SAKURA internet

忙しい人のためのGitOps入門

@Open Developers Conference 2019

https://www.sakura.ad.jp/

DAY
2019/8/24

DEPARTMENT
技術本部

NAME
伊藤竜一 (@amaya382)

SAKURA internet

伊藤竜一

🌸しごと

- 所属: さくらインターネット株式会社 技術本部
- 新規プロジェクト×2でKubernetesを導入中
  - 1つはAPIサーバのデプロイ先、バッチ処理実行環境として
  - 1つはスケーラブルなデータ処理基盤として

💻こじん

- Twitter: @amaya382
- Kubernetesに関する同人誌を書いたり
  - 技術書典7もKubernetes関係で執筆中📗✏️
  - お08C「かいていどうくつ」

SAKURA internet

前提

- Docker・Gitの基本的な概念

対象者

- k8s上にアプリをデプロイするイメージを掴みたい人

ゴール

- 「k8sでアプリデプロイ」→「GitOps」という選択肢を知る
  - なぜ必要なのか、導入するとどうなるのか

NOTゴール🤦‍♀️

- k8s・GitOpsの仕組み、実際のツール、導入に必要な物事

- k8sの概念をもう少し知りたい
  → https://speakerdeck.com/amaya382/kubernetestutehe-gadekirufalse-dounatuterufalse
- GitOpsの仕組み、導入方法まで知りたい（本資料のLong版）
  → https://speakerdeck.com/amaya382/kuberneteswozui-da-xian-nihuo-kasutamefalsegitopsru-men

SAKURA internet

👉 k8sを前提としたGitを中心に据えたCI/CD手法

- 原典: https://www.weave.works/technologies/gitops/

weaveworks

ストレスフリーな爆速デプロイ

ほぼ手放しで安定

全てのサービスの状態・設定をコード化、Gitで管理

- 属人化したHackを防ぐ
- デプロイ状態の確認を容易に
- 開発だけでなく、運用作業もGitの操作で統一

適切な関心の分離

- 権限の分離
- セキュリティリスクの抑制

👉 k8sを前提としたGitを中心に据えたCI/CD手法

- 原典: https://www.weave.works/technologies/gitops/

weaveworks

ストレスフリーな爆速デプロイ

全てのサービスの状態・設定をコード化、Gitで管理

- 属人化したHackを防ぐ
- デプロイ状態の確認を容易に
- 開発だけでなく、運用作業もGitの操作で統一

例えば…
Git Revertでサービス巻き戻し

適切な関心の分離

- 権限の分離
- セキュリティリスクの抑制

SAKURA internet

👉 k8sを前提としたGitを中心に据えたCI/CD手法

- 原典: https://www.weave.works/technologies/gitops/

weaveworks

ストレスフリーな爆速デプロイ

全てのサービスの状態・設定をコード化、Gitで管理

- 属人化したHackを防ぐ
- デプロイ状態の確認を容易に
- 開発だけでなく、運用作業もGitの操作で統一

適切な関心の分離

- 権限の分離
- セキュリティリスクの抑制

CIの権限は最低限に

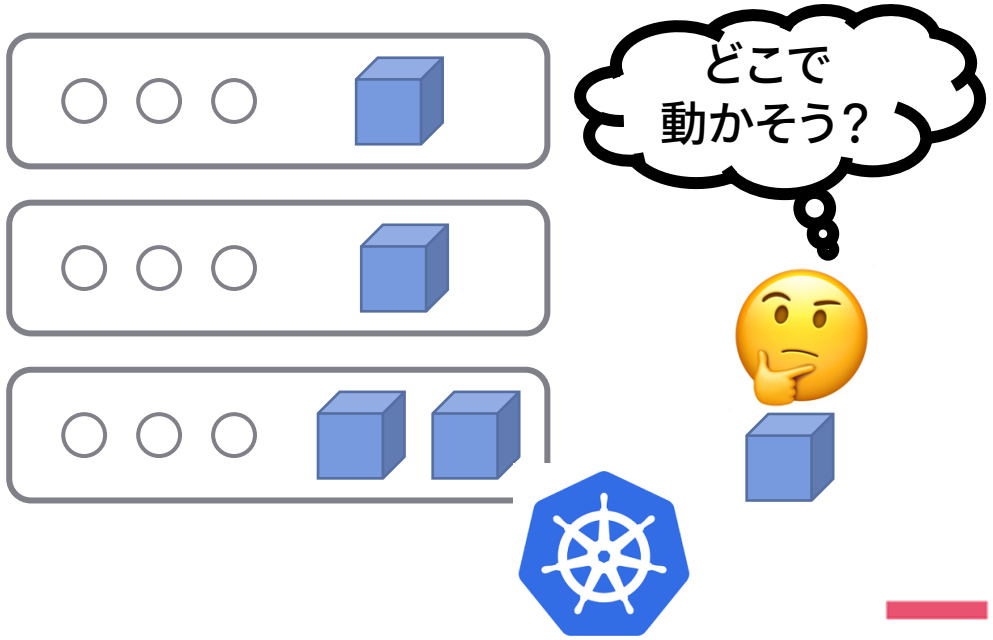
そもそも読み方がよく分からん

- Kubernetes: クーバーネィティス (諸説あり)
- k8s: ケーエイツ (ケーハチエス)
- kube*: キューブ*

コンテナベースのアプリを
本番運用するに必要な機能群

コンテナオーケストレーションシステム

- コンテナ間のネットワークの管理
- 計算機リソースの分配
- 自動的なスケール・障害時復旧
- etc.



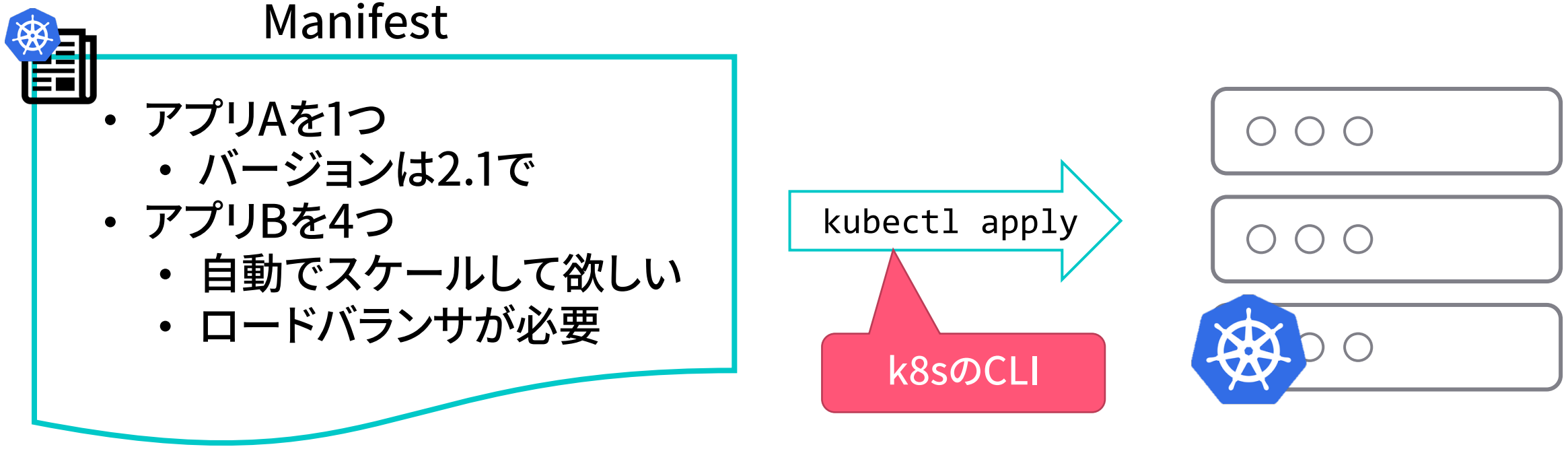
※Docker以外にも対応しているが、殆どの場合Dockerと利用される

- k8sの特徴:
  リソースをManifestと呼ばれるファイルで管理可能

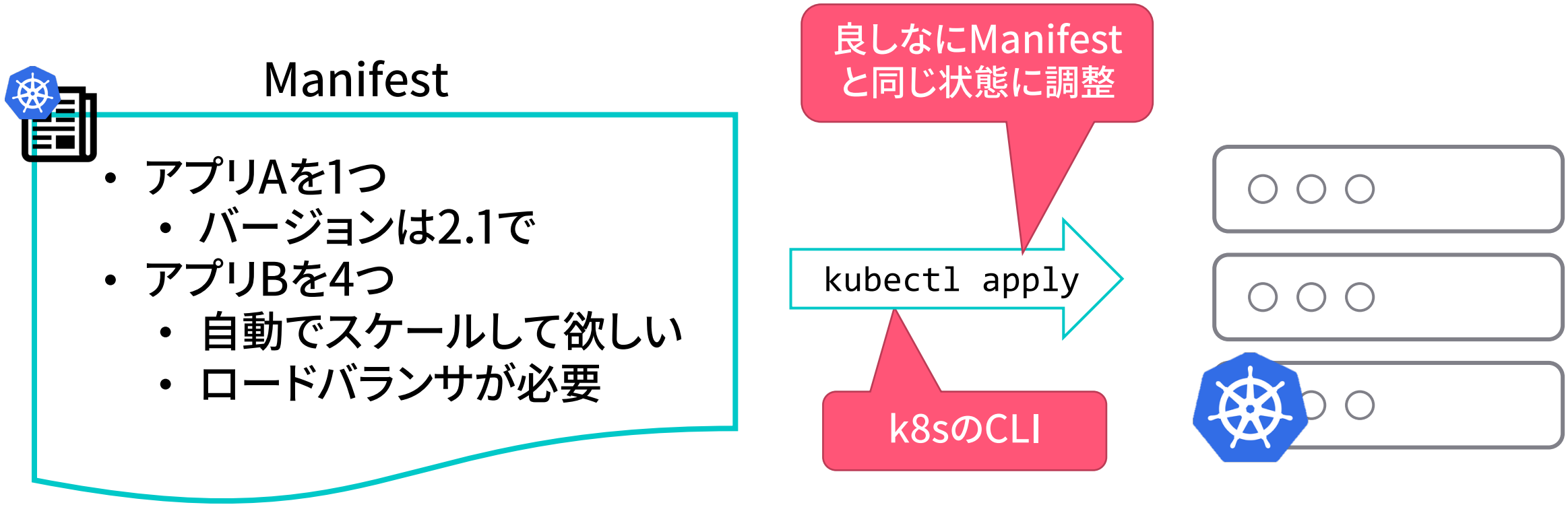
Manifest

- アプリAを1つ
  - バージョンは2.1で
- アプリBを4つ
  - 自動でスケールして欲しい
  - ロードバランサが必要

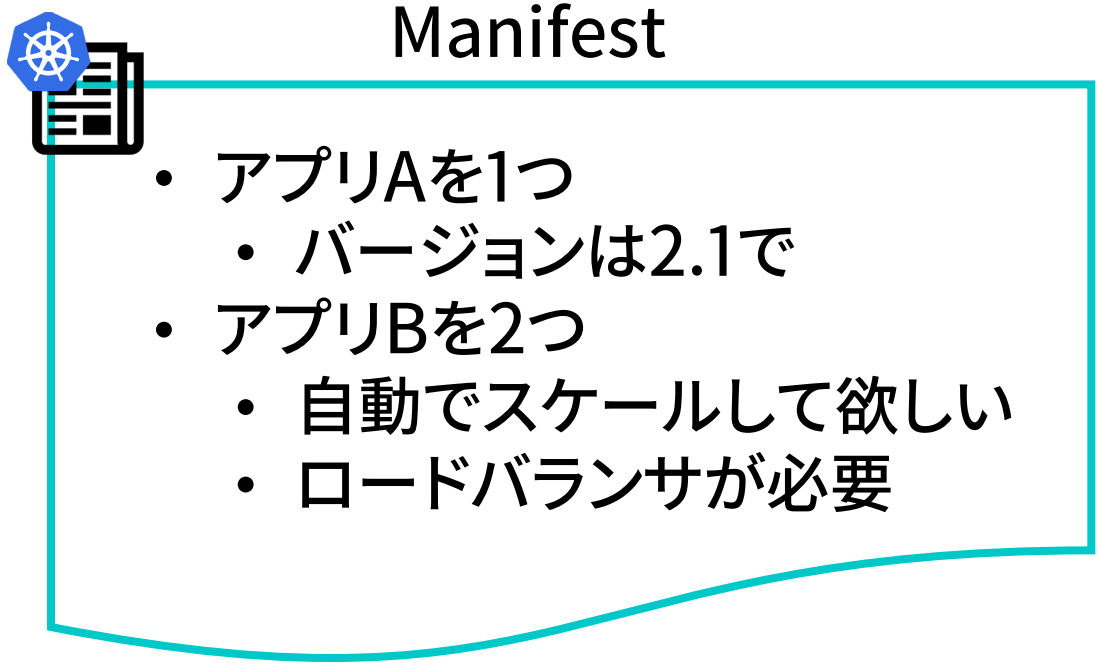
- k8sの特徴:
  リソースをManifestと呼ばれるファイルで管理可能

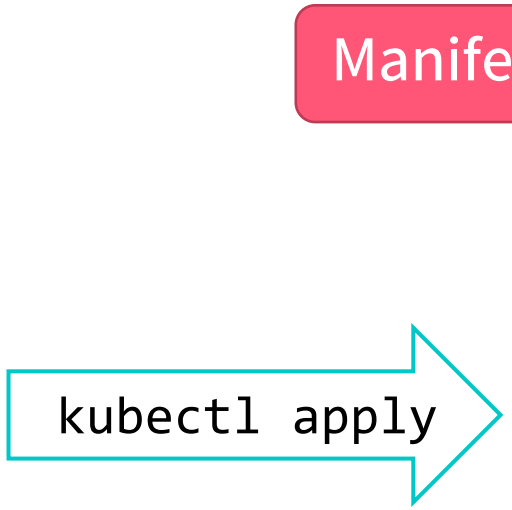


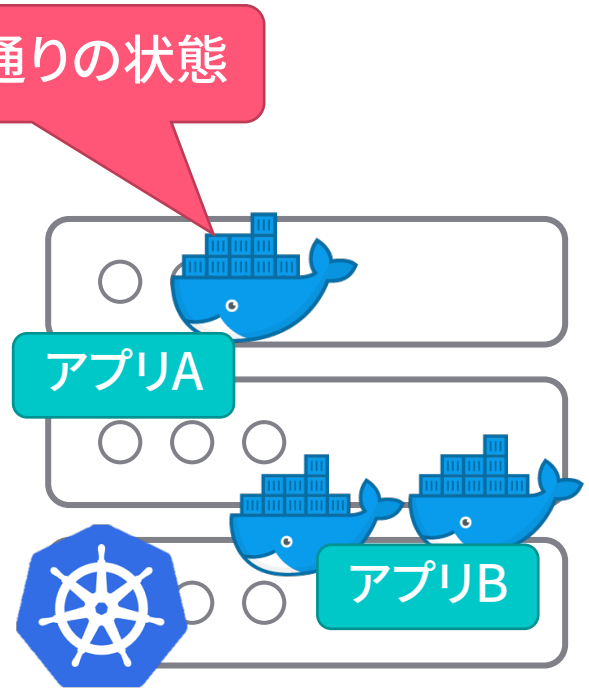
- k8sの特徴:
  リソースをManifestと呼ばれるファイルで管理可能



- k8sの特徴:
  リソースをManifestと呼ばれるファイルで管理可能







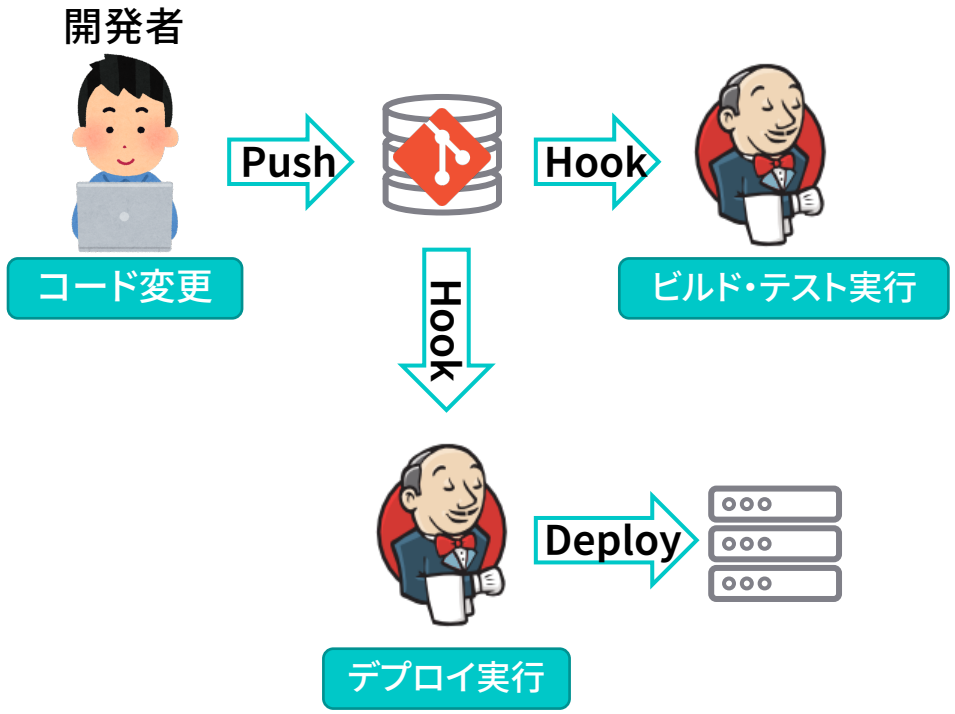
CI/CDの目的は？

- 変更があった際に必要な処理の自動化

CIとCDは似ているが、役割が少し異なる

- CI (Continuous Integration)
  - 自動的なテスト実行・ビルド成果物(DockerImage、npmパッケージ等)アップロード
- CD (Continuous Delivery/Deployment)
  - 自動的なアプリデプロイ

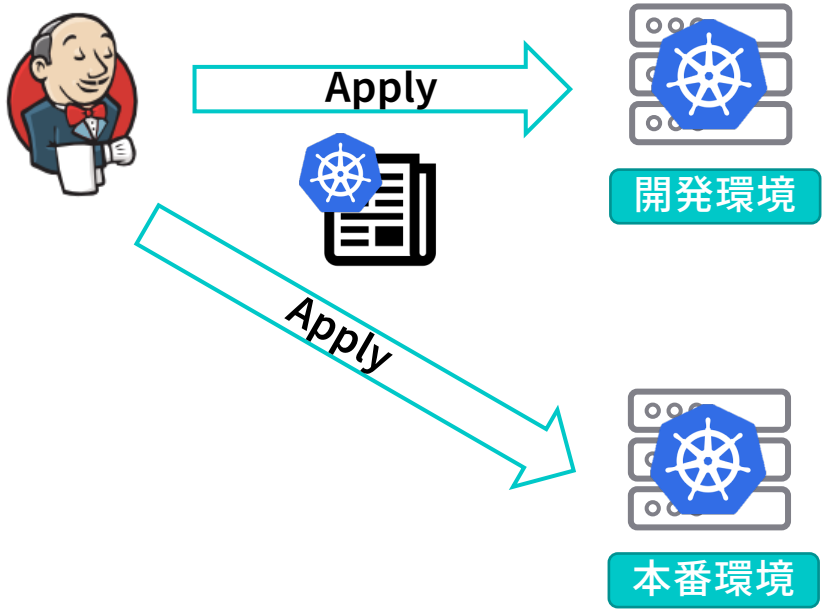
※2つまとめてCIと呼ぶことも多い



+ CI/CD

「CI/CDでkubectlすればいいじゃん💡」

- 👎サービスの世代管理が困難 🔢
- 👎意図したデプロイ状況になっているか分からない 🌀
- 👎属人化したHackが発生 🔪
- 👎パワフルな権限を持つCI 💪



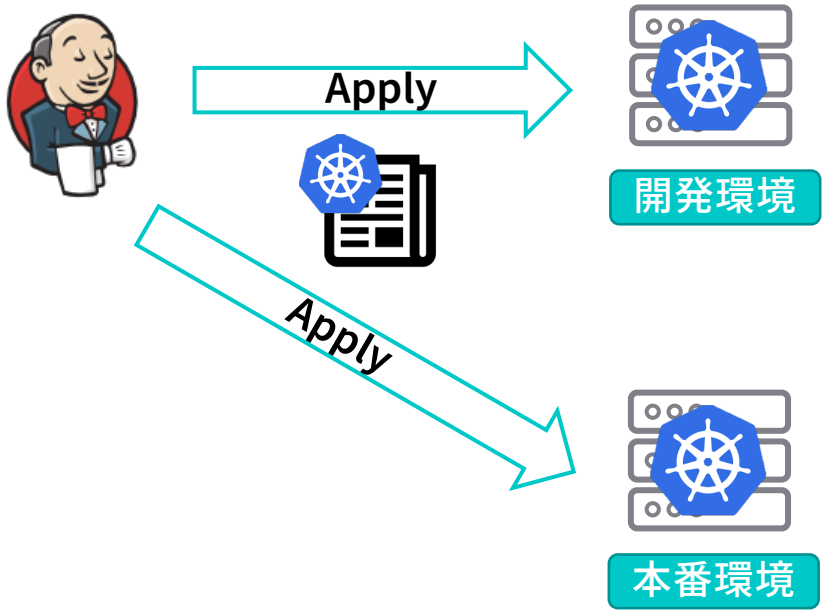
※GitOpsと対比して、従来のCI/CDをCIOpsと呼ぶ

- Push型

「CI/CDでkubectlすればいいじゃん💡」

変更をデプロイするだけ
状態を管理できるわけではない

- 👎サービスの世代管理が困難 🔢
- 👎意図したデプロイ状況になっているか分からない 🌀
- 👎属人化したHackが発生 🔪
- 👎パワフルな権限を持つCI 💪



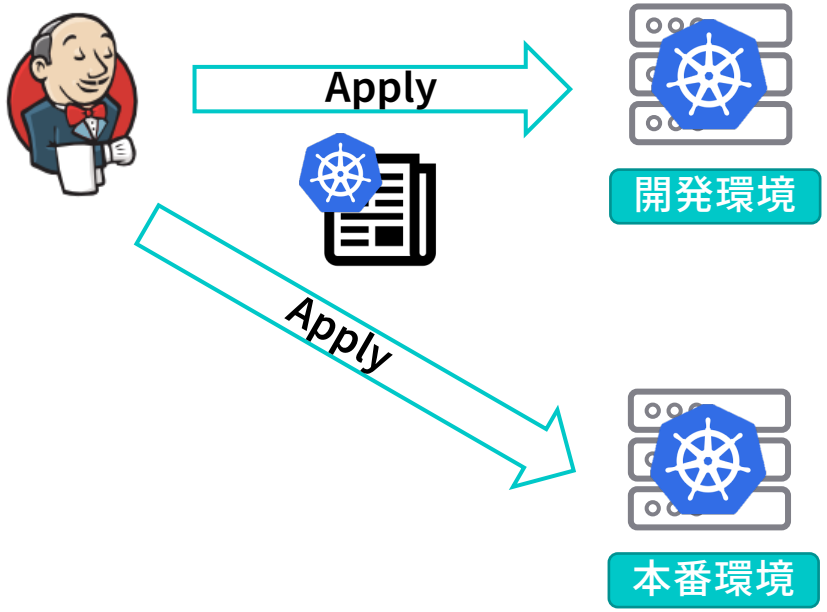
※GitOpsと対比して、従来のCI/CDをCIOpsと呼ぶ

- Push型

「CI/CDでkubectlすればいいじゃん」

- 👎 サービスの世代管理が困難 🔢
- 👎 意図したデプロイ状況になっているか分からない 🌀
- 👎 属人化したHackが発生 🔪
- 👎 パワフルな権限を持つCI 💪

誰も触れない秘伝のタレ

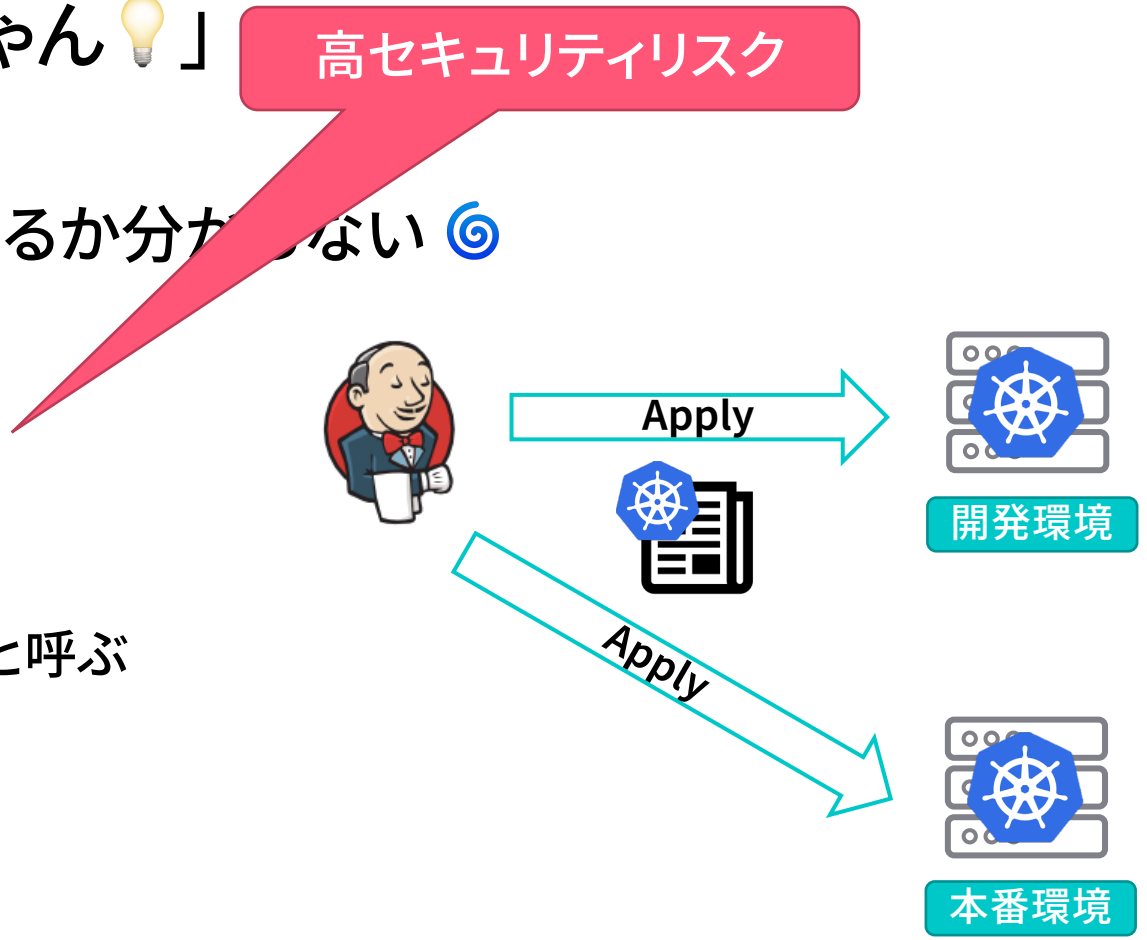


※GitOpsと対比して、従来のCI/CDをCIOpsと呼ぶ

- Push型

「CI/CDでkubectlすればいいじゃん💡」

- 👎サービスの世代管理が困難 🔢
- 👎意図したデプロイ状況になっているか分からない 🌀
- 👎属人化したHackが発生 🔪
- 👎パワフルな権限を持つCI 💪



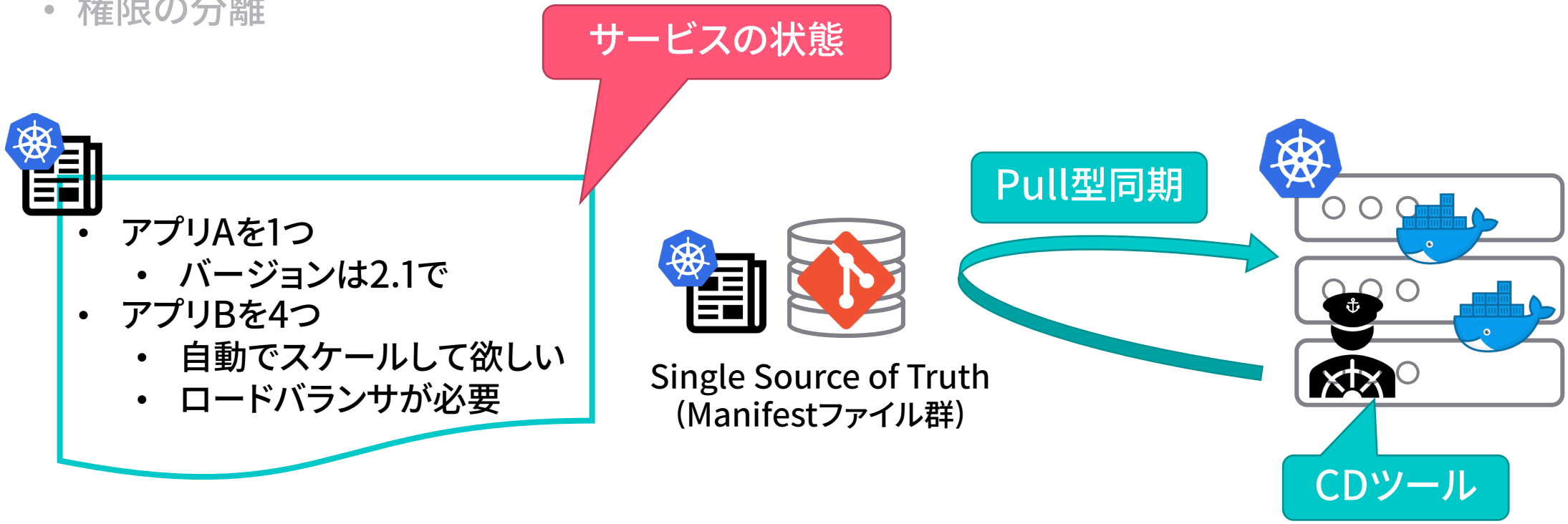
※GitOpsと対比して、従来のCI/CDをCIOpsと呼ぶ

- Push型

SAKURA internet

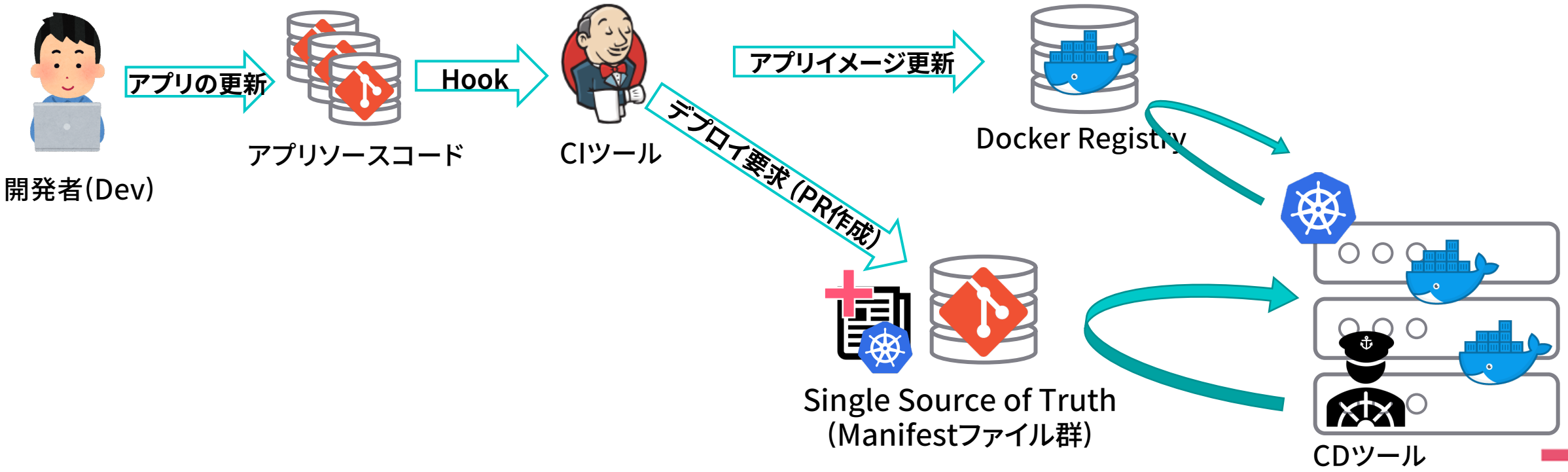
k8sを前提としたGitを中心に据えたCI/CD手法

- Single Source of Truthとして、サービスの状態を1つのGitリポジトリで持つ
- 関心 (CI/CD) の分離
  - 権限の分離



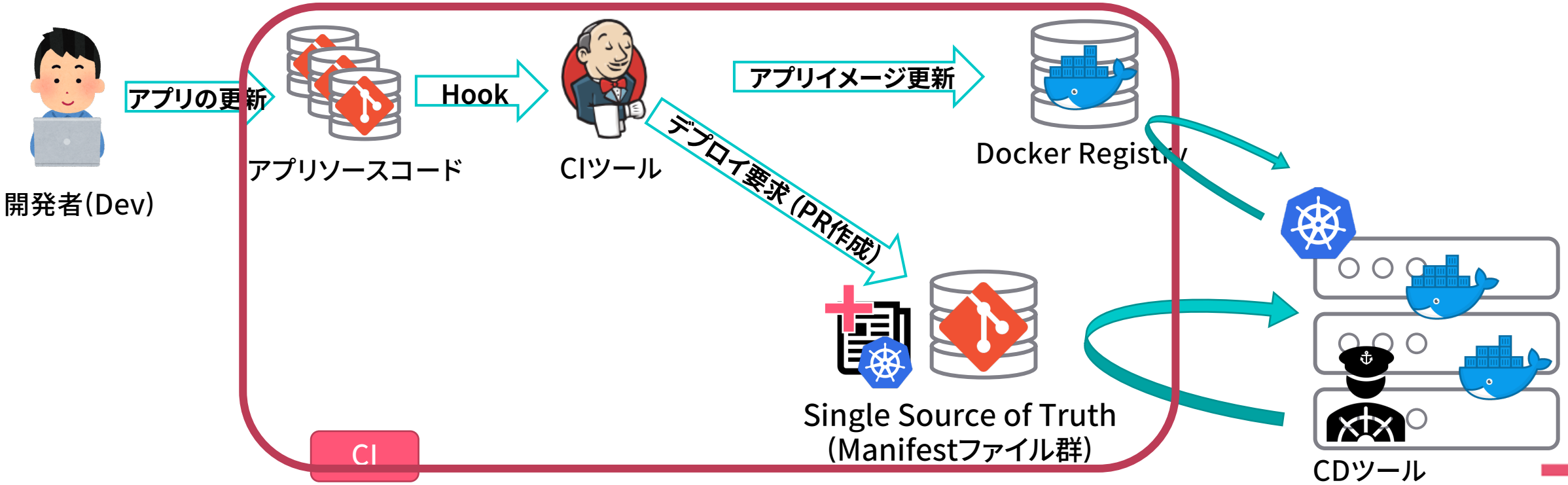
k8sを前提としたGitを中心に据えたCI/CD手法

- Single Source of Truthとして、サービスの状態を1つのGitリポジトリで持つ
- 関心 (CI/CD) の分離
  - 権限の分離



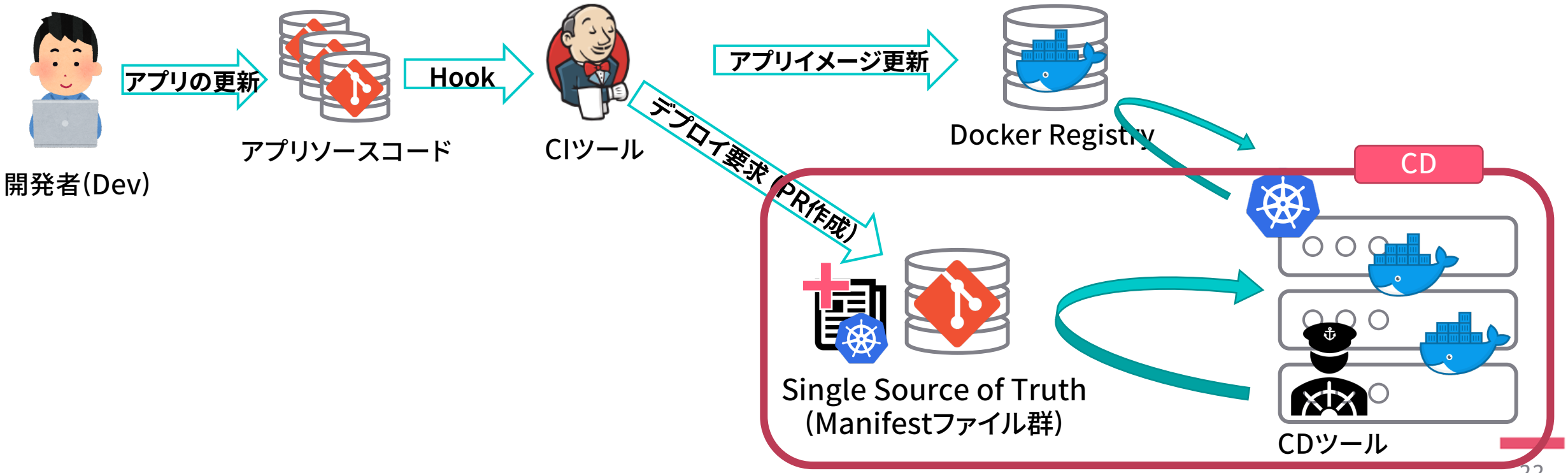
k8sを前提としたGitを中心に据えたCI/CD手法

- Single Source of Truthとして、サービスの状態を1つのGitリポジトリで持つ
- 関心 (CI/CD) の分離
  - 権限の分離



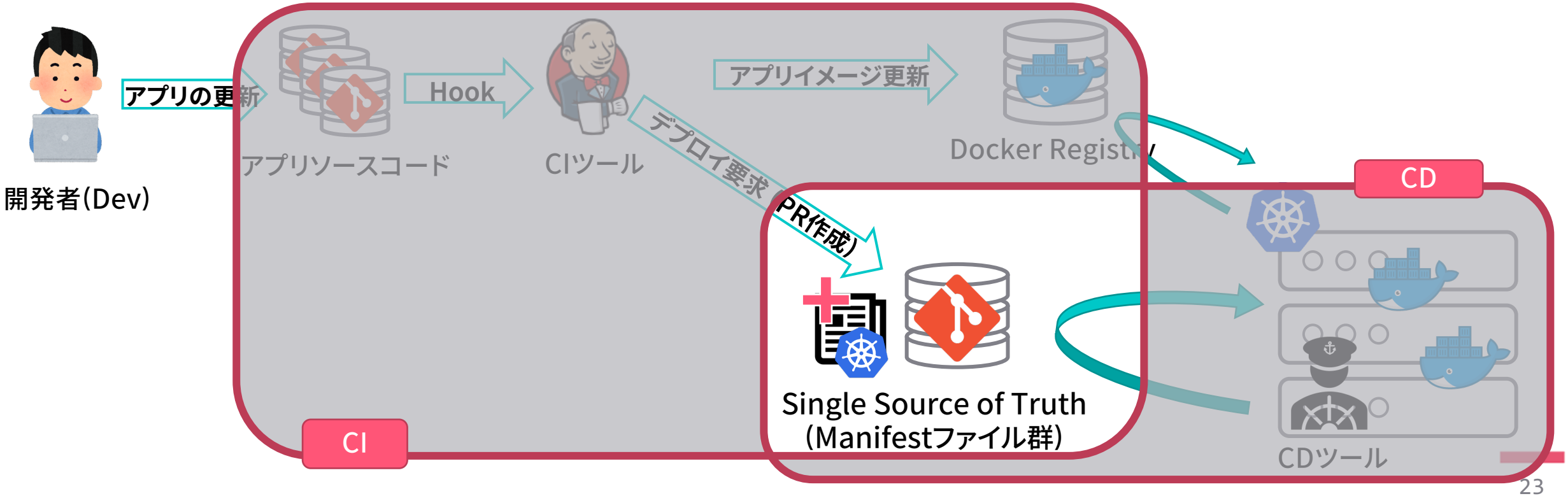
k8sを前提としたGitを中心に据えたCI/CD手法

- Single Source of Truthとして、サービスの状態を1つのGitリポジトリで持つ
- 関心 (CI/CD) の分離
  - 権限の分離



k8sを前提としたGitを中心に据えたCI/CD手法

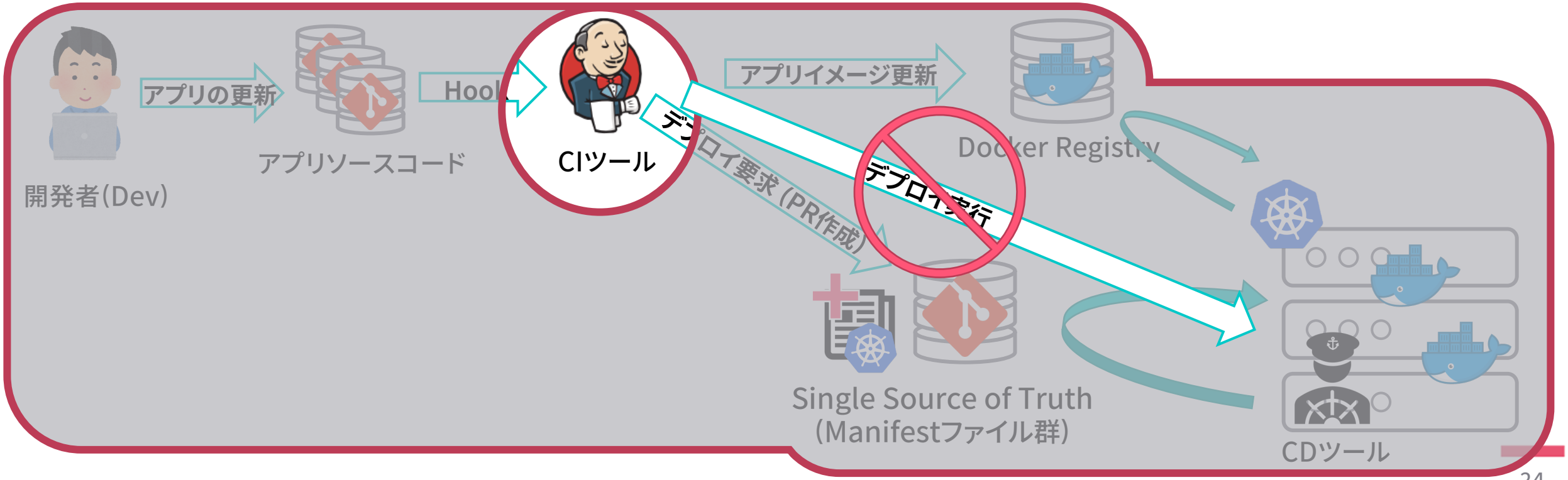
- Single Source of Truthとして、サービスの状態を1つのGitリポジトリで持つ
- 関心 (CI/CD) の分離
  - 権限の分離



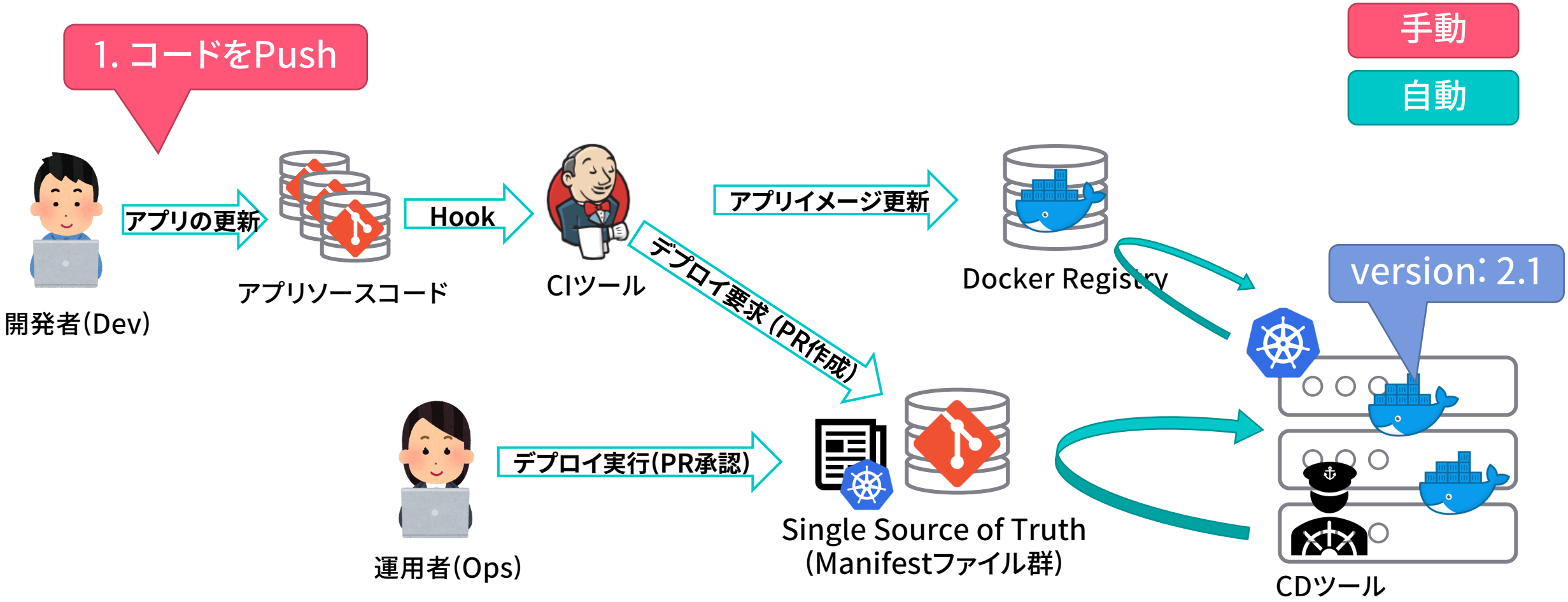
SAKURA internet

k8sを前提としたGitを中心に据えたCI/CD手法

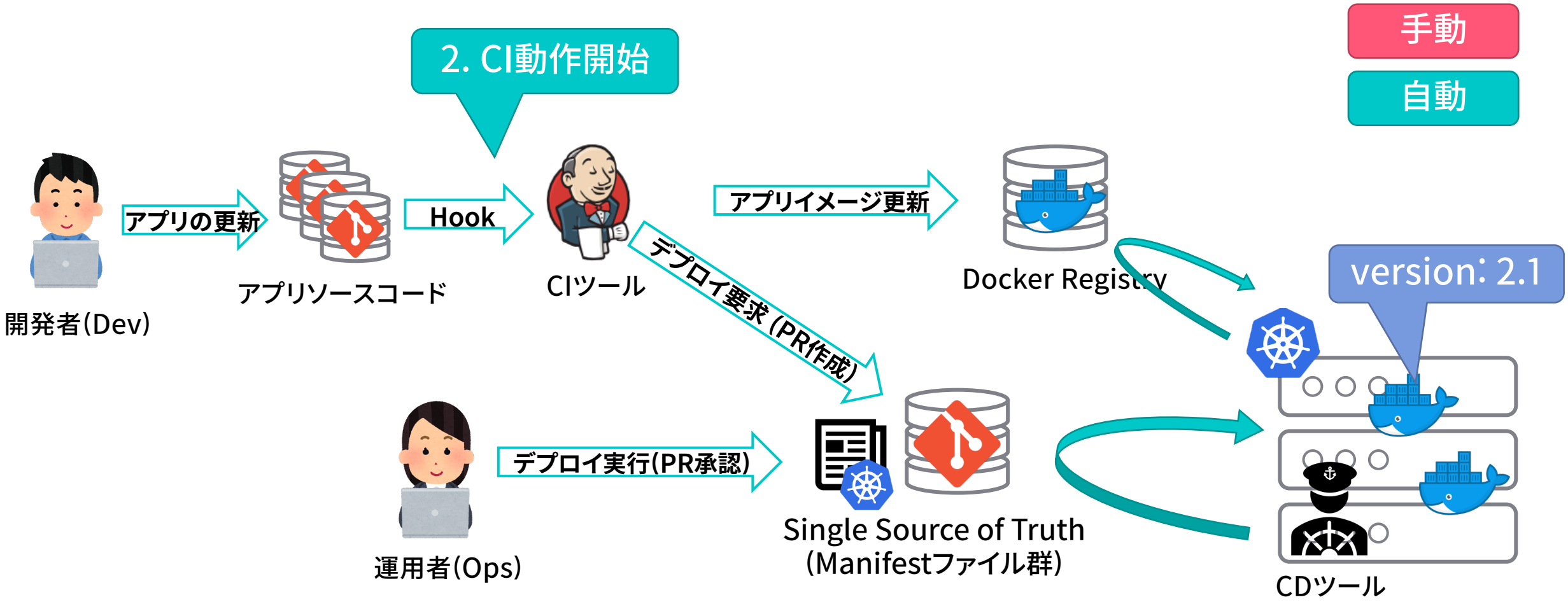
- Single Source of Truthとして、サービスの状態を1つのGitリポジトリで持つ
- 関心 (CI/CD) の分離
  - 権限の分離



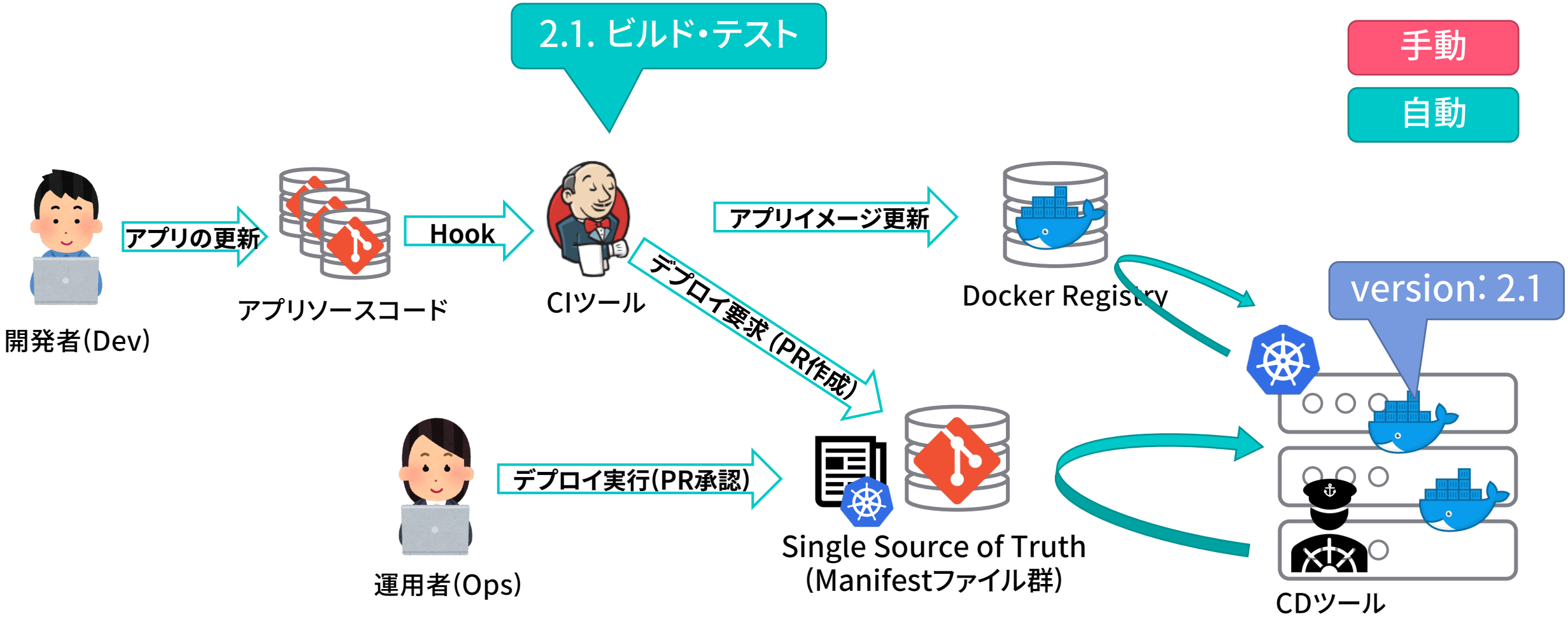
GitOpsの全体像


手動
自動
1. コードをPush
アプリの更新
Hook
アプリイメージ更新
デプロイ要求（PR作成）
version: 2.1
開発者(Dev)
アプリソースコード
CIツール
Docker Registry
デプロイ実行(PR承認)
運用者(Ops)
Single Source of Truth
(Manifestファイル群)
CDツール

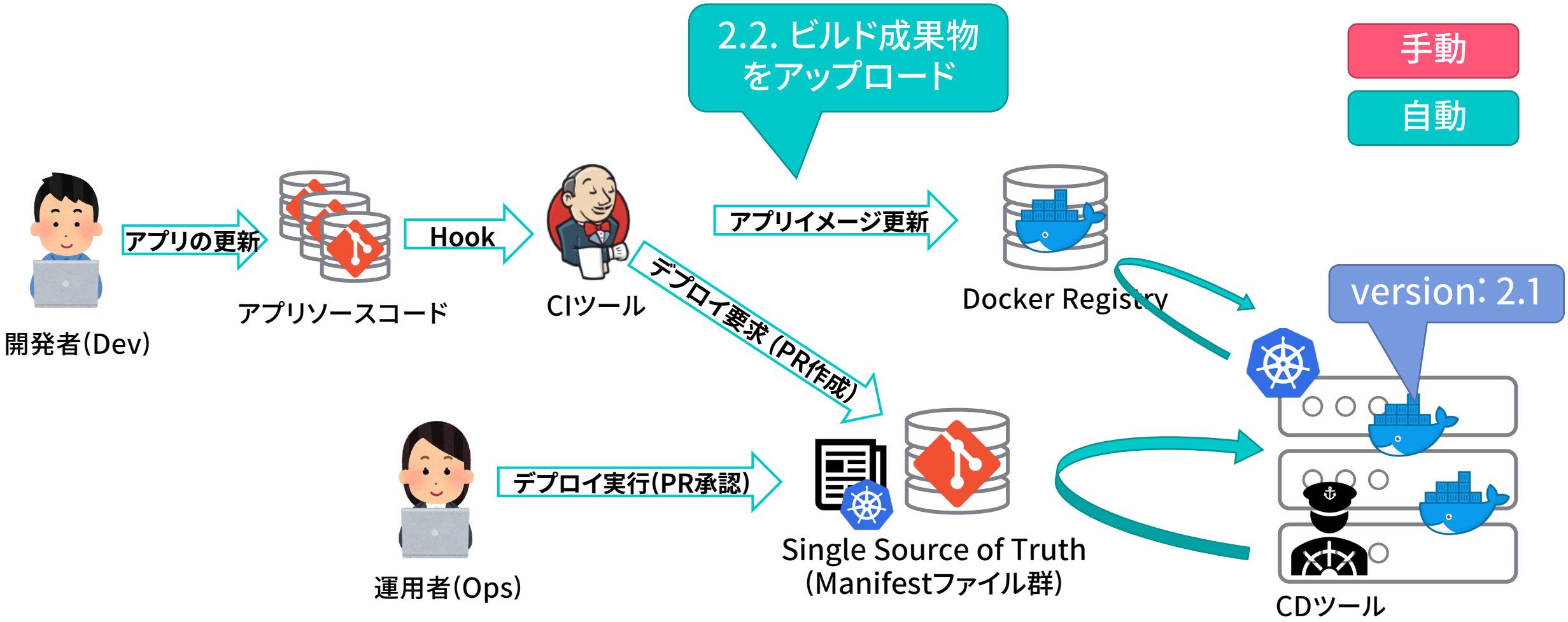
GitOpsの全体像


手動
自動
2. CI動作開始
開発者(Dev)
アプリの更新
アプリソースコード
Hook
CIツール
アプリイメージ更新
Docker Registry
デプロイ要求(PR作成)
version: 2.1
運用者(Ops)
デプロイ実行(PR承認)
Single Source of Truth
(Manifestファイル群)
CDツール

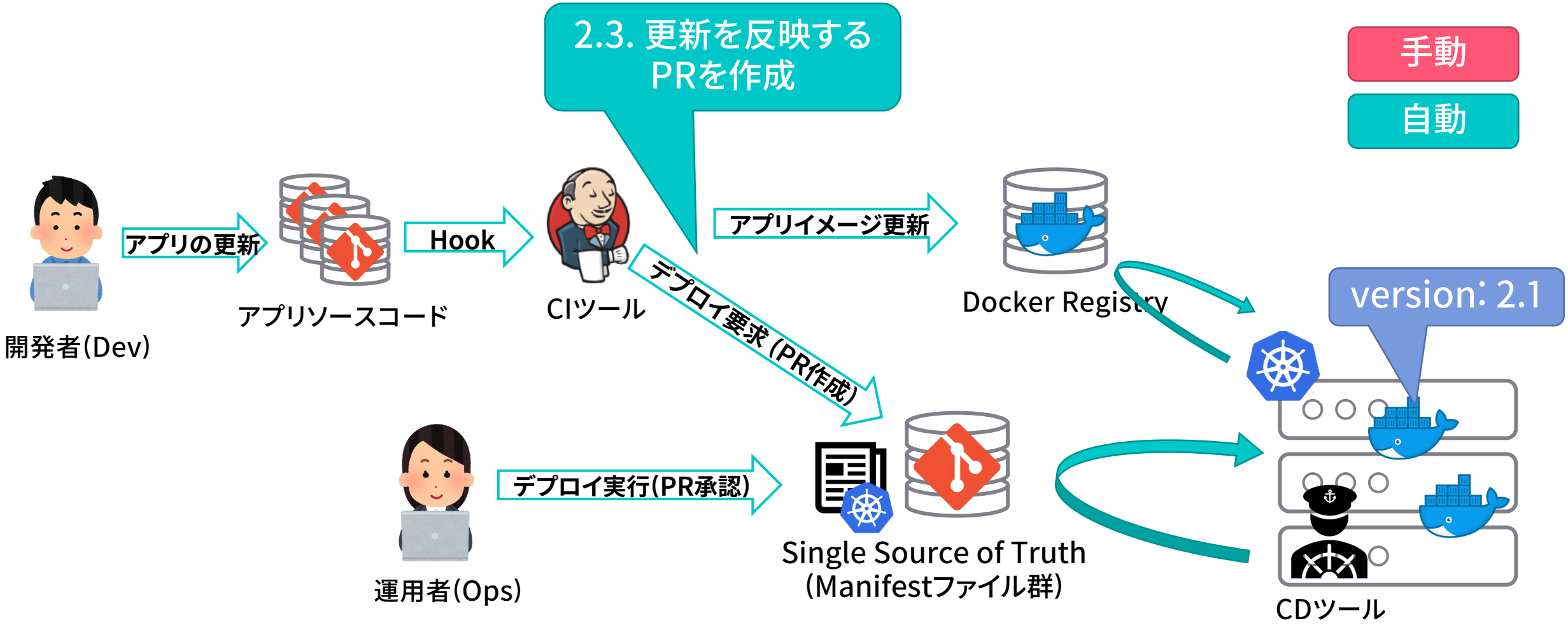
GitOpsの全体像


2.1. ビルド・テスト
手動
自動
アプリの更新
Hook
アプリイメージ更新
デプロイ要求（PR作成）
version: 2.1
開発者(Dev)
アプリソースコード
CIツール
Docker Registry
デプロイ実行(PR承認)
運用者(Ops)
Single Source of Truth
(Manifestファイル群)
CDツール

GitOpsの全体像


2.2. ビルド成果物をアップロード
手動
自動
アプリの更新
Hook
アプリイメージ更新
デプロイ要求（PR作成）
version: 2.1
開発者(Dev)
アプリソースコード
CIツール
Docker Registry
デプロイ実行(PR承認)
運用者(Ops)
Single Source of Truth
(Manifestファイル群)
CDツール

GitOpsの全体像


2.3. 更新を反映する
PRを作成
手動
自動
アプリの更新
Hook
アプリイメージ更新
デプロイ要求（PR作成）
version: 2.1
開発者(Dev)
アプリソースコード
CIツール
Docker Registry
デプロイ実行(PR承認)
運用者(Ops)
Single Source of Truth
(Manifestファイル群)
CDツール

デプロイ要求PullRequestの例

SAKURA internet

更新前

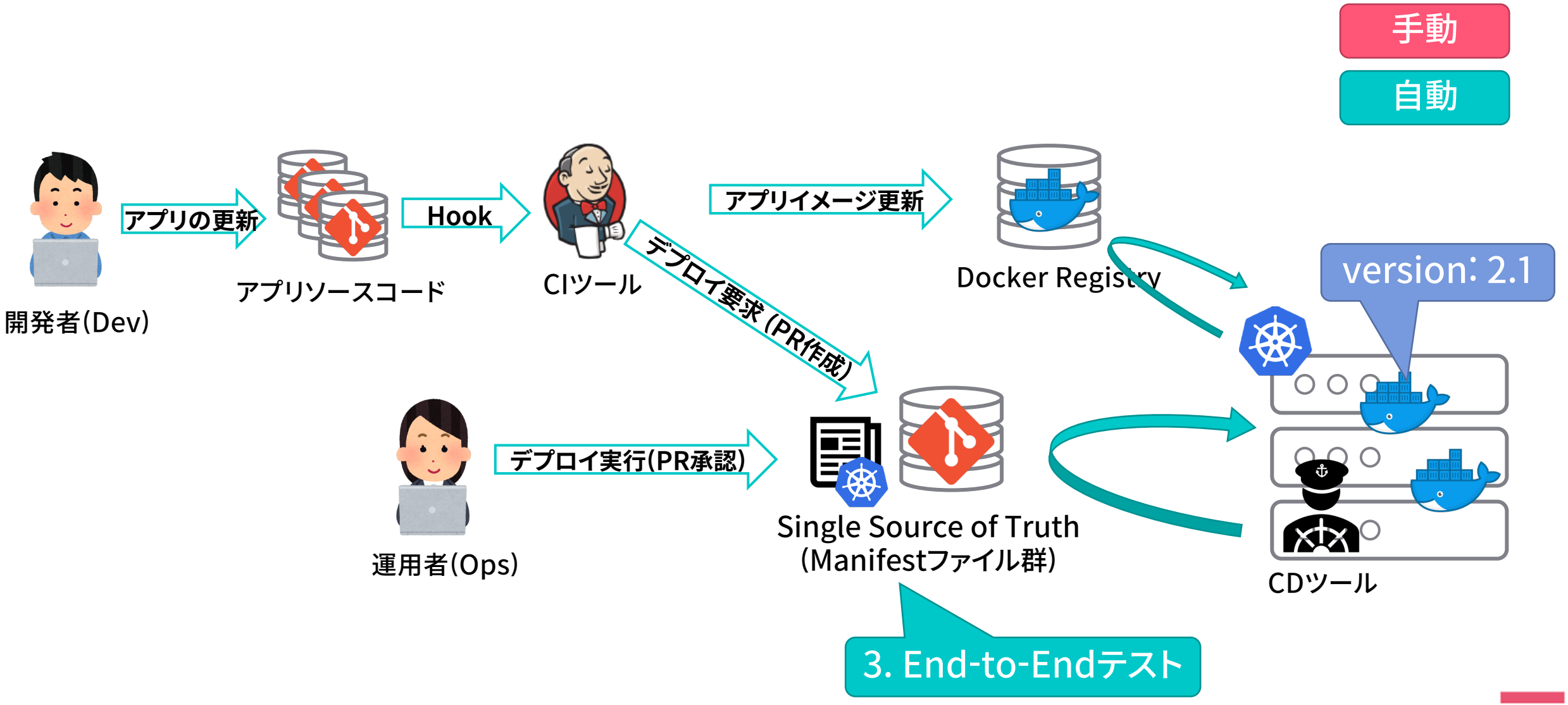
- アプリAを1つ
  - バージョンは2.1で
- アプリBを4つ
  - 自動でスケールして欲しい
  - ロードバランサが必要

更新後

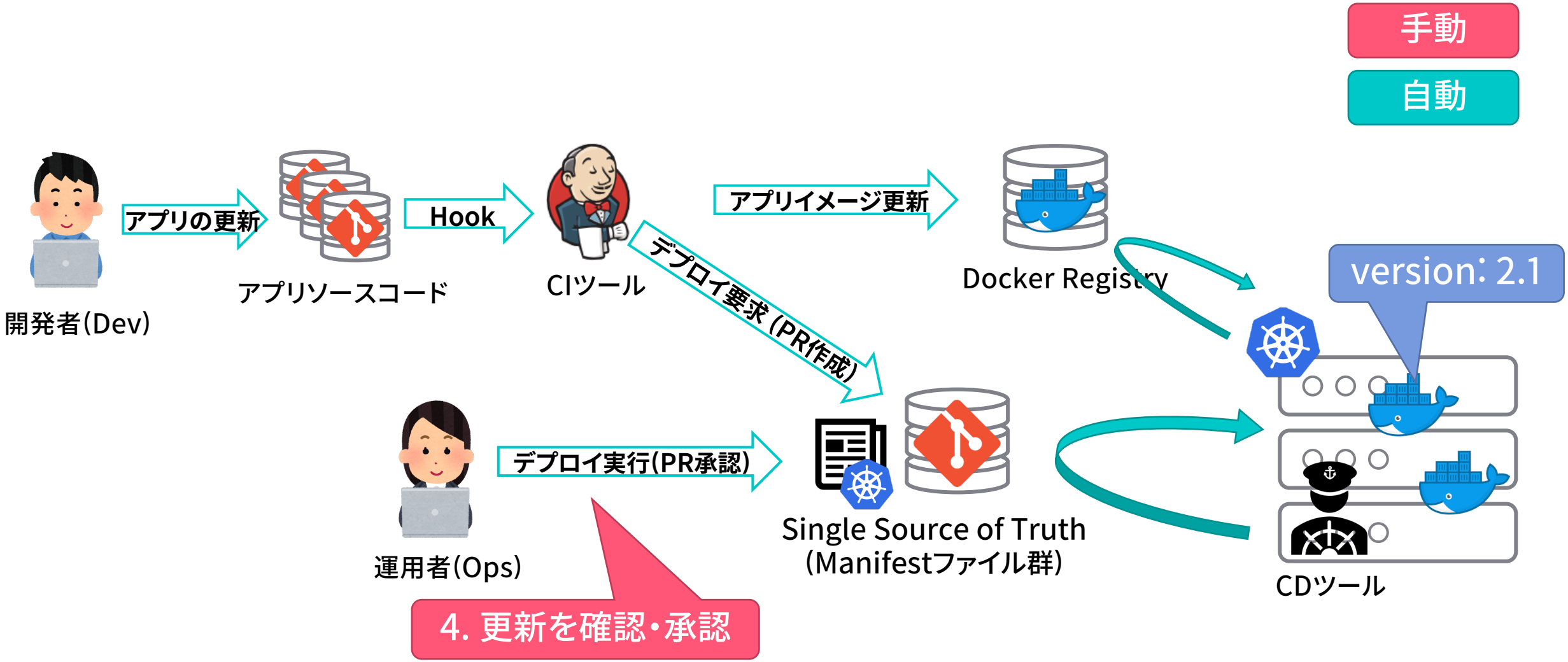
- アプリAを1つ
  - バージョンは2.2で
- アプリBを4つ
  - 自動でスケールして欲しい
  - ロードバランサが必要

※実際にはManifestに含まれるDockerImageタグの更新

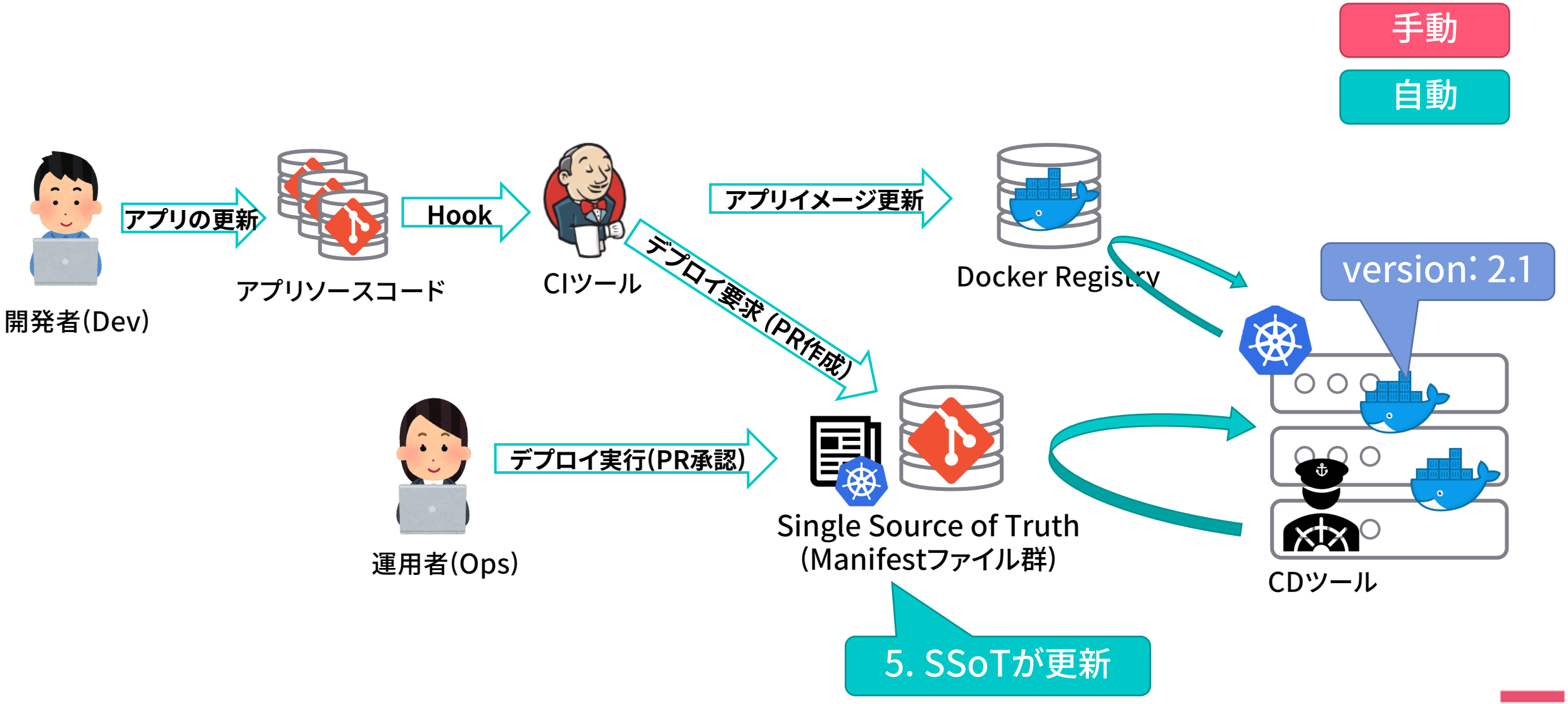
GitOpsの全体像


手動
自動
アプリの更新
Hook
アプリイメージ更新
デプロイ要求（PR作成）
開発者(Dev)
アプリソースコード
CIツール
Docker Registry
version: 2.1
デプロイ実行(PR承認)
運用者(Ops)
Single Source of Truth
(Manifestファイル群)
CDツール
3. End-to-Endテスト

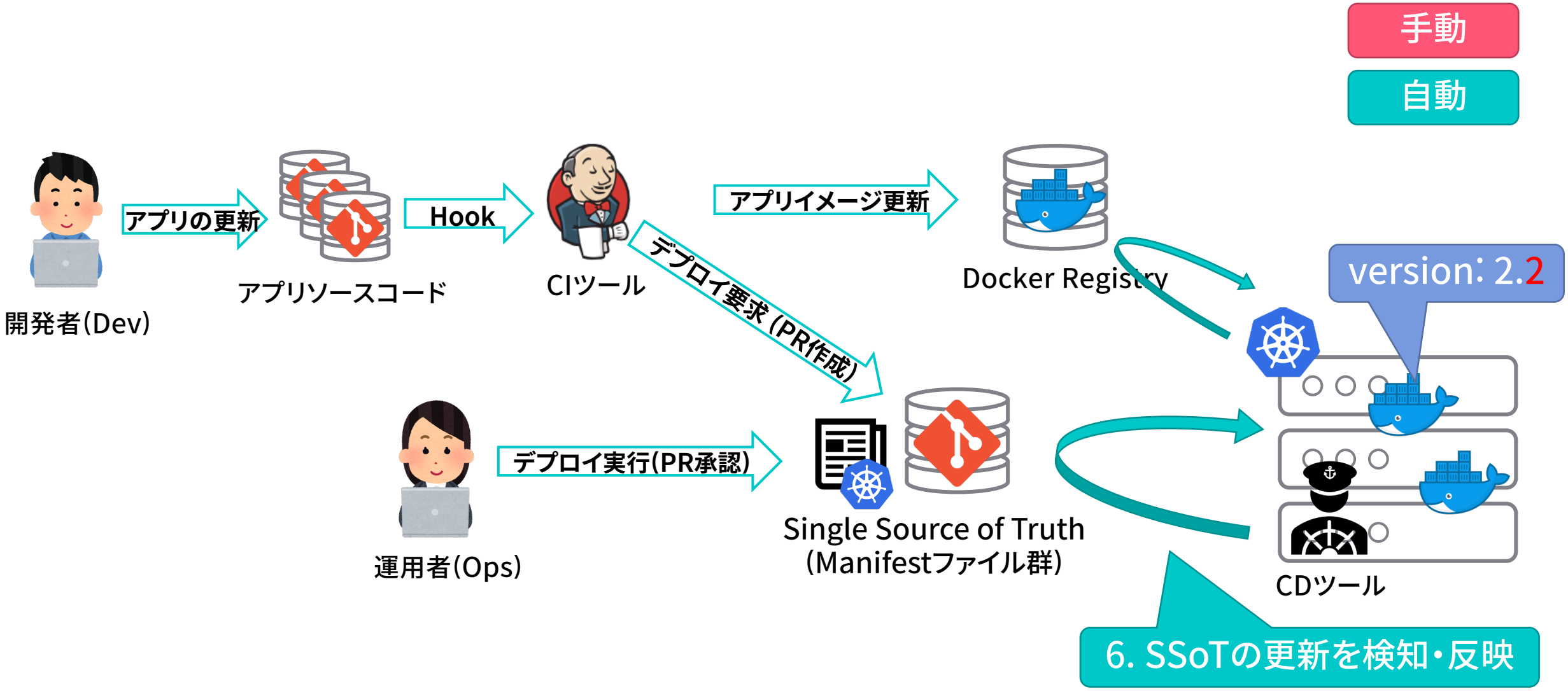
GitOpsの全体像


手動
自動
開発者(Dev)
アプリの更新
アプリソースコード
Hook
CIツール
アプリイメージ更新
Docker Registry
デプロイ要求(PR作成)
version: 2.1
運用者(Ops)
デプロイ実行(PR承認)
Single Source of Truth
(Manifestファイル群)
CDツール
4. 更新を確認・承認

GitOpsの全体像


手動
自動
開発者(Dev)
アプリの更新
アプリソースコード
Hook
CIツール
アプリイメージ更新
Docker Registry
デプロイ要求(PR作成)
version: 2.1
運用者(Ops)
デプロイ実行(PR承認)
Single Source of Truth
(Manifestファイル群)
CDツール
5. SSoTが更新

GitOpsの全体像


手動
自動
開発者(Dev)
アプリの更新
アプリソースコード
Hook
CIツール
アプリイメージ更新
Docker Registry
デプロイ要求(PR作成)
version: 2.2
運用者(Ops)
デプロイ実行(PR承認)
Single Source of Truth
(Manifestファイル群)
CDツール
6. SSoTの更新を検知・反映

SAKURA internet

Single Source of Truthとして、サービスの状態を1つのGitリポジトリで持つ

- コードベースでリソース管理できるk8sと相性👍⭕

CIとCDの分離

- CI
  - テストやビルドに注力
  - Push型
  - 実行環境への権限は不要
- CD
  - デプロイに注力
  - Pull型
  - 自身の実行環境への権限のみ持つ

YAML形式

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: debugkit
  name: debugkit
spec:
  replicas: 1
  selector:
    matchLabels:
      app: debugkit
  template:
    metadata:
      labels:
        app: debugkit
    spec:
      containers:
      - image: amaya382/k8s-debugkit
        name: k8s-debugkit
```

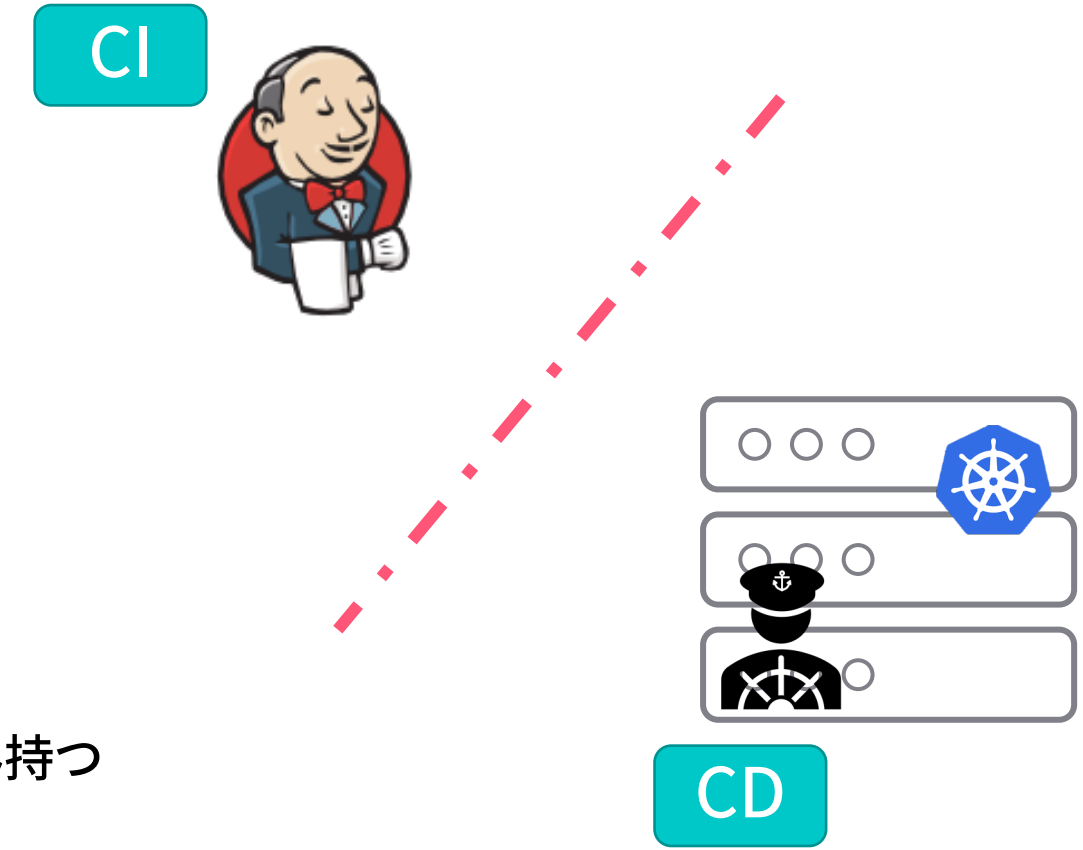
SAKURA internet

Single Source of Truthとして、サービスの状態を1つのGitリポジトリで持つ

- コードベースでリソース管理できるk8sと相性👍

CIとCDの分離

- CI
  - テストやビルドに注力
  - Push型
  - 実行環境への権限は不要
- CD
  - デプロイに注力
  - Pull型
  - 自身の実行環境への権限のみ持つ



👉 k8sを前提としたGitを中心に据えたCI/CD手法

特徴

- Single Source of Truthとして、デプロイ状態を1つのGitリポジトリで管理
- 関心 (CI/CD) の分離

得られるもの

- ストレスフリーな爆速デプロイ
  - アジャイル
- 全ての設定をコード化、Gitで管理
  - 属人化したHackを防ぐ
  - Gitリポジトリの状態 == デプロイの状態
  - 開発だけでなく、運用作業もGitの操作で統一
- 適切な関心の分離
  - 権限の分離
  - セキュリティリスクの抑制