OpenShift Container Platform 4.8

CLI ツール

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

Last Updated: 2023-04-07

OpenShift Container Platform 4.8 CLI ツール

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

法律上の通知

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

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。
- 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 4.8 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。

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 v4.8 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>
```

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 v4.8 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>
```

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 v4.8 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

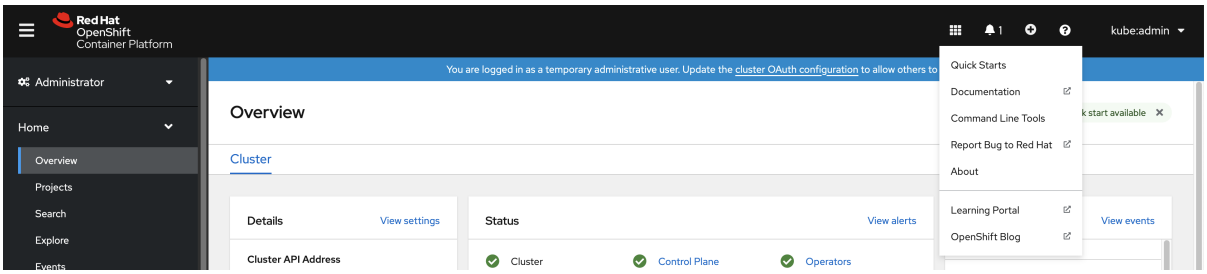
以前のバージョンの **oc** をインストールしている場合、これを使用して OpenShift Container Platform 4.8 のすべてのコマンドを実行することはできません。新規バージョンの **oc** をダウンロードし、インストールします。

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

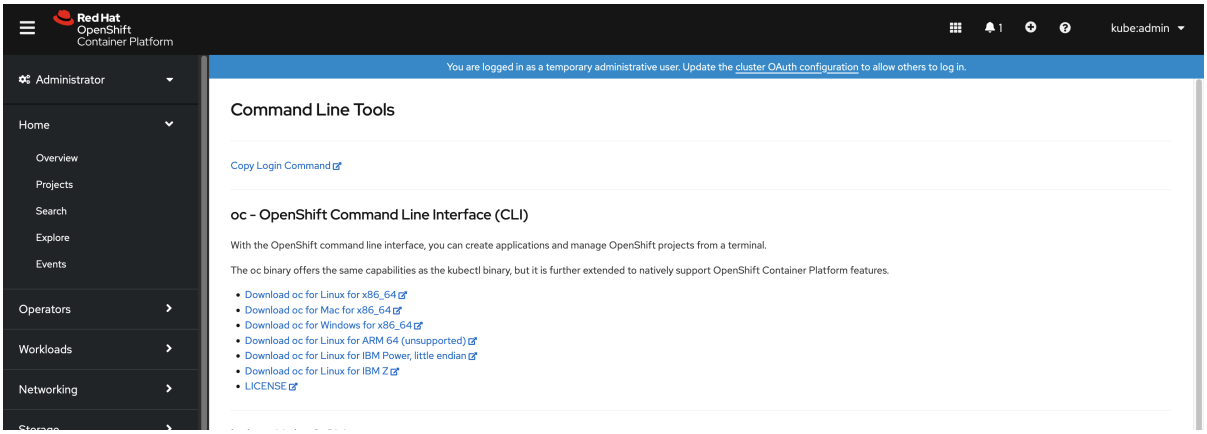
以下の手順を使用して、OpenShift CLI (**oc**) バイナリーを Linux にインストールできます。

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。

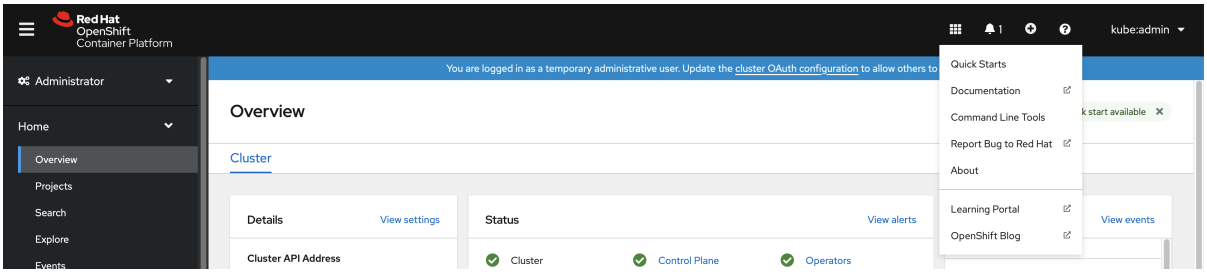
```
$ oc <command>
```

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

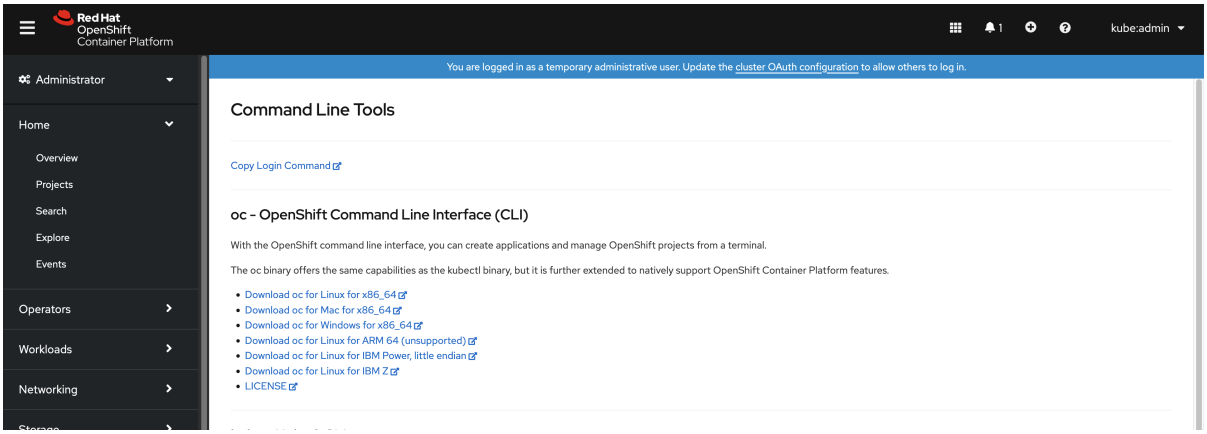
以下の手順を使用して、OpenShift CLI(**oc**) バイナリーを Windows にインストールできます。

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>
```

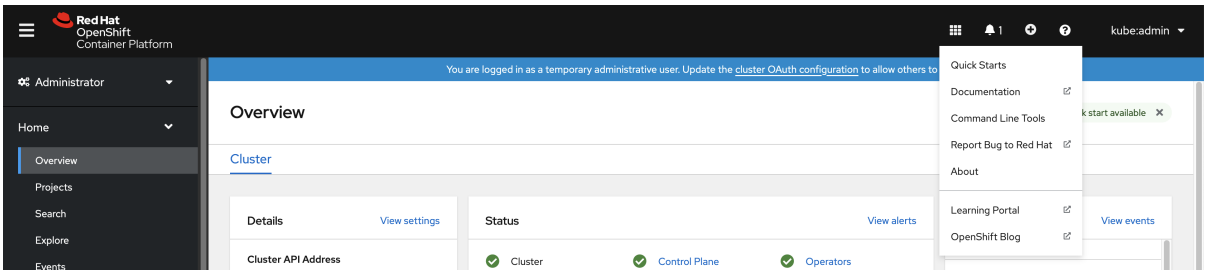
2.1.2.2.3. Web コンソールを使用した macOS への OpenShift CLI のインストール

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。

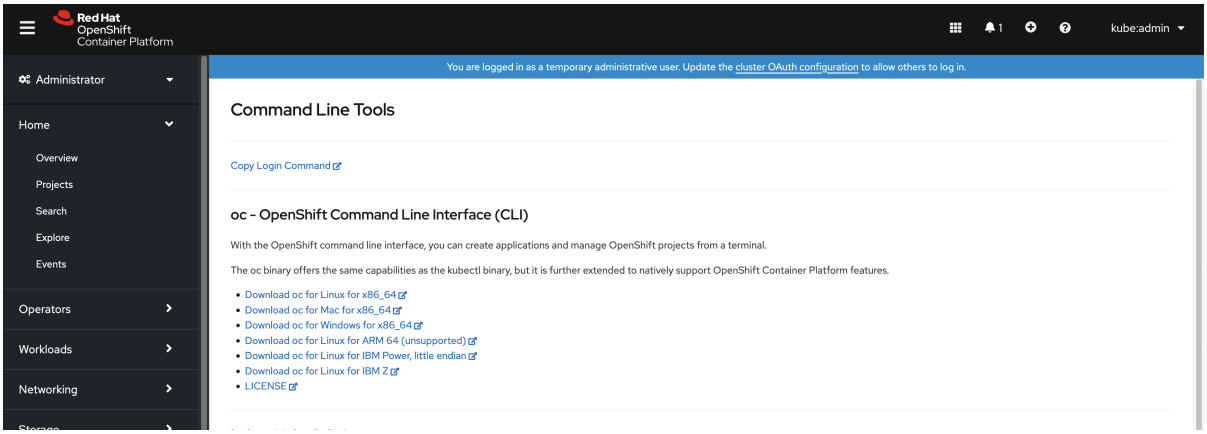
手順

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 4.8 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-4.8-for-rhel-8-x86_64-rpms"
```

- Red Hat Enterprise Linux 7 の場合:

```
# subscription-manager repos --enable="rhel-7-server-ose-4.8-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。

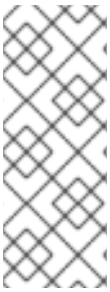
```
$ brew install openshift-cli
```

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

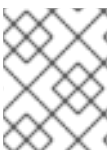
```
--> Found image 40de956 (9 days old) in imagestream "openshift/php" under tag "7.2" for "php"

...

    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".
```

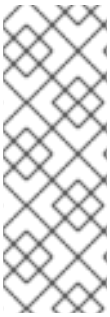
oc login コマンドを再度実行し、対話式プロセス中に必要な情報を指定して、ユーザー認証情報およびクラスターの詳細の他の組み合わせを使用してログインできます。コンテキストが存在しない場合は、コンテキストが指定される情報に基づいて作成されます。すでにログインしている場合で、現行ユーザーがアクセス可能な別のプロジェクトに切り替える場合には、**oc project** コマンドを使用してプロジェクトの名前を入力します。

```
$ 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。

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** のコマンドで起動します。

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。

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更新して実行可能にします。

```
$ 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
```

プラグインを呼び出すコマンドは、プラグインのファイル名に依存す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 開発者コマンドリファレンス

このリファレンスは、OpenShift CLI (**oc**) 開発者コマンドの説明とコマンド例を示しています。管理者コマンドについては、OpenShift CLI 管理者コマンドリファレンス を参照してください。

oc help を実行して、すべてのコマンドを表示するか、または **oc <command> --help** を実行して、特定のコマンドに関する追加情報を取得します。

2.5.1. OpenShift CLI (oc) 開発者コマンド

2.5.1.1. oc annotate

リソースへのアノテーションを更新します。

使用例

```
# Update pod 'foo' with the annotation 'description' and the value 'my frontend'.
# If the same annotation is set multiple times, only the last value will be applied
oc annotate pods foo description='my frontend'

# Update a pod identified by type and name in "pod.json"
oc annotate -f pod.json description='my frontend'

# Update pod 'foo' with the annotation 'description' and the value 'my frontend running nginx',
overwriting any existing value.
oc annotate --overwrite pods foo description='my frontend running nginx'

# Update all pods in the namespace
```

```
oc annotate pods --all description='my frontend running nginx'

# Update pod 'foo' only if the resource is unchanged from version 1.
oc annotate pods foo description='my frontend running nginx' --resource-version=1

# Update pod 'foo' by removing an annotation named 'description' if it exists.
# Does not require the --overwrite flag.
oc annotate pods foo description-
```

2.5.1.2. oc api-resources

サーバー上のサポートされている API リソースを出力します。

使用例

```
# Print the supported API Resources
oc api-resources

# Print the supported API Resources with more information
oc api-resources -o wide

# Print the supported API Resources sorted by a column
oc api-resources --sort-by=name

# Print the supported namespaced resources
oc api-resources --namespaced=true

# Print the supported non-namespaced resources
oc api-resources --namespaced=false

# Print the supported API Resources with specific APIGroup
oc api-resources --api-group=extensions
```

2.5.1.3. oc api-versions

group/version という形式で、サーバー上でサポートされる API バージョンを出力します。

使用例

```
# Print the supported API versions
oc api-versions
```

2.5.1.4. oc apply

設定をファイル名または標準入力 (stdin) 別のリソースに適用します。

使用例

```
# Apply the configuration in pod.json to a pod.
oc apply -f ./pod.json

# Apply resources from a directory containing kustomization.yaml - e.g. dir/kustomization.yaml.
oc apply -k dir/
```

```
# Apply the JSON passed into stdin to a pod.
cat pod.json | oc apply -f -

# Note: --prune is still in Alpha
# Apply the configuration in manifest.yaml that matches label app=nginx and delete all the other resources that are not in the file and match label app=nginx.
oc apply --prune -f manifest.yaml -l app=nginx

# Apply the configuration in manifest.yaml and delete all the other configmaps that are not in the file.
oc apply --prune -f manifest.yaml --all --prune-whitelist=core/v1/ConfigMap
```

2.5.1.5. oc apply edit-last-applied

リソース/オブジェクトの最新の last-applied-configuration アノテーションを編集します。

使用例

```
# Edit the last-applied-configuration annotations by type/name in YAML.
oc apply edit-last-applied deployment/nginx

# Edit the last-applied-configuration annotations by file in JSON.
oc apply edit-last-applied -f deploy.yaml -o json
```

2.5.1.6. oc apply set-last-applied

ファイルの内容に一致するように、ライブオブジェクトに last-applied-configuration アノテーションを設定します。

使用例

```
# Set the last-applied-configuration of a resource to match the contents of a file.
oc apply set-last-applied -f deploy.yaml

# Execute set-last-applied against each configuration file in a directory.
oc apply set-last-applied -f path/

# Set the last-applied-configuration of a resource to match the contents of a file, will create the annotation if it does not already exist.
oc apply set-last-applied -f deploy.yaml --create-annotation=true
```

2.5.1.7. oc apply view-last-applied

リソース/オブジェクトの最新の last-applied-configuration アノテーションを表示します。

使用例

```
# View the last-applied-configuration annotations by type/name in YAML.
oc apply view-last-applied deployment/nginx

# View the last-applied-configuration annotations by file in JSON
oc apply view-last-applied -f deploy.yaml -o json
```

2.5.1.8. oc attach

実行中のコンテナーに割り当てます。

使用例

```
# Get output from running pod mypod, use the oc.kubernetes.io/default-container annotation
# for selecting the container to be attached or the first container in the pod will be chosen
oc attach mypod

# Get output from ruby-container from pod mypod
oc attach mypod -c ruby-container

# Switch to raw terminal mode, sends stdin to 'bash' in ruby-container from pod mypod
# and sends stdout/stderr from 'bash' back to the client
oc attach mypod -c ruby-container -i -t

# Get output from the first pod of a ReplicaSet named nginx
oc attach rs/nginx
```

2.5.1.9. oc auth can-i

アクションが可能かどうかを確認します。

使用例

```
# Check to see if I can create pods in any namespace
oc auth can-i create pods --all-namespaces

# Check to see if I can list deployments in my current namespace
oc auth can-i list deployments.apps

# Check to see if I can do everything in my current namespace ("*" means all)
oc auth can-i '*' '*'

# Check to see if I can get the job named "bar" in namespace "foo"
oc auth can-i list jobs.batch/bar -n foo

# Check to see if I can read pod logs
oc auth can-i get pods --subresource=log

# Check to see if I can access the URL /logs/
oc auth can-i get /logs/

# List all allowed actions in namespace "foo"
oc auth can-i --list --namespace=foo
```

2.5.1.10. oc auth reconcile

RBAC Role、RoleBinding、ClusterRole、および ClusterRoleBinding オブジェクトのルールを調整します。

使用例

```
# Reconcile rbac resources from a file
oc auth reconcile -f my-rbac-rules.yaml
```

2.5.1.11. oc autoscale

デプロイメント設定、デプロイメント、レプリカセット、ステートフルセット、またはレプリケーションコントローラーを自動スケーリングします。

使用例

```
# Auto scale a deployment "foo", with the number of pods between 2 and 10, no target CPU utilization specified so a default autoscaling policy will be used:
oc autoscale deployment foo --min=2 --max=10

# Auto scale a replication controller "foo", with the number of pods between 1 and 5, target CPU utilization at 80%:
oc autoscale rc foo --max=5 --cpu-percent=80
```

2.5.1.12. oc cancel-build

実行中、保留中、または新規のビルドを取り消します。

使用例

```
# Cancel the build with the given name
oc cancel-build ruby-build-2

# Cancel the named build and print the build logs
oc cancel-build ruby-build-2 --dump-logs

# Cancel the named build and create a new one with the same parameters
oc cancel-build ruby-build-2 --restart

# Cancel multiple builds
oc cancel-build ruby-build-1 ruby-build-2 ruby-build-3

# Cancel all builds created from the 'ruby-build' build config that are in the 'new' state
oc cancel-build bc/ruby-build --state=new
```

2.5.1.13. oc cluster-info

クラスターの情報を表示します。

使用例

```
# Print the address of the control plane and cluster services
oc cluster-info
```

2.5.1.14. oc cluster-info dump

デバッグおよび診断に関する関連情報を多数ダンプします。

使用例

```
# Dump current cluster state to stdout
oc cluster-info dump

# Dump current cluster state to /path/to/cluster-state
oc cluster-info dump --output-directory=/path/to/cluster-state

# Dump all namespaces to stdout
oc cluster-info dump --all-namespaces

# Dump a set of namespaces to /path/to/cluster-state
oc cluster-info dump --namespaces default,kube-system --output-directory=/path/to/cluster-state
```

2.5.1.15. oc completion

指定されたシェル (bash または zsh) の補完コードを出力します。

使用例

```
# Installing bash completion on macOS using homebrew
## If running Bash 3.2 included with macOS
brew install bash-completion
## or, if running Bash 4.1+
brew install bash-completion@2
## If oc is installed via homebrew, this should start working immediately.
## If you've installed via other means, you may need add the completion to your completion directory
oc completion bash > $(brew --prefix)/etc/bash_completion.d/oc


# Installing bash completion on Linux
## If bash-completion is not installed on Linux, please install the 'bash-completion' package
## via your distribution's package manager.
## Load the oc completion code for bash into the current shell
source <(oc completion bash)
## Write bash completion code to a file and source it from .bash_profile
oc completion bash > ~/.kube/completion.bash.inc
printf "
# Kubectl shell completion
source '$HOME/.kube/completion.bash.inc'
" >> $HOME/.bash_profile
source $HOME/.bash_profile

# Load the oc completion code for zsh[1] into the current shell
source <(oc completion zsh)
# Set the oc completion code for zsh[1] to autoload on startup
oc completion zsh > "${fpath[1]}/_oc"
```

2.5.1.16. oc config current-context

current-context を表示します

使用例

```
# Display the current-context
oc config current-context
```

2.5.1.17. oc config delete-cluster

kubeconfig から指定されたクラスターを削除します。

使用例

```
# Delete the minikube cluster
oc config delete-cluster minikube
```

2.5.1.18. oc config delete-context

kubeconfig から指定されたコンテキストを削除します。

使用例

```
# Delete the context for the minikube cluster
oc config delete-context minikube
```

2.5.1.19. oc config delete-user

kubeconfig から指定されたユーザーを削除します。

使用例

```
# Delete the minikube user
oc config delete-user minikube
```

2.5.1.20. oc config get-clusters

kubeconfig に定義されるクラスターを表示します。

使用例

```
# List the clusters oc knows about
oc config get-clusters
```

2.5.1.21. oc config get-contexts

コンテキストを 1 つまたは複数記述します。

使用例

```
# List all the contexts in your kubeconfig file
oc config get-contexts

# Describe one context in your kubeconfig file.
oc config get-contexts my-context
```

2.5.1.22. oc config get-users

kubeconfig で定義されるユーザーを表示します。

使用例

```
# List the users oc knows about
oc config get-users
```

2.5.1.23. oc config rename-context

kubeconfig ファイルからのコンテキストの名前を変更します。

使用例

```
# Rename the context 'old-name' to 'new-name' in your kubeconfig file
oc config rename-context old-name new-name
```

2.5.1.24. oc config set

kubeconfig ファイルに個別の値を設定します。

使用例

```
# Set server field on the my-cluster cluster to https://1.2.3.4
oc config set clusters.my-cluster.server https://1.2.3.4

# Set certificate-authority-data field on the my-cluster cluster.
oc config set clusters.my-cluster.certificate-authority-data $(echo "cert_data_here" | base64 -i -)

# Set cluster field in the my-context context to my-cluster.
oc config set contexts.my-context.cluster my-cluster

# Set client-key-data field in the cluster-admin user using --set-raw-bytes option.
oc config set users.cluster-admin.client-key-data cert_data_here --set-raw-bytes=true
```

2.5.1.25. oc config set-cluster

kubeconfig でクラスターエントリーを設定します。

使用例

```
# Set only the server field on the e2e cluster entry without touching other values.
oc config set-cluster e2e --server=https://1.2.3.4

# Embed certificate authority data for the e2e cluster entry
oc config set-cluster e2e --embed-certs --certificate-authority=~/.kube/e2e/kubernetes.ca.crt

# Disable cert checking for the dev cluster entry
oc config set-cluster e2e --insecure-skip-tls-verify=true

# Set custom TLS server name to use for validation for the e2e cluster entry
oc config set-cluster e2e --tls-server-name=my-cluster-name
```

2.5.1.26. oc config set-context

kubeconfig のコンテキストエントリーを設定します。

使用例

```
  # Set the user field on the gce context entry without touching other values
  oc config set-context gce --user=cluster-admin
```

2.5.1.27. oc config set-credentials

kubeconfig のユーザーエントリーを設定します。

使用例

```
  # Set only the "client-key" field on the "cluster-admin"
  # entry, without touching other values:
  oc config set-credentials cluster-admin --client-key=~/.kube/admin.key

  # Set basic auth for the "cluster-admin" entry
  oc config set-credentials cluster-admin --username=admin --password=uXFGweU9l35qcif

  # Embed client certificate data in the "cluster-admin" entry
  oc config set-credentials cluster-admin --client-certificate=~/.kube/admin.crt --embed-certs=true

  # Enable the Google Compute Platform auth provider for the "cluster-admin" entry
  oc config set-credentials cluster-admin --auth-provider=gcp

  # Enable the OpenID Connect auth provider for the "cluster-admin" entry with additional args
  oc config set-credentials cluster-admin --auth-provider=oidc --auth-provider-arg=client-id=foo --auth-
provider-arg=client-secret=bar

  # Remove the "client-secret" config value for the OpenID Connect auth provider for the "cluster-
admin" entry
  oc config set-credentials cluster-admin --auth-provider=oidc --auth-provider-arg=client-secret-

  # Enable new exec auth plugin for the "cluster-admin" entry
  oc config set-credentials cluster-admin --exec-command=/path/to/the/executable --exec-api-
version=client.authentication.k8s.io/v1beta1

  # Define new exec auth plugin args for the "cluster-admin" entry
  oc config set-credentials cluster-admin --exec-arg=arg1 --exec-arg=arg2

  # Create or update exec auth plugin environment variables for the "cluster-admin" entry
  oc config set-credentials cluster-admin --exec-env=key1=val1 --exec-env=key2=val2

  # Remove exec auth plugin environment variables for the "cluster-admin" entry
  oc config set-credentials cluster-admin --exec-env=var-to-remove-
```

2.5.1.28. oc config unset

kubeconfig ファイルでの個別値の設定を解除します。

使用例

```
# Unset the current-context.
oc config unset current-context

# Unset namespace in foo context.
oc config unset contexts.foo.namespace
```

2.5.1.29. oc config use-context

kubeconfig ファイルで current-context を設定します。

使用例

```
# Use the context for the minikube cluster
oc config use-context minikube
```

2.5.1.30. oc config view

マージされた kubeconfig 設定または指定された kubeconfig ファイルを表示します。

使用例

```
# Show merged kubeconfig settings.
oc config view

# Show merged kubeconfig settings and raw certificate data.
oc config view --raw

# Get the password for the e2e user
oc config view -o jsonpath='{.users[?(@.name == "e2e")].user.password}'
```

2.5.1.31. oc 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。

使用例

```
# !!!Important Note!!!
# Requires that the 'tar' binary is present in your container
# image.  If 'tar' is not present, 'oc cp' will fail.
#
# For advanced use cases, such as symlinks, wildcard expansion or
# file mode preservation consider using 'oc exec'.

# Copy /tmp/foo local file to /tmp/bar in a remote pod in namespace <some-namespace>
tar cf - /tmp/foo | oc exec -i -n <some-namespace> <some-pod> -- tar xf - -C /tmp/bar

# Copy /tmp/foo from a remote pod to /tmp/bar locally
oc exec -n <some-namespace> <some-pod> -- tar cf - /tmp/foo | tar xf - -C /tmp/bar

# Copy /tmp/foo_dir local directory to /tmp/bar_dir in a remote pod in the default namespace
oc cp /tmp/foo_dir <some-pod>:/tmp/bar_dir

# Copy /tmp/foo local file to /tmp/bar in a remote pod in a specific container
```

```
oc cp /tmp/foo <some-pod>:/tmp/bar -c <specific-container>

# Copy /tmp/foo local file to /tmp/bar in a remote pod in namespace <some-namespace>
oc cp /tmp/foo <some-namespace>/<some-pod>:/tmp/bar

# Copy /tmp/foo from a remote pod to /tmp/bar locally
oc cp <some-namespace>/<some-pod>:/tmp/foo /tmp/bar
```

2.5.1.32. oc create

ファイルまたは標準入力 (stdin) からリソースを作成します。

使用例

```
# Create a pod using the data in pod.json.
oc create -f ./pod.json

# Create a pod based on the JSON passed into stdin.
cat pod.json | oc create -f -

# Edit the data in docker-registry.yaml in JSON then create the resource using the edited data.
oc create -f docker-registry.yaml --edit -o json
```

2.5.1.33. oc create build

新規ビルドを作成します。

使用例

```
# Create a new build
oc create build myapp
```

2.5.1.34. oc create clusterresourcequota

クラスターリソースクォータを作成します。

使用例

```
# Create a cluster resource quota limited to 10 pods
oc create clusterresourcequota limit-bob --project-annotation-selector=openshift.io/requester=user-bob --hard=pods=10
```

2.5.1.35. oc create clusterrole

ClusterRole を作成します。

使用例

```
# Create a ClusterRole named "pod-reader" that allows user to perform "get", "watch" and "list" on pods
oc create clusterrole pod-reader --verb=get,list,watch --resource=pods
```

```
  # Create a ClusterRole named "pod-reader" with ResourceName specified
  oc create clusterrole pod-reader --verb=get --resource=pods --resource-name=readablepod --
resource-name=anotherpod

  # Create a ClusterRole named "foo" with API Group specified
  oc create clusterrole foo --verb=get,list,watch --resource=rs.extensions

  # Create a ClusterRole named "foo" with SubResource specified
  oc create clusterrole foo --verb=get,list,watch --resource=pods,pods/status

  # Create a ClusterRole name "foo" with NonResourceURL specified
  oc create clusterrole "foo" --verb=get --non-resource-url=/logs/*

  # Create a ClusterRole name "monitoring" with AggregationRule specified
  oc create clusterrole monitoring --aggregation-rule="rbac.example.com/aggregate-to-
monitoring=true"
```

2.5.1.36. oc create clusterrolebinding

特定の ClusterRole の ClusterRoleBinding を作成します。

使用例

```
  # Create a ClusterRoleBinding for user1, user2, and group1 using the cluster-admin ClusterRole
  oc create clusterrolebinding cluster-admin --clusterrole=cluster-admin --user=user1 --user=user2 --
group=group1
```

2.5.1.37. oc create configmap

ローカルファイル、ディレクトリー、またはリテラル値から configmap を作成します。

使用例

```
  # Create a new configmap named my-config based on folder bar
  oc create configmap my-config --from-file=path/to/bar

  # Create a new configmap named my-config with specified keys instead of file basenames on disk
  oc create configmap my-config --from-file=key1=/path/to/bar/file1.txt --from-
file=key2=/path/to/bar/file2.txt

  # Create a new configmap named my-config with key1=config1 and key2=config2
  oc create configmap my-config --from-literal=key1=config1 --from-literal=key2=config2

  # Create a new configmap named my-config from the key=value pairs in the file
  oc create configmap my-config --from-file=path/to/bar

  # Create a new configmap named my-config from an env file
  oc create configmap my-config --from-env-file=path/to/bar.env
```

2.5.1.38. oc create cronjob

指定の名前で cronjob を作成します。

使用例

```
# Create a cronjob
oc create cronjob my-job --image=busybox --schedule="*/1 * * * *"

# Create a cronjob with command
oc create cronjob my-job --image=busybox --schedule="*/1 * * * *" -- date
```

2.5.1.39. oc create deployment

指定の名前のデプロイメントを作成します。

使用例

```
# Create a deployment named my-dep that runs the busybox image.
oc create deployment my-dep --image=busybox

# Create a deployment with command
oc create deployment my-dep --image=busybox -- date

# Create a deployment named my-dep that runs the nginx image with 3 replicas.
oc create deployment my-dep --image=nginx --replicas=3

# Create a deployment named my-dep that runs the busybox image and expose port 5701.
oc create deployment my-dep --image=busybox --port=5701
```

2.5.1.40. oc create deploymentconfig

デフォルトのオプションを指定して特定のイメージを使用するデプロイメント設定を作成します。

使用例

```
# Create an nginx deployment config named my-nginx
oc create deploymentconfig my-nginx --image=nginx
```

2.5.1.41. oc create identity

アイデンティティーを手動で作成します (自動作成が無効になっている場合のみが必要)。

使用例

```
# Create an identity with identity provider "acme_ldap" and the identity provider username
"adamjones"
oc create identity acme_ldap:adamjones
```

2.5.1.42. oc create imagestream

空のイメージストリームを新たに作成します。

使用例

```
# Create a new image stream
oc create imagestream mysql
```

2.5.1.43. oc create imagestreamtag

新規イメージストリームタグを作成します。

使用例

```
# Create a new image stream tag based on an image in a remote registry
oc create imagestreamtag mysql:latest --from-image=myregistry.local/mysql/mysql:5.0
```

2.5.1.44. oc create ingress

指定の名前で Ingress を作成します。

使用例

```
# Create a single ingress called 'simple' that directs requests to foo.com/bar to svc
# svc1:8080 with a tls secret "my-cert"
oc create ingress simple --rule="foo.com/bar=svc1:8080,tls=my-cert"

# Create a catch all ingress of "/path" pointing to service svc:port and Ingress Class as
"otheringress"
oc create ingress catch-all --class=otheringress --rule="/path=svc:port"

# Create an ingress with two annotations: ingress.annotation1 and ingress.annotations2
oc create ingress annotated --class=default --rule="foo.com/bar=svc:port" \
--annotation ingress.annotation1=foo \
--annotation ingress.annotation2=bla

# Create an ingress with the same host and multiple paths
oc create ingress multipath --class=default \
--rule="foo.com/=svc:port" \
--rule="foo.com/admin/=svcadmin:portadmin"

# Create an ingress with multiple hosts and the pathType as Prefix
oc create ingress ingress1 --class=default \
--rule="foo.com/path*=svc:8080" \
--rule="bar.com/admin*=svc2:http"

# Create an ingress with TLS enabled using the default ingress certificate and different path types
oc create ingress ingtls --class=default \
--rule="foo.com/=svc:https,tls" \
--rule="foo.com/path/subpath*=othersvc:8080"

# Create an ingress with TLS enabled using a specific secret and pathType as Prefix
oc create ingress ingsecret --class=default \
--rule="foo.com/*=svc:8080,tls=secret1"

# Create an ingress with a default backend
oc create ingress ingdefault --class=default \
--default-backend=defaultsvc:http \
--rule="foo.com/*=svc:8080,tls=secret1"
```

2.5.1.45. oc create job

指定の名前でジョブを作成します。

使用例

```
# Create a job
oc create job my-job --image=busybox

# Create a job with command
oc create job my-job --image=busybox -- date

# Create a job from a CronJob named "a-cronjob"
oc create job test-job --from=cronjob/a-cronjob
```

2.5.1.46. oc create namespace

指定の名前で namespace を作成します。

使用例

```
# Create a new namespace named my-namespace
oc create namespace my-namespace
```

2.5.1.47. oc create poddisruptionbudget

指定の名前で Pod Disruption Budget (PDB) を作成します。

使用例

```
# Create a pod disruption budget named my-pdb that will select all pods with the app=rails label
# and require at least one of them being available at any point in time.
oc create poddisruptionbudget my-pdb --selector=app=rails --min-available=1

# Create a pod disruption budget named my-pdb that will select all pods with the app=nginx label
# and require at least half of the pods selected to be available at any point in time.
oc create pdb my-pdb --selector=app=nginx --min-available=50%
```

2.5.1.48. oc create priorityclass

指定の名前で priorityclass を作成します。

使用例

```
# Create a priorityclass named high-priority
oc create priorityclass high-priority --value=1000 --description="high priority"

# Create a priorityclass named default-priority that considered as the global default priority
oc create priorityclass default-priority --value=1000 --global-default=true --description="default
priority"

# Create a priorityclass named high-priority that can not preempt pods with lower priority
oc create priorityclass high-priority --value=1000 --description="high priority" --preemption-
policy="Never"
```

2.5.1.49. oc create quota

指定の名前でクォータを作成します。

使用例

```
  # Create a new resourcequota named my-quota
  oc create quota my-quota --
hard=cpu=1,memory=1G,pods=2,services=3,replicationcontrollers=2,resourcequotas=1,secrets=5,persis
tentvolumeclaims=10

  # Create a new resourcequota named best-effort
  oc create quota best-effort --hard=pods=100 --scopes=BestEffort
```

2.5.1.50. oc create role

単一ルールでロールを作成します。

使用例

```
  # Create a Role named "pod-reader" that allows user to perform "get", "watch" and "list" on pods
  oc create role pod-reader --verb=get --verb=list --verb=watch --resource=pods

  # Create a Role named "pod-reader" with ResourceName specified
  oc create role pod-reader --verb=get --resource=pods --resource-name=readablepod --resource-
name=anotherpod

  # Create a Role named "foo" with API Group specified
  oc create role foo --verb=get,list,watch --resource=rs.extensions

  # Create a Role named "foo" with SubResource specified
  oc create role foo --verb=get,list,watch --resource=pods,pods/status
```

2.5.1.51. oc create rolebinding

特定のロールまたは ClusterRole の RoleBinding を作成します。

使用例

```
  # Create a RoleBinding for user1, user2, and group1 using the admin ClusterRole
  oc create rolebinding admin --clusterrole=admin --user=user1 --user=user2 --group=group1
```

2.5.1.52. oc create route edge

edge TLS termination を使用するルートを作成します。

使用例

```
  # Create an edge route named "my-route" that exposes the frontend service
  oc create route edge my-route --service=frontend
```

```
# Create an edge route that exposes the frontend service and specify a path
# If the route name is omitted, the service name will be used
oc create route edge --service=frontend --path /assets
```

2.5.1.53. oc create route passthrough

passthrough TLS 終端を使用するルートを作成します。

使用例

```
# Create a passthrough route named "my-route" that exposes the frontend service
oc create route passthrough my-route --service=frontend

# Create a passthrough route that exposes the frontend service and specify
# a host name. If the route name is omitted, the service name will be used
oc create route passthrough --service=frontend --hostname=www.example.com
```

2.5.1.54. oc create route reencrypt

re-encrypt TLS 終端を使用するルートを作成します。

使用例

```
# Create a route named "my-route" that exposes the frontend service
oc create route reencrypt my-route --service=frontend --dest-ca-cert cert.cert

# Create a reencrypt route that exposes the frontend service, letting the
# route name default to the service name and the destination CA certificate
# default to the service CA
oc create route reencrypt --service=frontend
```

2.5.1.55. oc create secret docker-registry

Docker レジストリーで使用するシークレットを作成します。

使用例

```
# If you don't already have a .dockercfg file, you can create a dockercfg secret directly by using:
oc create secret docker-registry my-secret --docker-server=DOCKER_REGISTRY_SERVER --
docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-
email=DOCKER_EMAIL

# Create a new secret named my-secret from ~/.docker/config.json
oc create secret docker-registry my-secret --from-file=.dockerconfigjson=path/to/.docker/config.json
```

2.5.1.56. oc create secret generic

ローカルファイル、ディレクトリー、またはリテラル値からシークレットを作成します。

使用例

```
# Create a new secret named my-secret with keys for each file in folder bar
```

```
  oc create secret generic my-secret --from-file=path/to/bar

  # Create a new secret named my-secret with specified keys instead of names on disk
  oc create secret generic my-secret --from-file=ssh-privatekey=path/to/id_rsa --from-file=ssh-
publickey=path/to/id_rsa.pub

  # Create a new secret named my-secret with key1=supersecret and key2=topsecret
  oc create secret generic my-secret --from-literal=key1=supersecret --from-literal=key2=topsecret

  # Create a new secret named my-secret using a combination of a file and a literal
  oc create secret generic my-secret --from-file=ssh-privatekey=path/to/id_rsa --from-
literal=passphrase=topsecret

  # Create a new secret named my-secret from an env file
  oc create secret generic my-secret --from-env-file=path/to/bar.env
```

2.5.1.57. oc create secret tls

TLS シークレットを作成します。

使用例

```
  # Create a new TLS secret named tls-secret with the given key pair:
  oc create secret tls tls-secret --cert=path/to/tls.cert --key=path/to/tls.key
```

2.5.1.58. oc create service clusterip

ClusterIP サービスを作成します。

使用例

```
  # Create a new ClusterIP service named my-cs
  oc create service clusterip my-cs --tcp=5678:8080

  # Create a new ClusterIP service named my-cs (in headless mode)
  oc create service clusterip my-cs --clusterip="None"
```

2.5.1.59. oc create service externalname

ExternalName サービスを作成します。

使用例

```
  # Create a new ExternalName service named my-ns
  oc create service externalname my-ns --external-name bar.com
```

2.5.1.60. oc create service loadbalancer

Pod に LoadBalancer サービスを作成します。

使用例

```
# Create a new LoadBalancer service named my-lbs
oc create service loadbalancer my-lbs --tcp=5678:8080
```

2.5.1.61. oc create service nodeport

NodePort サービスを作成します。

使用例

```
# Create a new NodePort service named my-ns
oc create service nodeport my-ns --tcp=5678:8080
```

2.5.1.62. oc create serviceaccount

指定の名前でサービスアカウントを作成します。

使用例

```
# Create a new service account named my-service-account
oc create serviceaccount my-service-account
```

2.5.1.63. oc create user

ユーザーを手動で作成します (自動作成が無効になっている場合のみ必要)。

使用例

```
# Create a user with the username "ajones" and the display name "Adam Jones"
oc create user ajones --full-name="Adam Jones"
```

2.5.1.64. oc create useridentitymapping

アイデンティティーをユーザーに手動でマップします。

使用例

```
# Map the identity "acme_ldap:adamjones" to the user "ajones"
oc create useridentitymapping acme_ldap:adamjones ajones
```

2.5.1.65. oc debug

デバッグ用に Pod の新規インスタンスを起動します。

使用例

```
# Start a shell session into a pod using the OpenShift tools image
oc debug

# Debug a currently running deployment by creating a new pod
oc debug deploy/test
```

```
  # Debug a node as an administrator
  oc debug node/master-1

  # Launch a shell in a pod using the provided image stream tag
  oc debug istag/mysql:latest -n openshift

  # Test running a job as a non-root user
  oc debug job/test --as-user=1000000

  # Debug a specific failing container by running the env command in the 'second' container
  oc debug daemonset/test -c second -- /bin/env

  # See the pod that would be created to debug
  oc debug mypod-9xbc -o yaml

  # Debug a resource but launch the debug pod in another namespace
  # Note: Not all resources can be debugged using --to-namespace without modification. For example,
  # volumes and service accounts are namespace-dependent. Add '-o yaml' to output the debug pod definition
  # to disk.  If necessary, edit the definition then run 'oc debug -f -' or run without --to-namespace
  oc debug mypod-9xbc --to-namespace testns
```

2.5.1.66. oc delete

ファイル名、stdin、リソースおよび名前、またはリソースおよびラベルセレクター別にリソースを削除します。

使用例

```
  # Delete a pod using the type and name specified in pod.json.
  oc delete -f ./pod.json

  # Delete resources from a directory containing kustomization.yaml - e.g. dir/kustomization.yaml.
  oc delete -k dir

  # Delete a pod based on the type and name in the JSON passed into stdin.
  cat pod.json | oc delete -f -

  # Delete pods and services with same names "baz" and "foo"
  oc delete pod,service baz foo

  # Delete pods and services with label name=myLabel.
  oc delete pods,services -l name=myLabel

  # Delete a pod with minimal delay
  oc delete pod foo --now

  # Force delete a pod on a dead node
  oc delete pod foo --force

  # Delete all pods
  oc delete pods --all
```

2.5.1.67. oc describe

特定のリソースまたはリソースのグループの詳細を表示します。

使用例

```
# Describe a node
oc describe nodes kubernetes-node-emt8.c.myproject.internal

# Describe a pod
oc describe pods/nginx

# Describe a pod identified by type and name in "pod.json"
oc describe -f pod.json

# Describe all pods
oc describe pods

# Describe pods by label name=myLabel
oc describe po -l name=myLabel

# Describe all pods managed by the 'frontend' replication controller (rc-created pods
# get the name of the rc as a prefix in the pod the name).
oc describe pods frontend
```

2.5.1.68. oc diff

ライブバーションと適用バージョンとの差異を確認します。

使用例

```
# Diff resources included in pod.json.
oc diff -f pod.json

# Diff file read from stdin
cat service.yaml | oc diff -f -
```

2.5.1.69. oc edit

サーバーのリソースを編集します。

使用例

```
# Edit the service named 'docker-registry':
oc edit svc/docker-registry

# Use an alternative editor
KUBE_EDITOR="nano" oc edit svc/docker-registry

# Edit the job 'myjob' in JSON using the v1 API format:
oc edit job.v1.batch/myjob -o json

# Edit the deployment 'mydeployment' in YAML and save the modified config in its annotation:
oc edit deployment/mydeployment -o yaml --save-config
```

2.5.1.70. oc ex dockergc

Docker ストレージで領域を解放するためにガベージコレクションを実行します。

使用例

```
# Perform garbage collection with the default settings
oc ex dockergc
```

2.5.1.71. oc exec

コンテナーでコマンドを実行します。

使用例

```
# Get output from running 'date' command from pod mypod, using the first container by default
oc exec mypod -- date

# Get output from running 'date' command in ruby-container from pod mypod
oc exec mypod -c ruby-container -- date

# Switch to raw terminal mode, sends stdin to 'bash' in ruby-container from pod mypod
# and sends stdout/stderr from 'bash' back to the client
oc exec mypod -c ruby-container -i -t -- bash -il

# List contents of /usr from the first container of pod mypod and sort by modification time.
# If the command you want to execute in the pod has any flags in common (e.g. -i),
# you must use two dashes (--) to separate your command's flags/arguments.
# Also note, do not surround your command and its flags/arguments with quotes
# unless that is how you would execute it normally (i.e., do ls -t /usr, not "ls -t /usr").
oc exec mypod -i -t -- ls -t /usr

# Get output from running 'date' command from the first pod of the deployment mydeployment,
using the first container by default
oc exec deploy/mydeployment -- date

# Get output from running 'date' command from the first pod of the service myservice, using the first
container by default
oc exec svc/myservice -- date
```

2.5.1.72. oc explain

リソースのドキュメントを取得します。

使用例

```
# Get the documentation of the resource and its fields
oc explain pods

# Get the documentation of a specific field of a resource
oc explain pods.spec.containers
```

2.5.1.73. oc expose

複製されたアプリケーションをサービスまたはルートとして公開します。

使用例

```
  # Create a route based on service nginx. The new route will reuse nginx's labels
  oc expose service nginx

  # Create a route and specify your own label and route name
  oc expose service nginx -l name=myroute --name=fromdowntown

  # Create a route and specify a host name
  oc expose service nginx --hostname=www.example.com

  # Create a route with a wildcard
  oc expose service nginx --hostname=x.example.com --wildcard-policy=Subdomain
  # This would be equivalent to *.example.com. NOTE: only hosts are matched by the wildcard;
subdomains would not be included

  # Expose a deployment configuration as a service and use the specified port
  oc expose dc ruby-hello-world --port=8080

  # Expose a service as a route in the specified path
  oc expose service nginx --path=/nginx

  # Expose a service using different generators
  oc expose service nginx --name=exposed-svc --port=12201 --protocol="TCP" --
generator="service/v2"
  oc expose service nginx --name=my-route --port=12201 --generator="route/v1"

  # Exposing a service using the "route/v1" generator (default) will create a new exposed route with
the "--name" provided
  # (or the name of the service otherwise). You may not specify a "--protocol" or "--target-port" option
when using this generator
```

2.5.1.74. oc extract

シークレットまたは設定マップをディスクに抽出します。

使用例

```
  # Extract the secret "test" to the current directory
  oc extract secret/test

  # Extract the config map "nginx" to the /tmp directory
  oc extract configmap/nginx --to=/tmp

  # Extract the config map "nginx" to STDOUT
  oc extract configmap/nginx --to=-

  # Extract only the key "nginx.conf" from config map "nginx" to the /tmp directory
  oc extract configmap/nginx --to=/tmp --keys=nginx.conf
```

2.5.1.75. oc get

1 つ以上のリソースを表示します。

使用例

```
# List all pods in ps output format.
oc get pods

# List all pods in ps output format with more information (such as node name).
oc get pods -o wide

# List a single replication controller with specified NAME in ps output format.
oc get replicationcontroller web

# List deployments in JSON output format, in the "v1" version of the "apps" API group:
oc get deployments.v1.apps -o json

# List a single pod in JSON output format.
oc get -o json pod web-pod-13je7

# List a pod identified by type and name specified in "pod.yaml" in JSON output format.
oc get -f pod.yaml -o json

# List resources from a directory with kustomization.yaml - e.g. dir/kustomization.yaml.
oc get -k dir/

# Return only the phase value of the specified pod.
oc get -o template pod/web-pod-13je7 --template={{.status.phase}}

# List resource information in custom columns.
oc get pod test-pod -o custom-
columns=CONTAINER:.spec.containers[0].name,IMAGE:.spec.containers[0].image

# List all replication controllers and services together in ps output format.
oc get rc,services

# List one or more resources by their type and names.
oc get rc/web service/frontend pods/web-pod-13je7
```

2.5.1.76. oc idle

スケーラブルなリソースをアイドリングします。

使用例

```
# Idle the scalable controllers associated with the services listed in to-idle.txt
$ oc idle --resource-names-file to-idle.txt
```

2.5.1.77. oc image append

イメージにレイヤーを追加してレジストリーにプッシュします。

使用例

```
# Remove the entrypoint on the mysql:latest image
```

```
  oc image append --from mysql:latest --to myregistry.com/myimage:latest --image '{"Entrypoint":null}'

  # Add a new layer to the image
  oc image append --from mysql:latest --to myregistry.com/myimage:latest layer.tar.gz

  # Add a new layer to the image and store the result on disk
  # This results in $(pwd)/v2/mysql/blobs,manifests
  oc image append --from mysql:latest --to file://mysql:local layer.tar.gz

  # Add a new layer to the image and store the result on disk in a designated directory
  # This will result in $(pwd)/mysql-local/v2/mysql/blobs,manifests
  oc image append --from mysql:latest --to file://mysql:local --dir mysql-local layer.tar.gz

  # Add a new layer to an image that is stored on disk (~/mysql-local/v2/image exists)
  oc image append --from-dir ~/mysql-local --to myregistry.com/myimage:latest layer.tar.gz

  # Add a new layer to an image that was mirrored to the current directory on disk ($(pwd)/v2/image
exists)
  oc image append --from-dir v2 --to myregistry.com/myimage:latest layer.tar.gz

  # Add a new layer to a multi-architecture image for an os/arch that is different from the system's
os/arch
  # Note: Wildcard filter is not supported with append. Pass a single os/arch to append
  oc image append --from docker.io/library/busybox:latest --filter-by-os=linux/s390x --to
myregistry.com/myimage:latest layer.tar.gz
```

2.5.1.78. oc image extract

イメージからファイルシステムにファイルをコピーします。

使用例

```
  # Extract the busybox image into the current directory
  oc image extract docker.io/library/busybox:latest

  # Extract the busybox image into a designated directory (must exist)
  oc image extract docker.io/library/busybox:latest --path /:/tmp/busybox

  # Extract the busybox image into the current directory for linux/s390x platform
  # Note: Wildcard filter is not supported with extract. Pass a single os/arch to extract
  oc image extract docker.io/library/busybox:latest --filter-by-os=linux/s390x

  # Extract a single file from the image into the current directory
  oc image extract docker.io/library/centos:7 --path /bin/bash:.

  # Extract all .repo files from the image's /etc/yum.repos.d/ folder into the current directory
  oc image extract docker.io/library/centos:7 --path /etc/yum.repos.d/*.repo:.

  # Extract all .repo files from the image's /etc/yum.repos.d/ folder into a designated directory (must
exist)
  # This results in /tmp/yum.repos.d/*.repo on local system
  oc image extract docker.io/library/centos:7 --path /etc/yum.repos.d/*.repo:/tmp/yum.repos.d

  # Extract an image stored on disk into the current directory ($(pwd)/v2/busybox/blobs,manifests
exists)
```

```
  # --confirm is required because the current directory is not empty
  oc image extract file://busybox:local --confirm

  # Extract an image stored on disk in a directory other than $(pwd)/v2 into the current directory
  # --confirm is required because the current directory is not empty ($(pwd)/busybox-mirror-
dir/v2/busybox exists)
  oc image extract file://busybox:local --dir busybox-mirror-dir --confirm

  # Extract an image stored on disk in a directory other than $(pwd)/v2 into a designated directory
(must exist)
  oc image extract file://busybox:local --dir busybox-mirror-dir --path /:/tmp/busybox

  # Extract the last layer in the image
  oc image extract docker.io/library/centos:7[-1]

  # Extract the first three layers of the image
  oc image extract docker.io/library/centos:7[:3]

  # Extract the last three layers of the image
  oc image extract docker.io/library/centos:7[-3:]
```

2.5.1.79. oc image info

イメージに関する情報を表示します。

使用例

```
  # Show information about an image
  oc image info quay.io/openshift/cli:latest

  # Show information about images matching a wildcard
  oc image info quay.io/openshift/cli:4.*

  # Show information about a file mirrored to disk under DIR
  oc image info --dir=DIR file://library/busybox:latest

  # Select which image from a multi-OS image to show
  oc image info library/busybox:latest --filter-by-os=linux/arm64
```

2.5.1.80. oc image mirror

別のリポジトリーにイメージをミラーリングします。

使用例

```
  # Copy image to another tag
  oc image mirror myregistry.com/myimage:latest myregistry.com/myimage:stable

  # Copy image to another registry
  oc image mirror myregistry.com/myimage:latest docker.io/myrepository/myimage:stable

  # Copy all tags starting with mysql to the destination repository
  oc image mirror myregistry.com/myimage:mysql* docker.io/myrepository/myimage
```

```
  # Copy image to disk, creating a directory structure that can be served as a registry
  oc image mirror myregistry.com/myimage:latest file://myrepository/myimage:latest

  # Copy image to S3 (pull from <bucket>.s3.amazonaws.com/image:latest)
  oc image mirror myregistry.com/myimage:latest
s3://s3.amazonaws.com/<region>/<bucket>/image:latest

  # Copy image to S3 without setting a tag (pull via @<digest>)
  oc image mirror myregistry.com/myimage:latest s3://s3.amazonaws.com/<region>/<bucket>/image

  # Copy image to multiple locations
  oc image mirror myregistry.com/myimage:latest docker.io/myrepository/myimage:stable \
  docker.io/myrepository/myimage:dev

  # Copy multiple images
  oc image mirror myregistry.com/myimage:latest=myregistry.com/other:test \
  myregistry.com/myimage:new=myregistry.com/other:target

  # Copy manifest list of a multi-architecture image, even if only a single image is found
  oc image mirror myregistry.com/myimage:latest=myregistry.com/other:test \
  --keep-manifest-list=true

  # Copy specific os/arch manifest of a multi-architecture image
  # Run 'oc image info myregistry.com/myimage:latest' to see available os/arch for multi-arch images
  # Note that with multi-arch images, this results in a new manifest list digest that includes only
  # the filtered manifests
  oc image mirror myregistry.com/myimage:latest=myregistry.com/other:test \
  --filter-by-os=os/arch

  # Copy all os/arch manifests of a multi-architecture image
  # Run 'oc image info myregistry.com/myimage:latest' to see list of os/arch manifests that will be
mirrored
  oc image mirror myregistry.com/myimage:latest=myregistry.com/other:test \
  --keep-manifest-list=true

  # Note the above command is equivalent to
  oc image mirror myregistry.com/myimage:latest=myregistry.com/other:test \
  --filter-by-os=.*
```

2.5.1.81. oc import-image

コンテナーイメージレジストリーからイメージをインポートします。

使用例

```
  # Import tag latest into a new image stream
  oc import-image mystream --from=registry.io/repo/image:latest --confirm

  # Update imported data for tag latest in an already existing image stream
  oc import-image mystream

  # Update imported data for tag stable in an already existing image stream
  oc import-image mystream:stable

  # Update imported data for all tags in an existing image stream
```

```
oc import-image mystream --all

# Import all tags into a new image stream
oc import-image mystream --from=registry.io/repo/image --all --confirm

# Import all tags into a new image stream using a custom timeout
oc --request-timeout=5m import-image mystream --from=registry.io/repo/image --all --confirm
```

2.5.1.82. oc kustomize

ディレクトリーまたは URL から kustomization ターゲットをビルドします。

使用例

```
# Build the current working directory
oc kustomize

# Build some shared configuration directory
oc kustomize /home/config/production

# Build from github
oc kustomize https://github.com/kubernetes-sigs/kustomize.git/examples/helloWorld?ref=v1.0.6
```

2.5.1.83. oc label

リソースのラベルを更新します。

使用例

```
# Update pod 'foo' with the label 'unhealthy' and the value 'true'.
oc label pods foo unhealthy=true

# Update pod 'foo' with the label 'status' and the value 'unhealthy', overwriting any existing value.
oc label --overwrite pods foo status=unhealthy

# Update all pods in the namespace
oc label pods --all status=unhealthy

# Update a pod identified by the type and name in "pod.json"
oc label -f pod.json status=unhealthy

# Update pod 'foo' only if the resource is unchanged from version 1.
oc label pods foo status=unhealthy --resource-version=1

# Update pod 'foo' by removing a label named 'bar' if it exists.
# Does not require the --overwrite flag.
oc label pods foo bar-
```

2.5.1.84. oc login

サーバーにログインします。

使用例

```
# Log in interactively
oc login --username=myuser

# Log in to the given server with the given certificate authority file
oc login localhost:8443 --certificate-authority=/path/to/cert.crt

# Log in to the given server with the given credentials (will not prompt interactively)
oc login localhost:8443 --username=myuser --password=mypass
```

2.5.1.85. oc logout

現在のサーバーセッションを終了します。

使用例

```
# Log out
oc logout
```

2.5.1.86. oc logs

Pod 内のコンテナーのログを出力します。

使用例

```
# Start streaming the logs of the most recent build of the openldap build config
oc logs -f bc/openldap

# Start streaming the logs of the latest deployment of the mysql deployment config
oc logs -f dc/mysql

# Get the logs of the first deployment for the mysql deployment config. Note that logs
# from older deployments may not exist either because the deployment was successful
# or due to deployment pruning or manual deletion of the deployment
oc logs --version=1 dc/mysql

# Return a snapshot of ruby-container logs from pod backend
oc logs backend -c ruby-container

# Start streaming of ruby-container logs from pod backend
oc logs -f pod/backend -c ruby-container
```

2.5.1.87. oc new-app

新規アプリケーションを作成します。

使用例

```
# List all local templates and image streams that can be used to create an app
oc new-app --list

# Create an application based on the source code in the current git repository (with a public remote)
and a Docker image
oc new-app . --docker-image=registry/repo/langimage
```

```
  # Create an application myapp with Docker based build strategy expecting binary input
  oc new-app  --strategy=docker --binary --name myapp

  # Create a Ruby application based on the provided [image]~[source code] combination
  oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git

  # Use the public Docker Hub MySQL image to create an app. Generated artifacts will be labeled
with db=mysql
  oc new-app mysql MYSQL_USER=user MYSQL_PASSWORD=pass MYSQL_DATABASE=testdb -
l db=mysql

  # Use a MySQL image in a private registry to create an app and override application artifacts'
names
  oc new-app --docker-image=myregistry.com/mycompany/mysql --name=private

  # Create an application from a remote repository using its beta4 branch
  oc new-app https://github.com/openshift/ruby-hello-world#beta4

  # Create an application based on a stored template, explicitly setting a parameter value
  oc new-app --template=ruby-helloworld-sample --param=MYSQL_USER=admin

  # Create an application from a remote repository and specify a context directory
  oc new-app https://github.com/youruser/yourgitrepo --context-dir=src/build

  # Create an application from a remote private repository and specify which existing secret to use
  oc new-app https://github.com/youruser/yourgitrepo --source-secret=yoursecret

  # Create an application based on a template file, explicitly setting a parameter value
  oc new-app --file=./example/myapp/template.json --param=MYSQL_USER=admin

  # Search all templates, image streams, and Docker images for the ones that match "ruby"
  oc new-app --search ruby

  # Search for "ruby", but only in stored templates (--template, --image-stream and --docker-image
  # can be used to filter search results)
  oc new-app --search --template=ruby

  # Search for "ruby" in stored templates and print the output as YAML
  oc new-app --search --template=ruby --output=yaml
```

2.5.1.88. oc new-build

新規ビルド設定を作成します。

使用例

```
  # Create a build config based on the source code in the current git repository (with a public
  # remote) and a Docker image
  oc new-build . --docker-image=repo/langimage

  # Create a NodeJS build config based on the provided [image]~[source code] combination
  oc new-build centos/nodejs-8-centos7~https://github.com/sclorg/nodejs-ex.git

  # Create a build config from a remote repository using its beta2 branch
```

```
  oc new-build https://github.com/openshift/ruby-hello-world#beta2

  # Create a build config using a Dockerfile specified as an argument
  oc new-build -D $'FROM centos:7\nRUN yum install -y httpd'

  # Create a build config from a remote repository and add custom environment variables
  oc new-build https://github.com/openshift/ruby-hello-world -e RACK_ENV=development

  # Create a build config from a remote private repository and specify which existing secret to use
  oc new-build https://github.com/youruser/yourgitrepo --source-secret=yoursecret

  # Create a build config from a remote repository and inject the npmrc into a build
  oc new-build https://github.com/openshift/ruby-hello-world --build-secret npmrc:.npmrc

  # Create a build config from a remote repository and inject environment data into a build
  oc new-build https://github.com/openshift/ruby-hello-world --build-config-map env:config

  # Create a build config that gets its input from a remote repository and another Docker image
  oc new-build https://github.com/openshift/ruby-hello-world --source-image=openshift/jenkins-1-
centos7 --source-image-path=/var/lib/jenkins:tmp
```

2.5.1.89. oc new-project

新規プロジェクトを要求します。

使用例

```
  # Create a new project with minimal information
  oc new-project web-team-dev

  # Create a new project with a display name and description
  oc new-project web-team-dev --display-name="Web Team Development" --
description="Development project for the web team."
```

2.5.1.90. oc observe

リソースの変更を確認し、リソースに対応します (実験的)。

使用例

```
  # Observe changes to services
  oc observe services

  # Observe changes to services, including the clusterIP and invoke a script for each
  oc observe services --template '{ .spec.clusterIP }' -- register_dns.sh

  # Observe changes to services filtered by a label selector
  oc observe namespaces -l regist-dns=true --template '{ .spec.clusterIP }' -- register_dns.sh
```

2.5.1.91. oc patch

リソースのフィールドを更新します。

使用例

```
  # Partially update a node using a strategic merge patch. Specify the patch as JSON.
  oc patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'

  # Partially update a node using a strategic merge patch. Specify the patch as YAML.
  oc patch node k8s-node-1 -p $'spec:\n unschedulable: true'

  # Partially update a node identified by the type and name specified in "node.json" using strategic
merge patch.
  oc patch -f node.json -p '{"spec":{"unschedulable":true}}'

  # Update a container's image; spec.containers[*].name is required because it's a merge key.
  oc patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-serve-
hostname","image":"new image"}]}}'

  # Update a container's image using a json patch with positional arrays.
  oc patch pod valid-pod --type='json' -p='[{"op": "replace", "path": "/spec/containers/0/image",
"value":"new image"}]'
```

2.5.1.92. oc policy add-role-to-user

現在のプロジェクトのユーザーまたはサービスアカウントをロールに追加します。

使用例

```
  # Add the 'view' role to user1 for the current project
  oc policy add-role-to-user view user1

  # Add the 'edit' role to serviceaccount1 for the current project
  oc policy add-role-to-user edit -z serviceaccount1
```

2.5.1.93. oc policy scc-review

Pod を作成できるサービスアカウントを確認します。

使用例

```
  # Check whether service accounts sa1 and sa2 can admit a pod with a template pod spec specified
in my_resource.yaml
  # Service Account specified in myresource.yaml file is ignored
  oc policy scc-review -z sa1,sa2 -f my_resource.yaml

  # Check whether service accounts system:serviceaccount:bob:default can admit a pod with a
template pod spec specified in my_resource.yaml
  oc policy scc-review -z system:serviceaccount:bob:default -f my_resource.yaml

  # Check whether the service account specified in my_resource_with_sa.yaml can admit the pod
  oc policy scc-review -f my_resource_with_sa.yaml

  # Check whether the default service account can admit the pod; default is taken since no service
account is defined in myresource_with_no_sa.yaml
  oc policy scc-review -f myresource_with_no_sa.yaml
```

2.5.1.94. oc policy scc-subject-review

ユーザーまたはサービスアカウントが Pod を作成できるかどうかを確認します。

使用例

```
  # Check whether user bob can create a pod specified in myresource.yaml
  oc policy scc-subject-review -u bob -f myresource.yaml

  # Check whether user bob who belongs to projectAdmin group can create a pod specified in
myresource.yaml
  oc policy scc-subject-review -u bob -g projectAdmin -f myresource.yaml

  # Check whether a service account specified in the pod template spec in myresourcewithsa.yaml
can create the pod
  oc policy scc-subject-review -f myresourcewithsa.yaml
```

2.5.1.95. oc port-forward

1 つ以上のローカルポートを Pod に転送します。

使用例

```
  # Listen on ports 5000 and 6000 locally, forwarding data to/from ports 5000 and 6000 in the pod
  oc port-forward pod/mypod 5000 6000

  # Listen on ports 5000 and 6000 locally, forwarding data to/from ports 5000 and 6000 in a pod
selected by the deployment
  oc port-forward deployment/mydeployment 5000 6000

  # Listen on port 8443 locally, forwarding to the targetPort of the service's port named "https" in a pod
selected by the service
  oc port-forward service/myservice 8443:https

  # Listen on port 8888 locally, forwarding to 5000 in the pod
  oc port-forward pod/mypod 8888:5000

  # Listen on port 8888 on all addresses, forwarding to 5000 in the pod
  oc port-forward --address 0.0.0.0 pod/mypod 8888:5000

  # Listen on port 8888 on localhost and selected IP, forwarding to 5000 in the pod
  oc port-forward --address localhost,10.19.21.23 pod/mypod 8888:5000

  # Listen on a random port locally, forwarding to 5000 in the pod
  oc port-forward pod/mypod :5000
```

2.5.1.96. oc process

リソースの一覧に対してテンプレートを処理します。

使用例

```
  # Convert the template.json file into a resource list and pass to create
  oc process -f template.json | oc create -f -

  # Process a file locally instead of contacting the server
```

```
oc process -f template.json --local -o yaml

# Process template while passing a user-defined label
oc process -f template.json -l name=mytemplate

# Convert a stored template into a resource list
oc process foo

# Convert a stored template into a resource list by setting/overriding parameter values
oc process foo PARM1=VALUE1 PARM2=VALUE2

# Convert a template stored in different namespace into a resource list
oc process openshift//foo

# Convert template.json into a resource list
cat template.json | oc process -f -
```

2.5.1.97. oc project

別のプロジェクトに切り替えます。

使用例

```
# Switch to the 'myapp' project
oc project myapp

# Display the project currently in use
oc project
```

2.5.1.98. oc projects

既存プロジェクトを表示します。

使用例

```
# List all projects
oc projects
```

2.5.1.99. oc proxy

Kubernetes API サーバーに対してプロキシーを実行します。

使用例

```
# To proxy all of the kubernetes api and nothing else.
oc proxy --api-prefix=/

# To proxy only part of the kubernetes api and also some static files.
# You can get pods info with 'curl localhost:8001/api/v1/pods'
oc proxy --www=/my/files --www-prefix=/static/ --api-prefix=/api/

# To proxy the entire kubernetes api at a different root.
# You can get pods info with 'curl localhost:8001/custom/api/v1/pods'
```

```
oc proxy --api-prefix=/custom/

# Run a proxy to kubernetes apiserver on port 8011, serving static content from ./local/www/
oc proxy --port=8011 --www=./local/www/

# Run a proxy to kubernetes apiserver on an arbitrary local port.
# The chosen port for the server will be output to stdout.
oc proxy --port=0

# Run a proxy to kubernetes apiserver, changing the api prefix to k8s-api
# This makes e.g. the pods api available at localhost:8001/k8s-api/v1/pods/
oc proxy --api-prefix=/k8s-api
```

2.5.1.100. oc registry info

統合レジストリーについての情報を表示します。

使用例

```
# Display information about the integrated registry
oc registry info
```

2.5.1.101. oc registry login

統合レジストリーにログインします。

使用例

```
# Log in to the integrated registry
oc registry login

# Log in as the default service account in the current namespace
oc registry login -z default

# Log in to different registry using BASIC auth credentials
oc registry login --registry quay.io/myregistry --auth-basic=USER:PASS
```

2.5.1.102. oc replace

リソースをファイル名または stdin に置き換えます。

使用例

```
# Replace a pod using the data in pod.json.
oc replace -f ./pod.json

# Replace a pod based on the JSON passed into stdin.
cat pod.json | oc replace -f -

# Update a single-container pod's image version (tag) to v4
oc get pod mypod -o yaml | sed 's/\(image: myimage\):.*$/\1:v4/' | oc replace -f -
```

```
# Force replace, delete and then re-create the resource
oc replace --force -f ./pod.json
```

2.5.1.103. oc rollback

アプリケーションの一部を以前のデプロイメントに戻します。

使用例

```
# Perform a rollback to the last successfully completed deployment for a deployment config
oc rollback frontend

# See what a rollback to version 3 will look like, but do not perform the rollback
oc rollback frontend --to-version=3 --dry-run

# Perform a rollback to a specific deployment
oc rollback frontend-2

# Perform the rollback manually by piping the JSON of the new config back to oc
oc rollback frontend -o json | oc replace dc/frontend -f -

# Print the updated deployment configuration in JSON format instead of performing the rollback
oc rollback frontend -o json
```

2.5.1.104. oc rollout cancel

進行中のデプロイメントをキャンセルします。

使用例

```
# Cancel the in-progress deployment based on 'nginx'
oc rollout cancel dc/nginx
```

2.5.1.105. oc rollout history

ロールアウト履歴を表示します。

使用例

```
# View the rollout history of a deployment
oc rollout history dc/nginx

# View the details of deployment revision 3
oc rollout history dc/nginx --revision=3
```

2.5.1.106. oc rollout latest

トリガーからの最新状態を使用して、デプロイメント設定の新規ロールアウトを開始します。

使用例

```
# Start a new rollout based on the latest images defined in the image change triggers
oc rollout latest dc/nginx

# Print the rolled out deployment config
oc rollout latest dc/nginx -o json
```

2.5.1.107. oc rollout pause

提供されたリソースを一時停止としてマークします。

使用例

```
# Mark the nginx deployment as paused. Any current state of
# the deployment will continue its function, new updates to the deployment will not
# have an effect as long as the deployment is paused
oc rollout pause dc/nginx
```

2.5.1.108. oc rollout restart

リソースを再起動します。

使用例

```
# Restart a deployment
oc rollout restart deployment/nginx

# Restart a daemonset
oc rollout restart daemonset/abc
```

2.5.1.109. oc rollout resume

一時停止したリソースを再開します。

使用例

```
# Resume an already paused deployment
oc rollout resume dc/nginx
```

2.5.1.110. oc rollout retry

失敗したロールアウトを再試行します。

使用例

```
# Retry the latest failed deployment based on 'frontend'
# The deployer pod and any hook pods are deleted for the latest failed deployment
oc rollout retry dc/frontend
```

2.5.1.111. oc rollout status

ロールアウトのステータスを表示します。

使用例

```
# Watch the status of the latest rollout
oc rollout status dc/nginx
```

2.5.1.112. oc rollout undo

以前のロールアウトを元に戻します。

使用例

```
# Roll back to the previous deployment
oc rollout undo dc/nginx

# Roll back to deployment revision 3. The replication controller for that version must exist
oc rollout undo dc/nginx --to-revision=3
```

2.5.1.113. oc rsh

コンテナーでシェルセッションを開始します。

使用例

```
# Open a shell session on the first container in pod 'foo'
oc rsh foo

# Open a shell session on the first container in pod 'foo' and namespace 'bar'
# (Note that oc client specific arguments must come before the resource name and its arguments)
oc rsh -n bar foo

# Run the command 'cat /etc/resolv.conf' inside pod 'foo'
oc rsh foo cat /etc/resolv.conf

# See the configuration of your internal registry
oc rsh dc/docker-registry cat config.yml

# Open a shell session on the container named 'index' inside a pod of your job
oc rsh -c index job/sheduled
```

2.5.1.114. oc rsync

ローカルファイルシステムと Pod 間でファイルをコピーします。

使用例

```
# Synchronize a local directory with a pod directory
oc rsync ./local/dir/ POD:/remote/dir

# Synchronize a pod directory with a local directory
oc rsync POD:/remote/dir/ ./local/dir
```

2.5.1.115. oc run

クラスターで特定のイメージを実行します。

使用例

```
  # Start a nginx pod.
  oc run nginx --image=nginx

  # Start a hazelcast pod and let the container expose port 5701.
  oc run hazelcast --image=hazelcast/hazelcast --port=5701

  # Start a hazelcast pod and set environment variables "DNS_DOMAIN=cluster" and
"POD_NAMESPACE=default" in the container.
  oc run hazelcast --image=hazelcast/hazelcast --env="DNS_DOMAIN=cluster" --
env="POD_NAMESPACE=default"

  # Start a hazelcast pod and set labels "app=hazelcast" and "env=prod" in the container.
  oc run hazelcast --image=hazelcast/hazelcast --labels="app=hazelcast,env=prod"

  # Dry run. Print the corresponding API objects without creating them.
  oc run nginx --image=nginx --dry-run=client

  # Start a nginx pod, but overload the spec with a partial set of values parsed from JSON.
  oc run nginx --image=nginx --overrides='{ "apiVersion": "v1", "spec": { ... } }'

  # Start a busybox pod and keep it in the foreground, don't restart it if it exits.
  oc run -i -t busybox --image=busybox --restart=Never

  # Start the nginx pod using the default command, but use custom arguments (arg1 .. argN) for that
command.
  oc run nginx --image=nginx -- <arg1> <arg2> ... <argN>

  # Start the nginx pod using a different command and custom arguments.
  oc run nginx --image=nginx --command -- <cmd> <arg1> ... <argN>
```

2.5.1.116. oc scale

Deployment、ReplicaSet、または Replication コントローラーに新規サイズを設定します。

使用例

```
  # Scale a replicaset named 'foo' to 3.
  oc scale --replicas=3 rs/foo

  # Scale a resource identified by type and name specified in "foo.yaml" to 3.
  oc scale --replicas=3 -f foo.yaml

  # If the deployment named mysql's current size is 2, scale mysql to 3.
  oc scale --current-replicas=2 --replicas=3 deployment/mysql

  # Scale multiple replication controllers.
  oc scale --replicas=5 rc/foo rc/bar rc/baz

  # Scale statefulset named 'web' to 3.
  oc scale --replicas=3 statefulset/web
```

2.5.1.117. oc secrets link

サービスアカウントにシークレットをリンクします。

使用例

```
  # Add an image pull secret to a service account to automatically use it for pulling pod images
  oc secrets link serviceaccount-name pull-secret --for=pull

  # Add an image pull secret to a service account to automatically use it for both pulling and pushing build images
  oc secrets link builder builder-image-secret --for=pull,mount

  # If the cluster's serviceAccountConfig is operating with limitSecretReferences: True, secrets must be added to the pod's service account whitelist in order to be available to the pod
  oc secrets link pod-sa pod-secret
```

2.5.1.118. oc secrets unlink

サービスアカウントからシークレットをデタッチします。

使用例

```
  # Unlink a secret currently associated with a service account
  oc secrets unlink serviceaccount-name secret-name another-secret-name ...
```

2.5.1.119. oc serviceaccounts create-kubeconfig

サービスアカウントの kubeconfig ファイルを生成します。

使用例

```
  # Create a kubeconfig file for service account 'default'
  oc serviceaccounts create-kubeconfig 'default' > default.kubeconfig
```

2.5.1.120. oc serviceaccounts get-token

サービスアカウントに割り当てられたトークンを取得します。

使用例

```
  # Get the service account token from service account 'default'
  oc serviceaccounts get-token 'default'
```

2.5.1.121. oc serviceaccounts new-token

サービスアカウントの新規トークンを生成します。

使用例

```
  # Generate a new token for service account 'default'
  oc serviceaccounts new-token 'default'
```

```
# Generate a new token for service account 'default' and apply
# labels 'foo' and 'bar' to the new token for identification
oc serviceaccounts new-token 'default' --labels foo=foo-value,bar=bar-value
```

2.5.1.122. oc set build-hook

ビルド設定のビルドフックを更新します。

使用例

```
# Clear post-commit hook on a build config
oc set build-hook bc/mybuild --post-commit --remove

# Set the post-commit hook to execute a test suite using a new entrypoint
oc set build-hook bc/mybuild --post-commit --command -- /bin/bash -c /var/lib/test-image.sh

# Set the post-commit hook to execute a shell script
oc set build-hook bc/mybuild --post-commit --script="/var/lib/test-image.sh param1 param2 &&
/var/lib/done.sh"
```

2.5.1.123. oc set build-secret

ビルド設定のビルドシークレットを更新します。

使用例

```
# Clear the push secret on a build config
oc set build-secret --push --remove bc/mybuild

# Set the pull secret on a build config
oc set build-secret --pull bc/mybuild mysecret

# Set the push and pull secret on a build config
oc set build-secret --push --pull bc/mybuild mysecret

# Set the source secret on a set of build configs matching a selector
oc set build-secret --source -l app=myapp gitsecret
```

2.5.1.124. oc set data

設定マップまたはシークレット内のデータを更新します。

使用例

```
# Set the 'password' key of a secret
oc set data secret/foo password=this_is_secret

# Remove the 'password' key from a secret
oc set data secret/foo password-

# Update the 'haproxy.conf' key of a config map from a file on disk
oc set data configmap/bar --from-file=../haproxy.conf
```

```
# Update a secret with the contents of a directory, one key per file
oc set data secret/foo --from-file=secret-dir
```

2.5.1.125. oc set deployment-hook

デプロイメント設定のデプロイメントフックを更新します。

使用例

```
# Clear pre and post hooks on a deployment config
oc set deployment-hook dc/myapp --remove --pre --post

# Set the pre deployment hook to execute a db migration command for an application
# using the data volume from the application
oc set deployment-hook dc/myapp --pre --volumes=data -- /var/lib/migrate-db.sh

# Set a mid deployment hook along with additional environment variables
oc set deployment-hook dc/myapp --mid --volumes=data -e VAR1=value1 -e VAR2=value2 --
/var/lib/prepare-deploy.sh
```

2.5.1.126. oc set env

Pod テンプレートの環境変数を更新します。

使用例

```
# Update deployment config 'myapp' with a new environment variable
oc set env dc/myapp STORAGE_DIR=/local

# List the environment variables defined on a build config 'sample-build'
oc set env bc/sample-build --list

# List the environment variables defined on all pods
oc set env pods --all --list

# Output modified build config in YAML
oc set env bc/sample-build STORAGE_DIR=/data -o yaml

# Update all containers in all replication controllers in the project to have ENV=prod
oc set env rc --all ENV=prod

# Import environment from a secret
oc set env --from=secret/mysecret dc/myapp

# Import environment from a config map with a prefix
oc set env --from=configmap/myconfigmap --prefix=MYSQL_ dc/myapp

# Remove the environment variable ENV from container 'c1' in all deployment configs
oc set env dc --all --containers="c1" ENV-

# Remove the environment variable ENV from a deployment config definition on disk and
# update the deployment config on the server
oc set env -f dc.json ENV-
```

```
  # Set some of the local shell environment into a deployment config on the server
  oc set env | grep RAILS_ | oc env -e - dc/myapp
```

2.5.1.127. oc set image

Pod テンプレートのイメージを更新します。

使用例

```
  # Set a deployment configs's nginx container image to 'nginx:1.9.1', and its busybox container image to 'busybox'.
  oc set image dc/nginx busybox=busybox nginx=nginx:1.9.1

  # Set a deployment configs's app container image to the image referenced by the imagestream tag 'openshift/ruby:2.3'.
  oc set image dc/myapp app=openshift/ruby:2.3 --source=imagestreamtag

  # Update all deployments' and rc's nginx container's image to 'nginx:1.9.1'
  oc set image deployments,rc nginx=nginx:1.9.1 --all

  # Update image of all containers of daemonset abc to 'nginx:1.9.1'
  oc set image daemonset abc *=nginx:1.9.1

  # Print result (in yaml format) of updating nginx container image from local file, without hitting the server
  oc set image -f path/to/file.yaml nginx=nginx:1.9.1 --local -o yaml
```

2.5.1.128. oc set image-lookup

アプリケーションのデプロイ時にイメージを解決する方法を変更します。

使用例

```
  # Print all of the image streams and whether they resolve local names
  oc set image-lookup

  # Use local name lookup on image stream mysql
  oc set image-lookup mysql

  # Force a deployment to use local name lookup
  oc set image-lookup deploy/mysql

  # Show the current status of the deployment lookup
  oc set image-lookup deploy/mysql --list

  # Disable local name lookup on image stream mysql
  oc set image-lookup mysql --enabled=false

  # Set local name lookup on all image streams
  oc set image-lookup --all
```

2.5.1.129. oc set probe

Pod テンプレートでプローブを更新します。

使用例

```
# Clear both readiness and liveness probes off all containers
oc set probe dc/myapp --remove --readiness --liveness

# Set an exec action as a liveness probe to run 'echo ok'
oc set probe dc/myapp --liveness -- echo ok

# Set a readiness probe to try to open a TCP socket on 3306
oc set probe rc/mysql --readiness --open-tcp=3306

# Set an HTTP startup probe for port 8080 and path /healthz over HTTP on the pod IP
oc probe dc/webapp --startup --get-url=http://:8080/healthz

# Set an HTTP readiness probe for port 8080 and path /healthz over HTTP on the pod IP
oc probe dc/webapp --readiness --get-url=http://:8080/healthz

# Set an HTTP readiness probe over HTTPS on 127.0.0.1 for a hostNetwork pod
oc set probe dc/router --readiness --get-url=https://127.0.0.1:1936/stats

# Set only the initial-delay-seconds field on all deployments
oc set probe dc --all --readiness --initial-delay-seconds=30
```

2.5.1.130. oc set resources

オブジェクトのリソース要求/制限を Pod テンプレートで更新します。

使用例

```
# Set a deployments nginx container CPU limits to "200m and memory to 512Mi"
oc set resources deployment nginx -c=nginx --limits=cpu=200m,memory=512Mi

# Set the resource request and limits for all containers in nginx
oc set resources deployment nginx --limits=cpu=200m,memory=512Mi --
requests=cpu=100m,memory=256Mi

# Remove the resource requests for resources on containers in nginx
oc set resources deployment nginx --limits=cpu=0,memory=0 --requests=cpu=0,memory=0

# Print the result (in YAML format) of updating nginx container limits locally, without hitting the server
oc set resources -f path/to/file.yaml --limits=cpu=200m,memory=512Mi --local -o yaml
```

2.5.1.131. oc set route-backends

ルートのバックエンドを更新します。

使用例

```
# Print the backends on the route 'web'
oc set route-backends web

# Set two backend services on route 'web' with 2/3rds of traffic going to 'a'
```

```
oc set route-backends web a=2 b=1

# Increase the traffic percentage going to b by 10%% relative to a
oc set route-backends web --adjust b=+10%%

# Set traffic percentage going to b to 10%% of the traffic going to a
oc set route-backends web --adjust b=10%%

# Set weight of b to 10
oc set route-backends web --adjust b=10

# Set the weight to all backends to zero
oc set route-backends web --zero
```

2.5.1.132. oc set selector

リソースにセレクターを設定します。

使用例

```
# Set the labels and selector before creating a deployment/service pair.
oc create service clusterip my-svc --clusterip="None" -o yaml --dry-run | oc set selector --local -f - 'environment=qa' -o yaml | oc create -f -
oc create deployment my-dep -o yaml --dry-run | oc label --local -f - environment=qa -o yaml | oc create -f -
```

2.5.1.133. oc set serviceaccount

リソースの ServiceAccount を更新します。

使用例

```
# Set deployment nginx-deployment's service account to serviceaccount1
oc set serviceaccount deployment nginx-deployment serviceaccount1

# Print the result (in YAML format) of updated nginx deployment with service account from a local file, without hitting the API server
oc set sa -f nginx-deployment.yaml serviceaccount1 --local --dry-run -o yaml
```

2.5.1.134. oc set subject

RoleBinding/ClusterRoleBinding でユーザー、グループ、または ServiceAccount を更新します。

使用例

```
# Update a cluster role binding for serviceaccount1
oc set subject clusterrolebinding admin --serviceaccount=namespace:serviceaccount1

# Update a role binding for user1, user2, and group1
oc set subject rolebinding admin --user=user1 --user=user2 --group=group1
```

```
  # Print the result (in YAML format) of updating role binding subjects locally, without hitting the server
  oc create rolebinding admin --role=admin --user=admin -o yaml --dry-run | oc set subject --local -f -
--user=foo -o yaml
```

2.5.1.135. oc set triggers

1 つ以上のオブジェクトでトリガーを更新します。

使用例

```
  # Print the triggers on the deployment config 'myapp'
  oc set triggers dc/myapp

  # Set all triggers to manual
  oc set triggers dc/myapp --manual

  # Enable all automatic triggers
  oc set triggers dc/myapp --auto

  # Reset the GitHub webhook on a build to a new, generated secret
  oc set triggers bc/webapp --from-github
  oc set triggers bc/webapp --from-webhook

  # Remove all triggers
  oc set triggers bc/webapp --remove-all

  # Stop triggering on config change
  oc set triggers dc/myapp --from-config --remove

  # Add an image trigger to a build config
  oc set triggers bc/webapp --from-image=namespace1/image:latest

  # Add an image trigger to a stateful set on the main container
  oc set triggers statefulset/db --from-image=namespace1/image:latest -c main
```

2.5.1.136. oc set volumes

Pod テンプレートでボリュームを更新します。

使用例

```
  # List volumes defined on all deployment configs in the current project
  oc set volume dc --all

  # Add a new empty dir volume to deployment config (dc) 'myapp' mounted under
  # /var/lib/myapp
  oc set volume dc/myapp --add --mount-path=/var/lib/myapp

  # Use an existing persistent volume claim (pvc) to overwrite an existing volume 'v1'
  oc set volume dc/myapp --add --name=v1 -t pvc --claim-name=pvc1 --overwrite

  # Remove volume 'v1' from deployment config 'myapp'
  oc set volume dc/myapp --remove --name=v1
```

```
# Create a new persistent volume claim that overwrites an existing volume 'v1'
oc set volume dc/myapp --add --name=v1 -t pvc --claim-size=1G --overwrite

# Change the mount point for volume 'v1' to /data
oc set volume dc/myapp --add --name=v1 -m /data --overwrite

# Modify the deployment config by removing volume mount "v1" from container "c1"
# (and by removing the volume "v1" if no other containers have volume mounts that reference it)
oc set volume dc/myapp --remove --name=v1 --containers=c1

# Add new volume based on a more complex volume source (AWS EBS, GCE PD,
# Ceph, Gluster, NFS, ISCSI, ...)
oc set volume dc/myapp --add -m /data --source=<json-string>
```

2.5.1.137. oc start-build

新しいビルドを開始します。

使用例

```
# Starts build from build config "hello-world"
oc start-build hello-world

# Starts build from a previous build "hello-world-1"
oc start-build --from-build=hello-world-1

# Use the contents of a directory as build input
oc start-build hello-world --from-dir=src/

# Send the contents of a Git repository to the server from tag 'v2'
oc start-build hello-world --from-repo=../hello-world --commit=v2

# Start a new build for build config "hello-world" and watch the logs until the build
# completes or fails
oc start-build hello-world --follow

# Start a new build for build config "hello-world" and wait until the build completes. It
# exits with a non-zero return code if the build fails
oc start-build hello-world --wait
```

2.5.1.138. oc status

現在のプロジェクトの概要を表示します。

使用例

```
# See an overview of the current project
oc status

# Export the overview of the current project in an svg file
oc status -o dot | dot -T svg -o project.svg

# See an overview of the current project including details for any identified issues
oc status --suggest
```

2.5.1.139. oc 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。

使用例

```
  # Tag the current image for the image stream 'openshift/ruby' and tag '2.0' into the image stream
'yourproject/ruby with tag 'tip'
  oc tag openshift/ruby:2.0 yourproject/ruby:tip

  # Tag a specific image
  oc tag
openshift/ruby@sha256:6b646fa6bf5e5e4c7fa41056c27910e679c03ebe7f93e361e6515a9da7e258cc
yourproject/ruby:tip

  # Tag an external container image
  oc tag --source=docker openshift/origin-control-plane:latest yourproject/ruby:tip

  # Tag an external container image and request pullthrough for it
  oc tag --source=docker openshift/origin-control-plane:latest yourproject/ruby:tip --reference-
policy=local

  # Remove the specified spec tag from an image stream
  oc tag openshift/origin-control-plane:latest -d
```

2.5.1.140. oc version

クライアントおよびサーバーのバージョン情報を出力します。

使用例

```
  # Print the OpenShift client, kube-apiserver, and openshift-apiserver version information for the
current context
  oc version

  # Print the OpenShift client, kube-apiserver, and openshift-apiserver version numbers for the current
context
  oc version --short

  # Print the OpenShift client version information for the current context
  oc version --client
```

2.5.1.141. oc wait

実験的: 1 つ以上のリソースの特定の条件を待機します。

使用例

```
  # Wait for the pod "busybox1" to contain the status condition of type "Ready".
  oc wait --for=condition=Ready pod/busybox1

  # The default value of status condition is true, you can set false.
  oc wait --for=condition=Ready=false pod/busybox1
```

```
  # Wait for the pod "busybox1" to be deleted, with a timeout of 60s, after having issued the "delete" command.
  oc delete pod/busybox1
  oc wait --for=delete pod/busybox1 --timeout=60s
```

2.5.1.142. oc whoami

現行セッションに関する情報を返します。

使用例

```
  # Display the currently authenticated user
  oc whoami
```

2.5.2. 関連情報

- OpenShift CLI 管理者コマンドリファレンス

2.6. OPENSHIFT CLI 管理者コマンドリファレンス

このリファレンスは、OpenShift CLI (**oc**) 管理者コマンドの説明およびコマンド例を示しています。これらの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。

開発者コマンドは、OpenShift CLI 開発者コマンドリファレンス を参照してください。

oc adm -h を実行して、すべての管理者コマンドを表示するか、または **oc <command> --help** を実行して、特定のコマンドに関する追加情報を取得します。

2.6.1. OpenShift CLI (oc) 管理者コマンド

2.6.1.1. oc adm build-chain

ビルドの入力と依存関係を出力します。

使用例

```
  # Build the dependency tree for the 'latest' tag in <image-stream>
  oc adm build-chain <image-stream>

  # Build the dependency tree for the 'v2' tag in dot format and visualize it via the dot utility
  oc adm build-chain <image-stream>:v2 -o dot | dot -T svg -o deps.svg

  # Build the dependency tree across all namespaces for the specified image stream tag found in the 'test' namespace
  oc adm build-chain <image-stream> -n test --all
```

2.6.1.2. oc adm catalog mirror

operator-registry カタログをミラーリングします。

使用例

```
# Mirror an operator-registry image and its contents to a registry
oc adm catalog mirror quay.io/my/image:latest myregistry.com

# Mirror an operator-registry image and its contents to a particular namespace in a registry
oc adm catalog mirror quay.io/my/image:latest myregistry.com/my-namespace

# Mirror to an airgapped registry by first mirroring to files
oc adm catalog mirror quay.io/my/image:latest file:///local/index
oc adm catalog mirror file:///local/index/my/image:latest my-airgapped-registry.com

# Configure a cluster to use a mirrored registry
oc apply -f manifests/imageContentSourcePolicy.yaml

# Edit the mirroring mappings and mirror with "oc image mirror" manually
oc adm catalog mirror --manifests-only quay.io/my/image:latest myregistry.com
oc image mirror -f manifests/mapping.txt

# Delete all ImageContentSourcePolicies generated by oc adm catalog mirror
oc delete imagecontentsourcepolicy -l operators.openshift.org/catalog=true
```

2.6.1.3. oc adm completion

指定されたシェル (bash または zsh) の補完コードを出力します。

使用例

```
# Installing bash completion on macOS using homebrew
## If running Bash 3.2 included with macOS
brew install bash-completion
## or, if running Bash 4.1+
brew install bash-completion@2
## If oc is installed via homebrew, this should start working immediately.
## If you've installed via other means, you may need add the completion to your completion directory
oc completion bash > $(brew --prefix)/etc/bash_completion.d/oc


# Installing bash completion on Linux
## If bash-completion is not installed on Linux, please install the 'bash-completion' package
## via your distribution's package manager.
## Load the oc completion code for bash into the current shell
source <(oc completion bash)
## Write bash completion code to a file and source it from .bash_profile
oc completion bash > ~/.kube/completion.bash.inc
printf "
# Kubectl shell completion
source '$HOME/.kube/completion.bash.inc'
" >> $HOME/.bash_profile
source $HOME/.bash_profile

# Load the oc completion code for zsh[1] into the current shell
source <(oc completion zsh)
# Set the oc completion code for zsh[1] to autoload on startup
oc completion zsh > "${fpath[1]}/_oc"
```

2.6.1.4. oc adm config current-context

current-context を表示します

使用例

```
# Display the current-context
oc config current-context
```

2.6.1.5. oc adm config delete-cluster

kubeconfig から指定されたクラスターを削除します。

使用例

```
# Delete the minikube cluster
oc config delete-cluster minikube
```

2.6.1.6. oc adm config delete-context

kubeconfig から指定されたコンテキストを削除します。

使用例

```
# Delete the context for the minikube cluster
oc config delete-context minikube
```

2.6.1.7. oc adm config delete-user

kubeconfig から指定されたユーザーを削除します。

使用例

```
# Delete the minikube user
oc config delete-user minikube
```

2.6.1.8. oc adm config get-clusters

kubeconfig に定義されるクラスターを表示します。

使用例

```
# List the clusters oc knows about
oc config get-clusters
```

2.6.1.9. oc adm config get-contexts

コンテキストを 1 つまたは複数記述します。

使用例

```
# List all the contexts in your kubeconfig file
oc config get-contexts

# Describe one context in your kubeconfig file.
oc config get-contexts my-context
```

2.6.1.10. oc adm config get-users

kubeconfig で定義されるユーザーを表示します。

使用例

```
# List the users oc knows about
oc config get-users
```

2.6.1.11. oc adm config rename-context

kubeconfig ファイルからのコンテキストの名前を変更します。

使用例

```
# Rename the context 'old-name' to 'new-name' in your kubeconfig file
oc config rename-context old-name new-name
```

2.6.1.12. oc adm config set

kubeconfig ファイルに個別の値を設定します。

使用例

```
# Set server field on the my-cluster cluster to https://1.2.3.4
oc config set clusters.my-cluster.server https://1.2.3.4

# Set certificate-authority-data field on the my-cluster cluster.
oc config set clusters.my-cluster.certificate-authority-data $(echo "cert_data_here" | base64 -i -)

# Set cluster field in the my-context context to my-cluster.
oc config set contexts.my-context.cluster my-cluster

# Set client-key-data field in the cluster-admin user using --set-raw-bytes option.
oc config set users.cluster-admin.client-key-data cert_data_here --set-raw-bytes=true
```

2.6.1.13. oc adm config set-cluster

kubeconfig でクラスターエントリーを設定します。

使用例

```
# Set only the server field on the e2e cluster entry without touching other values.
oc config set-cluster e2e --server=https://1.2.3.4

# Embed certificate authority data for the e2e cluster entry
```

```
oc config set-cluster e2e --embed-certs --certificate-authority=~/.kube/e2e/kubernetes.ca.crt

# Disable cert checking for the dev cluster entry
oc config set-cluster e2e --insecure-skip-tls-verify=true

# Set custom TLS server name to use for validation for the e2e cluster entry
oc config set-cluster e2e --tls-server-name=my-cluster-name
```

2.6.1.14. oc adm config set-context

kubeconfig のコンテキストエントリーを設定します。

使用例

```
# Set the user field on the gce context entry without touching other values
oc config set-context gce --user=cluster-admin
```

2.6.1.15. oc adm config set-credentials

kubeconfig のユーザーエントリーを設定します。

使用例

```
# Set only the "client-key" field on the "cluster-admin"
# entry, without touching other values:
oc config set-credentials cluster-admin --client-key=~/.kube/admin.key

# Set basic auth for the "cluster-admin" entry
oc config set-credentials cluster-admin --username=admin --password=uXFGweU9l35qcif

# Embed client certificate data in the "cluster-admin" entry
oc config set-credentials cluster-admin --client-certificate=~/.kube/admin.crt --embed-certs=true

# Enable the Google Compute Platform auth provider for the "cluster-admin" entry
oc config set-credentials cluster-admin --auth-provider=gcp

# Enable the OpenID Connect auth provider for the "cluster-admin" entry with additional args
oc config set-credentials cluster-admin --auth-provider=oidc --auth-provider-arg=client-id=foo --auth-
provider-arg=client-secret=bar

# Remove the "client-secret" config value for the OpenID Connect auth provider for the "cluster-
admin" entry
oc config set-credentials cluster-admin --auth-provider=oidc --auth-provider-arg=client-secret-

# Enable new exec auth plugin for the "cluster-admin" entry
oc config set-credentials cluster-admin --exec-command=/path/to/the/executable --exec-api-
version=client.authentication.k8s.io/v1beta1

# Define new exec auth plugin args for the "cluster-admin" entry
oc config set-credentials cluster-admin --exec-arg=arg1 --exec-arg=arg2

# Create or update exec auth plugin environment variables for the "cluster-admin" entry
oc config set-credentials cluster-admin --exec-env=key1=val1 --exec-env=key2=val2
```

```
# Remove exec auth plugin environment variables for the "cluster-admin" entry
oc config set-credentials cluster-admin --exec-env=var-to-remove-
```

2.6.1.16. oc adm config unset

kubeconfig ファイルでの個別値の設定を解除します。

使用例

```
# Unset the current-context.
oc config unset current-context

# Unset namespace in foo context.
oc config unset contexts.foo.namespace
```

2.6.1.17. oc adm config use-context

kubeconfig ファイルで current-context を設定します。

使用例

```
# Use the context for the minikube cluster
oc config use-context minikube
```

2.6.1.18. oc adm config view

マージされた kubeconfig 設定または指定された kubeconfig ファイルを表示します。

使用例

```
# Show merged kubeconfig settings.
oc config view

# Show merged kubeconfig settings and raw certificate data.
oc config view --raw

# Get the password for the e2e user
oc config view -o jsonpath='{.users[?(@.name == "e2e")].user.password}'
```

2.6.1.19. oc adm cordon

ノードにスケジュール対象外 (unschedulable) のマークを付けます。

使用例

```
# Mark node "foo" as unschedulable.
oc adm cordon foo
```

2.6.1.20. oc adm 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。

使用例

```
# Output a bootstrap project template in YAML format to stdout
oc adm create-bootstrap-project-template -o yaml
```

2.6.1.21. oc adm create-error-template

エラーページのテンプレートを作成します。

使用例

```
# Output a template for the error page to stdout
oc adm create-error-template
```

2.6.1.22. oc adm create-login-template

ログインテンプレートを作成します。

使用例

```
# Output a template for the login page to stdout
oc adm create-login-template
```

2.6.1.23. oc adm create-provider-selection-template

プロバイダー選択のテンプレートを作成します。

使用例

```
# Output a template for the provider selection page to stdout
oc adm create-provider-selection-template
```

2.6.1.24. oc adm drain

ノードをドレイン (解放) してメンテナンスを準備します。

使用例

```
  # Drain node "foo", even if there are pods not managed by a ReplicationController, ReplicaSet, Job,
DaemonSet or StatefulSet on it.
  $ oc adm drain foo --force

  # As above, but abort if there are pods not managed by a ReplicationController, ReplicaSet, Job,
DaemonSet or StatefulSet, and use a grace period of 15 minutes.
  $ oc adm drain foo --grace-period=900
```

2.6.1.25. oc adm groups add-users

ユーザーをグループに追加します。

使用例

```
# Add user1 and user2 to my-group
oc adm groups add-users my-group user1 user2
```

2.6.1.26. oc adm groups new

新規グループを作成します。

使用例

```
# Add a group with no users
oc adm groups new my-group

# Add a group with two users
oc adm groups new my-group user1 user2

# Add a group with one user and shorter output
oc adm groups new my-group user1 -o name
```

2.6.1.27. oc adm groups prune

外部プロバイダーから欠落しているレコードを参照する以前の OpenShift グループを削除します。

使用例

```
# Prune all orphaned groups
oc adm groups prune --sync-config=/path/to/ldap-sync-config.yaml --confirm

# Prune all orphaned groups except the ones from the blacklist file
oc adm groups prune --blacklist=/path/to/blacklist.txt --sync-config=/path/to/ldap-sync-config.yaml --
confirm

# Prune all orphaned groups from a list of specific groups specified in a whitelist file
oc adm groups prune --whitelist=/path/to/whitelist.txt --sync-config=/path/to/ldap-sync-config.yaml --
confirm

# Prune all orphaned groups from a list of specific groups specified in a whitelist
oc adm groups prune groups/group_name groups/other_name --sync-config=/path/to/ldap-sync-
config.yaml --confirm
```

2.6.1.28. oc adm groups remove-users

グループからユーザーを削除します。

使用例

```
# Remove user1 and user2 from my-group
oc adm groups remove-users my-group user1 user2
```

2.6.1.29. oc adm groups sync

OpenShift グループと外部プロバイダーからのレコードを同期します。

使用例

```
  # Sync all groups with an LDAP server
  oc adm groups sync --sync-config=/path/to/ldap-sync-config.yaml --confirm

  # Sync all groups except the ones from the blacklist file with an LDAP server
  oc adm groups sync --blacklist=/path/to/blacklist.txt --sync-config=/path/to/ldap-sync-config.yaml --
confirm

  # Sync specific groups specified in a whitelist file with an LDAP server
  oc adm groups sync --whitelist=/path/to/whitelist.txt --sync-config=/path/to/sync-config.yaml --
confirm

  # Sync all OpenShift groups that have been synced previously with an LDAP server
  oc adm groups sync --type=openshift --sync-config=/path/to/ldap-sync-config.yaml --confirm

  # Sync specific OpenShift groups if they have been synced previously with an LDAP server
  oc adm groups sync groups/group1 groups/group2 groups/group3 --sync-config=/path/to/sync-
config.yaml --confirm
```

2.6.1.30. oc adm inspect

指定のリソースのデバッグデータを収集します。

使用例

```
  # Collect debugging data for the "openshift-apiserver" clusteroperator
  oc adm inspect clusteroperator/openshift-apiserver

  # Collect debugging data for the "openshift-apiserver" and "kube-apiserver" clusteroperators
  oc adm inspect clusteroperator/openshift-apiserver clusteroperator/kube-apiserver

  # Collect debugging data for all clusteroperators
  oc adm inspect clusteroperator

  # Collect debugging data for all clusteroperators and clusterversions
  oc adm inspect clusteroperators,clusterversions
```

2.6.1.31. oc adm migrate template-instances

テンプレートインスタンスを更新して、最新の group-version-kinds を参照するようにします。

使用例

```
  # Perform a dry-run of updating all objects
  oc adm migrate template-instances

  # To actually perform the update, the confirm flag must be appended
  oc adm migrate template-instances --confirm
```

2.6.1.32. oc adm must-gather

Pod の新規インスタンスを起動してデバッグ情報を収集します。

使用例

```
  # Gather information using the default plug-in image and command, writing into ./must-gather.local.
<rand>
  oc adm must-gather

  # Gather information with a specific local folder to copy to
  oc adm must-gather --dest-dir=/local/directory

  # Gather audit information
  oc adm must-gather -- /usr/bin/gather_audit_logs

  # Gather information using multiple plug-in images
  oc adm must-gather --image=quay.io/kubevirt/must-gather --image=quay.io/openshift/origin-must-
gather

  # Gather information using a specific image stream plug-in
  oc adm must-gather --image-stream=openshift/must-gather:latest

  # Gather information using a specific image, command, and pod-dir
  oc adm must-gather --image=my/image:tag --source-dir=/pod/directory -- myspecial-command.sh
```

2.6.1.33. oc adm new-project

新規プロジェクトを作成します。

使用例

```
  # Create a new project using a node selector
  oc adm new-project myproject --node-selector='type=user-node,region=east'
```

2.6.1.34. oc adm node-logs

ノードのログを表示し、フィルターします。

使用例

```
  # Show kubelet logs from all masters
  oc adm node-logs --role master -u kubelet

  # See what logs are available in masters in /var/logs
  oc adm node-logs --role master --path=/

  # Display cron log file from all masters
  oc adm node-logs --role master --path=cron
```

2.6.1.35. oc adm pod-network isolate-projects

プロジェクトネットワークを分離します。

使用例

```
  # Provide isolation for project p1
```

```
oc adm pod-network isolate-projects <p1>

# Allow all projects with label name=top-secret to have their own isolated project network
oc adm pod-network isolate-projects --selector='name=top-secret'
```

2.6.1.36. oc adm pod-network join-projects

プロジェクトネットワークに参加します。

使用例

```
# Allow project p2 to use project p1 network
oc adm pod-network join-projects --to=<p1> <p2>

# Allow all projects with label name=top-secret to use project p1 network
oc adm pod-network join-projects --to=<p1> --selector='name=top-secret'
```

2.6.1.37. oc adm pod-network make-projects-global

プロジェクトネットワークをグローバルにします。

使用例

```
# Allow project p1 to access all pods in the cluster and vice versa
oc adm pod-network make-projects-global <p1>

# Allow all projects with label name=share to access all pods in the cluster and vice versa
oc adm pod-network make-projects-global --selector='name=share'
```

2.6.1.38. oc adm policy add-role-to-user

現在のプロジェクトのユーザーまたはサービスアカウントをロールに追加します。

使用例

```
# Add the 'view' role to user1 for the current project
oc policy add-role-to-user view user1

# Add the 'edit' role to serviceaccount1 for the current project
oc policy add-role-to-user edit -z serviceaccount1
```

2.6.1.39. oc adm policy add-scc-to-group

SCC (Security Context Constraints) オブジェクトをグループに追加します。

使用例

```
# Add the 'restricted' security context constraint to group1 and group2
oc adm policy add-scc-to-group restricted group1 group2
```

2.6.1.40. oc adm policy add-scc-to-user

SCC (security context constraint) をユーザーまたはサービスアカウントに追加します。

使用例

```
  # Add the 'restricted' security context constraint to user1 and user2
  oc adm policy add-scc-to-user restricted user1 user2

  # Add the 'privileged' security context constraint to serviceaccount1 in the current namespace
  oc adm policy add-scc-to-user privileged -z serviceaccount1
```

2.6.1.41. oc adm policy scc-review

Pod を作成できるサービスアカウントを確認します。

使用例

```
  # Check whether service accounts sa1 and sa2 can admit a pod with a template pod spec specified in my_resource.yaml
  # Service Account specified in myresource.yaml file is ignored
  oc policy scc-review -z sa1,sa2 -f my_resource.yaml

  # Check whether service accounts system:serviceaccount:bob:default can admit a pod with a template pod spec specified in my_resource.yaml
  oc policy scc-review -z system:serviceaccount:bob:default -f my_resource.yaml

  # Check whether the service account specified in my_resource_with_sa.yaml can admit the pod
  oc policy scc-review -f my_resource_with_sa.yaml

  # Check whether the default service account can admit the pod; default is taken since no service account is defined in myresource_with_no_sa.yaml
  oc policy scc-review -f myresource_with_no_sa.yaml
```

2.6.1.42. oc adm policy scc-subject-review

ユーザーまたはサービスアカウントが Pod を作成できるかどうかを確認します。

使用例

```
  # Check whether user bob can create a pod specified in myresource.yaml
  oc policy scc-subject-review -u bob -f myresource.yaml

  # Check whether user bob who belongs to projectAdmin group can create a pod specified in myresource.yaml
  oc policy scc-subject-review -u bob -g projectAdmin -f myresource.yaml

  # Check whether a service account specified in the pod template spec in myresourcewithsa.yaml can create the pod
  oc policy scc-subject-review -f myresourcewithsa.yaml
```

2.6.1.43. oc adm prune builds

以前の完了済みおよび失敗したビルドを削除します。

使用例

```
# Dry run deleting older completed and failed builds and also including
# all builds whose associated build config no longer exists
oc adm prune builds --orphans

# To actually perform the prune operation, the confirm flag must be appended
oc adm prune builds --orphans --confirm
```

2.6.1.44. oc adm prune deployments

以前の完了済みおよび失敗したデプロイメント設定を削除します。

使用例

```
# Dry run deleting all but the last complete deployment for every deployment config
oc adm prune deployments --keep-complete=1

# To actually perform the prune operation, the confirm flag must be appended
oc adm prune deployments --keep-complete=1 --confirm
```

2.6.1.45. oc adm prune groups

外部プロバイダーから欠落しているレコードを参照する以前の OpenShift グループを削除します。

使用例

```
# Prune all orphaned groups
oc adm prune groups --sync-config=/path/to/ldap-sync-config.yaml --confirm

# Prune all orphaned groups except the ones from the blacklist file
oc adm prune groups --blacklist=/path/to/blacklist.txt --sync-config=/path/to/ldap-sync-config.yaml --
confirm

# Prune all orphaned groups from a list of specific groups specified in a whitelist file
oc adm prune groups --whitelist=/path/to/whitelist.txt --sync-config=/path/to/ldap-sync-config.yaml --
confirm

# Prune all orphaned groups from a list of specific groups specified in a whitelist
oc adm prune groups groups/group_name groups/other_name --sync-config=/path/to/ldap-sync-
config.yaml --confirm
```

2.6.1.46. oc adm prune images

参照されていないイメージを削除します。

使用例

```
# See what the prune command would delete if only images and their referrers were more than an
hour old
# and obsoleted by 3 newer revisions under the same tag were considered
oc adm prune images --keep-tag-revisions=3 --keep-younger-than=60m
```

```
  # To actually perform the prune operation, the confirm flag must be appended
  oc adm prune images --keep-tag-revisions=3 --keep-younger-than=60m --confirm

  # See what the prune command would delete if we are interested in removing images
  # exceeding currently set limit ranges ('openshift.io/Image')
  oc adm prune images --prune-over-size-limit

  # To actually perform the prune operation, the confirm flag must be appended
  oc adm prune images --prune-over-size-limit --confirm

  # Force the insecure http protocol with the particular registry host name
  oc adm prune images --registry-url=http://registry.example.org --confirm

  # Force a secure connection with a custom certificate authority to the particular registry host name
  oc adm prune images --registry-url=registry.example.org --certificate-
authority=/path/to/custom/ca.crt --confirm
```

2.6.1.47. oc adm release extract

更新ペイロードの内容をディスクに抽出します。

使用例

```
  # Use git to check out the source code for the current cluster release to DIR
  oc adm release extract --git=DIR

  # Extract cloud credential requests for AWS
  oc adm release extract --credentials-requests --cloud=aws
```

2.6.1.48. oc adm release info

リリースに関する情報を表示します。

使用例

```
  # Show information about the cluster's current release
  oc adm release info

  # Show the source code that comprises a release
  oc adm release info 4.2.2 --commit-urls

  # Show the source code difference between two releases
  oc adm release info 4.2.0 4.2.2 --commits

  # Show where the images referenced by the release are located
  oc adm release info quay.io/openshift-release-dev/ocp-release:4.2.2 --pullspecs
```

2.6.1.49. oc adm release mirror

リリースを別のイメージレジストリーの場所にミラーリングします。

使用例

```
  # Perform a dry run showing what would be mirrored, including the mirror objects
  oc adm release mirror 4.3.0 --to myregistry.local/openshift/release \
  --release-image-signature-to-dir /tmp/releases --dry-run

  # Mirror a release into the current directory
  oc adm release mirror 4.3.0 --to file://openshift/release \
  --release-image-signature-to-dir /tmp/releases

  # Mirror a release to another directory in the default location
  oc adm release mirror 4.3.0 --to-dir /tmp/releases

  # Upload a release from the current directory to another server
  oc adm release mirror --from file://openshift/release --to myregistry.com/openshift/release \
  --release-image-signature-to-dir /tmp/releases

  # Mirror the 4.3.0 release to repository registry.example.com and apply signatures to connected
cluster
  oc adm release mirror --from=quay.io/openshift-release-dev/ocp-release:4.3.0-x86_64 \
  --to=registry.example.com/your/repository --apply-release-image-signature
```

2.6.1.50. oc adm release new

新しい OpenShift リリースを作成します。

使用例

```
  # Create a release from the latest origin images and push to a DockerHub repo
  oc adm release new --from-image-stream=4.1 -n origin --to-image
docker.io/mycompany/myrepo:latest

  # Create a new release with updated metadata from a previous release
  oc adm release new --from-release registry.svc.ci.openshift.org/origin/release:v4.1 --name 4.1.1 \
  --previous 4.1.0 --metadata ... --to-image docker.io/mycompany/myrepo:latest

  # Create a new release and override a single image
  oc adm release new --from-release registry.svc.ci.openshift.org/origin/release:v4.1 \
  cli=docker.io/mycompany/cli:latest --to-image docker.io/mycompany/myrepo:latest

  # Run a verification pass to ensure the release can be reproduced
  oc adm release new --from-release registry.svc.ci.openshift.org/origin/release:v4.1
```

2.6.1.51. oc adm taint

1 つ以上のノードでテイントを更新します。

使用例

```
  # Update node 'foo' with a taint with key 'dedicated' and value 'special-user' and effect 'NoSchedule'.
  # If a taint with that key and effect already exists, its value is replaced as specified.
  oc adm taint nodes foo dedicated=special-user:NoSchedule

  # Remove from node 'foo' the taint with key 'dedicated' and effect 'NoSchedule' if one exists.
  oc adm taint nodes foo dedicated:NoSchedule-
```

```
# Remove from node 'foo' all the taints with key 'dedicated'
oc adm taint nodes foo dedicated-

# Add a taint with key 'dedicated' on nodes having label mylabel=X
oc adm taint node -l myLabel=X  dedicated=foo:PreferNoSchedule

# Add to node 'foo' a taint with key 'bar' and no value
oc adm taint nodes foo bar:NoSchedule
```

2.6.1.52. oc adm top images

イメージの使用状況の統計を表示します。

使用例

```
# Show usage statistics for images
oc adm top images
```

2.6.1.53. oc adm top imagestreams

イメージストリームの使用状況の統計を表示します。

使用例

```
# Show usage statistics for image streams
oc adm top imagestreams
```

2.6.1.54. oc adm top node

ノードのリソース (CPU/メモリー) の使用状況を表示します。

使用例

```
# Show metrics for all nodes
oc adm top node

# Show metrics for a given node
oc adm top node NODE_NAME
```

2.6.1.55. oc adm top pod

Pod のリソース (CPU/メモリー) の使用状況を表示します。

使用例

```
# Show metrics for all pods in the default namespace
oc adm top pod

# Show metrics for all pods in the given namespace
oc adm top pod --namespace=NAMESPACE

# Show metrics for a given pod and its containers
```

```
oc adm top pod POD_NAME --containers

# Show metrics for the pods defined by label name=myLabel
oc adm top pod -l name=myLabel
```

2.6.1.56. oc adm uncordon

ノードにスケジュール対象 (schedulable) のマークを付けます。

使用例

```
# Mark node "foo" as schedulable.
$ oc adm uncordon foo
```

2.6.1.57. oc adm verify-image-signature

イメージ署名に含まれるイメージ ID を確認します。

使用例

```
# Verify the image signature and identity using the local GPG keychain
oc adm verify-image-signature
sha256:c841e9b64e4579bd56c794bdd7c36e1c257110fd2404bebbb8b613e4935228c4 \
--expected-identity=registry.local:5000/foo/bar:v1

# Verify the image signature and identity using the local GPG keychain and save the status
oc adm verify-image-signature
sha256:c841e9b64e4579bd56c794bdd7c36e1c257110fd2404bebbb8b613e4935228c4 \
--expected-identity=registry.local:5000/foo/bar:v1 --save

# Verify the image signature and identity via exposed registry route
oc adm verify-image-signature
sha256:c841e9b64e4579bd56c794bdd7c36e1c257110fd2404bebbb8b613e4935228c4 \
--expected-identity=registry.local:5000/foo/bar:v1 \
--registry-url=docker-registry.foo.com

# Remove all signature verifications from the image
oc adm verify-image-signature
sha256:c841e9b64e4579bd56c794bdd7c36e1c257110fd2404bebbb8b613e4935228c4 --remove-all
```

2.6.2. 関連情報

- OpenShift CLI 開発者コマンドリファレンス

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。

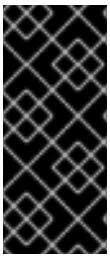
2.7.1. oc バイナリー

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 を参照してください。

- 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 4.8 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 footnote:versionpolicyn **N** は 1 以上の数値 (**oc** クライアント)
X.Y (サーバー)	1	3
X.Y+N footnote:versionpolicyn[] (サーバー)	2	1

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

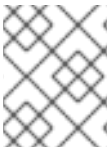
odo は、オペレーティングシステムのメトリクス、RAM、CPU、コア数、**odo** バージョン、エラー、成功/失敗、および **odo** コマンドの完了までにかかる時間を含む、使用方法に関する情報を収集します。

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

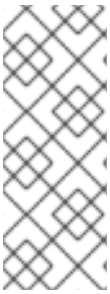
利用可能な Operator およびそれらの関連サービスを一覧表示するには、以下のコマンドを実行します。

```
$ 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。

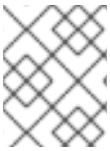
ソースコードが現在のディレクトリーに存在する場合は、**--context** フラグを使用してパスを指定できます。たとえば、nodejs コンポーネントのソースが現在の作業ディレクトリーと相対的に **node-backend** というフォルダーにある場合は、以下のコマンドを実行します。

```
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:

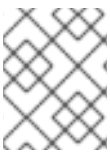
```
The component 'nodejs-testing-xnfg' has the following storage attached:
NAME      SIZE     PATH      CONTAINER     STATE
store     1Gi      /data     nodejs2       Not Pushed
```

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 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。

4.1. 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。

4.2. 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5章 PIPELINES CLI (TKN)

5.1. 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。

5.1.1. 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
```

5.1.2. 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-1.5-for-rhel-8-x86_64-rpms"
```

- Linux on IBM Z and LinuxONE (s390x)

```
# subscription-manager repos --enable="pipelines-1.5-for-rhel-8-s390x-rpms"
```

- Linux on IBM Power Systems (ppc64le)

```
# subscription-manager repos --enable="pipelines-1.5-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
```

5.1.3. 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
```

5.1.4. 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
```

5.2. 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。

5.2.1. 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。

5.3. 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。

5.3.1. 基本的な構文

tkn [command or options] [arguments…]

5.3.2. グローバルオプション

--help, -h

5.3.3. ユーティリティーコマンド

5.3.3.1. tkn

tkn CLI の親コマンド。

例: すべてのオプションの表示

```
$ tkn
```

5.3.3.2. 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
```

5.3.3.3. 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
```

5.3.4. Pipelines 管理コマンド

5.3.4.1. パイプライン

Pipeline を管理します。

例: ヘルプの表示

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

5.3.4.2. pipeline delete

Pipeline を削除します。

例: **namespace** から **mypipeline Pipeline** を削除します。

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

5.3.4.3. pipeline describe

Pipeline を記述します。

例: **mypipeline Pipeline** を記述します。

```
$ tkn pipeline describe mypipeline
```

5.3.4.4. pipeline list

Pipeline の一覧を表示します。

例: **Pipeline** の一覧を表示します。

```
$ tkn pipeline list
```

5.3.4.5. pipeline logs

特定の Pipeline のログを表示します。

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

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

5.3.4.6. pipeline start

Pipeline を起動します。

例: **mypipeline Pipeline** を起動します。

```
$ tkn pipeline start mypipeline
```

5.3.5. Pipeline 実行コマンド

5.3.5.1. pipelinerun

Pipeline 実行を管理します。

例: ヘルプの表示

```
$ tkn pipelinerun -h
```

5.3.5.2. pipelinerun cancel

Pipeline 実行をキャンセルします。

例: **namespace** からの **mypipelinerun Pipeline** 実行を取り消します。

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

5.3.5.3. pipelinerun delete

Pipeline 実行を削除します。

例: **namespace** からの **Pipeline** 実行を削除します。

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

例: 最近実行された 5 つの **Pipeline** 実行を除き、**namespace** からすべての **Pipeline** 実行を削除します。

```
$ tkn pipelinerun delete -n myspace --keep 5 (1)
```

(1) **5** を、保持する最近実行された Pipeline 実行の数に置き換えます。

5.3.5.4. pipelinerun describe

Pipeline 実行を記述します。

例: **namespace** での **mypipelinerun Pipeline** 実行を記述します。

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

5.3.5.5. pipelinerun list

Pipeline 実行を一覧表示します。

例: **namespace** での **Pipeline** 実行の一覧を表示します。

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

5.3.5.6. pipelinerun logs

Pipeline 実行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 Pipeline** 実行のログを表示します。

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

5.3.6. タスク管理コマンド

5.3.6.1. task

タスクを管理します。

例: ヘルプの表示

```
$ tkn task -h
```

5.3.6.2. task delete

タスクを削除します。

例: namespace からの **mytask1** および **mytask2** タスクを削除します。

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

5.3.6.3. task describe

タスクを記述します。

例: namespace の **mytask** タスクを記述します。

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

5.3.6.4. task list

タスクを一覧表示します。

例: namespace のすべてのタスクを一覧表示します。

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

5.3.6.5. task logs

タスクログを表示します。

例: **mytask** タスクの **mytaskrun** タスク実行のログを表示します。

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

5.3.6.6. task start

タスクを開始します。

例: namespace の **mytask** タスクを開始します。

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

5.3.7. タスク実行コマンド

5.3.7.1. taskrun

タスク実行を管理します。

例: ヘルプの表示

```
$ tkn taskrun -h
```

5.3.7.2. taskrun cancel

タスク実行をキャンセルします。

例: **namespace** からの **mytaskrun** タスク実行を取り消します。

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

5.3.7.3. taskrun delete

TaskRun を削除します。

例: **namespace** からの **mytaskrun1** および **mytaskrun2** タスク実行を削除します。

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

例: **namespace** から最近実行された **5** つのタスク以外のすべてのタスクを削除します。

```
$ tkn taskrun delete -n myspace --keep 5 1
```

1 **5** を、保持する最近実行したタスク実行の数に置き換えます。

5.3.7.4. taskrun describe

タスク実行を記述します。

例: **namespace** での **mytaskrun** タスク実行を記述します。

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

5.3.7.5. taskrun list

タスク実行を一覧表示します。

例: **namespace** のすべてのタスク実行を一覧表示します。

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

5.3.7.6. taskrun logs

タスク実行ログを表示します。

例: **namespace** での **mytaskrun** タスク実行のライブログを表示します。

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

5.3.8. 条件管理コマンド

5.3.8.1. condition

条件を管理します。

例: ヘルプの表示

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

5.3.8.2. 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
```

5.3.8.3. 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
```

5.3.8.4. 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
```

5.3.9. Pipeline リソース管理コマンド

5.3.9.1. resource

Pipeline リソースを管理します。

例: ヘルプの表示

```
$ tkn resource -h
```

5.3.9.2. 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。

5.3.9.3. 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
```

5.3.9.4. 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
```

5.3.9.5. 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
```

5.3.10. ClusterTask 管理コマンド

5.3.10.1. clustertask

ClusterTask を管理します。

例: ヘルプの表示

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

5.3.10.2. 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
```

5.3.10.3. clustertask describe

ClusterTask を記述します。

例: **mytask ClusterTask** の記述

```
$ tkn clustertask describe mytask1
```

5.3.10.4. clustertask list

ClusterTask を一覧表示します。

例: ClusterTask の一覧表示

```
$ tkn clustertask list
```

5.3.10.5. clustertask start

ClusterTask を開始します。

例: **mytask ClusterTask** の開始

```
$ tkn clustertask start mytask
```

5.3.11. 管理コマンドのトリガー

5.3.11.1. eventlistener

EventListener を管理します。

例: ヘルプの表示

```
$ tkn eventlistener -h
```

5.3.11.2. 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
```

5.3.11.3. eventlistener describe

EventListener を記述します。

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

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

5.3.11.4. 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
```

5.3.11.5. 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
```

5.3.11.6. triggerbinding

TriggerBinding を管理します。

例: TriggerBindings ヘルプの表示

```
$ tkn triggerbinding -h
```

5.3.11.7. 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
```

5.3.11.8. triggerbinding describe

TriggerBinding を記述します。

例: namespace の mybinding TriggerBinding の記述

```
$ tkn triggerbinding describe mybinding -n myspace
```

5.3.11.9. 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
```

5.3.11.10. triggertemplate

TriggerTemplate を管理します。

例: TriggerTemplate ヘルプの表示

```
$ tkn triggertemplate -h
```

5.3.11.11. 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`
```

5.3.11.12. triggertemplate describe

TriggerTemplate を記述します。

例: namespace の **mytemplate TriggerTemplate** の記述

```
$ tkn triggertemplate describe mytemplate -n `myspace`
```

5.3.11.13. 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
```

5.3.11.14. clustertriggerbinding

ClusterTriggerBinding を管理します。

例: ClusterTriggerBinding のヘルプの表示

```
$ tkn clustertriggerbinding -h
```

5.3.11.15. clustertriggerbinding delete

ClusterTriggerBinding を削除します。

例: **myclusterbinding1** および **myclusterbinding2 ClusterTriggerBinding** の削除

```
$ tkn clustertriggerbinding delete myclusterbinding1 myclusterbinding2
```

5.3.11.16. clustertriggerbinding describe

ClusterTriggerBinding を記述します。

例: **myclusterbinding ClusterTriggerBinding** の記述

```
$ tkn clustertriggerbinding describe myclusterbinding
```

5.3.11.17. 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
```

5.3.12. 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。

5.3.12.1. 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。

5.3.12.2. 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
```

5.3.12.3. 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
```

5.3.12.4. 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
```

5.3.12.5. 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
```

5.3.12.6. 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
```

5.3.12.7. 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
```

5.3.12.8. 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6章 OPM CLI

6.1. 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。

追加リソース

- 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。

6.2. 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"}
```

6.3. 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7章 OPERATOR SDK

7.1. 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 4.8 以降は Operator SDK v1.8.0 をサポートします。

7.1.1. 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 v1.16+
- **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. **4.8.4** 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-v1.8.0-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: "v1.8.0-ocp", ...
```

7.2. 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。

7.2.1. 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。

7.2.1.1. validate

bundle validate サブコマンドは Operator バンドルを検証します。

表7.1 **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。

7.2.2. 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7.2 **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。

7.2.3. 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7.3 **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7.4 **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
```

7.2.4. 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。

7.2.4.1. 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7.5 **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。

7.2.5. 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。

7.2.5.1. 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7.6 **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。

7.2.5.2. 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。

7.2.5.2.1. 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7.7 **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。

7.2.6. 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 メトリック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7.8 **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。

7.2.7. 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。

7.2.7.1. 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7.9 **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。

7.2.7.2. 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7.10 **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。

7.2.8. 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7.11 **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、スコアカードを使用した Operator の検証 を参照してください。