OpenShift Container Platform 4.7

CLI ツール

OpenShift Container Platform コマンドラインツールの使用方法

Last Updated: 2023-02-20

OpenShift Container Platform 4.7 CLI ツール

OpenShift Container Platform コマンドラインツールの使用方法

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

本書では、OpenShift Container Platform コマンドラインツールのインストール、設定および使用について説明します。また、CLI コマンドの参照情報およびそれらの使用方法についての例も記載しています。

目次

第1章 OPENSHIFT CONTAINER PLATFORM CLI ツールの概要

OpenShift Container Platform での作業中に、次のようなさまざまな操作を実行します。

- クラスターの管理
- アプリケーションのビルド、デプロイ、および管理
- デプロイメントプロセスの管理
- Operator の開発
- Operator カタログの作成と保守

OpenShift Container Platform には、一連のコマンドラインインターフェイス (CLI) ツールが同梱されており、ユーザーがターミナルからさまざまな管理および開発操作を実行できるようにしてこれらのタスクを簡素化します。これらのツールでは、アプリケーションの管理だけでなく、システムの各コンポーネントを操作する簡単なコマンドを利用できます。

1.1. CLI ツールのリスト

OpenShift Container Platform では、以下の CLI ツールのセットを使用できます。

- OpenShift CLI (oc): これは OpenShift Container Platform ユーザーが最も一般的に使用する CLI ツールです。これは、クラスター管理者と開発者の両方が、ターミナルを使用して OpenShift Container Platform 全体でエンドツーエンドの操作が行えるようにします。Web コンソールとは異なり、ユーザーはコマンドスクリプトを使用してプロジェクトのソースコードを直接操作できます。
- Developer CLI (odo): **odo** CLI ツールは、複雑な Kubernetes および OpenShift Container Platform の概念を取り除くことで、開発者が OpenShift Container Platform でアプリケーションを作成および保守するという主目的に集中できるようにします。これにより、開発者はクラスターを管理する必要なしに、ターミナルからクラスターでのアプリケーション作成、ビルド、およびデバッグを行うことができます。
- Helm CLI: Helm は Kubernetes アプリケーションのパッケージマネージャーで、Helm チャートとしてパッケージ化されたアプリケーションの定義、インストール、およびアップグレードを可能にします。Helm CLI は、ターミナルからの簡単なコマンドを使用して、ユーザーがアプリケーションおよびサービスを OpenShift Container Platform クラスターに簡単にデプロイできるようにします。
- Knative CLI (kn): (**kn**) CLI ツールは、Knative Serving や Eventing などの OpenShift サーバーレスコンポーネントの操作に使用できるシンプルで直感的なターミナルコマンドを提供します。
- Pipelines CLI (tkn):OpenShift Pipelines は、内部で Tekton を使用する OpenShift Container Platform の継続的インテグレーションおよび継続的デリバリー (CI / CD) ソリューションです。**tkn** CLI ツールには、シンプルで直感的なコマンドが同梱されており、ターミナルを使用して OpenShift パイプラインを操作できます。
- opm CLI:**opm** CLI ツールは、オペレーター開発者とクラスター管理者がターミナルからオペレーターのカタログを作成および保守するのに役立ちます。
- Operator SDK: Operator Framework のコンポーネントである Operator SDK は、Operator 開発者がターミナルから Operator のビルド、テストおよびデプロイに使用できる CLI ツールを提供します。これにより、Kubernetes ネイティブアプリケーションを構築するプロセスが簡素化されます。これには、アプリケーション固有の深い運用知識が必要になる場合があります。

第2章 OPENSHIFT CLI (OC)

2.1. OPENSHIFT CLI の使用を開始する

2.1.1. OpenShift CLI について

OpenShift のコマンドラインインターフェイス (CLI)、**oc** を使用すると、ターミナルからアプリケーションを作成し、OpenShift Container Platform プロジェクトを管理できます。OpenShift CLI は以下の状況に適しています。

- プロジェクトソースコードを直接使用している。
- OpenShift Container Platform 操作をスクリプト化する。
- 帯域幅リソースによる制限があり、Web コンソールが利用できない状況でのプロジェクトの管理

2.1.2. OpenShift CLI のインストール。

OpenShift CLI(**oc**) をインストールするには、バイナリーをダウンロードするか、RPM を使用します。

2.1.2.1. バイナリーのダウンロードによる OpenShift CLI のインストール

コマンドラインインターフェイスを使用して OpenShift Container Platform と対話するために CLI (**oc**) をインストールすることができます。**oc** は Linux、Windows、または macOS にインストールできます。

重要

以前のバージョンの **oc** をインストールしている場合、これを使用して OpenShift Container Platform 4.7 のすべてのコマンドを実行することはできません。新規バージョンの **oc** をダウンロードし、インストールします。

2.1.2.1.1. Linux への OpenShift CLI のインストール

以下の手順を使用して、OpenShift CLI (**oc**) バイナリーを Linux にインストールできます。

手順

1. Red Hat カスタマーポータルの OpenShift Container Platform ダウンロードページ に移動します。
2. **Version** ドロップダウンメニューで適切なバージョンを選択します。
3. **OpenShift v4.7 Linux Client** エントリーの横にある **Download Now** をクリックして、ファイルを保存します。
4. アーカイブを展開します。

```
$ tar xvzf <file>
```

5. **oc** バイナリーを、**PATH** にあるディレクトリーに配置します。
PATH を確認するには、以下のコマンドを実行します。

```
$ echo $PATH
```

OpenShift CLI のインストール後に、**oc** コマンドを使用して利用できます。

```
$ oc <command>
```

2.1.2.1.2. Windows への OpenShift CLI のインストール

以下の手順を使用して、OpenShift CLI (**oc**) バイナリーを Windows にインストールできます。

手順

1. Red Hat カスタマーポータルの OpenShift Container Platform ダウンロードページ に移動します。
2. **Version** ドロップダウンメニューで適切なバージョンを選択します。
3. **OpenShift v4.7 Windows Client** エントリーの横にある **Download Now** をクリックして、ファイルを保存します。
4. ZIP プログラムでアーカイブを解凍します。
5. **oc** バイナリーを、**PATH** にあるディレクトリーに移動します。
PATH を確認するには、コマンドプロンプトを開いて以下のコマンドを実行します。

```
C:\> path
```

OpenShift CLI のインストール後に、**oc** コマンドを使用して利用できます。

```
C:\> oc <command>
```

2.1.2.1.3. macOC への OpenShift CLI のインストール

以下の手順を使用して、OpenShift CLI (**oc**) バイナリーを macOS にインストールできます。

手順

1. Red Hat カスタマーポータルの OpenShift Container Platform ダウンロードページ に移動します。
2. **Version** ドロップダウンメニューで適切なバージョンを選択します。
3. **OpenShift v4.7 MacOSX Client** エントリーの横にある **Download Now** をクリックして、ファイルを保存します。
4. アーカイブを展開し、解凍します。
5. **oc** バイナリーをパスにあるディレクトリーに移動します。
PATH を確認するには、ターミナルを開き、以下のコマンドを実行します。

```
$ echo $PATH
```

OpenShift CLI のインストール後に、**oc** コマンドを使用して利用できます。

```
$ oc <command>
```

2.1.2.2. Web コンソールを使用した OpenShift CLI のインストール

OpenShift CLI(**oc**) をインストールして、Web コンソールから OpenShift Container Platform と対話できます。**oc** は Linux、Windows、または macOS にインストールできます。

重要

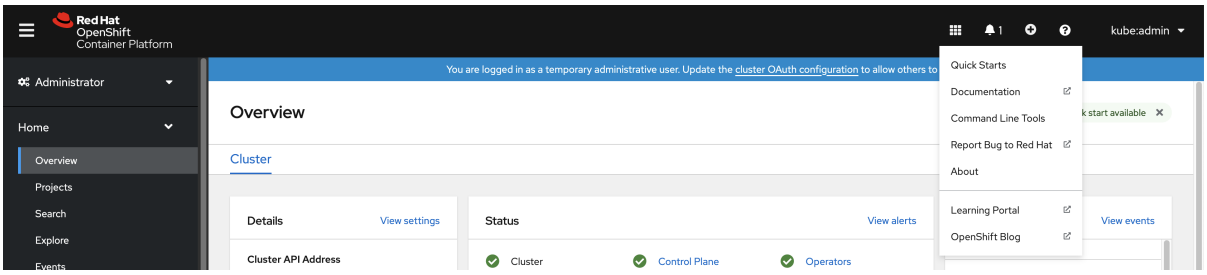
以前のバージョンの **oc** をインストールしている場合、これを使用して OpenShift Container Platform 4.7 のすべてのコマンドを実行することはできません。新規バージョンの **oc** をダウンロードし、インストールします。

2.1.2.2.1. Web コンソールを使用した Linux への OpenShift CLI のインストール

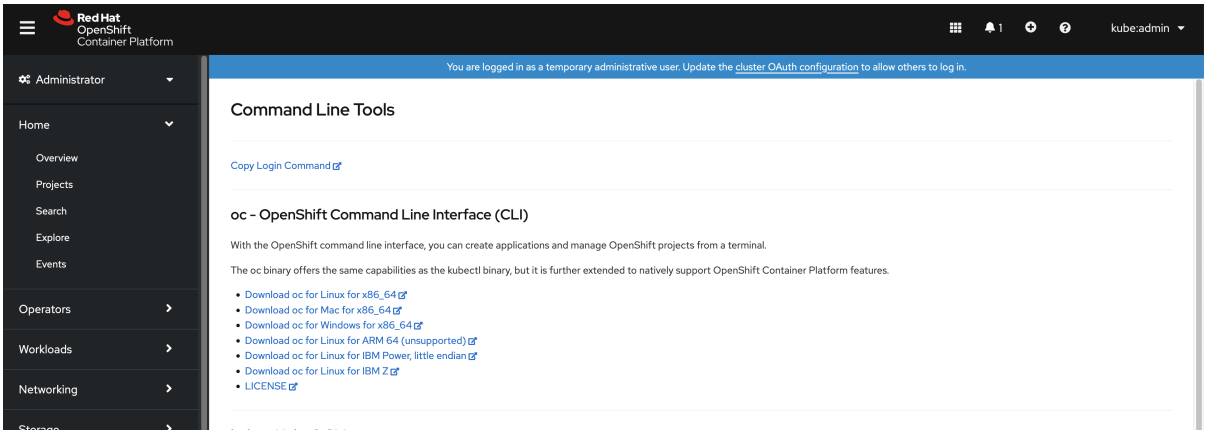
以下の手順を使用して、OpenShift CLI (**oc**) バイナリーを Linux にインストールできます。

手順

1. Web コンソールで **?** をクリックします。



2. コマンドラインツール をクリックします。



3. Linux プラットフォームに適した **oc** binary を選択してから、**Download oc for Linux**をクリックします。

4. ファイルを保存します。

5. アーカイブを展開します。

```
$ tar xvzf <file>
```

6. **oc** バイナリーを、**PATH** にあるディレクトリーに移動します。
PATH を確認するには、以下のコマンドを実行します。

```
$ echo $PATH
```

OpenShift CLI のインストール後に、**oc** コマンドを使用して利用できます。

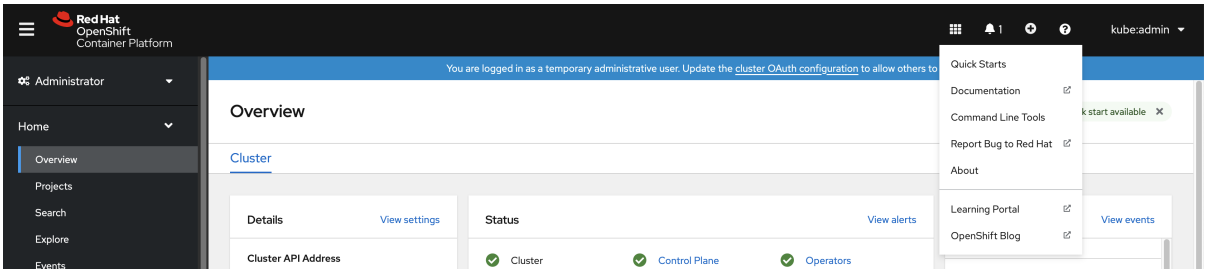
```
$ oc <command>
```

2.1.2.2.2. Web コンソールを使用した Windows への OpenShift CLI のインストール

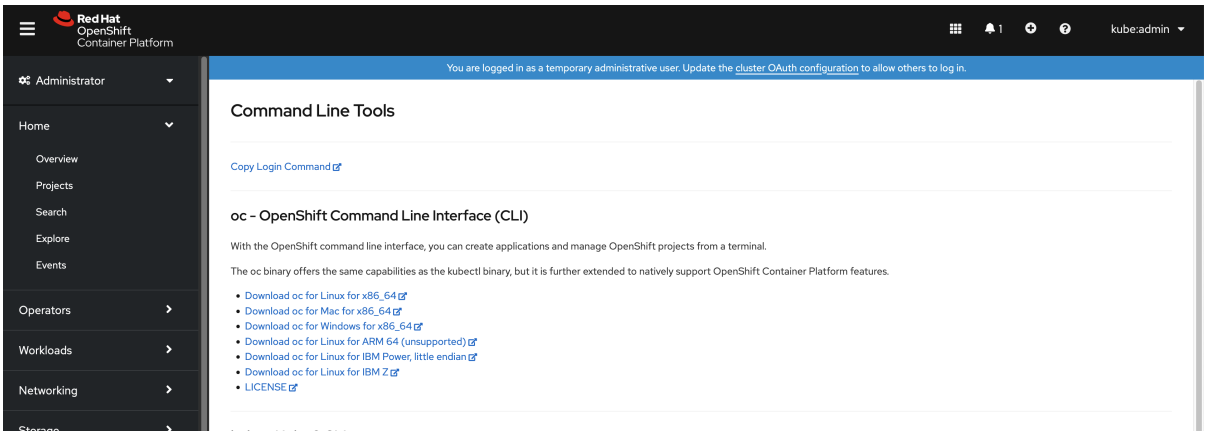
以下の手順を使用して、OpenShift CLI(**oc**) バイナリーを Windows にインストールできます。

手順

1. Web コンソールで ? をクリックします。



2. コマンドラインツール をクリックします。



3. Windows プラットフォームの **oc** バイナリーを選択してから、**Download oc for Windows for x86_64** をクリックします。

4. ファイルを保存します。

5. ZIP プログラムでアーカイブを解凍します。

6. **oc** バイナリーを、**PATH** にあるディレクトリーに移動します。
PATH を確認するには、コマンドプロンプトを開いて以下のコマンドを実行します。

```
C:\> path
```

OpenShift CLI のインストール後に、**oc** コマンドを使用して利用できます。

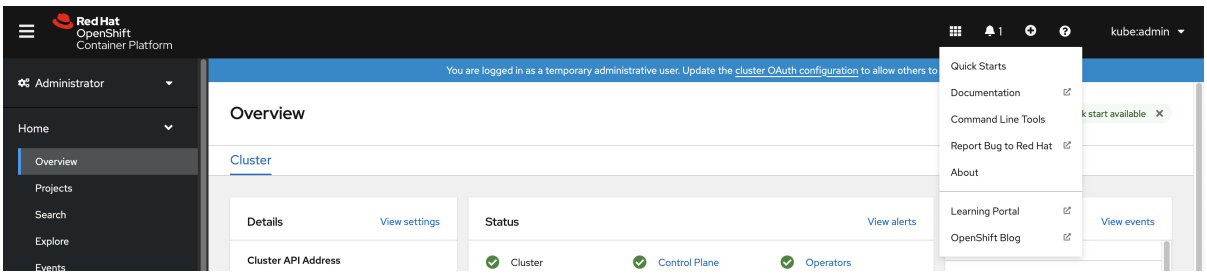
```
C:\> oc <command>
```

2.1.2.2.3. Web コンソールを使用した macOS への OpenShift CLI のインストール

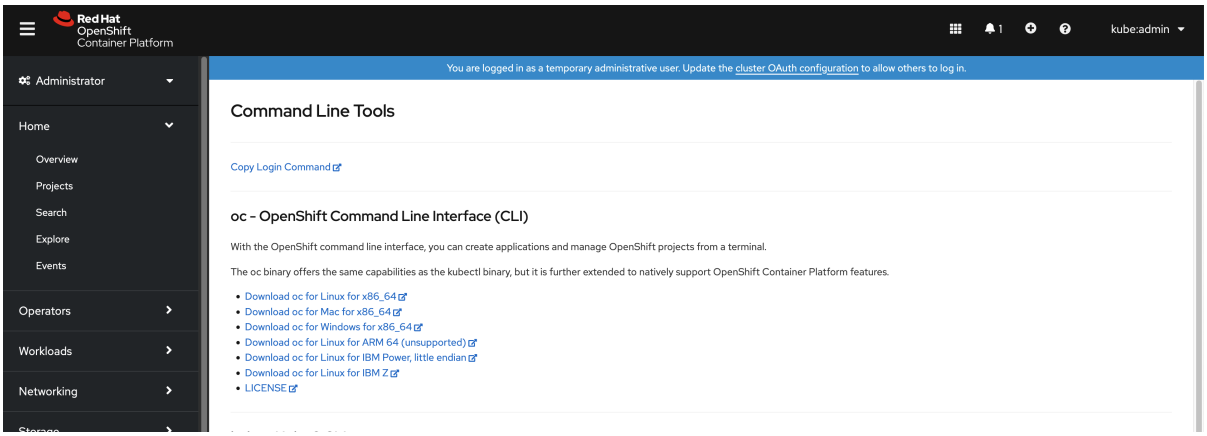
以下の手順を使用して、OpenShift CLI (**oc**) バイナリーを macOS にインストールできます。

手順

1. Web コンソールで ? をクリックします。



2. コマンドラインツール をクリックします。



3. macOS プラットフォームの **oc** バイナリーを選択し、**Download oc for Mac for x86_64**をクリックします。

4. ファイルを保存します。

5. アーカイブを展開し、解凍します。

6. **oc** バイナリーをパスにあるディレクトリーに移動します。
PATH を確認するには、ターミナルを開き、以下のコマンドを実行します。

```
$ echo $PATH
```

OpenShift CLI のインストール後に、**oc** コマンドを使用して利用できます。

```
$ oc <command>
```

2.1.2.3. RPM を使用した OpenShift CLI のインストール

Red Hat Enterprise Linux (RHEL) の場合、Red Hat アカウントに有効な OpenShift Container Platform サブスクリプションがある場合は、OpenShift CLI (**oc**) を RPM としてインストールできます。

前提条件

- root または sudo の権限が必要です。

手順

1. Red Hat Subscription Manager に登録します。

```
# subscription-manager register
```

2. 最新のサブスクリプションデータをプルします。

```
# subscription-manager refresh
```

3. 利用可能なサブスクリプションを一覧表示します。

```
# subscription-manager list --available --matches '*OpenShift*'
```

4. 直前のコマンドの出力で、OpenShift Container Platform サブスクリプションのプール ID を見つけ、これを登録されたシステムにアタッチします。

```
# subscription-manager attach --pool=<pool_id>
```

5. OpenShift Container Platform 4.7 で必要なリポジトリーを有効にします。

- Red Hat Enterprise Linux 8 の場合:

```
# subscription-manager repos --enable="rhocp-4.7-for-rhel-8-x86_64-rpms"
```

- Red Hat Enterprise Linux 7 の場合:

```
# subscription-manager repos --enable="rhel-7-server-ose-4.7-rpms"
```

6. **openshift-clients** パッケージをインストールします。

```
# yum install openshift-clients
```

CLI のインストール後は、**oc** コマンドを使用して利用できます。

```
$ oc <command>
```

2.1.2.4. Homebrew を使用した OpenShift CLI のインストール

macOS の場合、Homebrew パッケージマネージャーを使用して OpenShift CLI (**oc**) をインストールできます。

前提条件

- Homebrew (**brew**) がインストールされている必要があります。

手順

- 以下のコマンドを実行して openshift-cli パッケージをインストールします。

```
$ brew install openshift-cli
```

2.1.3. OpenShift CLI へのログイン

OpenShift CLI (**oc**) にログインしてクラスターにアクセスし、これを管理できます。

前提条件

- OpenShift Container Platform クラスターへのアクセスが必要です。
- OpenShift CLI (**oc**) がインストールされている必要があります。

注記

HTTP プロキシーサーバー上でのみアクセスできるクラスターにアクセスするには、**HTTP_PROXY**、**HTTPS_PROXY** および **NO_PROXY** 変数を設定できます。これらの環境変数は、クラスターとのすべての通信が HTTP プロキシーを経由するように **oc** CLI で使用されます。

認証ヘッダーは、HTTPS トランスポートを使用する場合にのみ送信されます。

手順

1. **oc login** コマンドを入力し、ユーザー名を渡します。

```
$ oc login -u user1
```

2. プロンプトが表示されたら、必要な情報を入力します。

出力例

```
Server [https://localhost:8443]: https://openshift.example.com:6443 (1)
The server uses a certificate signed by an unknown authority.
You can bypass the certificate check, but any data you send to the server could be
intercepted by others.
Use insecure connections? (y/n): y (2)

Authentication required for https://openshift.example.com:6443 (openshift)
Username: user1
Password: (3)
Login successful.

You don't have any projects. You can try to create a new project, by running

    oc new-project <projectname>

Welcome! See 'oc help' to get started.
```

(1) OpenShift Container Platform サーバー URL を入力します。

(2) 非セキュアな接続を使用するかどうかを入力します。

(3) ユーザーのパスワードを入力します。

注記

Web コンソールにログインしている場合には、トークンおよびサーバー情報を含む **oc login** コマンドを生成できます。このコマンドを使用して、対話プロンプトなしに OpenShift Container Platform CLI にログインできます。コマンドを生成するには、Web コンソールの右上にあるユーザー名のドロップダウンメニューから **Copy login command** を選択します。

これで、プロジェクトを作成でき、クラスターを管理するための他のコマンドを実行することができます。

2.1.4. OpenShift CLI の使用

以下のセクションで、CLI を使用して一般的なタスクを実行する方法を確認します。

2.1.4.1. プロジェクトの作成

新規プロジェクトを作成するには、**oc new-project** コマンドを使用します。

```
$ oc new-project my-project
```

出力例

```
Now using project "my-project" on server "https://openshift.example.com:6443".
```

2.1.4.2. 新しいアプリケーションの作成

新規アプリケーションを作成するには、**oc new-app** コマンドを使用します。

```
$ oc new-app https://github.com/sclorg/cakephp-ex
```

出力例

```
--> Found image 40de956 (9 days old) in imagestream "openshift/php" under tag "7.2" for "php"

...

    Run 'oc status' to view your app.
```

2.1.4.3. Pod の表示

現在のプロジェクトの Pod を表示するには、**oc get pods** コマンドを使用します。

注記

Pod 内で **oc** を実行し、namespace を指定しない場合、Pod の namespace はデフォルトで使用されます。

```
$ oc get pods -o wide
```

出力例

```
NAME                READY   STATUS      RESTARTS   AGE     IP           NODE
NOMINATED NODE
cakephp-ex-1-build  0/1     Completed   0          5m45s   10.131.0.10  ip-10-0-141-74.ec2.internal
<none>
cakephp-ex-1-deploy 0/1     Completed   0          3m44s   10.129.2.9   ip-10-0-147-65.ec2.internal
<none>
cakephp-ex-1-ktz97  1/1     Running     0          3m33s   10.128.2.11  ip-10-0-168-105.ec2.internal
<none>
```

2.1.4.4. Pod ログの表示

特定の Pod のログを表示するには、**oc logs** コマンドを使用します。

```
$ oc logs cakephp-ex-1-deploy
```

出力例

```
--> Scaling cakephp-ex-1 to 1
--> Success
```

2.1.4.5. 現在のプロジェクトの表示

現在のプロジェクトを表示するには、**oc project** コマンドを使用します。

```
$ oc project
```

出力例

```
Using project "my-project" on server "https://openshift.example.com:6443".
```

2.1.4.6. 現在のプロジェクトのステータスの表示

サービス、デプロイメント、およびビルド設定などの現在のプロジェクトについての情報を表示するには、**oc status** コマンドを使用します。

```
$ oc status
```

出力例

```
In project my-project on server https://openshift.example.com:6443

svc/cakephp-ex - 172.30.236.80 ports 8080, 8443
  dc/cakephp-ex deploys istag/cakephp-ex:latest <-
    bc/cakephp-ex source builds https://github.com/sclorg/cakephp-ex on openshift/php:7.2
    deployment #1 deployed 2 minutes ago - 1 pod

3 infos identified, use 'oc status --suggest' to see details.
```

2.1.4.7. サポートされる API のリソースの一覧表示

サーバー上でサポートされる API リソースの一覧を表示するには、**oc api-resources** コマンドを使用します。

```
$ oc api-resources
```

出力例

```
NAME                              SHORTNAMES   APIGROUP                       NAMESPACED   KIND
bindings                                                                      true         Binding
componentstatuses                 cs                                          false        ComponentStatus
configmaps                        cm                                          true         ConfigMap
...
```

2.1.5. ヘルプの表示

CLI コマンドおよび OpenShift Container Platform リソースに関するヘルプを以下の方法で表示することができます。

- 利用可能なすべての CLI コマンドの一覧および説明を表示するには、**oc help** を使用します。

  例: CLI についての一般的なヘルプの表示

  ```
  $ oc help
  ```

  出力例

  ```
  OpenShift Client

  This client helps you develop, build, deploy, and run your applications on any OpenShift or
  Kubernetes compatible
  platform. It also includes the administrative commands for managing a cluster under the 'adm'
  subcommand.

  Usage:
    oc [flags]

  Basic Commands:
    login           Log in to a server
    new-project     Request a new project
    new-app         Create a new application

  ...
  ```

- 特定の CLI コマンドについてのヘルプを表示するには、**--help** フラグを使用します。

  例: **oc create** コマンドについてのヘルプの表示

  ```
  $ oc create --help
  ```

  出力例

  ```
  Create a resource by filename or stdin
  ```

```
JSON and YAML formats are accepted.

Usage:
  oc create -f FILENAME [flags]

...
```

- 特定リソースについての説明およびフィールドを表示するには、**oc explain** コマンドを使用します。

  例: **Pod** リソースのドキュメントの表示

```
$ oc explain pods
```

  出力例

```
KIND:     Pod
VERSION:  v1

DESCRIPTION:
     Pod is a collection of containers that can run on a host. This resource is
     created by clients and scheduled onto hosts.

FIELDS:
   apiVersion <string>
     APIVersion defines the versioned schema of this representation of an
     object. Servers should convert recognized schemas to the latest internal
     value, and may reject unrecognized values. More info:
     https://git.k8s.io/community/contributors/devel/api-conventions.md#resources

...
```

2.1.6. OpenShift CLI からのログアウト

OpenShift CLI からログアウトし、現在のセッションを終了することができます。

- **oc logout** コマンドを使用します。

```
$ oc logout
```

  出力例

```
Logged "user1" out on "https://openshift.example.com"
```

これにより、サーバーから保存された認証トークンが削除され、設定ファイルから除去されます。

2.2. OPENSHIFT CLI の設定

2.2.1. タブ補完の有効化

Bash または Zsh シェルのタブ補完を有効にできます。

2.2.1.1. Bash のタブ補完を有効にする

OpenShift CLI (**oc**) ツールをインストールした後に、タブ補完を有効にして **oc** コマンドの自動補完を実行するか、または Tab キーを押す際にオプションの提案が表示されるようにできます。次の手順では、Bash シェルのタブ補完を有効にします。

前提条件

- OpenShift CLI (**oc**) がインストールされている必要があります。
- **bash-completion** パッケージがインストールされている。

手順

1. Bash 補完コードをファイルに保存します。

```
$ oc completion bash > oc_bash_completion
```

2. ファイルを **/etc/bash_completion.d/** にコピーします。

```
$ sudo cp oc_bash_completion /etc/bash_completion.d/
```

さらにファイルをローカルディレクトリーに保存した後に、これを **.bashrc** ファイルから取得できるようにすることができます。

タブ補完は、新規ターミナルを開くと有効にされます。

2.2.1.2. Zsh のタブ補完を有効にする

OpenShift CLI (**oc**) ツールをインストールした後に、タブ補完を有効にして **oc** コマンドの自動補完を実行するか、または Tab キーを押す際にオプションの提案が表示されるようにできます。次の手順では、Zsh シェルのタブ補完を有効にします。

前提条件

- OpenShift CLI (**oc**) がインストールされている必要があります。

手順

- **oc** のタブ補完を **.zshrc** ファイルに追加するには、次のコマンドを実行します。

```
$ cat >>~/.zshrc<<EOF
if [ $commands[oc] ]; then
  source <(oc completion zsh)
  compdef _oc oc
fi
EOF
```

タブ補完は、新規ターミナルを開くと有効にされます。

2.3. MANAGING CLI PROFILES

CLI 設定ファイルでは、CLI ツールの概要 で使用するさまざまなプロファイルまたはコンテキストを設定できます。コンテキストは、ユーザー認証 および ニックネーム と関連付けられた OpenShift Container Platform サーバー情報から設定されます。

2.3.1. CLI プロファイル間のスイッチについて

CLI 操作を使用する場合に、コンテキストを使用すると、複数の OpenShift Container Platform サーバーまたはクラスターにまたがって、複数ユーザー間の切り替えが簡単になります。ニックネームを使用すると、コンテキスト、ユーザーの認証情報およびクラスターの詳細情報の省略された参照を提供することで、CLI 設定の管理が容易になります。CLI を使用して初めてログインした後、OpenShift Container Platform は **~/.kube/config** ファイルを作成します (すでに存在しない場合)。**oc login** 操作中に自動的に、または CLI プロファイルを手動で設定することにより、より多くの認証と接続の詳細が CLI に提供されると、更新された情報が設定ファイルに保存されます。

CLI 設定ファイル

```
apiVersion: v1
clusters: (1)
- cluster:
    insecure-skip-tls-verify: true
    server: https://openshift1.example.com:8443
  name: openshift1.example.com:8443
- cluster:
    insecure-skip-tls-verify: true
    server: https://openshift2.example.com:8443
  name: openshift2.example.com:8443
contexts: (2)
- context:
    cluster: openshift1.example.com:8443
    namespace: alice-project
    user: alice/openshift1.example.com:8443
  name: alice-project/openshift1.example.com:8443/alice
- context:
    cluster: openshift1.example.com:8443
    namespace: joe-project
    user: alice/openshift1.example.com:8443
  name: joe-project/openshift1/alice
current-context: joe-project/openshift1.example.com:8443/alice (3)
kind: Config
preferences: {}
users: (4)
- name: alice/openshift1.example.com:8443
  user:
    token: xZHd2piv5_9vQrg-SKXRJ2Dsl9SceNJdhNTljEKTb8k
```

(1) **clusters** セクションは、マスターサーバーのアドレスを含む OpenShift Container Platform クラスターの接続の詳細を定義します。この例では、1 つのクラスターのニックネームは **openshift1.example.com:8443** で、もう 1 つのクラスターのニックネームは **openshift2.example.com:8443** となっています。

(2) この **contexts** セクションでは、2 つのコンテキストを定義します。1 つは **alice-project/openshift1.example.com:8443/alice** というニックネームで、**alice-project** プロジェクト、**openshift1.example.com:8443** クラスター、および **alice** ユーザーを使用します。もう 1 つは **joe-project/openshift1.example.com:8443/alice** というニックネームで、**joe-project** プロジェクト、**openshift1.example.com:8443** クラスター、および **alice** ユーザーを使用します。

3 **current-context** パラメーターは、**joe-project/openshift1.example.com:8443/alice** コンテキストが現在使用中であることを示しています。これにより、**alice** ユーザーは

4 **users** セクションは、ユーザーの認証情報を定義します。この例では、ユーザーニックネーム **alice/openshift1.example.com:8443** は、アクセストークンを使用します。

CLI は、実行時にロードされ、コマンドラインから指定されたオーバーライドオプションとともにマージされる複数の設定ファイルをサポートできます。ログイン後に、**oc status** または **oc project** コマンドを使用して、現在の作業環境を確認できます。

現在の作業環境の確認

```
$ oc status
```

出力例

```
oc status
In project Joe's Project (joe-project)

service database (172.30.43.12:5434 -> 3306)
  database deploys docker.io/openshift/mysql-55-centos7:latest
    #1 deployed 25 minutes ago - 1 pod

service frontend (172.30.159.137:5432 -> 8080)
  frontend deploys origin-ruby-sample:latest <-
    builds https://github.com/openshift/ruby-hello-world with joe-project/ruby-20-centos7:latest
    #1 deployed 22 minutes ago - 2 pods

To see more information about a service or deployment, use 'oc describe service <name>' or 'oc describe dc <name>'.
You can use 'oc get all' to see lists of each of the types described in this example.
```

現在のプロジェクトの一覧表示

```
$ oc project
```

出力例

```
Using project "joe-project" from context named "joe-project/openshift1.example.com:8443/alice" on server "https://openshift1.example.com:8443".
```

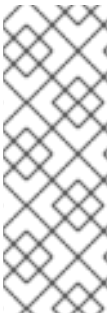
oc login コマンドを再度実行し、対話式プロセス中に必要な情報を指定して、ユーザー認証情報およびクラスターの詳細の他の組み合わせを使用してログインできます。コンテキストが存在しない場合は、コンテキストが指定される情報に基づいて作成されます。すでにログインしている場合で、現行ユーザーがアクセス可能な別のプロジェクトに切り替える場合には、**oc project** コマンドを使用してプロジェクトの名前を入力します。

```
$ oc project alice-project
```

出力例

```
Now using project "alice-project" on server "https://openshift1.example.com:8443".
```

出力に示されるように、いつでも **oc config view** コマンドを使用して、現在の CLI 設定を表示できます。高度な使用方法で利用できる CLI 設定コマンドが他にもあります。



注記

管理者の認証情報にアクセスできるが、デフォルトのシステムユーザー**system:admin**としてログインしていない場合は、認証情報が CLI 設定ファイルに残っている限り、いつでもこのユーザーとして再度ログインできます。以下のコマンドはログインを実行し、デフォルトプロジェクトに切り替えます。

```
$ oc login -u system:admin -n default
```

2.3.2. CLI プロファイルの手動設定

注記

このセクションでは、CLI 設定の高度な使用方法について説明します。ほとんどの場合、**oc login** コマンドおよび **oc project** コマンドを使用してログインし、コンテキスト間とプロジェクト間の切り替えを実行できます。

CLI 設定ファイルを手動で設定する必要がある場合は、ファイルを直接変更せずに **oc config** コマンドを使用することができます。**oc config** コマンドには、この目的で役立ついくつかのサブコマンドが含まれています。

表2.1 CLI 設定サブコマンド

サブコマンド	使用法
set-cluster	CLI 設定ファイルにクラスターエントリーを設定します。参照されるクラスターのニックネームがすでに存在する場合、指定情報はマージされます。 `$ oc config set-cluster <cluster_nickname> [--server=<master_ip_or_fqdn>] [--certificate-authority=<path/to/certificate/authority>] [--api-version=<apiversion>] [--insecure-skip-tls-verify=true]`
set-context	CLI 設定ファイルにコンテキストエントリーを設定します。参照されるコンテキストのニックネームがすでに存在する場合、指定情報はマージされます。 `$ oc config set-context <context_nickname> [--cluster=<cluster_nickname>] [--user=<user_nickname>] [--namespace=<namespace>]`
use-context	指定されたコンテキストのニックネームを使用して、現在のコンテキストを設定します。 `$ oc config use-context <context_nickname>`

サブコマンド	使用法
set	CLI 設定ファイルに個別の値を設定します。 $ oc config set <property_name> <property_value> **<property_name>** はドットで区切られた名前です。ここで、それぞれのトークンは属性名またはマップキーのいずれかを表します。**<property_value>** は設定される新しい値です。
unset	CLI 設定ファイルでの個別の値の設定を解除します。 $ oc config unset <property_name> **<property_name>** はドットで区切られた名前です。ここで、それぞれのトークンは属性名またはマップキーのいずれかを表します。

view	現在使用中のマージされた CLI 設定を表示します。
	
	$ oc config view
	
	指定された CLI 設定ファイルの結果を表示します。
	
	$ oc config view --config=<specific_filename>

使用例

- アクセストークンを使用するユーザーとしてログインします。このトークンは **alice** ユーザーによって使用されます。

```
$ oc login https://openshift1.example.com --
token=ns7yVhuRNpDM9cgzfhhxQ7bM5s7N2ZVrkZepSRf4LC0
```

- 自動的に作成されたクラスターエントリーを表示します。

```
$ oc config view
```

出力例

```
apiVersion: v1
clusters:
- cluster:
    insecure-skip-tls-verify: true
    server: https://openshift1.example.com
  name: openshift1-example-com
contexts:
- context:
    cluster: openshift1-example-com
    namespace: default
    user: alice/openshift1-example-com
```

```
  name: default/openshift1-example-com/alice
current-context: default/openshift1-example-com/alice
kind: Config
preferences: {}
users:
- name: alice/openshift1.example.com
  user:
    token: ns7yVhuRNpDM9cgzfhhxQ7bM5s7N2ZVrkZepSRf4LC0
```

- 現在のコンテキストを更新して、ユーザーが必要な namespace にログインできるようにします。

```
$ oc config set-context `oc config current-context` --namespace=<project_name>
```

- 現在のコンテキストを調べて、変更が実装されていることを確認します。

```
$ oc whoami -c
```

後続のすべての CLI 操作は、オーバーライドする CLI オプションにより特に指定されていない限り、またはコンテキストが切り替わるまで、新しいコンテキストを使用します。

2.3.3. ルールの読み込みおよびマージ

CLI 設定のロードおよびマージ順序の CLI 操作を実行する際に、以下のルールを実行できます。

- CLI 設定ファイルは、以下の階層とマージルールを使用してワークステーションから取得されます。
  - **--config** オプションが設定されている場合、そのファイルのみが読み込まれます。フラグは一度設定され、マージは実行されません。
  - **$KUBECONFIG** 環境変数が設定されている場合は、これが使用されます。変数はパスの一覧である可能性があり、その場合、パスは 1 つにマージされます。値が変更される場合は、スタンザを定義するファイルで変更されます。値が作成される場合は、存在する最初のファイルで作成されます。ファイルがチェーン内に存在しない場合は、一覧の最後のファイルが作成されます。
  - または、**~/.kube/config** ファイルが使用され、マージは実行されません。
- 使用するコンテキストは、以下のフローの最初の一致に基づいて決定されます。
  - **--context** オプションの値。
  - CLI 設定ファイルの **current-context** 値。
  - この段階では空の値が許可されます。
- 使用するユーザーおよびクラスターが決定されます。この時点では、コンテキストがある場合とない場合があります。コンテキストは、以下のフローの最初の一致に基づいて作成されます。このフローは、ユーザー用に 1 回、クラスター用に 1 回実行されます。
  - ユーザー名の **--user** の値、およびクラスター名の **--cluster** オプション。
  - **--context** オプションがある場合は、コンテキストの値を使用します。
  - この段階では空の値が許可されます。

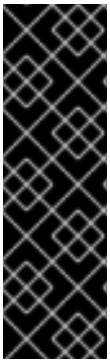
- 使用する実際のクラスター情報が決定されます。この時点では、クラスター情報がある場合とない場合があります。各クラスター情報は、以下のフローの最初の一致に基づいて構築されます。
  - 以下のコマンドラインオプションのいずれかの値。
    - **--server**
    - **--api-version**
    - **--certificate-authority**
    - **--insecure-skip-tls-verify**
  - クラスター情報および属性の値がある場合は、それを使用します。
  - サーバーロケーションがない場合は、エラーが生じます。
- 使用する実際のユーザー情報が決定されます。ユーザーは、クラスターと同じルールを使用して作成されます。ただし、複数の手法が競合することによって操作が失敗することから、ユーザーごとの1つの認証手法のみを使用できます。コマンドラインのオプションは、設定ファイルの値よりも優先されます。以下は、有効なコマンドラインのオプションです。
  - **--auth-path**
  - **--client-certificate**
  - **--client-key**
  - **--token**
- 欠落している情報がある場合には、デフォルト値が使用され、追加情報を求めるプロンプトが出されます。

2.4. プラグインによる OPENSHIFT CLI の拡張

デフォルトの **oc** コマンドを拡張するためにプラグインを作成およびインストールし、これを使用して OpenShift Container Platform CLI で新規および追加の複雑なタスクを実行できます。

2.4.1. CLI プラグインの作成

コマンドラインのコマンドを作成できる任意のプログラミング言語またはスクリプトで、OpenShift Container Platform CLI のプラグインを作成できます。既存の **oc** コマンドを上書きするプラグインを使用することはできない点に注意してください。



重要

現時点で OpenShift CLI プラグインはテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

手順

以下の手順では、**oc foo** コマンドの実行時にターミナルにメッセージを出力する単純な Bash プラグインを作成します。

1. **oc-foo** というファイルを作成します。
   プラグインファイルの名前を付ける際には、以下の点に留意してください。

   - プログインとして認識されるように、ファイルの名前は **oc-** または **kubectl-** で開始する必要があります。

   - ファイル名は、プラグインを起動するコマンドを判別するものとなります。たとえば、ファイル名が **oc-foo-bar** のプラグインは、**oc foo bar** のコマンドで起動します。また、コマンドにダッシュを含める必要がある場合には、アンダースコアを使用することもできます。たとえば、ファイル名が **oc-foo_bar** のプラグインは **oc foo-bar** のコマンドで起動できます。

2. 以下の内容をファイルに追加します。

```
#!/bin/bash

# optional argument handling
if [[ "$1" == "version" ]]
then
    echo "1.0.0"
    exit 0
fi

# optional argument handling
if [[ "$1" == "config" ]]
then
    echo $KUBECONFIG
    exit 0
fi

echo "I am a plugin named kubectl-foo"
```

OpenShift Container Platform CLI のこのプラグインをインストールした後に、**oc foo** コマンドを使用してこれを起動できます。

関連情報

- Go で作成されたプラグインの例については、 サンプルのプラグインリポジトリー を参照してください。

- Go でのプラグインの作成を支援する一連のユーティリティーについては、CLI ランタイムリポジトリー を参照してください。

2.4.2. CLI プラグインのインストールおよび使用

OpenShift Container Platform CLI のカスタムプラグインの作成後に、これが提供する機能を使用できるようインストールする必要があります。

重要

現時点で OpenShift CLI プラグインはテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

前提条件

- **oc** CLI ツールをインストールしていること。
- **oc-** または **kubectl-** で始まる CLI プラグインファイルがあること。

手順

1. 必要に応じて、プラグインファイルを実行可能な状態になるように更新します。

```
$ chmod +x <plugin_file>
```

2. ファイルを **PATH** の任意の場所に置きます (例: **/usr/local/bin/**)。

```
$ sudo mv <plugin_file> /usr/local/bin/.
```

3. **oc plugin list** を実行し、プラグインが一覧表示されることを確認します。

```
$ oc plugin list
```

出力例

```
The following compatible plugins are available:

/usr/local/bin/<plugin_file>
```

プラグインがここに一覧表示されていない場合、ファイルが **oc-** または **kubectl-** で開始されるものであり、実行可能な状態で **PATH** 上にあることを確認します。

4. プラグインによって導入される新規コマンドまたはオプションを起動します。
たとえば、**kubectl-ns** プラグインを サンプルのプラグインリポジトリー からビルドし、インストールしている場合、以下のコマンドを使用して現在の namespace を表示できます。

```
$ oc ns
```

プラグインを起動するためのコマンドはプラグインファイル名によって異なることに注意してください。たとえば、ファイル名が **oc-foo-bar** のプラグインは **oc foo bar** コマンドによって起動します。

2.5. OPENSHIFT CLI 開発者コマンド

2.5.1. 基本的な CLI コマンド

2.5.1.1. explain

特定リソースのドキュメントを表示します。

例: Pod のドキュメントの表示

```
$ oc explain pods
```

2.5.1.2. login

OpenShift Container Platform サーバーにログインし、後続の使用のためにログイン情報を保存します。

例: 対話型ログイン

```
$ oc login -u user1
```

2.5.1.3. new-app

ソースコード、テンプレート、またはイメージを指定して新規アプリケーションを作成します。

例: ローカル Git リポジトリーからの新規アプリケーションの作成

```
$ oc new-app .
```

例: リモート Git リポジトリーからの新規アプリケーションの作成

```
$ oc new-app https://github.com/sclorg/cakephp-ex
```

例: プライベートリモートリポジトリーからの新規アプリケーションの作成

```
$ oc new-app https://github.com/youruser/yourprivaterepo --source-secret=yoursecret
```

2.5.1.4. new-project

新規プロジェクトを作成し、設定のデフォルトのプロジェクトとしてこれに切り替えます。

例: 新規プロジェクトの作成

```
$ oc new-project myproject
```

2.5.1.5. project

別のプロジェクトに切り替えて、これを設定でデフォルトにします。

例: 別のプロジェクトへの切り替え

```
$ oc project test-project
```

2.5.1.6. projects

現在のアクティブなプロジェクトおよびサーバー上の既存プロジェクトについての情報を表示します。

例: すべてのプロジェクトの一覧表示

```
$ oc projects
```

2.5.1.7. status

現在のプロジェクトのハイレベルの概要を表示します。

例: 現在のプロジェクトのステータスの表示

```
$ oc status
```

2.5.2. CLI コマンドのビルドおよびデプロイ

2.5.2.1. cancel-build

実行中、保留中、または新規のビルドを取り消します。

例: ビルドの取り消し

```
$ oc cancel-build python-1
```

例: **python** ビルド設定からの保留中のすべてのビルドの取り消し

```
$ oc cancel-build buildconfig/python --state=pending
```

2.5.2.2. import-image

イメージリポジトリーから最新のタグおよびイメージ情報をインポートします。

例: 最新のイメージ情報のインポート

```
$ oc import-image my-ruby
```

2.5.2.3. new-build

ソースコードから新規のビルド設定を作成します。

例: ローカル **Git** リポジトリーからのビルド設定の作成

```
$ oc new-build .
```

例: リモート **Git** リポジトリーからのビルド設定の作成

```
$ oc new-build https://github.com/sclorg/cakephp-ex
```

2.5.2.4. rollback

アプリケーションを以前のデプロイメントに戻します。

例: 最後に成功したデプロイメントへのロールバック

```
$ oc rollback php
```

例: 特定バージョンへのロールバック

```
$ oc rollback php --to-version=3
```

2.5.2.5. rollout

新規ロールアウトを開始し、そのステータスまたは履歴を表示するか、またはアプリケーションの以前のバージョンにロールバックします。

例: 最後に成功したデプロイメントへのロールバック

```
$ oc rollout undo deploymentconfig/php
```

例: 最新状態のデプロイメントの新規ロールアウトの開始

```
$ oc rollout latest deploymentconfig/php
```

2.5.2.6. start-build

ビルド設定からビルドを開始するか、または既存ビルドをコピーします。

例: 指定されたビルド設定からのビルドの開始

```
$ oc start-build python
```

例: 以前のビルドからのビルドの開始

```
$ oc start-build --from-build=python-1
```

例: 現在のビルドに使用する環境変数の設定

```
$ oc start-build python --env=mykey=myvalue
```

2.5.2.7. tag

既存のイメージをイメージストリームにタグ付けします。

例: **ruby** イメージの **latest** タグを **2.0** タグのイメージを参照するように設定する

```
$ oc tag ruby:latest ruby:2.0
```

2.5.3. アプリケーション管理 CLI コマンド

2.5.3.1. annotate

1つ以上のリソースでアノテーションを更新します。

例: アノテーションのルートへの追加

```
$ oc annotate route/test-route haproxy.router.openshift.io/ip_whitelist="192.168.1.10"
```

例: ルートからのアノテーションの削除

```
$ oc annotate route/test-route haproxy.router.openshift.io/ip_whitelist-
```

2.5.3.2. apply

JSON または YAML 形式のファイル名または標準入力 (stdin) 別に設定をリソースに適用します。

例: **pod.json** の設定の **Pod** への適用

```
$ oc apply -f pod.json
```

2.5.3.3. autoscale

デプロイメントまたはレプリケーションコントローラーの自動スケーリングを実行します。

例: 最小の 2 つおよび最大の 5 つの **Pod** への自動スケーリング

```
$ oc autoscale deploymentconfig/parksmap-katacoda --min=2 --max=5
```

2.5.3.4. create

JSON または YAML 形式のファイル名または標準入力 (stdin) 別にリソースを作成します。

例: **pod.json** の内容を使用した **Pod** の作成

```
$ oc create -f pod.json
```

2.5.3.5. delete

リソースを削除します。

例: **parksmap-katacoda-1-qfqz4** という名前の **Pod** の削除

```
$ oc delete pod/parksmap-katacoda-1-qfqz4
```

例: **app=parksmap-katacoda** ラベルの付いたすべての **Pod** の削除

```
$ oc delete pods -l app=parksmap-katacoda
```

2.5.3.6. describe

特定のオブジェクトに関する詳細情報を返します。

例: **example** という名前のデプロイメントの記述

```
$ oc describe deployment/example
```

例: すべての Pod の記述

```
$ oc describe pods
```

2.5.3.7. edit

リソースを編集します。

例: デフォルトエディターを使用したデプロイメントの編集

```
$ oc edit deploymentconfig/parksmap-katacoda
```

例: 異なるエディターを使用したデプロイメントの編集

```
$ OC_EDITOR="nano" oc edit deploymentconfig/parksmap-katacoda
```

例: JSON 形式のデプロイメントの編集

```
$ oc edit deploymentconfig/parksmap-katacoda -o json
```

2.5.3.8. expose

ルートとしてサービスを外部に公開します。

例: サービスの公開

```
$ oc expose service/parksmap-katacoda
```

例: サービスの公開およびホスト名の指定

```
$ oc expose service/parksmap-katacoda --hostname=www.my-host.com
```

2.5.3.9. get

1 つ以上のリソースを表示します。

例: **default namespace** の Pod の一覧表示

```
$ oc get pods -n default
```

例: JSON 形式の **python** デプロイメントについての詳細の取得

```
$ oc get deploymentconfig/python -o json
```

2.5.3.10. label

1つ以上のリソースでラベルを更新します。

例: **python-1-mz2rf Pod** の **unhealthy** に設定されたラベル **status** での更新

```
$ oc label pod/python-1-mz2rf status=unhealthy
```

2.5.3.11. scale

レプリケーションコントローラーまたはデプロイメントの必要なレプリカ数を設定します。

例: **ruby-app** デプロイメントの 3 つの **Pod** へのスケーリング

```
$ oc scale deploymentconfig/ruby-app --replicas=3
```

2.5.3.12. secrets

プロジェクトのシークレットを管理します。

例: **my-pull-secret** の、**default** サービスアカウントによるイメージプルシークレットとしての使用を許可

```
$ oc secrets link default my-pull-secret --for=pull
```

2.5.3.13. serviceaccounts

サービスアカウントに割り当てられたトークンを取得するか、またはサービスアカウントの新規トークンまたは **kubeconfig** ファイルを作成します。

例: **default** サービスアカウントに割り当てられたトークンの取得

```
$ oc serviceaccounts get-token default
```

2.5.3.14. set

既存のアプリケーションリソースを設定します。

例: ビルド設定でのシークレットの名前の設定

```
$ oc set build-secret --source buildconfig/mybc mysecret
```

2.5.4. CLI コマンドのトラブルシューティングおよびデバッグ

2.5.4.1. attach

実行中のコンテナーにシェルを割り当てます。

例: **Pod python-1-mz2rf** の **python** コンテナーからの出力の取得

```
$ oc attach python-1-mz2rf -c python
```

2.5.4.2. cp

ファイルおよびディレクトリーのコンテナーへの/からのコピーを実行します。

例: **python-1-mz2rf Pod** からローカルファイルシステムへのファイルのコピー

```
$ oc cp default/python-1-mz2rf:/opt/app-root/src/README.md ~/mydirectory/.
```

2.5.4.3. debug

コマンドシェルを起動して、実行中のアプリケーションをデバッグします。

例: **python** デプロイメントのデバッグ

```
$ oc debug deploymentconfig/python
```

2.5.4.4. exec

コンテナーでコマンドを実行します。

例: **ls** コマンドの **Pod python-1-mz2rf** の **python** コンテナーでの実行

```
$ oc exec python-1-mz2rf -c python ls
```

2.5.4.5. logs

特定のビルド、ビルド設定、デプロイメント、または Pod のログ出力を取得します。

例: **python** デプロイメントからの最新ログのストリーミング

```
$ oc logs -f deploymentconfig/python
```

2.5.4.6. port-forward

1 つ以上のポートを Pod に転送します。

例: ポート **8888** でのローカルのリッスンおよび **Pod** のポート **5000** への転送

```
$ oc port-forward python-1-mz2rf 8888:5000
```

2.5.4.7. proxy

Kubernetes API サーバーに対してプロキシーを実行します。

例: **./local/www/** から静的コンテンツを提供するポート **8011** の **API** サーバーに対するプロキシーの実行

```
$ oc proxy --port=8011 --www=./local/www/
```

-

2.5.4.8. rsh

コンテナーへのリモートシェルセッションを開きます。

例: **python-1-mz2rf Pod** の最初のコンテナーでシェルセッションを開く

```
$ oc rsh python-1-mz2rf
```

2.5.4.9. rsync

ディレクトリーの内容の実行中の Pod コンテナーへの/からのコピーを実行します。変更されたファイルのみが、オペレーティングシステムから **rsync** コマンドを使用してコピーされます。

例: ローカルディレクトリーのファイルの **Pod** ディレクトリーとの同期

```
$ oc rsync ~/mydirectory/ python-1-mz2rf:/opt/app-root/src/
```

2.5.4.10. run

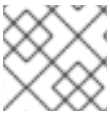
特定のイメージを実行する Pod を作成します。

例: **perl** イメージを実行する **Pod** の起動

```
$ oc run my-test --image=perl
```

2.5.4.11. wait

1 つ以上のリソースの特定の条件を待機します。



注記

このコマンドは実験的なもので、通知なしに変更される可能性があります。

例: **python-1-mz2rf Pod** の削除の待機

```
$ oc wait --for=delete pod/python-1-mz2rf
```

2.5.5. 上級開発者の CLI コマンド

2.5.5.1. api-resources

サーバーがサポートする API リソースの詳細の一覧を表示します。

例: サポートされている **API** リソースの一覧表示

```
$ oc api-resources
```

2.5.5.2. api-versions

サーバーがサポートする API バージョンの詳細の一覧を表示します。

例: サポートされている **API** バージョンの一覧表示

```
$ oc api-versions
```

2.5.5.3. auth

パーミッションを検査し、RBAC ロールを調整します。

例: 現行ユーザーが **Pod** ログを読み取ることができるかどうかのチェック

```
$ oc auth can-i get pods --subresource=log
```

例: ファイルの **RBAC** ロールおよびパーミッションの調整

```
$ oc auth reconcile -f policy.json
```

2.5.5.4. cluster-info

マスターおよびクラスターサービスのアドレスを表示します。

例: クラスター情報の表示

```
$ oc cluster-info
```

2.5.5.5. extract

設定マップまたはシークレットの内容を抽出します。設定マップまたはシークレットのそれぞれのキーがキーの名前を持つ別個のファイルとして作成されます。

例: **ruby-1-ca** 設定マップの内容の現行ディレクトリーへのダウンロード

```
$ oc extract configmap/ruby-1-ca
```

例: **ruby-1-ca** 設定マップの内容の標準出力 **(stdout)** への出力

```
$ oc extract configmap/ruby-1-ca --to=-
```

2.5.5.6. idle

スケーラブルなリソースをアイドリングします。アイドリングされたサービスは、トラフィックを受信するとアイドリング解除されます。 これは **oc scale** コマンドを使用して手動でアイドリング解除することもできます。

例: **ruby-app** サービスのアイドリング

```
$ oc idle ruby-app
```

2.5.5.7. image

OpenShift Container Platform クラスターでイメージを管理します。

例: イメージの別のタグへのコピー

```
$ oc image mirror myregistry.com/myimage:latest myregistry.com/myimage:stable
```

2.5.5.8. observe

リソースの変更を監視し、それらの変更に対するアクションを取ります。

例: サービスへの変更の監視

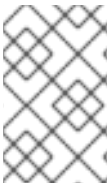
```
$ oc observe services
```

2.5.5.9. patch

JSON または YAML 形式のストテラテジーに基づくマージパッチを使用してオブジェクトの 1 つ以上のフィールドを更新します。

例: ノード **node1** の **spec.unschedulable** フィールドの **true** への更新

```
$ oc patch node/node1 -p '{"spec":{"unschedulable":true}}'
```



注記

カスタムリソース定義 (Custom Resource Definition) のパッチを適用する必要がある場合、コマンドに **--type merge** オプションまたは **--type json** オプションを含める必要があります。

2.5.5.10. policy

認可ポリシーを管理します。

例: **edit** ロールの現在のプロジェクトの **user1** への追加

```
$ oc policy add-role-to-user edit user1
```

2.5.5.11. process

リソースの一覧に対してテンプレートを処理します。

例: **template.json** をリソース一覧に変換し、 **oc create** に渡す

```
$ oc process -f template.json | oc create -f -
```

2.5.5.12. registry

OpenShift Container Platform で統合レジストリーを管理します。

例: 統合レジストリーについての情報の表示

```
$ oc registry info
```

2.5.5.13. replace

指定された設定ファイルに基づいて既存オブジェクトを変更します。

例: **pod.json** の内容を使用した Pod の更新

```
$ oc replace -f pod.json
```

2.5.6. CLI コマンドの設定

2.5.6.1. completion

指定されたシェルのシェル補完コードを出力します。

例: Bash の補完コードの表示

```
$ oc completion bash
```

2.5.6.2. config

クライアント設定ファイルを管理します。

例: 現在の設定の表示

```
$ oc config view
```

例: 別のコンテキストへの切り替え

```
$ oc config use-context test-context
```

2.5.6.3. logout

現行のセッションからログアウトします。

例: 現行セッションの終了

```
$ oc logout
```

2.5.6.4. whoami

現行セッションに関する情報を表示します。

例: 現行の認証ユーザーの表示

```
$ oc whoami
```

2.5.7. 他の開発者 CLI コマンド

2.5.7.1. help

CLI の一般的なヘルプ情報および利用可能なコマンドの一覧を表示します。

例: 利用可能なコマンドの表示

```
$ oc help
```

例: **new-project** コマンドのヘルプの表示

```
$ oc help new-project
```

2.5.7.2. plugin

ユーザーの **PATH** に利用可能なプラグインを一覧表示します。

例: 利用可能なプラグインの一覧表示

```
$ oc plugin list
```

2.5.7.3. version

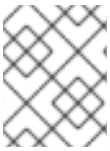
oc クライアントおよびサーバーのバージョンを表示します。

例: バージョン情報の表示

```
$ oc version
```

クラスター管理者の場合、OpenShift Container Platform サーバーバージョンも表示されます。

2.6. OPENSHIFT CLI 管理者コマンド



注記

これらの管理者コマンドを使用するには、**cluster-admin** または同等のパーミッションが必要です。

2.6.1. クラスター管理 CLI コマンド

2.6.1.1. inspect

特定のリソースについてのデバッグ情報を収集します。

注記

このコマンドは実験的なもので、通知なしに変更される可能性があります。

例: OpenShift API サーバークラスター Operator のデバッグデータの収集

```
$ oc adm inspect clusteroperator/openshift-apiserver
```

2.6.1.2. must-gather

問題のデバッグに必要なクラスターの現在の状態についてのデータを一括収集します。

注記

このコマンドは実験的なもので、通知なしに変更される可能性があります。

例: デバッグ情報の収集

```
$ oc adm must-gather
```

2.6.1.3. top

サーバー上のリソースの使用状況についての統計を表示します。

例: Pod の CPU およびメモリーの使用状況の表示

```
$ oc adm top pods
```

例: イメージの使用状況の統計の表示

```
$ oc adm top images
```

2.6.2. ノード管理 CLI コマンド

2.6.2.1. cordon

ノードにスケジュール対象外 (unschedulable) のマークを付けます。ノードにスケジュール対象外のマークを手動で付けると、いずれの新規 Pod もノードでスケジュールされなくなりますが、ノード上の既存の Pod にはこれによる影響がありません。

例: **node1** にスケジュール対象外のマークを付ける

```
$ oc adm cordon node1
```

2.6.2.2. drain

メンテナーンスの準備のためにノードをドレイン (解放) します。

例: **node1** のドレイン (解放)

```
$ oc adm drain node1
```

2.6.2.3. node-logs

ノードのログを表示し、フィルターします。

例: **NetworkManager** のログの取得

```
$ oc adm node-logs --role master -u NetworkManager.service
```

2.6.2.4. taint

1 つ以上のノードでテイントを更新します。

例: ユーザーのセットに対してノードを専用に割り当てるためのテイントの追加

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

例: ノード **node1** からキー **dedicated** のあるテイントを削除する

```
$ oc adm taint nodes node1 dedicated-
```

2.6.2.5. uncordon

ノードにスケジュール対象 (schedulable) のマークを付けます。

例: **node1** にスケジュール対象のマークを付ける

```
$ oc adm uncordon node1
```

2.6.3. セキュリティーおよびポリシー CLI コマンド

2.6.3.1. certificate

証明書署名要求 (CSR) を承認するか、または拒否します。

例: CSR の承認

```
$ oc adm certificate approve csr-sqgzp
```

2.6.3.2. groups

クラスター内のグループを管理します。

例: 新規グループの作成

```
$ oc adm groups new my-group
```

2.6.3.3. new-project

新規プロジェクトを作成し、管理オプションを指定します。

例: ノードセレクターを使用した新規プロジェクトの作成

```
$ oc adm new-project myproject --node-selector='type=user-node,region=east'
```

2.6.3.4. pod-network

クラスター内の Pod ネットワークを管理します。

例: **project1** および **project2** を他の非グローバルプロジェクトから分離する

```
$ oc adm pod-network isolate-projects project1 project2
```

2.6.3.5. policy

クラスター上のロールおよびポリシーを管理します。

例: すべてのプロジェクトについて **edit** ロールを **user1** に追加する

```
$ oc adm policy add-cluster-role-to-user edit user1
```

例: **privileged SCC (security context constraint)** のサービスアカウントへの追加

```
$ oc adm policy add-scc-to-user privileged -z myserviceaccount
```

2.6.4. メンテナーンス CLI コマンド

2.6.4.1. migrate

使用されるサブコマンドに応じて、クラスターのリソースを新規バージョンまたはフォーマットに移行します。

例: 保存されたすべてのオブジェクトの更新の実行

```
$ oc adm migrate storage
```

例: Pod のみの更新の実行

```
$ oc adm migrate storage --include=pods
```

2.6.4.2. prune

サーバーから古いバージョンのリソースを削除します。

例: ビルド設定がすでに存在しないビルドを含む、古いビルドのプルーニング

```
$ oc adm prune builds --orphans
```

2.6.5. 設定 CLI コマンド

2.6.5.1. create-bootstrap-project-template

ブートストラッププロジェクトテンプレートを作成します。

例: YAML 形式でのブートストラッププロジェクトテンプレートの標準出力 (stdout) への出力

```
$ oc adm create-bootstrap-project-template -o yaml
```

2.6.5.2. create-error-template

エラーページをカスタマイズするためのテンプレートを作成します。

例: エラーページのテンプレートの標準出力 (stdout) への出力

```
$ oc adm create-error-template
```

2.6.5.3. create-kubeconfig

クライアント証明書から基本的な **.kubeconfig** ファイルを作成します。

例: 提供されるクライアント証明書を使用した **.kubeconfig** ファイルの作成

```
$ oc adm create-kubeconfig \
  --client-certificate=/path/to/client.crt \
  --client-key=/path/to/client.key \
  --certificate-authority=/path/to/ca.crt
```

2.6.5.4. create-login-template

ログインページをカスタマイズするためのテンプレートを作成します。

例: ログインページのテンプレートの標準出力 (stdout) への出力

```
$ oc adm create-login-template
```

2.6.5.5. create-provider-selection-template

プロバイダー選択ページをカスタマイズするためのテンプレートを作成します。

例: プロバイダー選択ページのテンプレートの標準出力 (stdout) への出力

```
$ oc adm create-provider-selection-template
```

2.6.6. 他の管理者 CLI コマンド

2.6.6.1. build-chain

ビルドの入力と依存関係を出力します。

例: **perl** イメージストリームの依存関係の出力

```
$ oc adm build-chain perl
```

2.6.6.2. completion

指定されたシェルについての **oc adm** コマンドのシェル補完コードを出力します。

例: Bash の **oc adm** 補完コードの表示

```
$ oc adm completion bash
```

2.6.6.3. config

クライアント設定ファイルを管理します。このコマンドは、**oc config** コマンドと同じ動作を実行します。

例: 現在の設定の表示

```
$ oc adm config view
```

例: 別のコンテキストへの切り替え

```
$ oc adm config use-context test-context
```

2.6.6.4. release

リリースについての情報の表示、またはリリースの内容の検査などの OpenShift Container Platform リリースプロセスの様々な側面を管理します。

例: 2 つのリリース間の変更ログの生成および **changelog.md** への保存

```
$ oc adm release info --changelog=/tmp/git \
    quay.io/openshift-release-dev/ocp-release:4.7.0-x86_64 \
    quay.io/openshift-release-dev/ocp-release:4.7.1-x86_64 \
    > changelog.md
```

2.6.6.5. verify-image-signature

ローカルのパブリック GPG キーを使用して内部レジストリーにインポートされたイメージのイメージ署名を検証します。

例: **nodejs** イメージ署名の検証

```
$ oc adm verify-image-signature \
    sha256:2bba968aedb7dd2aafe5fa8c7453f5ac36a0b9639f1bf5b03f95de325238b288 \
    --expected-identity 172.30.1.1:5000/openshift/nodejs:latest \
    --public-key /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release \
    --save
```

2.7. OC および KUBECTL コマンドの使用

Kubernetes のコマンドラインインターフェイス (CLI) **kubectl** は、Kubernetes クラスターに対してコマンドを実行するために使用されます。OpenShift Container Platform は認定 Kubernetes ディストリビューションであるため、OpenShift Container Platform に同梱されるサポート対象の **kubectl** バイナリーを使用するか、または **oc** バイナリーを使用して拡張された機能を取得できます。

2.7.1. oc バイナリー

oc バイナリーは **kubectl** バイナリーと同じ機能を提供しますが、これは、以下を含む OpenShift Container Platform 機能をネイティブにサポートするように拡張されています。

- **OpenShift Container Platform** リソースの完全サポート
  DeploymentConfig、 **BuildConfig**、**Route**、**ImageStream**、および **ImageStreamTag** オブジェクトなどのリソースは OpenShift Container Platform ディストリビューションに固有のリソースであり、標準の Kubernetes プリミティブにビルドされます。

- 認証
  oc バイナリーは、認証を可能にするビルトインの **login** コマンドを提供し、Kubernetes namespace を認証ユーザーにマップする OpenShift Container Platform プロジェクトを使って作業できるようにします。詳細は、認証について 参照してください。

- 追加コマンド
  追加コマンドの **oc new-app** などは、既存のソースコードまたは事前にビルドされたイメージを使用して新規アプリケーションを起動することを容易にします。同様に、追加コマンドの **oc new-project** により、デフォルトとして切り替えることができるプロジェクトを簡単に開始できるようになります。

重要

以前のバージョンの **oc** バイナリーをインストールしている場合、これを使用して OpenShift Container Platform 4.7 のすべてのコマンドを実行することはできません。最新の機能が必要な場合は、お使いの OpenShift Container Platform サーバーバージョンに対応する最新バージョンの **oc** バイナリーをダウンロードし、インストールする必要があります。

セキュリティー以外の API の変更は、古い **oc** バイナリーの更新を可能にするために、2 つ以上のマイナーリリース (例: 4.1 から 4.2、そして 4.3 へ) が必要です。新機能を使用するには新規の **oc** バイナリーが必要になる場合があります。4.3 サーバーには、4.2 **oc** バイナリーが使用できない機能が追加されている場合や、4.3 **oc** バイナリーには 4.2 サーバーでサポートされていない追加機能が含まれる場合があります。

表2.2 互換性に関する表

	X.Y (**oc** クライアント)	X.Y+N[a] (**oc** クライアント)
X.Y (サーバー)	1	3
X.Y+N[a] (サーバー)	2	1

[a] ここでは、**N** は、1以上の数値です。

1 完全に互換性がある。

2 **oc** クライアントは、サーバー機能にアクセスできない場合があります。

3 **oc** クライアントは、アクセスされるサーバーと互換性のないオプションおよび機能を提供する可能性があります。

2.7.2. kubectl バイナリー

kubectl バイナリーは、標準の Kubernetes 環境を使用する新規 OpenShift Container Platform ユーザー、または **kubectl** CLI を優先的に使用するユーザーの既存ワークフローおよびスクリプトをサポートする手段として提供されます。**kubectl** の既存ユーザーはバイナリーを引き続き使用し、OpenShift Container Platform クラスターへの変更なしに Kubernetes のプリミティブと対話できます。

OpenShift CLI のインストール 手順に従って、サポートされている **kubectl** バイナリーをインストールできます。**kubectl** バイナリーは、バイナリーをダウンロードする場合にアーカイブに含まれます。または RPM を使用して CLI のインストール時にインストールされます。

詳細は、kubectl のドキュメント を参照してください。

第3章 DEVELOPER CLI (ODO)

3.1. ODO リリースノート

3.1.1. odo version 2.5.0 への主な変更点および改善点

- **adler32** ハッシュを使用して各コンポーネントに一意のルートを作成します。
- リソースの割り当て用に devfile の追加フィールドをサポートします。
  - cpuRequest
  - cpuLimit
  - memoryRequest
  - memoryLimit
- **--deploy** フラグを **odo delete** コマンドに追加し、**odo deploy** コマンドを使用してデプロイされたコンポーネントを削除します。

```
$ odo delete --deploy
```

- **odo link** コマンドにマッピングサポートを追加します。
- **volume** コンポーネントの **ephemeral** フィールドを使用して一時ボリュームをサポートします。
- Telemetry オプトインを要求する際に、デフォルトの回答を **yes** に設定します。
- 追加の Telemetry データを devfile レジストリーに送信してメトリクスを向上させます。
- ブートストラップイメージを**registry.access.redhat.com/ocp-tools-4/odo-init-container-rhel8:1.1.11** に更新します。
- アップストリームリポジトリーは https://github.com/redhat-developer/odo から入手できます。

3.1.2. バグ修正

- 以前のバージョンでは、**.odo/env** ファイルが存在しない場合、**odo deploy** は失敗していました。このコマンドは、必要に応じて **.odo/env** ファイルを作成するようになりました。
- 以前のバージョンでは、**odo create** コマンドを使用したインタラクティブなコンポーネントの作成は、クラスターからの切断時に失敗しました。この問題は最新リリースで修正されました。

3.1.3. サポート

製品

エラーを見つけた場合や、**odo** の機能に関するバグが見つかった場合やこれに関する改善案をお寄せいただける場合は、Bugzilla に報告してください。製品タイプとして **OpenShift Developer Tools and Services** を選択し、**odo** をコンポーネントとして選択します。

問題の詳細情報をできる限り多く入力します。

ドキュメント

エラーを見つけた場合、またはドキュメントを改善するための提案がある場合は、最も関連性の高いドキュメントコンポーネントの Jira issue を提出してください。

3.2. ODO について

Red Hat OpenShift Developer CLI(**odo**) は、アプリケーションを OpenShift Container Platform および Kubernetes で作成するためのツールです。**odo** を使用すると、プラットフォームを詳細に理解しなくても、マイクロサービスベースのアプリケーションを Kubernetes クラスターで開発、テスト、デバッグ、デプロイできます。

odo は 作成とプッシュ のワークフローに従います。ユーザーとして 作成 すると、情報 (またはマニフェスト) が設定ファイルに保存されます。プッシュ すると、対応するリソースが Kubernetes クラスターに作成されます。この設定はすべて、シームレスなアクセスと機能のために Kubernetes API に格納されます。

odo は、**service** および **link** コマンドを使用して、コンポーネントおよびサービスをリンクします。**odo** は、クラスターの Kubernetes Operator に基づいてサービスを作成し、デプロイしてこれを実行します。サービスは、Operator Hub で利用可能な任意の Operator を使用して作成できます。サービスをリンクした後に、**odo** はサービス設定をコンポーネントに挿入します。その後、アプリケーションはこの設定を使用して、Operator がサポートするサービスと通信できます。

3.2.1. odo キー機能

odo は、Kubernetes の開発者フレンドリーなインターフェイスとなるように設計されており、以下を実行できます。

- 新規マニフェストを作成するか、または既存のマニフェストを使用して、Kubernetes クラスターでアプリケーションを迅速にデプロイします。
- Kubernetes 設定ファイルを理解および維持しなくても、コマンドを使用してマニフェストを簡単に作成および更新できます。
- Kubernetes クラスターで実行されるアプリケーションへのセキュアなアクセスを提供します。
- Kubernetes クラスターのアプリケーションの追加ストレージを追加および削除します。
- Operator がサポートするサービスを作成し、アプリケーションをそれらのサービスにリンクします。
- **odo** コンポーネントとしてデプロイされる複数のマイクロサービス間のリンクを作成します。
- IDE で **odo** を使用してデプロイしたアプリケーションをリモートでデバッグします。
- **odo**を使用して Kubernetes にデプロイされたアプリケーションを簡単にテスト

3.2.2. odo のコアとなる概念

odo は、Kubernetes の概念を開発者に馴染みのある用語に抽象化します。

アプリケーション

特定のタスクを実行するために使用される、クラウドネイティブなアプローチ で開発された通常のアプリケーション。

アプリケーション の例には、オンラインビデオストリーミング、オンラインショッピング、ホテルの予約システムなどがあります。

コンポーネント

個別に実行でき、デプロイできる Kubernetes リソースのセット。クラウドネイティブアプリケーションは、小規模で独立した、緩く結合された コンポーネント の集まりです。
コンポーネント の例には、API バックエンド、Web インターフェイス、支払いバックエンドなどがあります。

プロジェクト

ソースコード、テスト、ライブラリーを含む単一のユニット。

コンテキスト

単一コンポーネントのソースコード、テスト、ライブラリー、および **odo** 設定ファイルが含まれるディレクトリー。

URL

クラスター外からアクセスするためにコンポーネントを公開するメカニズム。

ストレージ

クラスター内の永続ストレージ。これは、再起動およびコンポーネントの再構築後もデータを永続化します。

サービス

コンポーネントに追加機能を提供する外部アプリケーション。
サービス の例には、PostgreSQL、MySQL、Redis、RabbitMQ などがあります。

odo では、サービスは OpenShift Service Catalog からプロビジョニングされ、クラスター内で有効にされる必要があります。

devfile

コンテナー化された開発環境を定義するためのオープン標準。これにより、開発者用ツールはワークフローを簡素化し、高速化することができます。詳細は、https://devfile.io のドキュメントを参照してください。
公開されている **devfile** レジストリーに接続するか、またはセキュアなレジストリーをインストールできます。

3.2.3. odo でのコンポーネントの一覧表示

odo は移植可能な **devfile** 形式を使用してコンポーネントおよびそれらの関連する URL、ストレージ、およびサービスを記述します。**odo** はさまざまな devfile レジストリーに接続して、さまざまな言語およびフレームワークの devfile をダウンロードできます。devfile 情報を取得するために **odo** で使用されるレジストリーを管理する方法についての詳細は、**odo registry** コマンドのドキュメントを参照してください。

odo catalog list components コマンドを使用して、さまざまなレジストリーで利用可能な **devfile** をすべて一覧表示できます。

手順

1. **odo** でクラスターにログインします。

```
$ odo login -u developer -p developer
```

2. 利用可能な **odo** コンポーネントを一覧表示します。

```
$ odo catalog list components
```

出力例

```
Odo Devfile Components:
NAME                        DESCRIPTION                                        REGISTRY
dotnet50                    Stack with .NET 5.0
DefaultDevfileRegistry
dotnet60                    Stack with .NET 6.0
DefaultDevfileRegistry
dotnetcore31                Stack with .NET Core 3.1
DefaultDevfileRegistry
go                          Stack with the latest Go version
DefaultDevfileRegistry
java-maven                  Upstream Maven and OpenJDK 11
DefaultDevfileRegistry
java-openliberty            Java application Maven-built stack using the Open Liberty ru...
DefaultDevfileRegistry
java-openliberty-gradle     Java application Gradle-built stack using the Open Liberty r...
DefaultDevfileRegistry
java-quarkus                Quarkus with Java
DefaultDevfileRegistry
java-springboot             Spring Boot® using Java
DefaultDevfileRegistry
java-vertx                  Upstream Vert.x using Java
DefaultDevfileRegistry
java-websphereliberty       Java application Maven-built stack using the WebSphere
Liber...    DefaultDevfileRegistry
java-websphereliberty-gradle    Java application Gradle-built stack using the WebSphere
Libe...    DefaultDevfileRegistry
java-wildfly                Upstream WildFly
DefaultDevfileRegistry
java-wildfly-bootable-jar   Java stack with WildFly in bootable Jar mode, OpenJDK 11
and...    DefaultDevfileRegistry
nodejs                      Stack with Node.js 14
DefaultDevfileRegistry
nodejs-angular              Stack with Angular 12
DefaultDevfileRegistry
nodejs-nextjs               Stack with Next.js 11
DefaultDevfileRegistry
nodejs-nuxtjs               Stack with Nuxt.js 2
DefaultDevfileRegistry
nodejs-react                Stack with React 17
DefaultDevfileRegistry
nodejs-svelte               Stack with Svelte 3
DefaultDevfileRegistry
nodejs-vue                  Stack with Vue 3
DefaultDevfileRegistry
php-laravel                 Stack with Laravel 8
DefaultDevfileRegistry
python                      Python Stack with Python 3.7
```

```
DefaultDevfileRegistry
python-django                Python3.7 with Django
DefaultDevfileRegistry
```

3.2.4. odo での Telemetry

odo は、オペレーティングシステムのメトリクス、RAM、CPU、コア数、**odo** バージョン、エラー、成功/失敗、および **odo** コマンドの完了までにかかる時間を含む、使用方法に関する情報を収集します。

odo preference コマンドを使用して Telemetry の承諾を変更できます。

- **odo preference set ConsentTelemetry true** は Telemetry を承諾します。
- **odo preference unset ConsentTelemetry** は Telemetry を無効化します。
- **odo preference view** は現在の設定を表示します。

3.3. ODO のインストール

odo CLI は、バイナリーをダウンロードして、Linux、Windows、または macOS にインストールできます。また、**odo** と **oc** の両方のバイナリーを使用して、OpenShift Container Platform クラスターと対話する OpenShift VS Code 拡張機能をインストールすることもできます。Red Hat Enterprise Linux(RHEL) の場合、**odo** CLI を RPM としてインストールできます。

注記

現時点では、**odo** はネットワークが制限された環境でのインストールをサポートしていません。

3.3.1. odo の Linux へのインストール

odo CLI はバイナリーとしてダウンロードでき、以下を含む複数のオペレーティングシステムおよびアーキテクチャーの tarball としてダウンロードできます。

オペレーティングシステム	バイナリー	Tarball
Linux	odo-linux-amd64	odo-linux-amd64.tar.gz
Linux on IBM Power	odo-linux-ppc64le	odo-linux-ppc64le.tar.gz
Linux on IBM Z および LinuxONE	odo-linux-s390x	odo-linux-s390x.tar.gz

手順

1. コンテンツゲートウェイ に移動し、オペレーティングシステムおよびアーキテクチャーに適したファイルをダウンロードします。
   - バイナリーをダウンロードする場合は、これを **odo** に変更します。

```
$ curl -L https://developers.redhat.com/content-gateway/rest/mirror/pub/openshift-v4/clients/odo/latest/odo-linux-amd64 -o odo
```

- tarball をダウンロードする場合は、バイナリーを展開します。

```
$ curl -L https://developers.redhat.com/content-gateway/rest/mirror/pub/openshift-v4/clients/odo/latest/odo-linux-amd64.tar.gz -o odo.tar.gz
$ tar xvzf odo.tar.gz
```

2. バイナリーのパーミッションを変更します。

```
$ chmod +x <filename>
```

3. **odo** バイナリーを、**PATH** にあるディレクトリーに配置します。
PATH を確認するには、以下のコマンドを実行します。

```
$ echo $PATH
```

4. **odo** がシステムで利用可能になっていることを確認します。

```
$ odo version
```

3.3.2. odo の Windows へのインストール

Windows 用の**odo** CLI は、バイナリーおよびアーカイブとしてダウンロードできます。

オペレーティングシステム	バイナリー	Tarball
Windows	odo-windows-amd64.exe	odo-windows-amd64.exe.zip

手順

1. コンテンツゲートウェイ に移動し、適切なファイルをダウンロードします。
   - バイナリーをダウンロードする場合は、名前を **odo.exe** に変更します。
   - アーカイブをダウンロードする場合は、ZIP プログラムでバイナリーを展開し、名前を **odo.exe** に変更します。
2. **odo.exe** バイナリーを **PATH** にあるディレクトリーに移動します。
PATH を確認するには、コマンドプロンプトを開いて以下のコマンドを実行します。

```
C:\> path
```

3. **odo** がシステムで利用可能になっていることを確認します。

```
C:\> odo version
```

3.3.3. odo の macOS へのインストール

macOS の **odo** CLI は、バイナリーおよび tarball としてダウンロードできます。

オペレーティングシステム	バイナリー	Tarball
macOS	odo-darwin-amd64	odo-darwin-amd64.tar.gz

手順

1. コンテンツゲートウェイ に移動し、適切なファイルをダウンロードします。
   - バイナリーをダウンロードする場合は、これを **odo** に変更します。

     ```
     $ curl -L https://developers.redhat.com/content-gateway/rest/mirror/pub/openshift-v4/clients/odo/latest/odo-darwin-amd64 -o odo
     ```

   - tarball をダウンロードする場合は、バイナリーを展開します。

     ```
     $ curl -L https://developers.redhat.com/content-gateway/rest/mirror/pub/openshift-v4/clients/odo/latest/odo-darwin-amd64.tar.gz -o odo.tar.gz
     $ tar xvzf odo.tar.gz
     ```

2. バイナリーのパーミッションを変更します。

   ```
   # chmod +x odo
   ```

3. **odo** バイナリーを、**PATH** にあるディレクトリーに配置します。
   PATH を確認するには、以下のコマンドを実行します。

   ```
   $ echo $PATH
   ```

4. **odo** がシステムで利用可能になっていることを確認します。

   ```
   $ odo version
   ```

3.3.4. odo の VS Code へのインストール

OpenShift VS Code 拡張 は、**odo** と **oc** バイナリーの両方を使用して OpenShift Container Platform クラスターと対話します。これらの機能を使用するには、OpenShift VS Code 拡張を VS Code にインストールします。

前提条件

- VS Code がインストールされていること。

手順

1. VS Code を開きます。
2. **Ctrl**+**P** で VS Code Quick Open を起動します。

3. 以下のコマンドを入力します。

```
$ ext install redhat.vscode-openshift-connector
```

3.3.5. RPM を使用した odo の Red Hat Enterprise Linux(RHEL) へのインストール

Red Hat Enterprise Linux(RHEL) の場合、**odo** CLI を RPM としてインストールできます。

手順

1. Red Hat Subscription Manager に登録します。

```
# subscription-manager register
```

2. 最新のサブスクリプションデータをプルします。

```
# subscription-manager refresh
```

3. 利用可能なサブスクリプションを一覧表示します。

```
# subscription-manager list --available --matches '*OpenShift Developer Tools and Services*'
```

4. 直前のコマンドの出力で、OpenShift Container Platform サブスクリプションの **Pool ID** フィールドを見つけ、これを登録されたシステムに割り当てます。

```
# subscription-manager attach --pool=<pool_id>
```

5. **odo** で必要なリポジトリーを有効にします。

```
# subscription-manager repos --enable="ocp-tools-4.9-for-rhel-8-x86_64-rpms"
```

6. **odo** パッケージをインストールします。

```
# yum install odo
```

7. **odo** がシステムで利用可能になっていることを確認します。

```
$ odo version
```

3.4. ODO CLI の設定

odo のグローバル設定は、デフォルトで **$HOME/.odo** ディレクトリーにある **preference.yaml** ファイルにあります。

GLOBALODOCONFIG 変数をエクスポートして、**preference.yaml** ファイルに別の場所を設定できます。

3.4.1. 現在の設定の表示

以下のコマンドを使用して、現在の **odo** CLI 設定を表示できます。

```
$ odo preference view
```

出力例

```
PARAMETER           CURRENT_VALUE
UpdateNotification
NamePrefix
Timeout
BuildTimeout
PushTimeout
Ephemeral
ConsentTelemetry    true
```

3.4.2. 値の設定

以下のコマンドを使用して、preference キーの値を設定できます。

```
$ odo preference set <key> <value>
```

注記

優先キーは大文字と小文字を区別しません。

コマンドの例

```
$ odo preference set updatenotification false
```

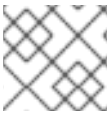
出力例

```
Global preference was successfully updated
```

3.4.3. 値の設定解除

以下のコマンドを使用して、preference キーの値の設定を解除できます。

```
$ odo preference unset <key>
```



注記

-f フラグを使用して確認を省略できます。

コマンドの例

```
$ odo preference unset updatenotification
? Do you want to unset updatenotification in the preference (y/N) y
```

出力例

```
Global preference was successfully updated
```

3.4.4. preference キーの表

以下の表は、**odo** CLI の preference キーを設定するために使用できるオプションを示しています。

preference キー	説明	デフォルト値
UpdateNotification	**odo** を更新する通知を表示するかどうかを制御します。	True
NamePrefix	**odo** リソースのデフォルト名接頭辞を設定します。例: **component** または **storage**。	現在のディレクトリー名
タイムアウト	Kubernetes サーバー接続チェックのタイムアウト。	1 秒
BuildTimeout	git コンポーネントのビルドが完了するまでのタイムアウト。	300 秒
PushTimeout	コンポーネントが起動するまで待機するタイムアウト。	240 秒
一時ストレージ	ソースコードを保存するために **odo** が **emptyDir** ボリュームを作成するかどうかを制御します。	True
ConsentTelemetry	**odo** がユーザーの **odo** の使用のために Telemetry を収集できるかどうかを制御します。	False

3.4.5. ファイルまたはパターンを無視する

アプリケーションのルートディレクトリーにある **.odoignore** ファイルを変更して、無視するファイルまたはパターンの一覧を設定できます。これは、**odo push** および **odo watch** の両方に適用されます。

.odoignore ファイルが存在 しない 場合、特定のファイルおよびフォルダーを無視するように **.gitignore** ファイルが代わりに使用されます。

.git ファイル、**.js** 拡張子のあるファイルおよびフォルダー **tests** を無視するには、以下を **.odoignore** または **.gitignore** ファイルのいずれかに追加します。

```
.git
*.js
tests/
```

.odoignore ファイルはすべての glob 表現を許可します。

3.5. ODO CLI リファレンス

3.5.1. odo build-images

odo は Dockerfile に基づいてコンテナーイメージをビルドし、それらのイメージをレジストリーにプッシュできます。

odo build-images コマンドを実行すると、**odo** は **image** タイプで **devfile.yaml** 内のすべてのコンポーネントを検索します。以下に例を示します。

```
components:
- image:
    imageName: quay.io/myusername/myimage
    dockerfile:
      uri: ./Dockerfile (1)
      buildContext: ${PROJECTS_ROOT} (2)
  name: component-built-from-dockerfile
```

(1) **uri** フィールドは、**devfile.yaml** を含むディレクトリーとの関連で使用する Dockerfile の相対パスを示します。devfile 仕様は **uri** が HTTP URL である可能性があることを示しますが、この場合は odo ではまだサポートされていません。

(2) **buildContext** は、ビルドコンテキストとして使用されるディレクトリーを示します。デフォルト値は **${PROJECTS_ROOT}** です。

各イメージコンポーネントについて、odo は **podman** または **docker** (この順序で最初に見つかったもの) を実行し、指定された Dockerfile、ビルドコンテキスト、および引数でイメージをビルドします。

--push フラグがコマンドに渡されると、イメージはビルド後にレジストリーにプッシュされます。

3.5.2. odo catalog

odo は異なる カタログ を使用して コンポーネント および サービス をデプロイします。

3.5.2.1. コンポーネント

odo は移植可能な **devfile** 形式を使用してコンポーネントを記述します。さまざまな devfile レジストリーに接続して、さまざまな言語およびフレームワークの devfile をダウンロードできます。詳細は、**odo registry** を参照してください。

3.5.2.1.1. コンポーネントの一覧表示

異なるレジストリーで利用可能な **devfile** の一覧を表示するには、以下のコマンドを実行します。

```
$ odo catalog list components
```

出力例

```
NAME            DESCRIPTION                       REGISTRY
go              Stack with the latest Go version     DefaultDevfileRegistry
java-maven      Upstream Maven and OpenJDK 11        DefaultDevfileRegistry
nodejs          Stack with Node.js 14                DefaultDevfileRegistry
php-laravel     Stack with Laravel 8                 DefaultDevfileRegistry
python          Python Stack with Python 3.7         DefaultDevfileRegistry
[...]
```

3.5.2.1.2. コンポーネントに関する情報の取得

特定のコンポーネントに関する詳細情報を取得するには、以下のコマンドを実行します。

```
$ odo catalog describe component
```

たとえば、以下のコマンドを実行します。

```
$ odo catalog describe component nodejs
```

出力例

```
* Registry: DefaultDevfileRegistry 1

Starter Projects: 2
---
name: nodejs-starter
attributes: {}
description: ""
subdir: ""
projectsource:
  sourcetype: ""
  git:
    gitlikeprojectsource:
      commonprojectsource: {}
      checkoutfrom: null
      remotes:
        origin: https://github.com/odo-devfiles/nodejs-ex.git
  zip: null
  custom: null
```

1 **Registry** は、devfile の取得元のレジストリーです。

2 **Starter projects** は、devfile の同じ言語およびフレームワークにあるサンプルプロジェクトです。これは、新規プロジェクトの起動に役立ちます。

スタータープロジェクトからプロジェクトを作成する方法については、**odo create** を参照してください。

3.5.2.2. サービス

odo は **Operator** を利用して サービス をデプロイでき ます。

odo では、Operator Lifecycle Manager を利用してデプロイされた Operator のみがサポートされます。

3.5.2.2.1. サービスの一覧表示

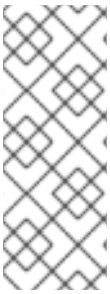
利用可能な Operator およびそれらの関連サービスを一覧表示するには、以下のコマンドを実行します。

```
$ odo catalog list services
```

出力例

```
Services available through Operators
NAME                            CRDs
postgresql-operator.v0.1.1          Backup, Database
redis-operator.v0.8.0               RedisCluster, Redis
```

この例では、2 つの Operator がクラスターにインストールされます。**postgresql-operator.v0.1.1** Operator は、PostgreSQL に関連するサービス: **Backup** と **Database** をデプロイします。**redis-operator.v0.8.0** Operator は、**RedisCluster** および **Redis** に関連するサービスをデプロイします。



注記

利用可能な Operator の一覧を取得するには、**odo** は **Succeeded** フェーズにある現在の namespace の ClusterServiceVersion (CSV) リソースを取得します。クラスター全体のアクセスをサポートする Operator の場合、新規 namespace が作成されると、これらのリソースがこれに自動的に追加されます。ただし、**Succeeded** フェーズに入るまでに時間がかかる場合がありますが、**odo** はリソースが準備状態になるまで空の一覧を返す可能性があります。

3.5.2.2.2. サービスの検索

キーワードで特定のサービスを検索するには、以下のコマンドを実行します。

```
$ odo catalog search service
```

たとえば、PostgreSQL サービスを取得するには、以下のコマンドを実行します。

```
$ odo catalog search service postgres
```

出力例

```
Services available through Operators
NAME                          CRDs
postgresql-operator.v0.1.1    Backup, Database
```

検索されたキーワードを名前に含む Operator の一覧が表示されます。

3.5.2.2.3. サービスに関する情報の取得

特定のサービスに関する詳細情報を取得するには、以下のコマンドを実行します。

```
$ odo catalog describe service
```

以下に例を示します。

```
$ odo catalog describe service postgresql-operator.v0.1.1/Database
```

出力例

```
KIND:    Database
VERSION: v1alpha1

DESCRIPTION:
    Database is the Schema for the the Database Database API

FIELDS:
  awsAccessKeyId (string)
```

```
    AWS S3 accessKey/token ID

    Key ID of AWS S3 storage. Default Value: nil Required to create the Secret
    with the data to allow send the backup files to AWS S3 storage.
[...]
```

サービスは、CustomResourceDefinition (CRD) リソースによってクラスターに表示されます。前のコマンドは、**kind**、**version**、このカスタムリソースのインスタンスを定義するために使用できるフィールドのリストなど、CRD に関する詳細を表示します。

フィールドの一覧は、CRD に含まれる **OpenAPI** スキーマ から抽出されます。この情報は CRD でオプションであり、存在しない場合は、サービスを表す ClusterServiceVersion (CSV) リソースから抽出されます。

CRD タイプの情報を指定せずに、Operator がサポートするサービスの説明を要求することもできます。CRD のないクラスターで Redis Operator を記述するには、以下のコマンドを実行します。

```
$ odo catalog describe service redis-operator.v0.8.0
```

出力例

```
NAME: redis-operator.v0.8.0
DESCRIPTION:

 A Golang based redis operator that will make/oversee Redis
 standalone/cluster mode setup on top of the Kubernetes. It can create a
 redis cluster setup with best practices on Cloud as well as the Bare metal
 environment. Also, it provides an in-built monitoring capability using

... (cut short for beverity)

 Logging Operator is licensed under [Apache License, Version
 2.0](https://github.com/OT-CONTAINER-KIT/redis-operator/blob/master/LICENSE)


CRDs:
 NAME          DESCRIPTION
 RedisCluster  Redis Cluster
 Redis         Redis
```

3.5.3. odo create

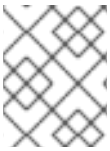
odo は devfile を使用してコンポーネントの設定を保存し、ストレージやサービスなどのコンポーネントのリソースを記述します。**odo create** コマンドはこのファイルを生成します。

3.5.3.1. コンポーネントの作成

既存のプロジェクトの **devfile** を 作成 するには、コンポーネントの名前とタイプ (たとえば、**nodejs** または **go**) を指定して **odo create** コマンドを実行します。

```
odo create nodejs mynodejs
```

この例では、**nodejs** はコンポーネントのタイプで、**mynodejs** は **odo** が作成するコンポーネントの名前です。



注記

サポートされるすべてのコンポーネントタイプの一覧については、コマンド **odo catalog list components** を実行します。

ソースコードが現在のディレクトリーに存在する場合は、**--context** フラグを使用してパスを指定できます。たとえば、nodejs コンポーネントのソースが現在の作業ディレクトリーと相対的に **node-backend** というフォルダーにある場合は、以下のコマンドを実行します。

```
odo create nodejs mynodejs --context ./node-backend
```

--context フラグは、相対パスおよび絶対パスをサポートします。

コンポーネントがデプロイされるプロジェクトまたはアプリケーションを指定するには、**--project** フラグおよび **--app** フラグを使用します。たとえば、**backend** プロジェクト内の **myapp** アプリの一部であるコンポーネントを作成するには、次のコマンドを実行します。

```
odo create nodejs --app myapp --project backend
```

注記

これらのフラグが指定されていない場合、デフォルトはアクティブなアプリケーションおよびプロジェクトに設定されます。

3.5.3.2. スタータープロジェクト

既存のソースコードがなく、devfile およびコンポーネントを迅速に稼働させる必要がある場合は、スタータープロジェクトを使用します。スタータープロジェクトを使用するには、**--starter** フラグを **odo create** コマンドに追加します。

コンポーネントタイプの利用可能なスタータープロジェクトの一覧を表示するには、**odo catalog describe component** コマンドを実行します。たとえば、nodejs コンポーネントタイプの利用可能なスタータープロジェクトをすべて取得するには、以下のコマンドを実行します。

```
odo catalog describe component nodejs
```

次に、**odo create** コマンドで **--starter** フラグを使用して必要なプロジェクトを指定します。

```
odo create nodejs --starter nodejs-starter
```

これにより、選択したコンポーネントタイプ (この例では **nodejs**) に対応するサンプルテンプレートがダウンロードされます。テンプレートは、現在のディレクトリーまたは **--context** フラグで指定された場所にダウンロードされます。スタータープロジェクトに独自の devfile がある場合、この devfile は保持されます。

3.5.3.3. 既存の devfile の使用

既存の devfile から新規コンポーネントを作成する場合は、**--devfile** フラグを使用して devfile へのパスを指定して実行できます。たとえば、GitHub の devfile に基づいて **mynodejs** というコンポーネントを作成するには、以下のコマンドを使用します。

```
odo create mynodejs --devfile https://raw.githubusercontent.com/odo-devfiles/registry/master/devfiles/nodejs/devfile.yaml
```

3.5.3.4. インタラクティブな作成

odo create コマンドを対話的に実行して、コンポーネントの作成に必要な手順をガイドすることもできます。

```
$ odo create

? Which devfile component type do you wish to create go
? What do you wish to name the new devfile component go-api
? What project do you want the devfile component to be created in default
Devfile Object Validation
 ✓  Checking devfile existence [164258ns]
 ✓  Creating a devfile component from registry: DefaultDevfileRegistry [246051ns]
Validation
 ✓  Validating if devfile name is correct [92255ns]
? Do you want to download a starter project Yes

Starter Project
 ✓  Downloading starter project go-starter from https://github.com/devfile-samples/devfile-stack-go.git [429ms]

Please use odo push command to create the component with source deployed
```

コンポーネントのコンポーネントタイプ、名前、およびプロジェクトを選択します。スタータープロジェクトをダウンロードするかどうかを選択することもできます。完了したら、新しい **devfile.yaml** ファイルが作業ディレクトリーに作成されます。

これらのリソースをクラスターにデプロイするには、**odo push** コマンドを実行します。

3.5.4. odo delete

odo delete コマンドは、**odo** によって管理されるリソースを削除するのに役立ちます。

3.5.4.1. コンポーネントの削除

devfile コンポーネントを削除するには、**odo delete** コマンドを実行します。

```
$ odo delete
```

コンポーネントがクラスターにプッシュされている場合、コンポーネントは依存するストレージ、URL、シークレット、他のリソースと共にクラスターから削除されます。コンポーネントがプッシュされていない場合、コマンドはクラスターのリソースが検出できなかったことを示すエラーを出して終了します。

確認質問を回避するには、**-f** フラグまたは **--force** フラグを使用します。

3.5.4.2. devfile Kubernetes コンポーネントのアンデプロイ

odo deploy でデプロイされた devfile Kubernetes コンポーネントをアンデプロイするには、 **--deploy** フラグを指定して **odo delete** コマンドを実行します。

```
$ odo delete --deploy
```

確認質問を回避するには、**-f** フラグまたは **--force** フラグを使用します。

3.5.4.3. すべて削除

以下の項目を含むすべてのアーティファクトを削除するには、**--all** フラグを指定して **odo delete** コマンドを実行します。

- **devfile** コンポーネント
- **odo deploy** コマンドを使用してデプロイされた devfile Kubernetes コンポーネント
- devfile
- ローカル設定

```
$ odo delete --all
```

3.5.4.4. 利用可能なフラグ

-f, --force

このフラグを使用して確認質問を回避します。

-w, --wait

このフラグを使用して、コンポーネントおよび依存関係が削除されるのを待機します。このフラグは、アンデプロイ時には機能しません。

Common Flags フラグに関するドキュメントでは、コマンドで利用可能なフラグの詳細情報が提供されています。

3.5.5. odo deploy

odo を使用すると、CI/CD システムを使用してコンポーネントをデプロイする方法と同様に、コンポーネントをデプロイできます。まず、**odo** はコンテナーイメージをビルドしてから、コンポーネントのデプロイに必要な Kubernetes リソースをデプロイします。

コマンド **odo deploy** を実行すると、**odo** は devfile で kind **deploy** のデフォルトコマンドを検索し、以下のコマンドを実行します。このタイプの **deploy** は、バージョン 2.2.0 以降の devfile 形式でサポートされます。

deploy コマンドは通常、いくつかの 適用 コマンドで設定される 複合 コマンドです。

- 適用されると、デプロイするコンテナーのイメージを構築し、それをレジストリーにプッシュする **image** コンポーネントを参照するコマンド。
- Kubernetes コンポーネント を参照するコマンドは、適用されるとクラスターに Kubernetes リソースを作成します。

以下の **devfile.yaml** ファイルのサンプルでは、コンテナーイメージはディレクトリーにある **Dockerfile** を使用してビルドされます。イメージはレジストリーにプッシュされ、この新規にビルドされたイメージを使用して Kubernetes Deployment リソースがクラスターに作成されます。

```
schemaVersion: 2.2.0
[...]
variables:
  CONTAINER_IMAGE: quay.io/phmartin/myimage
```

```
commands:
  - id: build-image
    apply:
      component: outerloop-build
  - id: deployk8s
    apply:
      component: outerloop-deploy
  - id: deploy
    composite:
      commands:
        - build-image
        - deployk8s
      group:
        kind: deploy
        isDefault: true
components:
  - name: outerloop-build
    image:
      imageName: "{{CONTAINER_IMAGE}}"
      dockerfile:
        uri: ./Dockerfile
        buildContext: ${PROJECTS_ROOT}
  - name: outerloop-deploy
    kubernetes:
      inlined: |
        kind: Deployment
        apiVersion: apps/v1
        metadata:
          name: my-component
        spec:
          replicas: 1
          selector:
            matchLabels:
              app: node-app
          template:
            metadata:
              labels:
                app: node-app
            spec:
              containers:
                - name: main
                  image: {{CONTAINER_IMAGE}}
```

3.5.6. odo link

odo link コマンドは、**odo** コンポーネントを Operator がサポートするサービスまたは別の **odo** コンポーネントにリンクするのに役立ちます。これは Service Binding Operator を使用して行います。現時点で、**odo** は必要な機能を実現するために Operator 自体ではなく、Service Binding ライブラリーを使用します。

3.5.6.1. 各種リンクオプション

odo は、コンポーネントを Operator がサポートするサービスまたは別の **odo** コンポーネントにリンクするための各種のオプションを提供します。これらのオプション (またはフラグ) はすべて、コンポーネントをサービスにリンクする場合でも、別のコンポーネントにリンクする場合でも使用できます。

3.5.6.1.1. デフォルト動作

デフォルトでは、**odo link** コマンドは、コンポーネントディレクトリーに **kubernetes/** という名前のディレクトリーを作成し、そこにサービスとリンクに関する情報 (YAML マニフェスト) を保存します。**odo push** を使用すると、**odo** はこれらのマニフェストを Kubernetes クラスター上のリソースの状態と比較し、ユーザーが指定したものと一致するようにリソースを作成、変更、または破棄する必要があるかどうかを判断します。

3.5.6.1.2. **--inlined** フラグ

odo link コマンドに **--inlined** フラグを指定すると、**odo** は、**kubernetes/** ディレクトリーの下にファイルを作成する代わりに、リンク情報をコンポーネントディレクトリーの **devfile.yaml** にインラインで保存します。**--inlined** フラグの動作は、**odo link** および **odo service create** コマンドの両方で似ています。このフラグは、すべてが単一の **devfile.yaml** に保存されている場合に便利です。コンポーネント用に実行する各 **odo link** および **odo service create** コマンドで **--inlined** フラグを使用するのを覚えておく必要があります。

3.5.6.1.3. **--map** フラグ

場合によっては、デフォルトで利用できる内容に加えて、コンポーネントにバインディング情報をさらに追加する必要がある場合があります。たとえば、コンポーネントをサービスにリンクしていて、サービスの仕様 (仕様の略) からの情報をバインドしたい場合は、**-map** フラグを使用できます。**odo** は、リンクされているサービスまたはコンポーネントの仕様に対して検証を実行しないことに注意してください。このフラグの使用は、Kubernetes YAML マニフェストの使用に慣れる場合にのみ推奨されます。

3.5.6.1.4. **--bind-as-files** フラグ

これまでに説明したすべてのリンクオプションについて、**odo** はバインディング情報を環境変数としてコンポーネントに挿入します。この情報をファイルとしてマウントする場合は、**--bind-as-files** フラグを使用できます。これにより、**odo** はバインディング情報をファイルとしてコンポーネントの Pod 内の **/bindings** の場所に挿入します。環境変数のシナリオと比較して、**-bind-as-files** を使用すると、ファイルはキーにちなんで名前が付けられ、これらのキーの値はこれらのファイルのコンテンツとして保存されます。

3.5.6.2. 例

3.5.6.2.1. デフォルトの **odo link**

以下の例では、バックエンドコンポーネントはデフォルトの **odo link** コマンドを使用して PostgreSQL サービスにリンクされています。バックエンドコンポーネントでは、コンポーネントおよびサービスがクラスターにプッシュされていることを確認します。

```
$ odo list
```

出力例

```
APP     NAME        PROJECT       TYPE       STATE      MANAGED BY ODO
app     backend     myproject     spring     Pushed     Yes
```

```
$ odo service list
```

出力例

```
NAME                    MANAGED BY ODO     STATE      AGE
PostgresCluster/hippo     Yes (backend)      Pushed     59m41s
```

ここで、**odo link** を実行してバックエンドコンポーネントを PostgreSQL サービスにリンクします。

```
$ odo link PostgresCluster/hippo
```

出力例

```
 ✓  Successfully created link between component "backend" and service "PostgresCluster/hippo"

To apply the link, please use `odo push`
```

次に、**odo push** を実行して Kubernetes クラスターにリンクを作成します。

odo push に成功すると、以下のような結果が表示されます。

1. バックエンドコンポーネントによってデプロイされたアプリケーションの URL を開くと、データベース内の **ToDo** アイテムのリストが表示されます。たとえば、**odo url list** コマンドの出力では、**todos** が記載されているパスが含まれます。

   ```
   $ odo url list
   ```

   出力例

   ```
   Found the following URLs for component backend
   NAME        STATE     URL                                PORT     SECURE     KIND
   8080-tcp    Pushed    http://8080-tcp.192.168.39.112.nip.io    8080     false      ingress
   ```

   URL の正しいパスは http://8080-tcp.192.168.39.112.nip.io/api/v1/todos になります。URL は設定によって異なります。また、追加しない限りデータベースには **todo** がないため、URL に空の JSON オブジェクトが表示される場合があることにも注意してください。

2. backend コンポーネントにインジェクトされる Postgres サービスに関連するバインディング情報を確認できます。このバインディング情報は、デフォルトで環境変数として挿入されます。バックエンドコンポーネントのディレクトリーから **odo describe** コマンドを使用してこれを確認できます。

   ```
   $  odo describe
   ```

   出力例:

   ```
   Component Name: backend
   Type: spring
   Environment Variables:
    · PROJECTS_ROOT=/projects
    · PROJECT_SOURCE=/projects
    · DEBUG_PORT=5858
   Storage:
    · m2 of size 3Gi mounted to /home/user/.m2
   URLs:
    · http://8080-tcp.192.168.39.112.nip.io exposed via 8080
   Linked Services:
   ```

```
・PostgresCluster/hippo
  Environment Variables:
  ・POSTGRESCLUSTER_PGBOUNCER-EMPTY
  ・POSTGRESCLUSTER_PGBOUNCER.INI
  ・POSTGRESCLUSTER_ROOT.CRT
  ・POSTGRESCLUSTER_VERIFIER
  ・POSTGRESCLUSTER_ID_ECDSA
  ・POSTGRESCLUSTER_PGBOUNCER-VERIFIER
  ・POSTGRESCLUSTER_TLS.CRT
  ・POSTGRESCLUSTER_PGBOUNCER-URI
  ・POSTGRESCLUSTER_PATRONI.CRT-COMBINED
  ・POSTGRESCLUSTER_USER
  ・pgImage
  ・pgVersion
  ・POSTGRESCLUSTER_CLUSTERIP
  ・POSTGRESCLUSTER_HOST
  ・POSTGRESCLUSTER_PGBACKREST_REPO.CONF
  ・POSTGRESCLUSTER_PGBOUNCER-USERS.TXT
  ・POSTGRESCLUSTER_SSH_CONFIG
  ・POSTGRESCLUSTER_TLS.KEY
  ・POSTGRESCLUSTER_CONFIG-HASH
  ・POSTGRESCLUSTER_PASSWORD
  ・POSTGRESCLUSTER_PATRONI.CA-ROOTS
  ・POSTGRESCLUSTER_DBNAME
  ・POSTGRESCLUSTER_PGBOUNCER-PASSWORD
  ・POSTGRESCLUSTER_SSHD_CONFIG
  ・POSTGRESCLUSTER_PGBOUNCER-FRONTEND.KEY
  ・POSTGRESCLUSTER_PGBACKREST_INSTANCE.CONF
  ・POSTGRESCLUSTER_PGBOUNCER-FRONTEND.CA-ROOTS
  ・POSTGRESCLUSTER_PGBOUNCER-HOST
  ・POSTGRESCLUSTER_PORT
  ・POSTGRESCLUSTER_ROOT.KEY
  ・POSTGRESCLUSTER_SSH_KNOWN_HOSTS
  ・POSTGRESCLUSTER_URI
  ・POSTGRESCLUSTER_PATRONI.YAML
  ・POSTGRESCLUSTER_DNS.CRT
  ・POSTGRESCLUSTER_DNS.KEY
  ・POSTGRESCLUSTER_ID_ECDSA.PUB
  ・POSTGRESCLUSTER_PGBOUNCER-FRONTEND.CRT
  ・POSTGRESCLUSTER_PGBOUNCER-PORT
  ・POSTGRESCLUSTER_CA.CRT
```

これらの変数の一部は、バックエンドコンポーネントの **src/main/resources/application.properties** ファイルで使用されるため、JavaSpringBoot アプリケーションは PostgreSQL データベースサービスに接続できます。

3. 最後に、**odo** はバックエンドコンポーネントのディレクトリーに **kubernetes/** というディレクトリーを作成しました。このディレクトリーには次のファイルが含まれています。

```
$ ls kubernetes
odo-service-backend-postgrescluster-hippo.yaml  odo-service-hippo.yaml
```

これらのファイルには、次の 2 つのリソースの情報 (YAML マニフェスト) が含まれています。

   a. **odo-service-hippo.yaml-odo service create**: **odo service create --from-file ../postgrescluster.yaml** コマンドを使用して作成された Postgres サービス。

b. **odo-service-backend-PostgresCluster-hippo.yaml-odolink**: **odo link** コマンドを使用して作成された リンク。

3.5.6.2.2. --inlined フラグでの odo link の使用

odo link コマンドで **--inlined** フラグを使用すると、これがバインディング情報を挿入するというフラグなしに **odo link** コマンドと同じ効果があります。ただし、上記の場合は、**kubernetes**/ ディレクトリーに 2 つのマニフェストファイルがあります。1 つは Postgres サービス用で、もう 1 つは backend コンポーネントとこのサービス間のリンク用です。ただし、**-inlined** フラグを渡すと、**odo** は **kubernetes**/ ディレクトリーの下に YAML マニフェストを保存するファイルを作成せず、**devfile.yaml** ファイルにインラインで保存します。

これを確認するには、最初に PostgreSQL サービスからコンポーネントをリンク解除します。

```
$ odo unlink PostgresCluster/hippo
```

出力例:

```
 ✓  Successfully unlinked component "backend" from service "PostgresCluster/hippo"

To apply the changes, please use `odo push`
```

クラスターでそれらをリンクするには、**odo push** を実行します。**kubernetes**/ ディレクトリーを検査すると、1 つのファイルのみが表示されます。

```
$ ls kubernetes
odo-service-hippo.yaml
```

次に、**--inlined** フラグを使用してリンクを作成します。

```
$ odo link PostgresCluster/hippo --inlined
```

出力例:

```
 ✓  Successfully created link between component "backend" and service "PostgresCluster/hippo"

To apply the link, please use `odo push`
```

--inlined フラグを省略する手順など、クラスターで作成されるリンクを取得するために **odo push** を実行する必要があります。**odo** は設定を **devfile.yaml** に保存します。このファイルに以下のようなエントリーが表示されます。

```
kubernetes:
  inlined: |
    apiVersion: binding.operators.coreos.com/v1alpha1
    kind: ServiceBinding
    metadata:
      creationTimestamp: null
      name: backend-postgrescluster-hippo
    spec:
      application:
        group: apps
        name: backend-app
```

```
        resource: deployments
        version: v1
      bindAsFiles: false
      detectBindingResources: true
      services:
      - group: postgres-operator.crunchydata.com
        id: hippo
        kind: PostgresCluster
        name: hippo
        version: v1beta1
    status:
      secret: ""
  name: backend-postgrescluster-hippo
```

odo unlink PostgresCluster/hippo を実行する場合に、**odo** はまず **devfile.yaml** からリンク情報を削除し、後続の **odo push** はクラスターからリンクを削除するようになりました。

3.5.6.2.3. カスタムバインディング

odo link は、カスタムバインディング情報をコンポーネントに挿入することのできるフラグ **--map** を受け入れます。このようなバインディング情報は、コンポーネントにリンクしているリソースのマニフェストから取得されます。たとえば、バックエンドコンポーネントおよび PostgreSQL サービスのコンテキストでは、PostgreSQL サービスのマニフェスト **postgrescluster.yaml** ファイルからの情報をバックエンドコンポーネントに注入することができます。

PostgresCluster サービスの名前が **hippo** (または PostgresCluster サービスの名前が異なる場合は **odo service list** の出力) の場合、その YAML 定義から **postgresVersion** の値をバックエンドコンポーネントに挿入するときは、次のコマンドを実行します。

```
$ odo link PostgresCluster/hippo --map pgVersion='{{ .hippo.spec.postgresVersion }}'
```

Postgres サービスの名前が **hippo** と異なる場合は、上記のコマンドで **pgVersion** の値の **.hippo** の代わりにそれを指定する必要があることに注意してください。

リンク操作後に、通常どおり **odo push** を実行します。プッシュ操作が正常に完了すると、バックエンドコンポーネントディレクトリーから次のコマンドを実行して、カスタムマッピングが適切に挿入されたかどうかを検証できます。

```
$ odo exec -- env | grep pgVersion
```

出力例:

```
pgVersion=13
```

カスタムバインディング情報を複数挿入したい可能性があるため、**odo link** は複数のキーと値のペアを受け入れます。唯一の制約は、これらを **--map <key>=<value>** として指定する必要があるということです。たとえば、PostgreSQL イメージ情報をバージョンと共に注入する場合には、以下を実行できます。

```
$ odo link PostgresCluster/hippo --map pgVersion='{{ .hippo.spec.postgresVersion }}' --map
pgImage='{{ .hippo.spec.image }}'
```

次に、**odo push** を実行します。両方のマッピングが正しくインジェクトされたかどうかを確認するには、以下のコマンドを実行します。

```
$ odo exec -- env | grep -e "pgVersion\|pgImage"
```

出力例:

```
pgVersion=13
pgImage=registry.developers.crunchydata.com/crunchydata/crunchy-postgres-ha:centos8-13.4-0
```

3.5.6.2.3.1. インラインかどうか。

odo link が **kubernetes**/ ディレクトリー下のリンクのマニフェストファイルを生成するデフォルトの動作を受け入れます。または、すべてを単一の **devfile.yaml** ファイルに保存する場合は、**-inlined** フラグを使用できます。

3.5.6.3. ファイルとしてのバインド

odo link が提供するもう 1 つの便利なフラグは、**--bind-as-files** です。このフラグが渡されると、バインディング情報は環境変数としてコンポーネントの Pod に挿入されませんが、ファイルシステムとしてマウントされます。

バックエンドコンポーネントと PostgreSQL サービスの間に既存のリンクがないことを確認します。これは、バックエンドコンポーネントのディレクトリーで **odo describe** を実行して、以下のような出力が表示されるかどうかを確認することで実行できます。

```
Linked Services:
・PostgresCluster/hippo
```

以下を使用してコンポーネントからサービスをリンクを解除します。

```
$ odo unlink PostgresCluster/hippo
$ odo push
```

3.5.6.4. --bind-as-files の例

3.5.6.4.1. デフォルトの odo link の使用

デフォルトでは、**odo** はリンク情報を保存するために **kubernetes**/ ディレクトリーの下にマニフェストファイルを作成します。バックエンドコンポーネントおよび PostgreSQL サービスをリンクします。

```
$ odo link PostgresCluster/hippo --bind-as-files
$ odo push
```

odo describe 出力例:

```
$ odo describe

Component Name: backend
Type: spring
Environment Variables:
・PROJECTS_ROOT=/projects
・PROJECT_SOURCE=/projects
・DEBUG_PORT=5858
・SERVICE_BINDING_ROOT=/bindings
```

```
 ・SERVICE_BINDING_ROOT=/bindings
Storage:
 ・m2 of size 3Gi mounted to /home/user/.m2
URLs:
 ・http://8080-tcp.192.168.39.112.nip.io exposed via 8080
Linked Services:
 ・PostgresCluster/hippo
   Files:
    ・/bindings/backend-postgrescluster-hippo/pgbackrest_instance.conf
    ・/bindings/backend-postgrescluster-hippo/user
    ・/bindings/backend-postgrescluster-hippo/ssh_known_hosts
    ・/bindings/backend-postgrescluster-hippo/clusterIP
    ・/bindings/backend-postgrescluster-hippo/password
    ・/bindings/backend-postgrescluster-hippo/patroni.yaml
    ・/bindings/backend-postgrescluster-hippo/pgbouncer-frontend.crt
    ・/bindings/backend-postgrescluster-hippo/pgbouncer-host
    ・/bindings/backend-postgrescluster-hippo/root.key
    ・/bindings/backend-postgrescluster-hippo/pgbouncer-frontend.key
    ・/bindings/backend-postgrescluster-hippo/pgbouncer.ini
    ・/bindings/backend-postgrescluster-hippo/uri
    ・/bindings/backend-postgrescluster-hippo/config-hash
    ・/bindings/backend-postgrescluster-hippo/pgbouncer-empty
    ・/bindings/backend-postgrescluster-hippo/port
    ・/bindings/backend-postgrescluster-hippo/dns.crt
    ・/bindings/backend-postgrescluster-hippo/pgbouncer-uri
    ・/bindings/backend-postgrescluster-hippo/root.crt
    ・/bindings/backend-postgrescluster-hippo/ssh_config
    ・/bindings/backend-postgrescluster-hippo/dns.key
    ・/bindings/backend-postgrescluster-hippo/host
    ・/bindings/backend-postgrescluster-hippo/patroni.crt-combined
    ・/bindings/backend-postgrescluster-hippo/pgbouncer-frontend.ca-roots
    ・/bindings/backend-postgrescluster-hippo/tls.key
    ・/bindings/backend-postgrescluster-hippo/verifier
    ・/bindings/backend-postgrescluster-hippo/ca.crt
    ・/bindings/backend-postgrescluster-hippo/dbname
    ・/bindings/backend-postgrescluster-hippo/patroni.ca-roots
    ・/bindings/backend-postgrescluster-hippo/pgbackrest_repo.conf
    ・/bindings/backend-postgrescluster-hippo/pgbouncer-port
    ・/bindings/backend-postgrescluster-hippo/pgbouncer-verifier
    ・/bindings/backend-postgrescluster-hippo/id_ecdsa
    ・/bindings/backend-postgrescluster-hippo/id_ecdsa.pub
    ・/bindings/backend-postgrescluster-hippo/pgbouncer-password
    ・/bindings/backend-postgrescluster-hippo/pgbouncer-users.txt
    ・/bindings/backend-postgrescluster-hippo/sshd_config
    ・/bindings/backend-postgrescluster-hippo/tls.crt
```

以前の **odo describe** 出力で **key=value** 形式の環境変数であったものはすべて、ファイルとしてマウントされるようになりました。**cat** コマンドを使用して、これらのファイルの一部を表示します。

コマンドの例:

```
$ odo exec -- cat /bindings/backend-postgrescluster-hippo/password
```

出力例:

```
q({JC:jn^mm/Bw}eu+j.GX{k
```

コマンドの例:

```
$ odo exec -- cat /bindings/backend-postgrescluster-hippo/user
```

出力例:

```
hippo
```

コマンドの例:

```
$ odo exec -- cat /bindings/backend-postgrescluster-hippo/clusterIP
```

出力例:

```
10.101.78.56
```

3.5.6.4.2. **--inlined** の使用

--bind-as-files と **--inlined** を一緒に使用した結果は、**odolink--inlined** を使用した場合と同様です。リンクのマニフェストは、**kubernetes**/ ディレクトリーの別のファイルに保存されるのではなく、**devfile.yaml** に保存されます。これ以外に、**odo describe** 出力は以前と同じになります。

3.5.6.4.3. カスタムバインディング

バックエンドコンポーネントを PostgreSQL サービスにリンクしているときにカスタムバインディングを渡すと、これらのカスタムバインディングは環境変数としてではなく、ファイルとしてマウントされます。以下に例を示します。

```
$ odo link PostgresCluster/hippo --map pgVersion='{{ .hippo.spec.postgresVersion }}' --map
pgImage='{{ .hippo.spec.image }}' --bind-as-files
$ odo push
```

これらのカスタムバインディングは、環境変数として挿入されるのではなく、ファイルとしてマウントされます。これが機能することを確認するには、以下のコマンドを実行します。

コマンドの例:

```
$ odo exec -- cat /bindings/backend-postgrescluster-hippo/pgVersion
```

出力例:

```
13
```

コマンドの例:

```
$ odo exec -- cat /bindings/backend-postgrescluster-hippo/pgImage
```

出力例:

```
registry.developers.crunchydata.com/crunchydata/crunchy-postgres-ha:centos8-13.4-0
```

3.5.7. odo registry

odo は移植可能な **devfile** 形式を使用してコンポーネントを記述します。**odo** は各種の devfile レジストリーに接続して、さまざまな言語およびフレームワークの devfile をダウンロードできます。

公開されている利用可能な devfile レジストリーに接続するか、または独自の **Secure Registry** をインストールできます。

odo registry コマンドを使用して、**odo** によって使用されるレジストリーを管理し、devfile 情報を取得できます。

3.5.7.1. レジストリーの一覧表示

odo で現在接続しているレジストリーを一覧表示するには、以下のコマンドを実行します。

```
$ odo registry list
```

出力例:

```
NAME                    URL                         SECURE
DefaultDevfileRegistry  https://registry.devfile.io  No
```

DefaultDevfileRegistry は odo によって使用されるデフォルトレジストリーです。これは devfile.io プロジェクトによって提供されます。

3.5.7.2. レジストリーの追加

レジストリーを追加するには、以下のコマンドを実行します。

```
$ odo registry add
```

出力例:

```
$ odo registry add StageRegistry https://registry.stage.devfile.io
New registry successfully added
```

独自の Secure Registry をデプロイしている場合、**--token** フラグを使用してセキュアなレジストリーに対して認証するためにパーソナルアクセストークンを指定できます。

```
$ odo registry add MyRegistry https://myregistry.example.com --token <access_token>
New registry successfully added
```

3.5.7.3. レジストリーの削除

レジストリーを削除するには、以下のコマンドを実行します。

```
$ odo registry delete
```

出力例:

```
$ odo registry delete StageRegistry
? Are you sure you want to delete registry "StageRegistry" Yes
Successfully deleted registry
```

--force (または **-f**) フラグを使用して、確認なしでレジストリーを強制的に削除します。

3.5.7.4. レジストリーの更新

すでに登録されているレジストリーの URL またはパーソナルアクセストークンを更新するには、以下のコマンドを実行します。

```
$ odo registry update
```

出力例:

```
$ odo registry update MyRegistry https://otherregistry.example.com --token <other_access_token>
? Are you sure you want to update registry "MyRegistry" Yes
Successfully updated registry
```

--force (または **-f**) フラグを使用して、確認なしでレジストリーの更新を強制します。

3.5.8. odo service

odo は Operator を利用して サービス をデプロイでき ます。

インストールに使用できるオペレーターとサービスのリストは、**odo catalog** コマンドを使用して見つけることができます。

サービスは コンポーネント のコンテキストで作成されるため、サービスをデプロイする前に **odo create** コマンドを実行してください。

サービスは、以下の 2 つのステップに従ってデプロイされます。

1. サービスを定義し、その定義を devfile に保存します。
2. **odo push** コマンドを使用して、定義されたサービスをクラスターにデプロイします。

3.5.8.1. 新しいサービスの作成

新規サービスを作成するには、以下のコマンドを実行します。

```
$ odo service create
```

たとえば、**my-redis-service** という名前の Redis サービスのインスタンスを作成するには、以下のコマンドを実行します。

出力例

```
$ odo catalog list services
Services available through Operators
NAME                    CRDs
redis-operator.v0.8.0   RedisCluster, Redis
```

```
$ odo service create redis-operator.v0.8.0/Redis my-redis-service
Successfully added service to the configuration; do 'odo push' to create service on the cluster
```

このコマンドは、サービスの定義を含む Kubernetes マニフェストを **kubernetes/** ディレクトリーに作成し、このファイルは **devfile.yaml** ファイルから参照されます。

```
$ cat kubernetes/odo-service-my-redis-service.yaml
```

出力例

```
apiVersion: redis.redis.opstreelabs.in/v1beta1
kind: Redis
metadata:
  name: my-redis-service
spec:
  kubernetesConfig:
    image: quay.io/opstree/redis:v6.2.5
    imagePullPolicy: IfNotPresent
    resources:
      limits:
        cpu: 101m
        memory: 128Mi
      requests:
        cpu: 101m
        memory: 128Mi
    serviceType: ClusterIP
  redisExporter:
    enabled: false
    image: quay.io/opstree/redis-exporter:1.0
  storage:
    volumeClaimTemplate:
      spec:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 1Gi
```

コマンドの例

```
$ cat devfile.yaml
```

出力例

```
[...]
components:
- kubernetes:
    uri: kubernetes/odo-service-my-redis-service.yaml
  name: my-redis-service
[...]
```

作成されるインスタンスの名前はオプションです。名前を指定しない場合は、サービスの小文字の名前

作成されたインスタンスの名前はオプションです。名前を指定しない場合は、サービスの小文字の名前です。たとえば、以下のコマンドは **redis** という名前の Redis サービスのインスタンスを作成します。

```
$ odo service create redis-operator.v0.8.0/Redis
```

3.5.8.1.1. マニフェストのインライン化

デフォルトで、新規マニフェストは **kubernetes/** ディレクトリーに作成され、**devfile.yaml** ファイルから参照されます。**--inlined** フラグを使用して、**devfile.yaml** ファイル内でマニフェストをインラインにすることができます。

```
$ odo service create redis-operator.v0.8.0/Redis my-redis-service --inlined
Successfully added service to the configuration; do 'odo push' to create service on the cluster
```

コマンドの例

```
$ cat devfile.yaml
```

出力例

```
[...]
components:
- kubernetes:
    inlined: |
      apiVersion: redis.redis.opstreelabs.in/v1beta1
      kind: Redis
      metadata:
        name: my-redis-service
      spec:
        kubernetesConfig:
          image: quay.io/opstree/redis:v6.2.5
          imagePullPolicy: IfNotPresent
          resources:
            limits:
              cpu: 101m
              memory: 128Mi
            requests:
              cpu: 101m
              memory: 128Mi
          serviceType: ClusterIP
        redisExporter:
          enabled: false
          image: quay.io/opstree/redis-exporter:1.0
        storage:
          volumeClaimTemplate:
            spec:
              accessModes:
              - ReadWriteOnce
              resources:
                requests:
                  storage: 1Gi
  name: my-redis-service
[...]
```

3.5.8.1.2. サービスの設定

特定のカスタマイズを行わないと、サービスはデフォルト設定で作成されます。コマンドライン引数またはファイルのいずれかを使用して、独自の設定を指定できます。

3.5.8.1.2.1. コマンドライン引数の使用

--parameters (または **-p**) フラグを使用して、独自の設定を指定します。

以下の例では、Redis サービスを 3 つのパラメーターで設定します。

```
$ odo service create redis-operator.v0.8.0/Redis my-redis-service \
    -p kubernetesConfig.image=quay.io/opstree/redis:v6.2.5 \
    -p kubernetesConfig.serviceType=ClusterIP \
    -p redisExporter.image=quay.io/opstree/redis-exporter:1.0
Successfully added service to the configuration; do 'odo push' to create service on the cluster
```

コマンドの例

```
$ cat kubernetes/odo-service-my-redis-service.yaml
```

出力例

```
apiVersion: redis.redis.opstreelabs.in/v1beta1
kind: Redis
metadata:
  name: my-redis-service
spec:
  kubernetesConfig:
    image: quay.io/opstree/redis:v6.2.5
    serviceType: ClusterIP
  redisExporter:
    image: quay.io/opstree/redis-exporter:1.0
```

odo catalog describe service コマンドを使用して、特定のサービスの使用可能なパラメーターを取得できます。

3.5.8.1.2.2. ファイルの使用

YAML マニフェストを使用して独自の仕様を設定します。以下の例では、Redis サービスは 3 つのパラメーターで設定されます。

1. マニフェストを作成します。

```
$ cat > my-redis.yaml <<EOF
apiVersion: redis.redis.opstreelabs.in/v1beta1
kind: Redis
metadata:
  name: my-redis-service
spec:
  kubernetesConfig:
    image: quay.io/opstree/redis:v6.2.5
    serviceType: ClusterIP
```

```
  redisExporter:
    image: quay.io/opstree/redis-exporter:1.0
EOF
```

2. マニフェストからサービスを作成します。

```
$ odo service create --from-file my-redis.yaml
Successfully added service to the configuration; do 'odo push' to create service on the cluster
```

3.5.8.2. サービスの削除

サービスを削除するには、以下のコマンドを実行します。

```
$ odo service delete
```

出力例

```
$ odo service list
NAME                      MANAGED BY ODO     STATE              AGE
Redis/my-redis-service     Yes (api)          Deleted locally     5m39s
```

```
$ odo service delete Redis/my-redis-service
? Are you sure you want to delete Redis/my-redis-service Yes
Service "Redis/my-redis-service" has been successfully deleted; do 'odo push' to delete service from
the cluster
```

--force (または **-f**) フラグを使用して、確認なしでサービスを強制的に削除します。

3.5.8.3. サービスの一覧表示

コンポーネント用に作成されたサービスを一覧表示するには、以下のコマンドを実行します。

```
$ odo service list
```

出力例

```
$ odo service list
NAME                      MANAGED BY ODO     STATE            AGE
Redis/my-redis-service-1   Yes (api)          Not pushed
Redis/my-redis-service-2   Yes (api)          Pushed           52s
Redis/my-redis-service-3   Yes (api)          Deleted locally   1m22s
```

サービスごとに、**STATE** は、サービスが **odo push** コマンドを使用してクラスターにプッシュされているか、またはサービスがクラスターで実行中であるが、**odo service delete** コマンドを使用してローカルで devfile から削除されるかどうかを示します。

3.5.8.4. サービスに関する情報の取得

設定したパラメーターの種類、バージョン、名前、および一覧などのサービスの詳細を取得するには、以下のコマンドを実行します。

```
$ odo service describe
```

出力例

```
$ odo service describe Redis/my-redis-service
Version: redis.redis.opstreelabs.in/v1beta1
Kind: Redis
Name: my-redis-service
Parameters:
NAME                        VALUE
kubernetesConfig.image      quay.io/opstree/redis:v6.2.5
kubernetesConfig.serviceType    ClusterIP
redisExporter.image         quay.io/opstree/redis-exporter:1.0
```

3.5.9. odo ストレージ

odo を使用すると、ユーザーはコンポーネントに割り当てられるストレージボリュームを管理できます。ストレージボリュームは、**emptyDir** Kubernetes ボリュームを使用するエフェメラルボリューム、または 永続ボリュームクレーム (PVC) のいずれかです。PVC を使用すると、ユーザーは特定のクラウド環境の詳細を理解していなくても、永続ボリューム (GCE PersistentDisk や iSCSI ボリュームなど) を要求できます。永続ストレージボリュームは、再起動時にデータを永続化し、コンポーネントの再ビルドに使用できます。

3.5.9.1. ストレージボリュームの追加

ストレージボリュームをクラスターに追加するには、以下のコマンドを実行します。

```
$ odo storage create
```

出力例:

```
$ odo storage create store --path /data --size 1Gi
✓  Added storage store to nodejs-project-ufyy

$ odo storage create tempdir --path /tmp --size 2Gi --ephemeral
✓  Added storage tempdir to nodejs-project-ufyy

Please use `odo push` command to make the storage accessible to the component
```

上記の例では、最初のストレージボリュームが **/data** パスにマウントされており、サイズは **1Gi** で、2 番目のボリュームが **/tmp** にマウントされ、一時的です。

3.5.9.2. ストレージボリュームの一覧表示

コンポーネントで現在使用されているストレージボリュームを確認するには、以下のコマンドを実行します。

```
$ odo storage list
```

出力例:

```
$ odo storage list
```

```
The component 'nodejs-project-ufyy' has the following storage attached:
NAME      SIZE     PATH      STATE
store     1Gi      /data     Not Pushed
tempdir   2Gi      /tmp      Not Pushed
```

3.5.9.3. ストレージボリュームの削除

ストレージボリュームを削除するには、以下のコマンドを実行します。

```
$ odo storage delete
```

出力例:

```
$ odo storage delete store -f
Deleted storage store from nodejs-project-ufyy

Please use `odo push` command to delete the storage from the cluster
```

上記の例では、**-f** フラグを使用すると、ユーザーパーミッションを要求せずにストレージを強制的に削除します。

3.5.9.4. 特定のコンテナーへのストレージの追加

devfile に複数のコンテナーがある場合、**odo storage create** コマンドで **--container** フラグを使用して、ストレージを割り当てるコンテナーを指定できます。

以下の例は、複数のコンテナーを持つ devfile の抜粋です。

```
components:
  - name: nodejs1
    container:
      image: registry.access.redhat.com/ubi8/nodejs-12:1-36
      memoryLimit: 1024Mi
      endpoints:
        - name: "3000-tcp"
          targetPort: 3000
      mountSources: true
  - name: nodejs2
    container:
      image: registry.access.redhat.com/ubi8/nodejs-12:1-36
      memoryLimit: 1024Mi
```

この例では、**nodejs1** と **nodejs2** の 2 つのコンテナーがあります。ストレージを **nodejs2** コンテナーに割り当てるには、以下のコマンドを使用します。

```
$ odo storage create --container
```

出力例:

```
$ odo storage create store --path /data --size 1Gi --container nodejs2
✓  Added storage store to nodejs-testing-xnfg

Please use `odo push` command to make the storage accessible to the component
```

–

odo storage list コマンドを使用して、ストレージリソースを一覧表示できます。

```
$ odo storage list
```

出力例:

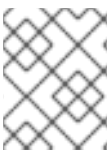
```
The component 'nodejs-testing-xnfg' has the following storage attached:
NAME      SIZE     PATH      CONTAINER     STATE
store     1Gi      /data     nodejs2       Not Pushed
```

3.5.10. 共通フラグ

以下のフラグは、ほとんどの **odo** コマンドで利用できます。

表3.1 odo フラグ

コマンド	説明
--context	コンポーネントを定義するコンテキストディレクトリーを設定します。
--project	コンポーネントのプロジェクトを設定します。デフォルトは、ローカル設定で定義されたプロジェクトです。利用できる場合は、クラスターの現在のプロジェクトです。
--app	コンポーネントのアプリケーションを設定します。デフォルトは、ローカル設定で定義されたアプリケーションです。存在しない場合は、app にします。
--kubeconfig	デフォルト設定を使用していない場合は、パスを **kubeconfig** 値に設定します。
--show-log	このフラグを使用してログを表示します。
-f, **--force**	このフラグを使用して、コマンドに対して確認を求めるプロンプトを出さないように指示します。
-v, **--v**	詳細レベルを設定します。詳細は、odo でのロギング について参照してください。
-h, **--help**	コマンドのヘルプを出力します。



注記

一部のコマンドでフラグを使用できない場合があります。**--help** フラグを指定してコマンドを実行して、利用可能なすべてのフラグの一覧を取得します。

3.5.11. JSON 出力

コンテンツを出力する **odo** コマンドは、通常、**-o json** フラグを受け入れて、このコンテンツを JSON 形式で出力します。これは、他のプログラムがこの出力をより簡単に解析するのに適しています。

出力構造は Kubernetes リソースに似ており、**kind**、**apiVersion**、**metadata**、**spec**、および **status** フィールドがあります。

リスト コマンドは、リストのアイテムを一覧表示する **items** (または同様の) フィールドを含む **List** リソースを返します。各アイテムも Kubernetes リソースに類似しています。

delete コマンドは **Status** リソースを返します。ステータス Kubernetes リソース を参照してください。

他のコマンドは、**Application**、**Storage**、**URL** などのコマンドに関連付けられたリソースを返します。

現在 **-o json** フラグを許可するコマンドの全一覧は以下のとおりです。

コマンド	種類 (バージョン)	リストアイテムの種類 (バージョン)	完全なコンテンツかどうか
odo application describe	Application (odo.dev/v1alpha1)	**該当なし**	いいえ
odo application list	List (odo.dev/v1alpha1)	Application (odo.dev/v1alpha1)	?
odo catalog list components	List (odo.dev/v1alpha1)	**missing**	はい
odo catalog list services	List (odo.dev/v1alpha1)	ClusterServiceVersion (operators.coreos.com/v1alpha1)	?
odo catalog describe component	**missing**	**該当なし**	はい
odo catalog describe service	CRDDescription (odo.dev/v1alpha1)	**該当なし**	はい
odo component create	Component (odo.dev/v1alpha1)	**該当なし**	はい
odo component describe	Component (odo.dev/v1alpha1)	**該当なし**	はい
odo component list	List (odo.dev/v1alpha1)	Component (odo.dev/v1alpha1)	はい
odo config view	DevfileConfiguration (odo.dev/v1alpha1)	**該当なし**	はい

コマンド	種類 (バージョン)	リストアイテムの種類 (バージョン)	完全なコンテンツかどうか
odo debug info	OdoDebugInfo (odo.dev/v1alpha1)	**該当なし**	はい
odo env view	EnvInfo (odo.dev/v1alpha1)	**該当なし**	はい
odo preference view	PreferenceList (odo.dev/v1alpha1)	**該当なし**	はい
odo project create	Project (odo.dev/v1alpha1)	**該当なし**	はい
odo project delete	Status (v1)	**該当なし**	はい
odo project get	Project (odo.dev/v1alpha1)	**該当なし**	はい
odo project list	List (odo.dev/v1alpha1)	Project (odo.dev/v1alpha1)	はい
odo registry list	List (odo.dev/v1alpha1)	**missing**	はい
odo service create	サービス	**該当なし**	はい
odo service describe	サービス	**該当なし**	はい
odo service list	List (odo.dev/v1alpha1)	サービス	はい
odo storage create	Storage (odo.dev/v1alpha1)	**該当なし**	はい
odo storage delete	Status (v1)	**該当なし**	はい
odo storage list	List (odo.dev/v1alpha1)	Storage (odo.dev/v1alpha1)	はい
odo url list	List (odo.dev/v1alpha1)	URL (odo.dev/v1alpha1)	はい

第4章 HELM CLI

4.1. HELM 3 のスタートガイド

4.1.1. Helm について

Helm は、アプリケーションやサービスの OpenShift Container Platform クラスターへのデプロイメントを単純化するソフトウェアパッケージマネージャーです。

Helm は **charts** というパッケージ形式を使用します。Helm チャートは、OpenShift Container Platform リソースを記述するファイルのコレクションです。

クラスター内のチャートの実行中のインスタンスは、リリース と呼ばれます。チャートがクラスターにインストールされているたびに、新規のリリースが作成されます。

チャートのインストール時、またはリリースがアップグレードまたはロールバックされるたびに、増分リビジョンが作成されます。

4.1.1.1. 主な特長

Helm は以下を行う機能を提供します。

- チャートリポジトリーに保存したチャートの大規模なコレクションの検索。
- 既存のチャートの変更。
- OpenShift Container Platform または Kubernetes リソースの使用による独自のチャートの作成。
- アプリケーションのチャートとしてのパッケージ化および共有。

4.1.2. Helm のインストール

以下のセクションでは、CLI を使用して各種の異なるプラットフォームに Helm をインストールする方法を説明します。

また、OpenShift Container Platform Web コンソールから最新のバイナリーへの URL を見つけるには、右上隅の ? アイコンをクリックし、**Command Line Tools** を選択します。

前提条件

- Go バージョン 1.13 以降がインストールされている。

4.1.2.1. Linux の場合

1. Helm バイナリーをダウンロードし、これをパスに追加します。

```
# curl -L https://mirror.openshift.com/pub/openshift-v4/clients/helm/latest/helm-linux-amd64 -o /usr/local/bin/helm
```

2. バイナリーファイルを実行可能にします。

```
# chmod +x /usr/local/bin/helm
```

3. インストールされたバージョンを確認します。

```
$ helm version
```

出力例

```
version.BuildInfo{Version:"v3.0",
GitCommit:"b31719aab7963acf4887a1c1e6d5e53378e34d93", GitTreeState:"clean",
GoVersion:"go1.13.4"}
```

4.1.2.2. Windows 7/8 の場合

1. 最新の **.exe** ファイル をダウンロードし、希望のディレクトリーに配置します。
2. **Start** を右クリックし、**Control Panel** をクリックします。
3. **System and Security** を選択してから **System** をクリックします。
4. 左側のメニューから、**Advanced systems settings** を選択し、下部にある **Environment Variables** をクリックします。
5. **Variable** セクションから **Path** を選択し、**Edit** をクリックします。
6. **New** をクリックして、 **.exe** ファイルのあるフォルダーへのパスをフィールドに入力するか、または **Browse** をクリックし、ディレクトリーを選択して **OK** をクリックします。

4.1.2.3. Windows 10 の場合

1. 最新の **.exe** ファイル をダウンロードし、希望のディレクトリーに配置します。
2. **Search** クリックして、**env** または **environment** を入力します。
3. **Edit environment variables for your account**を選択します。
4. **Variable** セクションから **Path** を選択し、**Edit** をクリックします。
5. **New** をクリックし、exe ファイルのあるディレクトリーへのパスをフィールドに入力するか、または **Browse** をクリックし、ディレクトリーを選択して **OK** をクリックします。

4.1.2.4. MacOS の場合

1. Helm バイナリーをダウンロードし、これをパスに追加します。

```
# curl -L https://mirror.openshift.com/pub/openshift-v4/clients/helm/latest/helm-darwin-amd64
-o /usr/local/bin/helm
```

2. バイナリーファイルを実行可能にします。

```
# chmod +x /usr/local/bin/helm
```

3. インストールされたバージョンを確認します。

```
$ helm version
```

出力例

```
version.BuildInfo{Version:"v3.0",
GitCommit:"b31719aab7963acf4887a1c1e6d5e53378e34d93", GitTreeState:"clean",
GoVersion:"go1.13.4"}
```

4.1.3. OpenShift Container Platform クラスターでの Helm チャートのインストール

前提条件

- 実行中の OpenShift Container Platform クラスターがあり、ログインしている。
- Helm がインストールされている。

手順

1. 新規プロジェクトを作成します。

```
$ oc new-project mysql
```

2. Helm チャートのリポジトリーをローカルの Helm クライアントに追加します。

```
$ helm repo add stable https://kubernetes-charts.storage.googleapis.com/
```

出力例

```
"stable" has been added to your repositories
```

3. リポジトリーを更新します。

```
$ helm repo update
```

4. MySQL チャートのサンプルをインストールします。

```
$ helm install example-mysql stable/mysql
```

5. チャートが正常にインストールされたことを確認します。

```
$ helm list
```

出力例

```
NAME NAMESPACE REVISION UPDATED STATUS CHART APP VERSION
example-mysql mysql 1 2019-12-05 15:06:51.379134163 -0500 EST deployed mysql-1.5.0
5.7.27
```

4.1.4. OpenShift Container Platform でのカスタム Helm チャートの作成

手順

1. 新規プロジェクトを作成します。

```
$ oc new-project nodejs-ex-k
```

2. OpenShift Container Platform オブジェクトが含まれる Node.js チャートのサンプルをダウンロードします。

```
$ git clone https://github.com/redhat-developer/redhat-helm-charts
```

3. サンプルチャートを含むディレクトリーに移動します。

```
$ cd redhat-helm-charts/alpha/nodejs-ex-k/
```

4. **Chart.yaml** ファイルを編集し、チャートの説明を追加します。

```
apiVersion: v2 (1)
name: nodejs-ex-k (2)
description: A Helm chart for OpenShift (3)
icon: https://static.redhat.com/libs/redhat/brand-assets/latest/corp/logo.svg (4)
```

(1) チャート API バージョン。これは、Helm 3 以上を必要とする Helm チャートの場合は **v2** である必要があります。

(2) チャートの名前。

(3) チャートの説明。

(4) アイコンとして使用するイメージへの URL。

5. チャートが適切にフォーマットされていることを確認します。

```
$ helm lint
```

出力例

```
[INFO] Chart.yaml: icon is recommended

1 chart(s) linted, 0 chart(s) failed
```

6. 直前のディレクトリーレベルに移動します。

```
$ cd ..
```

7. チャートをインストールします。

```
$ helm install nodejs-chart nodejs-ex-k
```

8. チャートが正常にインストールされたことを確認します。

```
$ helm list
```

出力例

```
NAME NAMESPACE REVISION UPDATED STATUS CHART APP VERSION
nodejs-chart nodejs-ex-k 1 2019-12-05 15:06:51.379134163 -0500 EST deployed nodejs-
0.1.0  1.16.0
```

4.2. カスタム HELM チャートリポジトリーの設定

Web コンソールの **Developer** パースペクティブの **Developer Catalog** には、クラスターで利用可能な Helm チャートが表示されます。デフォルトで、これは Red Hat Helm チャートリポジトリーの Helm チャートの一覧を表示します。チャートの一覧については、Red Hat **Helm** インデックス ファイルを参照してください。

クラスター管理者は、デフォルトのリポジトリーとは別に複数の Helm チャートリポジトリーを追加し、**Developer Catalog** でこれらのリポジトリーから Helm チャートを表示できます。

4.2.1. カスタム Helm チャートリポジトリーの追加

クラスター管理者は、カスタムの Helm チャートリポジトリーをクラスターに追加し、**Developer Catalog** のこれらのリポジトリーから Helm チャートへのアクセスを有効にできます。

手順

1. 新規の Helm Chart リポジトリーを追加するには、Helm Chart Repository カスタムリソース (CR) をクラスターに追加する必要があります。

   Helm チャートリポジトリー CR のサンプル

   ```
   apiVersion: helm.openshift.io/v1beta1
   kind: HelmChartRepository
   metadata:
     name: <name>
   spec:
    # optional name that might be used by console
    # name: <chart-display-name>
     connectionConfig:
       url: <helm-chart-repository-url>
   ```

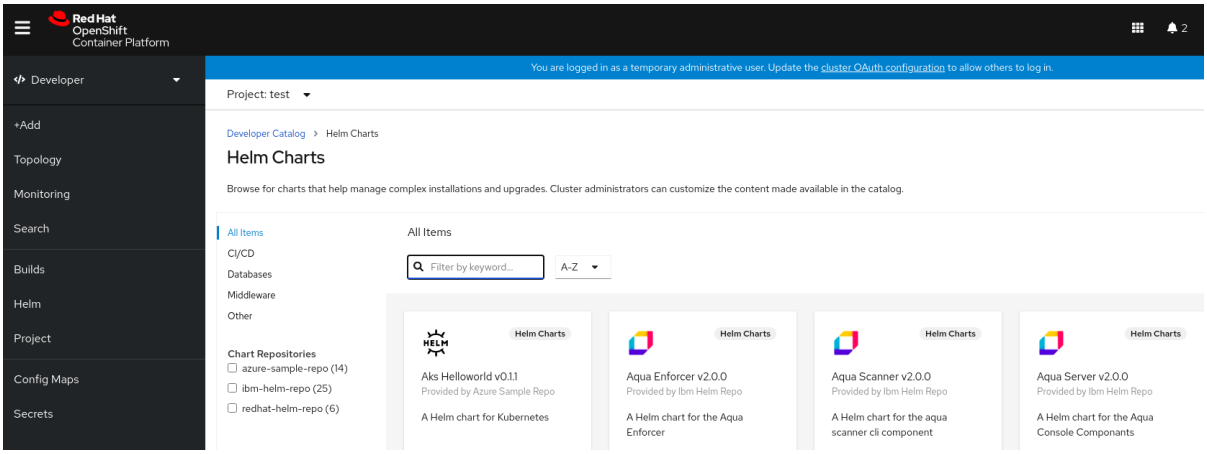
   たとえば、Azure サンプルチャートリポジトリーを追加するには、以下を実行します。

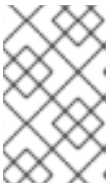
   ```
   $ cat <<EOF | oc apply -f -
   apiVersion: helm.openshift.io/v1beta1
   kind: HelmChartRepository
   metadata:
     name: azure-sample-repo
   spec:
     name: azure-sample-repo
     connectionConfig:
       url: https://raw.githubusercontent.com/Azure-Samples/helm-charts/master/docs
   EOF
   ```

2. Web コンソールで **Developer Catalog** に移動し、チャートリポジトリーの Helm チャートが表示されることを確認します。

たとえば、**Chart** リポジトリー フィルターを使用して、リポジトリーから Helm チャートを検索します。

図4.1 チャートリポジトリーのフィルター





注記

クラスター管理者がすべてのチャートリポジトリーを削除する場合は、**+Add** ビュー、**Developer Catalog**、および左側のナビゲーションパネルで Helm オプションを表示できません。

4.2.2. Helm チャートリポジトリーを追加するための認証情報および CA 証明書の作成

一部の Helm チャートリポジトリーに接続するには、認証情報とカスタム認証局 (CA) 証明書が必要です。Web コンソールと CLI を使用して認証情報と証明書を追加することができます。

手順

認証情報と証明書を設定し、CLI を使用して Helm チャートリポジトリーを追加します。

1. **openshift-config** namespace で、PEM でエンコードされた形式のカスタム CA 証明書で **ConfigMap** を作成し、これを設定マップ内の **ca-bundle.crt** キーに保存します。

```
$ oc create configmap helm-ca-cert \
--from-file=ca-bundle.crt=/path/to/certs/ca.crt \
-n openshift-config
```

2. **openshift-config** namespace で、クライアント TLS 設定を追加するために **Secret** オブジェクトを作成します。

```
$ oc create secret generic helm-tls-configs \
--from-file=tls.crt=/path/to/certs/client.crt \
--from-file=tls.key=/path/to/certs//client.key \
-n openshift-config
```

クライアント証明書とキーは PEM でエンコードされた形式であり、それぞれ **tls.crt** および **tls.key** キーに保存される必要があります。

3. 以下のように Helm リポジトリーを追加します。

```
$ cat <<EOF | oc apply -f -
apiVersion: helm.openshift.io/v1beta1
```

```
kind: HelmChartRepository
metadata:
  name: <helm-repository>
spec:
  name: <helm-repository>
  connectionConfig:
    url: <URL for the Helm repository>
    tlsConfig:
        name: helm-tls-configs
    ca:
 name: helm-ca-cert
EOF
```

ConfigMap および **Secret** は、**tlsConfig** および **ca** フィールドを使用して HelmChartRepository CR で使用されます。これらの証明書は、Helm リポジトリー URL への接続に使用されます。

4. デフォルトでは、認証されたユーザーはすべて設定済みのチャートにアクセスできます。ただし、証明書が必要なチャートリポジトリーの場合は、以下のように **openshift-config** namespace で **helm-ca-cert** 設定マップおよび **helm-tls-configs** シークレットへの読み取りアクセスを提供する必要があります。

```
$ cat <<EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: openshift-config
  name: helm-chartrepos-tls-conf-viewer
rules:
- apiGroups: [""]
  resources: ["configmaps"]
  resourceNames: ["helm-ca-cert"]
  verbs: ["get"]
- apiGroups: [""]
  resources: ["secrets"]
  resourceNames: ["helm-tls-configs"]
  verbs: ["get"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: openshift-config
  name: helm-chartrepos-tls-conf-viewer
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: 'system:authenticated'
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: helm-chartrepos-tls-conf-viewer
EOF
```

4.3. HELM チャートリポジトリーの無効化

クラスター管理者は、クラスターの Helm チャートリポジトリーを削除して、それらを **Developer Catalog** に表示されないようにすることができます。

4.3.1. クラスターでの Helm チャートリポジトリーの無効化

HelmChartRepository カスタムリソースに **disabled** プロパティーを追加して、カタログの Helm チャートを無効にすることができます。

手順

- CLI を使用して Helm チャートリポジトリーを無効にするには、**disabled: true** フラグをカスタムリソースに追加します。たとえば、Azure サンプルチャートリポジトリーを削除するには、以下を実行します。

```
$ cat <<EOF | oc apply -f -
apiVersion: helm.openshift.io/v1beta1
kind: HelmChartRepository
metadata:
  name: azure-sample-repo
spec:
  connectionConfig:
   url:https://raw.githubusercontent.com/Azure-Samples/helm-charts/master/docs
  disabled: true
EOF
```

- Web コンソールを使用して、最近追加された Helm チャートリポジトリーを無効にするには、以下を実行します。

  1. **Custom Resource Definitions**に移動し、 **HelmChartRepository** カスタムリソースを検索します。
  2. **Instances** に移動し、無効にするリポジトリーを見つけ、その名前をクリックします。
  3. **YAML** タブに移動し、**spec** セクションに **disabled: true** フラグを追加し、**Save** をクリックします。

     例

     ```
     spec:
       connectionConfig:
         url: <url-of-the-repositoru-to-be-disabled>
       disabled: true
     ```

     リポジトリーは無効にされ、カタログには表示されなくなります。

第5章 OPENSHIFT SERVERLESS で使用する KNATIVE CLI

Knative (**kn**) CLI は、OpenShift Container Platform の Knative コンポーネントとの簡単な対話を有効にします。

5.1. 主な特長

Knative (**kn**) CLI は、サーバーレスコンピューティングタスクを単純かつ簡潔にするように設計されています。Knative CLI の主な機能は次のとおりです。

- コマンドラインからサーバーレスアプリケーションをデプロイします。
- サービス、リビジョン、およびトラフィック分割などの Knative Serving の機能を管理します。
- イベントソースおよびトリガーなどの Knative Eventing コンポーネントを作成し、管理します。
- 既存の Kubernetes アプリケーションおよび Knative サービスを接続するために、sink binding を作成します。
- **kubectl** CLI と同様に、柔軟性のあるプラグインアーキテクチャーで Knative CLI を拡張します。
- Knative サービスの自動スケーリングパラメーターを設定します。
- 操作の結果を待機したり、カスタムロールアウトおよびロールバックストラテジーのデプロイなどのスクリプト化された使用。

5.2. KNATIVE CLI のインストール

Knative CLI のインストール について参照してください。

第6章 PIPELINES CLI (TKN)

6.1. TKN のインストール

tkn CLI を使用して、ターミナルから Red Hat OpenShift Pipeline を管理します。以下のセクションでは、各種の異なるプラットフォームに **tkn** をインストールする方法を説明します。

また、OpenShift Container Platform Web コンソールから最新のバイナリーへの URL を見つけるには、右上隅の？アイコンをクリックし、**Command Line Tools** を選択します。

6.1.1. Linux への Red Hat OpenShift Pipelines CLI (tkn) のインストール

Linux ディストリビューションの場合、CLI を **tar.gz** アーカイブとして直接ダウンロードできます。

手順

1. 関連する CLI をダウンロードします。
   - Linux (x86_64, amd64)
   - Linux on IBM Z and LinuxONE (s390x)
   - Linux on IBM Power Systems (ppc64le)
2. アーカイブを展開します。

```
$ tar xvzf <file>
```

3. **tkn** バイナリーを、**PATH** にあるディレクトリーに配置します。
4. **PATH** を確認するには、以下を実行します。

```
$ echo $PATH
```

6.1.2. RPM を使用した Red Hat OpenShift Pipelines CLI (tkn) の Linux へのインストール

Red Hat Enterprise Linux (RHEL) バージョン 8 の場合は、Red Hat OpenShift Pipelines CLI (**tkn**) を RPM としてインストールできます。

前提条件

- お使いの Red Hat アカウントに有効な OpenShift Container Platform サブスクリプションがある。
- ローカルシステムに root または sudo 権限がある。

手順

1. Red Hat Subscription Manager に登録します。

```
# subscription-manager register
```

2. 最新のサブスクリプションデータをプルします。

```
# subscription-manager refresh
```

3. 利用可能なサブスクリプションを一覧表示します。

```
# subscription-manager list --available --matches '*pipelines*'
```

4. 直前のコマンドの出力で、OpenShift Container Platform サブスクリプションのプール ID を見つけ、これを登録されたシステムにアタッチします。

```
# subscription-manager attach --pool=<pool_id>
```

5. Red Hat OpenShift Pipelines で必要なリポジトリーを有効にします。
   - Linux (x86_64, amd64)

```
# subscription-manager repos --enable="pipelines-1.4-for-rhel-8-x86_64-rpms"
```

   - Linux on IBM Z and LinuxONE (s390x)

```
# subscription-manager repos --enable="pipelines-1.4-for-rhel-8-s390x-rpms"
```

   - Linux on IBM Power Systems (ppc64le)

```
# subscription-manager repos --enable="pipelines-1.4-for-rhel-8-ppc64le-rpms"
```

6. **openshift-pipelines-client** パッケージをインストールします。

```
# yum install openshift-pipelines-client
```

CLI のインストール後は、**tkn** コマンドを使用して利用できます。

```
$ tkn version
```

6.1.3. Windows への Red Hat OpenShift Pipelines CLI (tkn) のインストール

Windows の場合、**tkn** CLI は **zip** アーカイブとして提供されます。

手順

1. CLI をダウンロードします。
2. ZIP プログラムでアーカイブを解凍します。
3. **tkn.exe** ファイルの場所を、 **PATH** 環境変数に追加します。
4. **PATH** を確認するには、コマンドプロンプトを開いて以下のコマンドを実行します。

```
C:\> path
```

6.1.4. macOS への Red Hat OpenShift Pipelines CLI (tkn) のインストール

macOS の場合、**tkn** CLI は **tar.gz** アーカイブとして提供されます。

手順

1. CLI をダウンロードします。
2. アーカイブを展開し、解凍します。
3. **tkn** バイナリーをパスにあるディレクトリーに移動します。
4. **PATH** を確認するには、ターミナルウィンドウを開き、以下を実行します。

```
$ echo $PATH
```

6.2. OPENSHIFT PIPELINES TKN CLI の設定

タブ補完を有効にするために Red Hat OpenShift Pipelines **tkn** CLI を設定します。

6.2.1. タブ補完の有効化

tkn CLI ツールをインストールした後に、タブ補完を有効にして **tkn** コマンドの自動補完を実行するか、または Tab キーを押す際にオプションの提案が表示されるようにできます。

前提条件

- **tkn** CLI ツールをインストールしていること。
- ローカルシステムに **bash-completion** がインストールされていること。

手順

以下の手順では、Bash のタブ補完を有効にします。

1. Bash 補完コードをファイルに保存します。

```
$ tkn completion bash > tkn_bash_completion
```

2. ファイルを **/etc/bash_completion.d/** にコピーします。

```
$ sudo cp tkn_bash_completion /etc/bash_completion.d/
```

   または、ファイルをローカルディレクトリーに保存した後に、これを **.bashrc** ファイルから取得できるようにすることができます。

タブ補完は、新規ターミナルを開くと有効にされます。

6.3. OPENSHIFT PIPELINES TKN リファレンス

このセクションでは、基本的な **tkn** CLI コマンドの一覧を紹介します。

6.3.1. 基本的な構文

tkn [command or options] [arguments…]

6.3.2. グローバルオプション

--help, -h

6.3.3. ユーティリティーコマンド

6.3.3.1. tkn

tkn CLI の親コマンド。

例: すべてのオプションの表示

```
$ tkn
```

6.3.3.2. completion [shell]

インタラクティブな補完を提供するために評価する必要があるシェル補完コードを出力します。サポートされるシェルは **bash** および **zsh** です。

例: **bash** シェルの補完コード

```
$ tkn completion bash
```

6.3.3.3. version

tkn CLI のバージョン情報を出力します。

例: **tkn** バージョンの確認

```
$ tkn version
```

6.3.4. Pipelines 管理コマンド

6.3.4.1. pipeline

Pipeline を管理します。

例: ヘルプの表示

```
$ tkn pipeline --help
```

6.3.4.2. pipeline delete

Pipeline を削除します。

例: namespace からの **mypipeline** Pipeline の削除

```
$ tkn pipeline delete mypipeline -n myspace
```

6.3.4.3. pipeline describe

Pipeline を記述します。

例: **mypipeline Pipeline** の記述

```
$ tkn pipeline describe mypipeline
```

6.3.4.4. pipeline list

Pipeline を一覧表示します。

例: **Pipeline** の一覧を表示します。

```
$ tkn pipeline list
```

6.3.4.5. pipeline logs

特定の Pipeline の Pipeline ログを表示します。

例: **mypipeline Pipeline** のライブログのストリーミング

```
$ tkn pipeline logs -f mypipeline
```

6.3.4.6. pipeline start

Pipeline を開始します。

例: **mypipeline Pipeline** の開始

```
$ tkn pipeline start mypipeline
```

6.3.5. PipelineRun コマンド

6.3.5.1. pipelinerun

PipelineRun を管理します。

例: ヘルプの表示

```
$ tkn pipelinerun -h
```

6.3.5.2. pipelinerun cancel

PipelineRun を取り消します。

例: namespace からの **mypipelinerun PipelineRun** の取り消し

```
$ tkn pipelinerun cancel mypipelinerun -n myspace
```

6.3.5.3. pipelinerun delete

PipelineRun を削除します。

例: namespace からの PipelineRun の削除

```
$ tkn pipelinerun delete mypipelinerun1 mypipelinerun2 -n myspace
```

6.3.5.4. pipelinerun describe

PipelineRun を記述します。

例: namespace の **mypipelinerun PipelineRun** の記述

```
$ tkn pipelinerun describe mypipelinerun -n myspace
```

6.3.5.5. pipelinerun list

PipelineRun を一覧表示します。

例: namespace の PipelineRun の一覧表示

```
$ tkn pipelinerun list -n myspace
```

6.3.5.6. pipelinerun logs

PipelineRun のログを表示します。

例: namespace のすべてのタスクおよび手順を含む **mypipelinerun PipelineRun** のログの表示

```
$ tkn pipelinerun logs mypipelinerun -a -n myspace
```

6.3.6. タスク管理コマンド

6.3.6.1. task

Task を管理します。

例: ヘルプの表示

```
$ tkn task -h
```

6.3.6.2. task delete

Task を削除します。

例: namespace からの **mytask1** および **mytask2 Task** の削除

```
$ tkn task delete mytask1 mytask2 -n myspace
```

6.3.6.3. task describe

Task を記述します。

例: namespace の **mytask** Task の記述

```
$ tkn task describe mytask -n myspace
```

6.3.6.4. task list

Task を一覧表示します。

例: namespace のすべての Task の一覧表示

```
$ tkn task list -n myspace
```

6.3.6.5. task logs

Task ログを表示します。

例: **mytask** Task の **mytaskrun** TaskRun のログの表示

```
$ tkn task logs mytask mytaskrun -n myspace
```

6.3.6.6. task start

Task を開始します。

例: namespace の **mytask** Task の開始

```
$ tkn task start mytask -s <ServiceAccountName> -n myspace
```

6.3.7. TaskRun コマンド

6.3.7.1. taskrun

TaskRun を管理します。

例: ヘルプの表示

```
$ tkn taskrun -h
```

6.3.7.2. taskrun cancel

TaskRun をキャンセルします。

例: namespace からの **mytaskrun** TaskRun の取り消し

```
$ tkn taskrun cancel mytaskrun -n myspace
```

6.3.7.3. taskrun delete

TaskRun を削除します。

例: namespace からの **mytaskrun1** および **mytaskrun2 TaskRun** の取り消し

```
$ tkn taskrun delete mytaskrun1 mytaskrun2 -n myspace
```

6.3.7.4. taskrun describe

TaskRun を記述します。

例: namespace の **mytaskrun TaskRun** の記述

```
$ tkn taskrun describe mytaskrun -n myspace
```

6.3.7.5. taskrun list

TaskRun を一覧表示します。

例: namespace のすべての TaskRun の一覧表示

```
$ tkn taskrun list -n myspace
```

6.3.7.6. taskrun logs

TaskRun ログを表示します。

例: namespace での **mytaskrun TaskRun** のライブログの表示

```
$ tkn taskrun logs -f mytaskrun -n myspace
```

6.3.8. 条件管理コマンド

6.3.8.1. condition

条件を管理します。

例: ヘルプの表示

```
$ tkn condition --help
```

6.3.8.2. condition delete

条件を削除します。

例: namespace からの **mycondition1** 条件の削除

```
$ tkn condition delete mycondition1 -n myspace
```

6.3.8.3. condition describe

条件を記述します。

例: namespace での **mycondition1** 条件の記述

```
$ tkn condition describe mycondition1 -n myspace
```

6.3.8.4. condition list

条件を一覧表示します。

例: namespace での条件の一覧表示

```
$ tkn condition list -n myspace
```

6.3.9. Pipeline リソース管理コマンド

6.3.9.1. resource

Pipeline リソースを管理します。

例: ヘルプの表示

```
$ tkn resource -h
```

6.3.9.2. resource create

Pipeline リソースを作成します。

例: namespace での Pipeline リソースの作成

```
$ tkn resource create -n myspace
```

これは、リソースの名前、リソースのタイプ、およびリソースのタイプに基づく値の入力を要求するインタラクティブなコマンドです。

6.3.9.3. resource delete

Pipeline リソースを削除します。

例: namespace から **myresource** Pipeline リソースを削除します。

```
$ tkn resource delete myresource -n myspace
```

6.3.9.4. resource describe

Pipeline リソースを記述します。

例: **myresource** Pipeline リソースの記述

```
$ tkn resource describe myresource -n myspace
```

6.3.9.5. resource list

Pipeline リソースを一覧表示します。

例: namespace のすべての Pipeline リソースの一覧表示

```
$ tkn resource list -n myspace
```

6.3.10. ClusterTask 管理コマンド

6.3.10.1. clustertask

ClusterTask を管理します。

例: ヘルプの表示

```
$ tkn clustertask --help
```

6.3.10.2. clustertask delete

クラスターの ClusterTask リソースを削除します。

例: **mytask1** および **mytask2 ClusterTask** の削除

```
$ tkn clustertask delete mytask1 mytask2
```

6.3.10.3. clustertask describe

ClusterTask を記述します。

例: **mytask ClusterTask** の記述

```
$ tkn clustertask describe mytask1
```

6.3.10.4. clustertask list

ClusterTask を一覧表示します。

例: ClusterTask の一覧表示

```
$ tkn clustertask list
```

6.3.10.5. clustertask start

ClusterTask を開始します。

例: **mytask ClusterTask** の開始

■

```
$ tkn clustertask start mytask
```

6.3.11. 管理コマンドのトリガー

6.3.11.1. eventlistener

EventListener を管理します。

例: ヘルプの表示

```
$ tkn eventlistener -h
```

6.3.11.2. eventlistener delete

EventListener を削除します。

例: namespace の **mylistener1** および **mylistener2 EventListener** の削除

```
$ tkn eventlistener delete mylistener1 mylistener2 -n myspace
```

6.3.11.3. eventlistener describe

EventListener を記述します。

例: namespace の **mylistener EventListener** の記述

```
$ tkn eventlistener describe mylistener -n myspace
```

6.3.11.4. eventlistener list

EventListener を一覧表示します。

例: namespace のすべての EventListener の一覧表示

```
$ tkn eventlistener list -n myspace
```

6.3.11.5. eventlistener ログ

EventListener のログを表示します。

例: namespace の **mylistener EventListener** のログ表示

```
$ tkn eventlistener logs mylistener -n myspace
```

6.3.11.6. triggerbinding

TriggerBinding を管理します。

例: TriggerBindings ヘルプの表示

```
$ tkn triggerbinding -h
```

6.3.11.7. triggerbinding delete

TriggerBinding を削除します。

例: namespace の **mybinding1** および **mybinding2 TriggerBinding** の削除

```
$ tkn triggerbinding delete mybinding1 mybinding2 -n myspace
```

6.3.11.8. triggerbinding describe

TriggerBinding を記述します。

例: namespace の **mybinding TriggerBinding** の記述

```
$ tkn triggerbinding describe mybinding -n myspace
```

6.3.11.9. triggerbinding list

TriggerBinding を一覧表示します。

例: namespace のすべての TriggerBinding の一覧表示

```
$ tkn triggerbinding list -n myspace
```

6.3.11.10. triggertemplate

TriggerTemplate を管理します。

例: TriggerTemplate ヘルプの表示

```
$ tkn triggertemplate -h
```

6.3.11.11. triggertemplate delete

TriggerTemplate を削除します。

例: namespace の **mytemplate1** および **mytemplate2 TriggerTemplate** の削除

```
$ tkn triggertemplate delete mytemplate1 mytemplate2 -n `myspace`
```

6.3.11.12. triggertemplate describe

TriggerTemplate を記述します。

例: namespace の **mytemplate TriggerTemplate** の記述

```
$ tkn triggertemplate describe mytemplate -n `myspace`
```

6.3.11.13. triggertemplate list

TriggerTemplate を一覧表示します。

例: namespace のすべての TriggerTemplate の一覧表示

```
$ tkn triggertemplate list -n myspace
```

6.3.11.14. clustertriggerbinding

ClusterTriggerBinding を管理します。

例: ClusterTriggerBinding のヘルプの表示

```
$ tkn clustertriggerbinding -h
```

6.3.11.15. clustertriggerbinding delete

ClusterTriggerBinding を削除します。

例: **myclusterbinding1** および **myclusterbinding2 ClusterTriggerBinding** の削除

```
$ tkn clustertriggerbinding delete myclusterbinding1 myclusterbinding2
```

6.3.11.16. clustertriggerbinding describe

ClusterTriggerBinding を記述します。

例: **myclusterbinding ClusterTriggerBinding** の記述

```
$ tkn clustertriggerbinding describe myclusterbinding
```

6.3.11.17. clustertriggerbinding list

ClusterTriggerBinding の一覧を表示します。

例: すべての ClusterTriggerBinding の一覧表示

```
$ tkn clustertriggerbinding list
```

6.3.12. hub 対話コマンド

タスクやパイプラインなど、リソースの Tekton Hub と対話します。

6.3.12.1. hub

ハブと対話します。

例: ヘルプの表示

```
$ tkn hub -h
```

例: ハブ API サーバーとの対話

```
$ tkn hub --api-server https://api.hub.tekton.dev
```

注記

それぞれの例で、対応するサブコマンドとフラグを取得するには、**tkn hub <command> --help** を実行します。

6.3.12.2. hub downgrade

インストール済みのリソースをダウングレードします。

例: **mynamespace** namespace の **mytask** タスクを古いバージョンにダウングレードします。

```
$ tkn hub downgrade task mytask --to version -n mynamespace
```

6.3.12.3. hub get

名前、種類、カタログ、およびバージョン別に、リソースマニフェストを取得します。

例: **tekton** カタログからの特定バージョンの **myresource Pipeline** またはタスクのマニフェスト取得

```
$ tkn hub get [pipeline | task] myresource --from tekton --version version
```

6.3.12.4. hub info

名前、種類、カタログ、およびバージョン別に、リソースに関する情報を表示します。

例: **tekton** カタログからの特定バージョンの **mytask** タスクについての情報表示

```
$ tkn hub info task mytask --from tekton --version version
```

6.3.12.5. hub install

種類、名前、バージョンごとにカタログからのリソースをインストールします。

例: **mynamespace** namespace の **tekton** カタログから **mytask** タスクの特定のバージョンのインストール

```
$ tkn hub install task mytask --from tekton --version version -n mynamespace
```

6.3.12.6. hub reinstall

種類および名前ごとにリソースを再インストールします。

例: **mynamespace** namespace の **tekton** カタログから **mytask** タスクの特定のバージョンの再インストール

```
$ tkn hub reinstall task mytask --from tekton --version version -n mynamespace
```

6.3.12.7. hub search

名前、種類、およびタグの組み合わせでリソースを検索します。

例: タグ **cli** でのリソースの検索

```
$ tkn hub search --tags cli
```

6.3.12.8. hub upgrade

インストール済みのリソースをアップグレードします。

例: **mynamespace** namespace のインストールされた **mytask** タスクの新規バージョンへのアップグレード

```
$ tkn hub upgrade task mytask --to version -n mynamespace
```

第7章 OPM CLI

7.1. OPM について

opm CLI ツールは、Operator Bundle Format で使用するために Operator Framework によって提供されます。このツールを使用して、ソフトウェアリポジトリーに相当する **index** と呼ばれるバンドルの一覧から Operator のカタログを作成し、維持することができます。結果として、インデックスイメージというコンテナーイメージをコンテナーレジストリーに保存し、その後にクラスターにインストールできます。

インデックスには、コンテナーイメージの実行時に提供される組み込まれた API を使用してクエリーできる、Operator マニフェストコンテンツへのポインターのデータベースが含まれます。OpenShift Container Platform では、Operator Lifecycle Manager (OLM) はインデックスイメージを **CatalogSource** オブジェクトで参照し、これをカタログとして使用できます。これにより、クラスター上にインストールされた Operator への頻度の高い更新を可能にするためにイメージを一定の間隔でポーリングできます。

関連情報

- Bundle Format についての詳細は、Operator Framework パッケージ形式 を参照してください。
- Operator SDK を使用してバンドルイメージを作成するには、バンドルイメージの使用 を参照してください。

7.2. OPM のインストール

opm CLI ツールは、Linux、macOS、または Windows ワークステーションにインストールできます。

前提条件

- Linux の場合は、以下のパッケージを指定する必要があります。RHEL 8 は、以下の要件を満たすようにします。
  - **podman** バージョン 1.9.3 以降 (バージョン 2.0 以降を推奨)
  - **glibc** バージョン 2.28 以降

手順

1. OpenShift mirror site に移動し、お使いのオペレーティングシステムに一致する最新バージョンの tarball をダウンロードします。
2. アーカイブを展開します。
   - Linux または macOS の場合:

     ```
     $ tar xvf <file>
     ```

   - Windows の場合、ZIP プログラムでアーカイブを解凍します。
3. ファイルを **PATH** の任意の場所に置きます。
   - Linux または macOS の場合:

a. **PATH** を確認します。

```
$ echo $PATH
```

b. ファイルを移動します。以下に例を示します。

```
$ sudo mv ./opm /usr/local/bin/
```

- Windows の場合:

a. **PATH** を確認します。

```
C:\> path
```

b. ファイルを移動します。

```
C:\> move opm.exe <directory>
```

検証

- **opm** CLI のインストール後に、これが利用可能であることを確認します。

```
$ opm version
```

出力例

```
Version: version.Version{OpmVersion:"v1.15.4-2-g6183dbb3",
GitCommit:"6183dbb3567397e759f25752011834f86f47a3ea", BuildDate:"2021-02-
13T04:16:08Z", GoOs:"linux", GoArch:"amd64"}
```

7.3. 関連情報

- インデックスイメージの作成、更新、プルーニングを含む **opm** の手順は、カスタムカタログの管理 を参照してください。

第8章 OPERATOR SDK

8.1. OPERATOR SDK CLI のインストール

Operator SDK は、Operator 開発者が Operator のビルド、テストおよびデプロイに使用できるコマンドラインインターフェイス (CLI) ツールを提供します。ワークステーションに Operator SDK CLI をインストールして、独自の Operator のオーサリングを開始することができます。

Operator SDK についての詳細は、Operator の開発 について参照してください。

注記

OpenShift Container Platform 4.7 は Operator SDK v1.3.0 をサポートします。

8.1.1. Operator SDK CLI のインストール

OpenShift SDK CLI ツールは Linux にインストールできます。

前提条件

- Go v1.13+
- **docker** v17.03+、 **podman** v1.9.3+、または **buildah** v1.7+

手順

1. OpenShift ミラーサイト に移動します。
2. **4.7.23** ディレクトリーから、Linux 用の最新バージョンの tarball をダウンロードします。
3. アーカイブを展開します。

```
$ tar xvf operator-sdk-v1.3.0-ocp-linux-x86_64.tar.gz
```

4. ファイルを実行可能にします。

```
$ chmod +x operator-sdk
```

5. 展開された **operator-sdk** バイナリーを **PATH** にあるディレクトリーに移動します。

ヒント

PATH を確認するには、以下を実行します。

```
$ echo $PATH
```

```
$ sudo mv ./operator-sdk /usr/local/bin/operator-sdk
```

検証

- Operator SDK CLI のインストール後に、これが利用可能であることを確認します。

```
$ operator-sdk version
```

出力例

```
operator-sdk version: "v1.3.0-ocp", ...
```

8.2. OPERATOR SDK CLI リファレンス

Operator SDK コマンドラインインターフェイス (CLI) は、Operator の作成を容易にするために設計された開発キットです。

Operator SDK CLI 構文

```
$ operator-sdk <command> [<subcommand>] [<argument>] [<flags>]
```

Kubernetes ベースのクラスター (OpenShift Container Platform など) へのクラスター管理者のアクセスのある Operator の作成者は、Operator SDK CLI を使用して Go、Ansible、または Helm をベースに独自の Operator を開発できます。Kubebuilder は Go ベースの Operator のスキャフォールディングソリューションとして Operator SDK に組み込まれます。つまり、既存の Kubebuilder プロジェクトは Operator SDK でそのまま使用でき、引き続き機能します。

Operator SDK についての詳細は、Operator の開発 について参照してください。

8.2.1. bundle

operator-sdk bundle コマンドは Operator バンドルメタデータを管理します。

8.2.1.1. validate

bundle validate サブコマンドは Operator バンドルを検証します。

表8.1 **bundle validate** フラグ

フラグ	説明
-h, **--help**	**bundle validate** サブコマンドのヘルプ出力。
--index-builder (文字列)	バンドルイメージをプルおよび展開するためのツール。バンドルイメージを検証する場合にのみ使用されます。使用できるオプションは、**docker** (デフォルト)、**podman**、または **none** です。
--list-optional	利用可能なすべてのオプションのバリデーターを一覧表示します。これが設定されている場合、バリデーターは実行されません。
--select-optional (文字列)	実行するオプションのバリデーターを選択するラベルセレクター。**--list-optional** フラグを指定して実行する場合は、利用可能なオプションのバリデーターを一覧表示します。

8.2.2. cleanup

operator-sdk cleanup コマンドは、**run** コマンドでデプロイされた Operator 用に作成されたリソースを破棄し、削除します。

表8.2 **cleanup** フラグ

フラグ	説明
-h, **--help**	**run bundle** サブコマンドのヘルプ出力。
--kubeconfig (文字列)	CLI 要求に使用する **kubeconfig** ファイルへのパス。
n, **--namespace** (文字列)	CLI 要求がある場合の CLI 要求を実行する namespace。
--timeout <duration>	コマンドが失敗せずに完了するまでの待機時間。デフォルト値は **2m0s** です。

8.2.3. completion

operator-sdk completion コマンドは、CLI コマンドをより迅速に、より容易に実行できるようにシェル補完を生成します。

表8.3 **completion** サブコマンド

サブコマンド	説明
bash	bash 補完を生成します。
zsh	zsh 補完を生成します。

表8.4 **completion** フラグ

フラグ	説明
-h, --help	使用方法についてのヘルプの出力。

以下に例を示します。

```
$ operator-sdk completion bash
```

出力例

```
# bash completion for operator-sdk                  -*- shell-script -*-
...
# ex: ts=4 sw=4 et filetype=sh
```

8.2.4. create

operator-sdk create コマンドは、Kubernetes API の作成または スキャフォールディング に使用されます。

8.2.4.1. api

create api サブコマンドは Kubernetes API をスキャフォールディングします。サブコマンドは、 **init** コマンドで初期化されたプロジェクトで実行する必要があります。

表8.5 **create api** フラグ

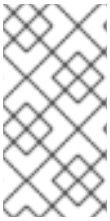
フラグ	説明
-h, **--help**	**run bundle** サブコマンドのヘルプ出力。

8.2.5. generate

operator-sdk generate コマンドは特定のジェネレーターを起動して、必要に応じてコードを生成します。

8.2.5.1. bundle

generate bundle サブコマンドは、Operator プロジェクトのバンドルマニフェスト、メタデータ、および **bundle.Dockerfile** ファイルのセットを生成します。



注記

通常は、最初に **generate kustomize manifests** サブコマンドを実行して、**generate bundle** サブコマンドで使用される入力された Kustomize ベースを生成します。ただし、初期化されたプロジェクトで **make bundle** コマンドを使用して、これらのコマンドの順次の実行を自動化できます。

表8.6 **generate bundle** フラグ

フラグ	説明
--channels (文字列)	バンドルが属するチャネルのコンマ区切りリスト。デフォルト値は **alpha** です。
--crds-dir (文字列)	**CustomResoureDefinition** マニフェストのルートディレクトリー。
--default-channel (文字列)	バンドルのデフォルトチャネル。
--deploy-dir (文字列)	デプロイメントや RBAC などの Operator マニフェストのルートディレクトリー。このディレクトリーは、**--input-dir** フラグに渡されるディレクトリーとは異なります。
-h, **--help**	**generate bundle** のヘルプ
--input-dir (文字列)	既存のバンドルを読み取るディレクトリー。このディレクトリーは、バンドル **manifests** ディレクトリーの親であり、**--deploy-dir** ディレクトリーとは異なります。

フラグ	説明
--kustomize-dir (文字列)	バンドルマニフェストの Kustomize ベースおよび **kustomization.yaml** ファイルを含むディレクトリー。デフォルトのパスは **config/manifests** です。
--manifests	バンドルマニフェストを生成します。
--metadata	バンドルメタデータと Dockerfile を生成します。
--output-dir (文字列)	バンドルを書き込むディレクトリー。
--overwrite	バンドルメタデータおよび Dockerfile を上書きします (ある場合)。デフォルト値は **true** です。
--package (文字列)	バンドルのパッケージ名。
-q、**--quiet**	quiet モードで実行します。
--stdout	バンドルマニフェストを標準出力に書き込みます。
--version (文字列)	生成されたバンドルの Operator のセマンティックバージョン。新規バンドルを作成するか、または Operator をアップグレードする場合にのみ設定します。

関連情報

- **generate bundle** サブコマンドを呼び出すための **make bundle** コマンドの使用を含む詳細な手順については、Operator のバンドルおよび Operator Lifecycle Manager を使用したデプロイを参照してください。

8.2.5.2. kustomize

generate kustomize サブコマンドには、Operator の Kustomize データを生成するサブコマンドが含まれます。

8.2.5.2.1. manifests

generate kustomize manifests は Kustomize ベースを生成または再生成し、**kustomization.yaml** ファイルを **config/manifests** ディレクトリーに生成または再生成します。これは、他の Operator SDK コマンドでバンドルマニフェストをビルドするために使用されます。このコマンドは、ベースがすでに存在しない場合や **--interactive=false** フラグが設定されていない場合に、デフォルトでマニフェストベースの重要なコンポーネントである UI メタデータを対話的に要求します。

表8.7 **generate kustomize manifests** フラグ

フラグ	説明
--apis-dir (文字列)	API タイプ定義のルートディレクトリー。
-h, **--help**	**generate kustomize manifests** のヘルプ。

フラグ	説明
--input-dir (文字列)	既存の Kustomize ファイルを含むディレクトリー。
--interactive	**false** に設定すると、Kustomize ベースが存在しない場合は、対話式コマンドプロンプトがカスタムメタデータを受け入れるように表示されます。
--output-dir (文字列)	Kustomize ファイルを書き込むディレクトリー。
--package (文字列)	パッケージ名。
-q、--quiet	quiet モードで実行します。

8.2.6. init

operator-sdk init コマンドは Operator プロジェクトを初期化し、指定されたプラグインのデフォルトのプロジェクトディレクトリーレイアウトを生成または スキャフォールド します。

このコマンドは、以下のファイルを作成します。

- ボイラープレートライセンスファイル
- ドメインおよびリポジトリーを含む **PROJECT** ファイル
- プロジェクトをビルドする **Makefile**
- プロジェクト依存関係のある **go.mod** ファイル
- マニフェストをカスタマイズするための **kustomization.yaml** ファイル
- マネージャーマニフェストのイメージをカスタマイズするためのパッチファイル
- Prometheus メトリクスを有効にするためのパッチファイル
- 実行する **main.go** ファイル

表8.8 **init** フラグ

フラグ	説明
--help, -h	**init** コマンドのヘルプ出力。
--plugins (文字列)	プロジェクトを初期化するプラグインの名前およびオプションのバージョン。利用可能なプラグインは **ansible.sdk.operatorframework.io/v1**、**go.kubebuilder.io/v2**、**go.kubebuilder.io/v3**、および **helm.sdk.operatorframework.io/v1** です。
--project-version	プロジェクトのバージョン。使用できる値は **2** および **3-alpha** (デフォルト) です。

8.2.7. run

operator-sdk run コマンドは、さまざまな環境で Operator を起動できるオプションを提供します。

8.2.7.1. bundle

run bundle サブコマンドは、Operator Lifecycle Manager (OLM) を使用してバンドル形式で Operator をデプロイします。

表8.9 **run bundle** フラグ

フラグ	説明
--index-image (文字列)	バンドルを挿入するインデックスイメージ。デフォルトのイメージは **quay.io/operator-framework/upstream-opm-builder:latest** です。
--install-mode <install_mode_value>	Operator のクラスターサービスバージョン (CSV) によってサポートされるインストールモード (例: **AllNamespaces** または **SingleNamespace**)。
--timeout <duration>	インストールのタイムアウト。デフォルト値は **2m0s** です。
--kubeconfig (文字列)	CLI 要求に使用する **kubeconfig** ファイルへのパス。
n, **--namespace** (文字列)	CLI 要求がある場合の CLI 要求を実行する namespace。
-h, **--help**	**run bundle** サブコマンドのヘルプ出力。

関連情報

- 使用可能なインストールモードに関する詳細は、Operator グループメンバーシップ を参照してください。

8.2.7.2. bundle-upgrade

run bundle-upgrade サブコマンドは、以前に Operator Lifecycle Manager (OLM) を使用してバンドル形式でインストールされた Operator をアップグレードします。

表8.10 **run bundle-upgrade** フラグ

フラグ	説明
--timeout <duration>	アップグレードのタイムアウト。デフォルト値は **2m0s** です。
--kubeconfig (文字列)	CLI 要求に使用する **kubeconfig** ファイルへのパス。
n, **--namespace** (文字列)	CLI 要求がある場合の CLI 要求を実行する namespace。
-h, **--help**	**run bundle** サブコマンドのヘルプ出力。

8.2.8. scorecard

operator-sdk scorecard コマンドは、スコアカードツールを実行して Operator バンドルを検証し、改善に向けた提案を提供します。このコマンドは、バンドルイメージまたはマニフェストおよびメタデータを含むディレクトリーのいずれかの引数を取ります。引数がイメージタグを保持する場合は、イメージはリモートに存在する必要があります。

表8.11 **scorecard** フラグ

フラグ	説明
-c, **--config** (文字列)	スコアカード設定ファイルへのパス。デフォルトのパスは **bundle/tests/scorecard/config.yaml** です。
-h, **--help**	**scorecard** コマンドのヘルプ出力。
--kubeconfig (文字列)	**kubeconfig** ファイルへのパス。
-L, **--list**	実行可能なテストを一覧表示します。
-n, --namespace (文字列)	テストイメージを実行する namespace。
-o, **--output** (文字列)	結果の出力形式。使用できる値はデフォルトの **text**、および **json** です。
-l, **--selector** (文字列)	実行されるテストを決定するラベルセレクター。
-s, **--service-account** (文字列)	テストに使用するサービスアカウント。デフォルト値は **default** です。
-x, **--skip-cleanup**	テストの実行後にリソースクリーンアップを無効にします。
-w, **--wait-time <duration>**	テストが完了するのを待つ秒数 (例: **35s**)。デフォルト値は **30s** です。

関連情報

- スコアカードツールの実行に関する詳細は、スコアカードツールを使用した Operator の検証について参照してください。