



# OpenShift Online 3

## CLI Reference

OpenShift Online CLI Reference



# OpenShift Online 3 CLI Reference

---

OpenShift Online CLI Reference

## Legal Notice

Copyright © 2018 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## Abstract

With the OpenShift Online command line interface (CLI), you can create applications and manage OpenShift projects from a terminal. These topics show you how to use CLI.

---

# Table of Contents

<b>CHAPTER 1. OVERVIEW</b> .....	<b>4</b>
<b>CHAPTER 2. GET STARTED WITH THE CLI</b> .....	<b>5</b>
2.1. OVERVIEW	5
2.2. INSTALLING THE CLI	5
2.2.1. For Windows	5
2.2.2. For Mac OS X	5
2.2.3. For Linux	5
2.3. BASIC SETUP AND LOGIN	6
2.4. CLI CONFIGURATION FILES	7
2.5. PROJECTS	7
<b>CHAPTER 3. MANAGING CLI PROFILES</b> .....	<b>9</b>
3.1. OVERVIEW	9
3.2. SWITCHING BETWEEN CLI PROFILES	9
3.3. MANUALLY CONFIGURING CLI PROFILES	11
3.4. LOADING AND MERGING RULES	13
<b>CHAPTER 4. DEVELOPER CLI OPERATIONS</b> .....	<b>15</b>
4.1. OVERVIEW	15
4.2. COMMON OPERATIONS	15
4.3. OBJECT TYPES	16
4.4. BASIC CLI OPERATIONS	17
4.4.1. whoami	17
4.4.2. types	17
4.4.3. login	17
4.4.4. logout	17
4.4.5. new-project	17
4.4.6. new-app	17
4.4.7. status	18
4.4.8. project	18
4.4.9. explain	18
4.4.10. cluster	18
4.4.11. completion	18
4.4.12. help	18
4.4.13. plugin	18
4.4.14. version	19
4.5. APPLICATION MODIFICATION CLI OPERATIONS	19
4.5.1. get	19
4.5.2. describe	19
4.5.3. edit	19
4.5.4. config	20
4.5.5. volume	20
4.5.6. label	20
4.5.7. annotate	20
4.5.8. expose	20
4.5.9. delete	20
4.5.10. set	21
4.5.10.1. set env	21
4.5.10.2. set build-secret	21
4.6. BUILD AND DEPLOYMENT CLI OPERATIONS	21
4.6.1. start-build	21

4.6.2. rollout	23
4.6.3. rollback	23
4.6.4. new-build	23
4.6.5. cancel-build	24
4.6.6. image	24
4.6.7. import	24
4.6.8. import-image	24
4.6.9. scale	24
4.6.10. tag	24
4.7. ADVANCED COMMANDS	25
4.7.1. adm	25
4.7.2. create	25
4.7.3. replace	25
4.7.4. apply	25
4.7.5. process	25
4.7.6. run	25
4.7.7. patch	26
4.7.8. export	26
4.7.9. extract	27
4.7.10. idle	27
4.7.11. observe	27
4.7.12. auth	27
4.7.13. policy	27
4.7.14. convert	27
4.7.15. secrets	27
4.7.16. serviceaccounts	27
4.7.17. autoscale	28
4.8. TROUBLESHOOTING AND DEBUGGING CLI OPERATIONS	28
4.8.1. debug	28
4.8.1.1. Usage	28
4.8.1.2. Examples	28
4.8.2. logs	29
4.8.3. exec	29
4.8.4. rsh	29
4.8.5. rsync	29
4.8.6. port-forward	29
4.8.7. proxy	29
4.8.8. attach	30
4.8.9. cp	30



## CHAPTER 1. OVERVIEW

With the OpenShift Online command line interface (CLI), you can [create applications](#) and manage OpenShift Online [projects](#) from a terminal. The CLI is ideal in situations where you are:

- Working directly with project source code.
- Scripting OpenShift Online operations.
- Restricted by bandwidth resources and cannot use the [web console](#).

The CLI is available using the **oc** command:

```
$ oc <command>
```

See [Get Started with the CLI](#) for installation and setup instructions.



## CHAPTER 2. GET STARTED WITH THE CLI

### 2.1. OVERVIEW

The OpenShift Online CLI exposes commands for managing your applications, as well as lower level tools to interact with each component of your system. This topic guides you through getting started with the CLI, including installation and logging in to create your first project.

### 2.2. INSTALLING THE CLI

Installation options for the CLI vary depending on your operating system.

To log in using the CLI, collect your token from the web console's **Command Line** page, which is accessed from **Command Line Tools** in the **Help** menu. The token is hidden, so you must click the **copy to clipboard** button at the end of the **oc login** line on the **Command Line Tools** page, then paste the copied contents to show the token.

#### 2.2.1. For Windows

The CLI for Windows is provided as a **zip** archive; you can download it from the **Command Line Tools** page on the web console.

Then, unzip the archive with a ZIP program and move the **oc** binary to a directory on your PATH. To check your PATH, open the Command Prompt and run:

```
C:\> path
```

#### 2.2.2. For Mac OS X

The CLI for Mac OS X is provided as a **tar.gz** archive; you can download it from the **Command Line Tools** page on the web console.

Then, unpack the archive and move the **oc** binary to a directory on your PATH. To check your PATH, open a Terminal window and run:

```
$ echo $PATH
```

#### 2.2.3. For Linux

The CLI for Linux is provided as a **tar.gz** archive; you can download it from the **Command Line Tools** page on the web console.

Then, unpack the archive and move the **oc** binary to a directory on your PATH. To check your path, run:

```
$ echo $PATH
```

To unpack the archive:

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

**NOTE**

If you do not use RHEL or Fedora, ensure that **libc** is installed in a directory on your library path. If **libc** is not available, you might see the following error when you run CLI commands:

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

**2.3. BASIC SETUP AND LOGIN**

The **oc login** command is the best way to initially set up the CLI, and it serves as the entry point for most users. The interactive flow helps you establish a session to an OpenShift Online server with the provided credentials. The information is automatically saved in a [CLI configuration file](#) that is then used for subsequent commands.

The following example shows the interactive setup and login using the **oc login** command:

**Example 2.1. Initial CLI Setup**

```
$ 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 The command prompts for the OpenShift Online server URL.
- 2 The command prompts for login credentials: a user name and password.
- 3 A session is established with the server, and a session token is received.
- 4 If you do not have a project, information is given on how to create one.

When you have completed the CLI configuration, subsequent commands use the configuration file for the server, session token, and project information.

You can log out of CLI using the **oc logout** command:

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

If you log in after creating or being granted access to a project, a project you have access to is automatically set as the current default, until [switching to another one](#):

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

Using project "aliceproject".
```

[Additional options](#) are also available for the **oc login** command.

## 2.4. CLI CONFIGURATION FILES

A CLI configuration file permanently stores **oc** options and contains a series of [authentication](#) mechanisms and OpenShift Online server connection information associated with nicknames.

As described in the previous section, the **oc login** command automatically creates and manages CLI configuration files. All information gathered by the command is stored in a configuration file located in `~/.kube/config`. The current CLI configuration can be viewed using the following command:

### Example 2.2. Viewing the CLI Configuration

```
$ 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 configuration files can be used to [setup multiple CLI profiles](#) using various OpenShift Online servers, namespaces, and users so that you can switch easily between them. The CLI can support multiple configuration files; they are loaded at runtime and merged together along with any override options specified from the command line.

## 2.5. PROJECTS

```
$ oc project
```

If you have access to multiple projects, use the following syntax to switch to a particular project by specifying the project name:

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

For example:

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

```
$ oc project project03  
Now using project 'project03'.
```

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

## CHAPTER 3. MANAGING CLI PROFILES

### 3.1. OVERVIEW

A CLI configuration file allows you to configure different profiles, or *contexts*, for use with the [OpenShift CLI](#). A context consists of [user authentication](#) and OpenShift Online server information associated with a *nickname*.

### 3.2. SWITCHING BETWEEN CLI PROFILES

Contexts allow you to easily switch between multiple users across multiple OpenShift Online servers, or *clusters*, when using issuing CLI operations. Nicknames make managing CLI configuration easier by providing short-hand references to contexts, user credentials, and cluster details.

After [logging in with the CLI](#) for the first time, OpenShift Online creates a `~/.kube/config` file if one does not already exist. As more authentication and connection details are provided to the CLI, either automatically during an `oc login` operation or by [setting them explicitly](#), the updated information is stored in the configuration file:

#### Example 3.1. CLI Configuration File

```
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-SKXRJ2Ds19ScenJdhNT1jEKTb8k
```

- ① The `clusters` section defines connection details for OpenShift Online clusters, including the address for their master server. In this example, one cluster is nicknamed `openshift1.example.com:8443` and another is nicknamed `openshift2.example.com:8443`.

- 2 This **contexts** section defines two contexts: one nicknamed **alice-project/openshift1.example.com:8443/alice**, using the **alice-project** project, **openshift1.example.com:8443** cluster, and **alice** user, and another nicknamed **joe-project/openshift1.example.com:8443/alice**, using the **joe-project** project, **openshift1.example.com:8443** cluster and **alice** user.
- 3 The **current - context** parameter shows that the **joe-project/openshift1.example.com:8443/alice** context is currently in use, allowing the **alice** user to work in the **joe-project** project on the **openshift1.example.com:8443** cluster.
- 4 The **users** section defines user credentials. In this example, the user nickname **alice/openshift1.example.com:8443** uses an [access token](#).

The CLI can support multiple configuration files; they are [loaded at runtime and merged together](#) along with any override options specified from the command line.

After you are logged in, you can use the **oc status** command or the **oc project** command to verify your current working environment:

### Example 3.2. Verifying the Current Working Environment

```
$ 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".
```

To log in using any other combination of user credentials and cluster details, run the **oc login** command again and supply the relevant information during the interactive process. A context is constructed based on the supplied information if one does not already exist.

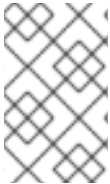
If you are already logged in and want to switch to another project the current user already has access to, use the **oc project** command and supply the name of the project:

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

At any time, you can use the **oc config view** command to view your current, full CLI configuration, as seen in the output.

Additional CLI configuration commands are also available for more [advanced usage](#).

### 3.3. MANUALLY CONFIGURING CLI PROFILES



#### NOTE

This section covers more advanced usage of CLI configurations. In most situations, you can simply use the **oc login** and **oc project** commands to log in and switch between contexts and projects.

If you want to manually configure your CLI configuration files, you can use the **oc config** command instead of modifying the files themselves. The **oc config** command includes a number of helpful subcommands for this purpose:

**Table 3.1. CLI Configuration Subcommands**

Subcommand	Usage
<b>set-credentials</b>	<p>Sets a user entry in the CLI configuration file. If the referenced user nickname already exists, the specified information is merged in.</p> <pre>\$ oc config set-credentials &lt;user_nickname&gt; [--client-certificate=&lt;path/to/certfile&gt;] [--client-key= &lt;path/to/keyfile&gt;] [--token=&lt;bearer_token&gt;] [--username=&lt;basic_user&gt;] [--password= &lt;basic_password&gt;]</pre>
<b>set-cluster</b>	<p>Sets a cluster entry in the CLI configuration file. If the referenced cluster nickname already exists, the specified information is merged in.</p> <pre>\$ oc config set-cluster &lt;cluster_nickname&gt; [--server= &lt;master_ip_or_fqdn&gt;] [--certificate-authority=&lt;path/to/certificate/authority&gt;] [--api-version=&lt;apiversion&gt;] [--insecure-skip-tls-verify=true]</pre>
<b>set-context</b>	<p>Sets a context entry in the CLI configuration file. If the referenced context nickname already exists, the specified information is merged in.</p> <pre>\$ oc config set-context &lt;context_nickname&gt; [--cluster= &lt;cluster_nickname&gt;] [--user=&lt;user_nickname&gt;] [--namespace=&lt;namespace&gt;]</pre>

Subcommand	Usage
<b>use-context</b>	<p>Sets the current context using the specified context nickname.</p> <pre>\$ oc config use-context &lt;context_nickname&gt;</pre>
<b>set</b>	<p>Sets an individual value in the CLI configuration file.</p> <pre>\$ oc config set &lt;property_name&gt; &lt;property_value&gt;</pre> <p>The <b>&lt;property_name&gt;</b> is a dot-delimited name where each token represents either an attribute name or a map key. The <b>&lt;property_value&gt;</b> is the new value being set.</p>
<b>unset</b>	<p>Unsets individual values in the CLI configuration file.</p> <pre>\$ oc config unset &lt;property_name&gt;</pre> <p>The <b>&lt;property_name&gt;</b> is a dot-delimited name where each token represents either an attribute name or a map key.</p>
<b>view</b>	<p>Displays the merged CLI configuration currently in use.</p> <pre>\$ oc config view</pre> <p>Displays the result of the specified CLI configuration file.</p> <pre>\$ oc config view --config=&lt;specific_filename&gt;</pre>

### Example Usage

Consider the following configuration workflow. First, set credentials for a user nickname **alice** that uses an [access token](#):

```
$ oc config set-credentials alice --
token=NDM2N2MwODgtNjI1Yy10N3VhLTg1YmItYzI4NDEzZDUyYzVi
```

Set a cluster entry named **openshift1**:

```
$ oc config set-cluster openshift1 --server=https://openshift1.example.com
```

Set a context named **alice** that uses the **alice** user and the **openshift1** cluster:

```
$ oc config set-context alice --cluster=openshift1 --user=alice
```

Now that the **alice** context has been created, switch to that context:

```
$ oc config use-context alice
```



Set the **aliceproject** namespace for the **alice** context:

```
$ oc config set contexts.alice.namespace aliceproject
```

You can now view the configuration that has been created:

```
$ oc config view
apiVersion: v1
clusters:
- cluster:
  server: https://openshift1.example.com
  name: openshift1
contexts:
- context:
  cluster: openshift1
  namespace: aliceproject
  user: alice
  name: alice
current-context: alice 1
kind: Config
preferences: {}
users:
- name: alice
  user:
    token: NDM2N2MwODgtNjI1Yy10N3VhLTg1YmItYzI4NDEzZDUyYzVi
```

**1** The current context is set to **alice**.

All subsequent CLI operations will use the **alice** context, unless otherwise specified by overriding CLI options or until the context is switched.

### 3.4. LOADING AND MERGING RULES

When issuing CLI operations, the loading and merging order for the CLI configuration follows these rules:

1. CLI configuration files are retrieved from your workstation, using the following hierarchy and merge rules:
  - If the **--config** option is set, then only that file is loaded. The flag may only be set once and no merging takes place.
  - If **\$KUBECONFIG** environment variable is set, then it is used. The variable can be a list of paths, and if so the paths are merged together. When a value is modified, it is modified in the file that defines the stanza. When a value is created, it is created in the first file that exists. If no files in the chain exist, then it creates the last file in the list.
  - Otherwise, the **~/.kube/config** file is used and no merging takes place.
2. The context to use is determined based on the first hit in the following chain:
  - The value of the **--context** option.
  - The **current-context** value from the CLI configuration file.

- An empty value is allowed at this stage.
3. The user and cluster to use is determined. At this point, you may or may not have a context; they are built based on the first hit in the following chain, which is run once for the user and once for the cluster:
    - The value of the **--user** option for user name and the **--cluster** option for cluster name.
    - If the **--context** option is present, then use the context's value.
    - An empty value is allowed at this stage.
  4. The actual cluster information to use is determined. At this point, you may or may not have cluster information. Each piece of the cluster information is built based on the first hit in the following chain:
    - The values of any of the following command line options:
      - **--server**,
      - **--api-version**
      - **--certificate-authority**
      - **--insecure-skip-tls-verify**
    - If cluster information and a value for the attribute is present, then use it.
    - If you do not have a server location, then there is an error.
  5. The actual user information to use is determined. Users are built using the same rules as clusters, except that you can only have one authentication technique per user; conflicting techniques cause the operation to fail. Command line options take precedence over configuration file values. Valid command line options are:
    - **--auth-path**
    - **--client-certificate**
    - **--client-key**
    - **--token**
  6. For any information that is still missing, default values are used and prompts are given for additional information.

## CHAPTER 4. DEVELOPER CLI OPERATIONS

### 4.1. OVERVIEW

This topic provides information on the developer CLI operations and their syntax. You must [setup and login](#) with the CLI before you can perform these operations.

### 4.2. COMMON OPERATIONS

The developer CLI allows interaction with the various objects that are managed by OpenShift Online. Many common **oc** operations are invoked using the following syntax:

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

This specifies:

- An **<action>** to perform, such as **get** or **describe**.
- The **<object\_type>** to perform the action on, such as **service** or the abbreviated **svc**.
- The **<object\_name>** of the specified **<object\_type>**.

For example, the **oc get** operation returns a complete list of services that are currently defined:

```
$ 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
```

The **oc describe** operation can then be used to return detailed information about a specific object:

```
$ 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.
```

**WARNING**

Versions of **oc** prior to 3.0.2.0 did not have the ability to negotiate API versions against a server. So if you are using **oc** up to 3.0.1.0 with a server that only supports v1 or higher versions of the API, make sure to pass **--api-version** in order to point the **oc** client to the correct API endpoint. For example: **oc get svc --api-version=v1**.

**4.3. OBJECT TYPES**

The CLI supports the following object types, some of which have abbreviated syntax:

Object Type	Abbreviated Version
<b>build</b>	
<b>buildConfig</b>	<b>bc</b>
<b>deploymentConfig</b>	<b>dc</b>
<b>event</b>	<b>ev</b>
<b>imageStream</b>	<b>is</b>
<b>imageStreamTag</b>	<b>istag</b>
<b>imageStreamImage</b>	<b>isimage</b>
<b>job</b>	
<b>LimitRange</b>	<b>limits</b>
<b>node</b>	
<b>pod</b>	<b>po</b>
<b>ResourceQuota</b>	<b>quota</b>
<b>replicationController</b>	<b>rc</b>
<b>secrets</b>	
<b>service</b>	<b>svc</b>

Object Type	Abbreviated Version
<b>ServiceAccount</b>	<b>sa</b>
<b>persistentVolume</b>	<b>pv</b>
<b>persistentVolumeClaim</b>	<b>pvc</b>

## 4.4. BASIC CLI OPERATIONS

The following table describes basic **oc** operations and their general syntax:

### 4.4.1. whoami

Return information about the current session:

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

### 4.4.2. types

Display an introduction to some core OpenShift Online concepts:

```
$ oc types
```

### 4.4.3. login

Log in to the OpenShift Online server:

```
$ oc login
```

### 4.4.4. logout

End the current session:

```
$ oc logout
```

### 4.4.5. new-project

Create a new project:

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

### 4.4.6. new-app

[Creates a new application](#) based on the source code in the current directory:

```
$ oc new-app
```

Creates a new application based on the source code in a remote repository:

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

Creates a new application based on the source code in a private remote repository:

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

#### 4.4.7. status

Show an overview of the current project:

```
$ oc status
```

#### 4.4.8. project

Switch to another project. Run without options to display the current project. To view all projects you have access to run **oc projects**.

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

#### 4.4.9. explain

See the documentation of a resource and its fields:

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

#### 4.4.10. cluster

Start or stop a OpenShift Online cluster:

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

#### 4.4.11. completion

Output shell completion code for the specified shell:

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

#### 4.4.12. help

Get help about any command:

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

#### 4.4.13. plugin

Run a command line plug-in:

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

#### 4.4.14. version

Display client and server versions:

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

## 4.5. APPLICATION MODIFICATION CLI OPERATIONS

### 4.5.1. get

Return a list of objects for the specified [object type](#). If the optional `<object_name>` is included in the request, then the list of results is filtered by that value.

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

You can use the `-o` or `--output` option to modify the output format.

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

The output format can be a JSON or YAML, or an extensible format like [custom columns](#), [golang template](#), and [jsonpath](#).

For example, the following command lists the name of the pods running in a specific project:

```
$ 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

Returns information about the specific object returned by the query. A specific `<object_name>` must be provided. The actual information that is available varies as described in [object type](#).

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

### 4.5.3. edit

Edit the desired object type:

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

Edit the desired object type with a specified text editor:

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

Edit the desired object in a specified format (eg: JSON):

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

#### 4.5.4. config

Change configuration files for the client:

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

#### 4.5.5. volume

Modify a [volume](#):

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

#### 4.5.6. label

Update the labels on a object:

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

#### 4.5.7. annotate

Update the annotations on a resource:

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

#### 4.5.8. expose

Look up a service and expose it as a route. There is also the ability to expose a deployment configuration, replication controller, service, or pod as a new service on a specified port. If no labels are specified, the new object will re-use the labels from the object it exposes.

If you are exposing a service, the default generator is **--generator=route/v1**. For all other cases the default is **--generator=service/v2**, which leaves the port unnamed. Generally, there is no need to set a generator with the **oc expose** command. A third generator, **--generator=service/v1**, is available with the port name default.

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

#### 4.5.9. delete



Delete the specified object. An object configuration can also be passed in through STDIN. The **oc delete all -l <label>** operation deletes all objects matching the specified **<label>**, including the [replication controller](#) so that pods are not re-created.

```
$ 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

Modify a specific property of the specified object.

### 4.5.10.1. set env

Sets an environment variable on a deployment configuration or a build configuration:

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

### 4.5.10.2. set build-secret

Sets the name of a secret on a build configuration. The secret may be an image pull or push secret or a source repository secret:

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

## 4.6. BUILD AND DEPLOYMENT CLI OPERATIONS

One of the fundamental capabilities of OpenShift Online is the ability to build applications into a container from source.

OpenShift Online provides CLI access to inspect and manipulate deployment configurations using standard **oc** resource operations, such as **get**, **create**, and **describe**.

### 4.6.1. start-build

Manually start the build process with the specified build configuration file:

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

Manually start the build process by specifying the name of a previous build as a starting point:

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

Manually start the build process by specifying either a configuration file or the name of a previous build and retrieve its build logs:

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

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

Wait for a build to complete and exit with a non-zero return code if the build fails:

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

Set or override environment variables for the current build without changing the build configuration. Alternatively, use **-e**.

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

Set or override the default build log level output during the build:

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

Specify the source code commit identifier the build should use; requires a build based on a Git repository:

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

Re-run build with name **<build\_name>**:

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

Archive **<dir\_name>** and build with it as the binary input:

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

Use existing archive as the binary input; unlike **--from-file** the archive will be extracted by the builder prior to the build process:

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

Use **<file\_name>** as the binary input for the build. This file must be the only one in the build source. For example, ***pom.xml*** or ***Dockerfile***.

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

Download the binary input using HTTP or HTTPS instead of reading it from the file system:

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

Download an archive and use its contents as the build source:

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

The path to a local source code repository to use as the binary input for a build:

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

Specify a webhook URL for an existing build configuration to trigger:

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

The contents of the post-receive hook to trigger a build:

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

The path to the Git repository for post-receive; defaults to the current directory:

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

List the webhooks for the specified build configuration or build; accepts **all**, **generic**, or **github**:

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

Override the **Spec.Strategy.SourceStrategy.Incremental** option of a source-strategy build:

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

Override the **Spec.Strategy.DockerStrategy.NoCache** option of a docker-strategy build:

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

## 4.6.2. rollout

Manage a Kubernetes deployment or an OpenShift deployment configuration. Start a new rollout, view its status or history, or rollback to a previous revision of your application:

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

## 4.6.3. rollback

Perform a [rollback](#):

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

## 4.6.4. new-build

Create a build configuration based on the source code in the current Git repository (with a public remote) and a container image:

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

Create a build configuration based on a remote git repository:

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

Create a build configuration based on a private remote git repository:

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

#### 4.6.5. cancel-build

Stop a build that is in progress:

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

Cancel multiple builds at the same time:

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

Cancel all builds created from the build configuration:

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

Specify the builds to be canceled:

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

Example values for **state** are **new** or **pending**.

#### 4.6.6. image

Useful commands for managing images.

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

#### 4.6.7. import

Commands that import applications into OpenShift Online.

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

#### 4.6.8. import-image

Import tag and image information from an external image repository:

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

#### 4.6.9. scale

Set the number of desired replicas for a [replication controller](#) or a deployment configuration to the number of specified replicas:

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

#### 4.6.10. tag

Take an existing tag or image from an image stream, or a container image "pull spec", and set it as the most recent image for a tag in one or more other image streams:

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

## 4.7. ADVANCED COMMANDS

### 4.7.1. adm

Administrative commands. Tools for managing a cluster:

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

### 4.7.2. create

Parse a configuration file and create one or more OpenShift Online objects based on the file contents. The **-f** flag can be passed multiple times with different file or directory paths. When the flag is passed multiple times, **oc create** iterates through each one, creating the objects described in all of the indicated files. Any existing resources are ignored.

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

### 4.7.3. replace

Attempt to modify an existing object based on the contents of the specified configuration file. The **-f** flag can be passed multiple times with different file or directory paths. When the flag is passed multiple times, **oc replace** iterates through each one, updating the objects described in all of the indicated files.

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

### 4.7.4. apply

Apply a configuration to a resource by file name or stdin:

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

### 4.7.5. process

Transform a project [template](#) into a project configuration file:

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

### 4.7.6. run

Create and run a particular image, possibly replicated. By default, create a deployment configuration to manage the created container(s). You can choose to create a different resource using the **--generator** flag:

API Resource	--generator Option
Deployment configuration	<b>deploymentconfig/v1</b> (default)
Pod	<b>run-pod/v1</b>
Replication controller	<b>run/v1</b>
Deployment using <b>extensions/v1beta1</b> endpoint	<b>deployment/v1beta1</b>
Deployment using <b>apps/v1beta1</b> endpoint	<b>deployment/apps.v1beta1</b>
Job	<b>job/v1</b>
Cron job	<b>cronjob/v2alpha1</b>

You can choose to run in the foreground for an interactive container execution.

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

#### 4.7.7. patch

Updates one or more fields of an object using strategic merge patch:

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

The <changes> is a JSON or YAML expression containing the new fields and the values. For example, to update the **spec. unschedulable** field of the node **node1** to the value **true**, the json expression is:

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

#### 4.7.8. export

Export resources to be used elsewhere:

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

If you are upgrading from OpenShift Online Starter to OpenShift Online Pro, use **oc export all** to export all of your existing objects. OpenShift Online Pro does not support per-object resource migration.

See [Creating a Template from Existing Objects](#) for more information on exporting existing objects from your project in template form.

### 4.7.9. extract

Extract secrets or config maps to disk:

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

### 4.7.10. idle

Idle scalable resources:

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

### 4.7.11. observe

Observe changes to resources and react to them:

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

### 4.7.12. auth

Inspect authorization:

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

### 4.7.13. policy

Manage authorization policies:

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

### 4.7.14. convert

Convert configuration files between different API versions:

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

### 4.7.15. secrets

Configure [secrets](#):

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

### 4.7.16. serviceaccounts

Manage service accounts in your project. Service accounts allow system components to access the API.

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

### 4.7.17. autoscale

Setup an [autoscaler](#) for your application. Requires metrics to be enabled in the cluster. Check with your cluster administrator to confirm whether metrics are enabled in your environment.

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

## 4.8. TROUBLESHOOTING AND DEBUGGING CLI OPERATIONS

### 4.8.1. debug

Launch a command shell to debug a running application.

```
$ oc debug -h
```

When debugging images and setup problems, you can get an exact copy of a running pod configuration and troubleshoot with a shell. Since a failing pod may not be started and not accessible to **rsh** or **exec**, running the **debug** command creates a carbon copy of that setup.

The default mode is to start a shell inside of the first container of the referenced pod, replication controller, or deployment configuration. The started pod will be a copy of your source pod, with labels stripped, the command changed to **/bin/sh**, and readiness and liveness checks disabled. If you just want to run a command, add **--** and a command to run. Passing a command will not create a TTY or send STDIN by default. Other flags are supported for altering the container or pod in common ways.

A common problem running containers is a security policy that prohibits you from running as a root user on the cluster. You can use this command to test running a pod as non-root (with **--as-user**) or to run a non-root pod as root (with **--as-root**).

The debug pod is deleted when the remote command completes or you interrupt the shell.

#### 4.8.1.1. Usage

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

#### 4.8.1.2. Examples

To debug a currently running deployment:

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

To test running a deployment as a non-root user:

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

To debug a specific failing container by running the **env** command in the **second** container:

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

To view the pod that would be created to debug:



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

### 4.8.2. logs

Retrieve the log output for a specific build, deployment, or pod. This command works for builds, build configurations, deployment configurations, and pods.

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

### 4.8.3. exec

Execute a command in an already-running container. You can optionally specify a container ID, otherwise it defaults to the first container.

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



#### IMPORTANT

For security purposes, the `oc exec` command does not work when accessing privileged containers except when the command is executed by a `cluster-admin` user. Administrators can SSH into a node host, then use the `docker exec` command on the desired container.

### 4.8.4. rsh

Open a remote shell session to a container:

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

### 4.8.5. rsync

Copy the contents to or from a directory in an already-running pod container. If you do not specify a container, it defaults to the first container in the pod.

To copy contents from a local directory to a directory in a pod:

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

To copy contents from a directory in a pod to a local directory:

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

### 4.8.6. port-forward

Forward one or more local ports to a pod:

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

### 4.8.7. proxy

Run a proxy to the Kubernetes API server:

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

#### **4.8.8. attach**

Attach to a running container:

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

#### **4.8.9. cp**

Copy files and directories to and from containers:

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