

テーマ 2 : CIの実践

レッドハット株式会社
テクニカルセールス本部

1 CI (Continuous Integration)の必要性

2 CIパイプラインの例

3 パイプライン作成の流れ

4 CIハンズオン



CI (Continuous Integration)の必要性

多くの企業のCI/CD取り組み状況

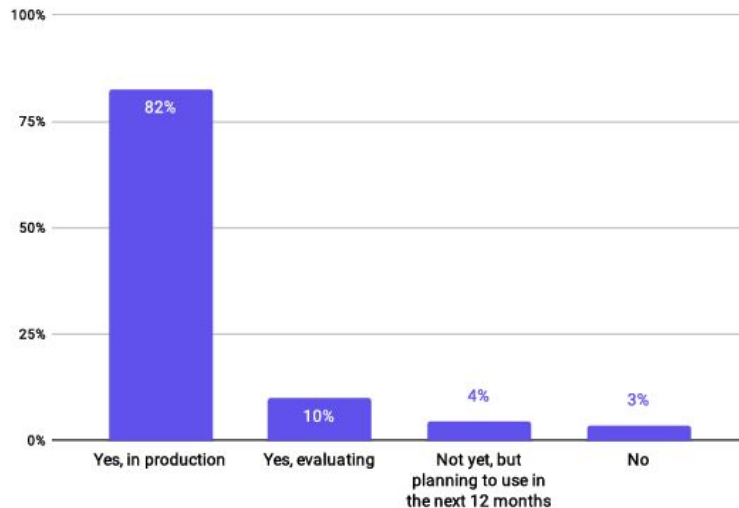
CI/CDパイプラインはあることが当たり前



CNCF(Cloud Native Computing Foundation)がユーザーコミュニティで実施した調査によると、8割を超える企業がCI/CDパイプラインを商用として稼働させている



Do you run Continuous Integration / Continuous Development (CI/CD) pipelines?



CNCF SURVEY 2020

https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf

なぜCI/CDパイプラインを利用するのか？

アプリケーションの開発サイクルを短縮し素早くビジネス価値を創出

CI/CDパイプラインはこれまで手作業で行っていた多くの開発作業や、アプリケーションのデプロイプロセスを自動化します。

Continuous Integration

- アプリケーションのビルド
- 単体/結合テスト
- セキュリティ脆弱性の検知
- コンテナイメージの作成

開発生産性の向上

運用の効率化

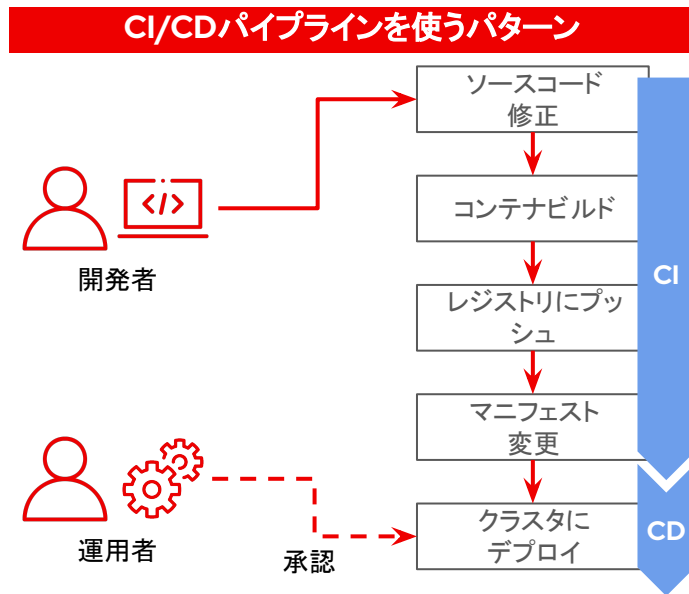
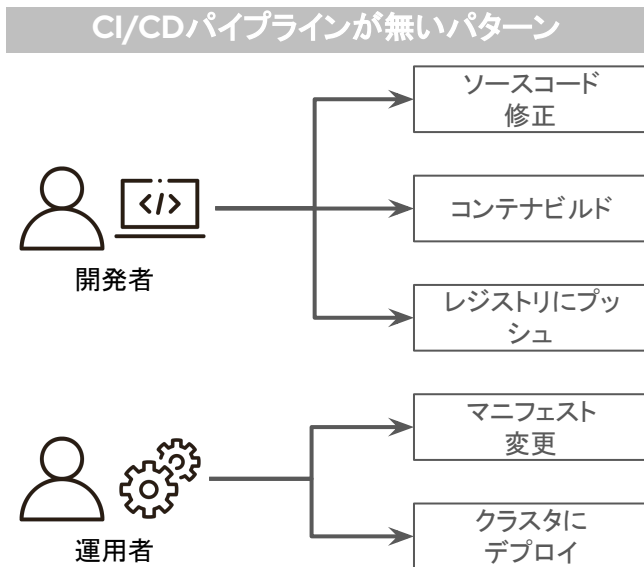
Continuous Delivery

- 開発/商用環境へのデプロイ
- 環境の監視、ロールバック
- 高度なリリース戦略の実現
(カナリアリリース、BlueGreen)

コンテナとCI/CD

コンテナの価値を享受するにはCI/CDが不可欠となる

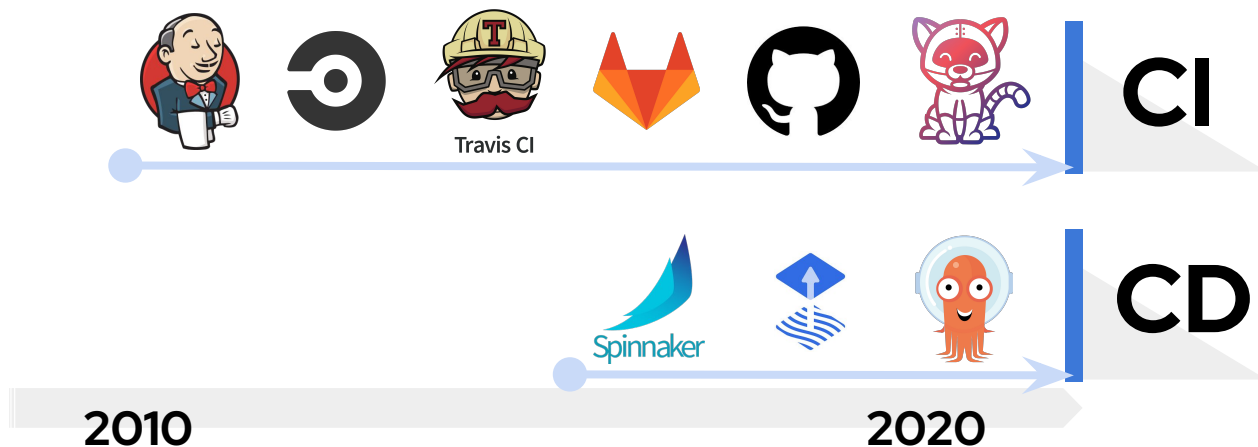
コンテナ化されたシステムの場合、システムに変更を加えるためには新たなコンテナイメージを作成し、それをデプロイする必要があります。CI/CDパイプラインはこれらの作業を自動化します。



CI/CDの進化

CIとCDの区別が明確化

以前はCIとCDの区別は今ほど明確ではなく、CIツールを使ってデプロイまで実施することが多くありました。しかし昨今ではより高度なCDを可能とする専用ツールが登場し、それぞれ別のツールで実施することが一般的になりつつあります。



モダンなCI/CDツールを使うメリット

過去のツールの様々な制約/課題を解消

Jenkinsなど以前から存在しているツールはツール自体に慣れている人が多い一方で、運用が属人化しやすいなどの課題が存在していました。昨今のCI/CDツールはこれらの課題を解決しており、より利用し易いものとなっています。



Traditional CI/CD

- ・CIとCDを一つのツールで実行
- ・専用のサーバーを構築/運用
- ・プラグインの依存関係の解消
- ・並列ジョブ実行のためのslaveサーバーの追加設定
- ・デプロイのロールバックは自前で用意



Cloud Native CI/CD

- ・CIとCDをそれぞれのツールで実行
- ・Operatorによるインストール/管理
- ・ジョブは独立したコンテナで実行されるため依存関係の考慮不要
- ・オンデマンドなスケーリング
- ・デプロイのロールバックはデフォルトで準備

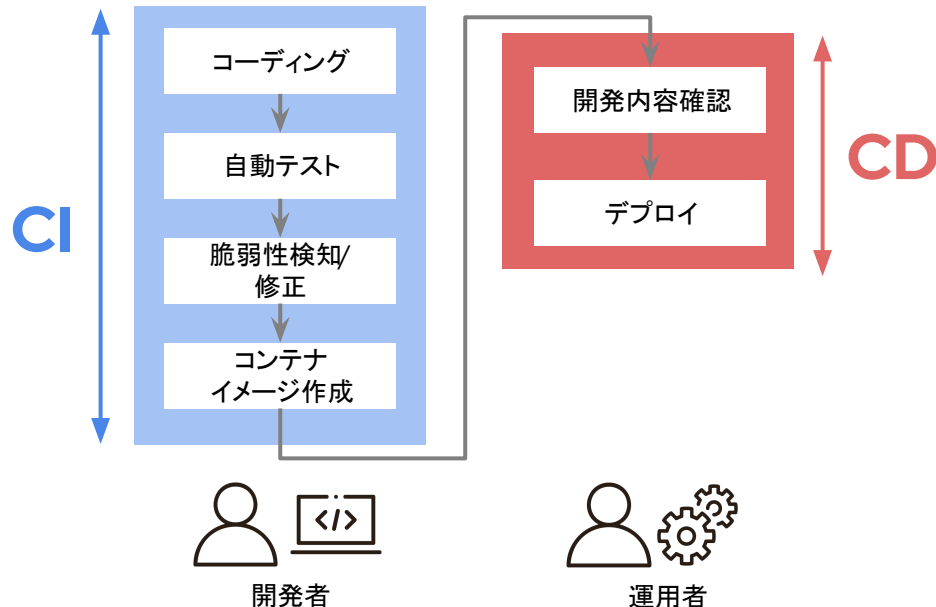


CIとCDの分離がなぜ重要なのか？

チーム間の責任分界点を明確化する

CIとCDを別々のツールを使って実施することで、役割分担が明確化し、各々のタスクに集中することが可能となります

- ・デプロイ可能な成果物を作成するまで
-> **開発チームの役割**
- ・成果物をデプロイして安定運用するまで
-> **運用チームの役割**

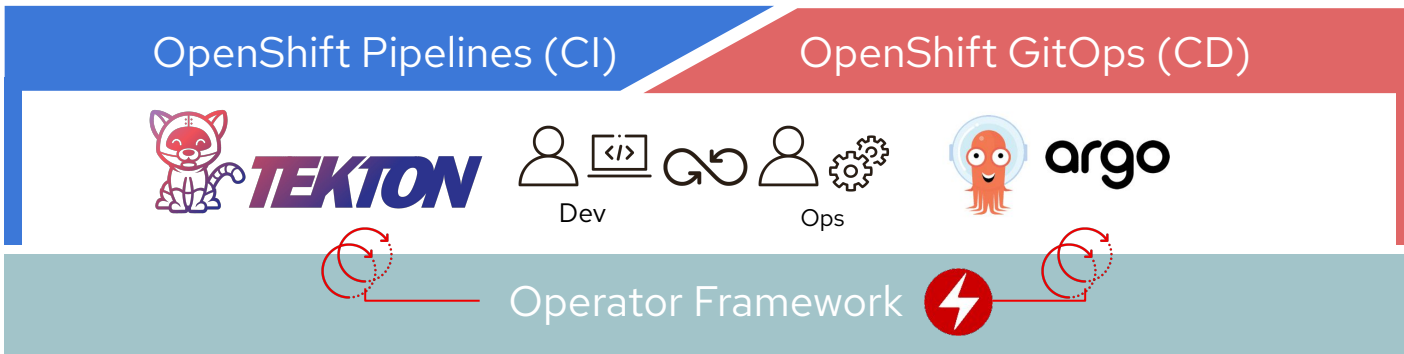


CI パイプライン例

OpenShift Pipelines / OpenShift GitOps

OpenShiftにおけるCI/CD

OpenShiftではCIをOSSのTektonをベースとするOpenShift Pipelinesで、CDをArgo CDをベースとしたOpenShift GitOpsにて実施します。それぞれOperatorを使いインストール/管理されます。



OpenShift Pipelinesの特徴

OpenShiftに最適化されたCIパイプラインの実行

OpenShift PipelinesはOSSのTektonを提供するだけでなく、既存のOpenShiftの機能と統合されています。そのためOpenShift上でシームレスな開発体験を実現できます。



Built for Kubernetes

- Kubernetesネイティブな宣言型 CIツール
- 中央サーバーの管理、プラグインの調整からの脱却



オンデマンドな拡張性

- 独立したコンテナ内で実行・拡張されるパイプライン
- 再現性のある予測可能な結果



セキュアなパイプラインの実行

- OpenShiftのRBACと連携し、アプリケーションのセキュリティポリシーと整合性が取れる安全なパイプライン



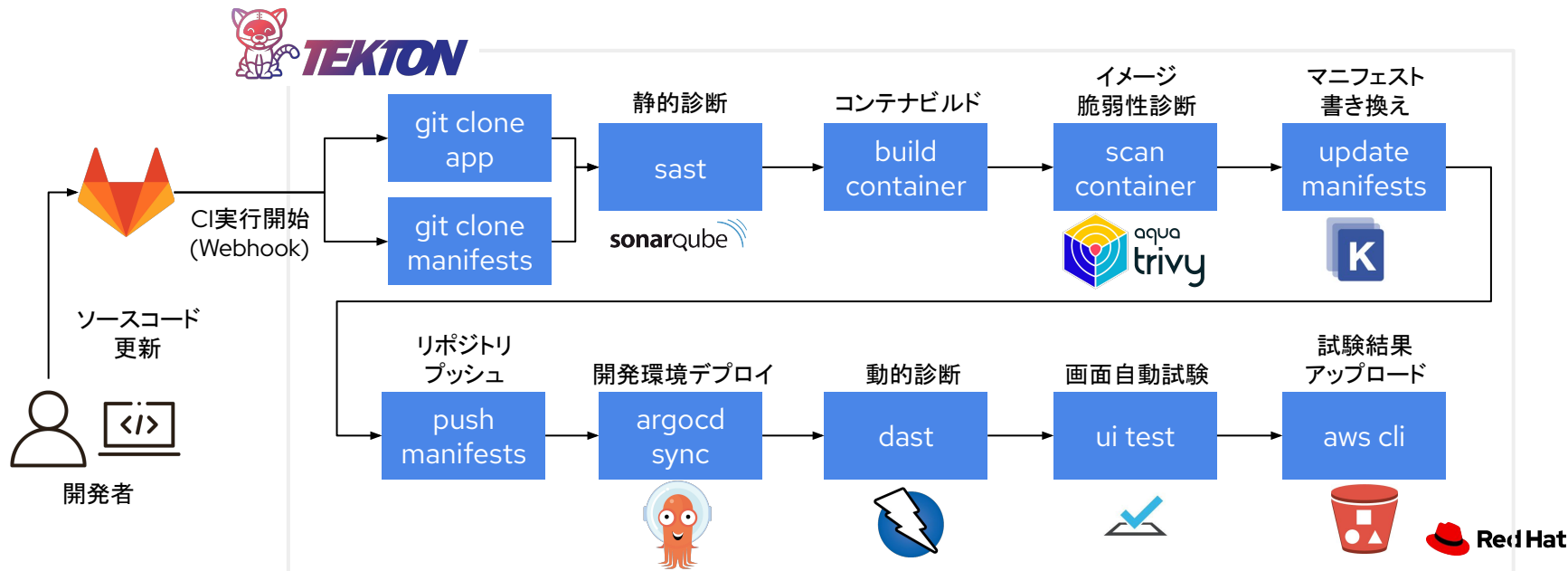
高いユーザービリティ

- OpenShiftのGUIや専用CLIを使ったパイプライン作成
- OperatorHubを使ったインストール・アップグレード
- 既存TaskとPipelineの利用(Tekton Hub)

シナリオ(全体の流れ)

Gitリポジトリを起点とした開発タスク自動化

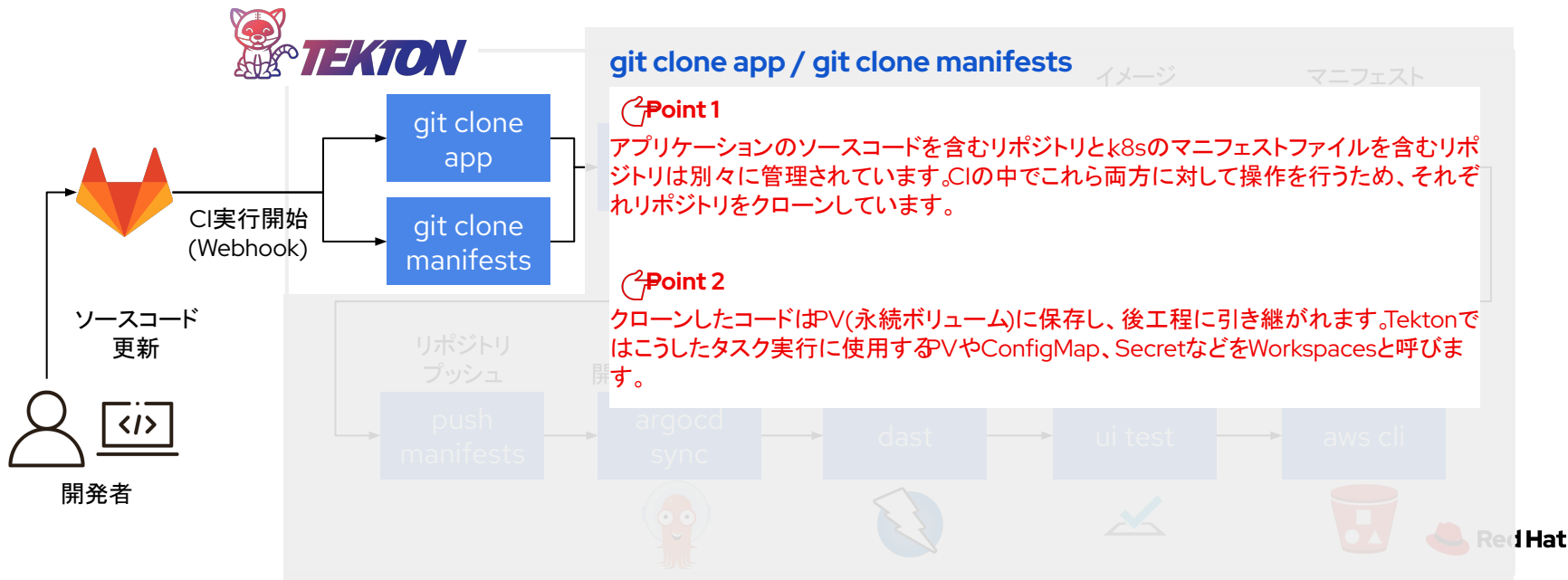
このデモでは健康管理アプリケーションを題材としています。このアプリのソースコード修正を行い、それを契機として実行されるCIパイプラインの一連の流れについてご紹介します。



シナリオ(個別タスク)

Gitクローン

CIパイプラインの対象とするリポジトリからソースコードを取得します。以降のCIの工程はここで取得したコードに対し実行します。



シナリオ(個別タスク)

SAST(静的診断: Static Application Security Testing)

SonarQubeというツールにより、アプリケーションのソースコードにバグや脆弱性が含まれていないか、アプリの実行を伴わない形でチェックします。



シナリオ(個別タスク)

コンテナイメージビルド/脆弱性スキャン

静的診断実施後のソースコードを対象にコンテナイメージをビルドします。ビルドされたコンテナイメージはOpenShiftの統合レジストリにプッシュされた後、イメージに対する脆弱性診断が実施されます。



シナリオ(個別タスク)

マニフェストファイルの更新

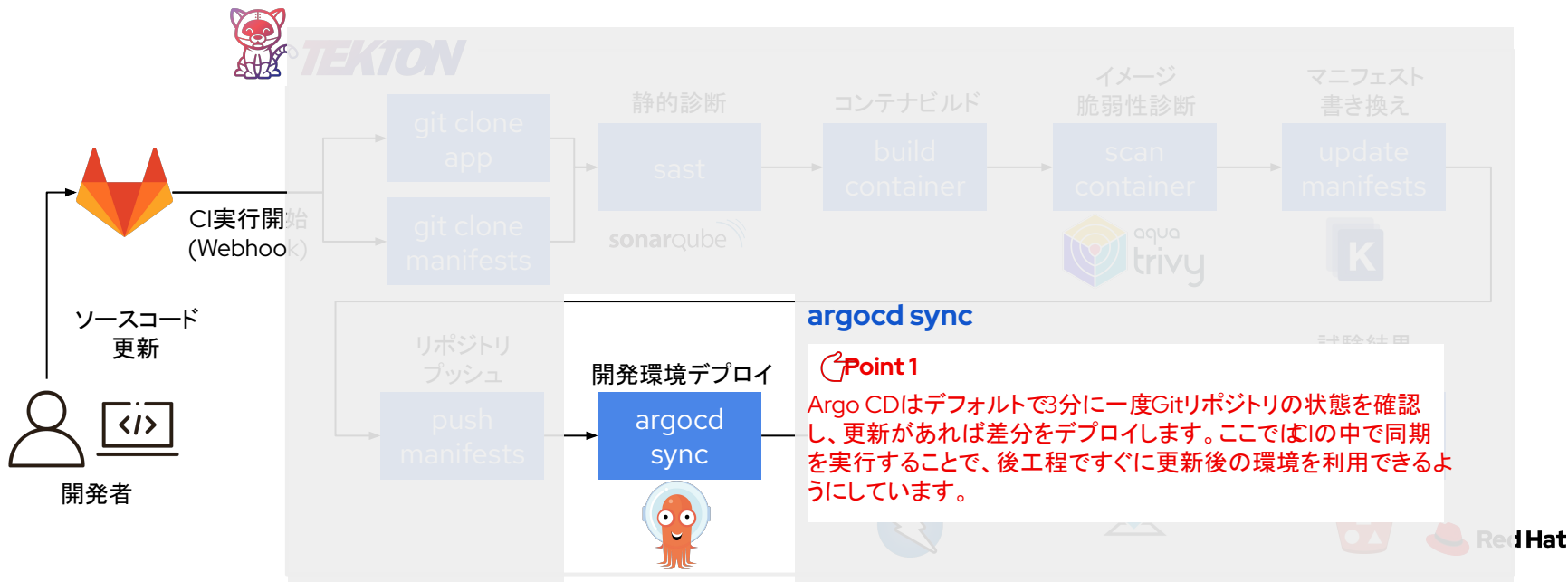
前工程でビルドしたコンテナイメージの情報で既存のマニフェストファイルを更新し、Gitリポジトリにプッシュします。



シナリオ(個別タスク)

開発環境へのデプロイ

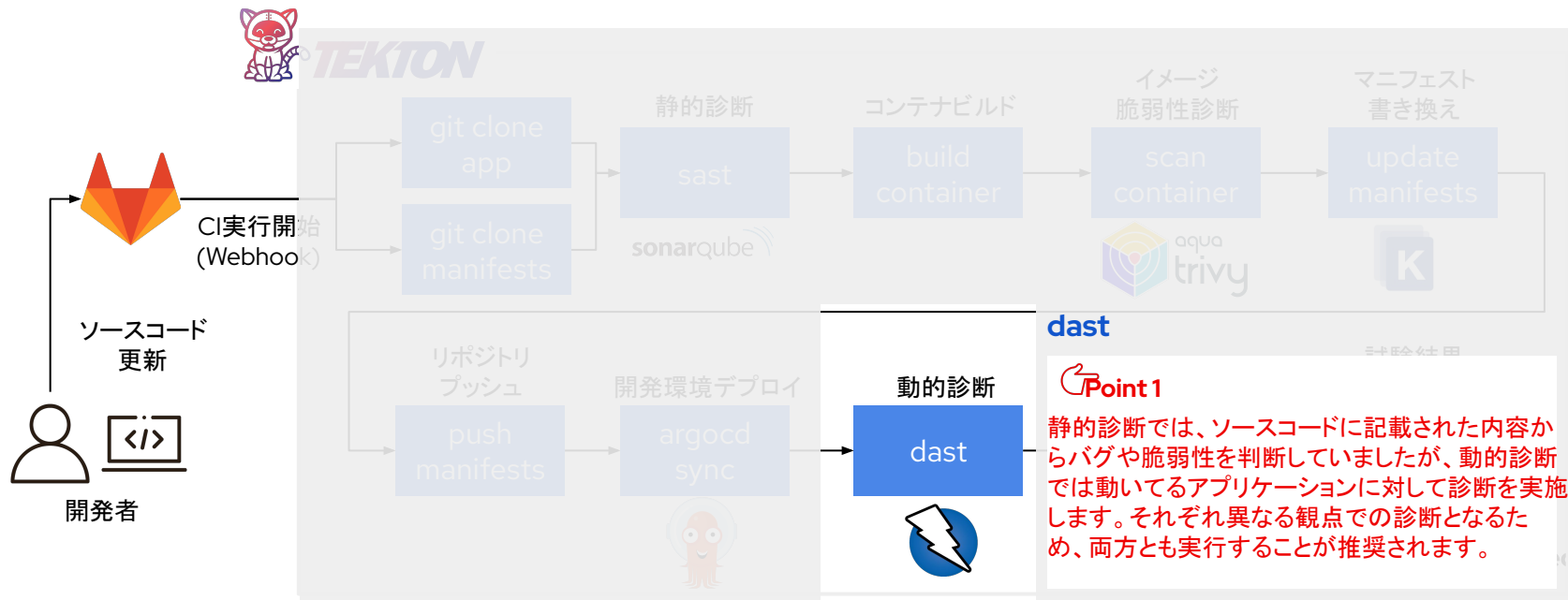
更新されたマニフェストファイルの情報を元に、Argo CDを使い開発環境に新たなコンテナイメージをデプロイします。



シナリオ(個別タスク)

DAST(動的診断: Dynamic Application Security Testing)

OWASP ZAPというツールを用い、開発環境にデプロイされた実行中のアプリケーションに対し疑似的な攻撃を実行して脆弱性が無いかチェックを実施します。



シナリオ(個別タスク)

E2E試験

E2Eテストツールを使い、デプロイ後の開発環境にログインしてUIの確認を実施します。各ページのスクリーンショットを取得し、オブジェクトストレージにアップロードします。



パイプライン作成の流れ

学習のステップ

Step1:



ブランチ戦略を理解する

開発におけるGitリポジトリのブランチ戦略を学びます。一般的によく使用されるブランチ戦略の種類とその特徴、またどんなアクションを契機にCIをスタートするか理解します。

Step2:



パイプライン作成を理解する

Tekton Pipelinesを使用したCIパイプラインの構築方法を学びます。Tekton Pipelinesで使用するカスタムリソースや、yamlでの記述方法のポイントを理解します。

Step3:



パイプラインのトリガーを理解する

Tekton Triggersを使用したCIパイプラインのトリガーの仕組みについて学びます。GitリポジトリからのWebhookを受信し、必要なパラメーターをパイプラインに受け渡して実行する方法を理解します。

Optional:



DevSecOpsを理解する

CIパイプラインの中でアプリケーションセキュリティを担保する方法を学びます。DevSecOpsの基本的な考え方と適用時の注意点について理解します。

ブランチ戦略の種類

CIパイプラインの起点

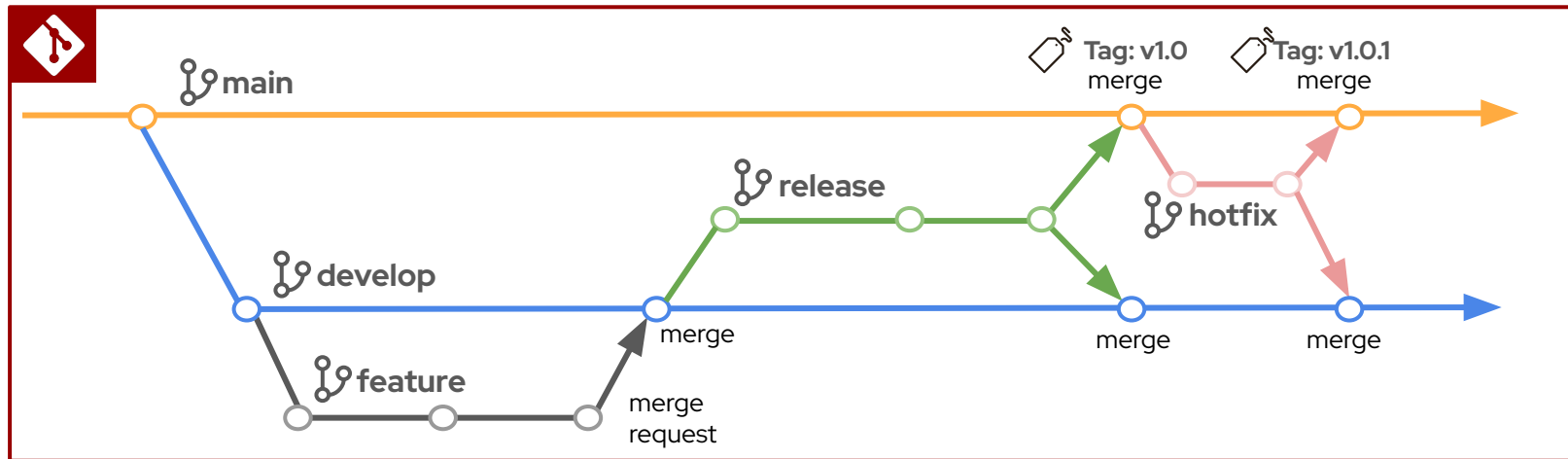
CIパイプラインはコードを管理するGitリポジトリからのWebhookを受け実行されます。そのため自チームの開発プロセスに合うブランチ戦略を採用し、どのブランチに対しどんなアクションを実施した時にCIを開始するのか定義する必要があります。



代表的なブランチ戦略

ブランチ戦略	使用するブランチ	特徴	デメリット
Git Flow	main, develop, feature, release, hotfix	<ul style="list-style-type: none">各環境（開発、ステージング、商用）とブランチが1:1の関係になる中～大規模開発向き	<ul style="list-style-type: none">ブランチ数が多くマージのプロセスが複雑初期学習コストが高い
GitHub Flow	main, feature, (hotfix)	<ul style="list-style-type: none">他のブランチ戦略と比較しブランチの数が少なくシンプル小～中規模開発向き	<ul style="list-style-type: none">リリース時や、リリース後のbugfixがやや複雑
GitLab Flow	main, feature, staging, production, hotfix	<ul style="list-style-type: none">Git Flowに近いが、upstreamファーストでbugfixを行う中～大規模開発向き	<ul style="list-style-type: none">ブランチ数が多くマージのプロセスが複雑初期学習コストが高い

Git Flow



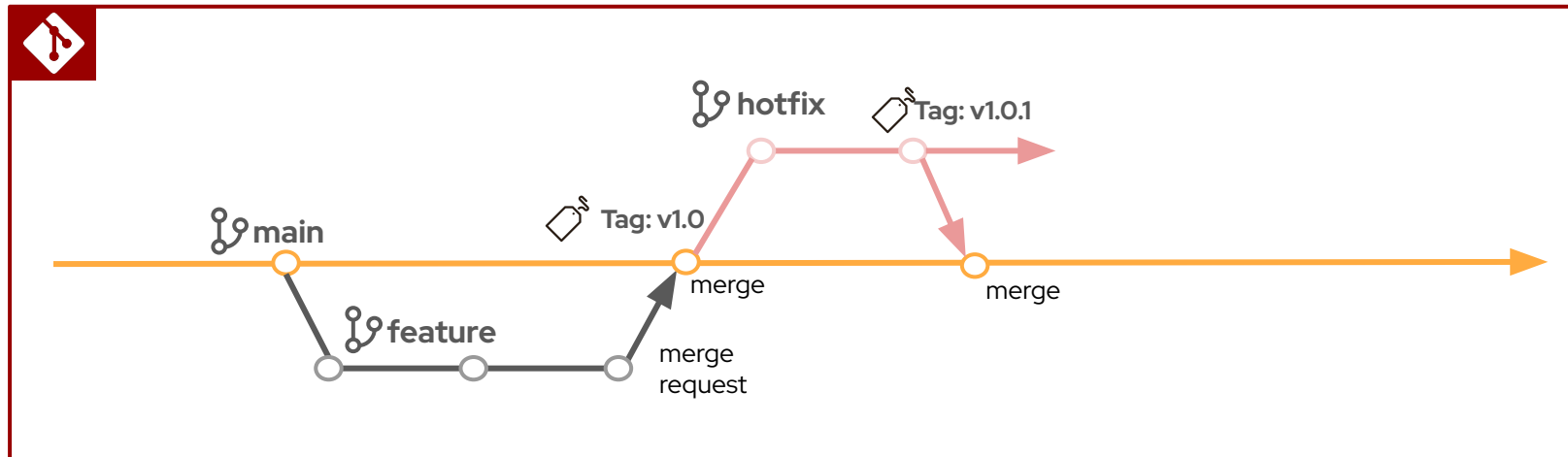
開発の流れ

- mainからdevelopブランチを作成
- developからfeatureブランチを作成し、開発者はfeatureで作業
- featureからdevelopにマージリクエストを作成し、レビュー実施後にマージ
- developからreleaseを作成し、ステージング環境にてテスト実施
- releaseからmainにマージリクエストを投げ、レビュー実施後にマージし、商用環境へリリース
- 商用環境で発生したバグはhotfixにて修正し、mainにマージ

CI実行の契機(例)

- featureからdevelopへのマージリクエストが作成された時
- featureがdevelopにマージされた時
- release / mainに新たなcommitが作成された時

GitHub Flow



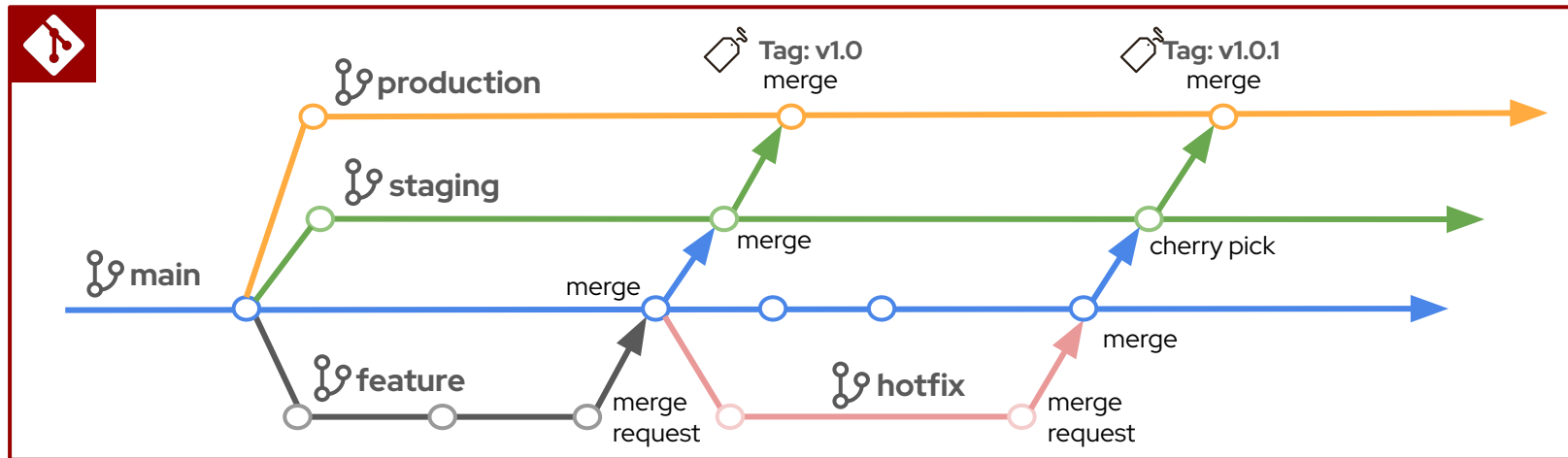
開発の流れ

- ・mainからfeatureブランチを作成し、開発者はfeatureで作業
- ・featureからmainにマージリクエストを作成し、レビュー実施後にマージ
- ・マージ後のmainをステージング環境にリリースしテスト実施
- ・テスト完了後、リリースタグを作成し商用環境へリリース
- ・商用環境で発生したバグはhotfixにて修正し、新たにリリースタグを作成

CI実行の契機(例)

- ・featureからmainへのマージリクエストが作成された時
- ・featureがmainにマージされた時
- ・新たなリリースタグが作成された時
- ・hotfixに新たなcommitが作成された時

GitLab Flow



開発の流れ

- mainからstaging / productionブランチを作成
- mainからfeatureブランチを作成し、開発者はfeatureで作業
- featureからmainにマージリクエストを作成し、レビュー実施後にマージ
- mainをstagingにマージし、ステージング環境でテスト実施
- stagingからproductionにマージリクエストを投げ、レビュー実施後にマージし、商用環境へリリース
- hotfixはmainから作成。stagingに修正を取り込む際はcherry pickを行う

CI実行の契機(例)

- feature / hotfixからmainへのマージリクエストが作成された時
- feature / hotfixがmainにマージされた時
- staging / productionに新たなcommitが作成された時

Webhookの設定

GitLabでの設定例

Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

URL

URL must be percent-encoded if necessary.

Secret token

Use this token to validate received payloads. It is sent with the request in the X-Gitlab-Token HTTP header.

Trigger

☒ **Push events**

URL is triggered by a push to the repository

☐ **Tag push events**

URL is triggered when a new tag is pushed to the repository

☐ **Comments**

URL is triggered when someone adds a comment

☐ **Confidential comments**

URL is triggered when someone adds a comment on a confidential issue

☐ **Issues events**

URL is triggered when an issue is created, updated, closed, or reopened

☐ **Confidential issues events**

URL is triggered when a confidential issue is created, updated, closed, or reopened

☐ **Merge request events**

URL is triggered when a merge request is created, updated, or merged

☐ **Job events**

URL is triggered when the job status changes

☐ **Pipeline events**

URL is triggered when the pipeline status changes

利用するブランチ戦略に応じて適切なタイミングでWebhookが実行されるよう設定を行います。

使用するGitリポジトリの設定画面から、Webhookの送信先や、Secret tokenの設定、トリガーのタイミングを決定します。

Point

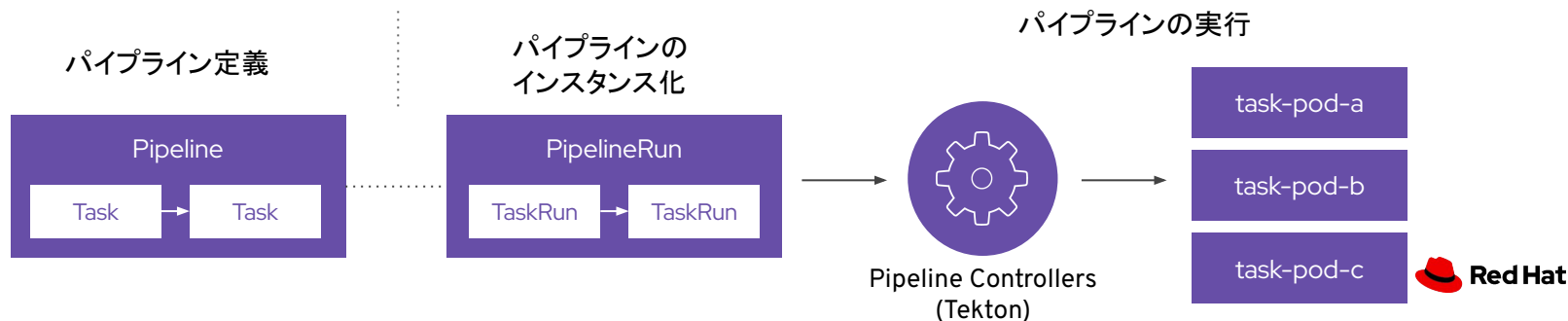
OpenShiftのSecretとしてSecret tokenを登録することで、Webhookの送信元の確認を行うことができます。

外部からの意図しないWebhookを許容しないことでセキュリティの向上に繋がります。

Tekton Pipelinesのカスタムリソース

Tekton PipelinesではTektonが提供するカスタムリソースを使いパイプラインの構成や、その中で実行する一つ一つのアクションを定義します。定義したパイプラインの実行もカスタムリソースにて実施します。

Custom Resource	説明
Task	コンテナ内で実行するコマンド定義の単位であるStepの集合。Task単位でPodが起動しStepを実行する。
TaskRun	TaskRunを作成するとPodを作成し該当Taskの処理が起動する。入力、出力、パラメータを設定する。
Pipeline	複数のTaskを組み合わせて一つのPipelineとして定義する。Taskの実行順序を制御。
PipelineRun	PipelineRun作成によりPipelineを実行する。Pipelineに定義されたTaskに対し順番にTaskRunを作成する。



Task

TaskはPipelineを構成する一つ一つのアクションを定義します。TaskはTaskRun実行時に一つのPodとして実行されます。

spec.steps

Task内で実行するアクションの最小単位をstepとして記載します。各stepは指定したコンテナイメージによりscriptを実行します。stepはリストとして複数記載することができ、その場合は一つのPodに複数コンテナが起動します。

spec.params

step実行時に利用可能なパラメーターを設定します。ここではデフォルト値のみ定義しており、実際に利用する値はTaskRun/PipelineRun等で設定します。

spec.workspaces

Task間でデータを受け渡すためのPVや、Secret、ConfigMapを設定します。実際にどのPVやSecretを利用するかについてはTaskRun/PipelineRun等で設定します。

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: aws-cli
spec:
  description: >-
    This task performs operations on Amazon Web Services resources
    using aws.
  steps:
    - name: awscli
      image: docker.io/amazon/aws-cli:latest
      script: "${(params.SCRIPT)}"
      args:
        - "${(params.ARGS)}"
  params:
    - name: SCRIPT
      description: The AWS script to run
      type: string
      default: "aws $@"
    - name: ARGS
      description: AWS cli arguments to be passed
      type: array
      default: ["help"]
  workspaces:
    - name: source
      optional: true
    - name: secrets
      optional: true
      mountPath: /root/.aws
```

デフォルトだとparamsで設定された"aws help"コマンドが実行されます

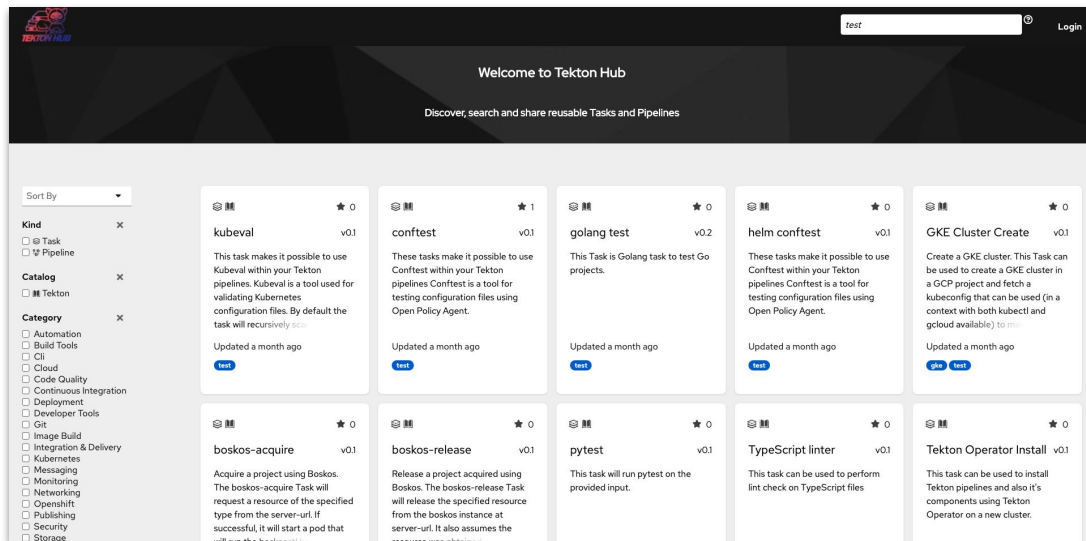
標準部品を使ったパイプラインの構築

Tekton HubからTaskをインストール

Tekton ProjectではCIパイプラインの中で実行される一般的なTaskをTekton Hubにて公開しています。これらのTaskを組み合わせることで最低限の労力でCIパイプラインを構築することが可能となります。

公開されている Task の例:

- maven: Javaのビルド、テスト、など
- conftest: マニフェストファイルのテスト
- go lang test: Go言語の単体テスト
- pytest: Pythonの単体テスト
- sonarqube scanner: アプリケーション静的診断
- Send message to Slack Channel: Slackへの通知



Pipeline

複数のTaskの実行順序を組み合わせPipelineを定義します。
またTask間で共有するworkspacesやparamsを設定します。

spec.workspaces

Taskで利用するworkspaceをリストとして定義します。

spec.params

Taskに受け渡し可能なPipeline内部で共有できるパラメーターを定義します。実際に設定する値はPipelineRunで設定します。

spec.tasks

Pipelineの中で実行するTaskを定義します。taskRefで利用するTaskを、runAfterでどのTaskの後に実行されるかを定義します。

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: sample-pipeline
spec:
  workspaces:
    - name: shared-workspace
  params:
    - name: git-url
      type: string
    - name: git-revision
      type: string
  tasks:
    - name: git-clone-app
      taskRef:
        name: git-clone
      params:
        - name: url
          value: ${params.git-url}
        - name: revision
          value: ${params.git-revision}
      workspaces:
        - name: output
          workspace: shared-workspace
    - name: scan-app
      taskRef:
        name: sonarqube-scanner
      runAfter:
        - git-clone-app
```

各Taskにworkspaceを割り当てる場合、このworkspace名を使います

利用するTaskの参照

各パラメーターのnameはTask定義時のnameを設定します

nameにはTask定義時のworkspace名を設定します

どのtaskの後にこのtaskを実行するか定義します

Pipeline作成のTips

利用するTaskのパラメーターとWorkspaceに注目する

多くのTaskは実行時にパラメーターや作業ボリュームを必要とするため、Pipelineの中でもそれらの設定を行う必要があります。またPipelineの中で実行するTaskの中でどのWorkspaceを共有にするかも重要なポイントとなります。

Task (git-clone)

```
spec:
  workspaces:
    - name: output ●
      description: The git repo will be cloned onto the volume backing this workspace
  params:
    - name: url ●
      description: git url to clone
      type: string
    - name: revision ●
      description: git revision to checkout (branch, tag, sha, ref...)
      type: string
      default: ""
    - name: refspec
      description: (optional) git refspec to fetch before checking out revision
      default: ""
```

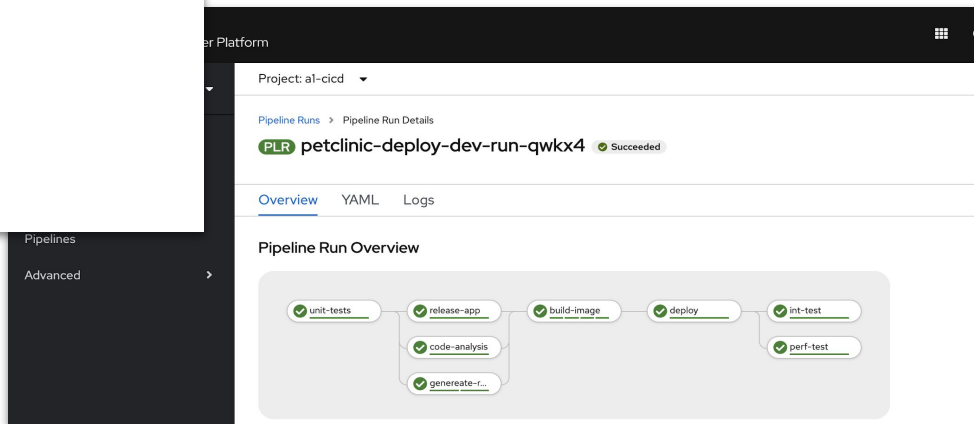
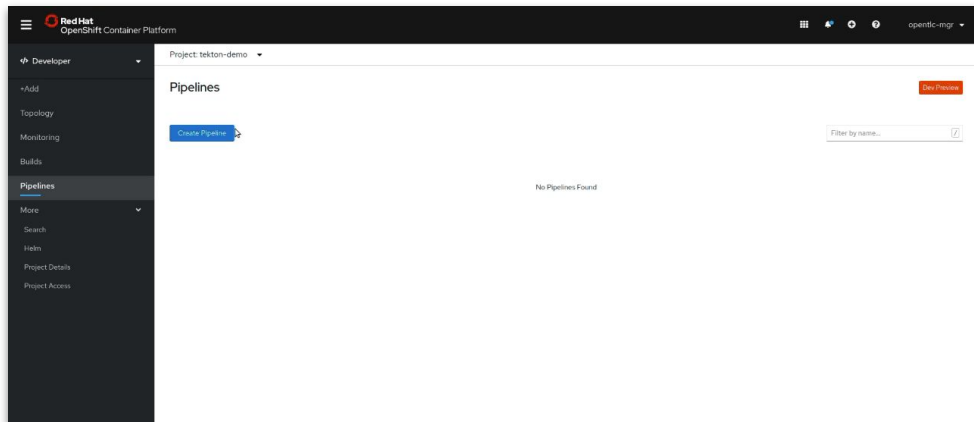
Pipeline

```
spec:
  workspaces:
    - name: shared-workspace ●
  params:
    - name: git-url ●
      type: string
    - name: git-revision ●
      type: string
  tasks:
    - name: git-clone-app
      taskRef:
        name: git-clone
      params:
        - name: url
          value: $(params.git-url) ●
        - name: revision
          value: $(params.git-revision) ●
      workspaces:
        - name: output
          workspace: shared-workspace ●
```


OpenShiftコンソールとの統合

GUIによるパイプライン構築

TektonはOpenShiftのコンソール画面と統合されており、開発者が利用する画面からパイプラインを構築し、実行状況を確認/管理することができます。



PipelineRun

定義したPipelineを実行します。Pipelineで設定したworkspacesやparamsに値を提供します。

spec.pipelineRef

このPipelineRunで参照するPipelineを設定します。

spec.params

Pipelineで設定したparamsに入力する値を設定します。

spec.workspaces

Pipelineで定義したworkspaceをPVCと紐付けます。SecretやConfigMapをworkspaceとして利用する場合も同様に利用するオブジェクトを指定します。

```
apiVersion: tekton.dev/v1beta1
```

```
kind: PipelineRun
```

```
metadata:
```

```
  generateName: sample-pipelinerun-
```

```
spec:
```

```
  pipelineRef:
```

```
    name: sample-pipeline
```

```
  params:
```

```
    - name: git-url
```

```
      value: 'https://gitlab.com/jpishikawa/rhf2021-cicd-app'
```

```
    - name: git-revision
```

```
      value: 'master'
```

```
  workspaces:
```

```
    - name: shared-workspace
```

```
      persistentVolumeClaim:
```

```
        claimName: shared-workspace
```

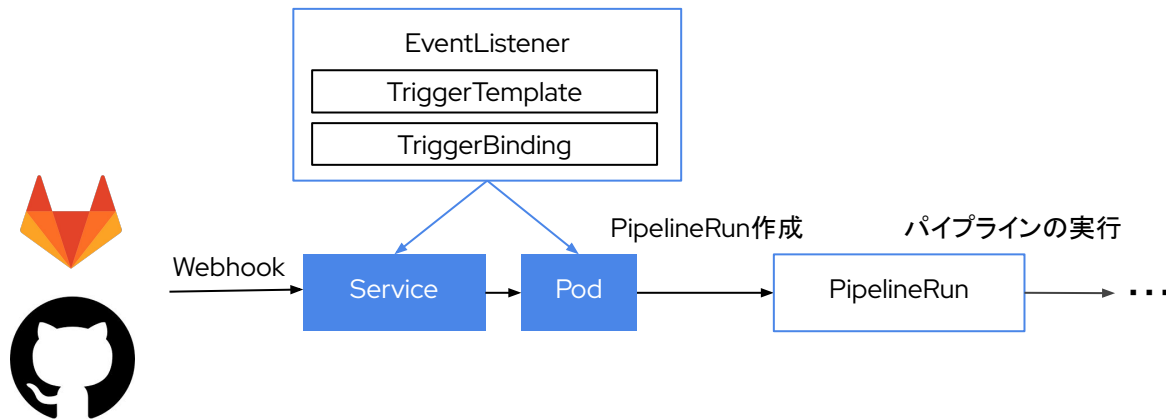
generateNameを設定することで、PipelineRunを作成する際の名前被りを防止できます

PVCを指定する以外にもvolumeClaimTemplateを設定し、PipelineRun実行ごとにPVを作成することもできます

Tekton Triggersのカスタムリソース

開発の流れの中でPipelineを実行するにはPipelineRunの作成を自動化する必要があります。Tekton TriggersはGitリポジトリからのWebhookを受信し、PipelineRunを作成することでCIを実行します。

Custom Resource	説明
TriggerTemplate	Webhookを受け取った際に自動作成するリソース(PipelineRun)を定義する。
TriggerBinding	Webhookで受け取ったJSONからPipelineRunに渡すパラメータを定義する。
EventListener	Webhookの受け口 (Service+Pod) の作成と、使用するTriggerTemplate、TriggerBindingの指定。



TriggerTemplate

EventListenerでのWebhook受信時に作成するリソースを定義します。

spec.params

spec.resourcetemplates内で利用するパラメーター名を定義します。パラメーターの値として何を設定するかはTriggerBindingで設定します。

spec.resourcetemplates

作成するリソース(PipelineRun)をリストで定義します。TriggerTemplateで定義するPipelineRunの場合、Webhookで受信したJSONデータの値をparamsとして設定できます。

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerTemplate
metadata:
  name: sample-pipeline-template
spec:
  params:
    - name: git-app-rev
      description: The git revision
      default: master
    - name: git-app-url
      description: The git repository url
  resourcetemplates:
    - apiVersion: tekton.dev/v1beta1
      kind: PipelineRun
      metadata:
        generateName: sample-pipelinerun-
      spec:
        pipelineRef:
          name: sample-pipeline
        params:
          - name: git-url
            value: $(tt.params.git-app-url)
          - name: git-revision
            value: $(tt.params.git-app-rev)
        workspaces:
          - name: shared-workspace
            persistentVolumeClaim:
              claimName: shared-workspace
```

TriggerBindingで設定するパラメーター名と同じ名前を使用する

params以外はPipelineRunを個別に作成する場合と同じ

TriggerBinding

EventListenerで受信したWebhookに含まれるデータのうちパラメーターとして利用するものを設定します。

spec.params

各パラメーターのvalueとしてWebhookに含まれる値を設定します。

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerBinding
metadata:
  name: sample-trigger-binding
spec:
  params:
    - name: git-app-rev
      value: $(body.checkout_sha)
    - name: git-app-url
      value: $(body.repository.git_http_url)
```

Webhookの設定値がパラメーターとして活用される

Point

Webhookに設定される値は、使用するGitリポジトリやイベントの種類により変わるため、以下のサイトなどを確認して適切に設定を行いましょう

GitLab Webhook Events

https://docs.gitlab.com/ee/user/project/integrations/webhook_events.html

GitHub Webhook events and payloads

<https://docs.github.com/ja/developers/webhooks-and-events/webhooks/webhook-events-and-payloads>

GitLabのPushイベントの場合のWebhook設定値

```
{
  "object_kind": "push",
  "event_name": "push",
  "before": "95790bf891e76fee5e1747ab589903a6a1f80f22",
  "after": "da1560886d4f094c3e6c9ef40349f7d38b5d27d7",
  "ref": "refs/heads/master",
  "checkout_sha": "da1560886d4f094c3e6c9ef40349f7d38b5d27d7",
  "user_id": 4,
  "user_name": "John Smith",
  "user_username": "jsmith",
  "user_email": "john@example.com",
  ...
}
```

EventListener

利用するTriggerBindingとTriggerTemplateを設定します。またWebhook受信のためのServiceとPodを作成します。

spec.serviceAccountName

EventListenerのPodを動かすServiceAccountを指定します。

spec.triggers

あらかじめ作成したTriggerBindingとTriggerTemplateを指定します。TriggerBindingについてはリストとして複数設定することができます。



Point

spec.triggers以下のinterceptorsでは、Webhookを受信した場合でも条件に合わない場合はPipelineを実行しないよう設定することができます。

これを使うことで、特定のブランチ間でのマージリクエスト時のみを実行するなど詳細な設定を行うことができます。

Configuring Interceptors

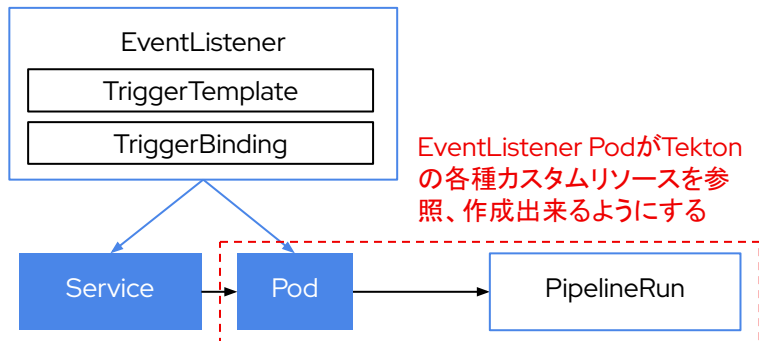
<https://tekton.dev/vault/triggers-main/interceptors/>

```
apiVersion: triggers.tekton.dev/v1alpha1
kind: EventListener
metadata:
  name: sample-event-listener
spec:
  serviceAccountName: sample-sa
  triggers:
    - bindings:
        - ref: sample-trigger-binding
      template:
        ref: sample-pipeline-template
      interceptors:
        - ref:
            name: "gitlab"
            kind: ClusterInterceptor
          params:
            - name: "secretRef"
              value:
                secretName: gitlab-webhook
                secretKey: secretkey
            - name: "eventTypes"
              value: ["Push Hook"]
```

“secretRef”ではWebhook設定時のSecret tokenが設定されているか、“eventTypes”ではWebhookがPushイベントを契機に実行されたかを確認しています

EventListenerのServiceAccount

EventListenerにより作成されたPodは、Webhookを受信すると、TriggerTemplateに設定されたリソースを作成します。そのためリソース作成のためにPodが必要な権限を使用できるようにRBACの設定を行います。



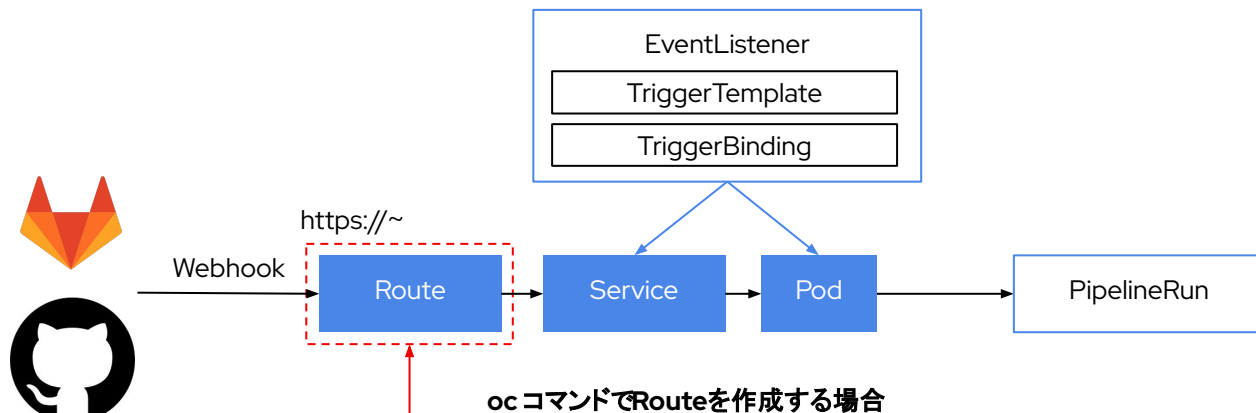
```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: sample-sa
secrets:
  - name: gitlab-webhook
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: triggers-eventlistener-binding
subjects:
- kind: ServiceAccount
  name: sample-sa
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tekton-triggers-eventlistener-roles
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: triggers-eventlistener-clusterbinding
subjects:
- kind: ServiceAccount
  name: sample-sa
  namespace: app-develop
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tekton-triggers-eventlistener-clusterroles
  
```

Tektonインストール時に必要なClusterRoleが作成されるため、それを使用するSAにバインドする

EventListenerの外部公開

EventListenerにより作成されたServiceは、type: ClusterIP となるためデフォルトでは外部公開されません。そのため外部からのWebhookを受信するには、IngressやRouteを作成する必要があります。



oc コマンドでRouteを作成する場合

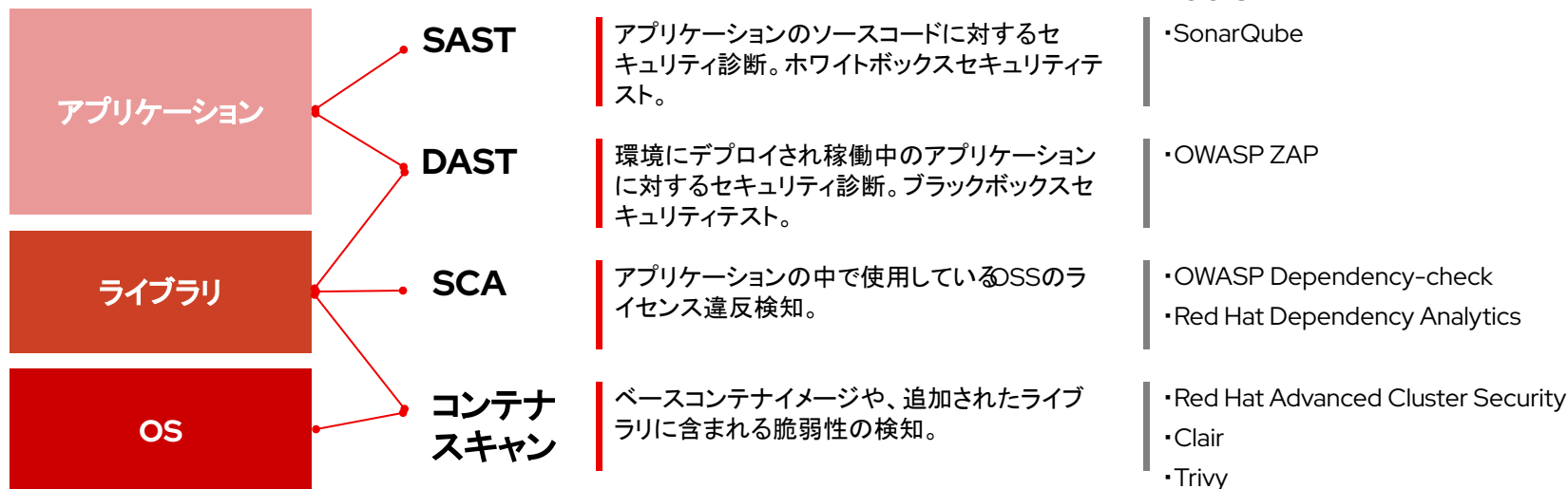
```
# oc expose svc el-sample-event-listener
```

EventListenerが作成するServiceは
"el-{EventListener名}"という名前で作成されます。
上記コマンドでRouteを作成することで、外部向けのエンド
ポイントを公開できます。

DevSecOps

開発段階でのセキュリティ対策

CIパイプラインの中にセキュリティ診断ツールを組み込み、開発早期において脆弱性を特定・修正していくプラクティスをDevSecOpsと呼びます。DevSecOpsを実践することで商用環境でバグや脆弱性が見つかるリスクを低減し、対応に必要なインパクトを最小化できます。



Red Hat Dependency Analytics

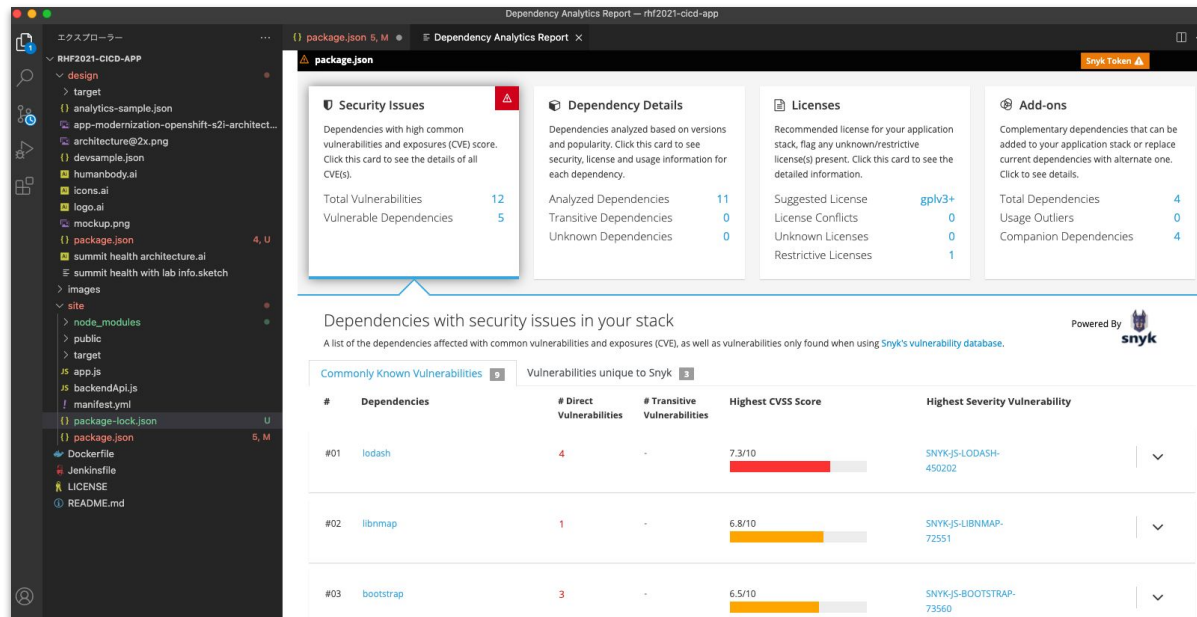
エディターで利用可能なセキュリティプラグイン

Red Hat Dependency Analyticsは、OpenShift Dev SpacesやVSCodeなどのコードエディターにインストールできるプラグインツールです。Snyk社が提供している脆弱性データベースの情報を元に、利用するパッケージの脆弱性情報や、ライセンス情報を検知します。

- 様々な言語に対応
(Java, Node, Python, Go)
- 利用するパッケージのCVE 情報を表示
- 脆弱性が検知されたパッケージについては修正をリコmend
- 脆弱性情報をレポート形式で表示

Dependency Analytics

<https://marketplace.visualstudio.com/items?itemName=redhat.fabric8-analytics>



The screenshot shows the Red Hat Dependency Analytics report in VS Code. The report is titled 'Dependency Analytics Report - rnf2021-cicd-app' and displays the following sections:

- Security Issues:** Dependencies with high common vulnerabilities and exposures (CVE) score. Total Vulnerabilities: 12, Vulnerable Dependencies: 5.
- Dependency Details:** Dependencies analyzed based on versions and popularity. Analyzed Dependencies: 11, Transitive Dependencies: 0, Unknown Dependencies: 0.
- Licenses:** Recommended license for your application stack. Suggested License: gplv3+, License Conflicts: 0, Unknown Licenses: 0, Restrictive Licenses: 1.
- Add-ons:** Complementary dependencies that can be added to your application stack or replace current dependencies with alternate one. Total Dependencies: 4, Usage Outliers: 0, Companion Dependencies: 4.

Below these sections, the report lists 'Dependencies with security issues in your stack' with a table of vulnerabilities:

#	Dependencies	# Direct Vulnerabilities	# Transitive Vulnerabilities	Highest CVSS Score	Highest Severity Vulnerability
#01	lodash	4	-	7.3/10	SNYK-JS-Lodash-450202
#02	libnpm	1	-	6.8/10	SNYK-JS-LIBNPM-72551
#03	bootstrap	3	-	6.5/10	SNYK-JS-BOOTSTRAP-73560

RHACSのCIパイプラインへの組み込み

脆弱性検知とポリシーチェック

RHACS(Red Hat Advanced Cluster Security for Kubernetes)は機能の一部としてCLIを提供しています。これをCIパイプラインに組み込むことで、ビルドしたイメージの脆弱性診断や、ポリシーチェックを自動化できます。

- 脆弱性(CVE)+ポリシーチェック
- デフォルトポリシーだけでなくカスタマイズに対応
- 診断結果の詳細をRHACSコンソールから確認可能

```
# roxctl image check -e=$ROX_CENTRAL_ADDRESS --token-file=acs-token --output=table --image=registry.redhat.io/rhel8/nodejs-16:latest
...
```

```
Policy check results for image: registry.redhat.io/rhel8/nodejs-16:latest
(TOTAL: 2, LOW: 2, MEDIUM: 0, HIGH: 0, CRITICAL: 0)
```

Policy check result: Image registry: node:10.16.0/nodejs:10.latest
 (TOTAL: 2, LOW: 2, MEDIUM: 0, HIGH: 0, CRITICAL: 0)

ポリシーの説明			違反理由	修正方法	
POLICY	SEVERITY	BREAKS BUILD	DESCRIPTION	VIOLATION	REMEDATION
Latest tag	LOW	-	Alert on deployments with images using tag 'latest'	- Image has tag 'latest'	Consider moving to semantic versioning based on code releases (semver.org) or using the first 12 characters of the source control SHA. This will allow you to tie the Docker image to the code.
Red Hat Package Manager in Image	LOW	-	Alert on deployments with components of the Red Hat/Fedora/CentOS package management system.	- Image includes component 'dnf' (version 4.7.0-4.el8.noarch) - Image includes component 'rpm' (version 4.14.3-19.el8.x86_64) - Image includes component 'yum' (version 4.7.0-4.el8.noarch)	Run <code>rpm -e --nodeps \$(rpm -qa '*rpm*' '*dnf*' '*libsolv*' '*hawkey*' '*yum*')</code> in the image build for production containers.

```
WARN: A total of 2 policies have been violated
```

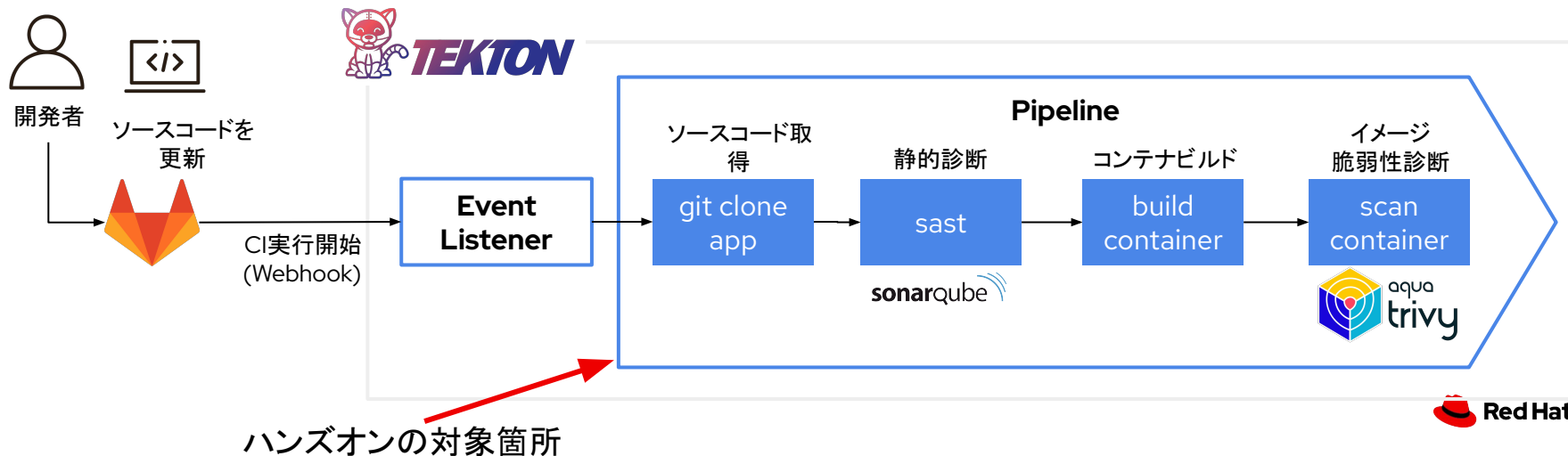
CI ハンズオン

ハンズオンシナリオ

CI パイプラインの一部を作成し、コンテナイメージに含まれるパッケージの脆弱性診断を自動化します。

ハンズオンのリンクはこちらになります。 <https://gitlab.com/openshift-starter-kit/ci-practice>

- 1. ハンズオン環境準備と2. CIパイプラインの作成が対象となります。
- 3. Event Listenerの作成は、時間の余裕があれば、チャレンジしてください。



参考図書



Tektonについて更に詳しく知りたい、またCIだけでなくArgo CDによるCDの方法についても併せて勉強したいという方はこちらの書籍がお勧めです。

KubernetesにおけるCI/CD実装ガイド

本書は、Kubernetesを活用したアプリケーション開発やそのリリースサイクルを自動化するためのノウハウについて解説しています。1冊全体を通してKubernetes 環境におけるアプリケーションライフサイクルの構築を順を追って体験します。継続的インテグレーションと継続的デリバリーによって、「いかに少ない労力で開発プロセスを運用し続けるか」という課題に取り組みます。従来の開発プロセスからクラウドネイティブな開発プロセスへの変化を理解し、実践することにより、運用負担の軽減や迅速なサービス展開が可能となります。(出版社サイトより抜粋)

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make

Red Hat a trusted adviser to the Fortune 500.

 linkedin.com/company/red-hat

 youtube.com/user/RedHatVideos

 facebook.com/redhatinc

 twitter.com/RedHat