これから学ぶKubernetesのReconciliation Loop

脱初心者への道！
Kubernetes Middle Way!! - cndjp第16回

@yosshi_

自己紹介

- 吉村 翔太
- ゼットラボ株式会社 ソフトウェアエンジニア
- 経歴
  SIer 6年、通信事業者のR&D 2年 etc
- Kurbernetes 、Prometheus etc
- 登壇 CNDT “**Kubernetes Logging入門**” etc
- コミュニティ活動 “**Cloud Native Developers JP**”

“**Prometheus Meetup Tokyo**”

@yosshi_

本日のゴール

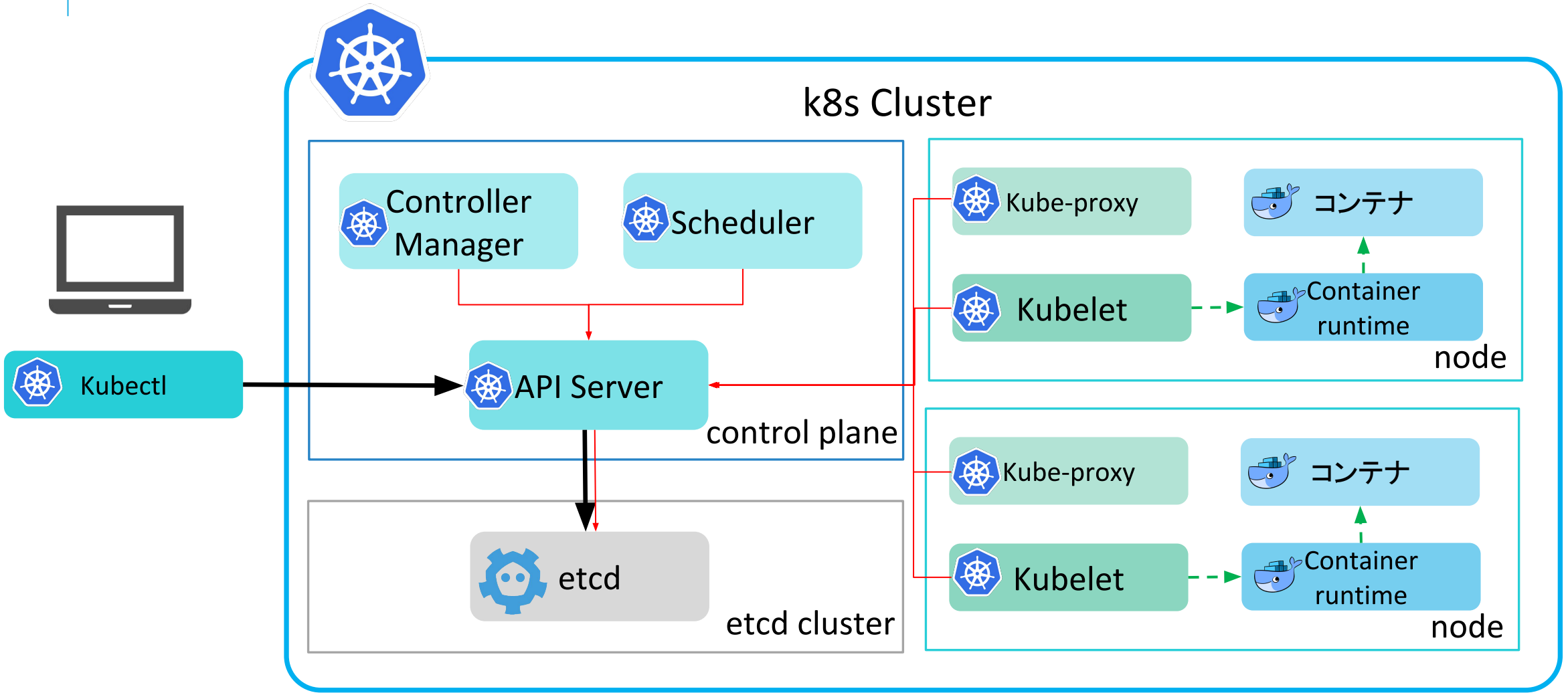
- 発表を聴き終えた聴講者の状態
  - Reconciliation Loop の誕生した背景や考え方が分かる
- 想定する聴講者のスキルレベル
  - 「これからはじめる！Kubernetes基礎」が分かる
  - 「ゼロから始めるKubernetes Controller / Under the Kubernetes Controller」はちょっと難しい

< https://speakerdeck.com/cotoc/ochacafe-korekarahazimeru-kubernetesji-chu >

< https://speakerdeck.com/govargo/under-the-kubernetes-controller-36f9b71b-9781-4846-9625-23c31da93014 >

Reconciliation Loop とは

Kubernetesの全体像



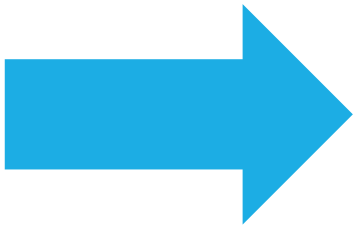
API Server経由でetcdの操作を行い、etcdの状態に応じてアクションする

Podが起動するまで(1/2)

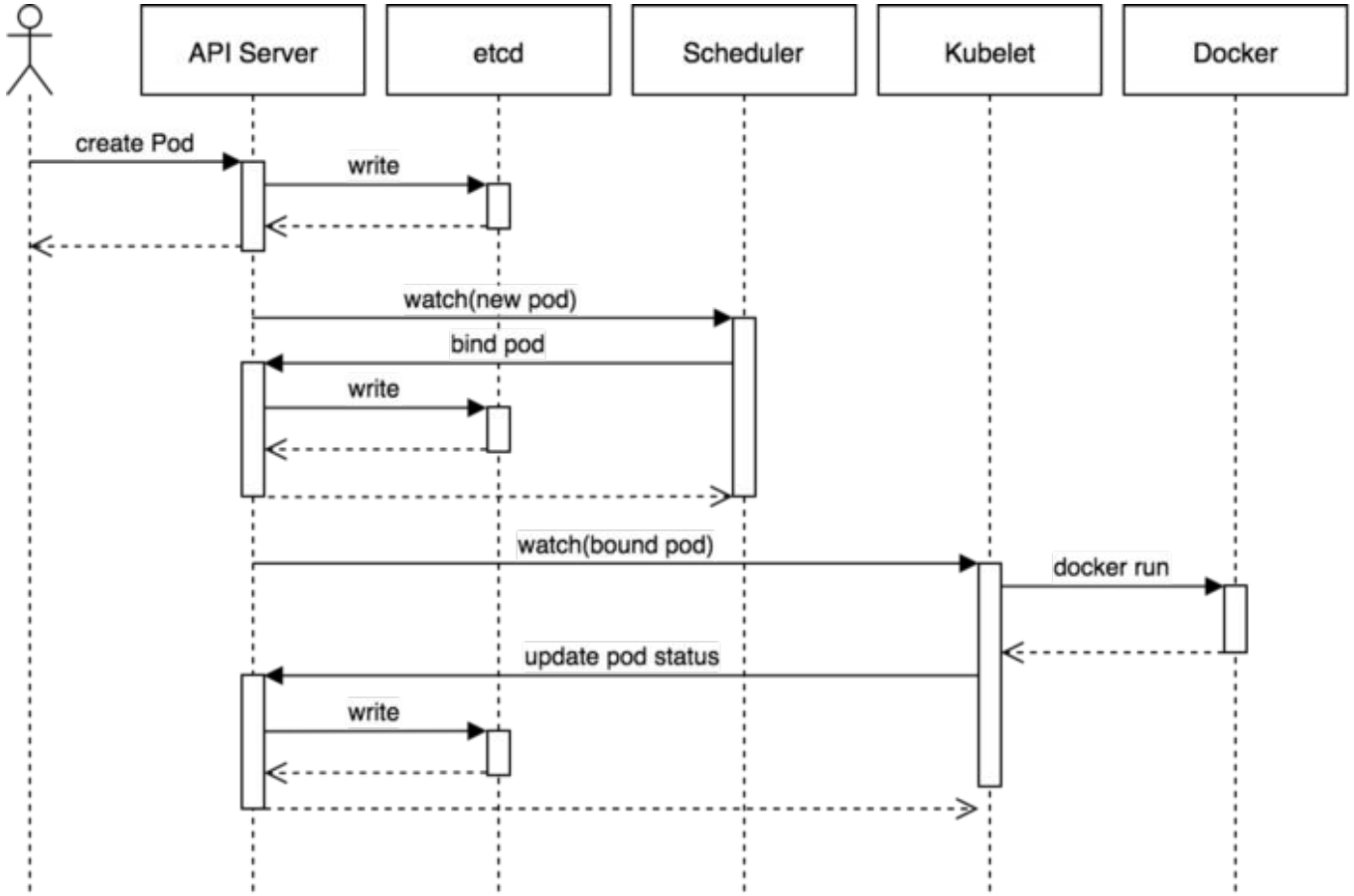
PodのManifest

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

apply



Podが起動するまで(2/2)



1. API Server 経由でPodの情報がetcdに書き込む
2. Scheduler が Pod が実行するNodeを決める
3. Kubeletは自身にNodeにPodが
   割り当てられていたら起動する
4. Kubelet はPodの状態が変化すると、
   API Server 経由でetcd更新する

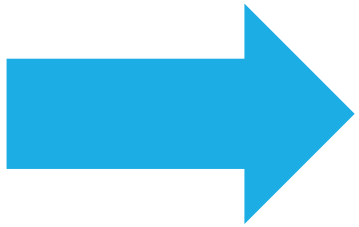
参考 : Core Kubernetes: Jazz Improv over Orchestration
< https://blog.heptio.com/core-kubernetes-jazz-improv-over-orchestration-a7903ea92ca >

Deploymentを作成してからPodが起動するまで（1/3）

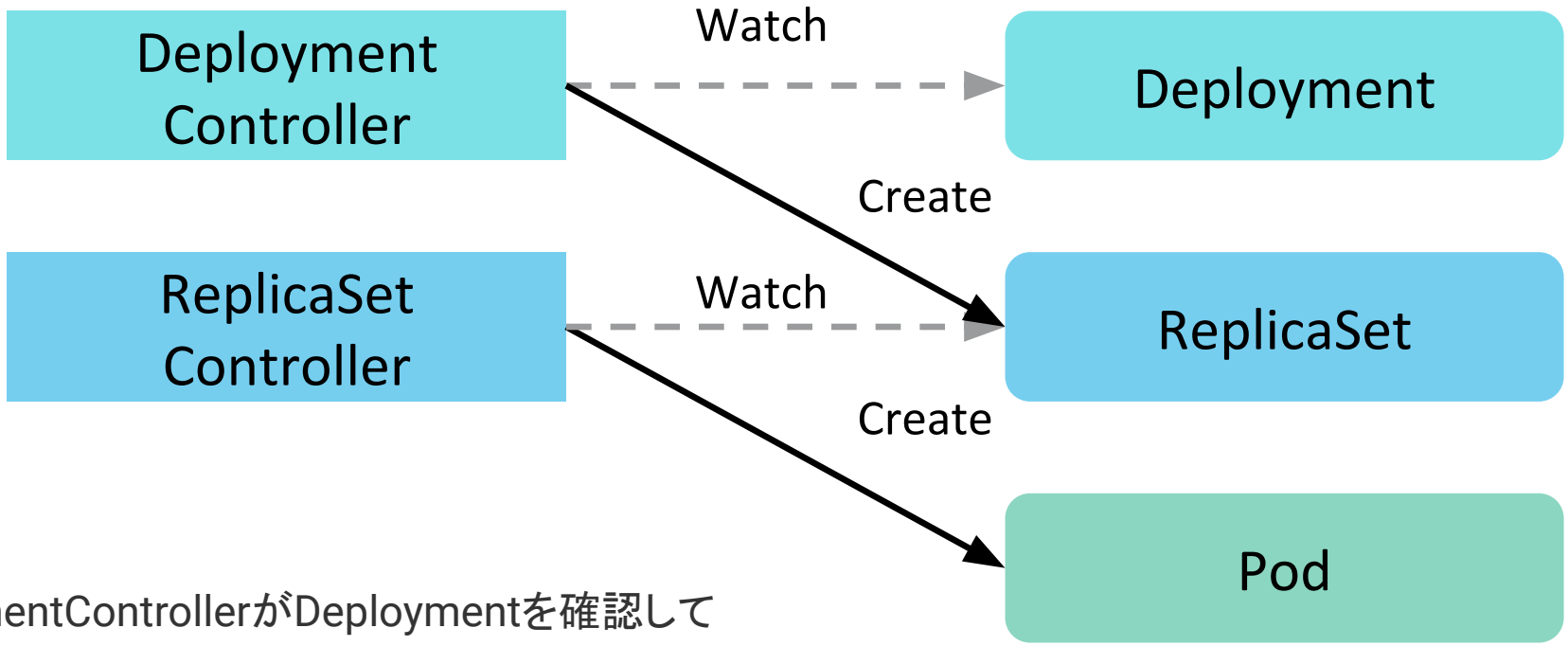
DeploymentのManifest

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
```

apply



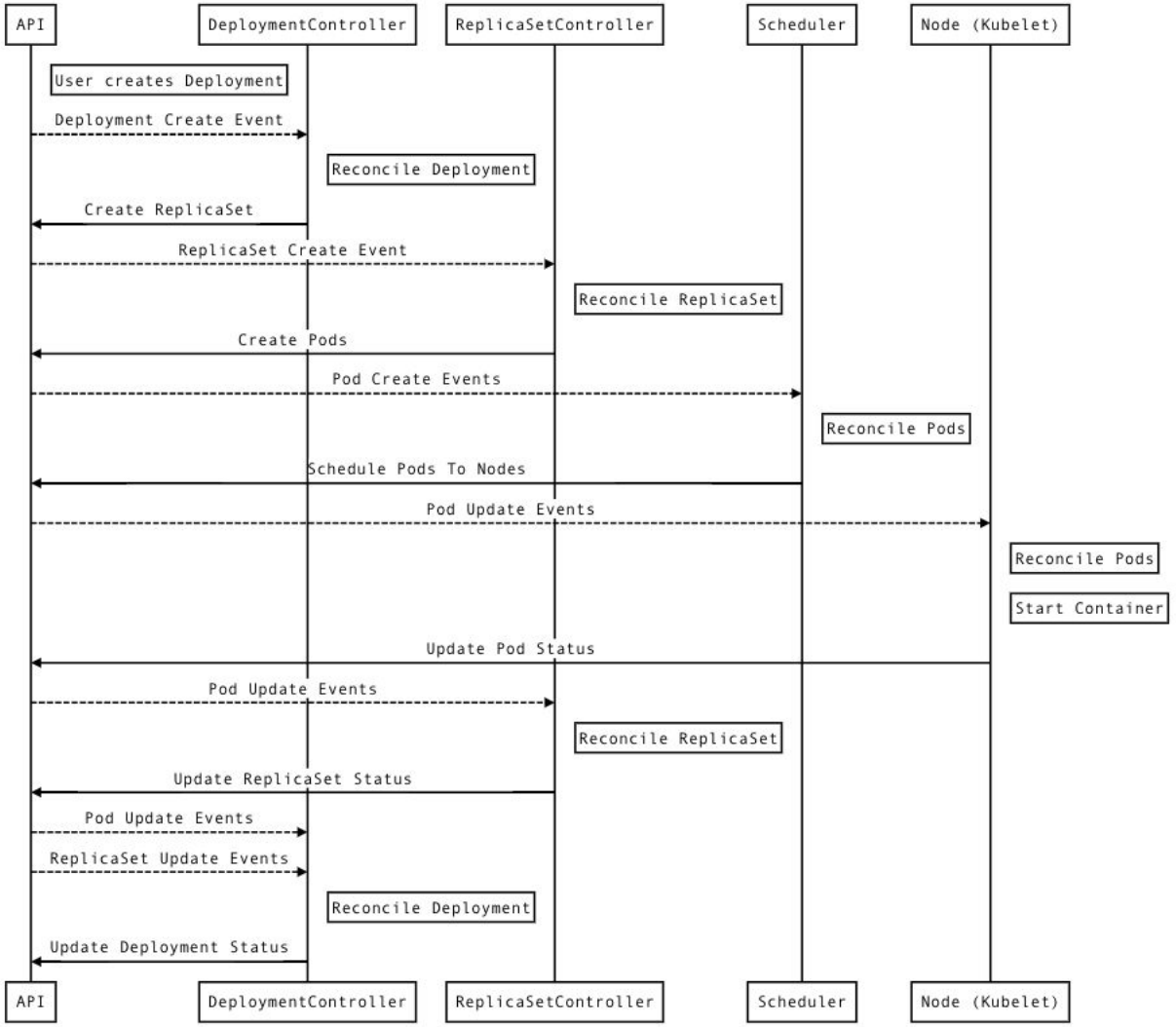
Deploymentを作成してからPodが起動するまで(2/3)



1. DeploymentControllerがDeploymentを確認して ReplicaSetを作る

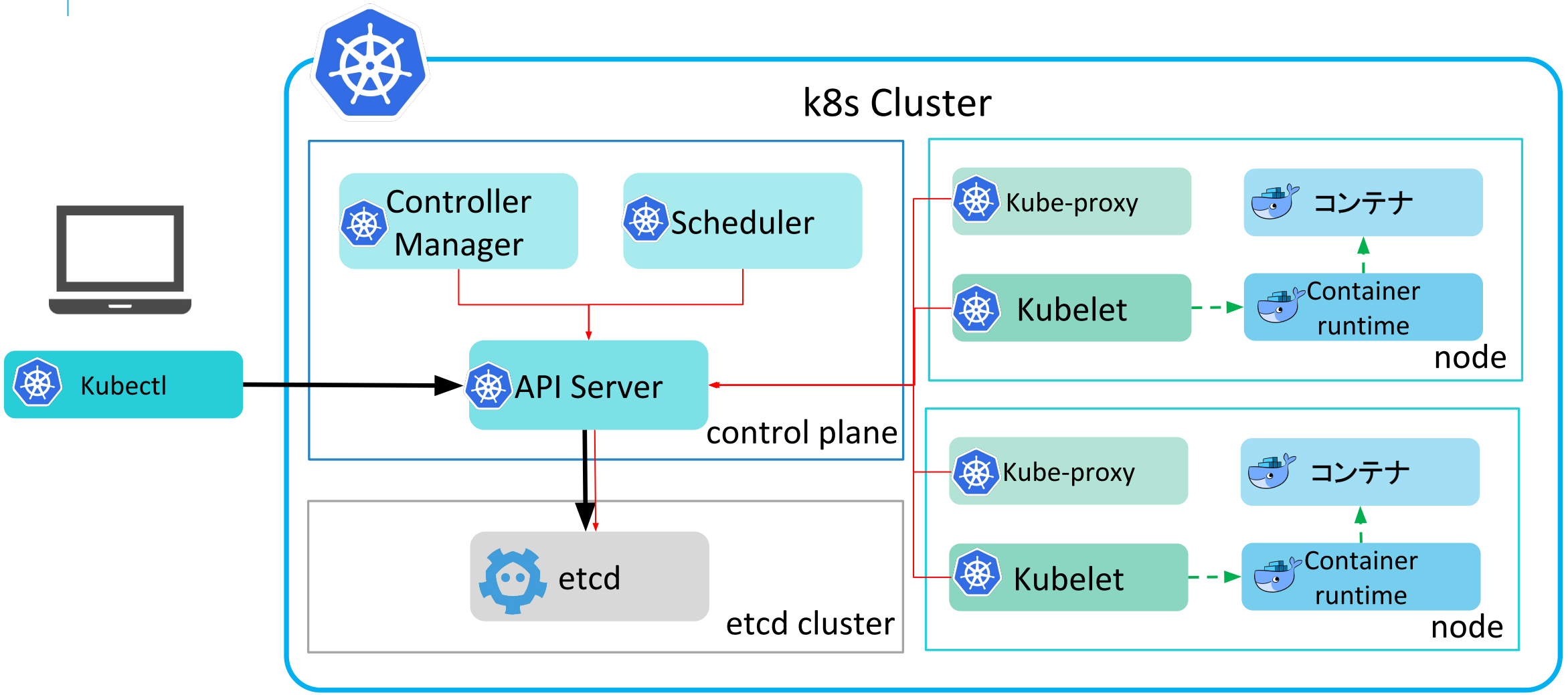
2. ReplicaSetControllerがReplicaSetを確認して Podを作る

Deploymentを作成してからPodが起動するまで（3/3）



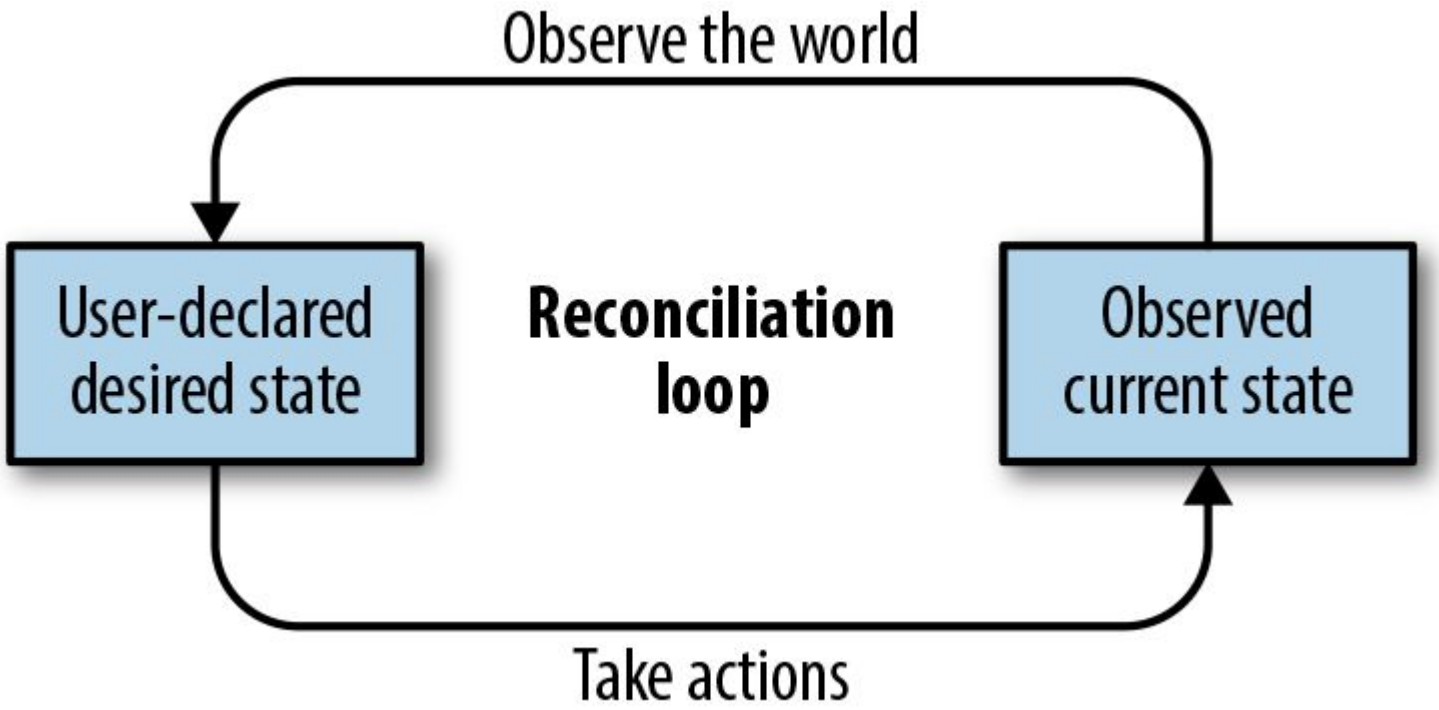
参考 : kubebuilder
< https://book-v1.book.kubebuilder.io/basics/what_is_a_controller.html >

Kubernetesの全体像



API Server経由でetcdの操作を行い、etcdの状態に応じてアクションする

Reconciliation Loop



1. 実際の状態(Actual State)を観測する
2. 実際の状態と理想の状態(Desired State)を比較する
3. 実際の状態を理想の状態となるように変更する
4. 上記を繰り返す

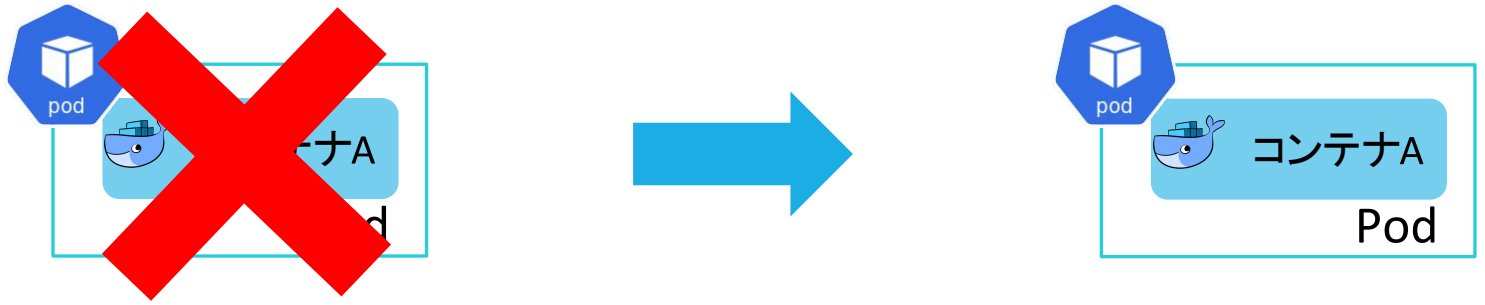
参考 : Managing Kubernetes
< https://learning.oreilly.com/library/view/managing-kubernetes/9781492033905/ >

Reconciliation Loopのおかげで嬉しい事（1/2）

- Self-healing が得られる

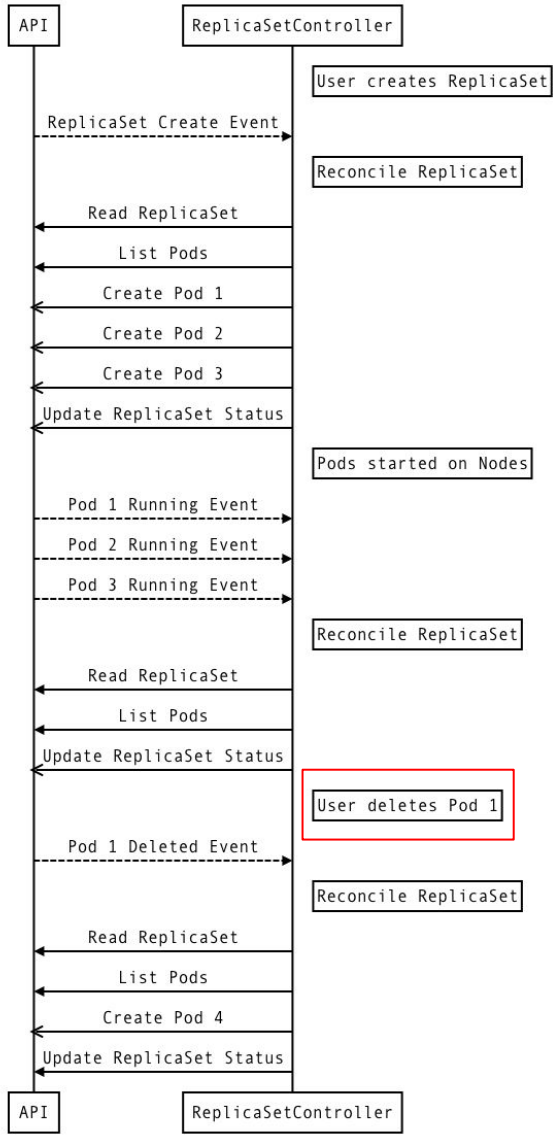
あるPodにトラブルが起こりダウン

人手を介さずに再起動



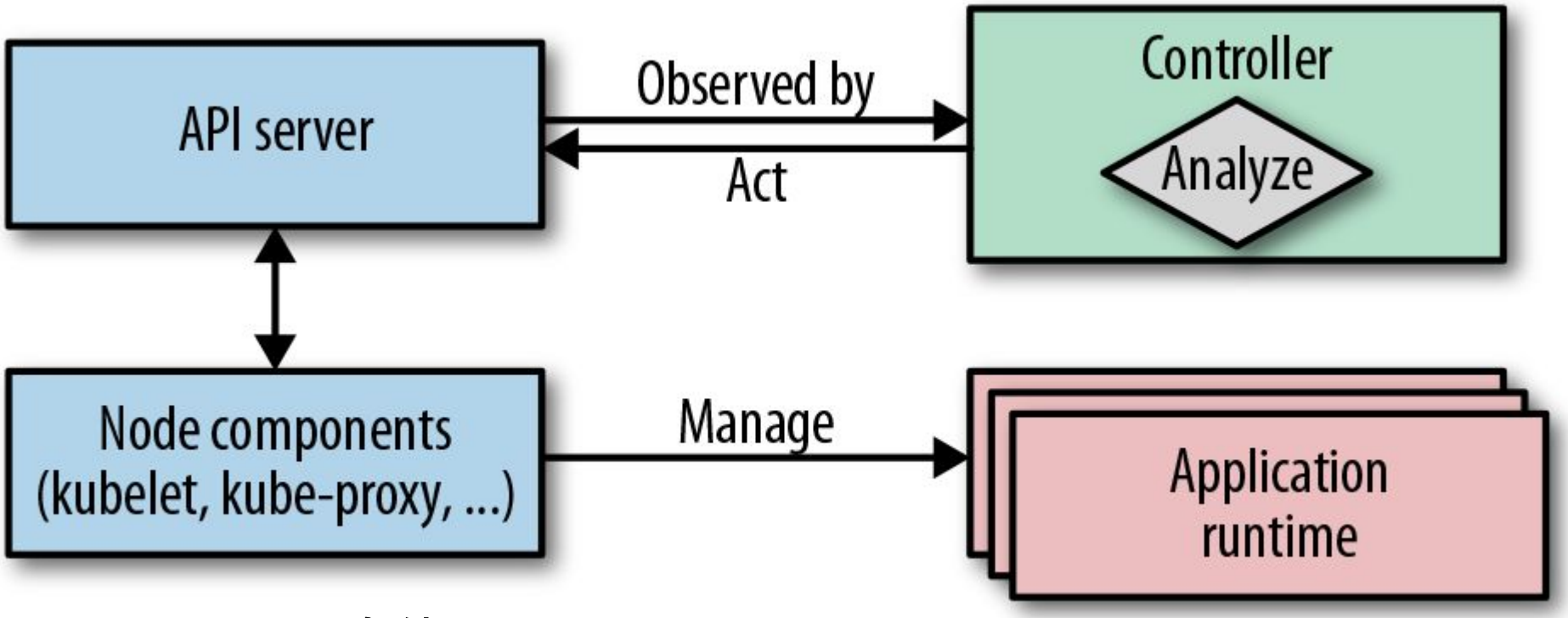
- Reconciliation Loop の仕組み中でSchedulerなどを実装する事で
コンテナオーケストレーションとしての様々な機能を実装している

Reconciliation Loopのおかげで嬉しい事（2/2）



参考 : kubebuilder
< https://book-v1.book.kubebuilder.io/basics/what_is_a_controller.html >

コンポーネント毎の制御対象の違い



- Controller：Kubernetes全体
- kubelet,kube-proxy：Node単位

参考：Kubernetes Patterns
< https://learning.oreilly.com/library/view/kubernetes-patterns/9781492050278/ >

どうして Reconciliation Loop が誕生したの？

Reconciliation Loopはいつから存在するの？

- Kubernetes が参考した Borg の時から存在している

> The idea of a reconciliation controller loop is shared throughout Borg, Omega, and Kubernetes to improve the resiliency of a system
>
> :
>
> :
>
> it compares a desired state (e.g., how many pods should match a label-selector query) against the observed state (the number of such pods that it can find), and takes actions to converge the observed and desired states.

「Borg, Omega, and Kubernetes」より

参考 : Borg, Omega, and Kubernetes
< https://research.google/pubs/pub44843/ >

Borg, Omega, and Kubernetes

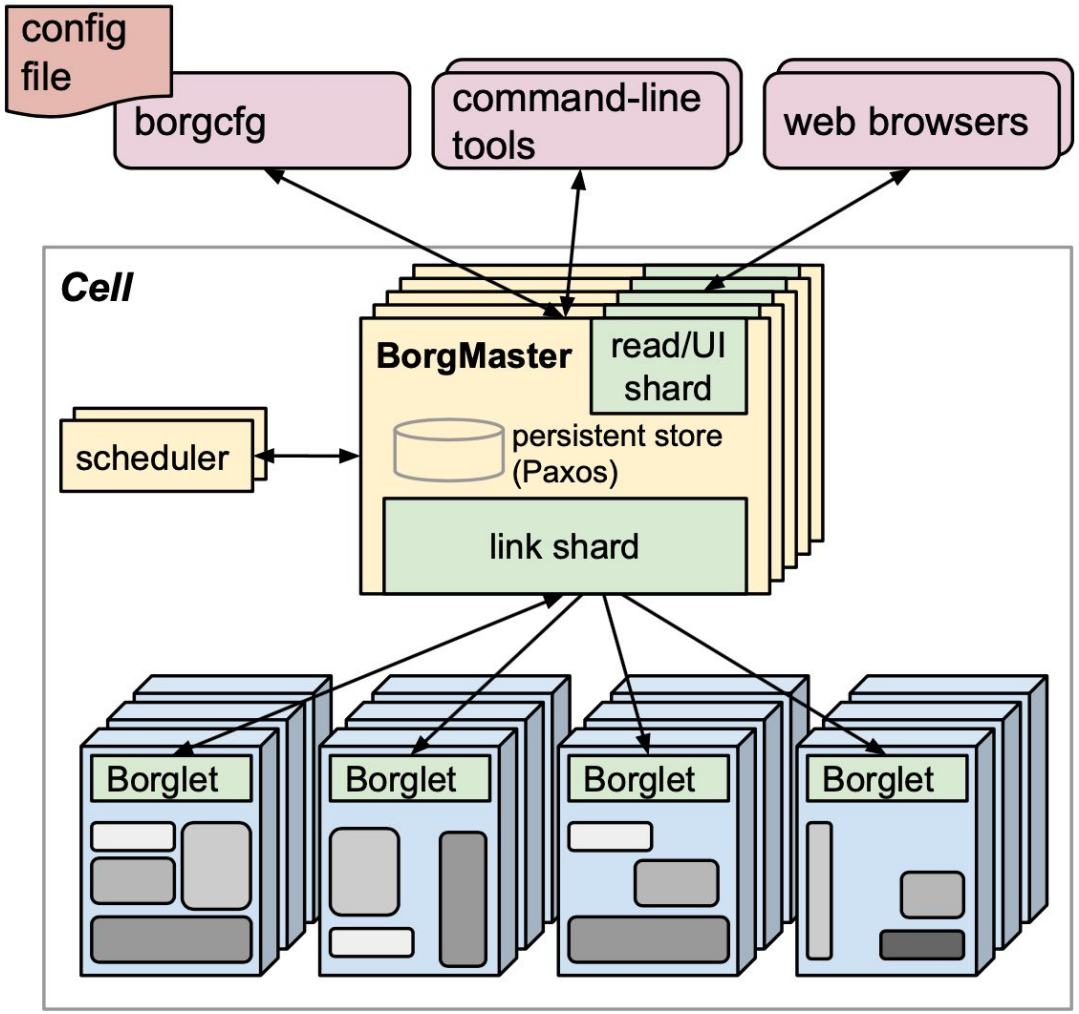
- Borg
  - 2003年からGoogle内で使用されていたコンテナオーケストレーション
- Omega
  - Google内で使用されていたコンテナオーケストレーション
  - Borgの登場以降、Kubernetesの登場以前に作られた模様
- Kubernetes
  - 2014年からGithubで公開されたコンテナオーケストレーション
  - BorgとOmegaを参考に作られている

> Borg is Google's internal container management platform. That project was started back in 2003

Kubernetes Podcast「Borg, Omega, and Kubernetes」より

参考：Kubernetes Podcast「Borg, Omega, Kubernetes and Beyond, with Brian Grant」
< https://kubernetespodcast.com/episode/043-borg-omega-kubernetes-beyond/ >

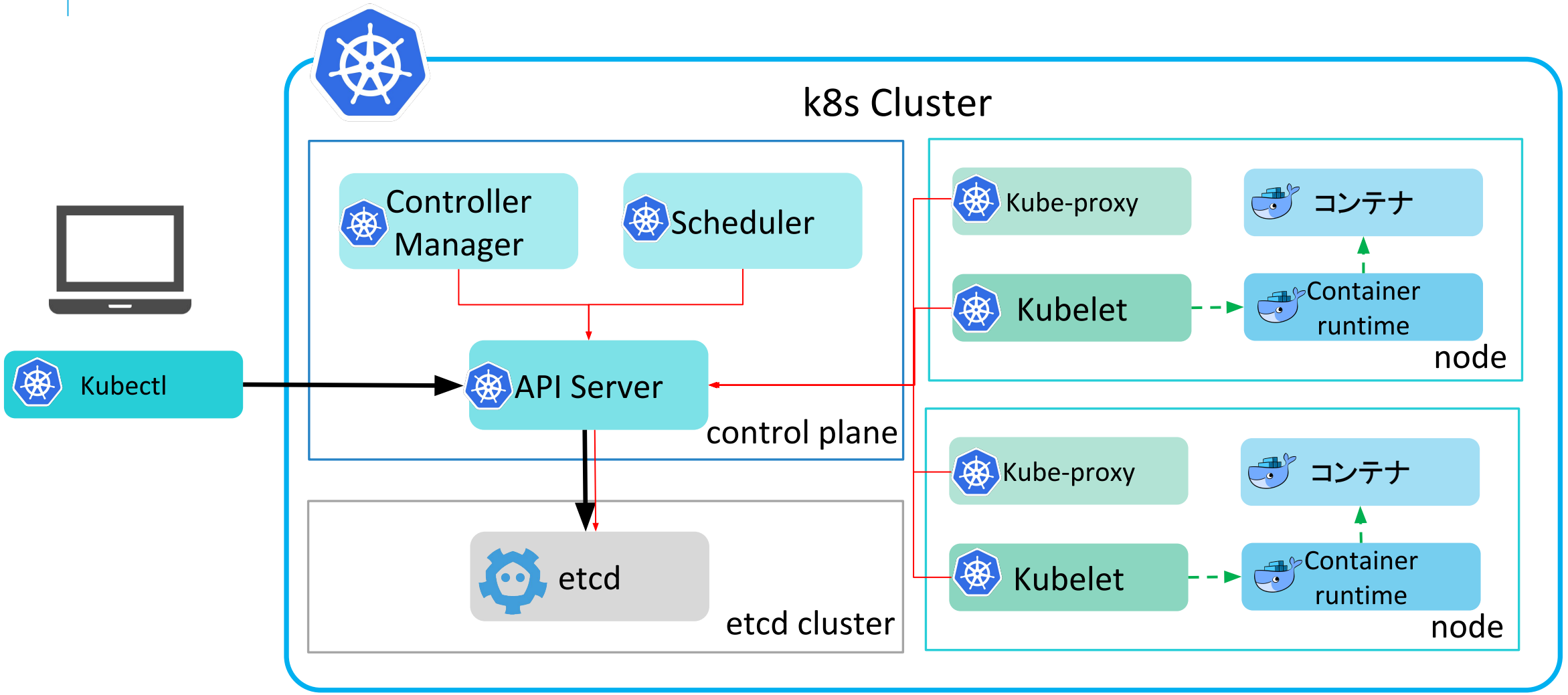
Borgのアーキテクチャ



Kubernetesとかなり似ている

参考：Large-scale cluster management at Google with Borg」
< https://research.google/pubs/pub43438/ >

Kubernetesの全体像



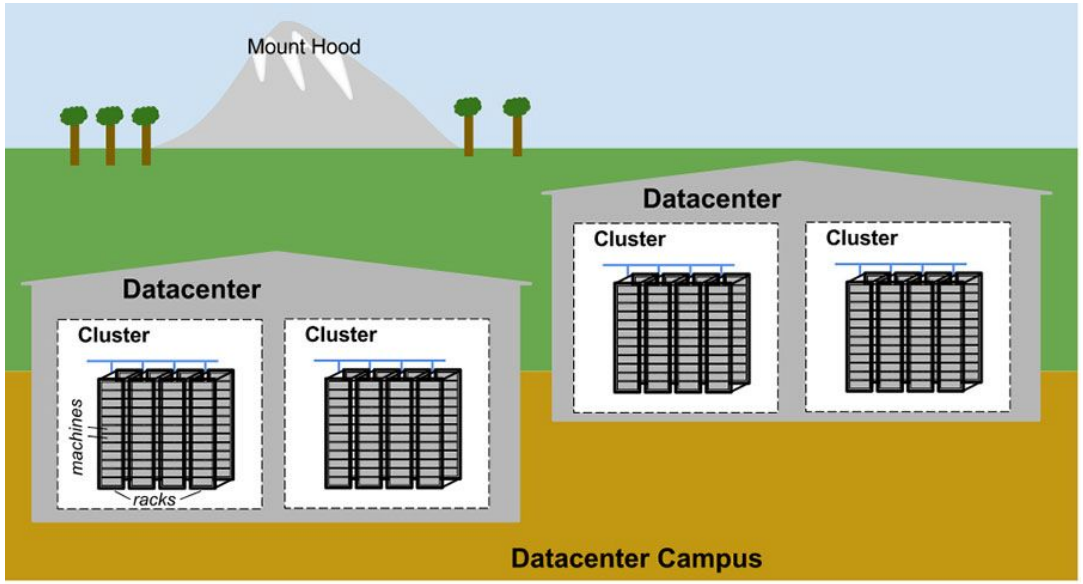
API Server経由でetcdの操作を行い、etcdの状態に応じてアクションする

当時のGoogleの環境を振り返る

- 2008年の当時でサーバ台数は20万台以上
  - サーバの故障が年間1000件、HDDの故障は数千件、電力配電装置が故障すれば一度に500台以上が停止
  - Borgが使用され始めた2003年の時点でも相当数の台数があったと考えられる

参考 : Google I/O 2008 - Underneath the Covers at Google
< https://www.youtube.com/watch?v=qsan-GQaeyk >

データセンターのイメージ



参考 : Site Reliability Engineering
< https://sre.google/books/ >

Googleの考え方を振り返る

- Eliminating Toil（トイルの撲滅）

Toil Defined

:

O(n) with service growth

If the work involved in a task scales up linearly with service size, traffic volume, or user count, that task is probably toil. An ideally managed and designed service can grow by at least one order of magnitude with zero additional work, other than some one-time efforts to add resources.

「Site Reliability Engineering」より

サービスが成長することによって増えていくタスクを
自動化しようとする姿勢が感じられる

参考 : Site Reliability Engineering
< https://sre.google/books/ >

Borg projectの動機

Once the multicore errors started, **people realized we need something more powerful that could binpack**, and that was really the motivation for Borg. And it actually was designed to slide right into the work queue hole, and be used to schedule map reduces, and it even runs today on the same port that the WorkQueue ran on. And it also subsumes some of the roles of Babysitter, so it actually created this unified platform where you could run both services and batchwork loads, and other workloads, all kinds of workloads-- eventually, almost everything Google now runs on Borg.

Kubernetes Podcast「Borg, Omega, and Kubernetes」より

The global computer is—it must be **self-repairing** to operate once it grows past a certain size, due to the essentially statistically guaranteed large number of failures taking place every second. This implies that as we move systems up the hierarchy from manually triggered, to automatically triggered, to autonomous, some capacity for self-introspection is necessary to survive.

「Site Reliability Engineering」より

自動化する必要性があった事が感じられる

参考 : Kubernetes Podcast「Borg, Omega, Kubernetes and Beyond, with Brian Grant」
< https://kubernetespodcast.com/episode/043-borg-omega-kubernetes-beyond/ >

参考 : Site Reliability Engineering
< https://sre.google/books/ >

BorgとOmegaの違い

I observed how people were using Borg, **and some of the issues it had with extensibility and scalability**, and addressing some use cases better. And that motivated the Omega project, which was really a project trying to figure out how we could improve some of the underlying primitives and internal infrastructure in Borg.

:

One of the big issues was that Borg-- Master, in particular, the control plane of Borg-- was not designed to have an extensible concept space. It had a very limited, fixed number of concepts that had machines, jobs-- tasks weren't even really a first class concept. Just arrays of tasks, which were jobs.

Kubernetes Podcast「Borg, Omega, and Kubernetes」より

The master is the kernel of a distributed system. **Borgmaster was originally designed as a monolithic system**, but over time, it became more of a kernel sitting at the heart of an ecosystem of services that cooperate to manage user jobs. For example, we split off the scheduler and the primary UI (Sigma) into separate processes, and added services for admission control, vertical and horizontal autoscaling, re-packing tasks, periodic job submission (cron), workflow management, and archiving system actions for off-line querying. Together, these have allowed us to scale up the workload and feature set without sacrificing performance or maintainability.

「Large-scale cluster management at Google with Borg」より

参考：Kubernetes Podcast「Borg, Omega, Kubernetes and Beyond, with Brian Grant」
< https://kubernetespodcast.com/episode/043-borg-omega-kubernetes-beyond/ >

参考：Large-scale cluster management at Google with Borg」
< https://research.google/pubs/pub43438/ >

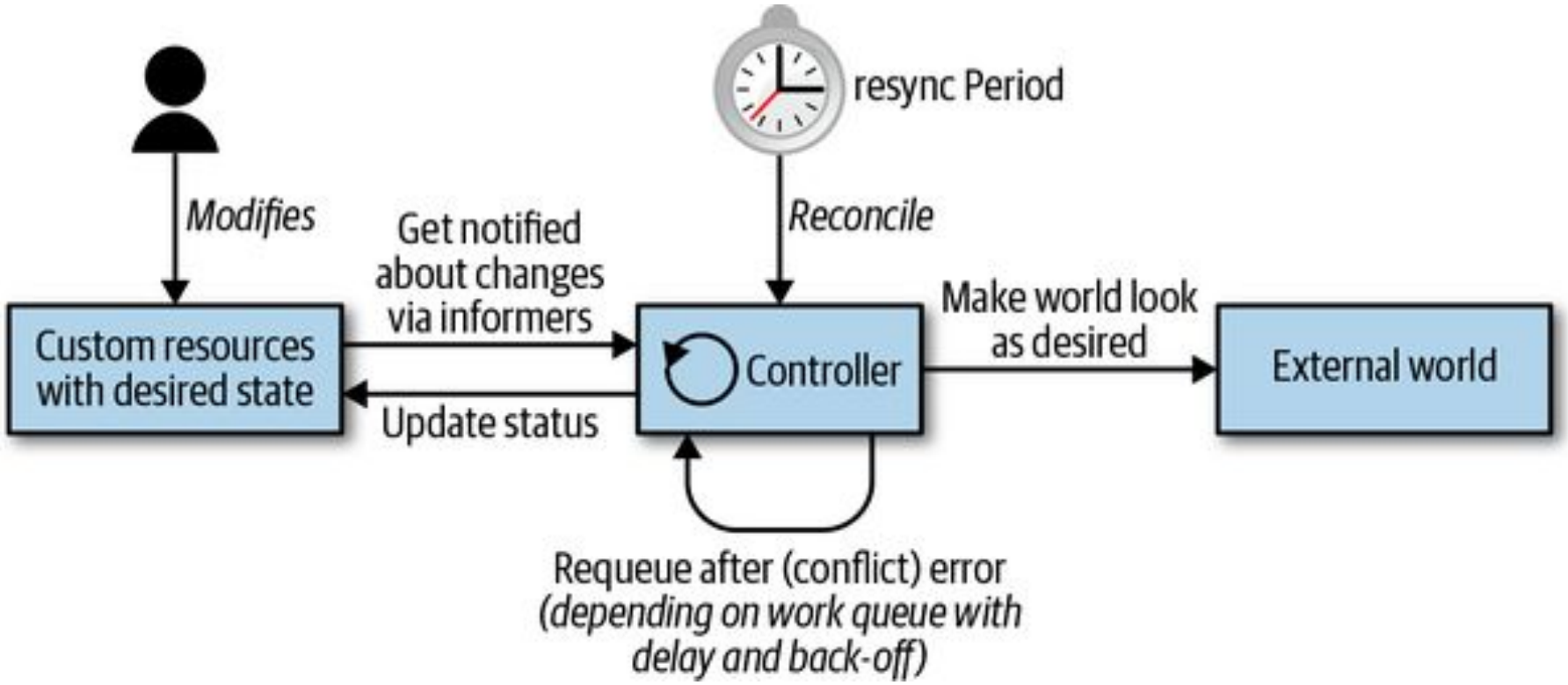
Reconciliation Loopが誕生したわけの振り返り

- サービスが成長するにつれてハードウェア障害が増える
  - 大きなサービスでは毎秒毎に複数箇所で故障が発生する
- サービスの成長に比例して肥大化するタスクの抑制が必要
  - 同じ作業の繰り返しに追われ、生産的な活動ができなくなる
- 自動化の必要があった
  - Reconciliation Loop を用いたSelf-healing が必要

あくまで、調査資料からの考察なので
他の要因も考えられる

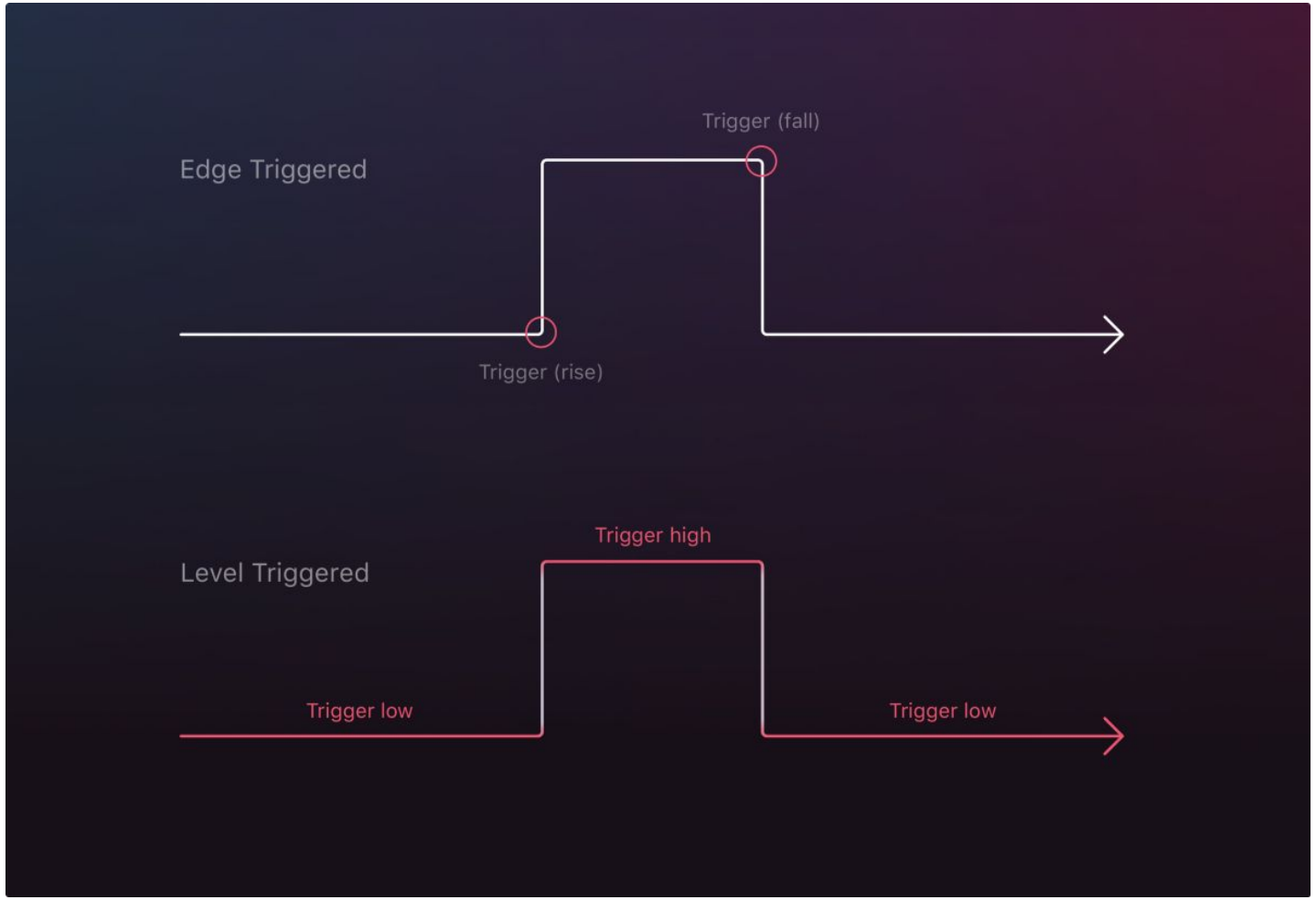
Reconciliation Loop の細かい話

Reconciliation Loop再び



参考 : Programming Kubernetes
< https://www.oreilly.com/library/view/programming-kubernetes/9781492047094/ >

Edge-driven triggersとLevel-driven triggersの概念（1/3）



Edge Triggered

```
> let a = 3 ;
> a += 4 ;
<
```

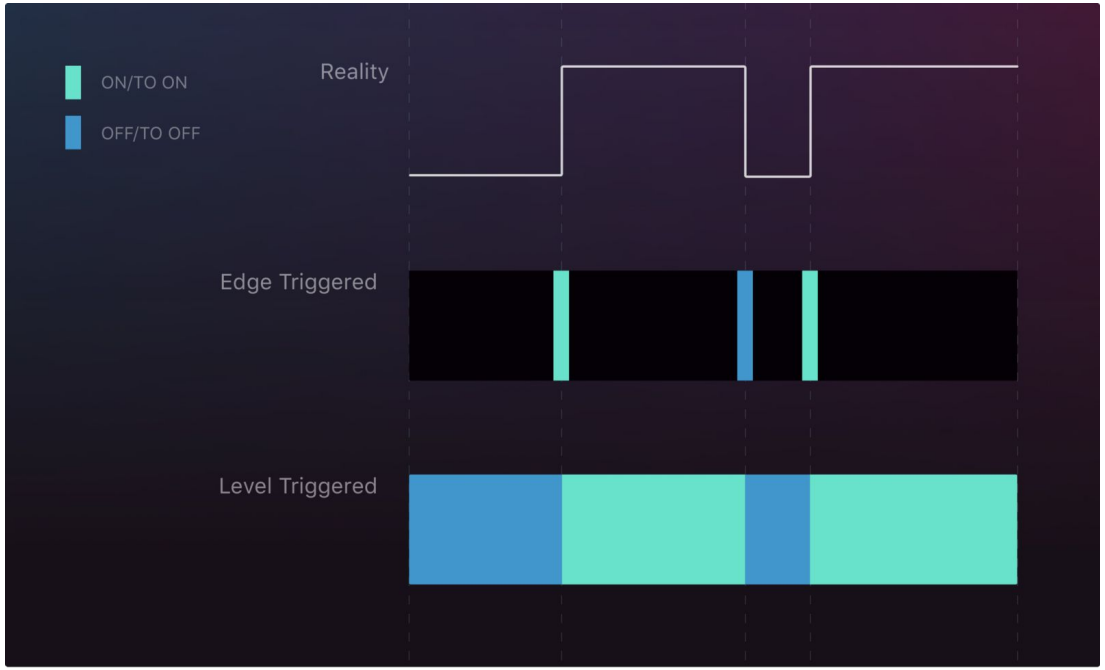
Level Triggered

a is 7

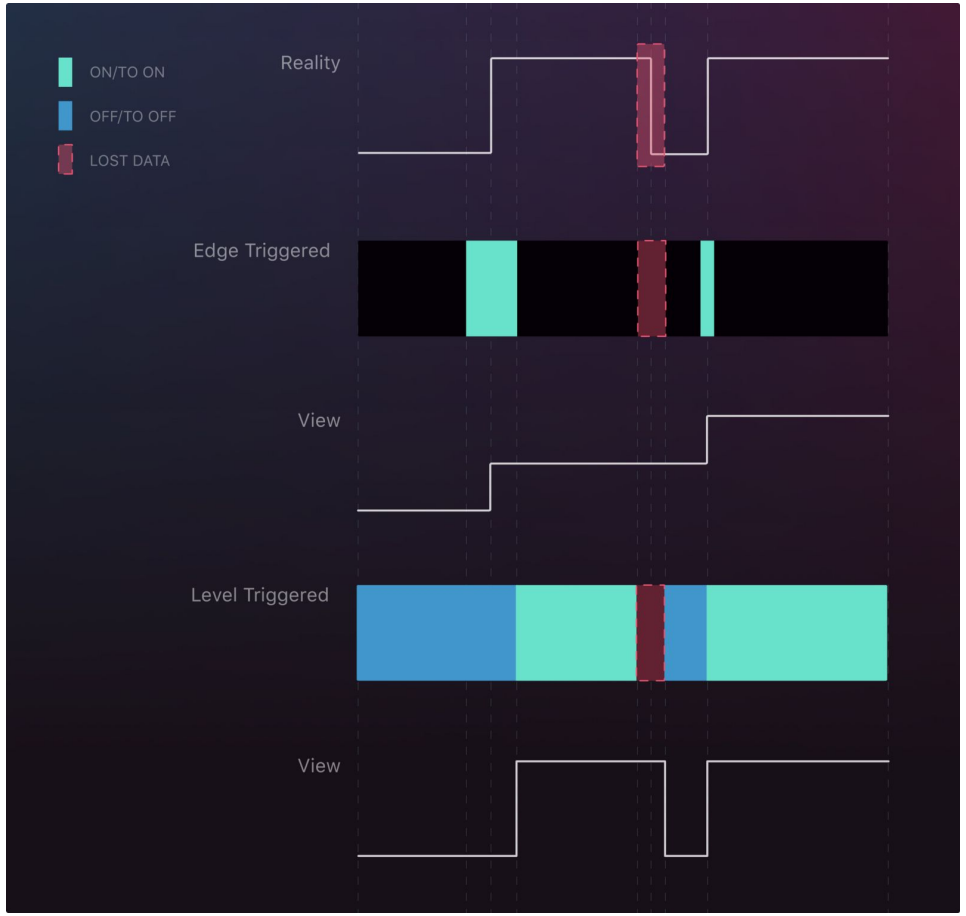
参考 : Level Triggering and Reconciliation in Kubernetes
< https://hackernoon.com/level-triggering-and-reconciliation-in-kubernetes-1f17fe30333d >

Edge-driven triggersとLevel-driven triggersの概念（2/3）

理想的な状態



障害が発生した状態



減算の命令がロスト

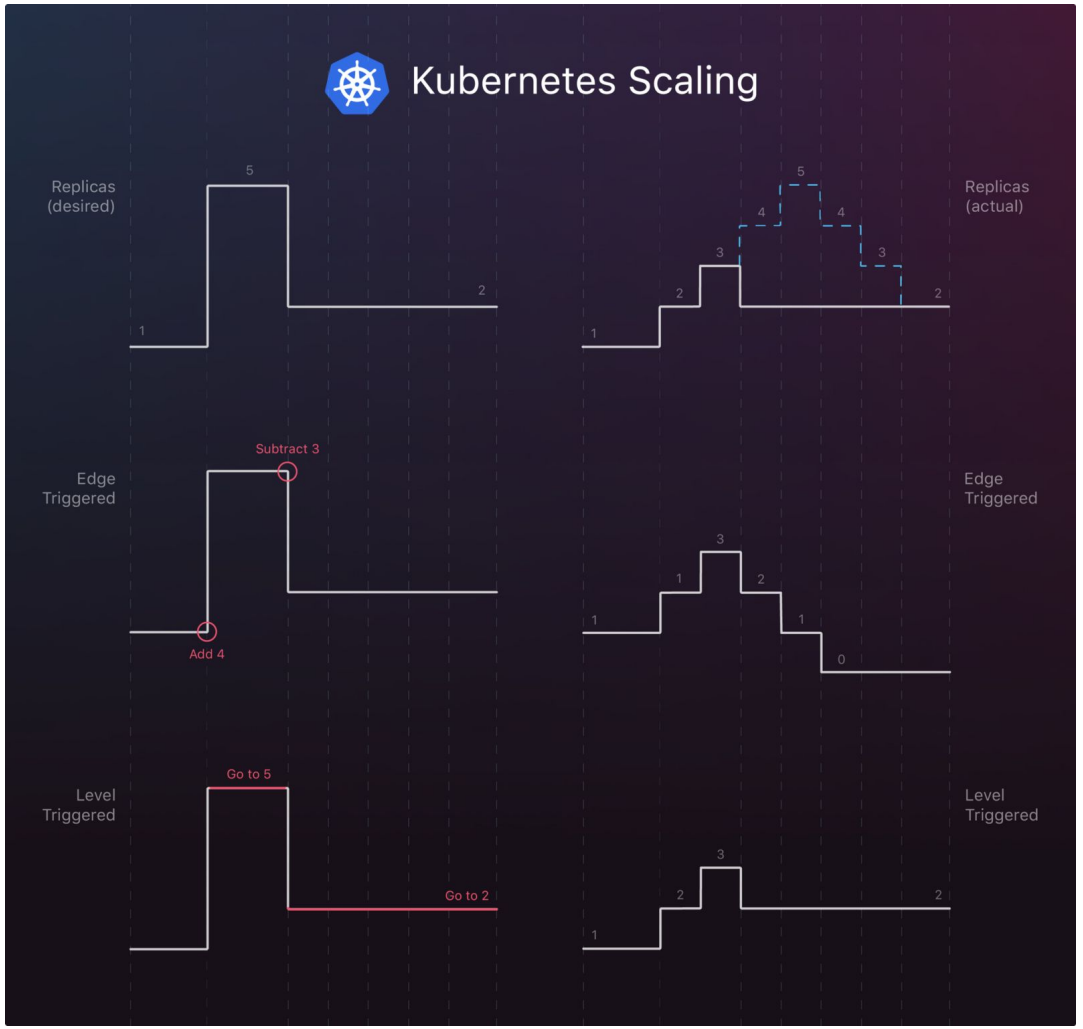
参考 : Level Triggering and Reconciliation in Kubernetes
< https://hackernoon.com/level-triggering-and-reconciliation-in-kubernetes-1f17fe30333d >

Edge-driven triggersとLevel-driven triggersの概念（3/3）

Desired

Edge Triggered

Level Triggered

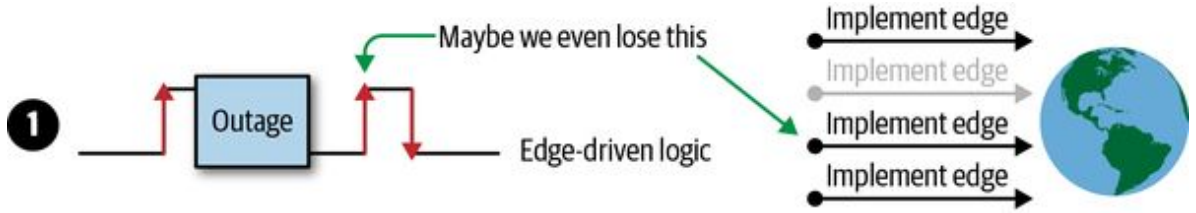


この例では
“Add 4” が完了する前に
“Subtract 3” が開始されている

参考 : Level Triggering and Reconciliation in Kubernetes
< https://hackernoon.com/level-triggering-and-reconciliation-in-kubernetes-1f17fe30333d >

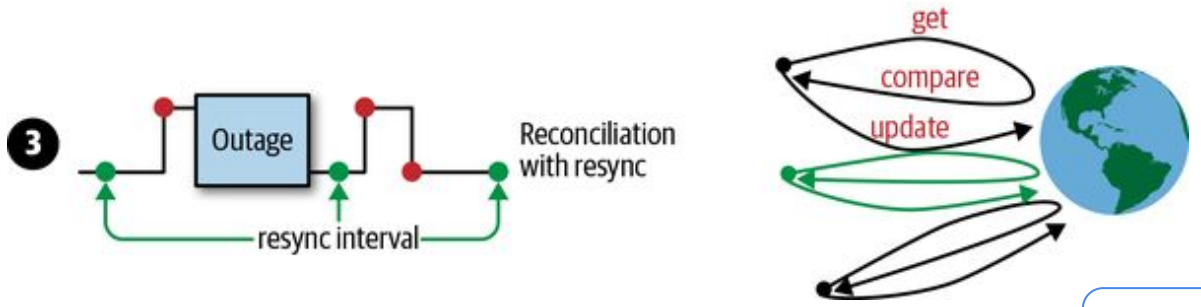
Edge-driven triggers と Level-driven triggersの実装の話

- 概念を実装していく方法



1. Edge driven triggersのみのロジック
(2個目の処理は失敗している)

2. Edge driven triggerによるイベント発生時に
最新の状態を取得して更新する
(Level triggerの動作をEdge drivenに行う)



3. 定期的に最新の状態を取得して更新する
(Level triggerの動作をresync interval間隔で行う)

参考 : Programming Kubernetes
< https://www.oreilly.com/library/view/programming-kubernetes/9781492047094/ >

定期的に更新するということは
定期的なポーリングを処理し切れる事が前提となっている

Reconciliation Loopを実現する方法

When designing a system like Kubernetes, there are generally two different approaches that you can take—a **monolithic state-based approach** or a **decentralized controller–based approach**.

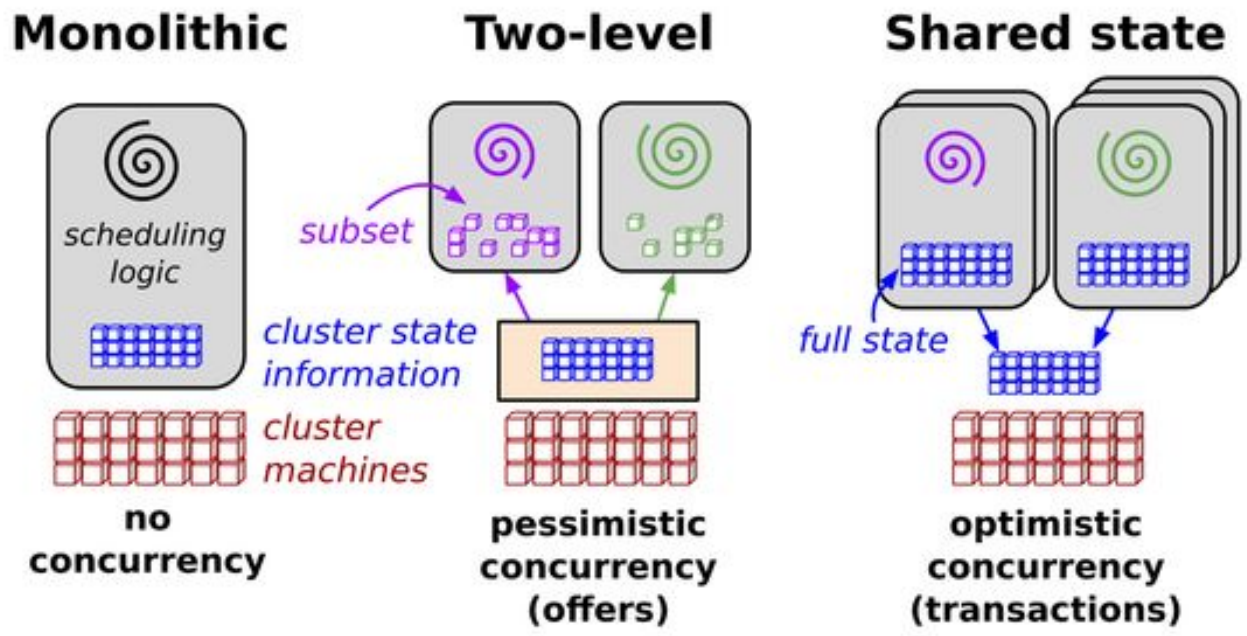
「Managing Kubernetes」より

- Monolithic state-based approach
  - 初期のBorgのアーキテクチャ
- Decentralized controller–based approach
  - Kubernetesのアーキテクチャ

SchedulerやControllerにコンポーネントが細かく分かれてるのはKubernetesでの実装上の話で、Reconciliation Loopはモノリスでも実装可

参考 : Managing Kubernetes
< https://learning.oreilly.com/library/view/managing-kubernetes/9781492033905/ >

Schedulerのアーキテクチャ



- Monolithic : 初期のBorg
- Two-level : Mesos
- Snared state : Omega, Kubernetes

- KubernetesではReconciliation Loopopに加えて
  API-Server(etd) 経由でのoptimistic concurrency (楽観的並行性制御) を用いることで
  controller, scheduler, kubelet 等を分離している

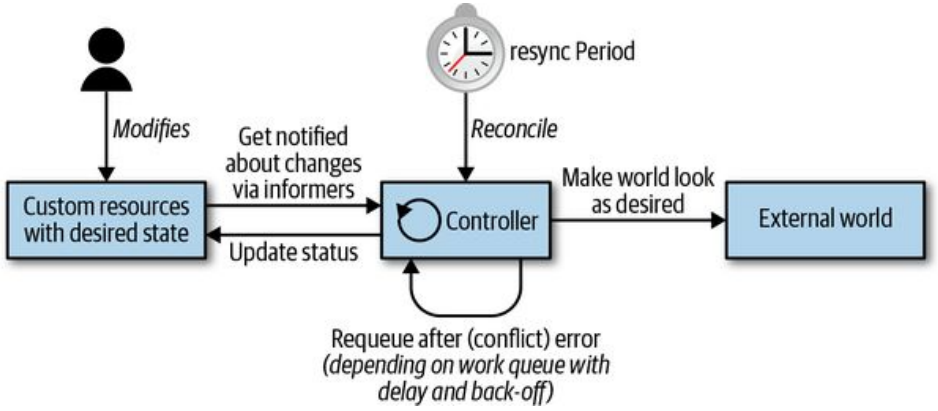
参考 : Omega: flexible, scalable schedulers for large compute clusters
< https://research.google/pubs/pub41684/ >

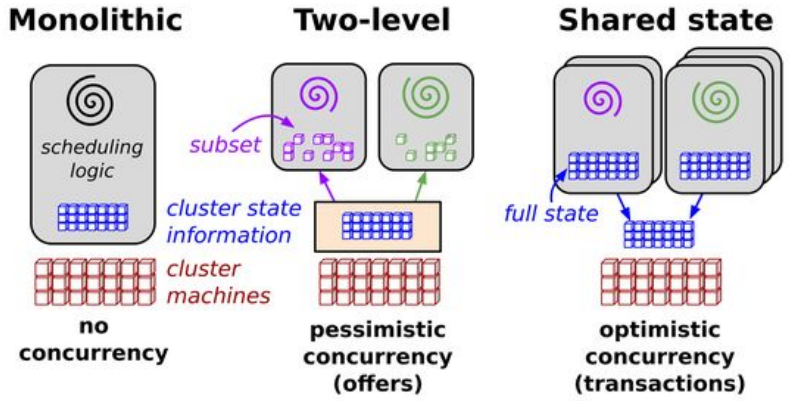
まとめ

KubernetesのControl Planeについて

- 各コンポーネントはReconciliation Loopの仕組みに基づいて
  必要な機能を実装している

- 各コンポーネントは、optimistic concurrencyの仕組みを用いて分離されている
  (競合時のリトライはclient側の責務)

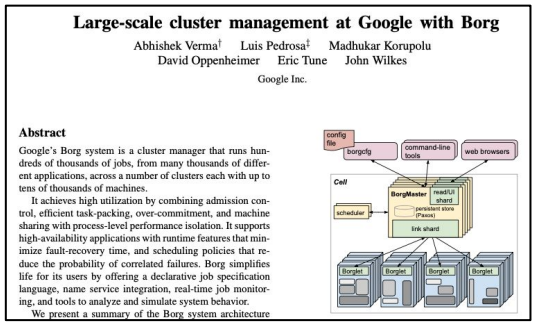




参考 : Programming Kubernetes
< https://www.oreilly.com/library/view/programming-kubernetes/9781492047094/ >

参考資料(1/4)

- 論文
  - Large-scale cluster management at Google with Borg
    < https://research.google/pubs/pub43438/ >
  - Borg, Omega, and Kubernetes
    < https://research.google/pubs/pub44843/ >
  - Omega: flexible, scalable schedulers for large compute clusters
    < https://research.google/pubs/pub41684/ >


Large-scale cluster management at Google with Borg

Abhishek Verma† Luis Pedrosa‡ Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes

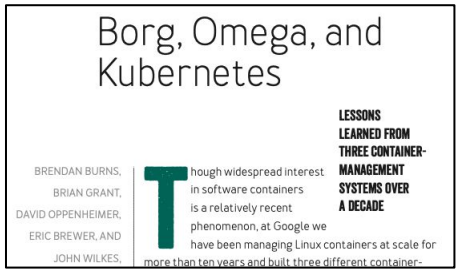
Google Inc.

Abstract

Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines.

It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior.

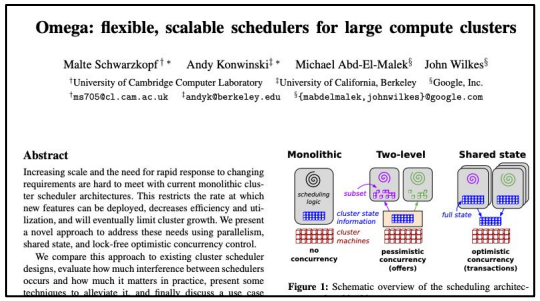
We present a summary of the Borg system architecture


Borg, Omega, and Kubernetes

BRENDAN BURNS, BRIAN GRANT, DAVID OPPENHEIMER, ERIC BREWER, AND JOHN WILKES,

LESSONS LEARNED FROM THREE CONTAINER-MANAGEMENT SYSTEMS OVER A DECADE

Though widespread interest in software containers is a relatively recent phenomenon, at Google we have been managing Linux containers at scale for more than ten years and built three different container-


Omega: flexible, scalable schedulers for large compute clusters

Malte Schwarzkopf† * Andy Konwinski‡ * Michael Abd-El-Malek§ John Wilkes§

†University of Cambridge Computer Laboratory ‡University of California, Berkeley §Google, Inc.

†ms705@cl.cam.ac.uk ‡andyk@berkeley.edu §{mabdelmalek,johnwilkes}@google.com

Abstract

Increasing scale and the need for rapid response to changing requirements are hard to meet with current monolithic cluster scheduler architectures. This restricts the rate at which new features can be deployed, decreases efficiency and utilization, and will eventually limit cluster growth. We present a novel approach to address these needs using parallelism, shared state, and lock-free optimistic concurrency control.

We compare this approach to existing cluster scheduler designs, evaluate how much interference between schedulers occurs and how much it matters in practice, present some techniques to alleviate it, and finally discuss a use case

Monolithic Two-level Shared state

scheduling logic

no concurrency

subset

cluster state information

cluster machines

pessimistic concurrency (offers)

full state

optimistic concurrency (transactions)

Figure 1: Schematic overview of the scheduling architec-

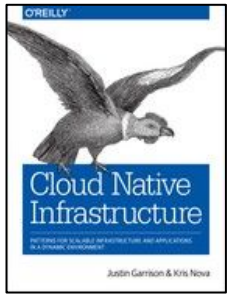
- Kubernetes Podcast
  - Borg, Omega, Kubernetes and Beyond, with Brian Grant
    < https://kubernetespodcast.com/episode/043-borg-omega-kubernetes-beyond/ >
  - Kubernetes Origins, with Joe Beda
    < https://kubernetespodcast.com/episode/012-kubernetes-origins/ >

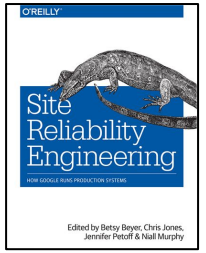
Kubernetes Podcast

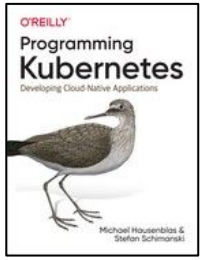
from Google

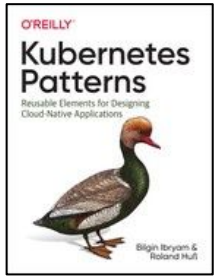
参考資料(2/4)

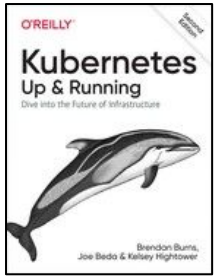
- 書籍
  - Cloud Native Infrastructure
    < https://www.oreilly.com/library/view/cloud-native-infrastructure/9781491984291/ >
  - Site Reliability Engineering
    < https://sre.google/books/>
  - Programming Kubernetes
    < https://www.oreilly.com/library/view/programming-kubernetes/9781492047094/ >
  - Kubernetes Patterns
    < https://learning.oreilly.com/library/view/kubernetes-patterns/9781492050278/ >
  - Kubernetes: Up and Running, 2nd Edition
    < ttps://learning.oreilly.com/library/view/kubernetes-up-and/9781492046523/ >
  - Managing Kubernetes
    < https://learning.oreilly.com/library/view/managing-kubernetes/9781492033905/ >
  - 実践入門 Kubernetesカスタムコントローラーへの道
    < https://nextpublishing.jp/book/11389.html >

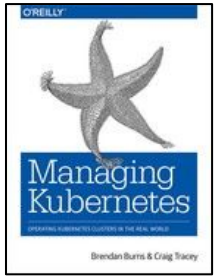












参考資料(3/4)

- ブログ
  - Level Triggering and Reconciliation in Kubernetes
    < https://hackernoon.com/level-triggering-and-reconciliation-in-kubernetes-1f17fe30333d >
  - Core Kubernetes: Jazz Improv over Orchestration
    < https://blog.heptio.com/core-kubernetes-jazz-improv-over-orchestration-a7903ea92ca >
  - Borg: The Predecessor to Kubernetes
    < https://kubernetes.io/blog/2015/04/borg-predecessor-to-kubernetes/ >
  - An introduction to containers, Kubernetes, and the trajectory of modern cloud computing
    < https://cloudplatform.googleblog.com/2015/01/in-coming-weeks-we-will-be-publishing.html >
  - Kubernetesを拡張しよう
    < https://www.ianlewis.org/jp/extending-kubernetes-ja >
  - What happens when … Kubernetes edition!
    < https://github.com/jamiehannaford/what-happens-when-k8s >
  - What is a Controller
    < https://book-v1.book.kubebuilder.io/basics/what_is_a_controller.html >

参考資料(4/4)

- スライド
  - これからはじめる！Kubernetes基礎
    < https://speakerdeck.com/cotoc/ochacafe-korekarahazimeru-kubernetesji-chu >
  - ゼロから始めるKubernetes Controller / Under the Kubernetes Controller
    < https://speakerdeck.com/govargo/under-the-kubernetes-controller-36f9b71b-9781-4846-9625-23c31da93014 >
  - Kubernetesを拡張して日々のオペレーションを自動化する
    < https://speakerdeck.com/ladicle/kuberneteswokuo-zhang-siteri-falseoperesiyonwozi-dong-hua-suru >

- Twitter
  - Brendan Burns(@brendandburns)
  - Joe Beda(@jbeda)
  - Brian Grant(@bgrant0607)