OpenShift Container Platform 3.11

CLI リファレンス

OpenShift Container Platform 3.11 CLI リファレンス

Last Updated: 2020-12-18

OpenShift Container Platform 3.11 CLI リファレンス

OpenShift Container Platform 3.11 CLI リファレンス

法律上の通知

概要

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 のプロジェクトを管理できます。以下のトピックでは、CLI の使用方法について説明します。

目次

第1章 概要

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 の プロジェクト を管理できます。CLI の使用は、以下のような場合に適しています。

- 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。
- 帯域幅のリソースによる制限があり、Web コンソール を使用できない。

以下の **oc** コマンドを使って、CLI を利用できます。

```
$ oc <command>
```

インストールと設定に関する説明は、「CLI の使用方法」を参照してください。

第2章 CLI の使用方法

2.1. 概要

OpenShift Container Platform の CLI は、アプリケーションを管理するためのコマンドだけでなく、システムの各コンポーネントと対話する低レベルのツールも公開しています。このトピックでは、インストールおよび最初のプロジェクトを作成するためのログインなど、CLI の使用方法について説明します。

2.2. 前提条件

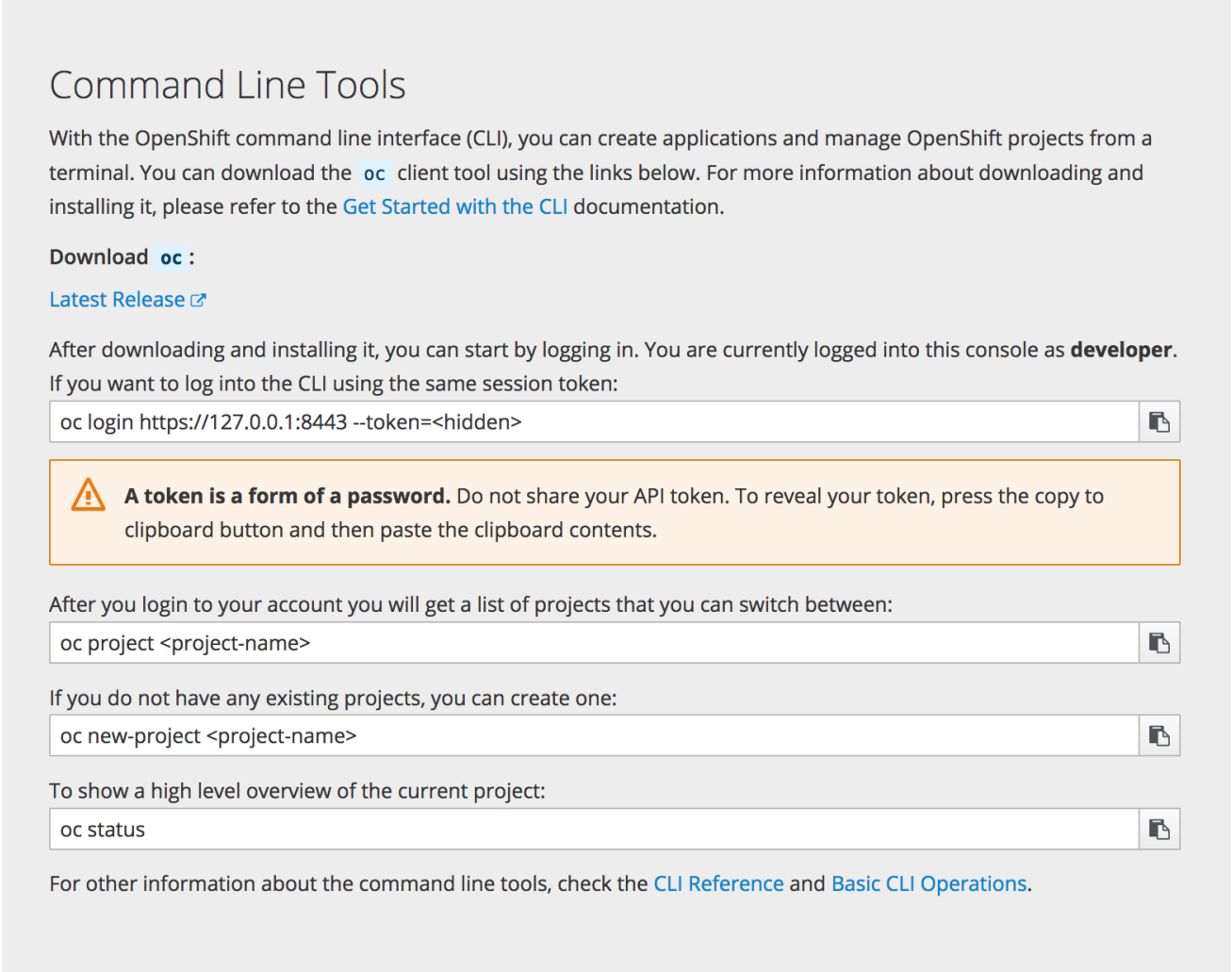
特定の操作では、Git がクライアント上にローカルにインストールされている必要があります。たとえば、リモート Git リポジトリーを使用してアプリケーションを作成するコマンドは以下のようになります。

```
$ oc new-app https://github.com/<your_user>/<your_git_repo>
```

ここで、Git をワークステーションにインストールします。お使いのワークステーションのオペレーティングシステムに応じたインストール方法については、Git の公式サイトの Git ドキュメント を参照してください。

2.3. CLI のインストール

CLI をダウンロードする最も簡単な方法として、Web コンソールの **About** ページにアクセスすることができます (クラスター管理者がダウンロードリンクを有効にしている場合)。



CLI のインストールオプションは、お使いのオペレーティングシステムによって異なります。

CLI を使用してログインする場合、Web コンソールの「**Command Line**」ページからトークンを取得します。このページには、「**Help**」メニューの「**Command Line Tools**」からアクセスできます。トークンは非表示になっているので、「**Command Line Tools**」ページにある **oc login** 行の端にある「**copy to clipboard**」ボタンをクリックし、コピーしたコンテンツを貼り付け、トークンを表示します。

2.3.1. Windows の場合

Windows 向けの CLI は、**zip** アーカイブとして提供されます。これは、Red Hat カスタマーポータル からダウンロードできます。Red Hat アカウントでログイン後、ダウンロードページにアクセスするには、有効な OpenShift Enterprise サブスクリプションが必要です。

Red Hat カスタマーポータルからの CLI のダウンロード

または、クラスター管理者がリンクを有効にしている場合は、Web コンソールの **About** ページから CLI をダウンロードし、展開することができます。

チュートリアルの動画

以下の動画ではこのプロセスを紹介しています。視聴するには、ここをクリックしてください。

次に ZIP プログラムでアーカイブを展開し、**oc** バイナリーを PATH 上の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を実行します。

```
C:\> path
```

2.3.2. Mac OS X の場合

Mac OS X 向けの CLI は、**tar.gz** アーカイブとして提供されます。これは Red Hat カスタマーポータルからダウンロードできます。Red Hat アカウントでログイン後、ダウンロードページにアクセスするには、有効な OpenShift Enterprise サブスクリプションが必要です。

Red Hat カスタマーポータルからの CLI のダウンロード

または、クラスター管理者がリンクを有効にしている場合は、Web コンソールの **About** ページから CLI をダウンロードし、展開することができます。

チュートリアルの動画

以下の動画ではこのプロセスを紹介しています。視聴するには、ここをクリックしてください。

次にアーカイブを展開し、**oc** バイナリーを PATH 上のディレクトリーに移動します。PATH を確認するには、ターミナルのウィンドウを開いて以下を実行します。

```
$ echo $PATH
```

2.3.3. Linux の場合

Red Hat Enterprise Linux (RHEL) 7 の場合、Red Hat アカウントに有効な OpenShift Enterprise サブスクリプションがある場合、Red Hat Subscription Management (RHSM) を使用して RPM として CLI をインストールできます。

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. 登録したシステムにサブスクリプションを割り当てます。

```
# subscription-manager attach --pool=<pool_id> 1
```

1 有効な OpenShift Enterprise サブスクリプションのプール ID

4. OpenShift Container Platform 3.11 で必要なリポジトリーを有効にします。

```
# subscription-manager repos --enable="rhel-7-server-ose-3.11-rpms"
```

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

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

RHEL、Fedora その他 Linux ディストリビューションの場合、 CLI を Red Hat カスタマーポータル から **tar.gz** アーカイブとして直接ダウンロードできます。Red Hat アカウントでログイン後、ダウンロードページにアクセスするには、有効な OpenShift Enterprise サブスクリプションが必要です。

Red Hat カスタマーポータルからの CLI のダウンロード

チュートリアルの動画

以下の動画ではこのプロセスを紹介しています。視聴するには、ここをクリックしてください。

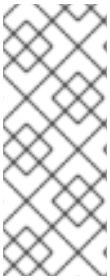
または、クラスター管理者がリンクを有効にしている場合は、Web コンソールの **About** ページから CLI をダウンロードし、展開することができます。

次にアーカイブを展開し、**oc** バイナリーを PATH 上のディレクトリーに移動します。PATH を確認するには、以下を実行します。

```
$ echo $PATH
```

アーカイブを展開するには、以下を実行します。

```
$ tar -xf <file>
```



注記

RHEL または Fedora を使用しない場合は、**libc** がライブラリーパスのディレクトリーにインストールされていることを確認してください。**libc** が利用できない場合は、CLI コマンドの実行時に以下のエラーが表示されます。

```
oc: No such file or directory
```

2.4. 基本的な設定およびログイン

CLI の初期設定を行う場合、**oc login** コマンドを使用することが最適な方法となり、このコマンドはほとんどのユーザーにとってのエントリーポイントとしての機能を果たします。対話型フローは、指定される認証情報を使用して OpenShift Container Platform サーバーへのセッションを確立するのに役立ち

ます。この情報は CLI 設定ファイル に自動的に保存され、その後のコマンドで使用されます。

以下の例では、**oc login** コマンドを使用した対話的な設定およびログインについて説明します。

例2.1 CLI の初期設定

```
$ oc login
```

出力例

```
OpenShift server [https://localhost:8443]: https://openshift.example.com (1)

Username: alice (2)
Authentication required for https://openshift.example.com (openshift)
Password: ******
Login successful. (3)

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

    $ oc new-project <projectname> (4)

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

1. このコマンドは、OpenShift Container Platform サーバー URL を求めるプロンプトを出します。
2. このコマンドは、ログイン認証情報 (ユーザー名とパスワード) を求めるプロンプトを出します。
3. セッションがサーバーで確立され、セッショントークンが受信されます。
4. プロジェクトがない場合は、プロジェクトの作成方法に関する情報が提供されます。

CLI 設定が完了すると、その後のコマンドがサーバーの設定ファイル、セッショントークン、およびプロジェクト情報を使用します。

CLI からログアウトするには、以下の **oc logout** コマンドを使用します。

```
$ oc logout
```

出力例

```
User, alice, logged out of https://openshift.example.com
```

プロジェクトの作成後か、またはプロジェクトへのアクセスが付与された後にログインする場合、アクセス可能なプロジェクトが現在のデフォルトとして自動的に設定されます。これは 別のプロジェクトに切り替える までデフォルトになります。

```
$ oc login
```

出力例

```
Username: alice
Authentication required for https://openshift.example.com (openshift)
Password:
Login successful.

Using project "aliceproject".
```

oc login コマンドでは、追加のオプション も利用可能です。

注記

管理者の認証情報がある場合でも デフォルトシステムユーザー の **system:admin** としてログインしていない場合、認証情報が CLI 設定ファイル にある限り、いつでもこのユーザーとしてログインし直すことができます。以下のコマンドはログインを実行し、**デフォルト** のプロジェクトに切り替えます。

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

2.5. CLI 設定ファイル

CLI 設定ファイルは、**oc** オプションを永続的に保存します。またこのファイルには、一連の 認証メカニズムおよびニックネームに関連付けられた OpenShift Container Platform サーバー接続情報が含まれます。

前のセクションで説明したように、**oc login** コマンドは、CLI 設定ファイルを自動的に作成し、管理します。このコマンドで収集されるすべての情報は、**~/.kube/config** にある設定ファイルに保存されます。現在の CLI 設定は、以下のコマンドを使用して表示することができます。

例2.2 CLI 設定の表示

```
$ oc config view
```

出力例

```
apiVersion: v1
clusters:
- cluster:
    server: https://openshift.example.com
  name: openshift
contexts:
- context:
    cluster: openshift
    namespace: aliceproject
    user: alice
  name: alice
current-context: alice
kind: Config
preferences: {}
users:
- name: alice
  user:
    token: NDM2N2MwODgtNjI1Yy10N3VhLTg1YmItYzI4NDEzZDUyYzVi
```

CLI 設定ファイルは、さまざまな OpenShift Container Platform サーバー、namespace、およびユーザーを使用して「複数の CLI プロファイルを設定」するために使用できます。これらの設定ファイルは、コマンドラインで指定した上書きオプションと共に、ランタイム時に読み込まれ、マージされます。

2.6. プロジェクト

OpenShift Container Platform の プロジェクト には、論理的なアプリケーションを構成する複数の オブジェクト が含まれます。

ほとんどの **oc** コマンドは、プロジェクト のコンテキストで実行されます。**oc login** は 初期設定時に、その後のコマンドで使用するデフォルトのプロジェクトを選択します。以下のコマンドを使用して、現在使用中のプロジェクトを表示します。

```
$ oc project
```

複数のプロジェクトにアクセスできる場合は、プロジェクト名を指定し、以下の構文を使って特定のプロジェクトに切り替えます。

```
$ oc project <project_name>
```

例:

Switch to Project project02

```
$ oc project project02
```

出力例

```
Now using project 'project02'.
```

プロジェクト project03 に切り替えます。

```
$ oc project project03
```

出力例

```
Now using project 'project03'.
```

List the Current Project

```
$ oc project
```

出力例

```
Using project 'project03'.
```

oc status コマンドは、現在使用中のプロジェクトのハイレベルの概要を表示します。以下の例が示すように、プロジェクトのコンポーネントおよび関係性なども表示されます。

```
$ oc status
```

-

出力例

```
In project OpenShift 3 Sample (test)

service database-test (172.30.17.113:6434 -> 3306)
  database-test deploys docker.io/library/mysql:latest
    #1 deployed 47 hours ago

service frontend-test (172.30.17.236:5432 -> 8080)
  frontend-test deploys origin-ruby-sample:test <-
    builds https://github.com/openshift/ruby-hello-world with docker.io/openshift/ruby-20-centos7:latest
    not built yet
    #1 deployment waiting on image

To see more information about a service or deployment config, use 'oc describe service <name>' or
'oc describe dc <name>'.
You can use 'oc get pods,svc,dc,bc,builds' to see lists of each of the types described above.
```

2.7. 次のステップ

ログイン後、新規のアプリケーションを作成し、一般的な CLI 操作をいくつか試すことができます。

第3章 CLI プロファイルの管理

3.1. 概要

CLI 設定ファイルにより、OpenShift CLI で使用する様々なプロファイルまたは **コンテキスト** の設定が可能になります。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 サーバー情報から構成されます。

3.2. CLI プロファイル間の切り替え

CLI 実行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** での複数ユーザーの切り替えが簡単になります。ニックネームで、コンテキスト、ユーザーの認証情報およびクラスター情報の省略された参照情報を提供することで、CLI 設定の管理が簡単になります。

初回の CLI でのログイン 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)。追加の認証および接続の詳細情報が **oc login** 操作時に自動的に、または 明示的な設定によって CLI に提供されると、更新情報は設定ファイルに保存されます。

例3.1 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) **クラスター** 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について定義します。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 **コンテキスト** セクションは、2 つのコンテキストを定義します。1 つは、ニックネームが alice-project/openshift1.example.com:8443/alice 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 つはニックネームが joe-project/openshift1.example.com:8443/alice 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** のパラメーターは、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、openshift1.example.com:8443 クラスターの joe-project プロジェクトで作業することが可能になります。

4 **ユーザー** 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の alice/openshift1.example.com:8443 が、アクセストークン を使用します。

CLI は複数の設定ファイルをサポートできます。これらの設定ファイルは、コマンドラインで指定した上書きオプションと共に、ランタイム時に読み込まれ、マージされます。

ログイン後、**oc status** コマンドまたは **oc project** コマンドを使用して、現在稼働中の環境を確認できます。

例3.2 稼働中の環境の確認

```
$ 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 above.
```

List the Current Project

```
$ 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".
```

ユーザー認証情報およびクラスター詳細の組み合わせを使用してログインするには、**oc login** コマンドを再度実行し、対話プロセスで関連情報を指定します。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たい場合は、**oc project** コマンドを使用してプロジェクト名を指定します。

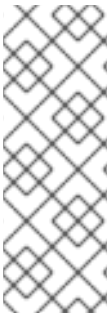
```
$ oc project alice-project
```

出力例

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

oc config view コマンドを使用すると、出力に示されるような現在の CLI 設定全体をいつでも表示することができます。

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

管理者の認証情報がある場合でも デフォルトシステムユーザー の **system:admin** としてログインしていない場合、認証情報が CLI 設定ファイル にある限り、いつでもこのユーザーとしてログインし直すことができます。以下のコマンドはログインを実行し、**デフォルト** のプロジェクトに切り替えます。

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

3.3. 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** コマンドと **oc project** コマンドを使用するだけで、ログインやコンテキスト間および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場合に、ファイル自体を変更する代わりに **oc config** コマンドを使用できます。**oc config** コマンドには、この手動設定に役立つ多数のサブコマンドが含まれています。

表3.1 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]`

サブコマンド	使用法
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>
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、各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る。ここで、各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

以下の設定ワークフローを見てみましょう。まず、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>
```

変更が有効になっていることを確認するには、現在のコンテキストを確認します。

```
$ oc whoami -c
```

上書きする CLI オプションで指定されるか、またはコンテキストが切り替えられない限り、後続のすべての CLI 操作は新規のコンテキストを使用します。

3.4. 読み込みおよびマージのルール

CLI 操作を実行する際、CLI 設定の読み込みおよびマージの順序は、以下のルールに従います。

1. CLI 設定ファイルは、以下の階層および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。フラグが一度だけ設定される可能性があり、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、これを使用します。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。
2. 使用するコンテキストは、以下のチェーンの最初のヒット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。

3. 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、以下のチェーンの最初のヒットに基づいて作成されます。これは、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** オプションおよびクラスター名の **--cluster** オプションの値。
   - **--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。

4. 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。それぞれのクラスター情報は、以下のチェーンの最初のヒットに基づいて作成されます。
   - 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。

5. 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**

6. 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。

第4章 開発者 CLI の各種操作

4.1. 概要

このトピックでは、開発者 CLI の各種操作およびそれらの構文に関する情報を提供します。これらの操作を実行する前に、CLI を使用して 設定およびログインしている必要があります。

oc コマンドを使用する開発者 CLI はプロジェクトレベルの操作で使用されます。これは管理者 CLI とは異なります。管理者 CLI では、より高度な管理者操作に **oc adm** コマンドを使用します。

4.2. 一般的な操作

開発者 CLI は、OpenShift Container Platform で管理される各種オブジェクトとの対話を許可します。以下の構文を使用して、多くの一般的な **oc** 操作が呼び出されます。

```
$ oc <action> <object_type> <object_name>
```

これにより、以下が指定されます。

- **get** または **describe** などの実行する **<action>**。
- **service** または **svc** (省略形) などのアクションを実行する **<object_type>**。
- 指定した **<object_type>** の **<object_name>**。

たとえば、**oc get** 操作は、現在定義されているサービスの完全な一覧を返します。

```
$ oc get svc
```

出力例

```
NAME              LABELS                                SELECTOR                IP              PORT(S)
docker-registry   docker-registry=default               docker-registry=default   172.30.78.158
5000/TCP
kubernetes        component=apiserver,provider=kubernetes   <none>                172.30.0.2
443/TCP
kubernetes-ro     component=apiserver,provider=kubernetes   <none>                172.30.0.1
80/TCP
```

次に **oc describe** 操作を使用して、特定のオブジェクトに関する詳細情報を返すことができます。

```
$ oc describe svc docker-registry
```

出力例

```
Name:   docker-registry
Labels:   docker-registry=default
Selector:  docker-registry=default
IP:   172.30.78.158
Port:   <unnamed> 5000/TCP
```

```
Endpoints:  10.128.0.2:5000
Session Affinity: None
No events.
```

4.3. オブジェクトタイプ

以下は、CLI がサポートする最も一般的なオブジェクトタイプの一覧です。これらの一部には省略された構文が含まれます。

オブジェクトタイプ	省略バージョン
Build	
BuildConfig	**bc**
DeploymentConfig	**dc**
Deployments	**deploy**
Event	**ev**
ImageStream	**is**
ImageStreamTag	**istag**
ImageStreamImage	**isimage**
Job	
CronJob (テクノロジープレビュー)	**cj**
LimitRange	**limits**
Node	
Pod	**po**
ResourceQuota	**quota**
ReplicationController	**rc**
ReplicaSet	**rs**
Secrets	
Service	**svc**

オブジェクトタイプ	省略バージョン
ServiceAccount	**sa**
StatefulSets	**sts**
PersistentVolume	**pv**
PersistentVolumeClaim	**pvc**

サーバーがサポートするリソースの詳細な一覧を把握しておく必要がある場合は、**oc api-resources** を使用します。

4.4. 基本的な CLI 操作

以下の表は、基本的な **oc** 操作と、それらの一般的な構文について説明しています。

4.4.1. types

OpenShift Container Platform の一部のコアとなるコンセプトの概要を表示します。

```
$ oc types
```

4.4.2. login

OpenShift Container Platform サーバーにログインします。

```
$ oc login
```

4.4.3. logout

現在のセッションを終了します。

```
$ oc logout
```

4.4.4. new-project

新規プロジェクトを作成します。

```
$ oc new-project <project_name>
```

4.4.5. new-app

現在のディレクトリー内のソースコードに基づいて新規アプリケーションを作成します。

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

リモートリポジトリー内のソースコードに基づいて新規アプリケーションを作成します。

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

プライベートリモートリポジトリー内のソースコードに基づいて新規アプリケーションを作成します。

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

4.4.6. status

現在のプロジェクトの概要を表示します。

```
$ oc status
```

4.4.7. project

別のプロジェクトに切り替えます。現在のプロジェクトを表示するには、オプションなしで実行します。アクセス可能なすべてのプロジェクトを表示するには、**oc projects** を実行します。

```
$ oc project <project_name>
```

4.5. アプリケーション変更操作

4.5.1. get

指定された「オブジェクトタイプ」のオブジェクトの一覧を返します。オプションの **<object_name>** が要求に含まれている場合には、結果の一覧はその値でフィルターされます。

```
$ oc get <object_type> [<object_name>]
```

出力形式を変更するには、**-o** または **--output** オプションを使用できます。

```
$ oc get <object_type> [<object_name>]-o|--output=json|yaml|wide|custom-columns=...|custom-
columns-file=...|go-template=...|go-template-file=...|jsonpath=...|jsonpath-file=...]
```

出力形式には、JSON や YAML、またはカスタムカラム、golang テンプレート、および jsonpath などの拡張性のある形式を使用できます。

たとえば、以下のコマンドは特定のプロジェクトで実行される Pod の名前を一覧表示します。

```
$ oc get pods -n default -o jsonpath='{range .items[*].metadata}{"Pod Name: "}{.name}{"\n"}{end}'
```

出力例

```
Pod Name: docker-registry-1-wvhrx
Pod Name: registry-console-1-ntq65
Pod Name: router-1-xzw69
```

4.5.2. describe

クエリーによって返される特定のオブジェクトに関する情報を返します。特定の **<object_name>** を指定する必要があります。オブジェクトタイプ で説明されるように、利用可能な実際の情報は状況によって異なります。

```
$ oc describe <object_type> <object_name>
```

4.5.3. edit

必要なオブジェクトタイプを編集します。

```
$ oc edit <object_type>/<object_name>
```

必要なオブジェクトタイプを指定のテキストエディターで編集します。

```
$ OC_EDITOR="<text_editor>" oc edit <object_type>/<object_name>
```

必要なオブジェクトを指定の形式 (例: JSON) で編集します。

```
$ oc edit <object_type>/<object_name> \
    --output-version=<object_type_version> \
    -o <object_type_format>
```

4.5.4. volume

ボリュームを変更します。

```
$ oc set volume <object_type>/<object_name> [--option]
```

4.5.5. label

オブジェクトのラベルを更新します。

```
$ oc label <object_type> <object_name> <label>
```

4.5.6. expose

サービスを検索し、これをルートとして公開します。デプロイメント設定、レプリケーションコントローラー、サービス、または Pod を指定されたポート上の新規サービスとして公開する機能もあります。ラベルが指定されていない場合、新規オブジェクトは公開するオブジェクトのラベルを再利用します。

サービスを公開する場合、デフォルトのジェネレーターは **--generator=route/v1** になります。デフォルトが **--generator=service/v2** になるその他すべてのケースでは、ポート名が指定されないままになります。通常は **oc expose** コマンドにジェネレーターを設定する必要はありません。3 つ目のジェネレーター **--generator=service/v1** はデフォルトのポート名で利用できます。

```
$ oc expose <object_type> <object_name>
```

4.5.7. delete

指定されたオブジェクトを削除します。オブジェクトの設定は STDIN で渡すこともできます。**oc delete all -l <label>** 操作は、指定された **<label>** に一致するすべてのオブジェクトを削除します。これにはレプリケーションコントローラーも含まれ、これが削除されると Pod は再作成されなくなります。

```
$ oc delete -f <file_path>
```

```
$ oc delete <object_type> <object_name>
```

```
$ oc delete <object_type> -l <label>
```

```
$ oc delete all -l <label>
```

4.5.8. set

指定したオブジェクトの特定のプロパティーを変更します。

4.5.8.1. set env

デプロイメント設定またはビルド設定の環境変数を設定します。

```
$ oc set env dc/mydc VAR1=value1
```

4.5.8.2. set build-secret

ビルド設定のシークレットの名前を設定します。シークレットは、イメージのプル/プッシュシークレットまたはソースリポジトリーシークレットになります。

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

4.6. ビルドおよびデプロイメントの各種操作

OpenShift Container Platform の基本的な機能の 1 つとして、アプリケーションをソースからコンテナーにビルドする機能があります。

OpenShift Container Platform では CLI のアクセスを提供し、**get**、**create**、および **describe** などの標準の **oc** リソース操作を使用してデプロイメント設定を検査したり、操作したりします。

4.6.1. start-build

指定されたビルド設定ファイルを使用して、手動でビルドプロセスを開始します。

```
$ oc start-build <buildconfig_name>
```

直前のビルドの名前を開始点として指定し、ビルドプロセスを手動で開始します。

```
$ oc start-build --from-build=<build_name>
```

設定ファイルを指定するか、または直前のビルドの名前を指定してビルドプロセスを手動で開始し、そのビルドログを取得します。

```
$ oc start-build --from-build=<build_name> --follow
```

```
$ oc start-build <buildconfig_name> --follow
```

ビルドが完了するまで待機し、ビルドが失敗する場合はゼロ以外のリターンコードを出して終了します。

```
$ oc start-build --from-build=<build_name> --wait
```

ビルド設定を変更することなく、現在のビルドの環境変数を設定するか、または上書きします。または、**-e** を使用します。

```
$ oc start-build --env <var_name>=<value>
```

ビルド時にデフォルトのビルドログレベルの出力を設定するか、または上書きします。

```
$ oc start-build --build-loglevel [0-5]
```

ビルドで使用する必要のあるソースコードのコミット ID を指定します。これには Git リポジトリーに基づくビルドが必要です。

```
$ oc start-build --commit=<hash>
```

<build_name> の名前でビルドを再実行します。

```
$ oc start-build --from-build=<build_name>
```

<dir_name> をアーカイブし、これを使用してバイナリー入力としてビルドします。

```
$ oc start-build --from-dir=<dir_name>
```

既存のアーカイブをバイナリー入力として使用します。**--from-file** とは異なり、ビルドプロセスの前にビルダーがアーカイブを展開します。

```
$ oc start-build --from-archive=<archive_name>
```

<file_name> をビルドのバイナリー入力として使用します。このファイルはビルドソース内の唯一のファイルでなければなりません。たとえば、**pom.xml** または **Dockerfile** などがこれに該当します。

```
$ oc start-build --from-file=<file_name>
```

ファイルシステムから読み取るのではなく、HTTP または HTTPS を使用してバイナリー入力をダウンロードします。

```
$ oc start-build --from-file=<file_URL>
```

アーカイブをダウンロードし、そのコンテンツをビルドソースとして使用します。

```
$ oc start-build --from-archive=<archive_URL>
```

ビルドのバイナリー入力として使用するローカルソースコードリポジトリーへのパスです。

```
$ oc start-build --from-repo=<path_to_repo>
```

トリガーする既存ビルド設定の Webhook URL を指定します。

```
$ oc start-build --from-webhook=<webhook_URL>
```

ビルドをトリガーする post-receive フックのコンテンツです。

```
$ oc start-build --git-post-receive=<contents>
```

post-receive の Git リポジトリーへのパスです。デフォルトは現在のディレクトリーに設定されます。

```
$ oc start-build --git-repository=<path_to_repo>
```

指定されたビルド設定またはビルドの Webhook を一覧表示します。**all**、**generic**、または **github** を受け入れます。

```
$ oc start-build --list-webhooks
```

source-strategy ビルドの **Spec.Strategy.SourceStrategy.Incremental** オプションを上書きします。

```
$ oc start-build --incremental
```

docker-strategy ビルドの **Spec.Strategy.DockerStrategy.NoCache** オプションを上書きします。

```
$oc start-build --no-cache
```

4.6.2. rollback

ロールバックを実行します。

```
$ oc rollback <deployment_name>
```

4.6.3. new-build

現在の Git リポジトリー (パブリックリモート) およびコンテナーイメージのソースコードに基づいてビルド設定を作成します。

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

リモート Git リポジトリーに基づくビルド設定を作成します。

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

プライベートリモート Git リポジトリーに基づいてビルド設定を作成します。

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

4.6.4. cancel-build

進行中のビルドを停止します。

```
$ oc cancel-build <build_name>
```

複数のビルドを同時にキャンセルします。

```
$ oc cancel-build <build1_name> <build2_name> <build3_name>
```

ビルド設定から作成されたビルドすべてをキャンセルします。

```
$ oc cancel-build bc/<buildconfig_name>
```

取り消すビルドを指定します。

```
$ oc cancel-build bc/<buildconfig_name> --state=<state>
```

state の値の例には、**new** または **pending** があります。

4.6.5. import-image

外部のイメージリポジトリーからタグおよびイメージ情報をインポートします。

```
$ oc import-image <image_stream>
```

4.6.6. scale

レプリケーションコントローラー またはデプロイメント設定の必要なレプリカ数を指定されるレプリカ数に設定します。

```
$ oc scale <object_type> <object_name> --replicas=<#_of_replicas>
```

4.6.7. tag

イメージストリームまたはコンテナーイメージの「プル仕様 (pull spec)」から既存のタグまたはイメージを取得し、1 つ以上の他のイメージストリームのタグに最新イメージとして設定します。

```
$ oc tag <current_image> <image_stream>
```

4.7. 高度なコマンド

4.7.1. create

設定ファイルを解析し、ファイルの内容に基づいて 1 つ以上の OpenShift Container Platform オブジェクトを作成します。**-f** フラグは、複数の異なるファイルまたはディレクトリーパスを使用して複数回渡すことが可能です。フラグが複数回渡される場合、**oc create** がフラグごとに繰り返され、指示されたファイルすべてに記述されるオブジェクトが作成されます。既存のリソースはいずれも無視されます。

```
$ oc create -f <file_or_dir_path>
```

4.7.2. replace

指定された設定ファイルの内容に基づいて、既存オブジェクトの変更を試行します。**-f** フラグは、異なるファイルまたはディレクトリーパスを指定して、複数回渡すことができます。フラグが複数回渡される場合、**oc replace** がフラグごとに繰り返され、指定のファイルすべてに記述されるオブジェクトが更新されます。

```
$ oc replace -f <file_or_dir_path>
```

4.7.3. process

プロジェクトの テンプレート をプロジェクト設定ファイルに変換します。

```
$ oc process -f <template_file_path>
```

4.7.4. run

特定のイメージを作成し、実行します。イメージがレプリケートされる場合もあります。デフォルトでは、デプロイメント設定を作成し、作成されたコンテナーを管理します。**--generator** フラグを使用して別のリソースの作成を選択することができます。

API リソース	--generator オプション
デプロイメント設定	**deploymentconfig/v1** (デフォルト)
Pod	**run-pod/v1**
レプリケーションコントローラー	**run/v1**
extensions/v1beta1 エンドポイントを使用したデプロイメント	**deployment/v1beta1**
apps/v1beta1 エンドポイントを使用したデプロイメント	**deployment/apps.v1beta1**
ジョブ	**job/v1**
Cron ジョブ	**cronjob/v2alpha1**

対話型コンテナーの場合には、フォアグラウンドでの実行を選択できます。

```
$ oc run NAME --image=<image> \
    [--generator=<resource>] \
    [--port=<port>] \
    [--replicas=<replicas>] \
    [--dry-run=<bool>] \
    [--overrides=<inline_json>] \
    [options]
```

4.7.5. patch

スト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。

```
$ oc patch <object_type> <object_name> -p <changes>
```

<changes> は、新しいフィールドおよび値を含む JSON または YAML 式です。たとえば、ノード **node1** の **spec.unschedulable** フィールドを **true** の値に更新する場合、JSON 式は以下のようになります。

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

4.7.6. policy

承認ポリシーを管理します。

```
$ oc policy [--options]
```

4.7.7. secrets

シークレットを設定します。

```
$ oc secrets [--options] path/to/ssh_key
```

4.7.8. autoscale

アプリケーションの Autoscaler を設定します。メトリクスをクラスターで有効にする必要があります。クラスター管理者向けの説明については「クラスターメトリクスの有効化」を随時参照してください。

```
$ oc autoscale dc/<dc_name> [--options]
```

4.8. トラブルシューティングおよびデバッグ操作

4.8.1. 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。

```
$ oc debug -h
```

イメージおよび設定の問題をデバッグする際、実行中の Pod 設定の正確なコピーを取得し、shell でトラブルシュートを実行することができます。障害が発生している Pod は起動できず、**rsh** または**exec** にアクセスできない可能性があるため、**debug** コマンドを実行して、対象の設定の正確なコピーを作成します。

デフォルトモードでは、参照される Pod、レプリケーションコントローラー、またはデプロイメント設定の最初のコンテナー内でシェルを起動します。起動される Pod はソース Pod のコピーになりますが、ラベルは取られ、コマンドは **/bin/sh** に変更され、readiness および liveness チェックは無効にされます。コマンドのみを実行する必要がある場合には、**--** と実行する 1 つのコマンドを追加します。コマ

ンドを渡しても、デフォルトで TTY が作成されたり、STDIN が送信されたりすることはありません。コンテナーまたは Pod を一般的な方法で変更する際に使用できる他のサポートされているフラグを使用することもできます。

コンテナーを実行する際の一般的な問題として、セキュリティーポリシーによってクラスター上で root ユーザーとしての実行が禁止されることがあります。このコマンドは、(**--as-user** を使用して) root ユーザー以外で Pod の実行をテストするか、または (**--as-root** を使用して) root ユーザーとして root 以外の Pod を実行するために使用できます。

デバッグ Pod はリモートコマンドの完了時またはシェルが中断する際に削除されます。

4.8.1.1. 使用法

```
$ oc debug RESOURCE/NAME [ENV1=VAL1 ...] [-c CONTAINER] [options] [-- COMMAND]
```

4.8.1.2. 例

現在実行中のデプロイメントをデバッグするには、以下を実行します。

```
$ oc debug dc/test
```

非 root ユーザーとしてデプロイメントの実行をテストするには、以下を実行します。

```
$ oc debug dc/test --as-user=1000000
```

second コンテナーで **env** コマンドを実行し、特定の失敗したコンテナーをデバッグするには、以下を実行します。

```
$ oc debug dc/test -c second -- /bin/env
```

デバッグするために作成される Pod を表示するには、以下を実行します。

```
$ oc debug dc/test -o yaml
```

4.8.2. logs

特定のビルド、デプロイメント、または Pod のログ出力を取得します。このコマンドは、ビルド、ビルド設定、デプロイメント設定、および Pod で機能します。

```
$ oc logs -f <pod>
```

4.8.3. exec

すでに実行中のコンテナーでコマンドを実行します。オプションでコンテナー ID を指定できますが、指定しない場合はデフォルトで最初のコンテナーが指定されます。

```
$ oc exec <pod> [-c <container>] <command>
```

4.8.4. 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。

```
$ oc rsh <pod>
```

4.8.5. rsync

すでに実行中の Pod コンテナーのディレクトリーへコンテンツをコピーするか、またはこのディレクトリーからコンテンツをコピーします。コンテナーを指定しない場合は、デフォルトで Pod 内の最初のコンテナーが指定されます。

ローカルディレクトリーから Pod 内のディレクトリーにコンテンツをコピーするには、以下を実行します。

```
$ oc rsync <local_dir> <pod>:<pod_dir> -c <container>
```

Pod 内のディレクトリーからローカルディレクトリーにコンテンツをコピーするには、以下を実行します。

```
$ oc rsync <pod>:<pod_dir> <local_dir> -c <container>
```

4.8.6. port-forward

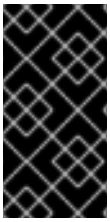
Pod に 1 つ以上のローカルポートを転送するには、以下を実行します。

```
$ oc port-forward <pod> <local_port>:<remote_port>
```

4.8.7. proxy

Kubernetes API サーバーのプロキシーを実行します。

```
$ oc proxy --port=<port> --www=<static_directory>
```



重要

セキュリティー上の理由により、**oc exec** コマンドは、特権付きコンテナーにアクセスする場合には機能しません。ただし、**cluster-admin** ユーザーがこのコマンドを実行する場合を除きます。管理者はノードホストに対して SSH を実行し、必要なコンテナーで **docker exec** コマンドを使用することができます。

4.9. トラブルシューティング OC

-v=X フラグを使用してログレベルを上げることにより、より詳細な出力を取得することができます。デフォルトで、ログレベルは **0** に設定されますが、この値 **0** を **10** に設定することができます。

各ログレベルの概要

- **1**-**5**: 通常、作成者がフローについてより詳細な説明を提供するようにした場合にコマンドによって内部で使用されます。
- **6**: HTTP 操作および URL など、クライアントとサーバー間の HTTP トラフィックについての基本的な情報を提供します。

- **7**: HTTP 操作、URL、要求ヘッダーおよび応答ステータスコードなどのより詳細な HTTP 情報を提供します。
- **8**: 本体を含む完全な HTTP 要求および応答を提供します。
- **9**: 本体およびサンプル **curl** 呼び出しを含む、完全な HTTP 要求および応答を提供します。
- **10**: コマンドが提供する可能性のあるすべての出力を提供します。

第5章 管理者 CLI の各種操作

5.1. 概要

このトピックでは、管理者 CLI の各種操作およびそれらの構文に関する情報を提供します。これらの操作を実行する前に、CLI を使用して 設定およびログインを行う必要があります。

openshift コマンドは、OpenShift Container Platform クラスターを構成するサービスを起動するために使用します。たとえば、**openshift start [master|node]** などを使用できます。ただし、これは **oc** コマンドや **oc adm** コマンドと同じアクションを **openshift cli** および **openshift admin** で 実行できるオールインワンのコマンドでもあります。

管理者 CLI は、開発者 CLI の通常のコマンドセットとは異なります。開発者 CLI は **oc** コマンドを使用し、プロジェクトレベルの操作でより多く使用されます。

5.2. 一般的な操作

管理者 CLI は、OpenShift Container Platform で管理される各種オブジェクトとの対話を許可します。数多くの一般的な **oc adm** 操作は以下の構文を使用して呼び出されます。

```
$ oc adm <action> <option>
```

これにより、以下が指定されます。

- **new-project** または **groups** など、実行する **<action>**。
- アクションを実行する際に利用可能な **<option>** およびオプションの値。オプションには **--output** が含まれます。

重要

oc adm コマンドの実行時は、Ansible ホストのインベントリーファイルに記載されている最初のマスターからのみ実行してください。デフォルトは、**/etc/ansible/hosts** です。

5.3. 基本的な CLI 操作

5.3.1. new-project

新しいプロジェクトを作成します。

```
$ oc adm new-project <project_name>
```

5.3.2. policy

承認ポリシーを管理します。

```
$ oc adm policy
```

5.3.3. groups

グループを管理します。

```
$ oc adm groups
```

5.4. CLI 操作のインストール

5.4.1. ルーター

ルーターをインストールします。

```
$ oc adm router <router_name>
```

5.4.2. ipfailover

ノードセットの IP フェイルオーバーグループをインストールします。

```
$ oc adm ipfailover <ipfailover_config>
```

5.4.3. レジストリー

統合コンテナーイメージレジストリーをインストールします。

```
$ oc adm registry
```

5.5. メンテナンス CLI の各種操作

5.5.1. build-chain

ビルドの入力と依存関係を出力します。

```
$ oc adm build-chain <image_stream>[:<tag>]
```

5.5.2. manage-node

ノードを管理します。たとえば、Pod の一覧表示または退避を実行します。または「Ready (準備完了)」のマークを付けます。

```
$ oc adm manage-node
```

5.5.3. prune

リソースの古いバージョンをサーバーから削除します。

```
$ oc adm prune
```

5.6. 設定 CLI の各種操作

5.6.1. config

kubelet 設定ファイルを変更します。

```
$ oc adm config <subcommand>
```

5.6.2. 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。

```
$ oc adm create-kubeconfig
```

5.6.3. create-api-client-config

ユーザーとしてサーバーに接続する設定ファイルを作成します。

```
$ oc adm create-api-client-config
```

5.7. 高度な CLI 操作

5.7.1. create-bootstrap-project-template

ブートストラッププロジェクトテンプレートを作成します。

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

5.7.2. 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。

```
$ oc adm create-bootstrap-policy-file
```

5.7.3. create-login-template

ログインテンプレートを作成します。

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

5.7.4. create-node-config

ノードの設定バンドルを作成します。

```
$ oc adm create-node-config
```

5.7.5. ca

証明書およびキーを管理します。

```
$ oc adm ca
```

第6章 OC と KUBECTL の相違点

6.1. KUBECTL ではなく OC を使用する場合

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 Kubernetes クラスターの上部で実行されるため、**kubectl** のコピーは OpenShift Container Platform のコマンドラインインターフェース (CLI) である **oc** にも含まれます。

これら 2 つのクライアントにはいくつかの類似点がありますが、本書では、これらの内の一方を使用する主な理由およびシナリオについて明確にすることを目的としています。

6.2. OC の使用

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 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 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 では利用できません。

認証

oc バイナリーは、認証を許可するビルドイン **login** コマンドを提供します。詳細は、開発者の認証および認証の設定について参照してください。

追加コマンド

追加コマンドの **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。

6.3. KUBECTL の使用

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 API に変更を加えずにバイナリーを使用できますが、先のセクションで説明されている追加機能を取得するには、**oc** へのアップグレードを検討する必要があります。

oc は **kubectl** の上部でビルドされているため、**kubectl** バイナリーを **oc** に変換することは、バイナリーの名前を **kubectl** から **oc** に変更するのと同じように簡単に実行できます。

インストールと設定に関する説明は、「CLI の使用方法」を参照してください。

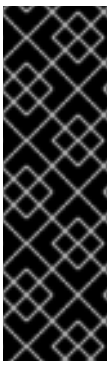
第7章 CLI の拡張

7.1. 概要

このトピックでは、CLI の拡張のインストールおよび作成方法について説明します。通常 **プラグイン** または **バイナリー拡張** と呼ばれるこの機能を使用すると、利用可能なデフォルトの **oc** コマンドセットを拡張でき、新規タスクを実行することができます。

プラグインはファイルのセットで構成されます。通常は少なくとも 1 つの **plugin.yaml** 記述子、1 つ以上のバイナリー、スクリプト、またはアセットファイルが含まれます。

現時点で CLI プラグインは **oc plugin** サブコマンドでのみ利用可能です。



重要

現時点で 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のリリースに先駆けてご提供することができ、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。

7.2. 前提条件

以下が必要になります。

- インストール済みの動作する **oc** バイナリー

7.3. プラグインのインストール

oc がプラグインを検索するファイルシステム内の場所に、プラグインの **plugin.yaml** 記述子、バイナリー、スクリプト、およびアセットファイルをコピーします。

現在、OpenShift Container Platform はプラグイン用のパッケージマネージャーを提供していません。したがって、お客様にプラグインファイルを適切な場所に配置していただく必要があります。各プラグインは独自のディレクトリーに置くことが推奨されます。

圧縮ファイルとして配布されているプラグインをインストールするには、「プラグインローダー」のセクションで指定される場所に展開します。

7.3.1. プラグインローダー

プラグインローダーは、「プラグインファイルの検索」のほか、プラグインを実行する上で必要最低限の情報をプラグインが提供するかどうかを確認します。正しい場所に置かれたファイルで、最低限の情報を提供しないもの (例: 不十分な **plugin.yaml** 記述子) は無視されます。

7.3.1.1. 検索の順序

プラグインローダーは、以下の順序で検索します。

1. **${KUBECTL_PLUGINS_PATH}**

これが指定されている場合は、検索はここで停止します。

KUBECTL_PLUGINS_PATH 環境変数がある場合、ローダーはこれをプラグインを検索する際の唯一の場所として使用します。**KUBECTL_PLUGINS_PATH** 環境変数はディレクトリーの一覧です。Linux および Mac では、この一覧はコロンで区切られています。Windows の場合、この一覧はセミコロンで区切られています。

KUBECTL_PLUGINS_PATH がない場合、ローダーは他の場所を探し始めます。

2. **${XDG_DATA_DIRS}/kubectl/plugins**
プラグインローダーは、XDG System Directory Structure の仕様に応じて指定された 1 つ以上のディレクトリーを検索します。

とりわけローダーは、**XDG_DATA_DIRS** 環境変数が指定したディレクトリーの場所を見つけます。プラグインローダーは、**XDG_DATA_DIRS** 環境変数が指定したディレクトリー内にある **kubectl/plugins** ディレクトリーを検索します。**XDG_DATA_DIRS** が指定されていない場合は、デフォルトで **/usr/local/share:/usr/share** を指定します。

3. **~/.kube/plugins**
ユーザーの **kubeconfig** ディレクトリーにある **plugins** ディレクトリーです。ほとんどのケースでは、これは **~/.kube/plugins** です。

```
# Loads plugins from both /path/to/dir1 and /path/to/dir2
$ KUBECTL_PLUGINS_PATH=/path/to/dir1:/path/to/dir2 kubectl plugin -h
```

7.4. プラグインの作成

プラグインは、CLI コマンドの作成に使用できるプログラミング言語またはスクリプトで作成できます。プラグインには必ずしもバイナリーコンポーネントが必要であるとは限りません。**echo**、**sed**、または **grep** などのオペレーティングシステムのユーティリティーに完全に依存させることができます。または、**oc** バイナリーに依存させることも可能です。

oc プラグインで唯一の必須要件として、**plugin.yaml** 記述子ファイルが必要となります。このファイルは、プラグインの登録に必要な最低限の属性を宣言する必要があり、「検索の順序」のセクションで指定された場所に置かれる必要があります。

7.4.1. plugin.yaml 記述子

記述子ファイルは、以下の属性をサポートします。

```
name: "great-plugin"            # REQUIRED: the plug-in command name, to be invoked under 'kubectl'
shortDesc: "great-plugin plug-in" # REQUIRED: the command short description, for help
longDesc: ""                    # the command long description, for help
example: ""                     # command example(s), for help
command: "./example"            # REQUIRED: the command, binary, or script to invoke when running the plug-in
flags:                          # flags supported by the plug-in
  - name: "flag-name"           # REQUIRED for each flag: flag name
    shorthand: "f"              # short version of the flag name
    desc: "example flag"        # REQUIRED for each flag: flag description
    defValue: "extreme"         # default value of the flag
tree:                           # allows the declaration of subcommands
  - ...                         # subcommands support the same set of attributes
```

前述の記述子は **great-plugin** プラグインを宣言します。これには、**-f | --flag-name** という名前の 1 つのフラグが含まれます。これは以下を実行して呼び出すことができます。

```
$ oc plugin great-plugin -f value
```

プラグインが呼び出されると、記述子ファイルと同じディレクトリーにある **example** バイナリーまたはスクリプトが呼び出され、多くの引数および環境変数が渡されます。「ランタイム属性へのアクセス」のセクションでは、**example** コマンドがフラグ値およびその他のランタイムコンテキストにアクセスする方法について説明しています。

7.4.2. 推奨されるディレクトリー構造

ファイルシステム内に、各プラグイン独自の (可能ならばプラグインコマンドと同じ名前の) サブディレクトリーを設定することが推奨されます。このディレクトリーには、**plugin.yaml** 記述子と任意のバイナリー、スクリプト、アセット、または必要とされるその他の依存関係が含まれている必要があります。

たとえば、**great-plugin** プラグインのディレクトリー構造は、以下のようになります。

```
~/.kube/plugins/
└── great-plugin
    ├── plugin.yaml
    └── example
```

7.4.3. ランタイム属性へのアクセス

多くの場合、プラグインをサポートするために作成するバイナリーまたはスクリプトファイルは、プラグインフレームワークで提供される一部のコンテキスト情報にアクセスできる必要があります。たとえば、記述子ファイルでフラグを宣言すると、プラグインは、ランタイムでユーザー提供のフラグ値へのアクセスが必要になります。

これはグローバルフラグの場合も同様です。プラグインフレームワークがこれを実行するので、プラグインの作成側で引数の解析を行う必要はありません。またこれにより、プラグインと通常の **oc** コマンド間の一貫性を最適なレベルに保つことができます。

プラグインは環境変数を使ってランタイムのコンテキスト属性にアクセスできます。たとえば、フラグで提供された値にアクセスするには、バイナリーまたはスクリプトの適切な関数呼び出しを使用して適切な環境変数の値を探します。

以下は、サポート対象の環境変数です。

- **KUBECTL_PLUGINS_CALLER**: 現在のコマンド呼び出しで使用された **oc** バイナリーへの完全パスです。プラグイン作成側として、Kubernetes API の認証およびこれにアクセスするためのロジックを実装する必要はありません。代わりに、この環境変数が提供する値を使用して **oc** を呼び出し、必要な情報を取得することができます。たとえば、**oc get --raw=/apis** などを使用できます。
- **KUBECTL_PLUGINS_CURRENT_NAMESPACE**: この呼び出しのコンテキストである現在の namespace です。これは namespace を使用する操作で考慮される実際の namespace です。つまり、これは kubeconfig、**--namespace** グローバルフラグ、環境変数などで提供され、優先順位が付けられた内容に従ってすでに処理されていることを意味します。
- **KUBECTL_PLUGINS_DESCRIPTOR_***: plugin.yaml 記述子で宣言されるすべての属性の環境変数です。たとえば、**KUBECTL_PLUGINS_DESCRIPTOR_NAME**、**KUBECTL_PLUGINS_DESCRIPTOR_CO**

MMAND などがあります。

- **KUBECTL_PLUGINS_GLOBAL_FLAG_***: **oc** がサポートするすべてのグローバルフラグの環境変数です。たとえば、**KUBECTL_PLUGINS_GLOBAL_FLAG_NAMESPACE**、**KUBECTL_PLUGINS_GLOBAL_FLAG_LOGLEVEL** などがあります。

- **KUBECTL_PLUGINS_LOCAL_FLAG_***: plugin.yaml 記述子で宣言されるすべてのローカルフラグの環境変数です。たとえば、前述の **great-plugin** の例で出てきた **KUBECTL_PLUGINS_LOCAL_FLAG_HEAT** などがあります。