#devaxconnect

aws DevAx connect

Kubernetes で実現する GitOps

DevAx::connect シーズン 3 rethink CI/CD

2022/6/23

Yasuhiro Horiuchi
Solutions Architect

自己紹介

#devaxconnect

堀内 保大 (Yasuhiro Horiuchi)

horiuyas@amazon.co.jp

@ka_shino_ki

- AWS Japan G.K. ソリューションアーキテクト
- 主に Digital 関連企業様を担当

バックグラウンド

- 前職：大手SIerで多岐にわたるシステムへの性能関連の技術支援
  （性能試験、監視、チューニング、非機能設計 etc）

好きなAWSサービス

コンテナ関連が好きです！！

- Amazon Elastic Kubernetes Service (EKS)
- AWS Fargate
- Amazon Managed Service for Prometheus (AMP)

趣味

- スノーボード、旅行

aws certified DevOps Engineer Professional

aws certified Solutions Architect Professional

CERTIFIED kubernetes ADMINISTRATOR

aws DevAx connect

GitOps ってなんですかね

- Git で管理されていれば GitOps ?
- Git を契機に自動でデプロイを実装しているから GitOps やれてる ?
- CI/CDと何が違うの ?

今日はこの点クリアになってもらえればうれしいです！！

Agenda

- CI/CD 振り返り
- GitOps とは？
- Kubernetes で実現する GitOps
- まとめ

#devaxconnect

CI/CD振り返り

aws DevAx connect

#devaxconnect

第1回 雰囲気でモダン開発手法の実践をしている人のためのCI/CD再入門

野村 侑志
アマゾンウェブサービスジャパン合同会社
ソリューションアーキテクト

CI/CD は組織で使っているけれど、なぜ CI/CD を使わないといけないかや具体的にどんないいことがあるのかが曖昧になっていませんか？本セッションでは、歴史的な背景やどのような意図で CI/CD を行うべきなのかという根本的な部分を見直すことによって、自分の開発環境の全体像を理解したり見直したりするきっかけにしていただくことを目的にしています。

CI/CDとは

#devaxconnect

継続的インテグレーション
Continuous Integration（CI）

＋

継続的デリバリー/デプロイ
Continuous Delivery/Deployment（CD）

aws DevAx

リリースプロセスのステージ

#devaxconnect

Source	Build	Test	Production
• ソースコードのチェックイン • コードのピアレビュー	• コードのコンパイル • ユニットテスト • スタイルチェッカー • コンテナイメージ、関数デプロイパッケージの作成	• 周辺システムとの統合テスト • 負荷テスト • UIテスト • セキュリティテスト	• 本番環境へのデプロイ • エラーを素早く検知するための本番環境のモニタリング

aws DevAx

リリースプロセスのステージ

#devaxconnect

Source → Build → Test → Production

CI：継続的インテグレーション

CD：継続的デリバリー

別のCD：継続的デプロイ

aws DevAx

#devaxconnect

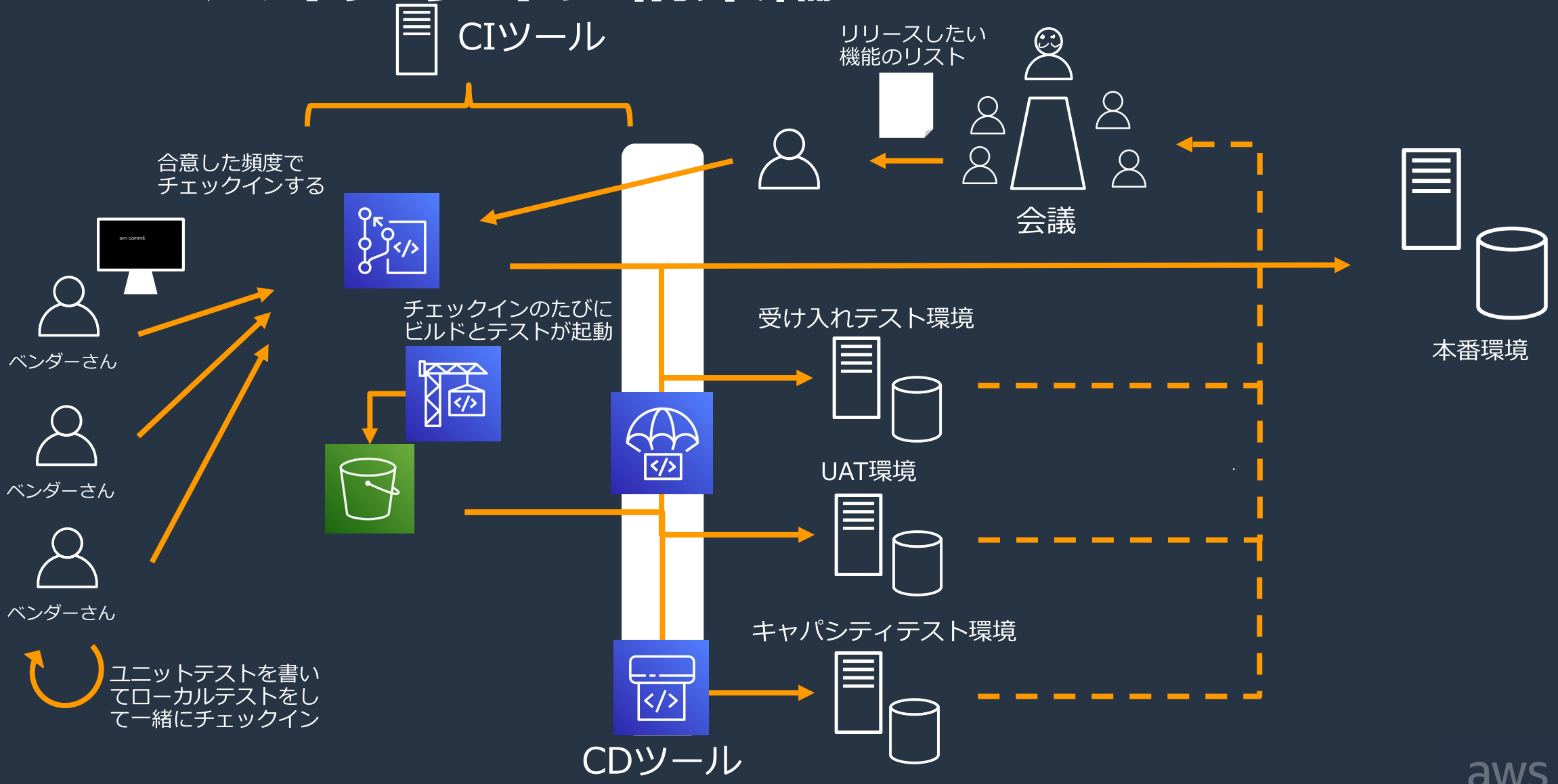
第2回 適材適所で選ぶ CI/CD パイプライン構築サービス&ツール

松本 雅博
アマゾンウェブサービスジャパン合同会社
シニアソリューションアーキテクト

AWS の CI/CD パイプラインサービスというと、 まずは CodePipeline を思い浮かべられる方が多いのではないでしょうか？現在、AWS には CDK Pipeline, AWS Copilot, AWS SAM Pipeline などのパイプライン構築に活用できるツールや、AWS Amplify Console, AWS Proton など CI/CD パイプラインが組み込まれたサービスがあります。また、3rd Party のサービス、ツールをご利用いただくこともできます。このセッションでは CI/CD に関する多彩なサービス、ツールをどのような場面で活用するのかについてお話しします。

CD：パイプライン構築編

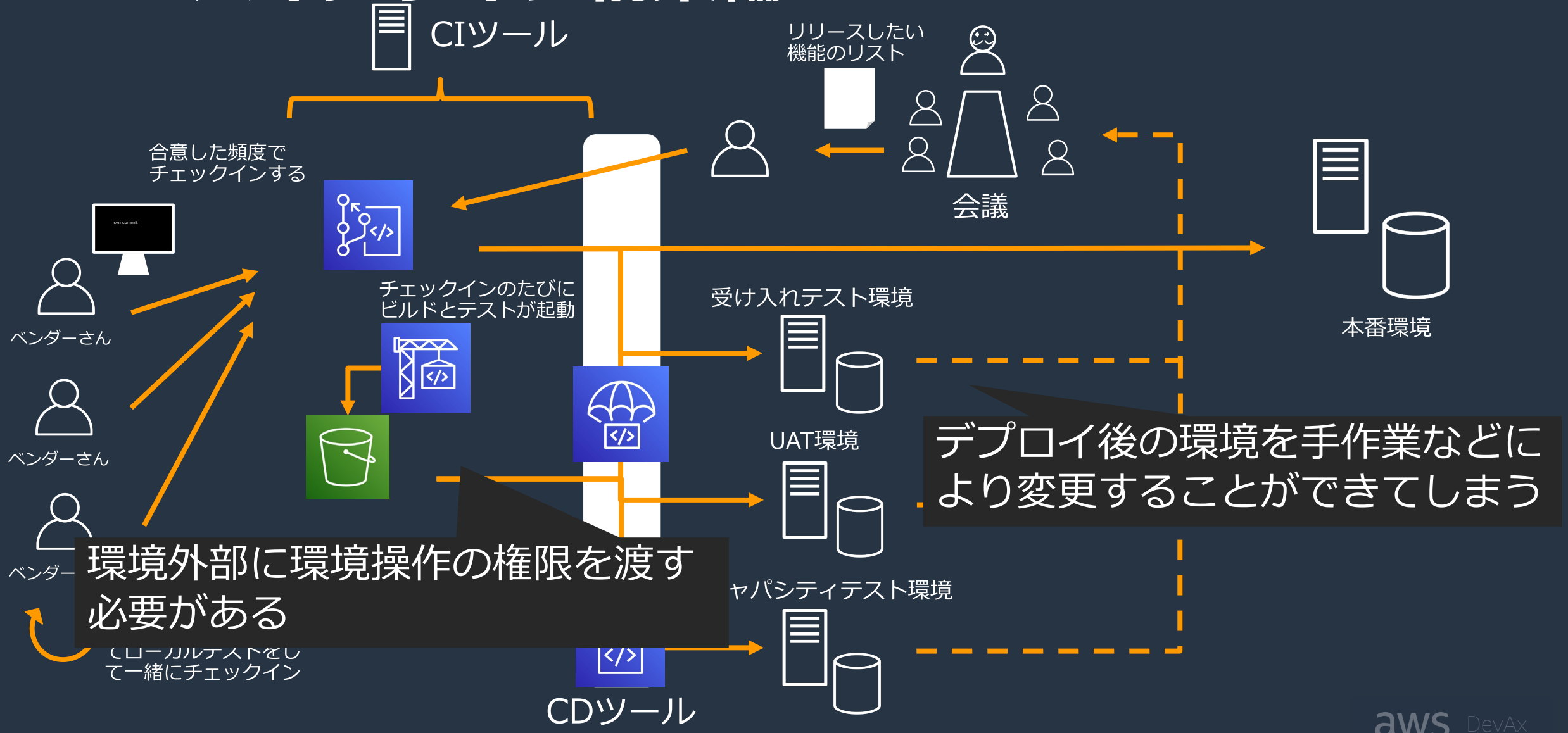
#devaxconnect



aws DevAx connect

CD：パイプライン構築編

#devaxconnect



デプロイ後の環境を手作業などにより変更することができてしまう

環境外部に環境操作の権限を渡す必要がある

GitOps とは？

GitOps とは？

- Weaveworks社が2017年に提唱
  - https://www.weave.works/blog/gitops-operations-by-pull-request
- GitOps は Git をデプロイパイプラインの中心に据えた IaC のデプロイの考え方
  - 開発者は使い慣れたツールを使用してプルリクエストを作成し、Kubernetes へのアプリケーションのデプロイと運用タスクの両方を高速化および簡素化します
- Git を中心に CI と CD を分離

GitOps の原則

システム全体が宣言的に記述されていること

望ましいシステム状態が Git でバージョン管理されている

変更を自動的に適用する

エージェントは、正確性を保証し、相違を警告する

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

GitOps の原則

#devaxconnect

宣言型ツールを使用することで、システムのあるべき状態をGitで管理可能になる

システム全体が宣言的に記述されていること

望ましいシステム状態が Git でバージョン管理されている

変更を自動的に適用する

エージェントは、正確性を保証し、相違を警告する

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

aws DevAx connect

宣言的な構成管理の自動化

- ここでいう「宣言的」とは？
- → 命令的にコマンドでインフラ構成を指示するのではなく、構成のあるべき状態を設定ファイルで定義すること

命令的なインフラ構築

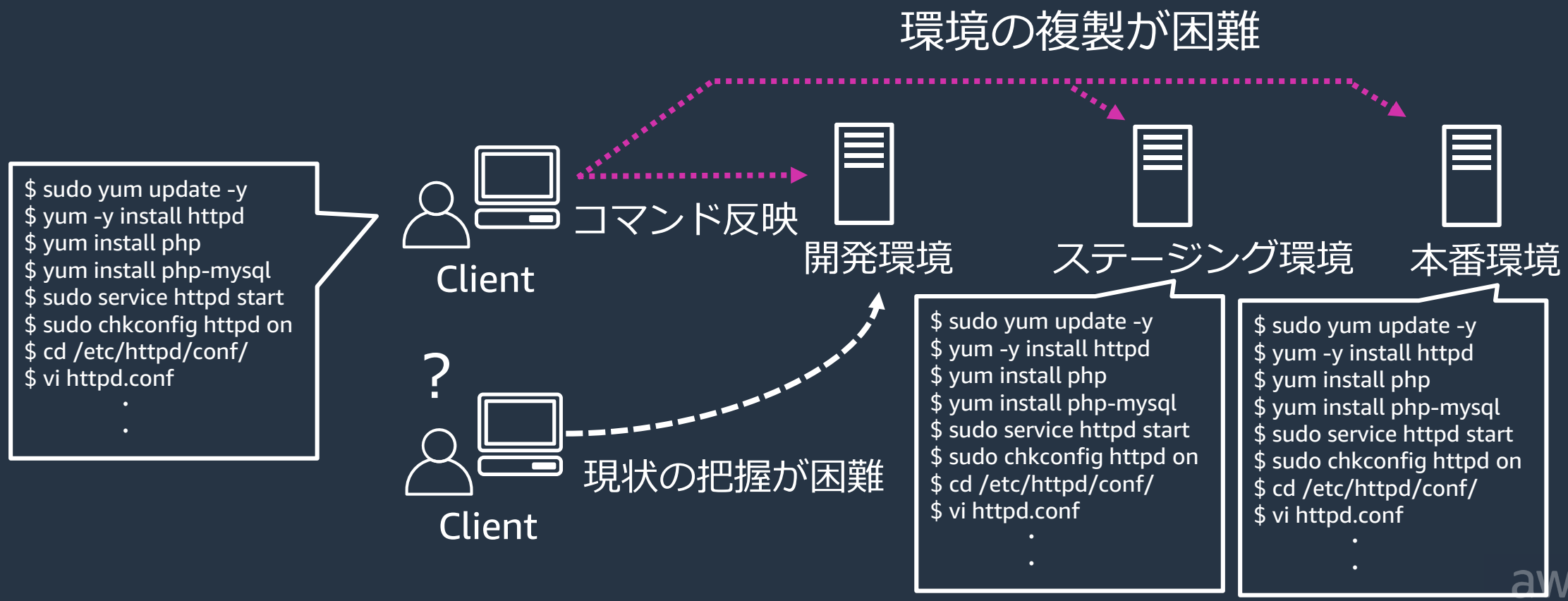
命令的な構築

必要に応じて一つ一つのコマンドを実行してインフラを構築していくこと

- アプリ version1 のコンテナをデプロイする
- 負荷が増えたのでサーバ台数を10台増やす
- アプリ version2 のコンテナデプロイする
- 負荷が減ったのでサーバ台数を3台減らす
- 特定のサーバで障害が発生したので、対象のサーバをロードバランサから切り離し新しいサーバを配置する

命令的なインフラ構築

命令的な構築のデメリット



宣言的なインフラ構築

宣言的な構築

構成のあるべき状態を設定ファイルで定義して

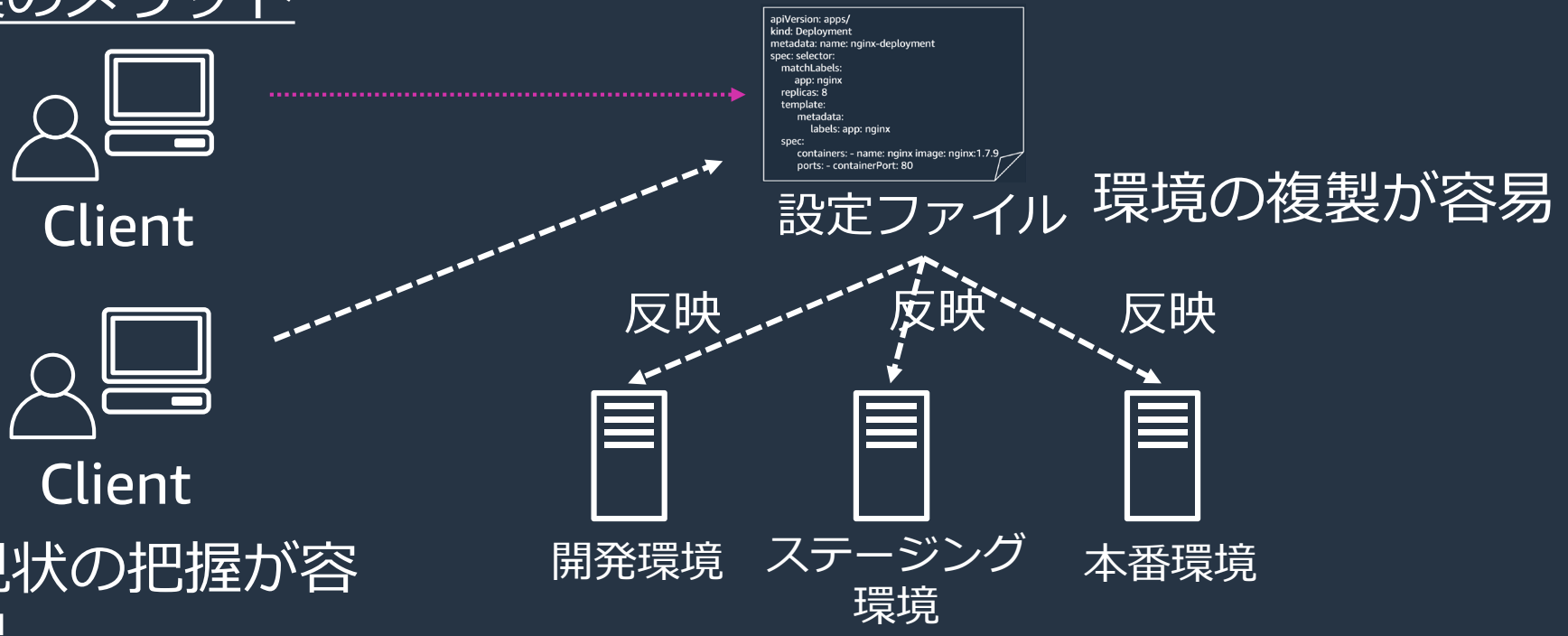
この設定ファイルをクラスターにデプロイしてインフラを構築していくこと

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
…
    spec:
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.21
        ports:
        - containerPort: 80
```

宣言的な構成管理

- ここでいう「宣言的」とは？
- 命令的にコマンドでインフラ構成を指示するのではなく、構成のあるべき状態を設定ファイルで定義すること

宣言的な構築のメリット



GitOps の原則

宣言型ツールを使用することで、システムのあるべき状態をGitで管理可能になる

システム全体が宣言的に記述されていること

望ましいシステム状態が Git でバージョン管理されている

変更を自動的に適用する

エージェントは、正確性を保証し、相違を警告する

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

GitOps の原則

信頼できる唯一の情報源(Single source of truth)としてGitを使用する
→開発者の使い慣れたツールで、変更追跡（「誰が」、「いつ」、「なぜ」変更したか）、バージョン管理、承認（プルリクエスト）

システム全体が宣言的に記述されていること

望ましいシステム状態が Git でバージョン管理されている

変更を自動的に適用する

エージェントは、正確性を保証し、相違を警告する

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

GitOps の原則

ソフトウェアエージェントを使用して、Git リポジトリとクラスターを同期

システム全体が宣言的に記述されていること

望ましいシステム状態が Git でバージョン管理されている

変更を自動的に適用する

エージェントは、正確性を保証し、相違を警告する

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

GitOps の原則

#devaxconnect

相違がある場合、Kubernetes は、クラスターを自動的に更新
→Git上のマニフェストと実環境の一致を保証

システム全体が宣言的に記述されていること

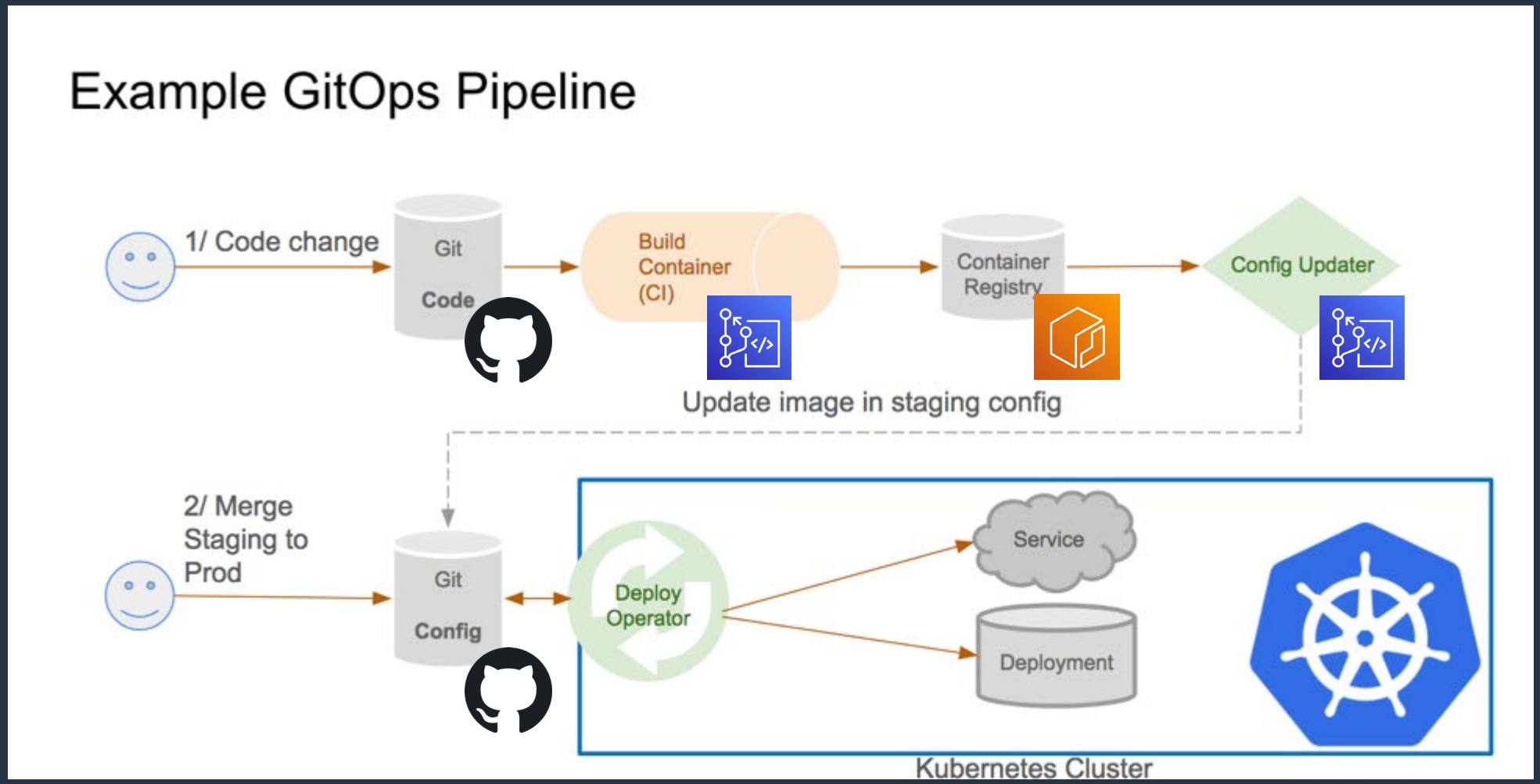
望ましいシステム状態が Git でバージョン管理されている

変更を自動的に適用する

エージェントは、正確性を保証し、相違を警告する

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

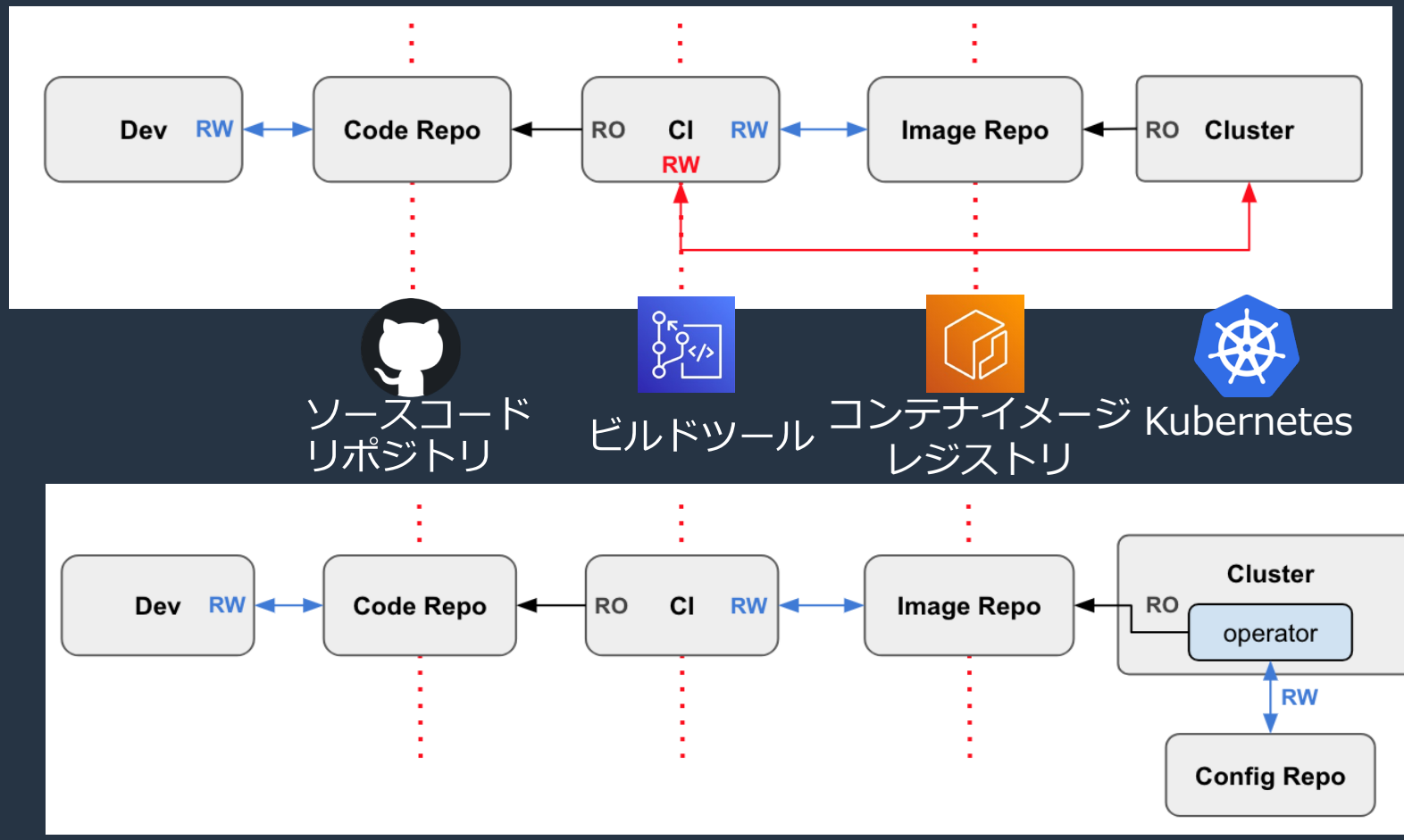
GitOps パイプライン



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

Push 型の CIOps と Pull 型の GitOps

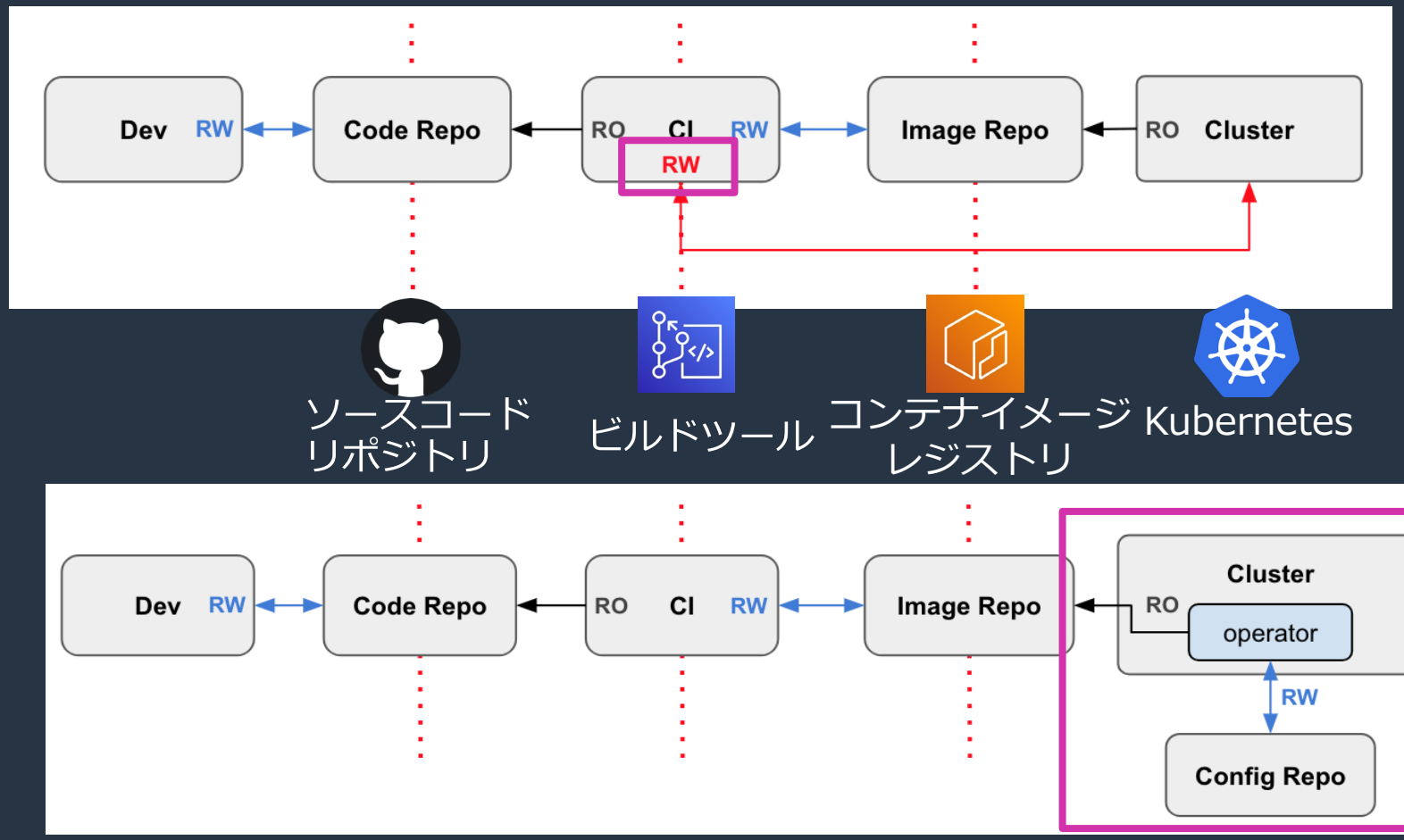
- Push 型
  - 手動 or ビルドツールから kubectlを打つ（apiを呼ぶ）
  - クラスター外からのデプロイ
  - CIツールからデプロイを行うため CIOps とも呼ばれる
- Pull 型
  - クラスター内のエージェントからkubectlを打つ（apiを呼ぶ）
  - 継続的な実行
  - CIとCDが分離している
  - GitOps の OSS はこちらの考え方



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

Push 型の CIOps と Pull 型の GitOps

- Push 型
  - 手動 or ビルドツールから kubectlを打つ（apiを呼ぶ）
  - クラスター外からのデプロイ
  - CIツールからデプロイを行うため CIOps とも呼ばれる
- Pull 型
  - クラスター内のエージェントからkubectlを打つ（apiを呼ぶ）
  - 継続的な実行
  - CIとCDが分離している
  - GitOps の OSS はこちらの考え方



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

従来の CD システムとの比較

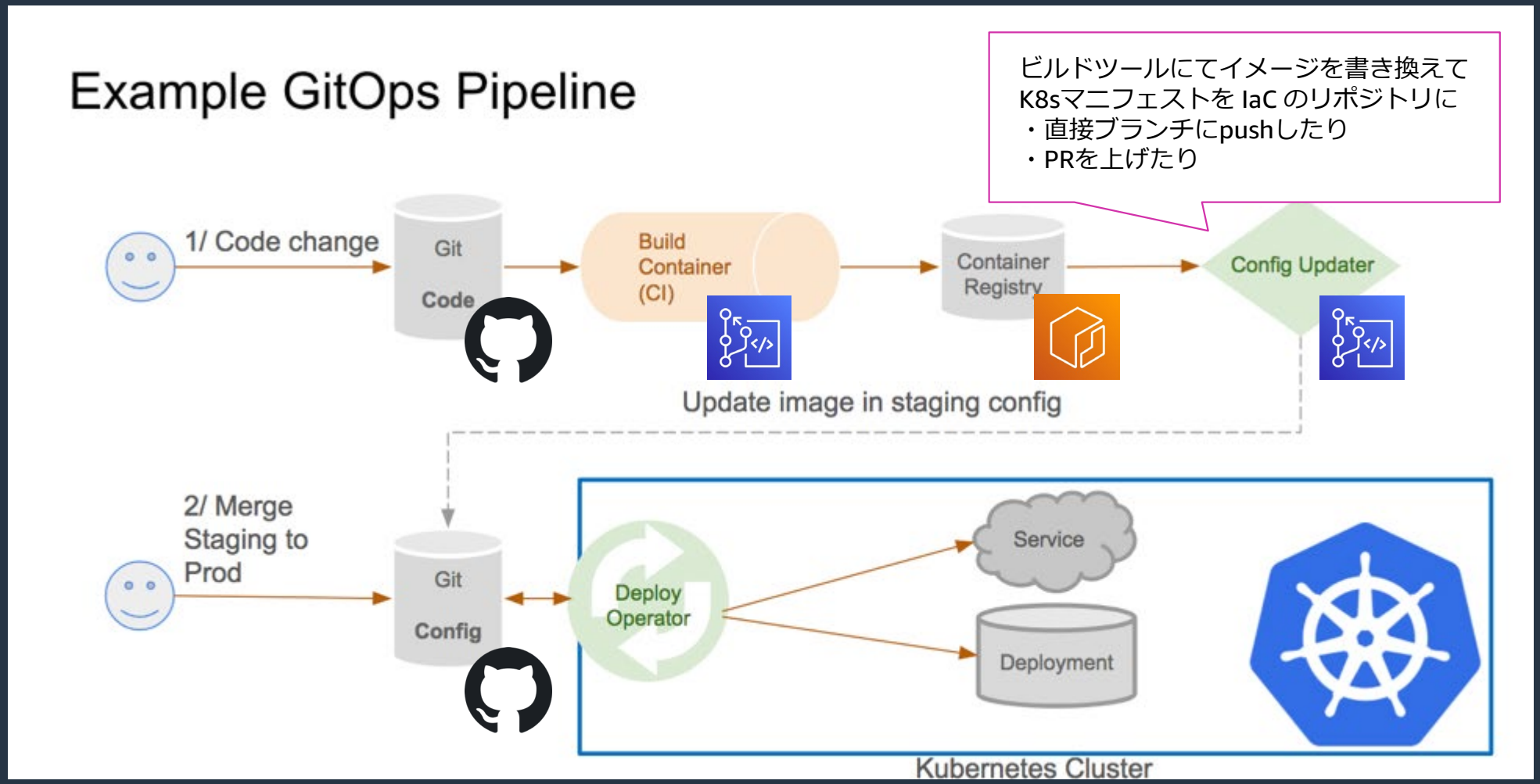
従来の CD システム（CIOps）

- イベント契機でプッシュする
- システムとリポジトリでずれることがある
- CD プロセスは環境にアクセスする必要があります

GitOps ベースのシステム

- 継続的に変更をポーリング
- リポジトリをSingle Source of Truthとして環境と同期
- 差分発生時も修正

GitOps パイプライン



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

GitOps がもたらす価値

- Single Source of Truth
  - Git上のアプリケーションのあるべき状態とクラスタが継続的に同期されている
  - Kubernetes の宣言的 API と相性が良い
- より迅速な開発
  - 開発者は使い慣れたGitを使用して、Kubernetes の更新と機能をより迅速に管理
  - 基盤の構成のバージョン管理が可能できるようになり、ロールバックや切り戻しが確実かつ容易
- より良い運用
  - 継続的インテグレーションと継続的デプロイのパイプラインはすべてプルリクエストによって駆動される（デプロイ前に確実にレビュープロセスが入る）
- より簡単なコンプライアンスと監査
  - Git にてシンプルに各リソース定義を「誰が」、「いつ」、「なぜ」変更したかが分かるようになる
  - CI ツールへの権限集中を回避（CI と CD を分離）

#devaxconnect

Kubernetes で実現する GitOps

aws DevAx connect

Kubernetes の概要

- 複数のホスト間でコンテナ化されたアプリケーションを管理するオープンソースシステム
- Cloud Native Computing Foundation (CNCF) によって管理、推進されている
- デプロイ、メンテナンス、スケーリングといった基本的な機能を提供している

#devaxconnect

Kubernetes における宣言的な Manifest管理

aws DevAx connect

宣言的なインフラ構築

宣言的な構築

構成のあるべき状態を設定ファイルで定義して

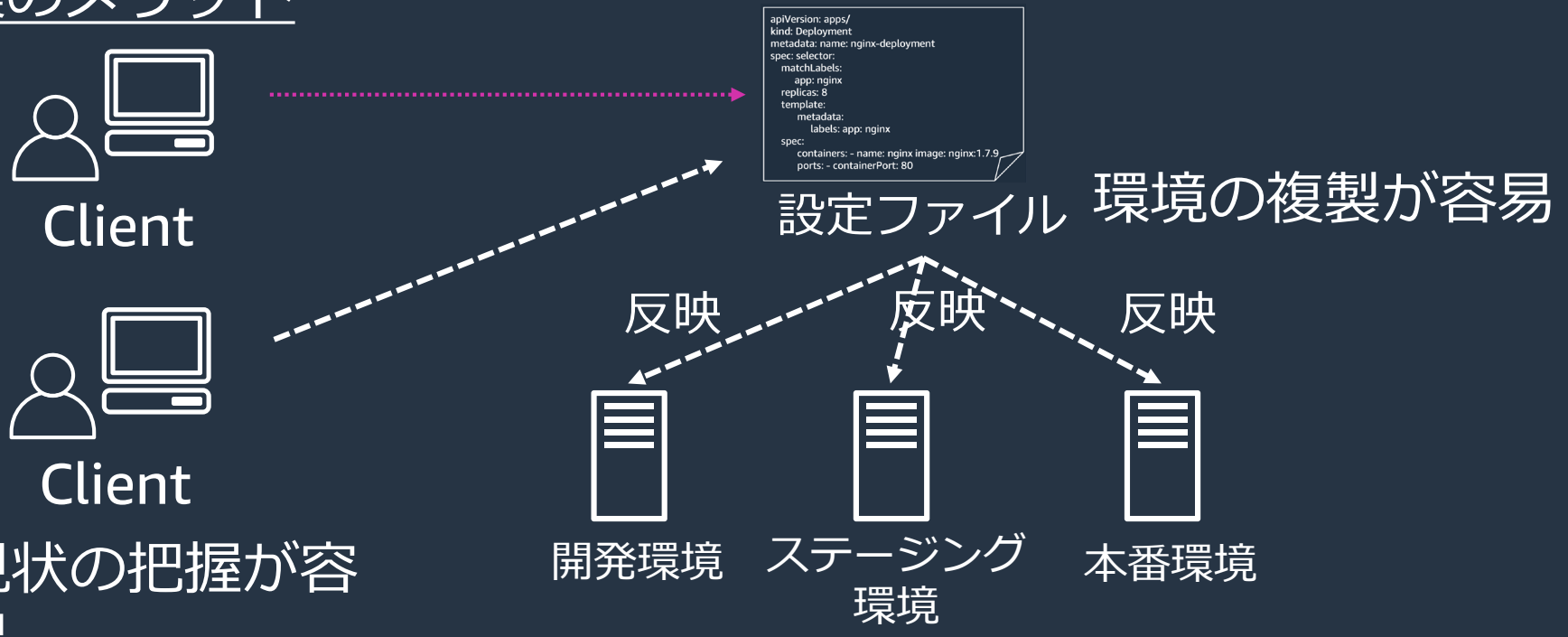
この設定ファイルをクラスターにデプロイしてインフラを構築していくこと

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
…
    spec:
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.21
        ports:
        - containerPort: 80
```

Kubernetes における宣言的な構成管理

- ここでいう「宣言的」とは？
- 命令的にコマンドでインフラ構成を指示するのではなく、構成のあるべき状態を設定ファイルで定義すること

宣言的な構築のメリット



GitOps の原則

#devaxconnect

宣言型ツールを使用することで、システムのあるべき状態をGitで管理可能になる

システム全体が宣言的に記述されていること

望ましいシステム状態が Git でバージョン管理されている

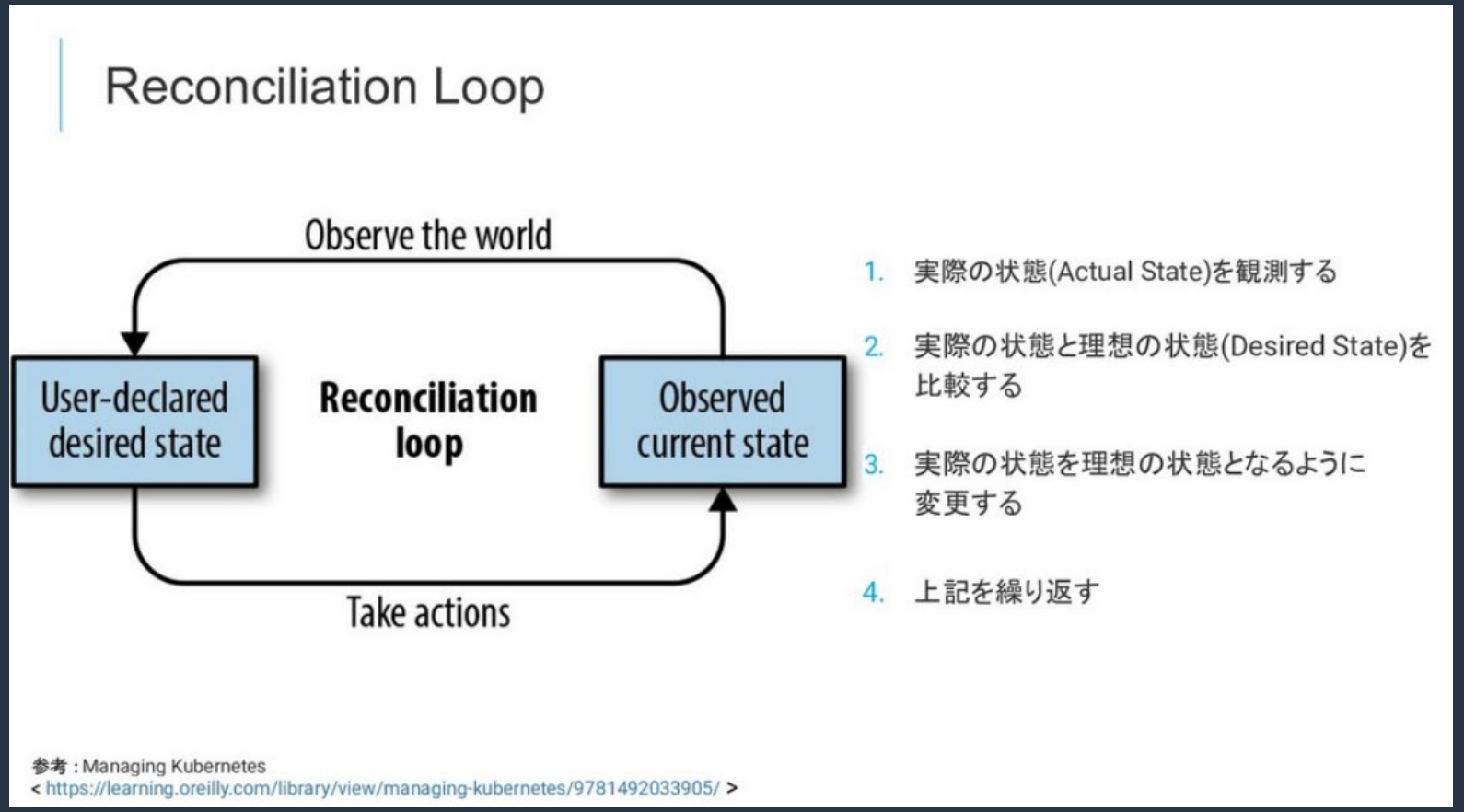
変更を自動的に適用する

エージェントは、正確性を保証し、相違を警告する

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

aws DevAx connect

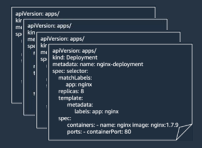
あるべき状態に収束させる機能



https://speakerdeck.com/yosshi_/korekaraxue-hukubernetesfalsereconciliation-loop?slide=12

Kubernetes における宣言的な構成管理

Kubernetes は設定ファイルからクラスタ全体をあるべき状態に収束させる



コンテナを3つ起動する設定

Client

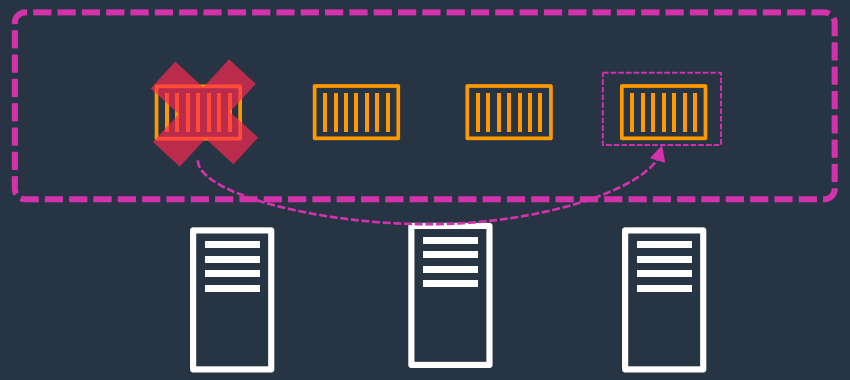
設定ファイル

設定を適用

Kubernetes

反映

クラスタ



障害などで宣言された状態が崩れた場合、自動であるべき状態に復旧させる

GitOps の原則

#devaxconnect

相違がある場合、Kubernetes は、クラスターを自動的に更新
→Git上のマニフェストと実環境の一致を保証

システム全体が宣言的に記述されていること

望ましいシステム状態が Git でバージョン管理されている

変更を自動的に適用する

エージェントは、正確性を保証し、相違を警告する

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

#devaxconnect

Kubernetes におけるデプロイ

aws DevAx connect

手作業でデプロイする場合

Kubernetes は設定ファイルからクラスタ全体をあるべき状態に収束させる

コンテナを3つ起動する設定

設定ファイル

kubectl apply -f

Deploy (Test/Prod)

Client

Kubernetes

反映

クラスタ

コンテナイメージ

Build

docker build

docker push

git push/pull

Source

開発者

ソースコードリポジトリ

コンテナイメージレジストリ

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.21
        ports:
        - containerPort: 80
```

CI（Source→Build）を自動化

Kubernetes は設定ファイルからクラスタ全体をあるべき状態に収束させる

コンテナを3つ起動する設定

設定ファイル

kubectl apply -f

Deploy (Test/Prod)

Client

Kubernetes

反映

クラスタ

開発者

git push

Source

ソースコード リポジトリ

コンテナイメージ レジストリ

Build

ビルドツール

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.21
        ports:
        - containerPort: 80
```

CDは？

Kubernetes は設定ファイルからクラスタ全体をあるべき状態に収束させる

コンテナを3つ起動する設定

設定ファイル

Client

kubectl apply -f

Deploy (Test/Prod)

Kubernetes

反映

クラスタ

開発者

git push

Source

ソースコード リポジトリ

コンテナイメージ レジストリ

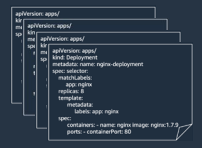
Build

ビルドツール

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.21
        ports:
        - containerPort: 80
```

CDは？

Kubernetes は設定ファイルからクラスタ全体をあるべき状態に収束させる



設定ファイル

コンテナを3つ起動する設定

Client

Kubernetes

Deploy (Test/Prod)

反映

クラスタ

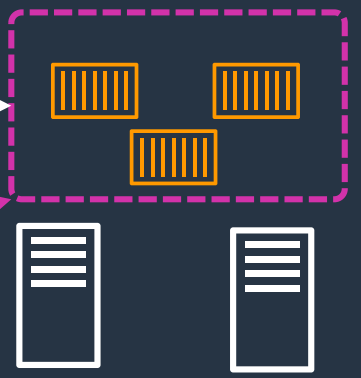
開発者

git push

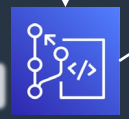
Source

ソースコードリポジトリ

コンテナイメージレジストリ



kubectl apply -f



Build

ビルドツール

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.21
        ports:
        - containerPort: 80
```

#devaxconnect

GitOps の構成方法

aws DevAx connect

GitOps を実現する OSS 例

- Kubernetes の Reconciliation Loop の仕組みを使って、クラスタ上の実際の状態とGit 上のあるべき状態を監視

- Weaveworks 社が開発元
- CLI のみでシンプル構成
- v2はまだGA前

Flux

https://github.com/fluxcd/flux2

- Intuit 社が開発元
- GUIも提供

ArgoCD

https://github.com/argoproj/argo-cd

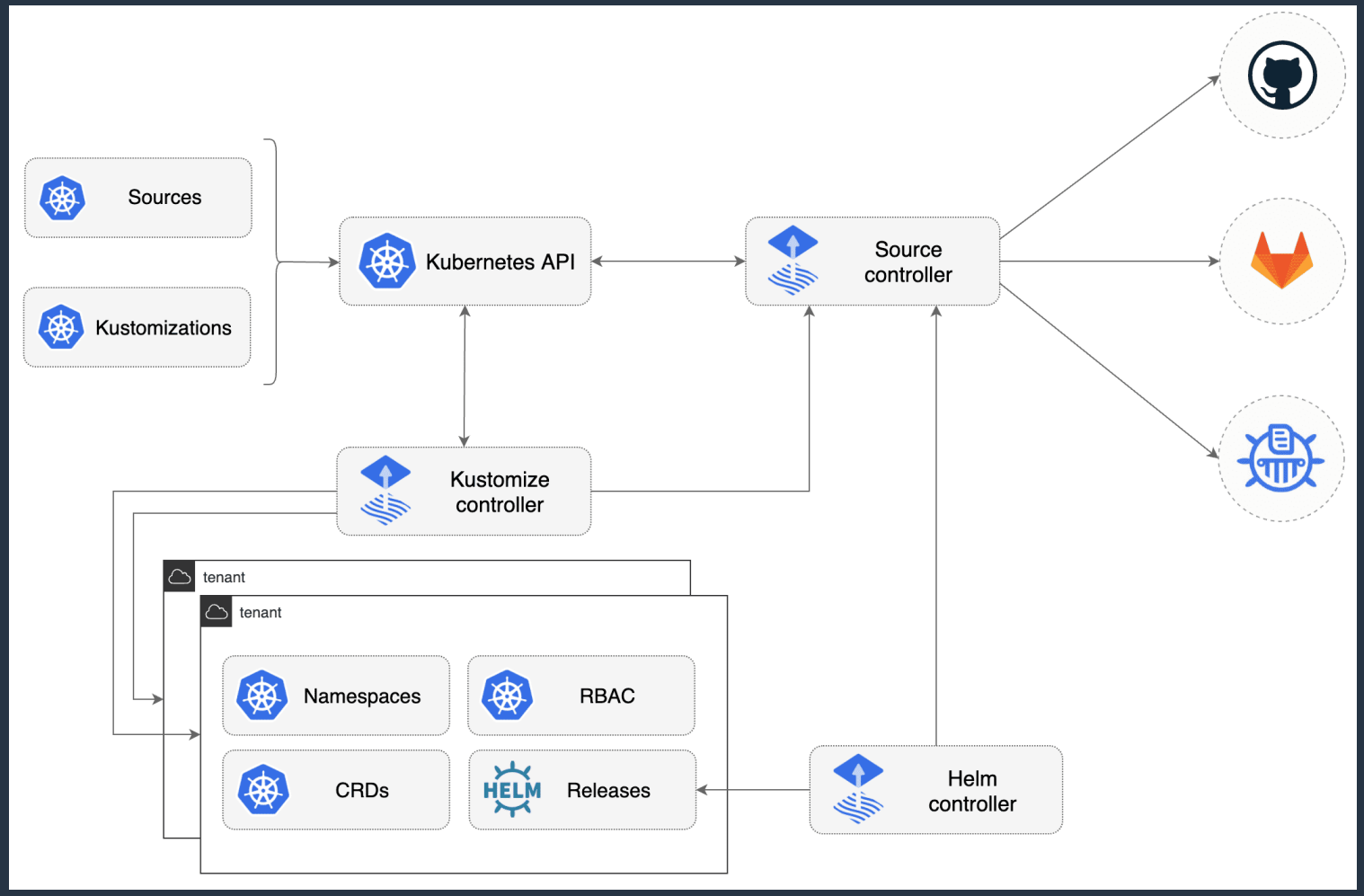
- Cyber Agent 社が開発元
- Kubernetes 以外のリソースにもGitOps 適用可能

PipeCD

https://github.com/pipe-cd/pipecd

GitOps の内部アーキテクチャ （Fluxの例）

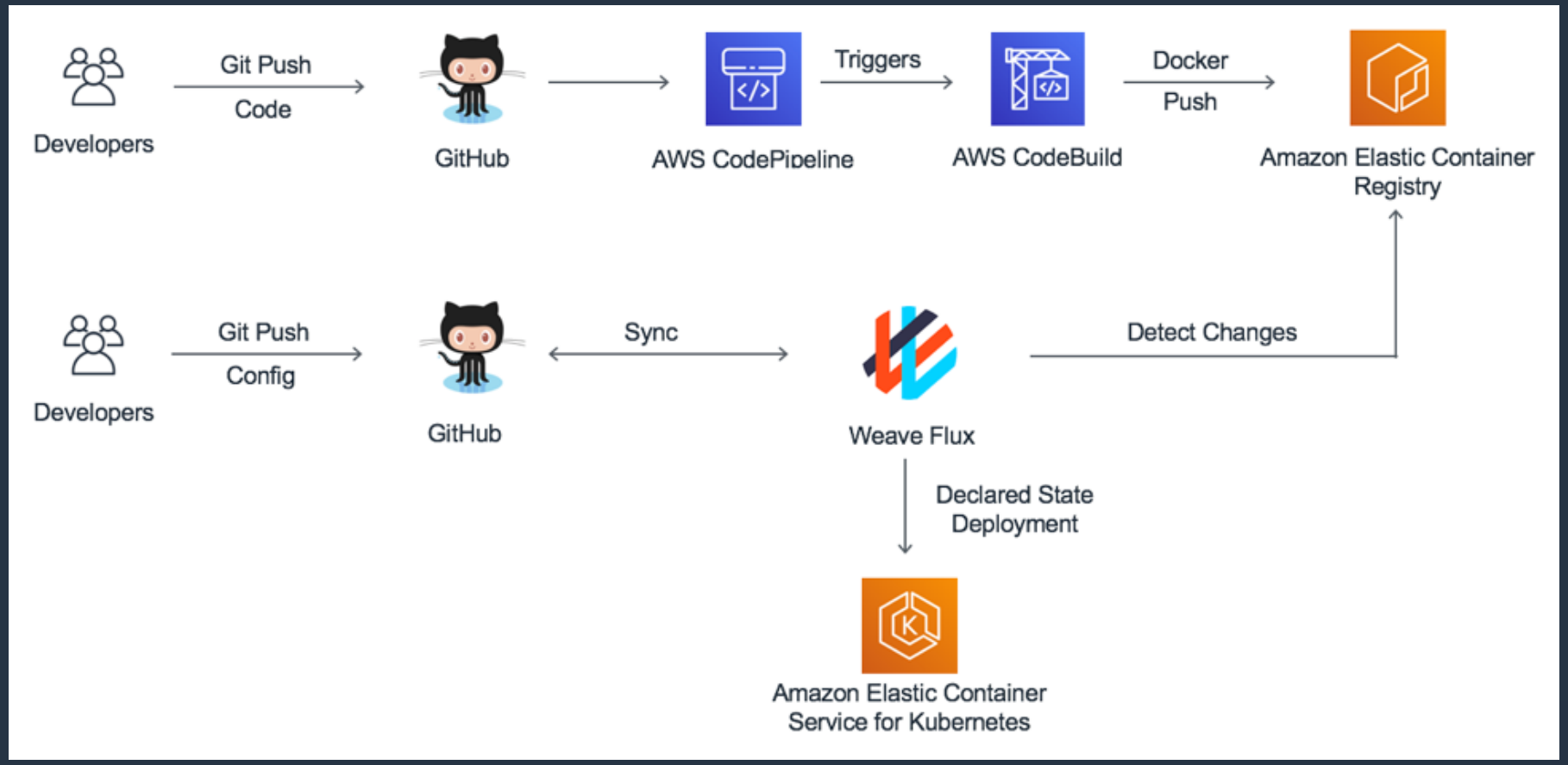
```
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: GitRepository
metadata:
  name: podinfo
  namespace: default
spec:
  interval: 5m
  url: https://github.com/stefanprodan/podinfo
  ref:
    branch: master
---
apiVersion:
kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
  name: podinfo
  namespace: default
spec:
  interval: 10m
  targetNamespace: default
  sourceRef:
    kind: GitRepository
    name: podinfo
  path: "./kustomize"
  prune: true
```



https://fluxcd.io/docs/get-started/

EKS Workshop で GitOps 試せます

https://www.eksworkshop.com/intermediate/260_weave_flux/
https://www.eksworkshop.com/intermediate/290_argocd/

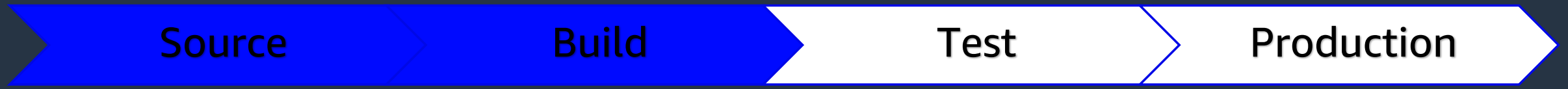


#devaxconnect

GitOps の開発フローへの導入方法

aws DevAx connect

デプロイメントパイプラインの例 / Build



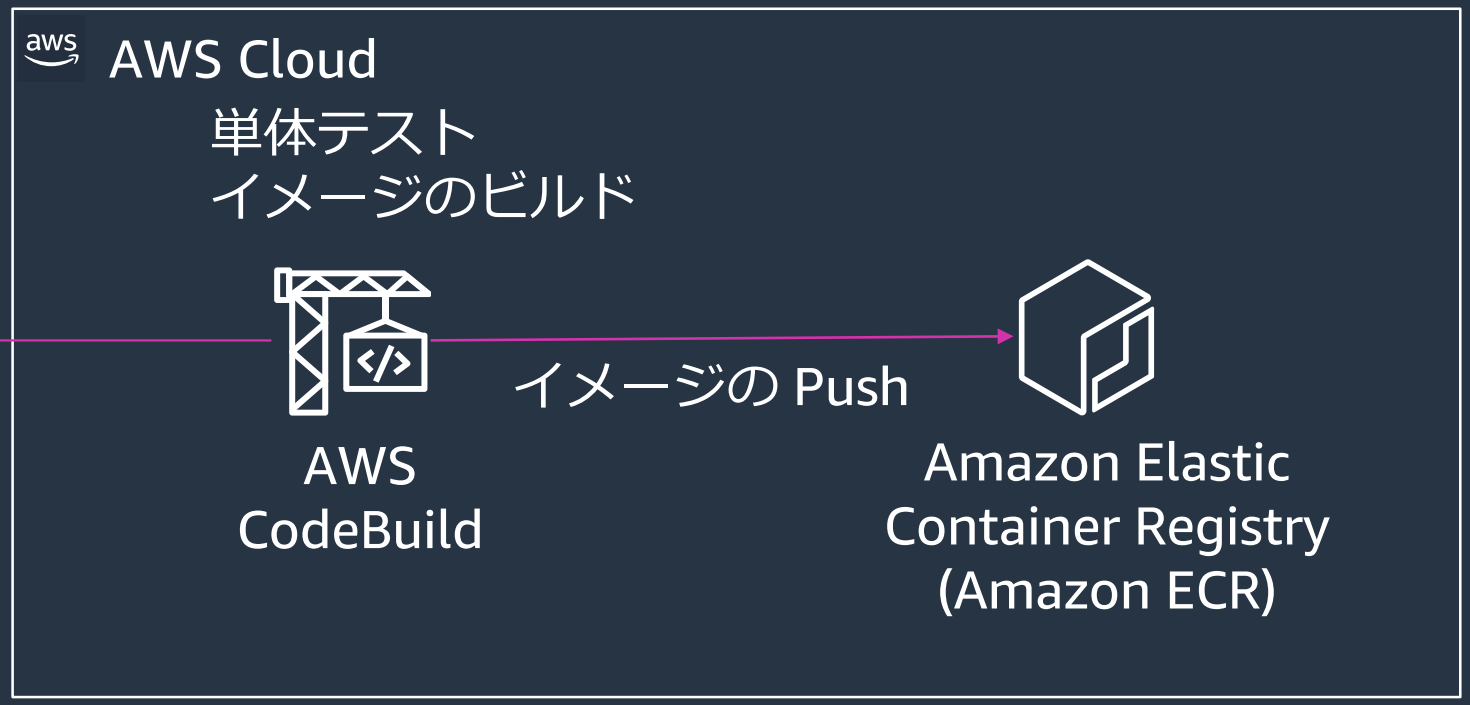
アプリケーション Git リポジトリ

アプリケーション
コード

チェックイン

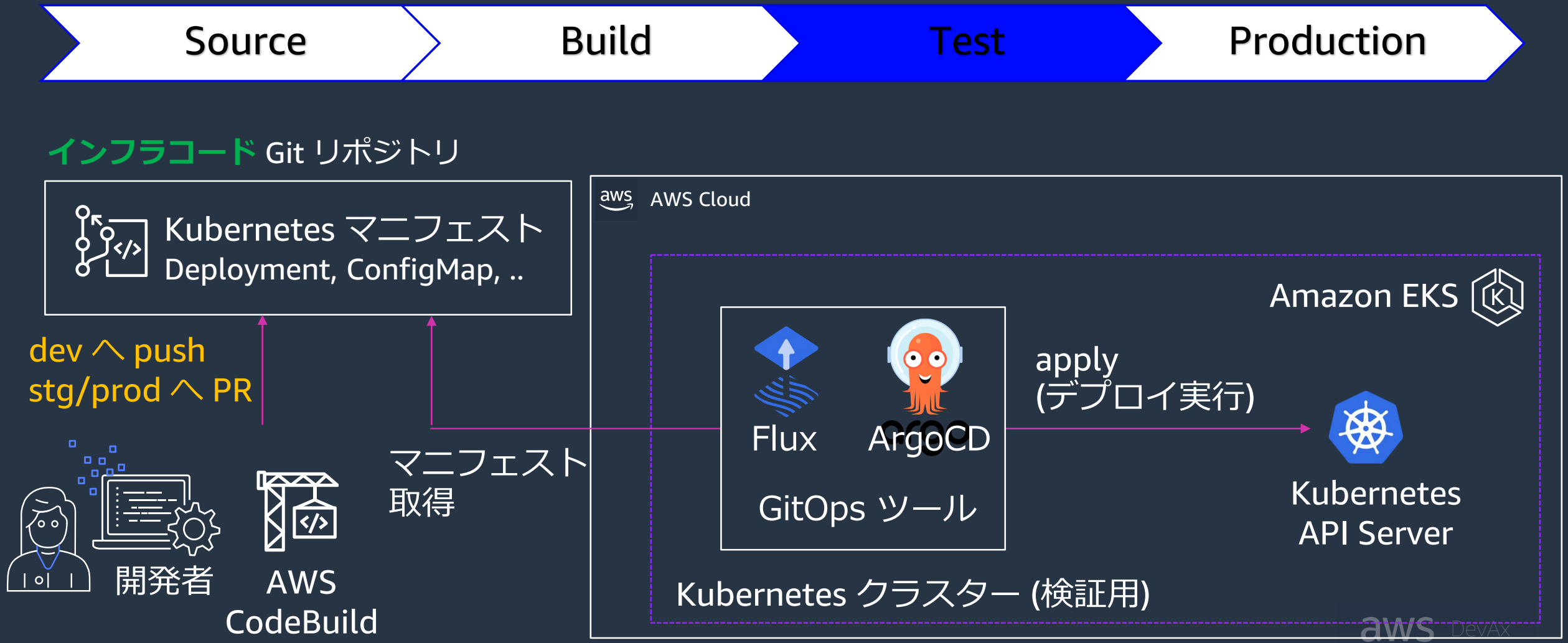
コードの取得

開発者



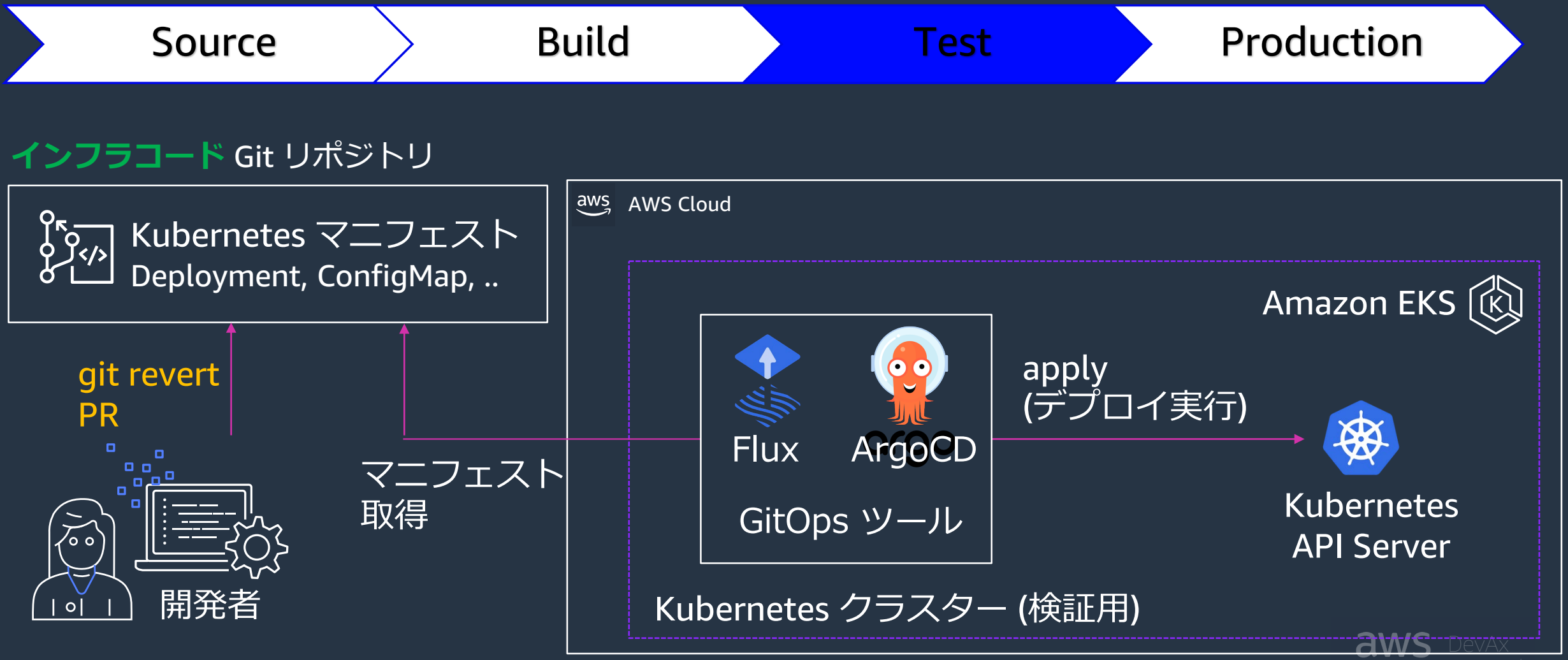
デプロイメントパイプラインの例 / Test

Source | Build | Test | Production



デプロイメントパイプラインの例 / rollback

Source → Build → Test → Production



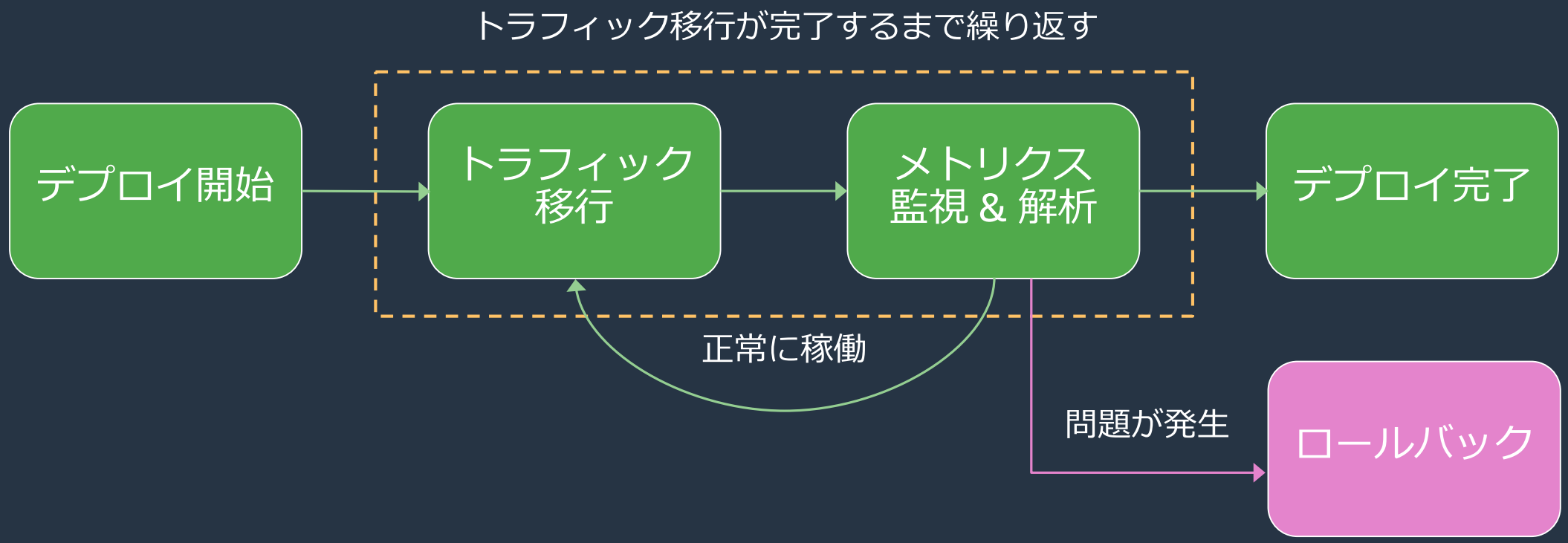
第5回 ECS で考える安全なデプロイメント

落水 恭介
アマゾンウェブサービスジャパン合同会社
ソリューションアーキテクト

CI/CD パイプラインの設計において、ロールバックプロセスの検討は最も重要な観点の1つです。アラートをトリガーに運用担当者へメールを送る、いくつかのメトリクスをターゲットにロールバックを判断するアクションをパイプラインに組み込むなど、その方法はさまざまです。本セッションでは、ECS へのデプロイを例に、ロールバックプロセスをどのように設計するのか、CI/CD パイプラインへのロールバックプロセスの導入についてお話します。

プログレッシブデリバリー

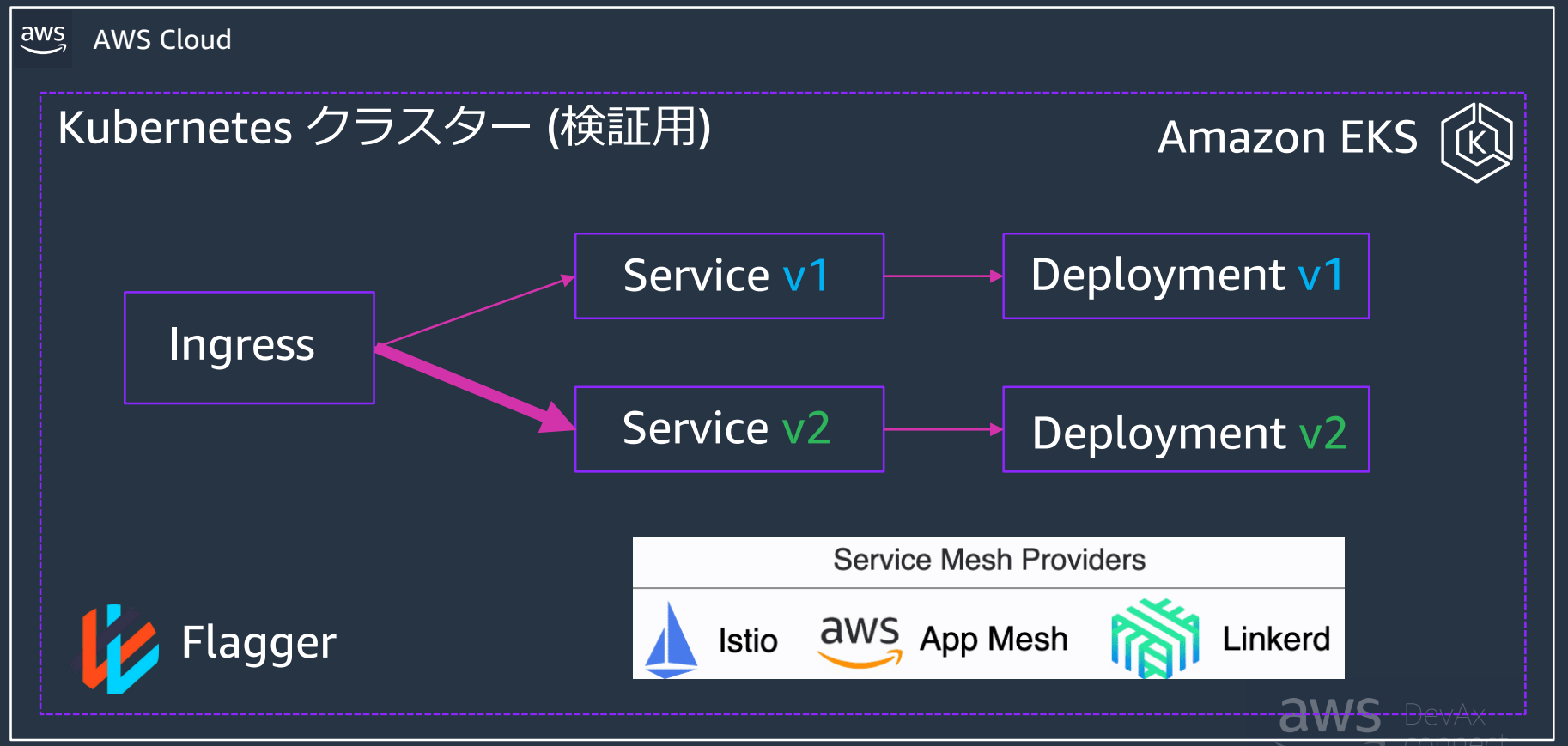
新バージョンのアプリケーションを段階的に公開



デプロイメントパイプラインの例 / Test

https://aws.amazon.com/jp/blogs/containers/progressive-delivery-using-aws-app-mesh-and-flagger/

Source | Build | Test | Production



Metrics

Amazon CloudWatch
Prometheus
Datadog
…

本番環境で GitOps を採用するための考慮事項

- Secrets の管理
  - E.g. AWS Secrets and Config Provider（ASCP）
- 可観測性の向上と自動ヘルスチェックのためのツール
  - E.g. 死活監視、外形監視など
  - E.g. RED(Rate, Errors, Duration) x USE(Utilization, Saturation, Errors) の監視
- 環境間差分の表現
  - E.g. Kuberentes テンプレートエンジンの活用（Kustomize, Helm）

#devaxconnect

まとめ

aws DevAx connect

Key Takeaways

- GitOps は Git をデプロイパイプラインの中心に据えた IaC デプロイの考え方
- GitOps の特徴
  - Single Source of Truth
    - Git上のアプリケーションのあるべき状態とクラスタが同期されており差分が明確
  - より迅速な開発
    - 開発者は使い慣れたGitを使用して、Kubernetes の更新と機能をより迅速に管理
  - より良い運用
    - 継続的インテグレーションと継続的デプロイのパイプラインはすべてプルリクエストによって駆動される
  - より簡単なコンプライアンスと監査
    - Git にてシンプルに各リソース定義を「誰が」、「いつ」、「なぜ」変更したかが分かるようになる
- GitOps のツールとして Flux2 と ArgoCD を紹介

参考

- https://www.weave.works/blog/gitops-operations-by-pull-request
- https://www.weave.works/technologies/gitops/
- https://aws.amazon.com/jp/blogs/news/building-a-gitops-pipeline-with-amazon-eks/
- https://fluxcd.io/docs/
- https://argo-cd.readthedocs.io/en/stable/
- https://www.eksworkshop.com/intermediate/260_weave_flux/
- https://www.eksworkshop.com/intermediate/290_argocd/

Thank you!

aws DevAx connect