

OpenShiftを用いた CI/CD開発体験

レッドハット株式会社
テクニカルセールス本部

アジェンダ

1 DevSecOpsの全体像について

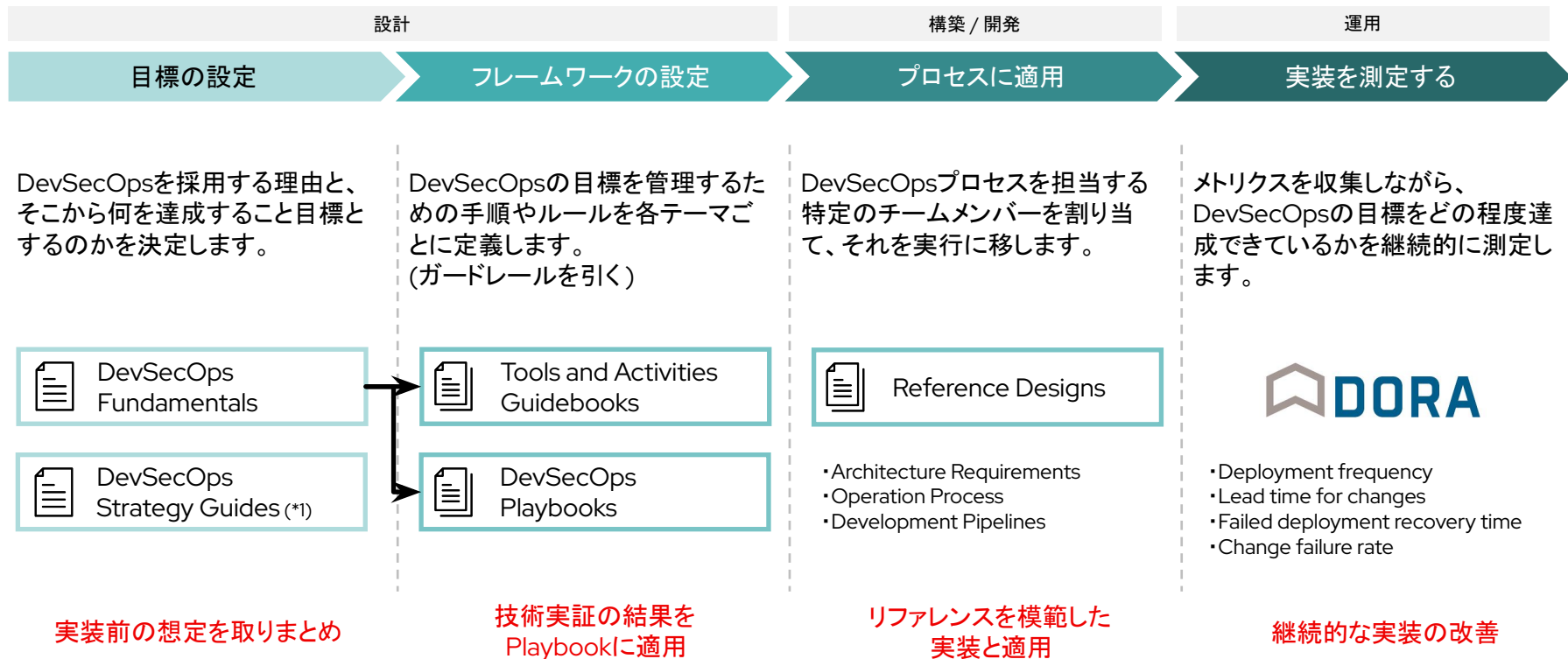
2 CIパイプラインの作成

3 CDパイプラインの作成

アプリケーションの セキュリティプラクティス

DoD – DevSecOps

DevSecOpsの適用ステップ (DoD)



DevSecOps Fundamentalsの内容



DevSecOps
Fundamentals

▶ DevSecOps Fundamentalsは、
アプリケーション/システム毎でなく基準書に取り入れるべき内容

概要

- 1. Introduction
- 2. Agile
- 3. Software Supply Chains
- 4. DevSecOps

5. DevSecOps Lifecycle

- 6. DevSecOps Platform
- 7. Current and Envisioned DevSecOps
Software Factory Reference Designs
- 8. Deployment Types
- 9. Minimal DevSecOps Tools Map
- 10. Measuring Success with Performance Metrics
- 11. DevSecOps Next Steps

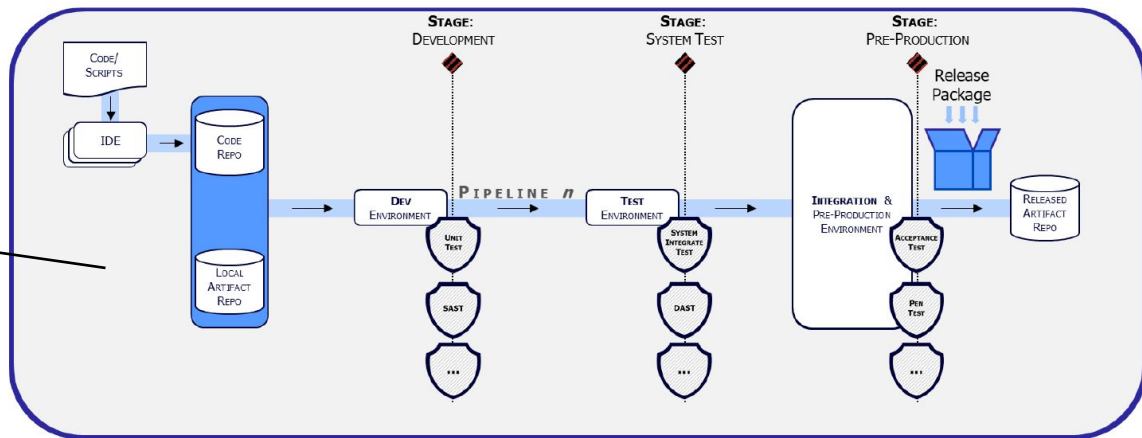
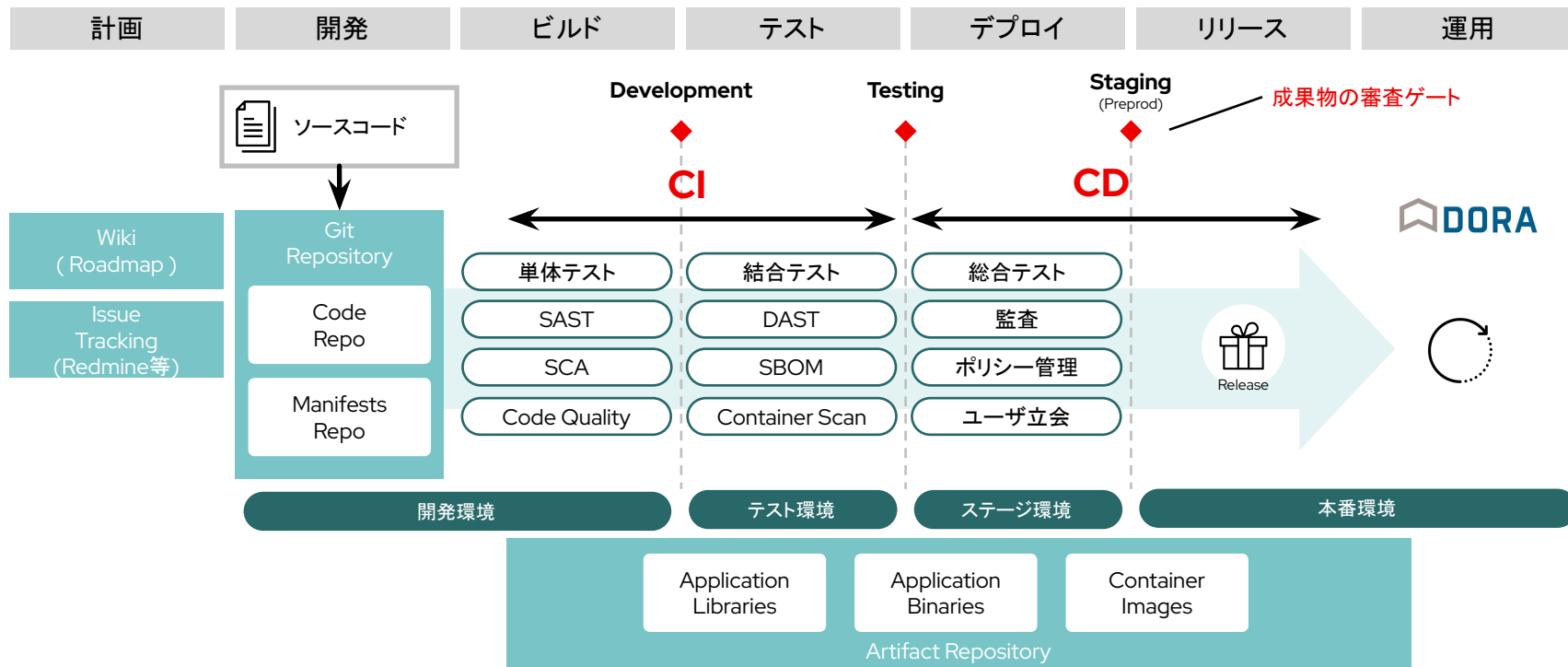


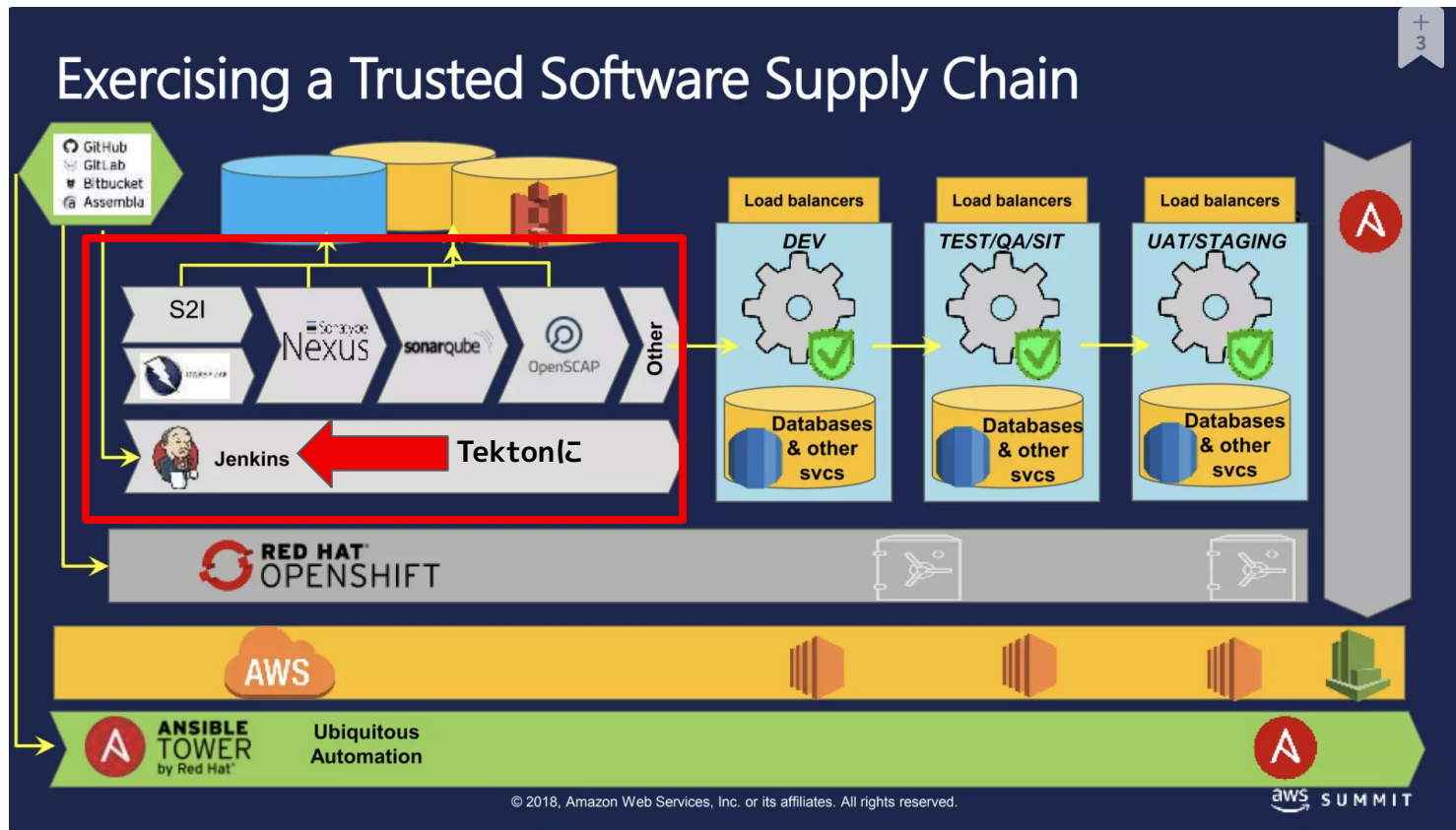
Figure 6 Notional expansion of a DevSecOps software factory with illustrative list of tests

DevSecOpsのライフサイクル



CI ハンズオン

参考にするCIパイプライン: US Navy C2C24



出展: Modernizing Software Development in the US Navy:

<https://www.slideshare.net/AmazonWebServices/modernizing-software-development-in-the-us-navy>

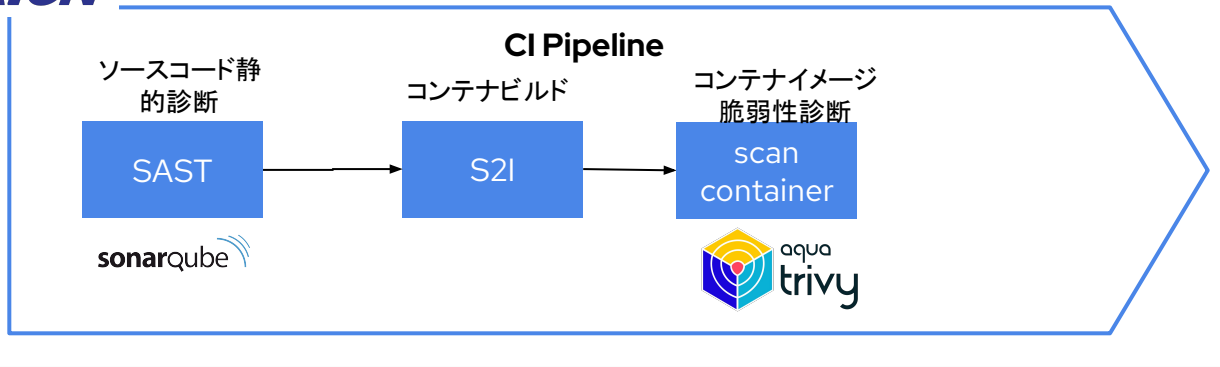
CIハンズオンシナリオ

C2C24と同等のCIパイプラインを作成します。

- ・ハンズオンの流れ
 - ・CIツールのTektonでPipelineを作成
 - ・Pipelineに、以下のタスクを追加
 - ・ソースコード静的診断: sonarQubeを使用
 - ・コンテナのビルド: S2Iを使用
 - ・コンテナイメージの脆弱性診断: aqua trivyを使用

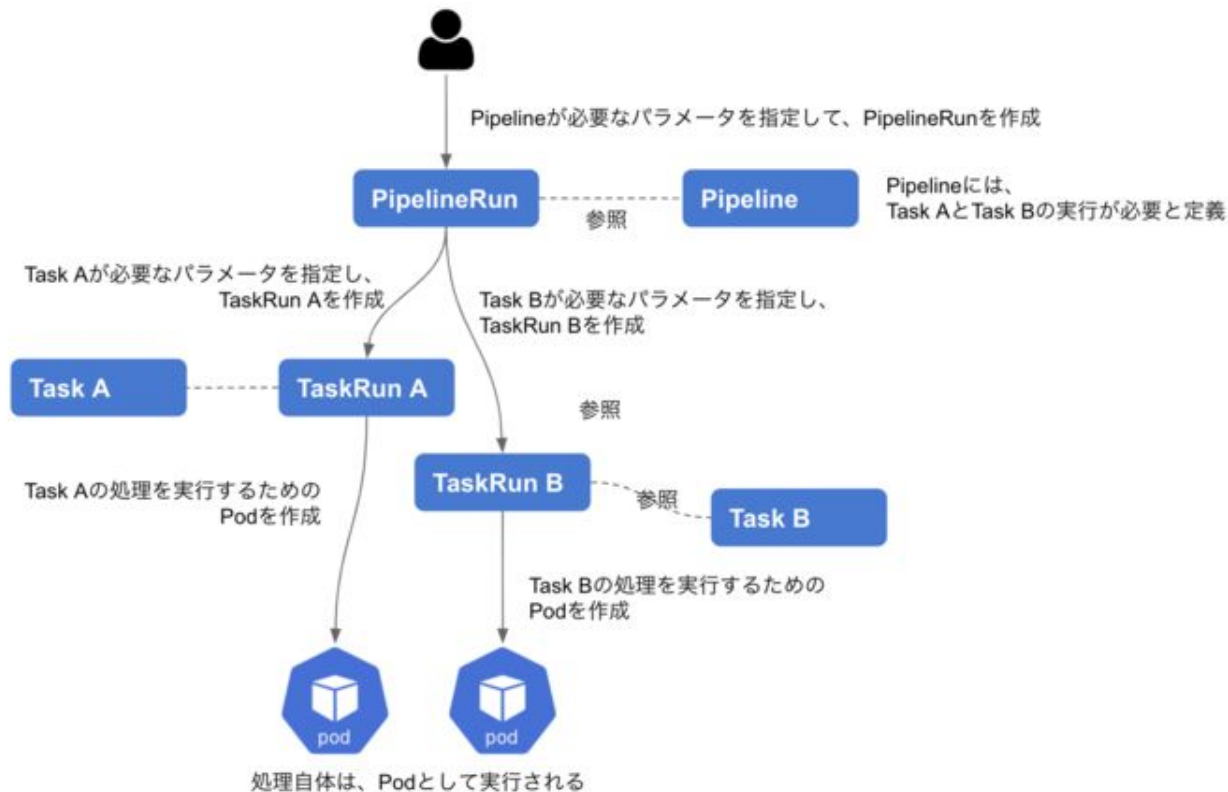


TEKTON



Tektonのパイプラインの考え方 (1/2)

最初は、どうしても慣れが必要です。



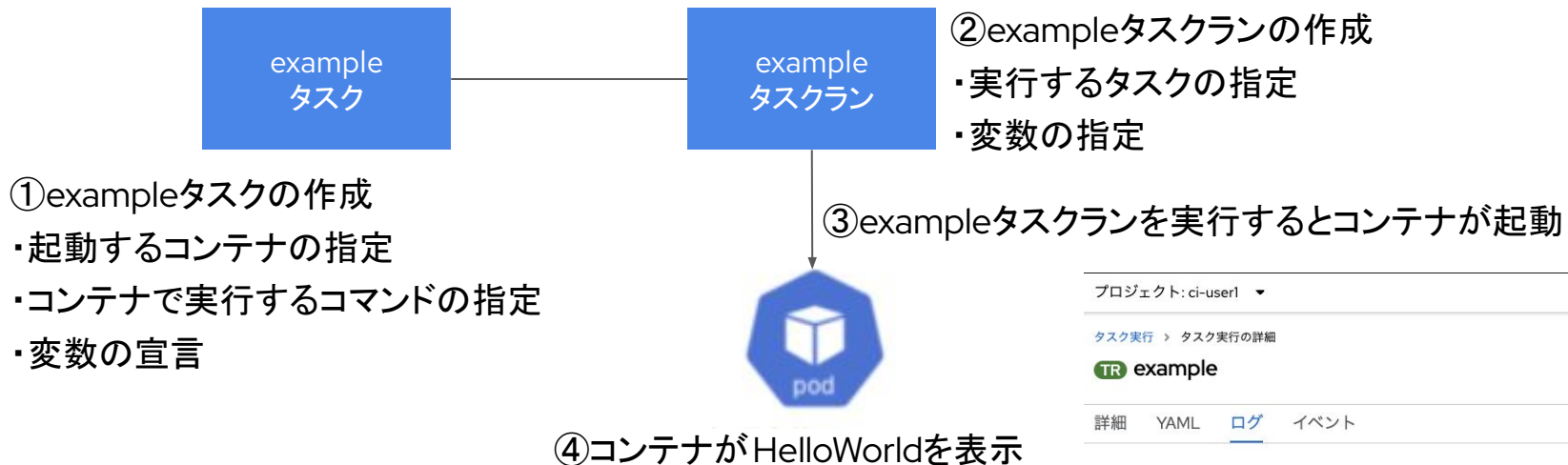
Tektonのパイプラインの考え方 (2/2)

最初は、どうしても慣れが必要です。

用語	説明
Task	Tektonで表現できる最小の実行単位。ひとつ以上のStepから成り立つ。Taskが必要とするパラメータなどを定義できる。
TaskRun	定義したTaskを実行するために利用。Taskの実行に必要な、入力や出力・パラメータなどを指定する。TaskRun単体として利用することもできるが、Pipelineを実行したときにTaskRunが作成される。
Pipeline	パイプライン。定義したTaskの集合体。パイプラインを実行するのに必要なパラメータなども定義できる。
PipelineRun	定義したPipelineを実行するために利用。パイプラインを実行するために、入力や出力、パラメータを指定する。

TektonタスクでHelloWorld(1/)

まずは、画面にHelloWorldを出力するシンプルなタスクを作成し、タスクの概要を理解します。
作成するタスクとタスクラン (タスクだけ実行するもの) を以下に示します。



プロジェクト: ci-user1 ▾

タスク実行 > タスク実行の詳細

TR example

詳細 YAML ログ イベント

example

STEP-HELLOWORLD

HelloWorld

TektonタスクでHelloWorld(1/6)

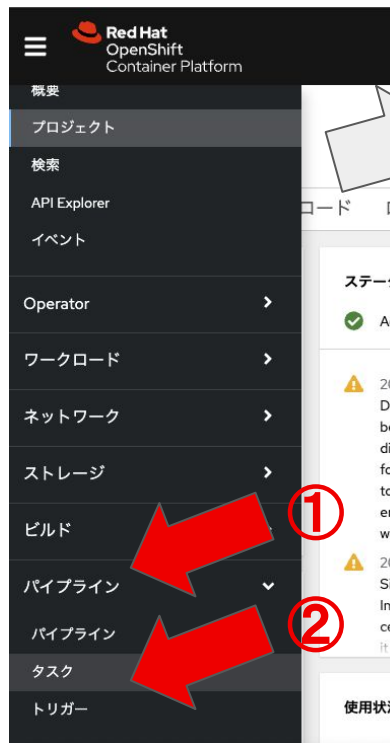
OpenShiftのWeb画面にログインします。(ログイン方法、ユーザ名・パスワードは講師が案内します)
CIパイプライン用のプロジェクトを選択します。

名前	表示名	ステータス	リクエスター
PR ci-user1	表示名なし	Active	user1
PR ci-user2	表示名なし	Active	user2
PR ci-user3	表示名なし	Active	user3
PR ci-user4	表示名なし	Active	user4
PR ci-user5	表示名なし	Active	user5
PR ci-user6	表示名なし	Active	user6
PR ci-user7	表示名なし	Active	user7
PR ci-user8	表示名なし	Active	user8
PR ci-user9	表示名なし	Active	user9
PR ci-user10	表示名なし	Active	user10
PR ci-user11	表示名なし	Active	user11
PR ci-user12	表示名なし	Active	user12
PR ci-user13	表示名なし	Active	user13
PR ci-user14	表示名なし	Active	user14
PR ci-user15	表示名なし	Active	user15

ci-<ユーザ名>
user1の場合は
ci-user1を選択

TektonタスクでHelloWorld(2/6)

タスクを作成します。



プロジェクト: ci-user1

タスクの作成

YAML または JSON 定義を手動で入力するか、またはファイルをエディターにドラッグし、ドロップして作成します。

```
1  apiVersion: tekton.dev/v1beta1
2  kind: Task
3  metadata:
4    name: example-task
5    namespace: ci-user1
6  spec:
7    params:
8      - name: appName
9        type: string
10   steps:
11     - image: registry.redhat.io/ubi7/ubi-minimal
12       command:
13         - /bin/bash
14         - '-c'
15         - echo
16         - ${inputs.params.appName}
17
```

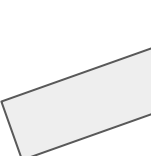
**タスクの作成画面
が開きます (次ページへ)**

TektonタスクでHelloWorld(3/6)

タスクを修正します。

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: example-task
  namespace: ci-user1
```

```
spec:
  params:
    - name: appName
      type: string
  steps:
    - image: registry.redhat.io/ubi7/ubi-minimal
      command:
        - /bin/bash
        - '-c'
        - echo
        - $(inputs.params.appName)
```



```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: example-task
  namespace: ci-user1
```

```
spec:
  params:
    - name: appName
      type: string
  steps:
    - image: registry.redhat.io/ubi7/ubi-minimal
      name: helloworld
      script: |
        #!/bin/bash
        echo "$(params.appName)"
```

paramsで変数の指定
appNameという変数を文字列 (String)

imageで利用するコンテナを指定

scriptで、処理を記載
(ほぼバッチのような内容)

TektonタスクでHelloWorld(4/6)

タスクを実行します。

The screenshot illustrates the steps to create a TaskRun in the Tekton console. It shows the 'タスク' (Tasks) section with the 'タスク実行' (TaskRuns) tab selected. A red arrow labeled '1' points to the 'タスク実行' tab. A red arrow labeled '2' points to the '作成' (Create) button. A red arrow labeled '3' points to the 'タスク実行' option in the dropdown menu. A large grey arrow points from the 'タスク実行' option to the 'タスク実行の作成' (Create TaskRun) dialog.

プロジェクト: ci-user1 ▼

タスク

タスク タスク実行 クラスタータスク

タスク実行が見つかりません

作成 ▼

タスク

タスク実行

クラスタータスク

プロジェクト: ci-user1 ▼

タスク実行の作成

YAML または JSON 定義を手動で入力するか、またはファイルをエディターにドラッグし、ドロップして作成します。

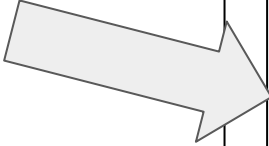
```
1 apiVersion: tekton.dev/v1beta1
2 kind: TaskRun
3 metadata:
4   name: example
5   namespace: ci-user1
6 spec: {}
7
```

作成 キャンセル

TektonタスクでHelloWorld(5/6)

タスクランを修正します。

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: example
  namespace: ci-user1
spec: {}
```



```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: example
  namespace: ci-user1
```

```
spec:
  taskRef:
    name: example-task
  params:
    - name: appName
      value: "HelloWorld"
```

taskRefで、さきほど作成したタスク
example-taskを指定

paramsで、変数appNameに、
値(Value)をHelloworldを指定

TektonタスクでHelloWorld(6/6)

タスクランを修正します。

プロジェクト: ci-user1 ▼

タスク実行 > タスク実行の詳細

TR example

詳細 YAML ログ イベント

タスク実行の詳細

名前
example

namespace
NS ci-user1

ラベル
編集 ✎
app.kubernetes.io/managed-by=tekton-pipelines
tekton.dev/task=example-task

アノテーション
1件のアノテーション ✎

ステータス
実行中

Pod
P example-pod

プロジェクト: ci-user1 ▼

タスク実行 > タスク実行の詳細

TR example

詳細 YAML ログ イベント

example

STEP=HELLOWORLD
HelloWorld

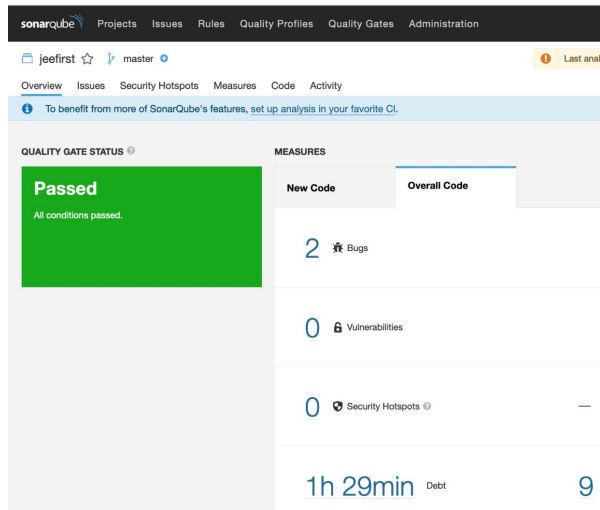
HelloWorldが出力されます

ダウンロード | 拡張

ソースコードの静的診断のタスク作成 (1/4)

SonarQubeについて

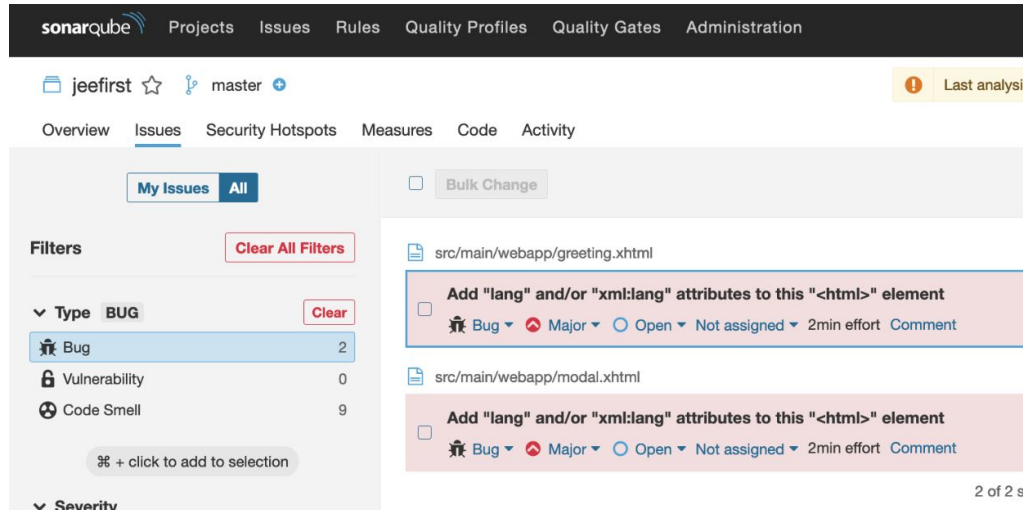
SonarSourceが開発したオープンソースプロジェクトで、約30のプログラミング言語 (Java,PHP,HTML,CSS3等)でコードの静的解析を実行し、コード品質を継続的に検査して、コード品質、コードセキュリティのチェックを実施するツールです。



The screenshot shows the SonarQube Quality Gate Status page. The overall status is 'Passed' with a green background. Below this, there are three categories of issues: 'New Code' (2 Bugs), 'Vulnerabilities' (0), and 'Security Hotspots' (0). The 'Overall Code' section shows a 'Debt' of 1h 29min and a total of 9 issues.

Category	Count
New Code	2 Bugs
Vulnerabilities	0
Security Hotspots	0
Overall Code Debt	1h 29min
Total Issues	9

静的解析結果



The screenshot shows the SonarQube Issues page. The 'Issues' tab is selected, and the 'All' filter is applied. The 'Filters' section shows 'Type' as 'BUG' with a count of 2. The 'Severity' section shows 'Vulnerability' (0) and 'Code Smell' (9). The 'Bulk Change' section shows two items with the message 'Add "lang" and/or "xml:lang" attributes to this "<html>" element'. Each item has a 'Bug' icon, a 'Major' severity, and a '2min effort' estimate.

Type	Count
Bug	2
Vulnerability	0
Code Smell	9

File	Message	Severity	Effort
src/main/webapp/greeting.xhtml	Add "lang" and/or "xml:lang" attributes to this "<html>" element	Major	2min effort
src/main/webapp/modal.xhtml	Add "lang" and/or "xml:lang" attributes to this "<html>" element	Major	2min effort

バグ抽出結果

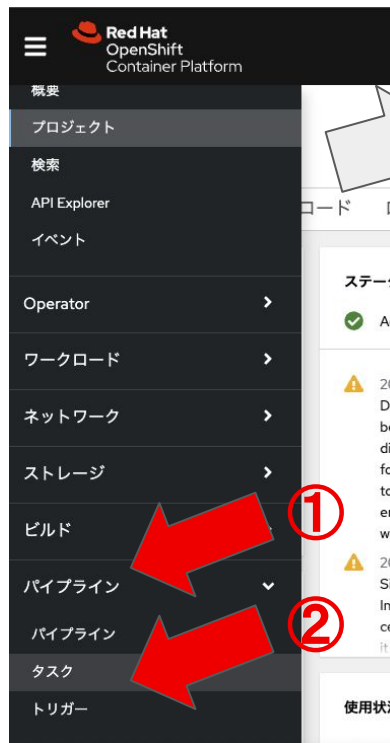
ソースコードの静的診断のタスク作成 (2/4)

ソースコードの静的診断のタスクを作成します。
作成するタスクの処理内容を以下に示します。

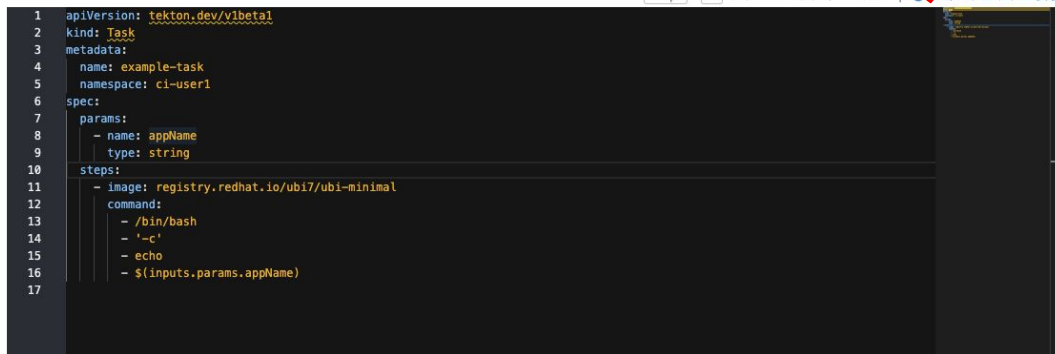
- cleanup-repo: ゴミ掃除
- clone-repo: サンプルソースコードの取得
サンプルソースコード
<https://github.com/takahiro-ino/cakephp-ex.git>
- run-sonar-scanner: ソースコードの静的診断
SonarQubeサーバに対して、sonar-scannerコマンドを用いて、静的解析を実行

ソースコードの静的診断のタスク作成 (3/4)

タスクを作成します。



**タスクの作成画面
が開きます (次ページへ)**



ソースコードの静的診断のタスク作成 (4/4)

下記の定義ファイルの内容をタスク作成の画面に貼り付け、作成します。

<https://github.com/takahiro-ino/cicd-handson/blob/main/sonar-scanner-task.yaml>

```
1  apiVersion: tekton.dev/v1beta1
2  kind: Task
3  metadata:
4    name: sonar-scanner-task
5  spec:
6    params:
7      - name: sonarHostUrl
8        type: string
9        description: The URL of the SonarQube server
10     - name: sonarLogin
11       type: string
12       description: The login ID for the SonarQube server
13     - name: sonarPassword
14       type: string
15       description: The login password for the SonarQube server
16     - name: gitRepositoryUrl
17       type: string
18       description: The URL of the GitHub repository to scan
19   workspaces:
20     - name: workspace
21       description: The workspace for the source code
22   steps:
23     - name: cleanup-repo
24       image: 'alpine'
25       script: |
26         #!/bin/sh
27         rm -rf $(workspaces.workspace.path)/repo
28     - name: clone-repo
29       image: 'alpine/git'
30       script: |
31         #!/bin/sh
32         git clone $(params.gitRepositoryUrl) $(workspaces.workspace.path)/repo
33     - name: run-sonar-scanner
34       image: 'docker.io/sonar-scanner/sonar-scanner-cli:4.6@sha256:edbf641d656ef38548eaf652b857a7de7ce6235f6b458a523ba1678a728d6ad'
35       script: |
36         #!/bin/bash
37         chown -R 1000:1000 $(workspaces.workspace.path)/repo
38         cd $(workspaces.workspace.path)/repo
39         sonar-scanner \
40           -Dsonar.projectKey=cakephp-ex \
41           -Dsonar.projectName=cakephp-ex \
42           -Dsonar.projectVersion=1.0 \
43           -Dsonar.sources=. \
44           -Dsonar.host.url=$(params.sonarHostUrl) \
45           -Dsonar.login=$(params.sonarLogin) \
46           -Dsonar.password=$(params.sonarPassword)
```

貼り付け



```
プロジェクト: ci-user1 ▼

タスクの作成

YAML または JSON 定義を手動で入力するか、またはファイルをエディターにドラッグ
```

```
1  apiVersion: tekton.dev/v1beta1
2  kind: Task
3  metadata:
4    name: sonar-scanner-task
5  spec:
6    params:
7      - name: sonarHostUrl
8        type: string
9        description: The URL of the SonarQube server
10     - name: sonarLogin
11       type: string
12       description: The login ID for the SonarQube server
13     - name: sonarPassword
14       type: string
15       description: The login password for the SonarQube server
16     - name: gitRepositoryUrl
17       type: string
18       description: The URL of the GitHub repository to scan
19   workspaces:
20     - name: workspace
21       description: The workspace for the source code
22   steps:
23     - name: cleanup-repo
24       image: 'alpine'
25       script: |
26         #!/bin/sh
27         rm -rf $(workspaces.workspace.path)/repo
28     - name: clone-repo
29       image: 'alpine/git'
30       script: |
31         #!/bin/bash
32         chown -R 1000:1000 $(workspaces.workspace.path)/repo
33         cd $(workspaces.workspace.path)/repo
34         sonar-scanner \
35           -Dsonar.projectKey=cakephp-ex \
36           -Dsonar.projectName=cakephp-ex \
37           -Dsonar.projectVersion=1.0 \
38           -Dsonar.sources=. \
39           -Dsonar.host.url=$(params.sonarHostUrl) \
40           -Dsonar.login=$(params.sonarLogin) \
41           -Dsonar.password=$(params.sonarPassword)
```

コンテナビルドのタスク作成

コンテナビルドのタスクは、既に OpenShiftで準備されていますので、作成は不要です。
パイプライン作成時に、パラメータを指定しますが、パラメータは後述します。

- Source to Imageビルドについて
アプリケーションソースコードとベースイメージを動的にビルドする機能 (s2i)
この機能によって、開発者はソースコード開発に専念できます。



コンテナイメージスキャンのタスク作成 (1/3)

コンテナイメージスキャンのタスクは、コンテナイメージの脆弱性診断ツールの Trivyを使用します。



Trivyの診断内容

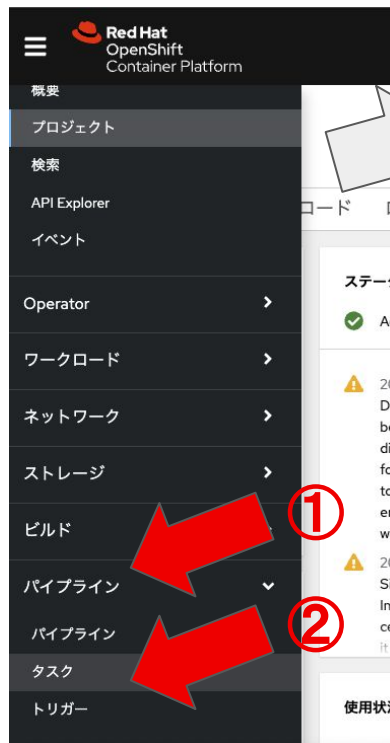
- ・使用しているOSのソフトウェアの依存関係 (SBOM)
- ・既知の脆弱性 (CVE)
- ・IaC の問題と設定ミス
- ・機密情報とシークレット情報

Trivyの脆弱性DB

- ・Trivyの脆弱性スキャンは、脆弱性 DB(trivy-db)をベースに行われます。
- ・脆弱性DBは、6時間おきに更新されています。

コンテナイメージスキャンのタスク作成 (2/3)

タスクを作成します。



タスクの作成

YAML または JSON 定義を手動で入力するか、またはファイルをエディターにドラッグし、ドロップして作成します。

```
1  apiVersion: tekton.dev/v1beta1
2  kind: Task
3  metadata:
4    name: example-task
5    namespace: ci-user1
6  spec:
7    params:
8      - name: appName
9        type: string
10   steps:
11     - image: registry.redhat.io/ubi7/ubi-minimal
12       command:
13         - /bin/bash
14         - '-c'
15         - echo
16         - ${inputs.params.appName}
17
```

**タスクの作成画面
が開きます (次ページへ)**

コンテナイメージスキャンのタスク作成 (3/3)

下記の定義ファイルの内容をタスク作成の画面に貼り付け、作成します。

<https://github.com/takahiro-ino/cicd-handson/blob/main/container-scan.yaml>

```
1  apiVersion: tekton.dev/v1beta1
2  kind: Task
3  metadata:
4    name: sonar-scanner-task
5  spec:
6    params:
7      - name: sonarHostUrl
8        type: string
9        description: The URL of the SonarQube server
10     - name: sonarLogin
11       type: string
12       description: The login ID for the SonarQube server
13     - name: sonarPassword
14       type: string
15       description: The login password for the SonarQube server
16     - name: gitRepositoryUrl
17       type: string
18       description: The URL of the GitHub repository to scan
19   workspaces:
20     - name: workspace
21       description: The workspace for the source code
22   steps:
23     - name: cleanup-repo
24       image: 'alpine'
25       script: |
26         #!/bin/sh
27         rm -rf $(workspaces.workspace.path)/repo
28     - name: clone-repo
29       image: 'alpine/git'
30       script: |
31         #!/bin/sh
32         git clone $(params.gitRepositoryUrl) $(workspaces.workspace.path)/repo
33     - name: run-sonar-scanner
34       image: docker.io/sonarsource/sonar-scanner-cli:4.6bsha256:ed0f641d056ef38548eaf652b857a7de7ce0235f6b458a523ba1678d6daed
35       script: |
36         #!/bin/bash
37         chown -R 1000:1000 $(workspaces.workspace.path)/repo
38         cd $(workspaces.workspace.path)/repo
39         sonar-scanner \
40           -Dsonar.projectKey=cakephp-ex \
41           -Dsonar.projectName=cakephp-ex \
42           -Dsonar.projectVersion=1.0 \
43           -Dsonar.sources=. \
44           -Dsonar.host.url=$(params.sonarHostUrl) \
45           -Dsonar.login=$(params.sonarLogin) \
46           -Dsonar.password=$(params.sonarPassword)
```

貼り付け



プロジェクト: ci-user25

タスクの作成

YAML または JSON 定義を手動で入力するか、またはファイルをエディターにドラッグし、ドロップして作成し

```
1  apiVersion: tekton.dev/v1beta1
2  kind: Task
3  metadata:
4    name: container-scan
5    labels:
6      app.kubernetes.io/version: "0.1"
7  annotations:
8    tekton.dev/pipelines.minVersion: "0.21.0"
9    tekton.dev/categories: Security
10   tekton.dev/tags: Security
11  spec:
12    description: >-
13      This Task scans Comprehensive Vulnerability using Trivy for Containers
14      and other Artifacts, Suitable in CI.
15    params:
16      - name: IMAGE
17        description: Reference of the image Trivy will scan.
18      - name: IMAGE_DIGEST
19        description: Reference of the image digest Trivy will scan.
20    workspaces:
21      - name: workspace
22    stepTemplate:
23      env:
24        - name: TRIVY_USERNAME
25          value: ""
26        - name: TRIVY_PASSWORD
27          value: ""
28    steps:
29      - name: build-report
30        image: aquasec/trivy:0.19.2
31        script: |
32          SCAN_CONTAINER_IMAGE=$(params.IMAGE)@$(tasks.s21-php.results.IMAGE_DIGEST)
33          mkdir -p $(workspaces.workspace.path)/trivy
34          trivy --cache-dir=$(workspaces.workspace.path)/trivy \
35            image --timeout 10m --exit-code=0 --severity=HIGH,CRITICAL \
36            $(SCAN_CONTAINER_IMAGE)
```

CIパイプラインの作成

ここまでの作業で、必要なタスクを作成しました。
タスクを組み合わせ、CIパイプラインを作成します。



CIパイプラインの作成

ワークスペースの設定をします。ワークスペースは作業用のディレクリ / フォルダに相当します。

プロジェクト: ci-user25 ▾

パイプラインビルダー

次を使用して設定: ☒ パイプラインビルダー ☐ YAML ビュー

名前 *

new-pipeline

タスク *

タスクの追加

+ 最終タスクの追加

①

パラメーター

このパイプラインにはパラメーターは関連付けられていません。

+ パラメーターの追加

リソース

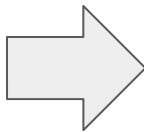
このパイプラインにはリソースは関連付けられていません。

+ リソースの追加

ワークスペース

このパイプラインに関連付けられているワークスペースはありません。

+ ワークスペースの追加



プロジェクト: ci-user25 ▾

パイプラインビルダー

次を使用して設定: ☒ パイプラインビルダー ☐ YAML ビュー

名前 *

new-pipeline

タスク *

タスクの追加

+ 最終タスクの追加

パラメーター

このパイプラインにはパラメーターは関連付けられていません。

+ パラメーターの追加

リソース

このパイプラインにはリソースは関連付けられていません。

+ リソースの追加

ワークスペース

名前 *

workspace

☐ オプションのワークスペース

+ ワークスペースの追加

作成 キャンセル



②
**workspace
と入力します**

CIパイプラインの作成

ソースコードの静的診断のタスクを追加します。

プロジェクト: ci-user25 ▾

パイプラインビルダー

次を使用して設定: ☒ パイプラインビルダー ☐ YAML ビュー

名前 *

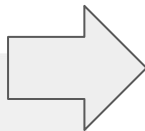
new-pipeline

タスク *

タスクの追加

+ 最終タスクの追加

①



+ sonar

T

sonar-scanner-task

Red Hat

Red Hat

T

sonarqube-scanner

コミュニティ

Security

sonar-scanner-task

Red Hat

追加

②

パラメーター

このパイプラインにはパラメーターは関連付けられていません。

+ パラメーターの追加

リソース

このパイプラインにはリソースは関連付けられていません。

+ リソースの追加

ワークスペース

このパイプラインに関連付けられているワークスペースはありません。

+ ワークスペースの追加

CIパイプラインの作成

ソースコードの静的診断のタスクを追加します。

プロジェクト: ci-user25 ▾

パイプラインビルダー

次を使用して設定: ☒ パイプラインビルダー ☐ YAML ビュー

名前 *

new-pipeline

タスク *

sonar-scanner... [最終タスクの追加](#)

パラメーター

このパイプラインにはパラメーターは関連付けられていません。
[パラメーターの追加](#)

リソース

このパイプラインにはリソースは関連付けられていません。
[リソースの追加](#)

ワークスペース 1

名前 *

workspace

☐ オプションのワークスペース

[ワークスペースの追加](#)

[作成](#) [キャンセル](#)

sonar-scanner-task

アクション ▾

[ショートカットの表示](#)

表示名 *

sonar-scanner-task

パラメーター

この形式で変数を参照する際には以下を使用します: `$()`

sonarHostUri *

`https://sonarqueue-ci-user25.apps.opentlc.com/m2fr.m2frsandbox2725.opentlc.com/`

The URL of the SonarQube server

gitRepositoryUri *

`https://github.com/takahiro-ino/akephp-ex.git`

The URL of the GitHub repository to scan

ワークスペース

workspace *

workspace

when 式

このタスクに関連付けられている when 式はありません
[when 式の追加](#)

②:メモ帳の値を使用

③:メモ帳の値を使用

④:workspaceを選択

CIパイプラインの作成

コンテナビルドの静的診断のタスクを追加します。

パイプラインビルダー

次を使用して設定: ☒ パイプラインビルダー ☐ YAML ビュー

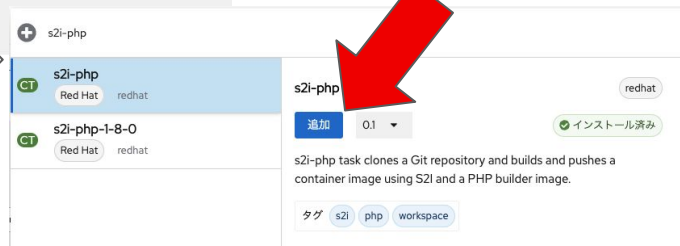
名前 *

new-pipeline

タスク *



①タスクを選択すると、+のアイコンが表示されます。作成したタスクの後の +の箇所をクリックします。



CIパイプラインの作成

コンテナビルドの静的診断のタスクを追加します。

プロジェクト: ci-user25 ▾

パイプラインビルダー

次を使用して設定: ☒ パイプラインビルダー ☐ YAML ビュー

名前 *

new-pipeline

タスク *

sonar-scanner... s2i-php 最終タスク

パラメーター

このパイプラインにはパラメーターは関連付けられていません。
[パラメーターの追加](#)

リソース

このパイプラインにはリソースは関連付けられていません。
[リソースの追加](#)

ワークスペース

名前 *

workspace

☐ オプションのワークスペース

[ワークスペースの追加](#)

作成 キャンセル

CT s2i-php

アクション ▾

[ショートカットの表示](#)

パラメーター

この形式で変数を参照する際には以下を使用します: `$()`

VERSION

latest

The tag of php imagestream for php version

PATH_CONTEXT

.

The location of the path to run s2i from.

TLSVERIFY

false

Verify the TLS on the registry endpoint (for push/pull to a non-TLS registry)

IMAGE *

image:registry.openshift-image-registry.svc:5000/ci-user25/cake-php

Location of the repo where image has to be pushed

BUILDER_IMAGE

registry.redhat.io/rhel8/buildah@sha256:0a86ecbdfbe86e9d225b7fe4b090

The location of the buildah builder image.

SKIP_PUSH

false

Skip pushing the built image

ENV_VARS

[値の追加](#)

ワークスペース

source *

workspace

①:false を指定

②:メモ帳の値を使用

③:workspaceを選択

CIパイプラインの作成

コンテナイメージスキャンのタスクを追加します。
さきほどと同様に、タスクを追加します。

プロジェクト: ci-user25 ▾

パイプラインビルダー

次を使用して設定: ☒ パイプラインビルダー ☐ YAML ビュー

名前 *

new-pipeline

タスク *

sonar-scanner... s2l-php タスクの追加 + 最終タスクの追加

パラメーター

このパイプラインにはパラメーターは関連付けられていません。

+ パラメーターの追加

リソース

このパイプラインにはリソースは関連付けられていません。

+ リソースの追加

ワークスペース ⓘ

名前 *

workspace

☐ オプションのワークスペース

作成 キャンセル

+ container-

container-scan Red Hat Red Hat

container-scan Red Hat

追加 0.1

インストール済み

This Task provides comprehensive Vulnerability using Trivy for Container Images, Artifacts, Suitable in CI.

カテゴリー Security

タグ Security

CIパイプラインの作成

コンテナイメージスキャンのタスクを追加します。
さきほどと同様に、タスクを追加します。

プロジェクト: ci-user25 ▾

パイプラインビルダー

次を使用して設定: ☒ パイプラインビルダー ☐ YAML ビュー

名前 *

new-pipeline

タスク *

sonar-scanner... s2i-php container-scan + 最終タスクの追加

パラメーター

このパイプラインにはパラメーターは関連付けられていません。
[パラメーターの追加](#)

リソース

このパイプラインにはリソースは関連付けられていません。
[リソースの追加](#)

ワークスペース ^①

名前 *

workspace

☐ オプションのワークスペース

[ワークスペースの追加](#)

作成

container-scan

アクション ▾

ショートカットの表示

表示名 *

container-scan

パラメーター

この形式で変数を参照する際には以下を使用します: `$()`

IMAGE *

image:registry.openshift-image-registry.svc:5000/ci-user25/cake-php

Reference of the image Trivy will scan.

IMAGE_DIGEST *

\$(tasks.s2i-php.results.IMAGE_DIGEST)

Reference of the image digest Trivy will scan.

ワークスペース

workspace *

workspace

when 式

このタスクに関連付けられている when 式はありません
[when 式の追加](#)

①:メモ帳の値を使用

②:メモ帳の値を使用

③:workspaceを選択

④

CIパイプラインの実行

作成したパイプラインを実行します。

プロジェクト: ci-user25 ▾

パイプライン > パイプラインの詳細

PL new-pipeline

詳細 メトリクス YAML パイプライン実行 パラメーター リソース

パイプラインの詳細



名前

new-pipeline

namespace

NS ci-user25

ラベル

ラベルなし

編集

アノテーション

0件のアノテーション

タスク

1 sonar-scanner-task

CT s2i-php

1 container-scan

ワークスペース

workspace

アクション

開始

トリガーの追加

ラベルの編集

アノテーションの編集

パイプラインの編集

パイプラインの削除

パイプラインの起動

ワークスペース

workspace *

永続ボリューム要求

PVC your-pvc

詳細オプション

> 認証情報オプションの表示

キャンセル

開始

③:永続ボリューム要求を選択

④:your-pvcを選択

⑤

CIパイプラインの実行

実行したパイプラインのログを確認します。

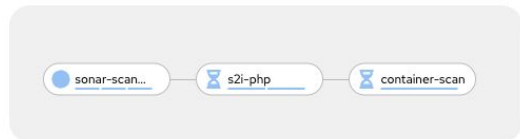
プロジェクト: ci-user25 ▾

パイプライン実行 > パイプライン実行の詳細

PLR new-pipeline-53gm3g

詳細 YAML タスク実行 ログ イベント

パイプライン実行の詳細



名前
new-pipeline-53gm3g

namespace
NS ci-user25

ラベル
tekton.dev/pipeline=new-pipeline

アノテーション
1件のアノテーション

作成した時間:
🕒 進行中

ステータス
実行中

パイプライン
PL new-pipeline

トリガー:
user25

ワークスペースリソース
PVC your-pvc (workspace)

プロジェクト: ci-user25 ▾

パイプライン実行 > パイプライン実行の詳細

PLR new-pipeline-53gm3g 実行中

詳細 YAML タスク実行 ログ イベント

📄 ダウンロード | 📄 すべてのタスクログをダウンロード | 🔍 拡張

sonar-scanner-task

s2i-php

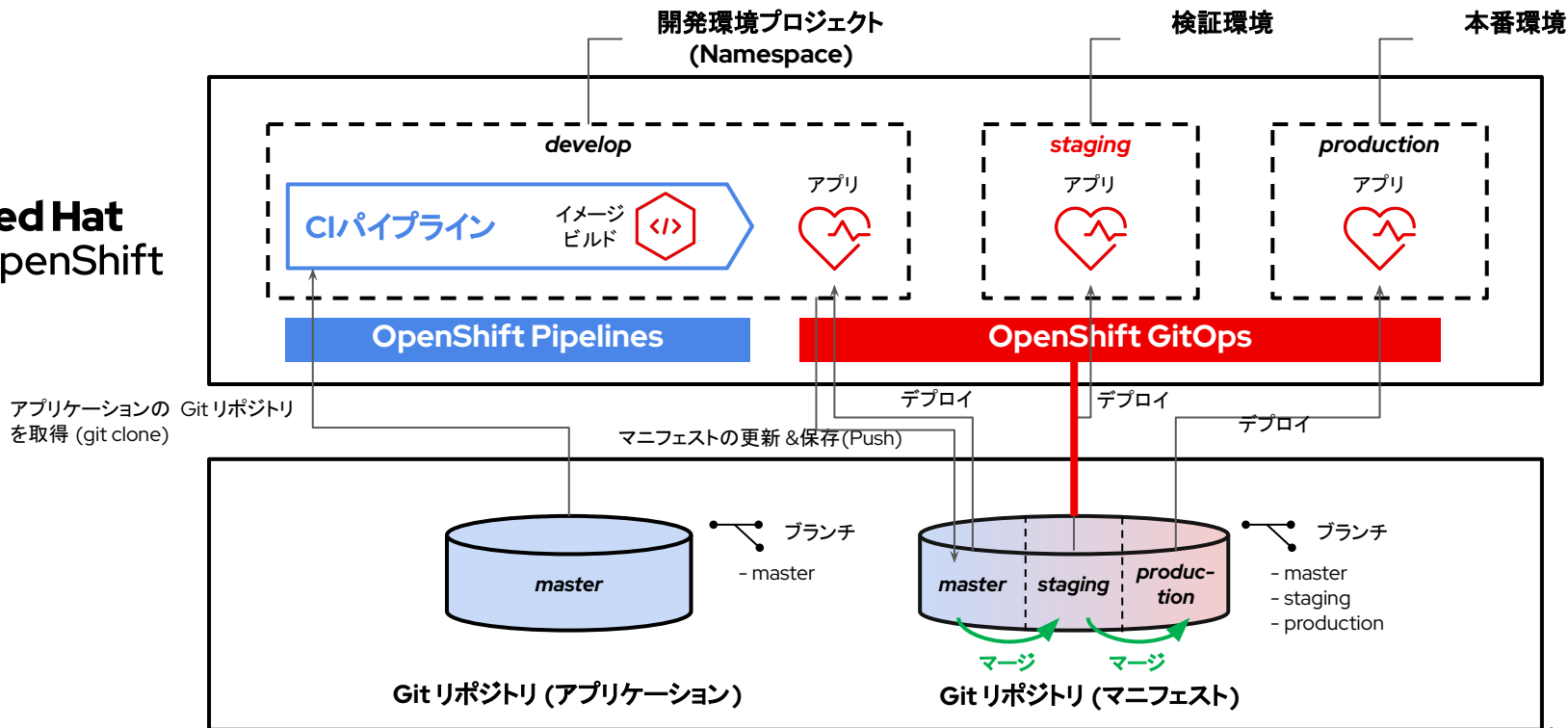
```
s2i-php
STEP-GENERATE
Processing Build Environment Variables
Application dockerfile generated in /gen-source/Dockerfile.gen

STEP-BUILD-AND-PUSH
STEP 1/8: FROM image-registry.openshift-image-registry.svc:5000/openshift/php:
Trying to pull image-registry.openshift-image-registry.svc:5000/openshift/php:
Getting image source signatures
Copying blob sha256:bcd0f2cc02413b220bbc752787c9977435b56a4ce3a2fc86965dbd6184
Copying blob sha256:8694db102e5bd27fa30106f87d5a0f0c5ccccac0e5cc38ba56080d7559
Copying blob sha256:7027f4e4058bde8aaa497e47562e962c293039ba16f5fbfd07ff43a0d1
Copying blob sha256:be575238ea985ef824635fbeat7b33eaed98ef6ba1db1822ac0714f509
Copying blob sha256:8694db102e5bd27fa30106f87d5a0f0c5ccccac0e5cc38ba56080d7559
Copying blob sha256:7027f4e4058bde8aaa497e47562e962c293039ba16f5fbfd07ff43a0d1
Copying blob sha256:bcd0f2cc02413b220bbc752787c9977435b56a4ce3a2fc86965dbd6184
Copying blob sha256:be575238ea985ef824635fbeat7b33eaed98ef6ba1db1822ac0714f509
```

CD ハンズオン

CDハンズオン

CIパイプラインで作成したコンテナイメージを用いて、環境に配備します。



デプロイ用の YAML 作成

Gitlabに、以下のGitの内容をクローンします。

<https://gitlab.apps.cluster-4qr6t.4qr6t.sandbox133.opentlc.com/tinoue/cicd-handson>

cdフォルダにあるcake-php-deployment.yamlを修正します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cake-php-deployment
  labels:
    app: cake-php
spec:
  replicas: 1
  ~
```

```
image-registry.openshift-image-registry.svc:5000/ci-userXX/cake-php:latest
ports:
  - containerPort: 8080
    protocol: TCP
  - containerPort: 8443
    protocol: TCP
```

 clone後、
自身のuserに変更します。

ArgoでのCD

Argoにログインします。

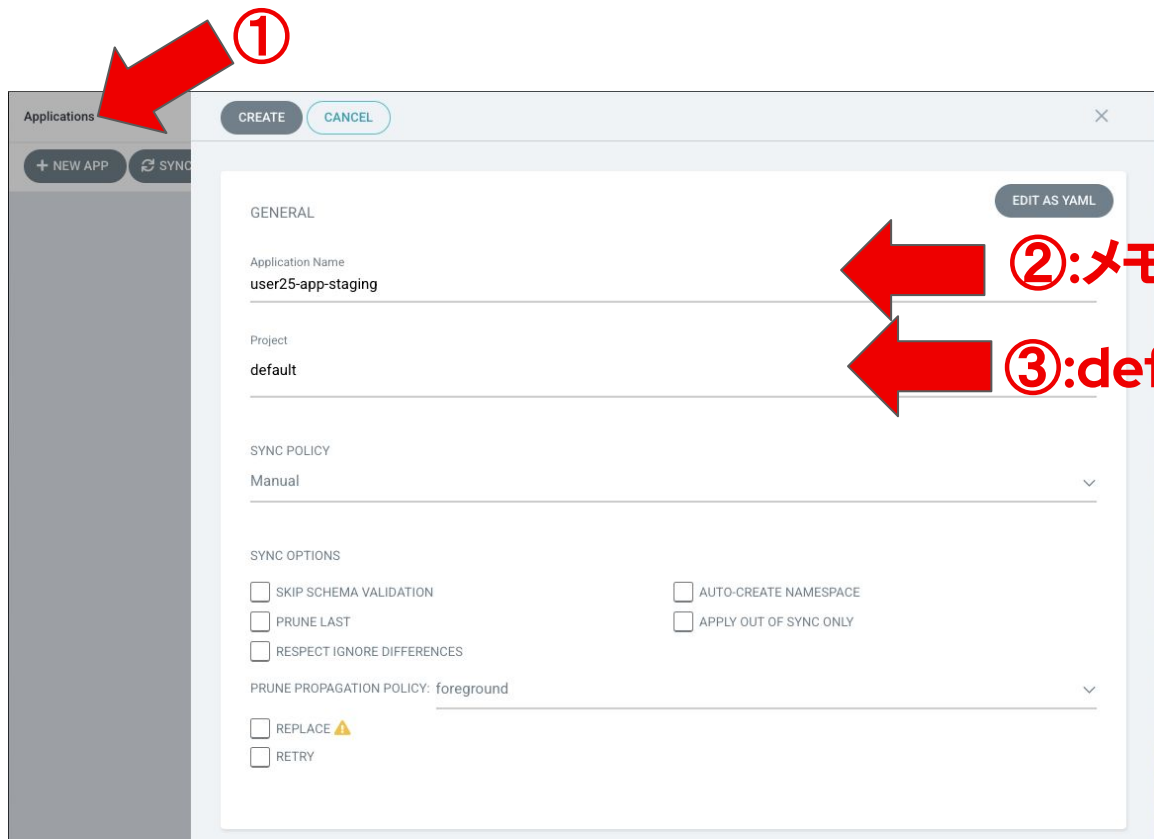
The image illustrates the process of logging into Argo CD through the Red Hat OpenShift Container Platform console. It consists of several components:

- Top Screenshot:** A screenshot of the Red Hat OpenShift Container Platform console. The left sidebar shows the navigation menu with options like '検索' (Search), 'API Explorer', 'イベント' (Events), and 'Operator'. The main area displays 'プロジェクト: ci-user25' and 'デプロイメント' (Deployments). A dropdown menu is open, showing 'Red Hat アプリケーション' (Red Hat Applications) with options: 'OpenShift Cluster Manager', 'OpenShift GitOps', and 'Cluster Argo CD'. A red arrow labeled '1' points to 'Cluster Argo CD'.
- Welcome Message:** A dark blue box with the text 'Let's get stuff deployed!' and an illustration of a smiling orange octopus wearing a space helmet, standing on a gear.
- Login Form:** A white box with the 'argo' logo and a 'LOG IN VIA OPENSIFT' button. Below the button are fields for 'Username' and 'Password', and a 'SIGN IN' link. A red arrow labeled '2' points to the 'LOG IN VIA OPENSIFT' button.
- Login Confirmation:** A white box titled 'アカウントにログイン' (Login to account). It contains fields for 'ユーザー名' (Username) with the value 'user25' and 'パスワード' (Password) with masked characters. A blue 'ログイン' (Login) button is at the bottom. A red arrow labeled '3' points to the password field. To the right of this box, the text '③userとパスワードを入力' (Enter user and password) is written in red.

Red Hat

ArgoでのCD

Argoで、CDの設定を実施します。



The screenshot shows the 'CREATE' dialog box in the Argo CD web interface. A red arrow labeled '①' points to the 'Applications' sidebar on the left. The dialog box has a 'CREATE' button and a 'CANCEL' button. The 'GENERAL' section contains the following fields:

- Application Name: `user25-app-staging`
- Project: `default`

The 'SYNC POLICY' section shows a dropdown menu set to 'Manual'. The 'SYNC OPTIONS' section contains four checkboxes:

- ☐ SKIP SCHEMA VALIDATION
- ☐ PRUNE LAST
- ☐ RESPECT IGNORE DIFFERENCES
- ☐ AUTO-CREATE NAMESPACE
- ☐ APPLY OUT OF SYNC ONLY

The 'PRUNE PROPAGATION POLICY' section shows a dropdown menu set to 'foreground'. There are also checkboxes for 'REPLACE' (with a warning icon) and 'RETRY'. A red arrow labeled '②' points to the 'Application Name' field, and a red arrow labeled '③' points to the 'Project' field.

②:メモ帳の値を使用

③:defaultを選択

ArgoでのCD

Argoで、CDの設定を実施します。

Applications

CREATE CANCEL

+ NL ⑤

SOURCE

Repository URL

https://github.com/takahiro-ino/cicd-handson.git

Revision

HEAD

Path

cd

DESTINATION

Cluster URL

https://kubernetes.default.svc

Namespace

user25staging

①:自身のgitのURL

②:cdを入力

③:左記を選択

④:メモ帳の値を使用

ArgoでのCD

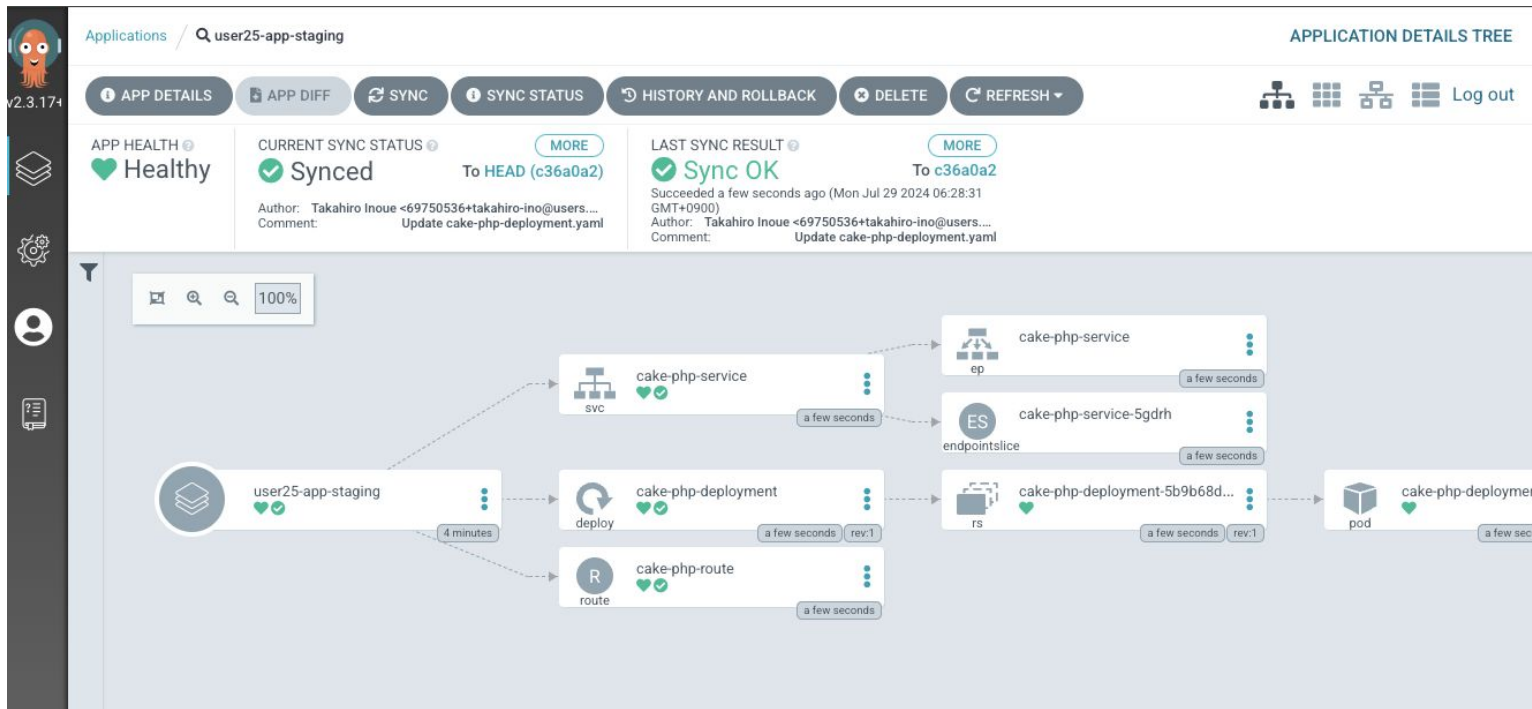
Argoで、CDの設定内容を同期をとります。

The screenshot displays the Argo CD web interface. On the left, the 'Applications' page shows a list of applications. The 'user25-app-staging' application is highlighted, showing its details: Project 'default', Repository 'https://github.com/takahiroino/cicd-handson.git', Target 'HEAD', Path 'cd', Destination 'in-cluster', and Name 'user25-staging'. A red arrow labeled '1' points to the 'SYNC' button in the application details panel.

On the right, the 'SYNCHRONIZE' dialog box is open. It shows the source repository 'https://github.com/takahiroino/cicd-handson.git' and the revision 'HEAD'. The dialog includes options for 'PRUNE', 'DRY RUN', 'APPLY ONLY', and 'FORCE'. There are also 'SYNC OPTIONS' for 'SKIP SCHEMA VALIDATION', 'AUTO-CREATE NAMESPACE', 'PRUNE LAST', 'APPLY OUT OF SYNC ONLY', and 'RESPECT IGNORE DIFFERENCES'. The 'PRUNE PROPAGATION POLICY' is set to 'foreground'. At the bottom, there are checkboxes for 'REPLACE' and 'RETRY'. The 'SYNCHRONIZE RESOURCES' section shows a list of resources to be synchronized: '/SERVICE/USER25-STAGING/CAKE-PHP-SERVICE', 'APPS/DEPLOYMENT/USER25-STAGING/CAKE-PHP-DEPLOYMENT', and 'ROUTE.OPENSIFT.IO/ROUTE/USER25-STAGING/CAKE-PHP-ROUTE'. A red arrow labeled '2' points to the 'SYNCHRONIZE' button in the dialog box.

ArgoでのCD

Argoで、gitの内容から、コンテナがデプロイされます。



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

TektonタスクでHelloWorld(5/6)

パイプラインを作成します。



パイプラインの作成画面
が開きます(次ページへ)

TektonタスクでHelloWorld(5/6)

パイプラインを作成します。

②先ほど作成した
example-taskを選択します

プロジェクト: ci-user1 ▾

パイプラインビルダー

次を使用して設定: ☒ パイプラインビルダー ☐ YAML ビュー

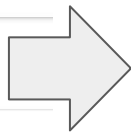
名前 *

new-pipeline

タスク *

タスクの追加

① + 最終タスクの追加



+ exam

example-task	Red Hat	Red Hat
git-batch-merge	コミュニティ	Git
github-open-pr	コミュニティ	Git
yq	コミュニティ	Developer Tools

example-task

追加

③

TektonタスクでHelloWorld(5/6)

パイプラインを作成します。

プロジェクト: ci-user1

パイプラインビルダー

次を使用して設定: ☒ パイプラインビルダー ☐ YAML ビュー

名前 *

new-pipeline

タスク *

example-task

パラメーター

この形式で変数を参照する際には以下を使用します: \$(

appName *

Hello World

when 式

このタスクに関連付けられている when 式はありません。

+ when 式の追加

リソース

このパイプラインにはリソースが関連付けられていません。

+ リソースの追加

作成 キャンセル

プロジェクト: ci-user1

パイプライン > パイプラインの詳細

PL new-pipeline

アクション

詳細 メトリクス YAML パイプライン実行 パラメーター リソース

フィルター

名前	タスクのステータス	開始	期間
PLR new-pipeline-4jyn5	成功	2024年7月27日 7:57	8 seconds

②
Hello World
を入力します

③
作成を押すと一緒に実行されます