OpenShift Container Platform 4.1

CLI リファレンス

OpenShift CLI の使用方法について

Last Updated: 2020-05-15

OpenShift Container Platform 4.1 CLI リファレンス

OpenShift CLI の使用方法について

法律上の通知

概要

本書では、OpenShift CLI (oc) 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 CLI の使用方法

1.1. CLI について

OpenShift Container Platform のコマンドラインインターフェース (CLI) 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。CLI の使用は、以下の場合に適しています。

- プロジェクトのソースコードを直接使用している。
- OpenShift Container Platform の操作をスクリプト化する。
- 帯域幅リソースの制限下にあり、Web コンソールを使用できない。

1.2. CLI のインストール

コマンドラインインターフェースを使用して OpenShift Container Platform と対話するために CLI をインストールすることができます。

手順

1. Red Hat OpenShift Cluster Manager サイトの Infrastructure Provider ページから、選択するインストールタイプのページに移動し、**Download Command-line Tools** をクリックします。
2. オペレーティングシステムおよびアーキテクチャーのフォルダーをクリックしてから、圧縮されたファイルをクリックします。
3. ファイルをファイルシステムに保存します。
4. 圧縮ファイルを展開します。
5. これを **PATH** にあるディレクトリーに配置します。

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>
```

1.3. CLI へのログイン

oc CLI 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あること。
- CLI をインストールしていること。

手順

- **oc login** コマンドを使用して CLI にログインし、プロンプトが出されたら必要な情報を入力します。

  ```
  $ oc login
  ```

```
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 (3)
Password: (4)
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. ログインに使用するユーザー名を入力します。
4. ユーザーのパスワードを入力します。

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。

1.4. 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。

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

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

...

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

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。

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

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

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

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、DeploymentConfig、および BuildConfig 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
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.
```

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
NAME                              SHORTNAMES   APIGROUP                       NAMESPACED   KIND
bindings                                                                      true         Binding
```

```
componentstatuses                 cs                                     false        ComponentStatus
configmaps                        cm                                     true         ConfigMap
...
```

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
  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
  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
  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

...
```

1.6. CLI からのログアウト

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
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。

第2章 CLI の設定

2.1. タブ補完の有効化

oc 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 **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。

前提条件

- **oc** CLI ツールをインストールしていること。

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。

```
$ 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。

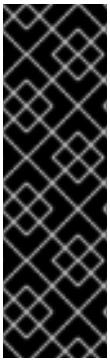
タブ補完は、新規ターミナルを開くと有効にされます。

第3章 プラグインによる 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。

3.1. CLI プラグインの作成

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



重要

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

詳細は、「テクノロジープレビュー機能のサポート範囲」を参照してください。

手順

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

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

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

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

2. 以下の内容をファイルに追加します。

```
#!/bin/bash

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

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

```
fi

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

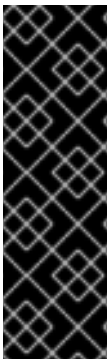
OpenShift Container Platform CLI のこのプラグインをインストールした後に、**oc foo** コマンドを使用してこれを起動できます。

追加リソース

- 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。

3.2. CLI プラグインのインストールおよび使用

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



重要

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

詳細は、「テクノロジープレビュー機能のサポート範囲」を参照してください。

前提条件

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

手順

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

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

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

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

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

```
$ oc plugin list
The following compatible plugins are available:

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

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

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

```
$ oc ns
```

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

第4章 開発者の CLI コマンド

4.1. 基本的な CLI コマンド

4.1.1. explain

特定リソースのドキュメントを表示します。

例: Pod のドキュメントの表示

```
$ oc explain pods
```

4.1.2. login

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

例: 対話型ログイン

```
$ oc login
```

例: ユーザー名を指定したログイン

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

4.1.3. new-app

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

例: ローカル Git リポジトリーからの新規アプリケーションの作成

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

例: リモート Git リポジトリーからの新規アプリケーションの作成

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

例: プライベートリモートリポジトリーからの新規アプリケーションの作成

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

4.1.4. new-project

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

例: 新規プロジェクトの作成

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

4.1.5. project

別のプロジェクトに切り替えて、これを設定でデフォルトにします。

例: 別のプロジェクトへの切り替え

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

4.1.6. projects

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

例: すべてのプロジェクトの一覧表示

```
$ oc projects
```

4.1.7. status

現在のプロジェクトのハイレベルの概要を表示します。

例: 現在のプロジェクトのステータスの表示

```
$ oc status
```

4.2. CLI コマンドのビルドおよびデプロイ

4.2.1. cancel-build

実行中、保留中、または新規のビルドを取り消します。

例：ビルドの取り消し

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

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

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

4.2.2. import-image

イメージリポジトリーから最新のタグおよびイメージ情報をインポートします。

例: 最新のイメージ情報のインポート

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

4.2.3. new-build

ソースコードから新規の **BuildConfig** を作成します。

例: ローカル Git リポジトリーからの BuildConfig の作成

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

例: リモート Git リポジトリーからの BuildConfig の作成

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

4.2.4. rollback

アプリケーションを以前のデプロイメントに戻します。

例: 最後に成功したデプロイメントへのロールバック

```
$ oc rollback php
```

例: 特定バージョンへのロールバック

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

4.2.5. rollout

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

例: 最後に成功したデプロイメントへのロールバック

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

例: 最新状態の DeploymentConfig の新規ロールアウトの開始

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

4.2.6. start-build

BuildConfig からビルドを開始するか、または既存ビルドをコピーします。

例: 指定された BuildConfig からのビルドの開始

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

例: 以前のビルドからのビルドの開始

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

例: 現在のビルドに使用する環境変数の設定

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

4.2.7. tag

既存のイメージをイメージストリームにタグ付けします。

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

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

4.3. アプリケーション管理 CLI コマンド

4.3.1. annotate

1 つ以上のリソースでアノテーションを更新します。

例: アノテーションのルートへの追加

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

例: ルートからのアノテーションの削除

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

4.3.2. apply

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

例: pod.json の設定の Pod への適用

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

4.3.3. autoscale

DeploymentConfig または ReplicationController の自動スケーリングを実行します。

例: 最小の 2 つおよび最大の 5 つの Pod への自動スケーリング

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

4.3.4. create

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

例: pod.json の内容を使用した Pod の作成

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

4.3.5. delete

リソースを削除します。

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

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

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

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

4.3.6. describe

特定のオブジェクトに関する詳細情報を返します。

例: example という名前のデプロイメントの記述

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

例: すべての Pod の記述

```
$ oc describe pods
```

4.3.7. edit

リソースを編集します。

例: デフォルトエディターを使用した DeploymentConfig の編集

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

例: 異なるエディターを使用した DeploymentConfig の編集

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

例: JSON 形式の DeploymentConfig の編集

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

4.3.8. expose

ルートとしてサービスを外部に公開します。

例: サービスの公開

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

例: サービスの公開およびホスト名の指定

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

4.3.9. get

1 つ以上のリソースを表示します。

例: default namespace の Pod の一覧表示

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

例: JSON 形式の python DeploymentConfig についての詳細の取得

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

4.3.10. label

1 つ以上のリソースでアノテーションを更新します。

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

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

4.3.11. scale

ReplicationController または DeploymentConfig の必要なレプリカ数を設定します。

例: ruby-app DeploymentConfig の 3 つの Pod へのスケーリング

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

4.3.12. secrets

プロジェクトのシークレットを管理します。

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

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

4.3.13. serviceaccounts

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

例: default サービスアカウントに割り当てられたトークンの取得

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

4.3.14. set

既存のアプリケーションリソースを設定します。

例: BuildConfig でのシークレットの名前の設定

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

4.4. CLI コマンドのトラブルシューティングおよびデバッグ

4.4.1. attach

実行中のコンテナーにシェルを割り当てます。

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

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

4.4.2. cp

ファイルおよびディレクトリーのコンテナーへの/からのコピーを実行します。

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

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

4.4.3. debug

コマンドシェルを起動して、実行中のアプリケーションをデバッグします。

例: python デプロイメントのデバッグ

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

4.4.4. exec

コンテナーでコマンドを実行します。

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

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

4.4.5. logs

特定のビルド、BuildConfig、DeploymentConfig、または Pod のログ出力を取得します。

例: python DeploymentConfig からの最新ログのストリーミング

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

4.4.6. port-forward

1 つ以上のポートを Pod に転送します。

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

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

4.4.7. proxy

Kubernetes API サーバーに対してプロキシーを実行します。

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

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

4.4.8. rsh

コンテナーへのリモートシェルセッションを開きます。

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

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

4.4.9. rsync

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

例: ローカルディレクトリーのファイルの Pod ディレクトリーとの同期

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

4.4.10. run

特定のイメージを作成し、実行します。デフォルトでは、これにより作成されたコンテナーを管理するための DeploymentConfig が作成されます。

例: 3 つのレプリカを持つ perl イメージのインスタンスの開始

```
$ oc run my-test --image=perl --replicas=3
```

4.4.11. wait

1 つ以上のリソースの特定の条件を待機します。

例: python-1-mz2rf Pod の削除の待機

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

4.5. 上級開発者の CLI コマンド

4.5.1. api-resources

サーバーがサポートする API リソースの詳細の一覧を表示します。

例: サポートされている API リソースの一覧表示

```
$ oc api-resources
```

4.5.2. api-versions

サーバーがサポートする API バージョンの詳細の一覧を表示します。

例: サポートされている API バージョンの一覧表示

```
$ oc api-versions
```

4.5.3. auth

パーミッションを検査し、RBAC ロールを調整します。

例: 現行ユーザーが Pod ログを読み取ることができるかどうかのチェック

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

例: ファイルの RBAC ロールおよびパーミッションの調整

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

4.5.4. cluster-info

マスターおよびクラスターサービスのアドレスを表示します。

例: クラスター情報の表示

```
$ oc cluster-info
```

4.5.5. convert

YAML または JSON 設定ファイルを異なる API バージョンに変換し、標準出力 (stdout) に出力します。

例: pod.yaml の最新バージョンへの変換

```
$ oc convert -f pod.yaml
```

4.5.6. extract

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

例: ruby-1-ca ConfigMap の内容の現行ディレクトリーへのダウンロード

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

例: ruby-1-ca ConfigMap の内容の標準出力 (stdout) への出力

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

4.5.7. idle

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

例: ruby-app サービスのアイドリング

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

4.5.8. image

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

例: イメージの別のタグへのコピー

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

4.5.9. observe

リソースの変更を監視し、それらの変更に対するアクションを取ります。

例: サービスへの変更の監視

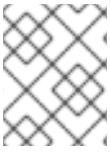
```
$ oc observe services
```

4.5.10. patch

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

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

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



注記

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

4.5.11. policy

認可ポリシーを管理します。

例: edit ロールの現在のプロジェクトの user1 への追加

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

4.5.12. process

リソースの一覧に対してテンプレートを処理します。

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

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

4.5.13. レジストリー

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

例: 統合レジストリーについての情報の表示

```
$ oc registry info
```

4.5.14. replace

指定された設定ファイルに基づいて既存オブジェクトを変更します。

例: pod.json の内容を使用した Pod の更新

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

4.6. CLI コマンドの設定

4.6.1. completion

指定されたシェルのシェル補完コードを出力します。

例: Bash の補完コードの表示

```
$ oc completion bash
```

4.6.2. config

クライアント設定ファイルを管理します。

例: 現在の設定の表示

```
$ oc config view
```

例: 別のコンテキストへの切り替え

```
-
```

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

4.6.3. logout

現行のセッションからログアウトします。

例: 現行セッションの終了

```
$ oc logout
```

4.6.4. whoami

現行セッションに関する情報を表示します。

例: 現行の認証ユーザーの表示

```
$ oc whoami
```

4.7. 他の開発者 CLI コマンド

4.7.1. help

CLI の一般的なヘルプ情報および利用可能なコマンドの一覧を表示します。

例: 利用可能なコマンドの表示

```
$ oc help
```

例: new-project コマンドのヘルプの表示

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

4.7.2. plugin

ユーザーの **PATH** に利用可能なプラグインを一覧表示します。

例: 利用可能なプラグインの一覧表示

```
$ oc plugin list
```

4.7.3. version

oc クライアントおよびサーバーのバージョンを表示します。

例: バージョン情報の表示

```
$ oc version
```

第5章 管理者 CLI コマンド

5.1. クラスター管理 CLI コマンド

5.1.1. must-gather

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

例: デバッグ情報の収集

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

5.1.2. top

サーバー上のリソースの使用状況についての統計を表示します。

例: Pod の CPU およびメモリーの使用状況の表示

```
$ oc adm top pods
```

例: イメージの使用状況の統計の表示

```
$ oc adm top images
```

5.2. ノード管理 CLI コマンド

5.2.1. cordon

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

例: node1 にスケジュール対象外のマークを付ける

```
$ oc adm cordon node1
```

5.2.2. drain

メンテナンスの準備のためにノードをドレイン (解放) します。

例: node1 のドレイン (解放)

```
$ oc adm drain node1
```

5.2.3. node-logs

ノードのログを表示し、フィルターします。

例: NetworkManager のログの取得

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

5.2.4. taint

1 つ以上のノードでテイントを更新します。

例: ユーザーのセットに対してノードを専用に割り当てるためのテイントの追加

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

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

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

5.2.5. uncordon

ノードにスケジュール対象 (schedulable) のマークを付けます。

例: node1 にスケジュール対象のマークを付ける

```
$ oc adm uncordon node1
```

5.3. セキュリティーおよびポリシー CLI コマンド

5.3.1. certificate

証明書署名要求 (CSR) を承認するか、または拒否します。

例: CSR の承認

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

5.3.2. groups

クラスター内のグループを管理します。

例: 新規グループの作成

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

5.3.3. new-project

新規プロジェクトを作成し、管理オプションを指定します。

例: ノードセレクターを使用した新規プロジェクトの作成

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

5.3.4. pod-network

クラスター内の Pod ネットワークを管理します。

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

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

5.3.5. policy

クラスター上のロールおよびポリシーを管理します。

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

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

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

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

5.4. メンテナンス CLI コマンド

5.4.1. migrate

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

例: 保存されたすべてのオブジェクトの更新の実行

```
$ oc adm migrate storage
```

例: Pod のみの更新の実行

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

5.4.2. prune

サーバーから古いバージョンのリソースを削除します。

例: BuildConfigs がすでに存在しないビルドを含む、古いビルドのプルーニング

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

5.5. 設定 CLI コマンド

5.5.1. create-api-client-config

サーバーに接続するためのクライアント接続を作成します。これにより、指定されたユーザーとしてマスターに接続するためのクライアント証明書、クライアントキー、サーバーの認証局、および **kubeconfig** ファイルが含まれるフォルダーが作成されます。

例: プロキシーのクライアント証明書の生成

```
$ oc adm create-api-client-config \
  --certificate-authority='/etc/origin/master/proxyca.crt' \
  --client-dir='/etc/origin/master/proxy' \
  --signer-cert='/etc/origin/master/proxyca.crt' \
  --signer-key='/etc/origin/master/proxyca.key' \
  --signer-serial='/etc/origin/master/proxyca.serial.txt' \
  --user='system:proxy'
```

5.5.2. create-bootstrap-policy-file

デフォルトのブートストラップポリシーを作成します。

例: デフォルトブートストラップポリシーでの policy.json ファイルの作成

```
$ oc adm create-bootstrap-policy-file --filename=policy.json
```

5.5.3. create-bootstrap-project-template

ブートストラッププロジェクトテンプレートを作成します。

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

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

5.5.4. create-error-template

エラーページをカスタマイズするためのテンプレートを作成します。

例: エラーページのテンプレートの標準出力 (stdout) への出力

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

5.5.5. create-kubeconfig

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

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

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

5.5.6. create-login-template

ログインページをカスタマイズするためのテンプレートを作成します。

例: ログインページのテンプレートの標準出力 (stdout) への出力

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

5.5.7. create-provider-selection-template

プロバイダー選択ページをカスタマイズするためのテンプレートを作成します。

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

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

5.6. 他の管理者 CLI コマンド

5.6.1. build-chain

ビルドの入力と依存関係を出力します。

例: perl イメージストリームの依存関係の出力

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

5.6.2. completion

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

例: Bash の oc adm 補完コードの表示

```
$ oc adm completion bash
```

5.6.3. config

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

例: 現在の設定の表示

```
$ oc adm config view
```

例: 別のコンテキストへの切り替え

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

5.6.4. release

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

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

```
$ oc adm release info --changelog=/tmp/git \
    quay.io/openshift-release-dev/ocp-release:4.1.0-rc.7 \
    quay.io/openshift-release-dev/ocp-release:4.1.0 \
    > changelog.md
```

5.6.5. verify-image-signature

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

例: nodejs イメージ署名の検証

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

第6章 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ース (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。

6.1. 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**
  DeploymentConfigs、BuildConfigs、Routes、ImageStreams、および ImageStreamTags 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、「Understanding authentication」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。

6.2. 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** の既存ユーザーは引き続きバイナリーを使用し、OpenShift Container Platform クラスターに必要な変更なしに Kubernetes のプリミティブと対話できます。

詳細は kubectl ドキュメントを参照してください。

---------- 最終更新日: 2020-05-07 ----------