OpenShift Dedicated 3

CLI Reference

OpenShift Dedicated 3 CLI Reference

Last Updated: 2019-05-07

OpenShift Dedicated 3 CLI Reference

OpenShift Dedicated 3 CLI Reference

Legal Notice

Abstract

With the OpenShift Dedicated 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

CHAPTER 1. OVERVIEW

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

- Working directly with project source code.
- Scripting OpenShift Dedicated 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 Dedicated 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. PREREQUISITES

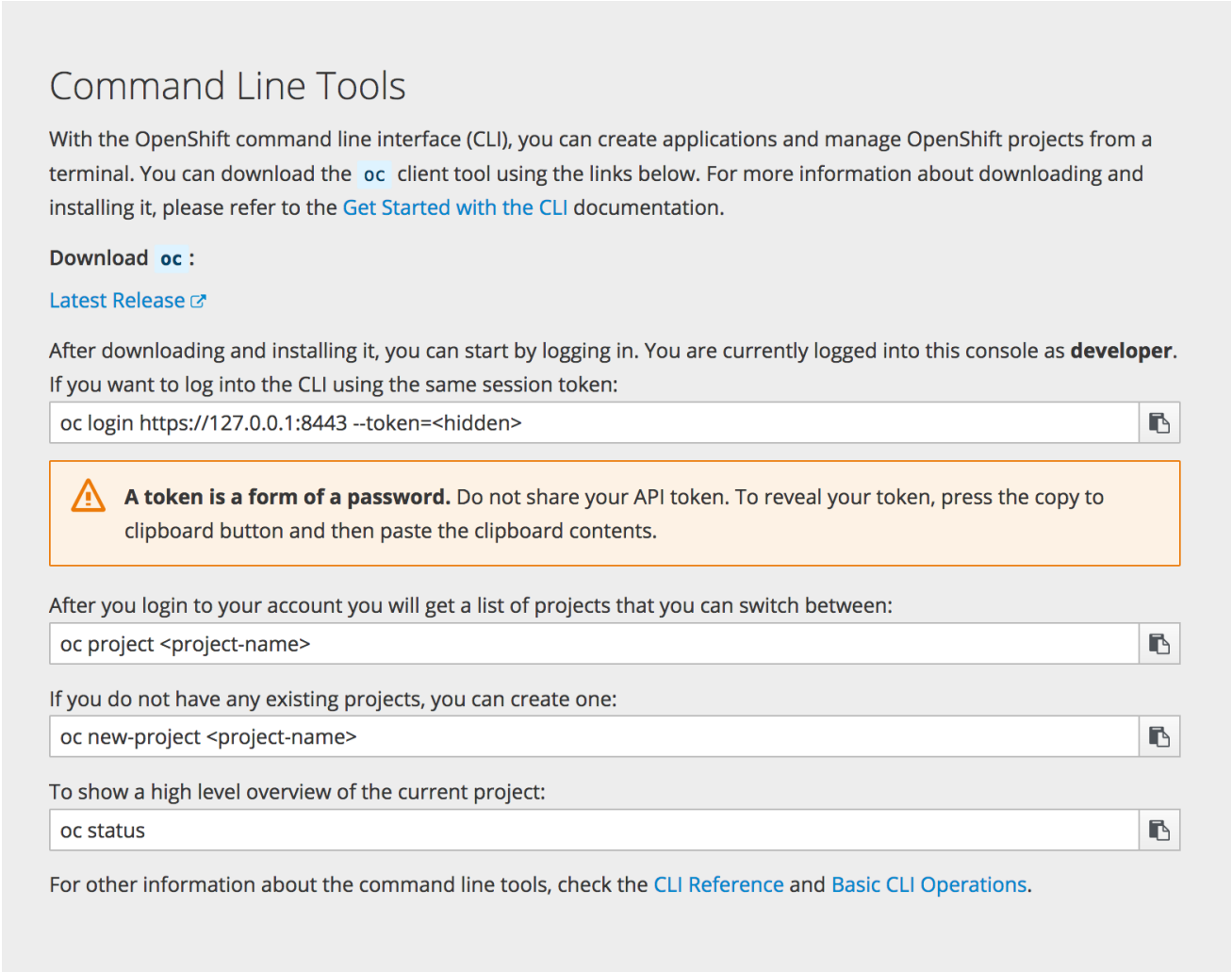
Certain operations require Git to be locally installed on a client. For example, the command to create an application using a remote Git repository:

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

Before proceeding, install Git on your workstation. See the official Git documentation for instructions per your workstation's operating system.

2.3. INSTALLING THE CLI

The easiest way to download the CLI is by accessing the **Command Line Tools** page on the web console:



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.

2.3.1. 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
```

2.3.2. 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
```

2.3.3. 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
```

2.4. 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 Dedicated 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 Dedicated 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.

NOTE

If you have access to administrator credentials but are no longer logged in as the default system user **system:admin**, you can log back in as this user at any time as long as the credentials are still present in your CLI configuration file. The following command logs in and switches to the **default** project:

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

2.5. CLI CONFIGURATION FILES

A CLI configuration file permanently stores **oc** options and contains a series of authentication mechanisms and OpenShift Dedicated 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 Dedicated 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.

2.6. PROJECTS

A project in OpenShift Dedicated contains multiple objects to make up a logical application.

Most **oc** commands run in the context of a project. The **oc login** selects a default project during initial setup to be used with subsequent commands. Use the following command to display the project currently in use:

```
$ 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'.
```

The **oc status** command shows a high level overview of the project currently in use, with its components and their relationships, as shown in the following example:

```
$ oc status
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. WHAT'S NEXT?

After you have logged in, you can create a new application and explore some common CLI operations.

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 Dedicated server information associated with a *nickname*.

3.2. SWITCHING BETWEEN CLI PROFILES

Contexts allow you to easily switch between multiple users across multiple OpenShift Dedicated 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 Dedicated 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-SKXRJ2Dsl9SceNJdhNTljEKTb8k
```

(1) The **clusters** section defines connection details for OpenShift Dedicated 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,

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.

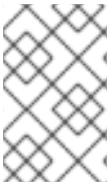
Additional CLI configuration commands are also available for more advanced usage.

NOTE

If you have access to administrator credentials but are no longer logged in as the default system user **system:admin**, you can log back in as this user at any time as long as the credentials are still present in your CLI configuration file. The following command logs in and switches to the **default** project:

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

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
set-credentials	Sets a user entry in the CLI configuration file. If the referenced user nickname already exists, the specified information is merged in. `$ oc config set-credentials <user_nickname> [--client-certificate=<path/to/certfile>] [--client-key=<path/to/keyfile>] [--token=<bearer_token>] [--username=<basic_user>] [--password=<basic_password>]`
set-cluster	Sets a cluster entry in the CLI configuration file. If the referenced cluster nickname already exists, the specified information is merged in. `$ 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	Sets a context entry in the CLI configuration file. If the referenced context nickname already exists, the specified information is merged in. `$ oc config set-context <context_nickname> [--cluster=<cluster_nickname>] [--user=<user_nickname>] [--namespace=<namespace>]`

Subcommand	Usage
use-context	Sets the current context using the specified context nickname. `$ oc config use-context <context_nickname>`
set	Sets an individual value in the CLI configuration file. `$ oc config set <property_name> <property_value>` The **<property_name>** is a dot-delimited name where each token represents either an attribute name or a map key. The **<property_value>** is the new value being set.
unset	Unsets individual values in the CLI configuration file. `$ oc config unset <property_name>` The **<property_name>** is a dot-delimited name where each token represents either an attribute name or a map key.

view	Displays the merged CLI configuration currently in use.
	`$ oc config view`
	Displays the result of the specified CLI configuration file.
	`$ oc config view --config=<specific_filename>`

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.

The developer CLI uses the **oc** command, and is used for project-level operations. This differs from the administrator CLI, which uses the **oc adm** command for more advanced, administrator operations.

4.2. COMMON OPERATIONS

The developer CLI allows interaction with the various objects that are managed by OpenShift Dedicated. 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.
```

4.3. OBJECT TYPES

Below is the list of the most common object types the CLI supports, some of which have abbreviated syntax:

Object Type	Abbreviated Version
Build	
BuildConfig	**bc**
DeploymentConfig	**dc**
Deployments	**deploy**
Event	**ev**
ImageStream	**is**
ImageStreamTag	**istag**
ImageStreamImage	**isimage**
Job	
CronJob (Technology Preview	**cj**
LimitRange	**limits**
Node	
Pod	**po**
ResourceQuota	**quota**
ReplicationController	**rc**
ReplicaSet	**rs**
Secrets	
Service	**svc**
ServiceAccount	**sa**
StatefulSets	**sts**
PersistentVolume	**pv**
PersistentVolumeClaim	**pvc**

If you want to know the full list of resources the server supports, use **oc api-resources**.

4.4. BASIC CLI OPERATIONS

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

4.4.1. types

Display an introduction to some core OpenShift Dedicated concepts:

```
$ oc types
```

4.4.2. login

Log in to the OpenShift Dedicated server:

```
$ oc login
```

4.4.3. logout

End the current session:

```
$ oc logout
```

4.4.4. new-project

Create a new project:

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

4.4.5. 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
```

4.4.6. status

Show an overview of the current project:

```
$ oc status
```

4.4.7. 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>
```

4.5. APPLICATION MODIFICATION 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. volume

Modify a volume:

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

4.5.5. label

Update the labels on a object:

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

4.5.6. 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>
```

4.5.7. 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>
```

4.5.8. set

Modify a specific property of the specified object.

4.5.8.1. set env

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

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

4.5.8.2. 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 OPERATIONS

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

OpenShift Dedicated 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. rollback

Perform a rollback:

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

4.6.3. 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
```

4.6.4. 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**.

4.6.5. import-image

Import tag and image information from an external image repository:

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

4.6.6. 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>
```

4.6.7. 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. create

Parse a configuration file and create one or more OpenShift Dedicated 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>
```

4.7.2. 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>
```

4.7.3. process

Transform a project template into a project configuration file:

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

4.7.4. 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	**deploymentconfig/v1** (default)
Pod	**run-pod/v1**
Replication controller	**run/v1**
Deployment using **extensions/v1beta1** endpoint	**deployment/v1beta1**
Deployment using **apps/v1beta1** endpoint	**deployment/apps.v1beta1**
Job	**job/v1**
Cron job	**cronjob/v2alpha1**

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]
```

4.7.5. 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}}'
```

4.7.6. policy

Manage authorization policies:

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

4.7.7. secrets

Configure secrets:

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

4.7.8. 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 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>
```

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

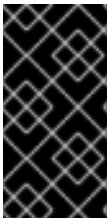
Forward one or more local ports to a pod:

```
$ 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>
```



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.

4.9. TROUBLESHOOTING OC

You can get more verbosed output from any command by increasing the loglevel using **-v=X** flag. By default, the loglevel is set to **0**, but you can set its value from **0** to **10**.

Overview of each loglevel

- **1**-**5** - are usually used internally by the commands, if the author decides to provide more explanation about the flow.
- **6** - provides basic information about HTTP traffic between the client and the server, such HTTP operation and URL.
- **7** - provides more thorough HTTP information, such as HTTP operation, URL, request headers and response status code.
- **8** - provides full HTTP request and response, including body.
- **9** - provides full HTTP request and response, including body and sample **curl** invocation.
- **10** - provides all possible output the command provides.

CHAPTER 5. ADMINISTRATOR CLI OPERATIONS

5.1. OVERVIEW

This topic provides information on the administrator CLI operations and their syntax. You must setup and login with the CLI before you can perform these operations.

The **oc adm** command (formerly the **oadm** command) is used for administrator CLI operations. The administrator CLI differs from the normal set of commands under the developer CLI, which uses the **oc** command, and is used more for project-level operations.

NOTE

Your login may or may not have access to the following administrative commands, depending on your account type.

5.2. COMMON OPERATIONS

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

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

This specifies:

- An **<action>** to perform, such as **new-project** or **groups**.
- An available **<option>** to perform the action on as well as a value for the option. Options include **--output**.

IMPORTANT

When running **oc adm** commands, you should run them only from the first master listed in the Ansible host inventory file, by default */etc/ansible/hosts*.

5.3. BASIC CLI OPERATIONS

5.3.1. new-project

Creates a new project:

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

5.3.2. policy

Manages authorization policies:

```
$ oc adm policy
```

5.3.3. groups

Manages groups:

```
$ oc adm groups
```

5.4. MAINTENANCE CLI OPERATIONS

5.4.1. build-chain

Outputs the inputs and dependencies of any builds:

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