OpenShift Online 3

CLI リファレンス

OpenShift Online CLI リファレンス

Last Updated: 2020-09-11

OpenShift Online 3 CLI リファレンス

OpenShift Online CLI リファレンス

法律上の通知

概要

OpenShift Online 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 Online 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 Online 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 Online 操作のスクリプトを作成する。
- 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 Online 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. CLI のインストール

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。これは、**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**」ページの **oc login** 行の末尾にある「**copy to clipboard**」ボタンをクリックしてから、コピーした内容を貼り付けてトークンを表示する必要があります。

2.2.1. Windows の場合

Windows 用の CLI は、**zip** アーカイブとして提供されます。これは Web コンソールの「**Command Line Tools**」ページからダウンロードできます。

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
```

2.2.2. Mac OS X の場合

Mac OS X 用の CLI は、**tar.gz** アーカイブとして提供されます。これは Web コンソールの「**Command Line Tools**」ページからダウンロードできます。

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、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
```

2.2.3. Linux の場合

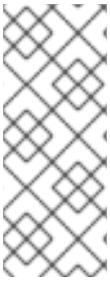
Linux 用の CLI は、**tar.gz** アーカイブとして提供されます。これは Web コンソールの「**Command Line Tools**」ページからダウンロードできます。

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る場合があります。

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

2.3. 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。この対話式フローでは、指定の認証情報を使用して OpenShift Online サーバーへのセッションを確立することができます。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
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 Online 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
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
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。

2.4. 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 Online 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
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 Online 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。

2.5. プロジェクト

```
$ 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>
```

以下は例になります。

```
$ oc project project02
Now using project 'project02'.

$ oc project project03
Now using project 'project03'.

$ oc project
Using project 'project03'.
```

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 Online 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 Online 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 Online 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 Online 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。

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
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.
```

```
$ oc project
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
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。

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]`
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、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。

サブコマンド	使用法
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。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

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
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、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>
```

変更が反映された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を行う必要があります。

4.2. 一般的な操作

開発者 CLI は、OpenShift Online 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
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
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.
```

警告

3.0.2.0 より前のバージョンの **oc** には、API バージョンをサーバーに対してネゴシエートする機能がありませんでした。そのため、v1 以降のバージョンの API のみをサポートするサーバーで 3.0.1.0 までの **oc** を使用している場合は、**oc** クライアントが正しい API エンドポイントを参照するように **--api-version** を渡してください。例: **oc get svc --api-version=v1**

4.3. オブジェクトタイプ

CLI は、以下のオブジェクトタイプをサポートします。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**
event	**ev**
imageStream	**is**
imageStreamTag	**istag**
imageStreamImage	**isimage**
job	
LimitRange	**limits**
node	
pod	**po**
ResourceQuota	**quota**
replicationController	**rc**
secrets	
service	**svc**

オブジェクトタイプ	省略バージョン
ServiceAccount	**sa**
persistentVolume	**pv**
persistentVolumeClaim	**pvc**

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. whoami

現行セッションに関する情報を返します。

```
$ oc whoami [--options]
```

4.4.2. types

OpenShift Online 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
```

4.4.3. login

OpenShift Online サーバーにログインします。

```
$ oc login
```

4.4.4. logout

現在のセッションを終了します。

```
$ oc logout
```

4.4.5. new-project

新規プロジェクトを作成します。

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

4.4.6. 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
```

4.4.7. status

現在のプロジェクトの概要を表示します。

```
$ oc status
```

4.4.8. 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>
```

4.4.9. explain

リソースとそのフィールドのドキュメントを参照してください。

```
$ oc explain <resource_name>
```

4.4.10. cluster

OpenShift Online クラスターを起動または停止します。

```
$ oc cluster [--options]
```

4.4.11. completion

指定されたシェルのシェル補完コードを出力します。

```
$ oc completion [--options]
```

4.4.12. help

任意のコマンドのヘルプを表示します。

```
$ oc <command> --help
```

4.4.13. plugin

コマンドラインプラグインを実行します。

```
$ oc plugin [--options]
```

4.4.14. version

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。

```
$ oc version [--options]
```

4.5. アプリケーション変更 CLI の各種操作

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}'

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. config

クライアントの設定ファイルを変更します。

```
$ oc config  --config=""
```

4.5.5. volume

ボリュームを変更します。

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

4.5.6. label

オブジェクトのラベルを更新します。

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

4.5.7. annotate

リソースでアノテーションを更新します。

```
$ oc annotate [--options]
```

4.5.8. 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>
```

4.5.9. 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>
```

4.5.10. 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。

4.5.10.1. 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
```

4.5.10.2. 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 CLI の各種操作

OpenShift Online 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 Online 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. rollout

Kubernetes デプロイメントまたは OpenShift デプロイメント設定を管理します。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。

```
$ oc rollout [--options]
```

4.6.3. rollback

ロールバックを実行します。

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

4.6.4. 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
```

4.6.5. 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。

4.6.6. image

イメージの管理に役立つコマンド。

```
$ oc image [--options]
```

4.6.7. import

アプリケーションを OpenShift Online にインポートするコマンド。

```
$ oc import [--options]
```

4.6.8. 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>
```

4.6.9. 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>
```

4.6.10. 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. adm

管理コマンドクラスターを管理するツール:

```
$ oc adm [--options]
```

4.7.2. 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 Online 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>
```

4.7.3. 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、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 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>
```

4.7.4. 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。

```
$ oc apply [--options]
```

4.7.5. 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>
```

4.7.6. run

特定のイメージを作成し、実行します。デフォルトで、作成されたコンテナーを管理するためにデプロイメント設定を作成します。**--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]
```

4.7.7. 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}}'
```

4.7.8. export

リソースをエクスポートし、どこでも使用できるようにします。

```
$ oc export <object_type> [--options]
```

OpenShift Online Starter から OpenShift Online Pro にアップグレードする場合、**oc export all** を使用してすべての既存オブジェクトをエクスポートします。OpenShift Online Pro は、オブジェクトごとのリソース移行をサポートしません。

テンプレート形式でプロジェクトから既存オブジェクトをエクスポートする方法についての詳細は、「既存のオブジェクトからのテンプレートの作成」を参照してください。

4.7.9. extract

シークレットまたは設定マップをディスクに展開します。

```
$ oc extract [--options]
```

4.7.10. idle

スケーラブルなリソースをアイドリングします。

```
$ oc idle [--options]
```

4.7.11. observe

リソースの変更を監視し、それらに対応します。

```
$ oc observe [--options]
```

4.7.12. auth

認証を検査します。

```
$ oc auth [--options]
```

4.7.13. policy

認証ポリシーを管理します。

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

4.7.14. convert

設定ファイルを異なる API バージョン間で変換します。

```
$ oc convert [--options]
```

4.7.15. secrets

シークレットを設定します。

```
$ oc secrets [--options]
```

4.7.16. serviceaccounts

プロジェクトのサービスアカウントを管理します。サービスアカウントにより、システムコンポーネントは API にアクセスできます。

```
$ oc serviceaccounts [--options]
```

4.7.17. 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。クラスター管理者に連絡し、メトリクスが環境で有効になっているかどうかを確認します。

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

4.8. CLI 操作のトラブルシューティングおよびデバッグ

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. 各種の例

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

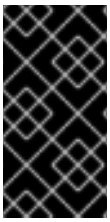
特定のビルド、デプロイメント、または Pod のログ出力を取得します。このコマンドは、ビルド、ビルド設定、デプロイメント設定、および Pod で機能します。

```
$ 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>
```



重要

セキュリティー保護の理由により、**oc exec** コマンドは、コマンドが **cluster-admin** ユーザーによって実行されている場合を除き、特権付きコンテナーにアクセスしようとしても機能しません。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。

4.8.4. 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

1 つ以上のローカルポートを Pod に転送します。

```
$ 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>
```

4.8.8. attach

実行中のコンテナーに割り当てます。

```
$ oc attach [--options]
```

4.8.9. 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。

```
$ oc cp [--options]
```