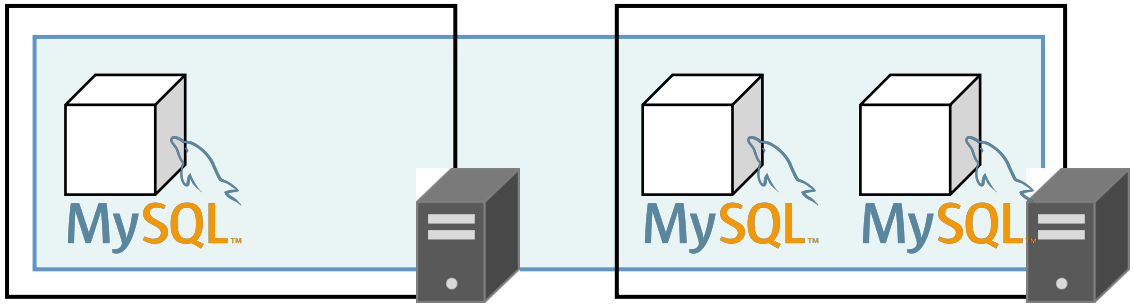
Platform for Platform

- **Database as a Service** on Kubernetes
- **Queue as a Service** on Kubernetes
- **Serverless as a Service** on Kubernetes
- **ML** as a Service on Kubernetes

『障害時のクラスタ復旧』
『バックアップなどの運用』
を自動化（マネージド）

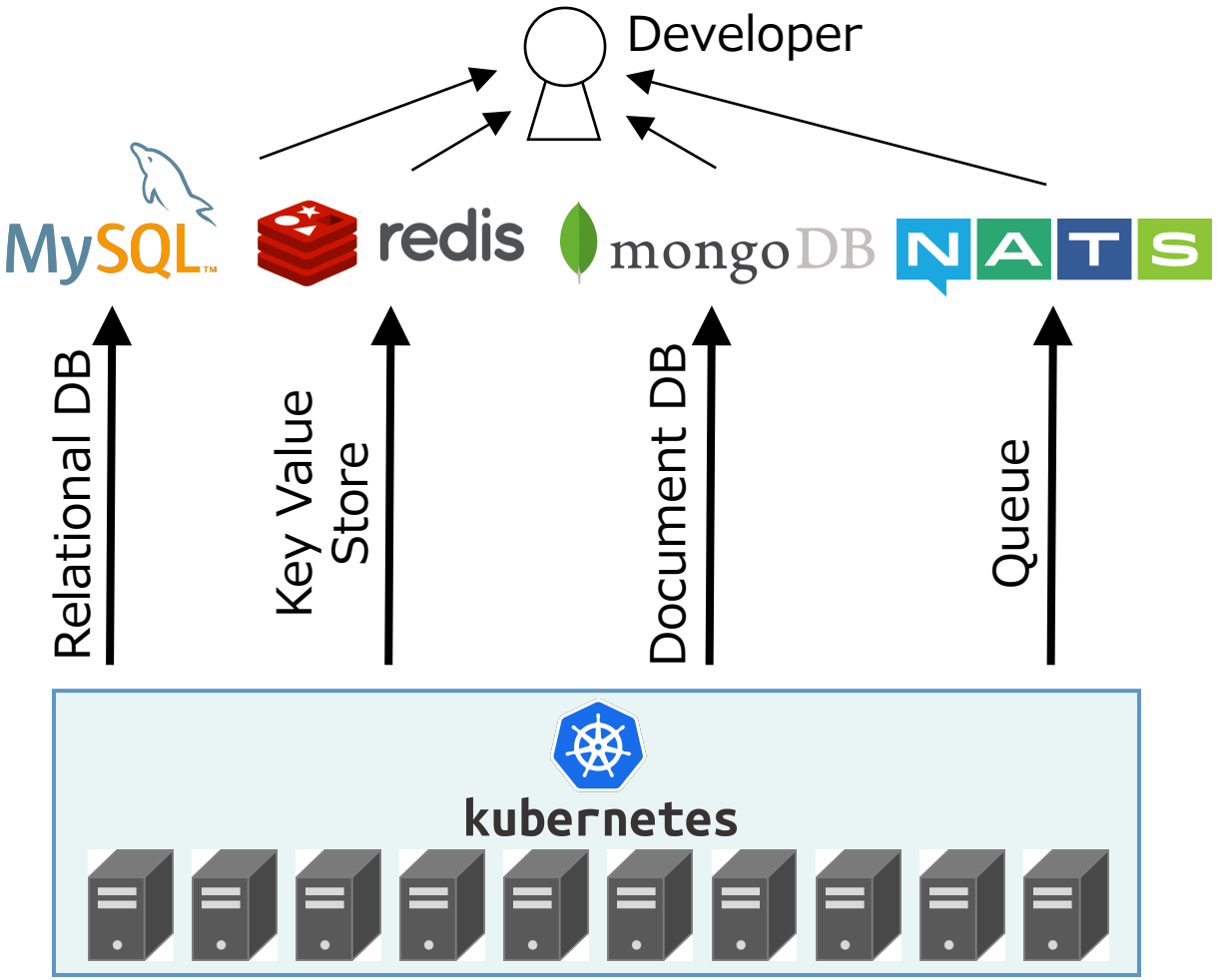
oracle/mysql-operator

```yaml
apiVersion: mysql.oracle.com/v1alpha1
kind: Cluster
metadata:
  name: mysql
spec:
  members: 3
  multiMaster: true
  rootPasswordSecret:
    name: mysql-root-user-secret
```



『X as a Service 基盤』としての Kubernetes

Kubernetes は **小さなクラウド** に近い



『障害時のクラスタ復旧』
『バックアップなどの運用』
を自動化（マネージド）

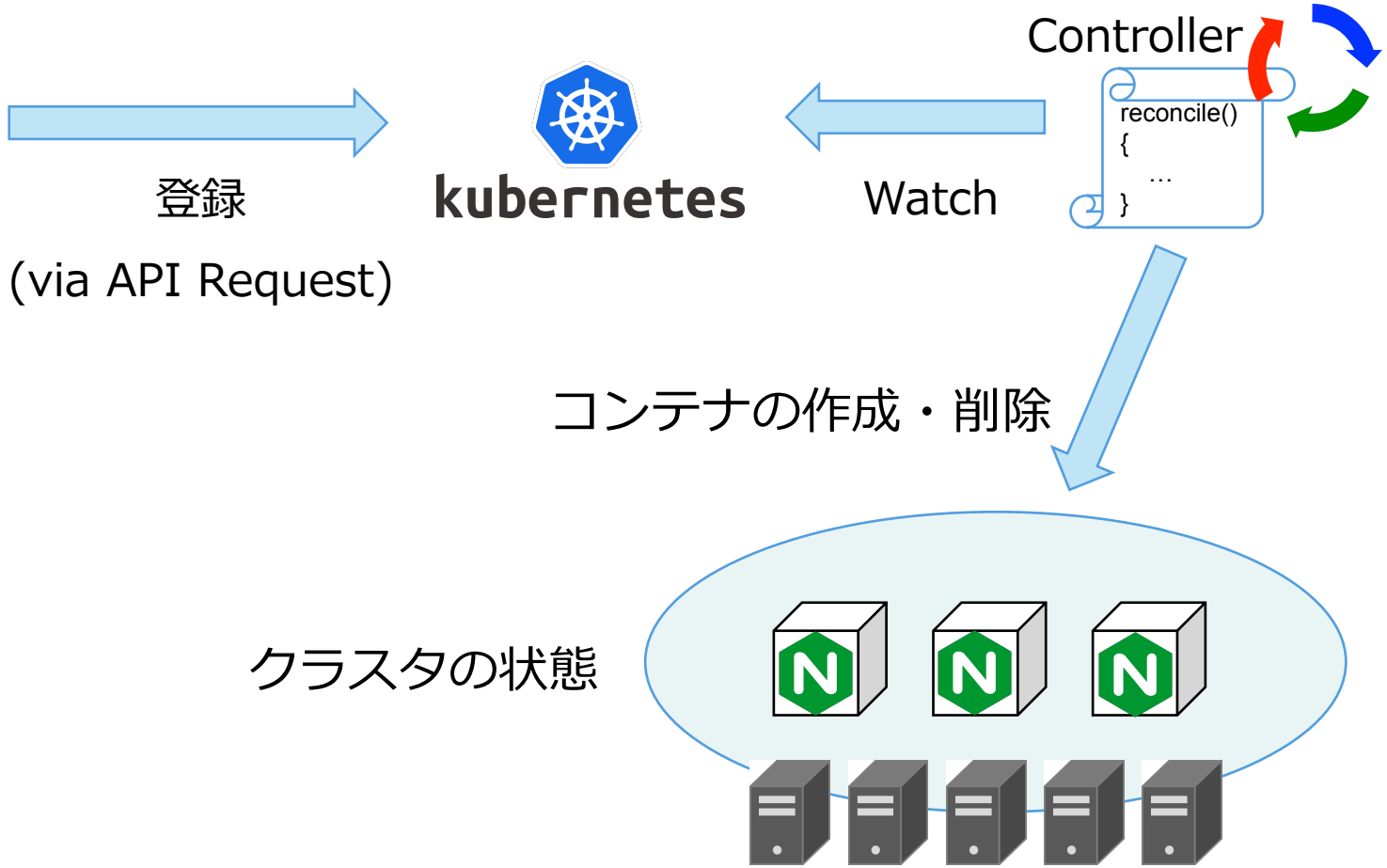
oracle/mysql-operator

```yaml
apiVersion: mysql.oracle.com/v1alpha1
kind: Cluster
metadata:
  name: mysql
spec:
  members: 3
  multiMaster: true
  rootPasswordSecret:
    name: mysql-root-user-secret
```

『分散システムフレームワーク』としての Kubernetes

Kubernetes のコアは『**Decralative API** 』と『**分散システムフレームワーク**』

```yaml
kind: ReplicaSet
metadata:
  name: sample-rs
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: nginx-container
          image: nginx:1.12
```



※ 厳密には Controller も API を用いて変更します。

『分散システムフレームワーク』としての Kubernetes

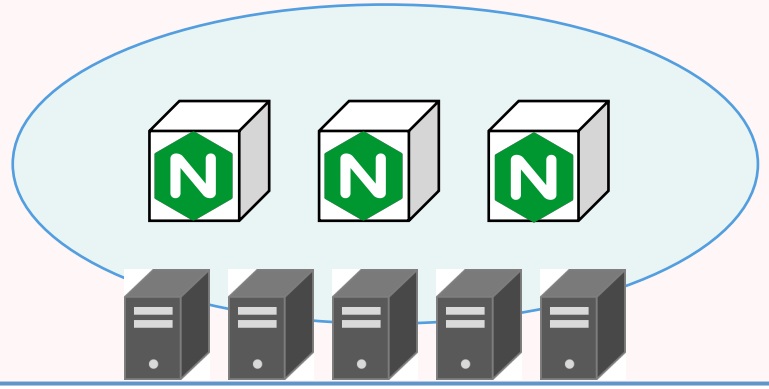
Reconcile Loop （調整 Loop）により、あるべき状態に収束

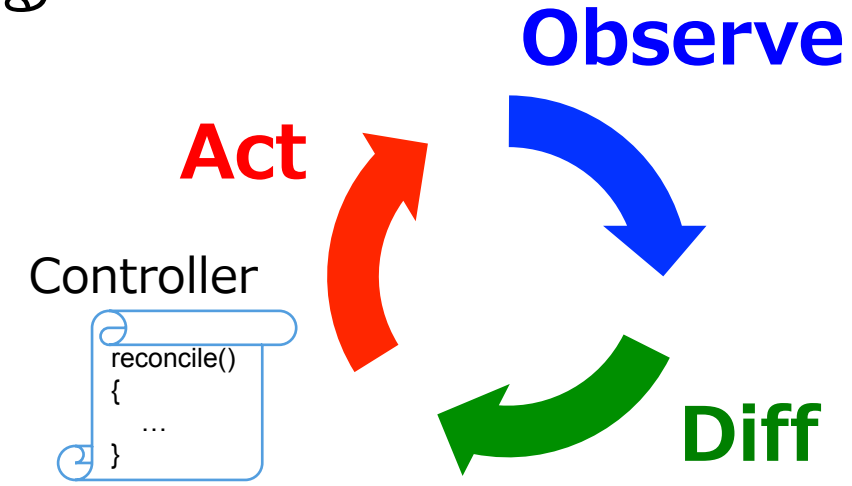
= ただ起動するわけではなく、その状態を維持する

理想の状態

```yaml
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: nginx-container
          image: nginx:1.12
```

クラスタの状態





Observe: 現状を確認

Diff: 理想と現実の差分を計算

Act: 差分に対する処理を実施

『分散システムフレームワーク』としての Kubernetes

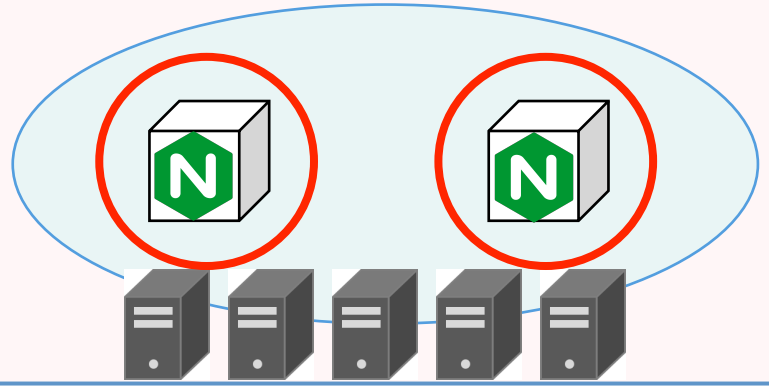
たとえば 2 つしかコンテナ（Pod）が起動していない場合…

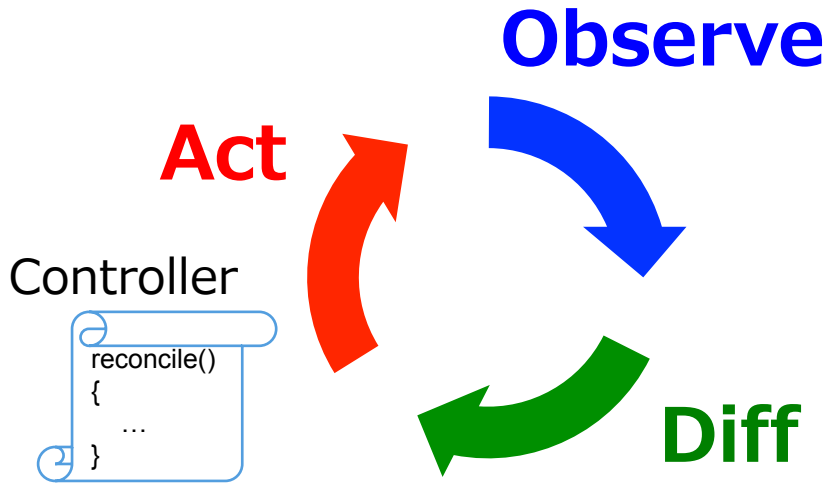
Observe: 理想=3、現状=2

理想の状態

```
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: nginx-container
          image: nginx:1.12
```

クラスタの状態





Observe: 現状を確認

Diff: 理想と現実の差分を計算

Act: 差分に対する処理を実施

『分散システムフレームワーク』としての Kubernetes

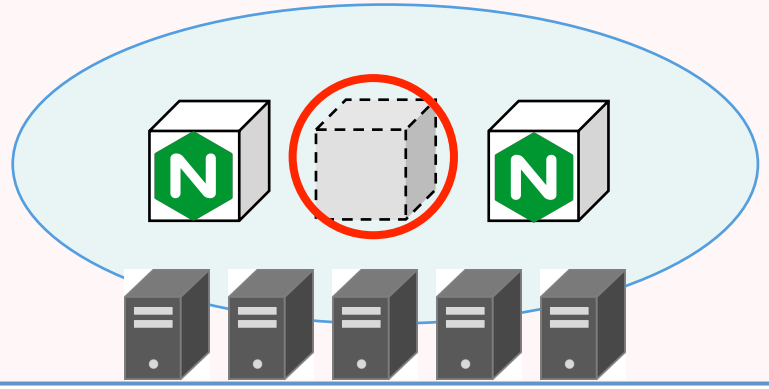
たとえば 2 つしかコンテナ（Pod）が起動していない場合…

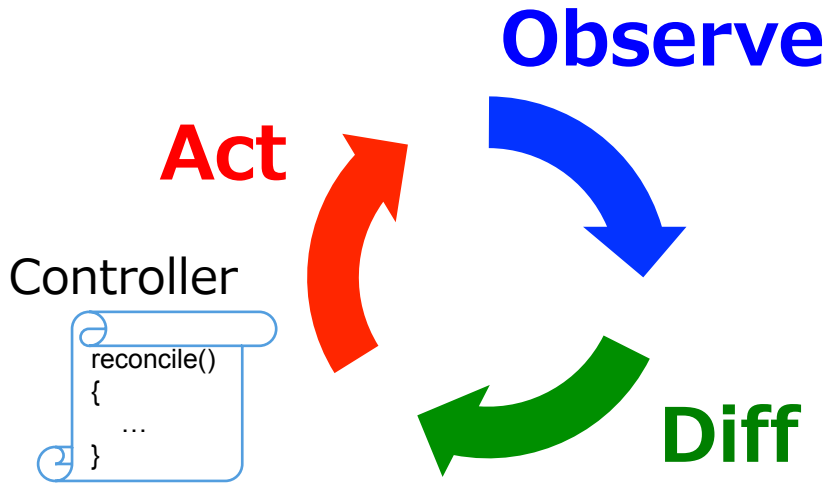
Diff: 1 つコンテナ（Pod）が足りない

理想の状態

```yaml
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: nginx-container
          image: nginx:1.12
```

クラスタの状態





Observe: 現状を確認

Diff: 理想と現実の差分を計算

Act: 差分に対する処理を実施

『分散システムフレームワーク』としての Kubernetes

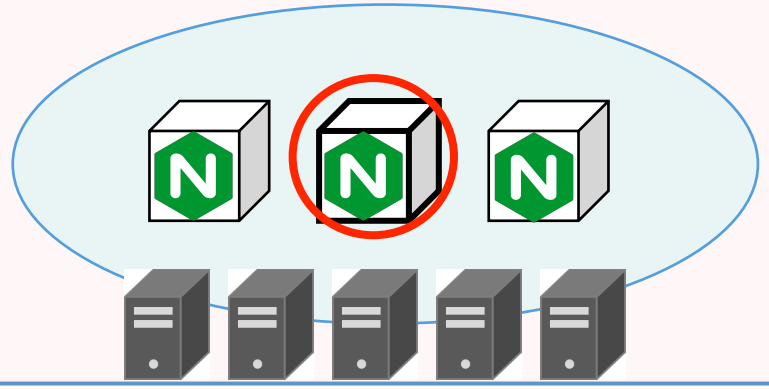
たとえば 2 つしかコンテナ（Pod）が起動していない場合…

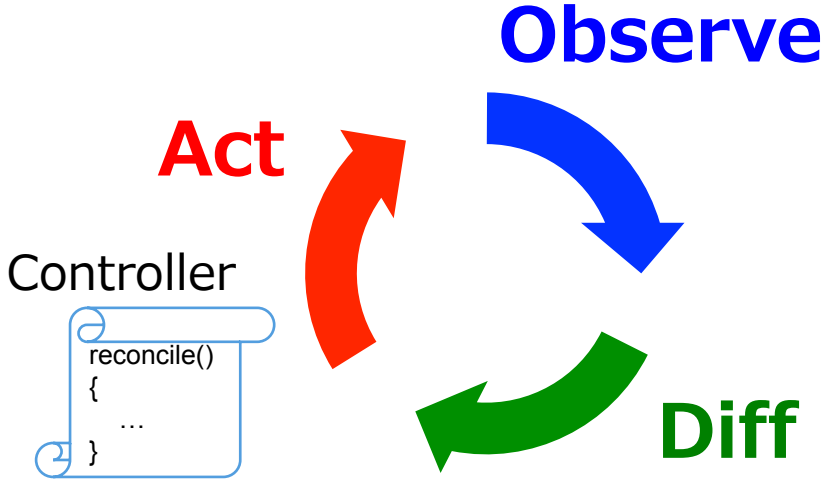
Act: 1つ nginx:1.12 のコンテナ（Pod）を作成する

理想の状態

```yaml
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: nginx-container
          image: nginx:1.12
```

クラスタの状態





Observe: 現状を確認

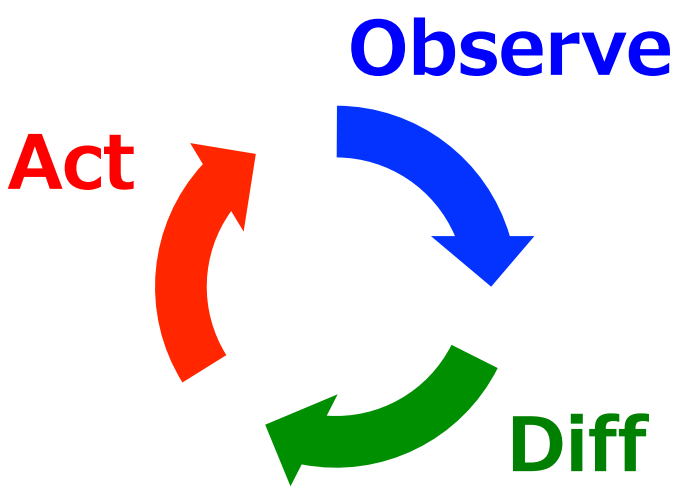
Diff: 理想と現実の差分を計算

Act: 差分に対する処理を実施

『分散システムフレームワーク』としての Kubernetes

Reconcile Loop （調整 Loop）により、あるべき状態に収束

= ただ起動するわけではなく、その状態を維持する



つまり、**<u>運用ナレッジをプログラム化することで自動化</u>**

このプログラムが Controller（Operator）

ReplicaSet Controller は必要なレプリカ数が

- 足りなかったら新しく立ち上げ
- 超過していたら停止させる

Observe: 現状を確認

Diff: 理想と現実の差分を計算

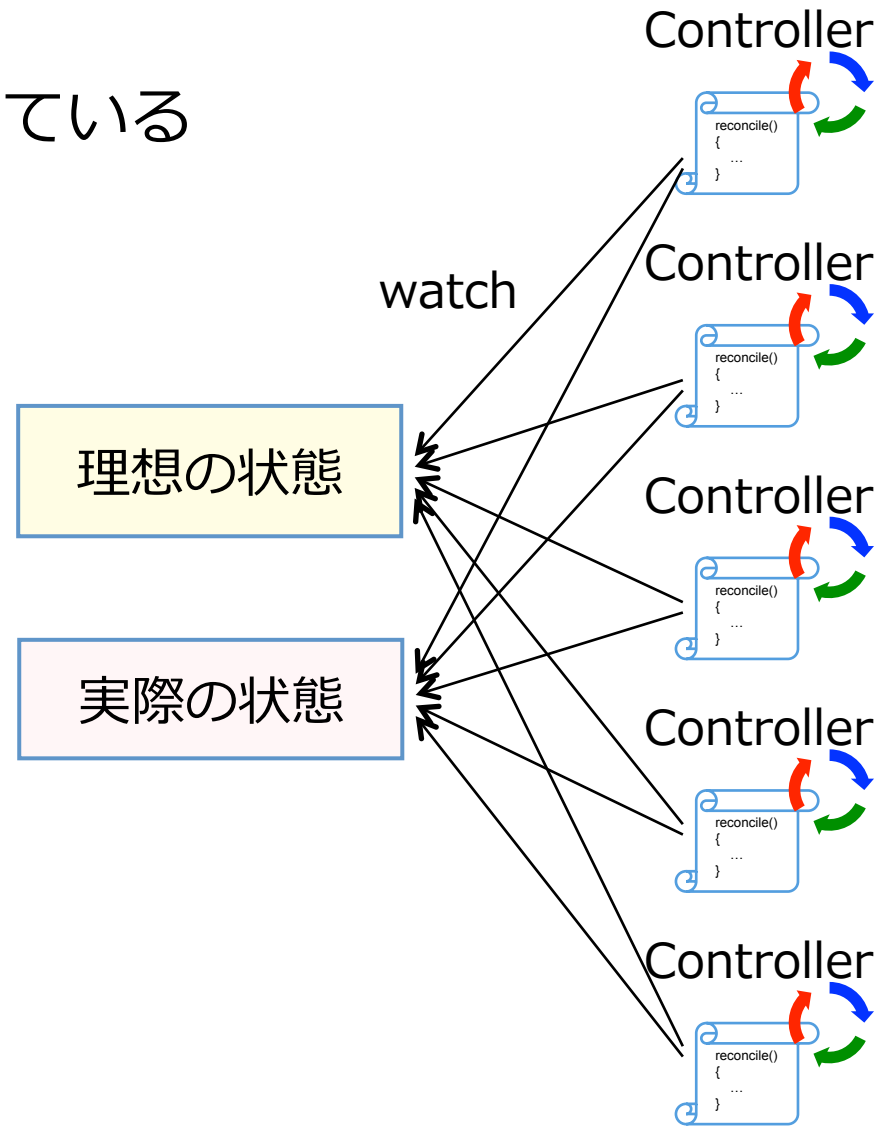
Act: 差分に対する処理を実施

『分散システムフレームワーク』としての Kubernetes

Kubernetes では非常に沢山の Controller が動いている

- Deployment Controller
- ReplicaSet Controller
- Endpoints Controller
- Cloud Controller
- etc.

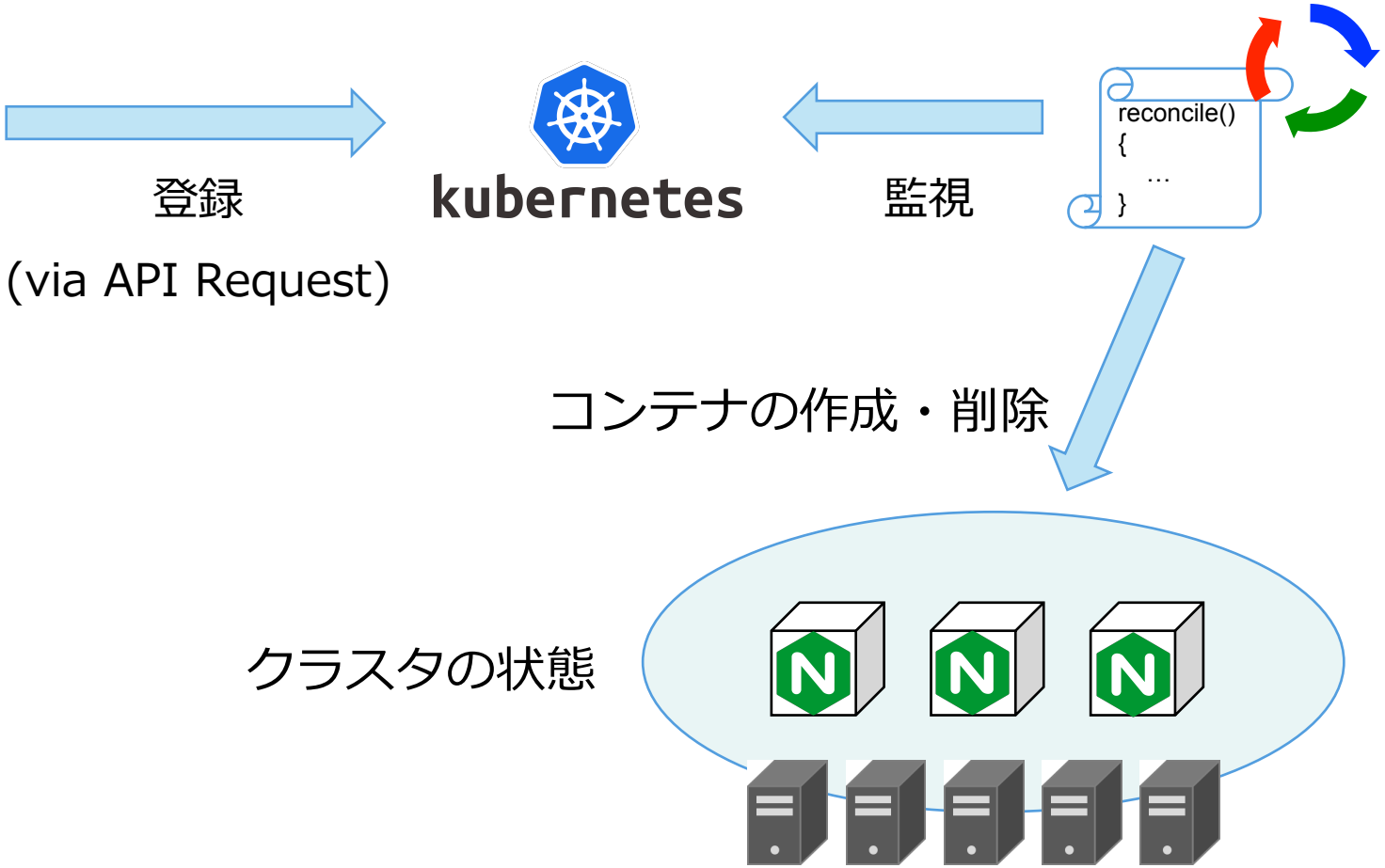
これらの Controller が非同期に動作することで
一つの分散システムとして成り立っている



（再掲） 『分散システムフレームワーク』としての Kubernetes

Kubernetes のコアは『**Decralative API** 』と『**分散システムフレームワーク**』

```yaml
kind: ReplicaSet
metadata:
  name: sample-rs
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: nginx-container
          image: nginx:1.12
```

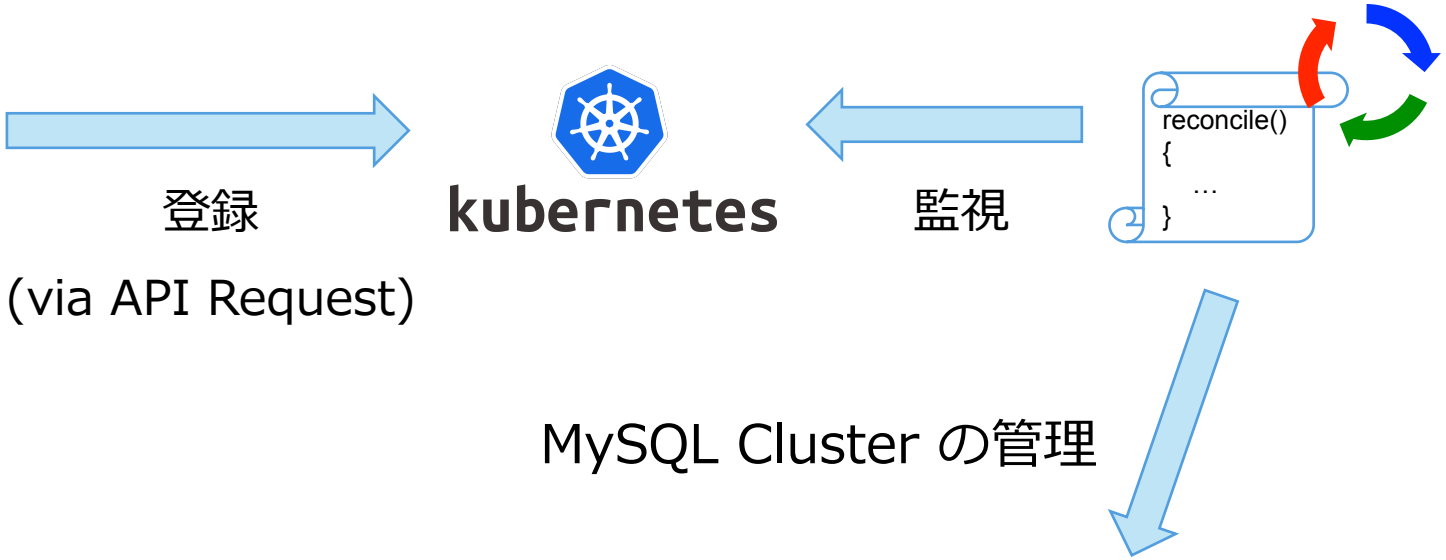


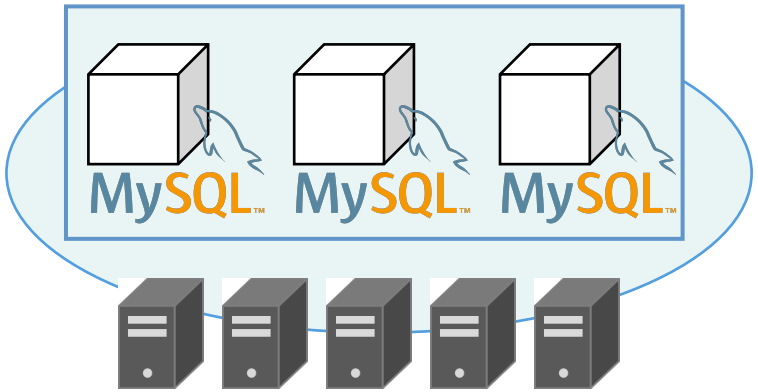
※ 厳密には Controller も API を用いて変更します。

『分散システムフレームワーク』としての Kubernetes

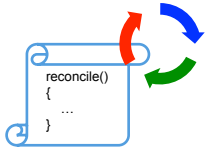
Kubernetes のコアは『**Decralative API** 』と『**分散システムフレームワーク**』

```yaml
apiVersion: mysql.oracle.com/v1alpha1
kind: Cluster
metadata:
  name: mysql
spec:
  members: 3
  multiMaster: true
  rootPasswordSecret:
    name: mysql-root-user-secret
```





Controller



独自のリソースに対してどのような処理をするか

Controller を書くことで運用が容易に

（運用ナレッジのプログラム化）

※ 厳密には Controller も API を用いて変更します。

MySQL operator

```yaml
apiVersion: mysql.oracle.com/v1alpha1
kind: Cluster
metadata:
  name: mysql
spec:
  members: 3
  multiMaster: true
  rootPasswordSecret:
    name: mysql-root-user-secret
```

Register

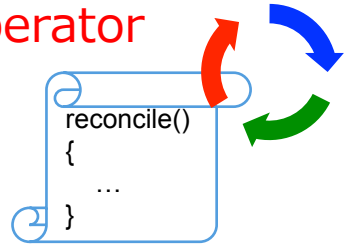
(via API Request)

kubernetes

Watch

MySQL

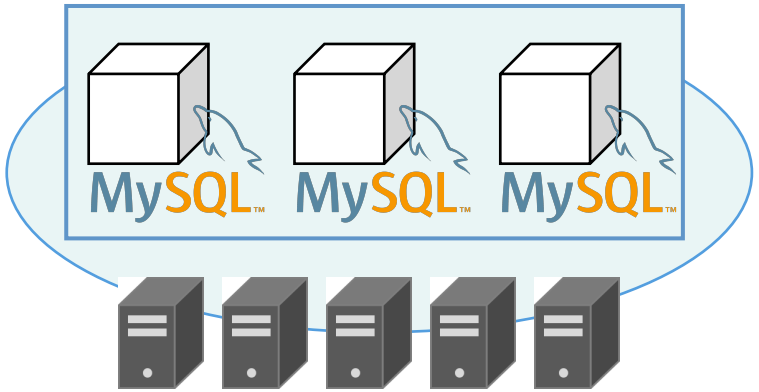
Operator



Manage StatefulSet

and so on

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  replicas: 3
  template:
    spec:
      containers:
      - command:
        - /bin/bash
        - -ecx
        - |2-

                # Set baseServerID
                base=1000

                # Finds the replica index from the hostname, and uses this to define
                # a unique server id for this instance.
                index=$(cat /etc/hostname | grep -o '[^-]*$')
                /entrypoint.sh --server_id=$(expr $base + $index) --datadir=/var/lib/mysql --user=mysql --gtid_mode=ON --log-bin
                  --binlog_checksum=NONE --enforce_gtid_consistency=ON --log-slave-updates=ON --binlog-format=ROW
                  --master-info-repository=TABLE --relay-log-info-repository=TABLE --transaction-write-set-extraction=XXHA5H64
                  --relay-log=mysql-${index}-relay-bin --report-host="mysql-${index}.mysql" --log-error-verbosity=3
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              key: password
              name: mysql-root-password
        - name: MYSQL_ROOT_HOST
          value: '%'
        - name: MYSQL_LOG_CONSOLE
          value: "true"
        image: mysql/mysql-server:8.0.12
        name: mysql
        ports:
        - containerPort: 3306
          protocol: TCP
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - mountPath: /var/lib/mysql
          name: mysqlvolume
          subPath: mysql
        - mountPath: /var/lib/mysql-agent
          name: mysqlbackupvolume
          subPath: mysql
      - args:
        - --v=4
        env:
        - name: MYSQL_CLUSTER_NAME
          value: mysql
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: metadata.namespace
        - name: REPLICATION_GROUP_SEEDS
          value: mysql-0.mysql:13306,mysql-1.mysql:13306,mysql-2.mysql:13306
        - name: MYSQL_CLUSTER_MULTI_MASTER
          value: "true"
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              key: password
              name: mysql-root-password
        - name: MY_POD_IP
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: status.podIP
        image: iad.ocir.io/oracle/mysql-agent:0.3.0
        imagePullPolicy: IfNotPresent
        livenessProbe:
          failureThreshold: 3
          httpGet:
            path: /live
            port: 10512
            scheme: HTTP
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        name: mysql-agent
        readinessProbe:
          failureThreshold: 3
          httpGet:
            path: /ready
            port: 10512
            scheme: HTTP
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - mountPath: /var/lib/mysql
          name: mysqlvolume
          subPath: mysql
        - mountPath: /var/lib/mysql-agent
          name: mysqlbackupvolume
          subPath: mysql
      serviceAccount: mysql-agent
      serviceAccountName: mysql-agent
      volumes:
      - emptyDir: {}
        name: mysqlvolume
      - emptyDir: {}
        name: mysqlbackupvolume
```



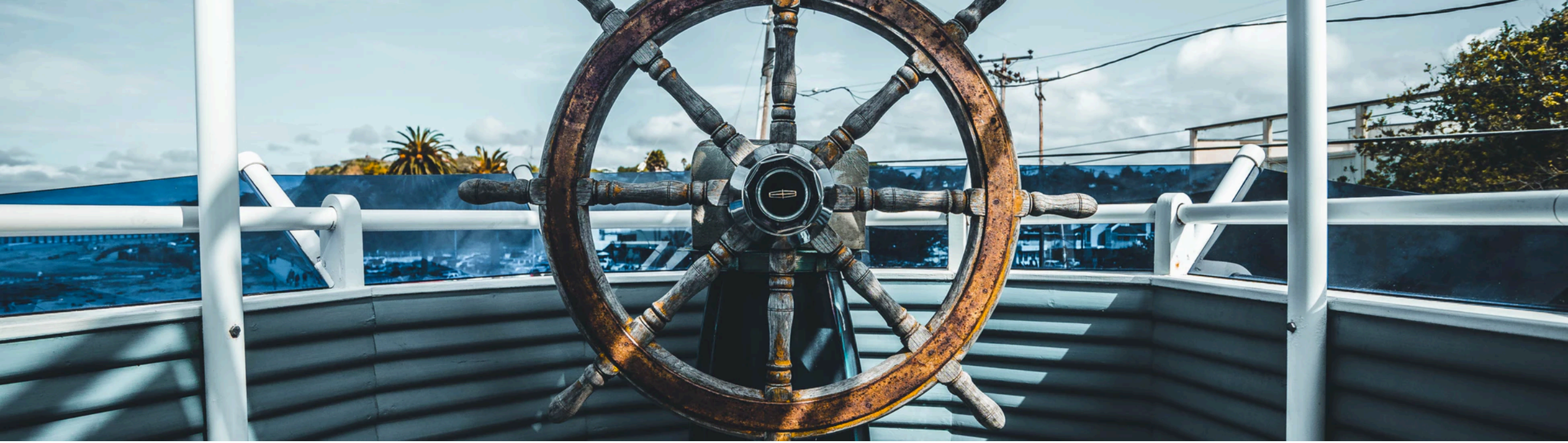
Key points

- KubernetesはGoogleが社内で長年使っていた
  コンテナクラスタマネージャBorgをベースにOSS化したもの
  → **様々なワークロードへの対応、洗練化された自動化、拡張性**

- 抽象化されており、ネットワークの操作などもYAMLを書くだけ
  → **Infrastructure Engineer（Ops）だけが使うものではなく**
  Server-side Engineer など（Dev）も使うもの

Kubernetes 上で
アプリケーションを安定稼働させることは可能

Kubernetes の運用自体も
Cloud Native にするためには？



Manage Kubernetes Cluster

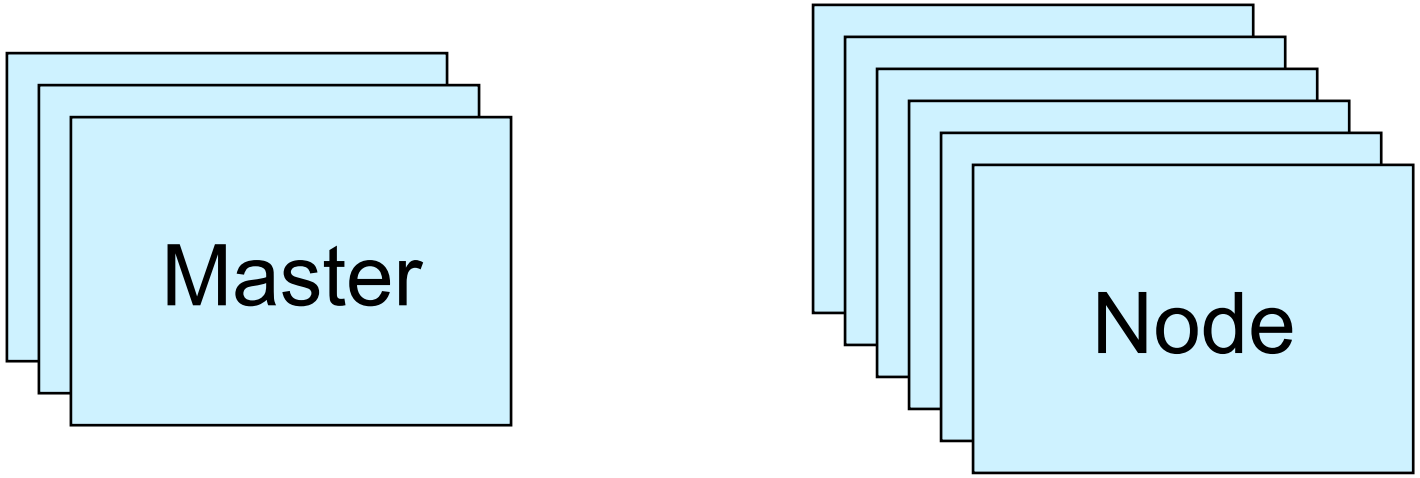
cluster auto-scaling, node-healing, version up

Kubernetes クラスタの構成

Kubernetes Cluster

= Kubernetes **Master** + Kubernetes **Node**

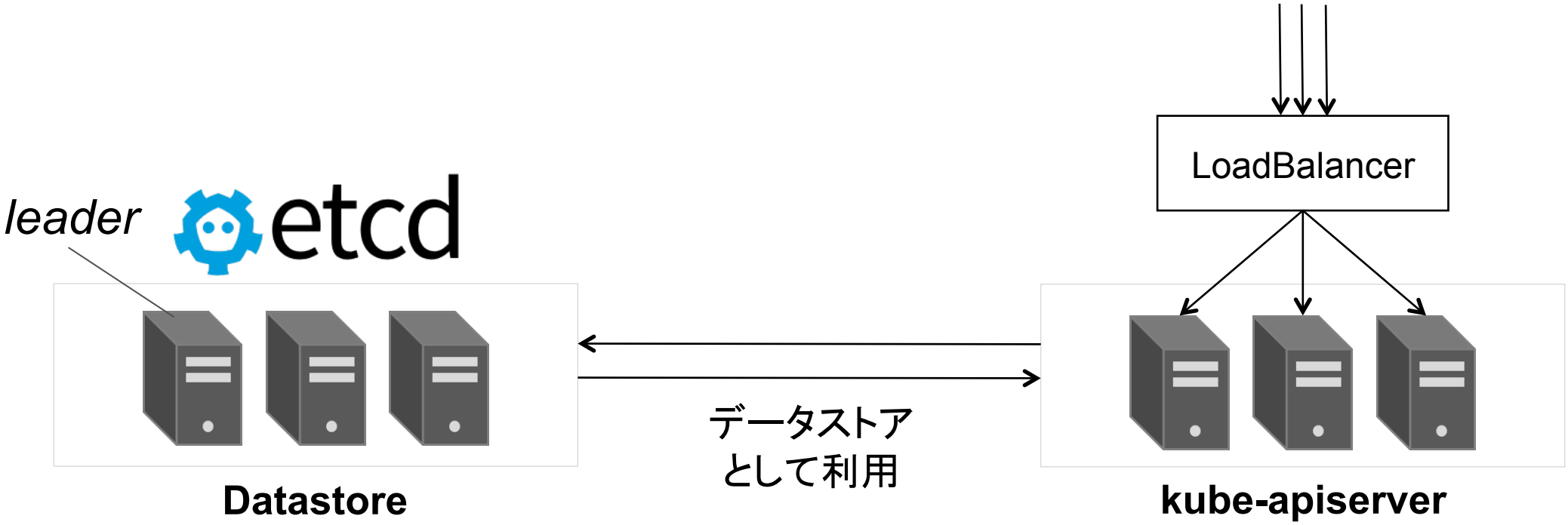
Master/Node 自体も複数のコンポーネントで構成



Kubernetes Master の構成

基本は **分散 Key-Value Store** の etcd と **API Server**

- etcd はマルチノード構成で **Raft** を用いて Leader ノードを選定
- API Server は KVS に対する R/W を担当

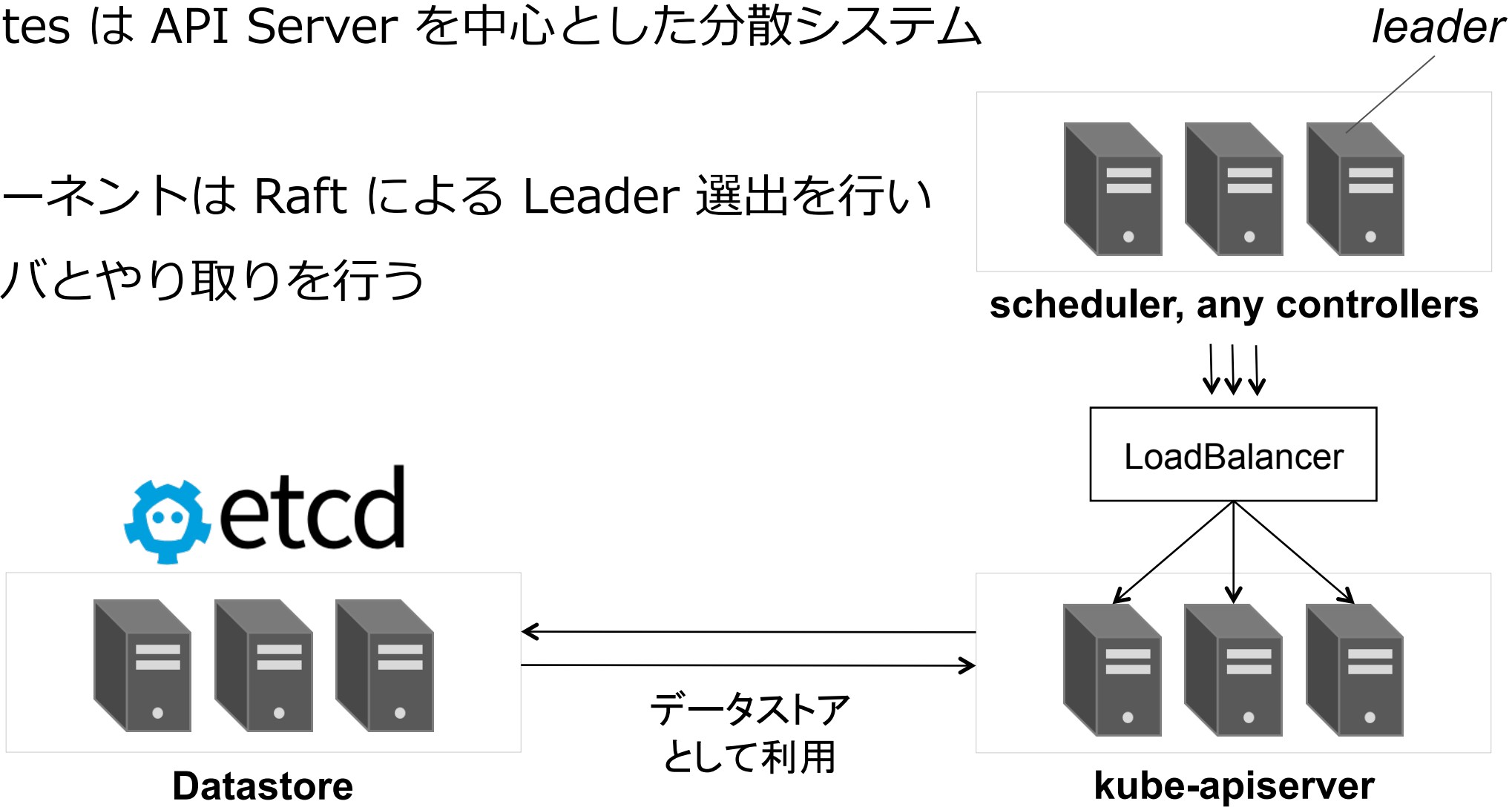


Kubernetes Master の構成

Kubernetes は API Server を中心とした分散システム

各コンポーネントは Raft による Leader 選出を行い

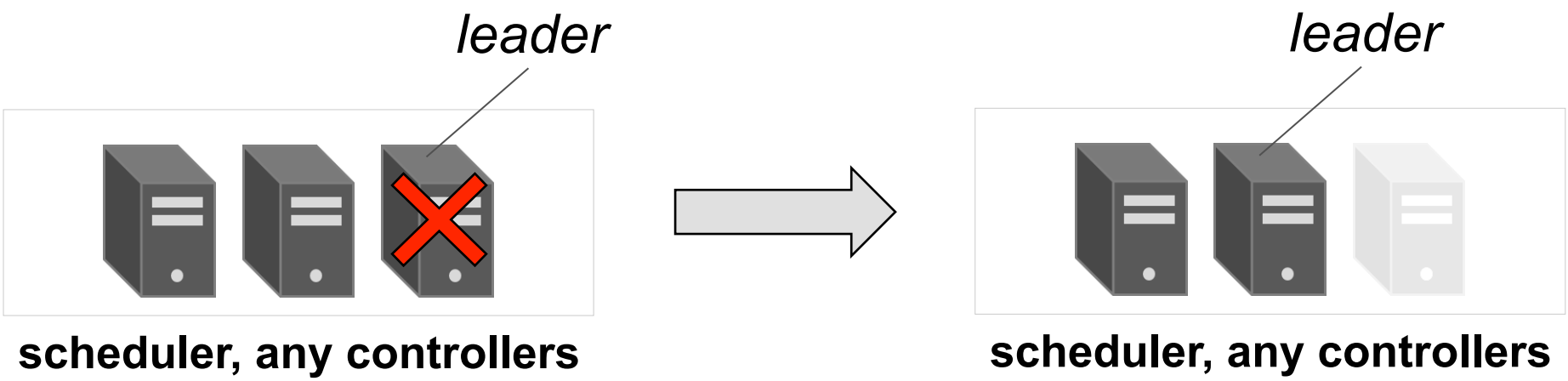
API サーバとやり取りを行う



Kubernetes Master の冗長性

1．Load Balancer を介したリクエスト分散

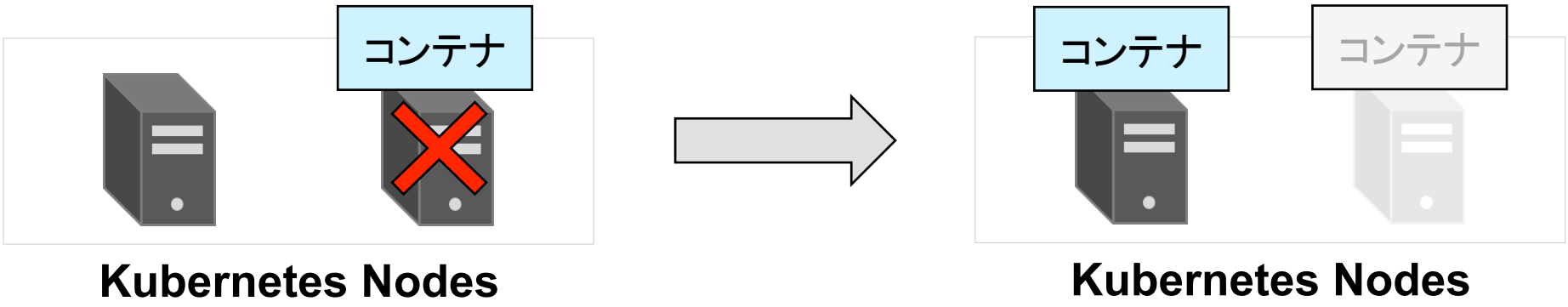
2．Raft による Leader 選出



Kubernetes Node の冗長性

Node にはコンテナランタイムやエージェントが起動

Kubernetes Node の障害にはコンテナのマイグレーションで対応



（参考）コンテナランタイム

最近は より分離性の高い
コンテナ技術が開発
イメージのフォーマットなどは
同じため、再利用可能

✕ Kubernetesは複数 ~~Dockerホスト~~ の管理を行う

○ Kubernetesは複数 コンテナホスト の管理を行う

- コンテナランタイム環境

CRI ランタイム（上位）

- docker
- cri-o
- containerd

×

OCI ランタイム（下位）

- runC
- gVisor
- Kata Containers
- Nabla Containers
- Firecracker

Kubernetes 環境の違いとは？

Certified Kubernetes Distribution / Platform

CNCF が提供する **Conformance Test（適合テスト）**に
通過したものだけが **Certified Kubernetes** として認定

Kubernetes 環境は現状 **123** 個

https://docs.google.com/spreadsheets/d/1LxSqBzjOxfGx3cmtZ4EbB_BGCxT_wlxW_xgHVVa23es/edit#gid=0

Installer：インストールツール（kubeadm、kubeaws、kubespray、etc）

Distribution：Kubernetes 準拠の環境（OpenShift、Docker EE、k3s、Rancher、etc）

Hosted：マネージドサービス（GKE、EKS、AKS、etc）

では Kubernetes を安定稼働させるためには？

Kubernetes クラスタの安定化

ノードオートヒーリング（like Self-healing of ReplicaSet）

Kubernetes Node に問題があった場合にノードの復旧を行う

クラスタのバージョンアップ（like Rolling Update of Deployment）

Kubernetes の Master 及び Node のコンポーネント郡の Rolling Upgrade

クラスタオートスケール（like HorizontalPodAutoscaler）

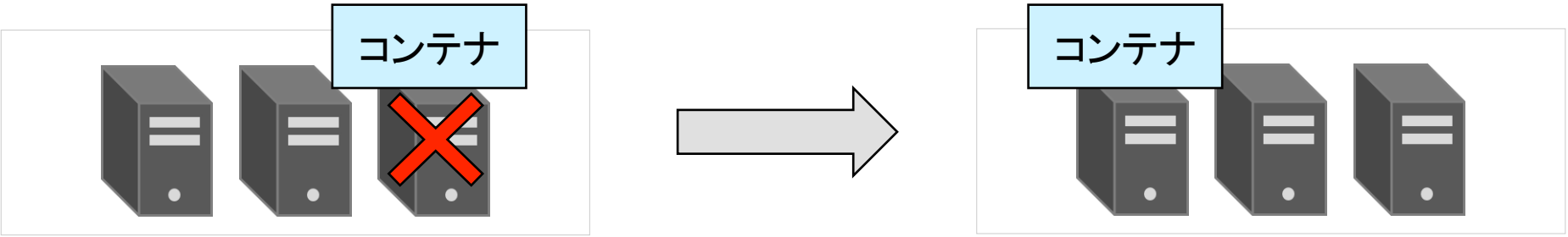
Kubernetes Node を追加/削除してクラスタのスケーリングを行う

ノードオートヒーリング

ノードの Disk/Memory/PID の状態やコンテナランタイムの状態が異常な場合に

コンテナを退避後、ノードを自動復旧させる機能

```
Conditions:
  Type                          Status  LastHeartbeatTime                 LastTransitionTime                Reason                      Message
  ----                          ------  -----------------                 ------------------                ------                      -------
  NetworkUnavailable            False   Mon, 05 Aug 2019 11:46:10 +0900   Mon, 05 Aug 2019 11:46:10 +0900   CalicoIsUp                  Calico is running on this node
  CorruptDockerOverlay2         False   Wed, 04 Sep 2019 09:13:23 +0900   Mon, 05 Aug 2019 11:45:54 +0900   CorruptDockerOverlay2
  FrequentKubeletRestart        False   Wed, 04 Sep 2019 09:13:23 +0900   Mon, 05 Aug 2019 11:45:54 +0900   FrequentKubeletRestart
  FrequentDockerRestart         False   Wed, 04 Sep 2019 09:13:23 +0900   Mon, 05 Aug 2019 11:45:55 +0900   FrequentDockerRestart
  FrequentContainerdRestart     False   Wed, 04 Sep 2019 09:13:23 +0900   Mon, 05 Aug 2019 11:45:56 +0900   FrequentContainerdRestart
  NTPProblem                    False   Wed, 04 Sep 2019 09:13:23 +0900   Mon, 05 Aug 2019 11:45:53 +0900   NTPIsDown                   NTP service is running
  FrequentUnregisterNetDevice   False   Wed, 04 Sep 2019 09:13:23 +0900   Mon, 05 Aug 2019 11:45:54 +0900   UnregisterNetDevice
  MemoryPressure                False   Wed, 04 Sep 2019 09:13:53 +0900   Mon, 05 Aug 2019 11:45:55 +0900   KubeletHasSufficientMemory  kubelet has sufficient memory available
  DiskPressure                  False   Wed, 04 Sep 2019 09:13:53 +0900   Mon, 05 Aug 2019 11:45:55 +0900   KubeletHasNoDiskPressure    kubelet has no disk pressure
  PIDPressure                   False   Wed, 04 Sep 2019 09:13:53 +0900   Mon, 05 Aug 2019 11:45:55 +0900   KubeletHasSufficientPID     kubelet has sufficient PID available
  Ready                         True    Wed, 04 Sep 2019 09:13:53 +0900   Mon, 05 Aug 2019 11:46:26 +0900   KubeletReady                kubelet is posting ready status
```



Kubernetes Nodes **Kubernetes Nodes**

クラスタのバージョンアップ

Kubernetes Master や Node を構成するコンポーネント郡の Rolling Update

Kubernetes Node の Upgrade 時には事前にコンテナの退避処理を行う

Master コンポーネント自体もコンテナ化されているケースも多い

```
$ kubectl get nodes
NAME                                STATUS   ROLES    AGE   VERSION
amsy810-mysql-ake-default-b-namtlxr Ready    node     29d   v1.13.8-ca
amsy810-mysql-ake-default-b-nchekzb Ready    node     29d   v1.13.8-ca
amsy810-mysql-ake-default-b-ngglbwn Ready    node     29d   v1.13.8-ca
amsy810-mysql-ake-default-b-nginlpx Ready    node     29d   v1.13.8-ca
amsy810-mysql-ake-default-b-ngsonvo Ready    node     29d   v1.13.8-ca
```

クラスタオートスケール

Kubernetes Node 自体のオートスケーリング

コンテナのオートスケールだけでは限界があるため、

通常はクラスタ側のオートスケール機能と合わせて利用

複数のインスタンスタイプのノードが入り乱れている場合は

特定のアルゴリズムによって計算される

クラウドプロバイダー特有の連携

Kubernetes はベンダーニュートラル

クラウド依存のコードは全て Pluggable な形式

- Load Balancer 連携
- ストレージ連携
- 認証/認可連携
- ノード管理周りの機能
- etc

その他の Cloud Provider ごとの違い

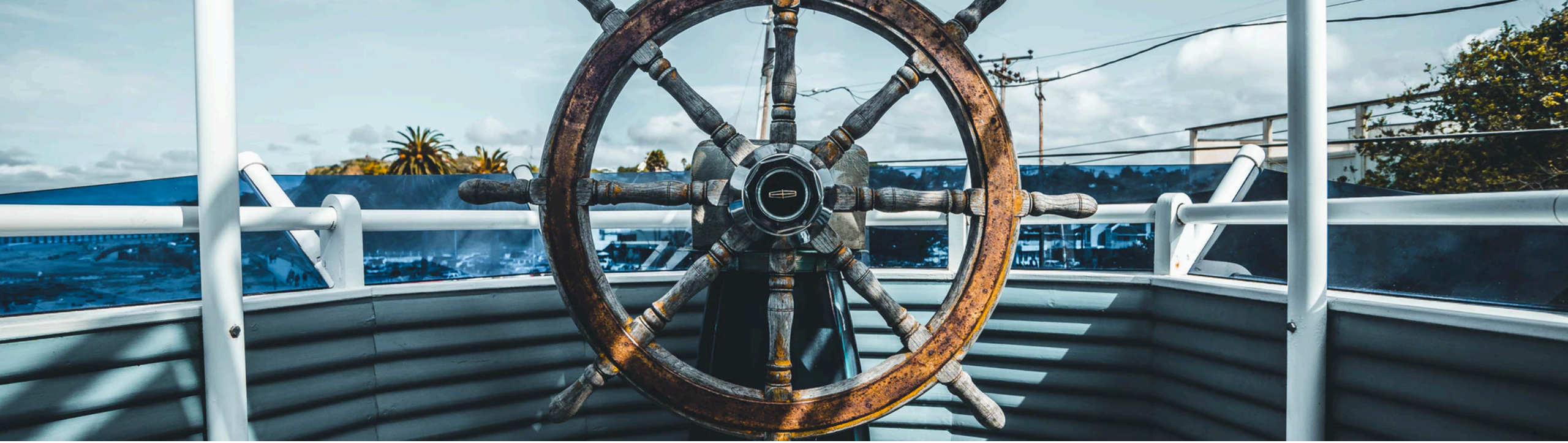
マネージド Kubernetes では

Master コンポーネントのカスタマイズができないことが多い

他にも CloudProvider 特有の Kubernetes 拡張（CustomResource）実装が

多々用意されている事が多い

- ConfigConnecter
- BackendConfig
- etc



Conclusion

Promote Cloud Native with Kubernetes

まとめ

K8s は Google Borg をベースとした Cloud Native を加速させる基盤の 1 つ

- 高い拡張性により様々なワークロードを抽象化して管理することが可能
- また、Operator により運用のナレッジをプログラム化して自動化可能

Kubernetes クラスタ自体の運用も考える必要がある

- Kubernetes 上で稼働させるコンテナアプリケーション自体は Kubernetes の責任範囲で安全に稼働することが保証されるようになっている
  - 例）ReplicaSet、Operator
- Kubernetes 自体もマイクロサービス化されており、容易な冗長構成が組める
- Kubernetes 環境の選定はクラスタの運用性や Cloud Provider 連携による

Thank you for your attention

Let’s enjoy with Kubernetes

follow me: **@amsy810**