MA22278

ゼロから始める Kubernetes 運用の自動化

モダンな CI/CD パイプラインの作り方

ヴイエムウェア株式会社

プロフェッショナルサービス統括本部
コンサルティングアーキテクト
若林 寛之

免責事項

- このセッションには、現在開発中の製品/サービスの機能が含まれている場合があります。
- 新しいテクノロジーに関するこのセッションおよび概要は、VMware が市販の製品/サービスにこれらの機能を搭載することを約束するものではありません。
- 機能は変更される場合があるため、いかなる種類の契約書、受注書、または販売契約書に記述してはなりません。
- 技術的な問題および市場の需要により、最終的に出荷される製品/サービスでは機能が変わる場合があります。
- ここで検討されているまたは提示されている新しいテクノロジーまたは機能の価格およびパッケージは、決定されたものではありません。

Agenda

CI/CD が求められる背景

構成例と実装における鍵

デザインパターンと高度化に向けたポイント

Agenda

CI/CD が求められる背景

構成例と実装における鍵

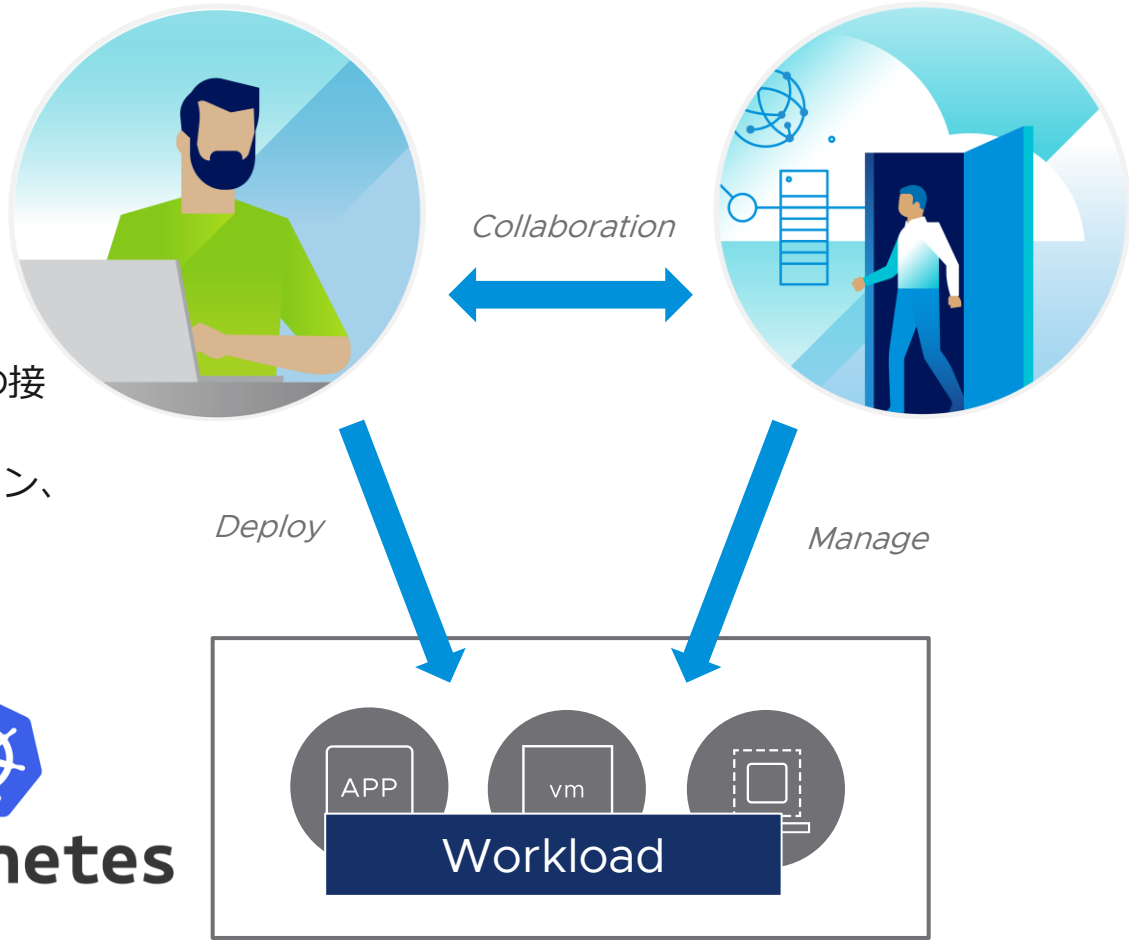
デザインパターンと高度化に向けたポイント

モダンアプリケーションと Kubernetes

リリースサイクルの高速化と Kubernetes

Developer

- 組織内での**最新**の開発者向けサービスの欠如
- **最新**のコーディングツールやバックエンドデータベースへの接続の遅延
- デプロイ、Day2 オペレーション、ライフサイクル**管理**が苦痛



IT Operator

- 開発者向けリソースのプロビジョニングが困難なため、インフラが**サイロ化**
- モダンアプリや繊細なデータベースのセキュリティ**分離**が困難
- **一貫性のない**オペレーションと機能横断なワークフローの懸念が消えない

Tanzu Kubernetes Grid の位置付け
インフラストラクチャアップとアプリケーションダウン

アプリケーション

OSS を活用した迅速なアプリケーション開発

アプリケーションのコード化とコンテナ化

アプリケーションデプロイの自動化

開発者（Developer）

DevOps

Kubernetes を共通言語とした
アプリケーション/インフラストラクチャの歩み寄り

基盤管理者（IT Operator）

多様な環境やユーザーへのKubernetes クラスタのデプロイ、管理

意思決定のために必要とされる可観測性

サービス間でのセキュアで信頼性の高い通信の保証

インフラストラクチャ

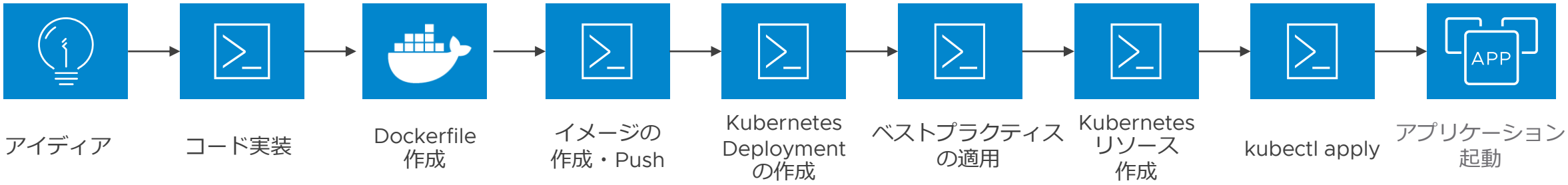
VMware vSphere | Azure | AWS

マルチクラウド環境

IBM Cloud | Google Cloud | VMware Cloud

Kubernetes 上でアプリケーションを動かすまで

数多くの工程と運用コスト



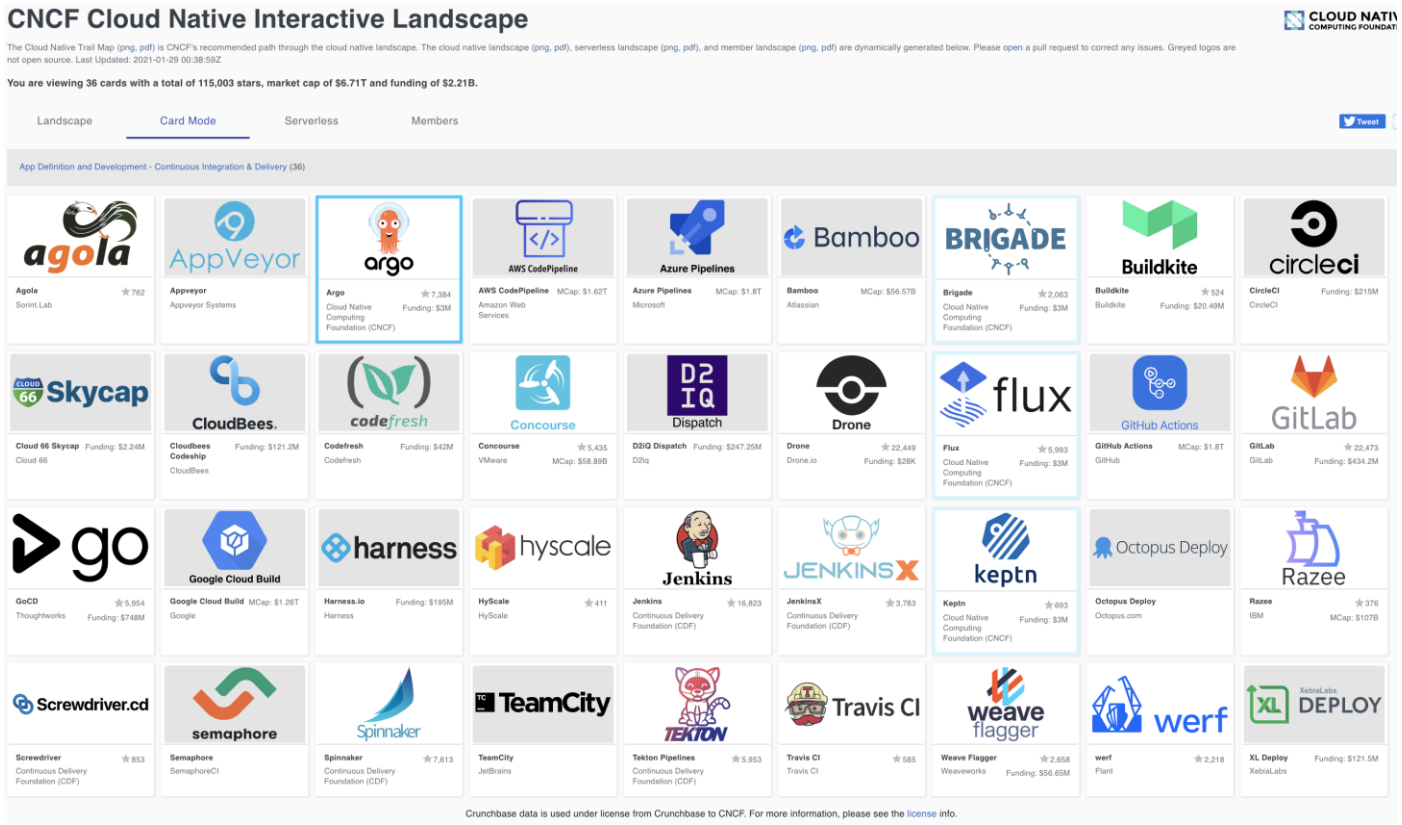
Kubernetes 上でアプリケーションをコンテナとして動かすまでには様々な工程が存在する

- 「Kubernetes 上で動かすが故に」「コンテナ化したが故に」必要となるステップも多い

すべての工程を手作業で行うと Kubernetes 導入前よりも運用コストは増大してしまい
Kubernetes による効果どころか本末転倒に

Kubernetes 上でアプリケーションを動かすまで

ツールの多さ



Source： https://landscape.cncf.io/card-mode?category=continuous-integration-delivery&grouping=category

CI/CD パイプラインツールは数多く存在する

特定の工程に特化したツールも多く、使い方や思想が異なる

ツール毎にカバー範囲も異なる

異なるインターフェイスやツールによるナレッジの分散や管理の煩雑さが課題に

CI/CD パイプラインによる解決

パイプライン工程における作業の自動化

Kubernetes 上でアプリケーションを動かすまでのすべての工程を継続的に実施するために、
ソースコードを変更してからのプロセスを可能な限り自動化する

ソフトウェアに関する様々な速度を向上することが目的

- デプロイにかかる時間、頻度、平均修復時間（MTTR）、平均故障間隔（MTBF）など

ビルド・テスト・リリース準備・デプロイをシステムとして接続し、手作業による運用コストを最小化

繰り返し行われるステップ・（手作業では）時間のかかるステップを
CI/CD パイプラインで自動化することで Kubernetes 導入の効果が最大化できる

Agenda

CI/CD が求められる背景

構成例と実装における鍵

デザインパターンと高度化に向けたポイント

検討のポイント

パイプライン設計におけるポイント

どのタスクを自動化するか？

タスクの洗い出し

- どのような作業が必要になるのかを事前に確認
- コマンドなどの手順ベースで洗い出せることが理想的

実装の優先度設定

- 実施頻度の高いもの
- 手作業で行うと時間のかかるもの

パイプラインを実行するのは誰か？

パイプラインの実行パターン

- ユーザー（手動）
- Git コミット
- コンテナイメージ Push
- 別パイプラインからの呼び出し

呼び出し元の違いによる考慮

- パイプラインが人間により実行される場合には、入力項目を柔軟に設定可能
- Webhook などを活用したその他のパターンでは、認証情報などを含め必要な入力情報を準備しておく必要がある

どのようなステップが必要か？

ステップの順序

- 処理間の依存関係
- ユーザーやシステムが結果を知る/取得するタイミング

外部連携

- ユーザーによる承認プロセスの有無
- SaaS や既存システムとの連携で必要な事前処理

検討のポイント

ツールの実装方式の違い

オンプレミス型

オンプレミスのリソースにツールをインストールすることで利用する形式

メリット

- 拡張性・カスタマイズ性が高い
- 既存運用との連携がしやすい

デプロイ形式の違い

- 仮想マシン
- Kubernetes リソース（in-cluster）

クラウド型

SaaS として提供されるツールと連携させることで利用する形式

メリット

- 運用が不要
- プラグインが豊富
- CAPEX の最小化が可能

パイプライン構成サンプル

構成の指針

VMware Tanzu Kubernetes Grid 上に構築する in-cluster 型のパイプライン

デプロイ先

容易なデプロイと
Kubernetes の自動修復機能
を活用した
運用負荷の低減

メンテナンス性

パイプライン定義を
マニフェストとして管理

デバッグ手法

kubectl による開発者・管理者
での共通手法の提供

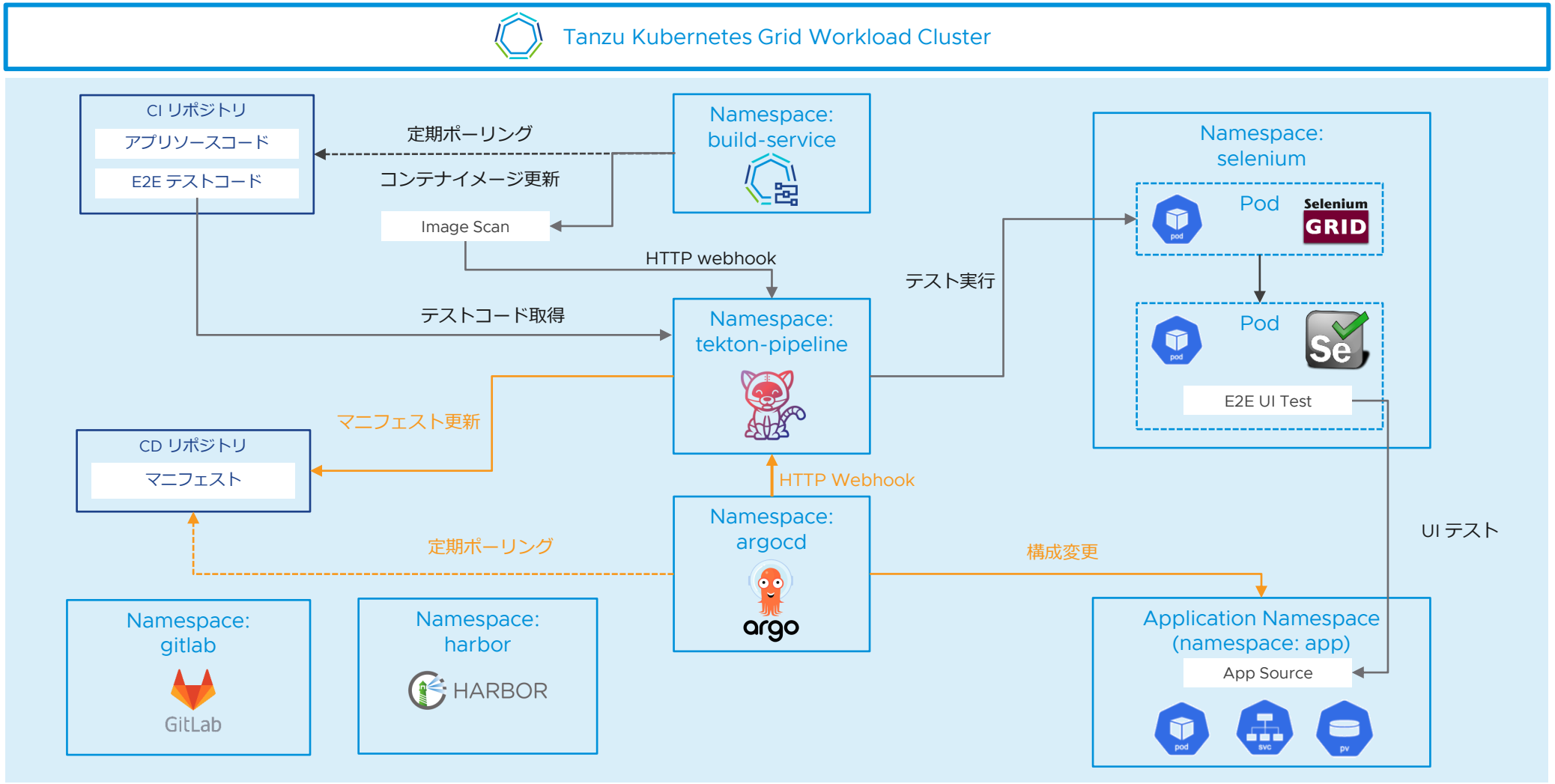
拡張性

ツールや対象の変更に対して
対応可能なコンポーネントの
分離

VMware Tanzu Kubernetes Grid	VMware Tanzu Build Service	Harbor	Tekton	ArgoCD	Selenium Grid
					

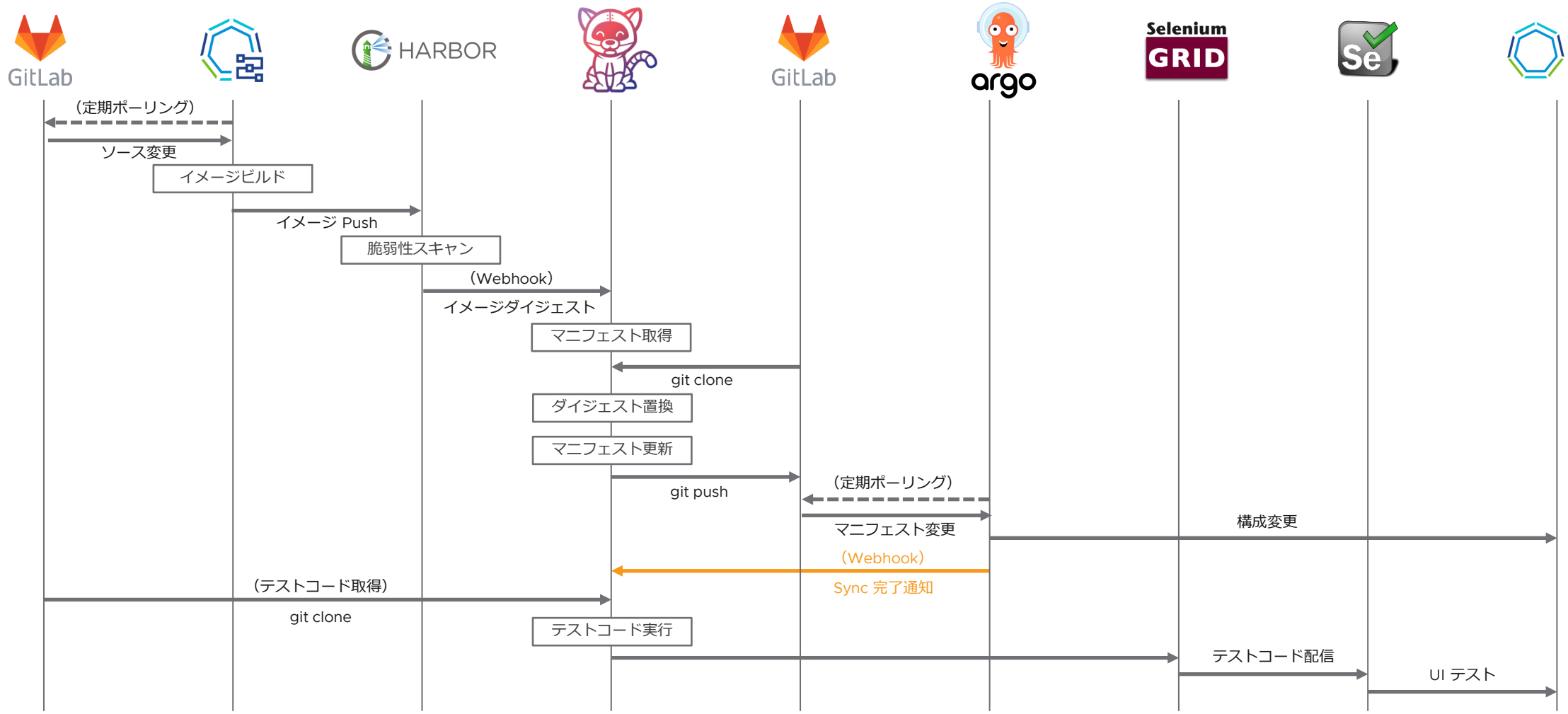
パイプライン構成サンプル

今回のセッションで目指す構成



パイプライン構成サンプル

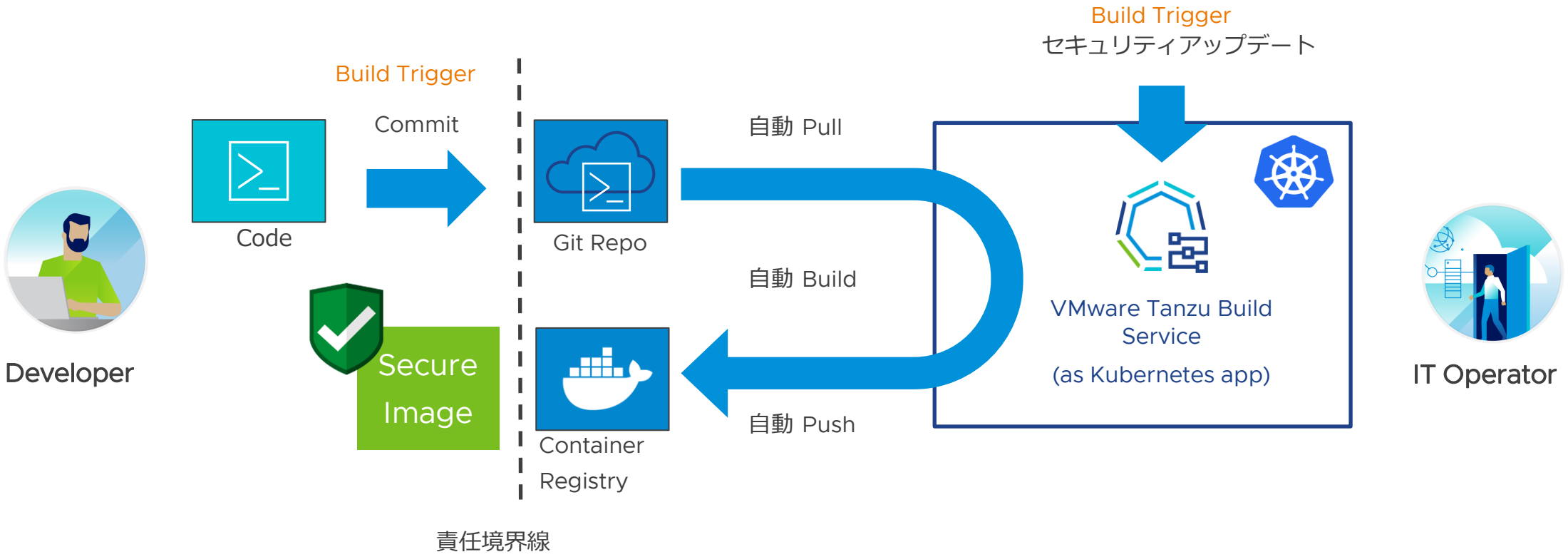
全体の処理シーケンス



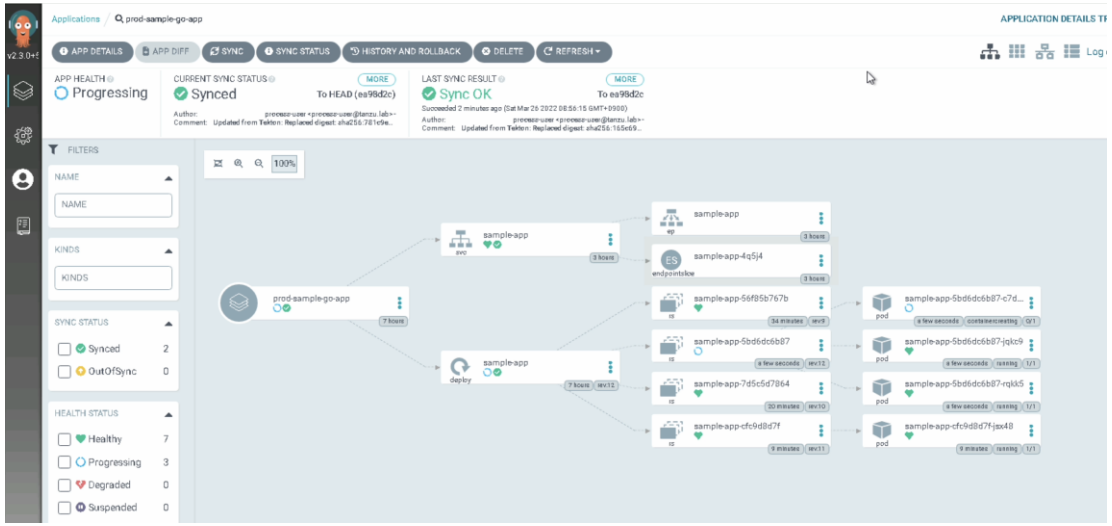
VMware Tanzu Build Service

ソースコードからセキュアなコンテナイメージをビルド（生成）するサービス

Kubernetes 上でコンテナイメージをビルドする仕組みを提供
ソースコードを VMware Tanzu Build Service が解析し、ベストプラクティスに沿ったセキュアなイメージをビルド



ArgoCD



Kubernetes へデプロイする GitOps に特化した OSS の CD ツール

Kubernetes のマニフェストが含まれた Git リポジトリを登録することで、リポジトリと同期を取る形で Kubernetes クラスタ上にアプリケーションをデプロイすることができる

リッチな UI を備えており、Kubernetes リソースの可視化が可能な側面も持つ

Tekton

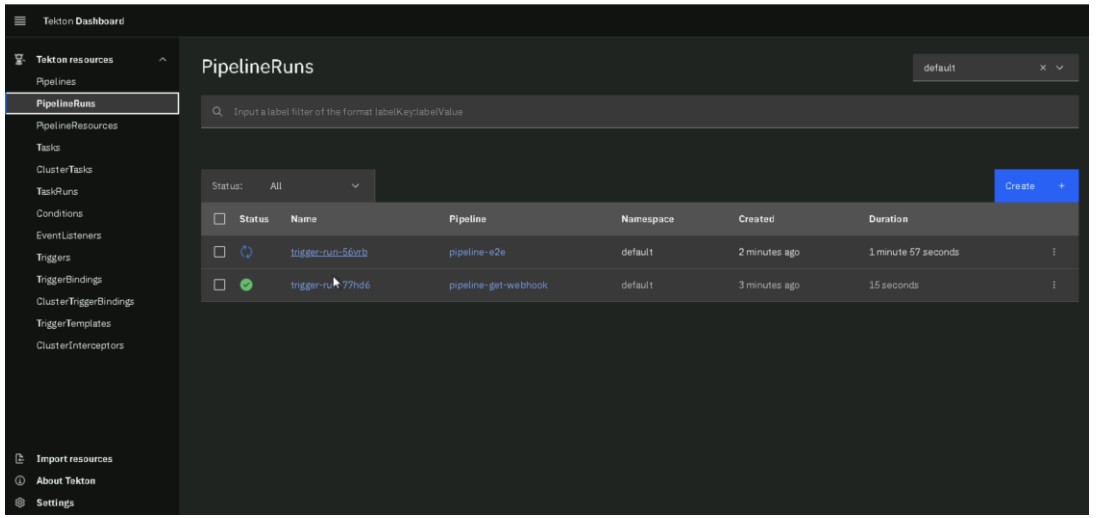
CI/CD パイプライン作成のための OSS

- パイプライン設定が Custom Resource として扱われる
- Pipeline/Trigger/Dashboard の 3 つのコンポーネントから構成される（Trigger および Dashboard の追加は任意）

各タスクやパイプラインの処理は
Kubernetes マニフェストとして定義することが可能

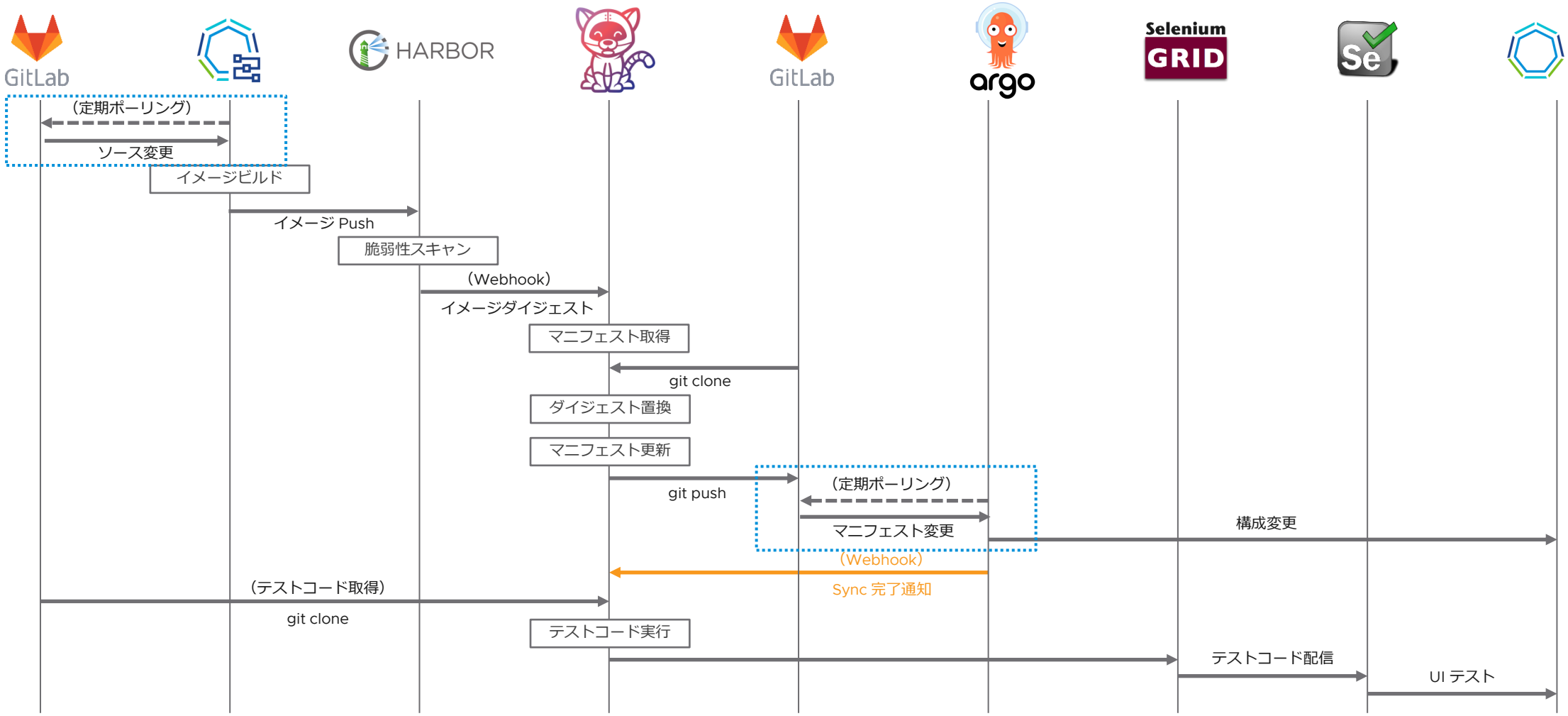
パイプライン中の「タスク」は Pod の中のコンテナで実行される

- タスクによる生成物は PV （Workspace）に保存される



パイプライン構成サンプル

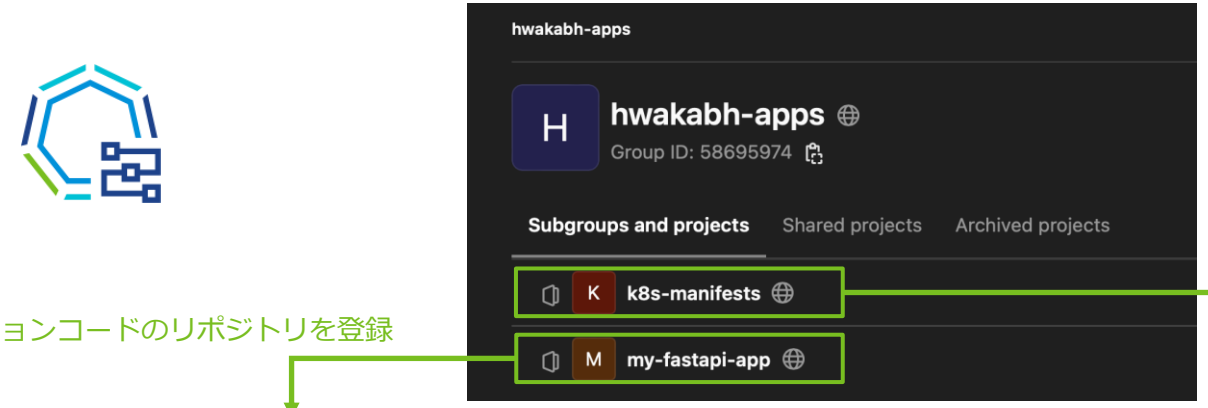
Git リポジトリとの連携



GitLab – VMware Tanzu Build Service / GitLab - ArgoCD

GitLab との連携

テストコードを含むアプリケーションのソースコードとマニフェストとをリポジトリ単位で分離することで、
CI プロセスと CD プロセスとを分離する



アプリケーションコードのリポジトリを登録

Kubernetes マニフェストのリポジトリを登録

```
kp image save my-fastapi-app ¥
--tag harbor.homelab.net/my-fastapi-app ¥
--namespace my-fastapi-app ¥
--wait ¥
--git https://gitlab.homelab.net/hwakabh-apps/my-fastapi-app.git ¥
--git-revision main
```

```
argocd app create my-fastapi-app ¥
--repo https://gitlab.homelab.net/hwakabh-apps/k8s-manifests.git ¥
--path ./ ¥
--dest-server https://kubernetes.default.svc ¥
--dest-namespace app
```

※ kp コマンド

VMware Tanzu Build Service の内部コンポーネントである kpack のクライアントツール

Git リポジトリとの連携における検討ポイント

アプリケーションコード・マニフェストをどのように取得するか

Pull 型

Git リポジトリを登録することで、ツール側が Git リポジトリを定期ポーリングすることで変更を検知する方式

タスク間の処理シーケンスが双方向となる

人手を介する必要がない一方で、ツール側のポーリング間隔に依存する

サンプル構成での採用方式

Push 型

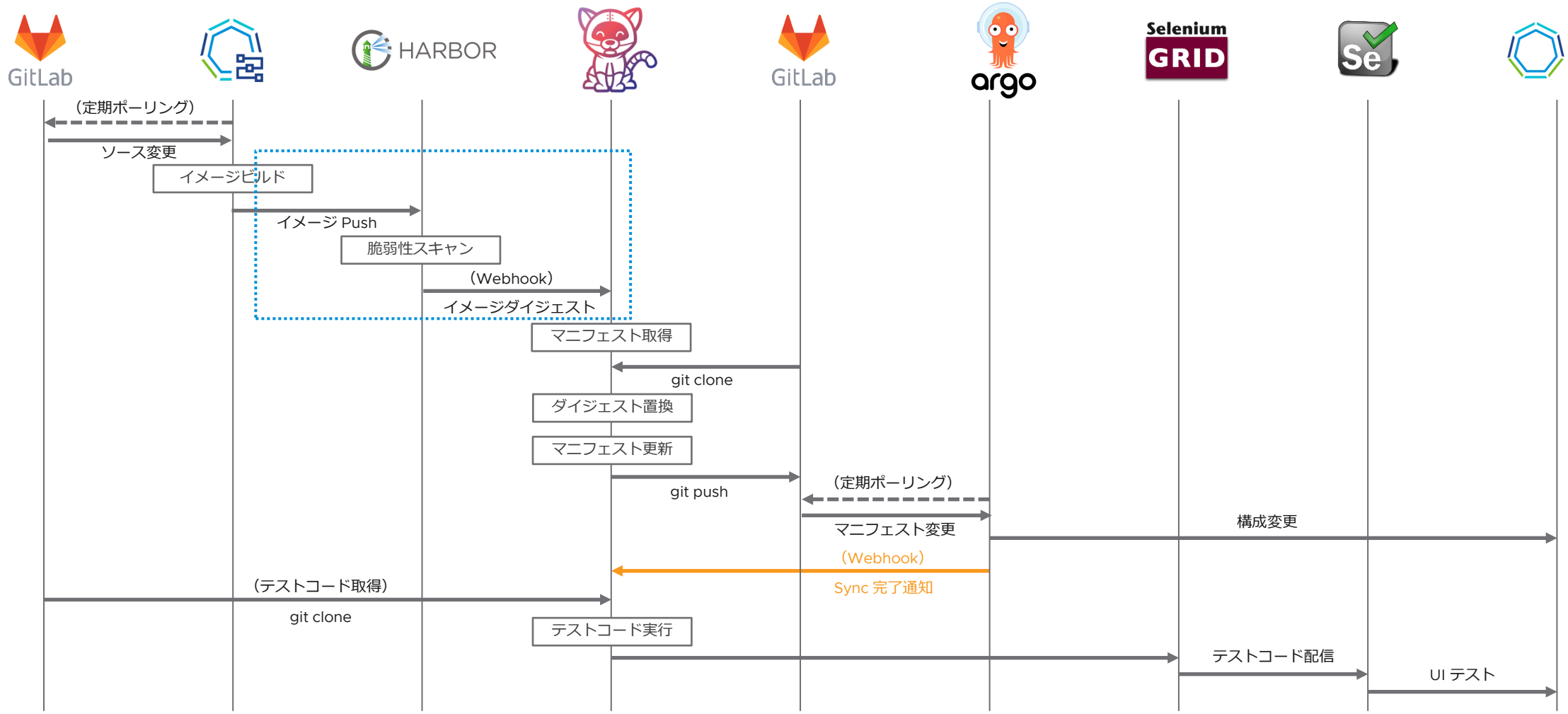
パイプライン内の処理で、クライアントツールを用いて CI/CD ツールを呼び出し変更を通知する方式

タスク間の処理シーケンスが一方向となる

変更後に即時後続ステップを呼び出せる一方で、パイプライン内に実装する処理が多くなる

パイプライン構成サンプル

イメージレジストリとの連携



Harbor - Tekton

EventListener を用いた Harbor Webhook の受信

Tekton では Triggers コンポーネントを導入することで、様々なイベントソースから情報を検出して抽出し、
ユーザーが定義したパイプラインを呼び出すことができる

EventListener リソース作成時に TriggerTemplate リソースを指定することで呼び出す処理を指定
EventListener リソースが作成されると Kubernetes クラスタ上の Tekton が外部からのリクエストを受ける Service リソースが展開される

Kind: EventListener

```
spec:
  resources:
    kubernetesResource:
      serviceType: LoadBalancer
  triggers:
    - bindings:
        - ref: tb-harbor
      template:
        ref: tt-harbor
```

TriggerBinding と TriggerTemplate の両リソースを紐付け、Webhook 受信のエンドポイントを作成する

Kind: TriggerBinding

```
spec:
  params:
    - name: resp-tag
      value: $(body.event_data.resources[0].digest)
```

Harbor の Webhook ペイロードから
目的のデータ（イメージダイジェスト）を取得

Kind: TriggerTemplate

```
spec:
  params:
    - name: resp-tag
      description: The hash of container image
  resourcetemplates:
    - apiVersion: tekton.dev/v1beta1
      kind: PipelineRun
      metadata:
        generateName: trigger-run-
      spec:
        pipelineRef:
          name: pipeline-get-webhook
```

定義した PipelineRun リソースを指定することで
任意の処理を呼び出す

イメージレジストリとの連携におけるポイント

Webhook による連携時の考慮点

Webhook 送信のタイミング・制約

送信機構の有無

- レジストリが Webhook の機能を持っているか
- Webhook が利用できない場合にはレジストリ側を定期ポーリングするなど異なる実装を検討する必要がある

各機能に対する Webhook の有無

- レジストリ側で利用する機能（スキャンなど）の処理結果を Webhook で通知することができるか
- 利用する機能によっては Webhook を利用できないケースもあるため注意

受信側の仕様・制約

受信機構の有無

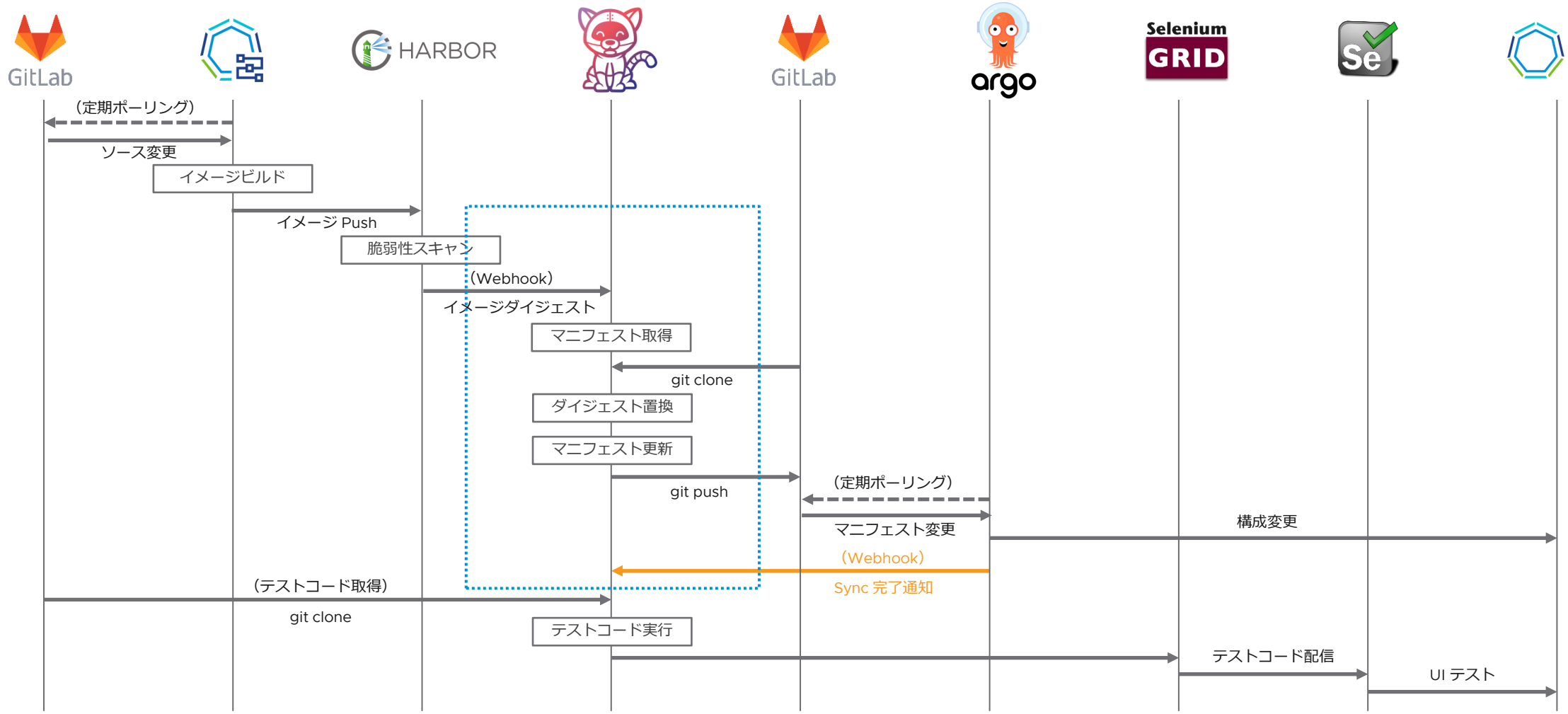
- レジストリ側での処理完了後に、次の処理に移るために Webhook を受信することができるか

ペイロードに対する要件

- 後続処理で必要とされる情報が Webhook のペイロードに含まれているか
- イメージダイジェストやイメージ名など、後続処理の入力を受信できない場合には、Webhook 連携だけでは後続処理に移ることができない

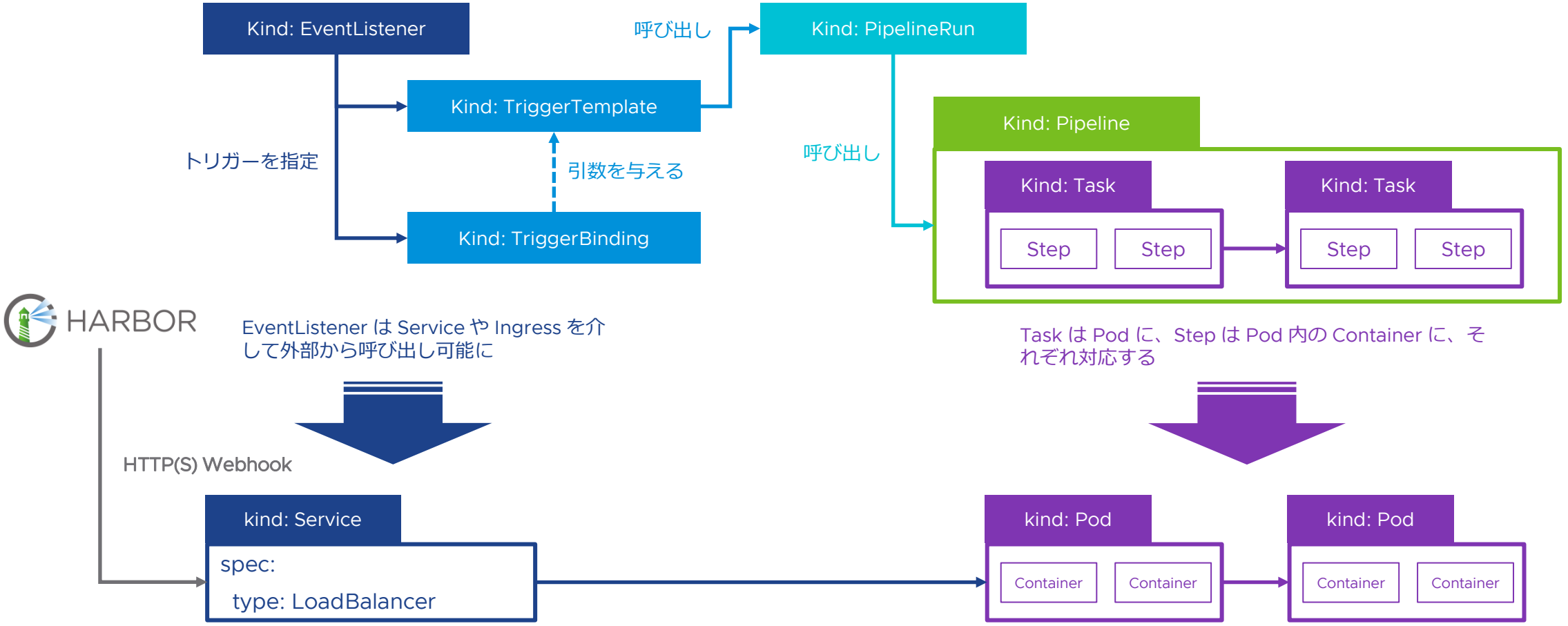
パイプライン構成サンプル

パイプライン処理の定義



Harbor - Tekton - GitLab

Tekton のカスタムリソースで作る CI のコア部分



Harbor - Tekton - GitLab

Pipeline リソースで Git リポジトリの操作を定義

Kind: Pipeline

```
spec:
 params:
  - name: image-tag-id
    type: string
 tasks:
  - name: parse-hash
    taskRef:
      name: get-image-hash
    params:
      - name: body
        value: "$(params.image-tag-id)"
  - name: run-git-commands
    taskRef:
      name: git-cli
    runAfter:
      - parse-hash
  - name: GIT_SCRIPT
        value: |
          echo -e $(tasks.echo-image-tag-id.results.SHASUM)
          # Git Operation as you want
```

Task リソース

- Tekton から実行する処理を定義する
- TektonHub でライブラリ化されている Task を活用することで属人化を避けることが可能

Pipeline リソース

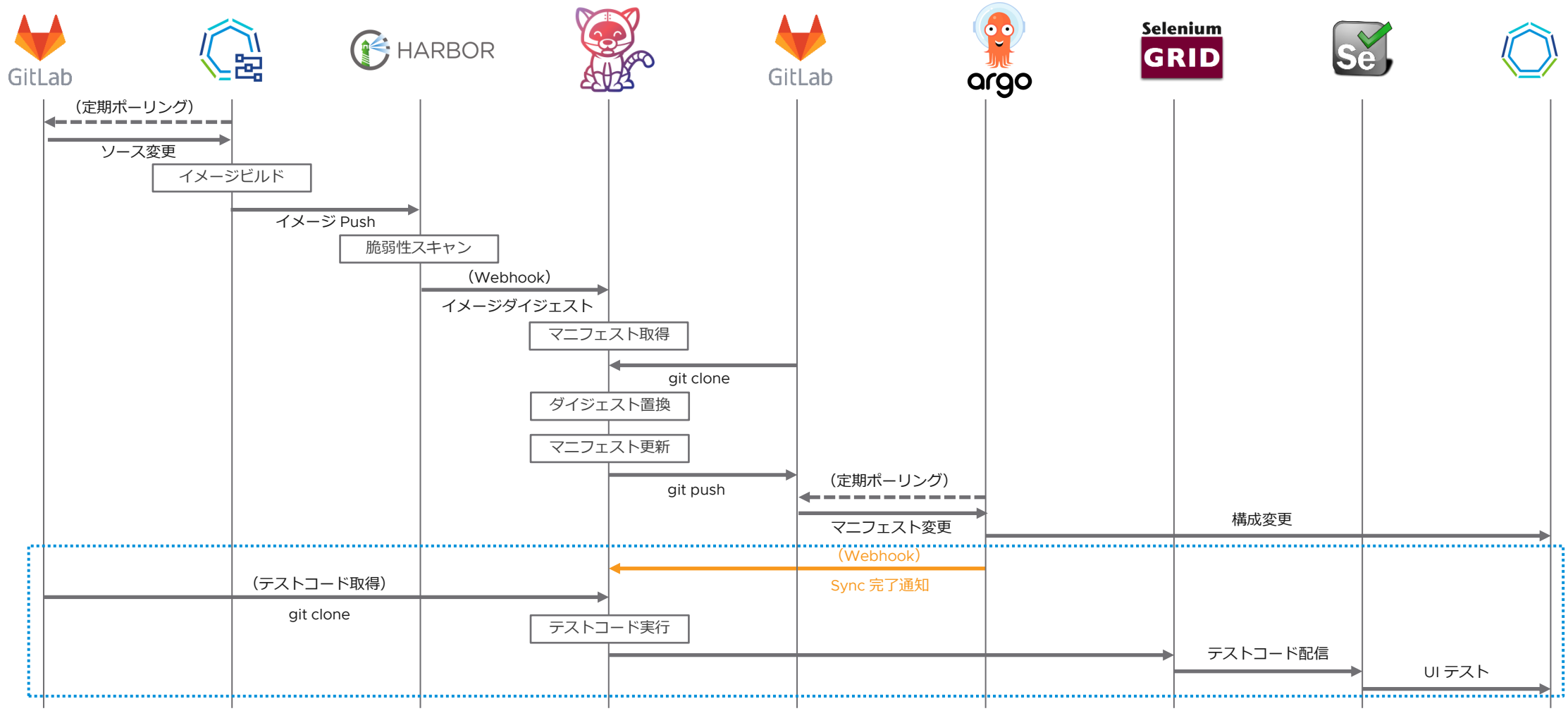
- Task の順番や Task の引数などを管理し、パイプラインにおける処理全体を定義する

PipelineRun リソース

- Pipeline を呼び出すために利用する

パイプライン構成サンプル

テストコードとツールの連携



Tekton - Selenium Grid

E2E テストの実装例（Go 言語）

Test Code: ${CI_REPO}/e2e/go-selenium.go (Package: main)

```
caps := selenium.Capabilities(map[string]interface{}{"browserName": "chrome"})

if webDriver, err = selenium.NewRemote(caps, "http://172.20.56.27:4444/wd/hub"); err != nil {
    fmt.Printf("Failed to open session: %s¥n", err)
    return
}
defer webDriver.Quit()
```

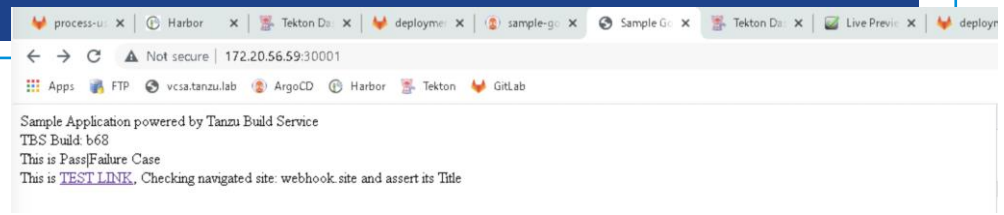
POST /wd/hub/session

```
err = webDriver.Get("http://172.20.56.59:30001")
if err != nil {
    fmt.Printf("Failed to load page: %s¥n", err)
return
}
```

Application Code: ${CI_REPO}/main.go (Package: main)

```
const INDEX = `<!DOCTYPE html>
<html>
[...]
</html>`

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, req *http.Request) {
        fmt.Fprint(w, INDEX)
    })
}
```



Type: LoadBalancer
(172.20.56.27:4444)

Selenium Grid Hub

pod

ClusterIP
(100.96.0.16)

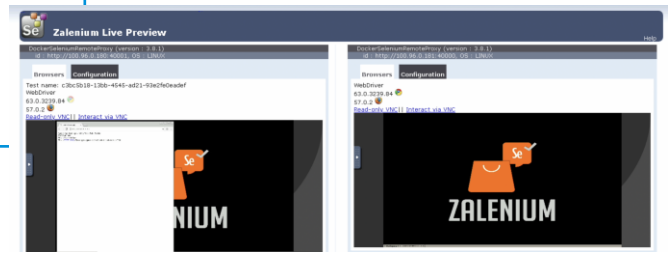
pod pod

ClusterIP
(100.96.0.17)

ClusterIP
(100.96.1.23)

/usr/bin/google-chrome
/usr/bin/chromedriver

Selenium Node(Chrome)



Type: NodePort
(172.20.56.59:30001)

Sample App

pod pod pod

Agenda

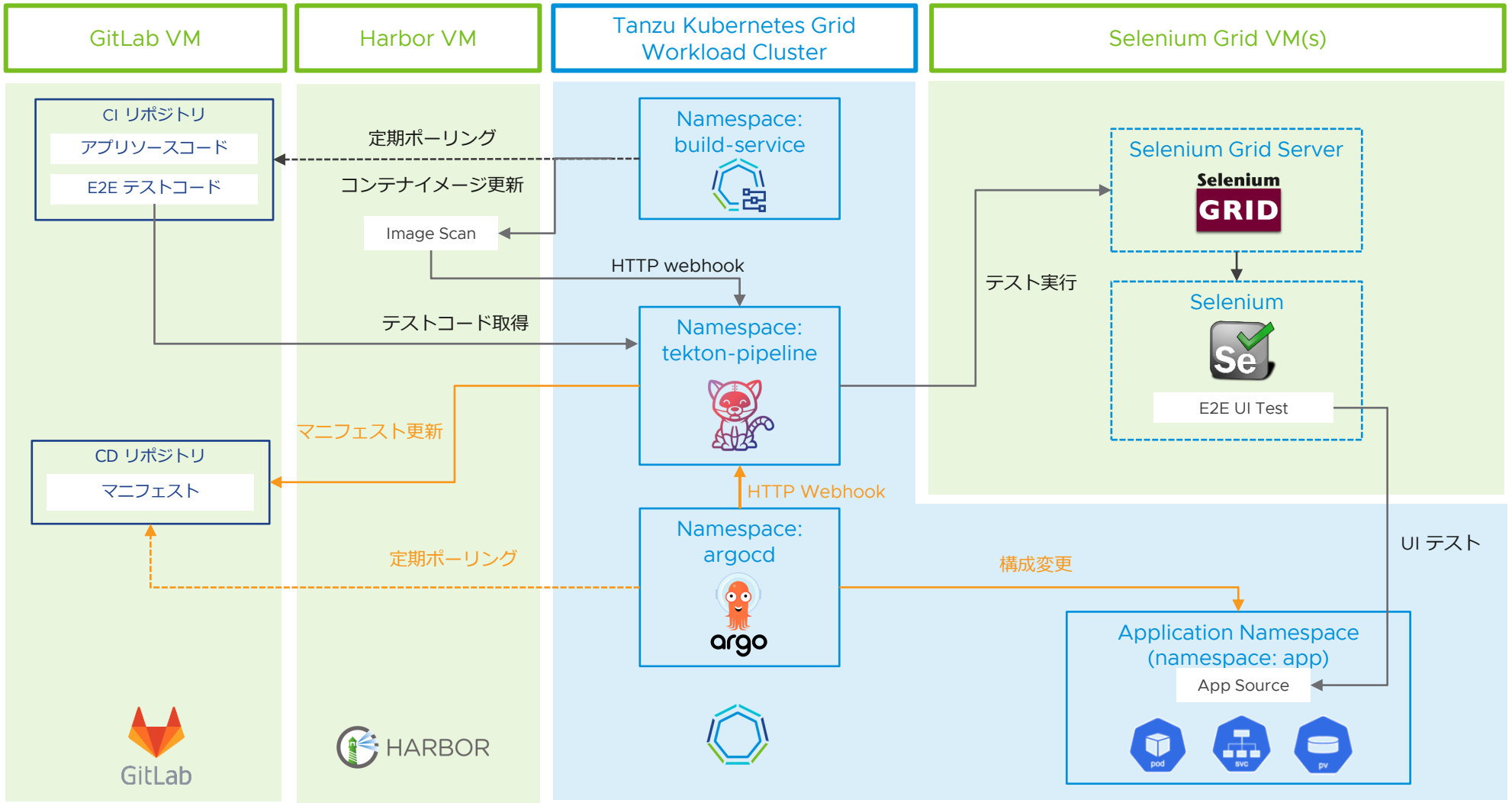
CI/CD が求められる背景

構成例と実装における鍵

デザインパターンと高度化に向けたポイント

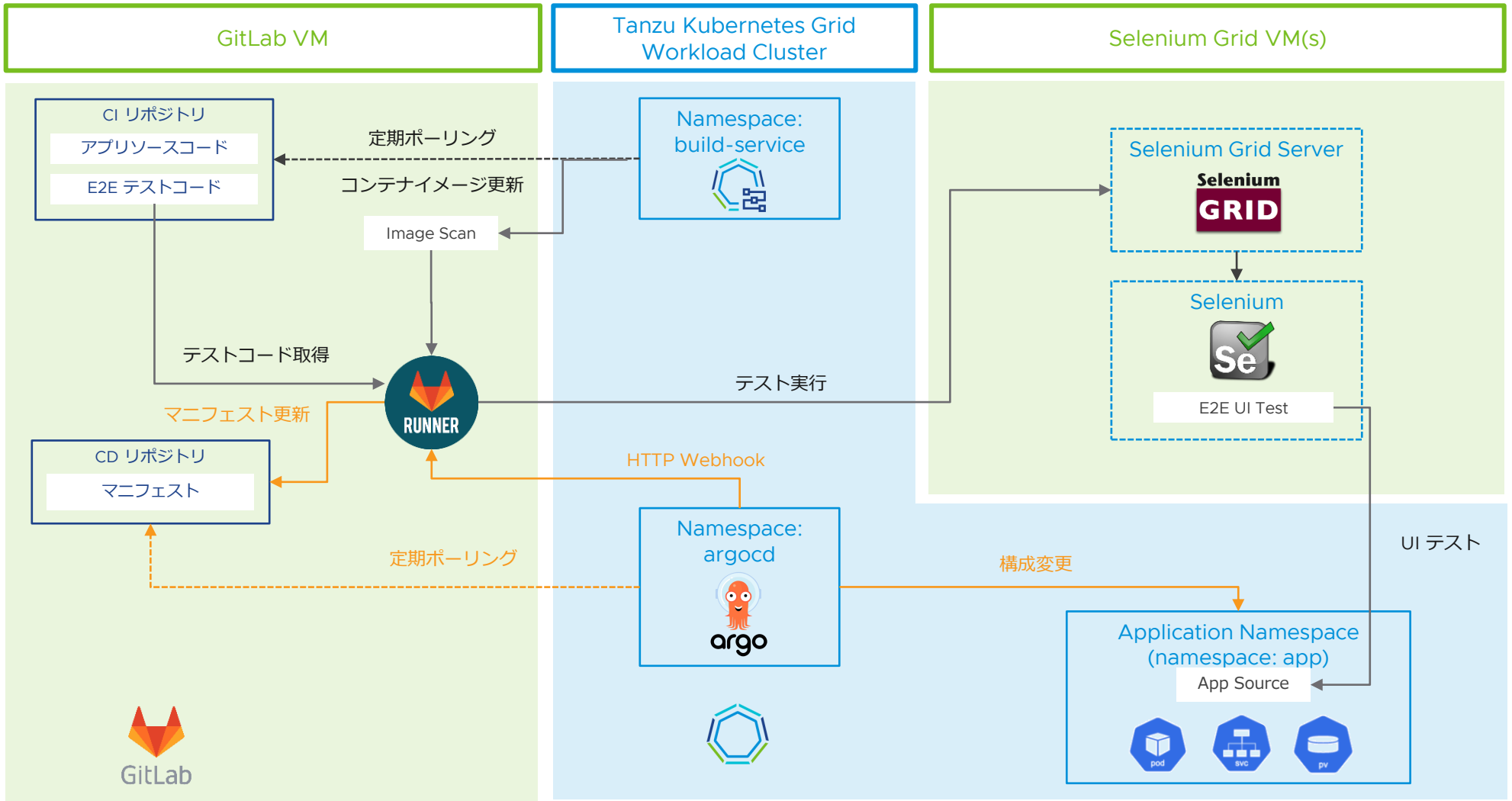
仮想マシンとの併用例

既存システムとの連携を重視する



コンポーネントの統一

GitLab に集約するパターン



パイプライン高度化のポイント

高度化へのアプローチ

コンポーネントの利用機能を拡張

例）

- GitLab
  - ブランチ戦略との連動
- Harbor
  - 脆弱性スキャンに加え、電子署名検証を行う
- VMware Tanzu Build Service
  - TanzuDependencyUpdate の有効化による自動更新

ステップの拡張

例）

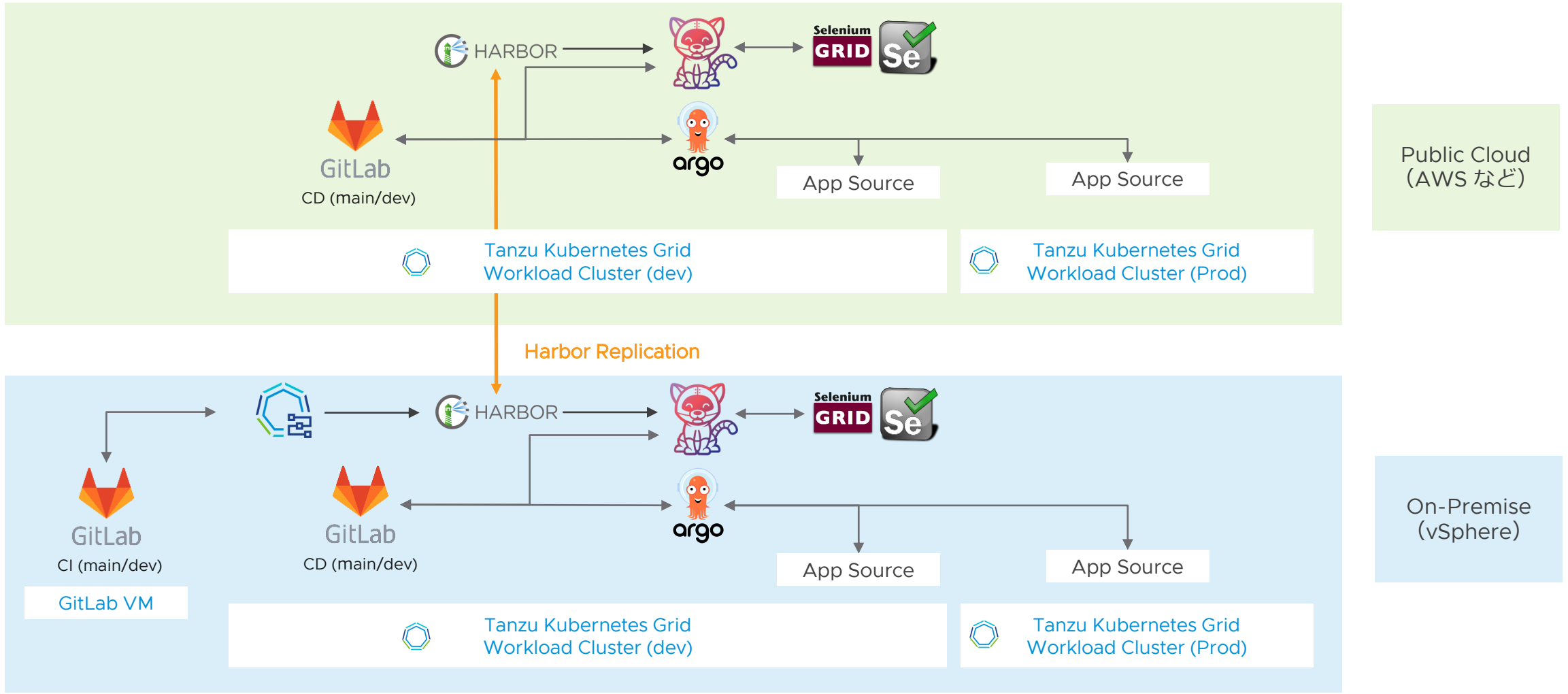
- 静的解析ツール
- 結合テスト/障害テスト
- ドキュメント自動生成

対象の拡張

例）

- 複数の Kubernetes クラスタ
- 複数の基盤（検証/商用）
- 複数のクラウド

ハイブリッドクラウドなパイプライン構成例



Key Takeaways
まとめ

Kubernetes による自動修復
kubectl で一元操作で学習コストを最小限に

マニフェストによるパイプライン定義で保守性とポータビリティを確保

オンプレミスだけなく
クラウド接続も踏まえた
柔軟なパイプライン構成に

本セッション受講の方へのお勧め

MA21157	11/15 15:10～	コンテナ活用を幻想から現実へ！待望の Tanzu Application Platform
MA22151	11/16 14:00～	DevSecOps 入門：開発から運用に至る一貫したコンテナセキュリティ
VI22263	11/16 13:00～	続・VMware ESXi お料理教室 ～ HashiCorp Packer で握った Tanzu Community Edition おにぎり ～

ご清聴
ありがとうございました