OpenShift Container Platform 3.11

CLI 参考

OpenShift Container Platform 3.11 CLI 参考

Last Updated: 2023-03-25

OpenShift Container Platform 3.11 CLI 参考

OpenShift Container Platform 3.11 CLI 参考

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

摘要

使用 OpenShift Container Platform 命令行界面(CLI)，您可以从终端创建应用程序并管理 OpenShift 项目。这些主题介绍了如何使用 CLI。

目录

第 1 章 概述

使用 OpenShift Container Platform 命令行界面(CLI)，您可以从 终端创建应用程序并管理 OpenShift Container Platform 项目。CLI 适用于以下情况：

- 直接使用项目源代码。
- 编写 OpenShift Container Platform 操作脚本。
- 受带宽资源限制，无法使用 Web 控制台。

CLI 使用 **oc** 命令可用：

```
$ oc <command>
```

有关安装和设置的信息，请参阅 CLI 入门。

第 2 章 CLI 入门

2.1. 概述

OpenShift Container Platform CLI 提供了管理应用程序的命令，以及较低级别的工具与系统的每个组件交互。本主题指导您完成 CLI 入门，包括安装并登录以创建您的第一个项目。

2.2. 先决条件

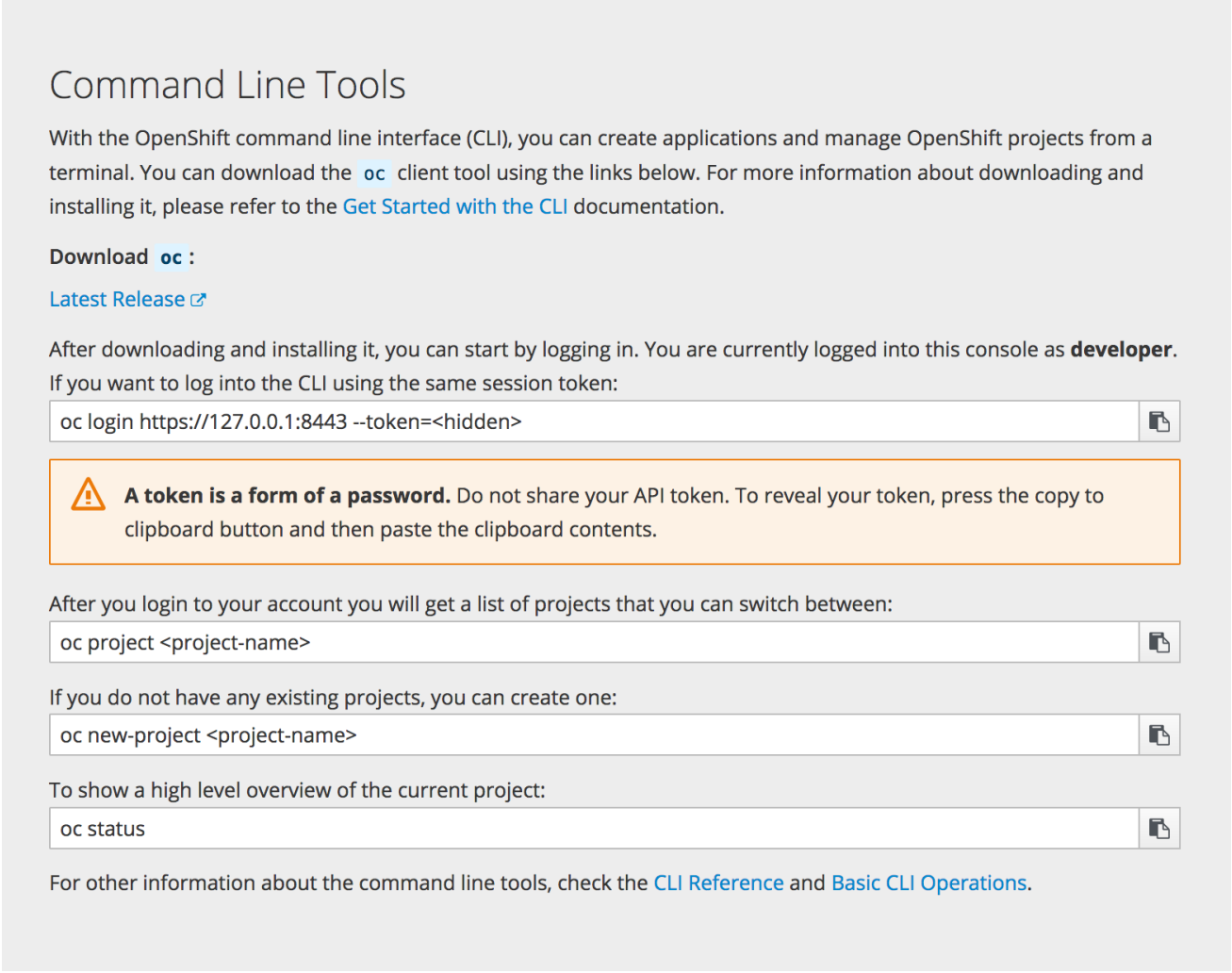
某些操作要求在客户端上本地安装 Git。例如，命令使用远程 Git 存储库创建应用程序：

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

在继续之前，请先在工作站上安装 Git。如需了解工作站操作系统的说明，请参阅官方 Git 文档。

2.3. 安装 CLI

如果集群管理员启用了下载链接，则下载 CLI 的最简单方法是访问 web 控制台中的 **About** 页面：



CLI 的安装选项会因您的操作系统而异。

要使用 CLI 登录，请从 Web 控制台的 **Command Line** 页面收集您的令牌，该页可通过 **Help** 菜单的 **Command Line Tools** 访问。令牌是隐藏的，因此您必须点击 **Command Line Tools** 页面中 **oc login** 行末尾的 **copy to clipboard** 按钮，然后粘贴复制的内容以显示令牌。

2.3.1. 对于 Windows

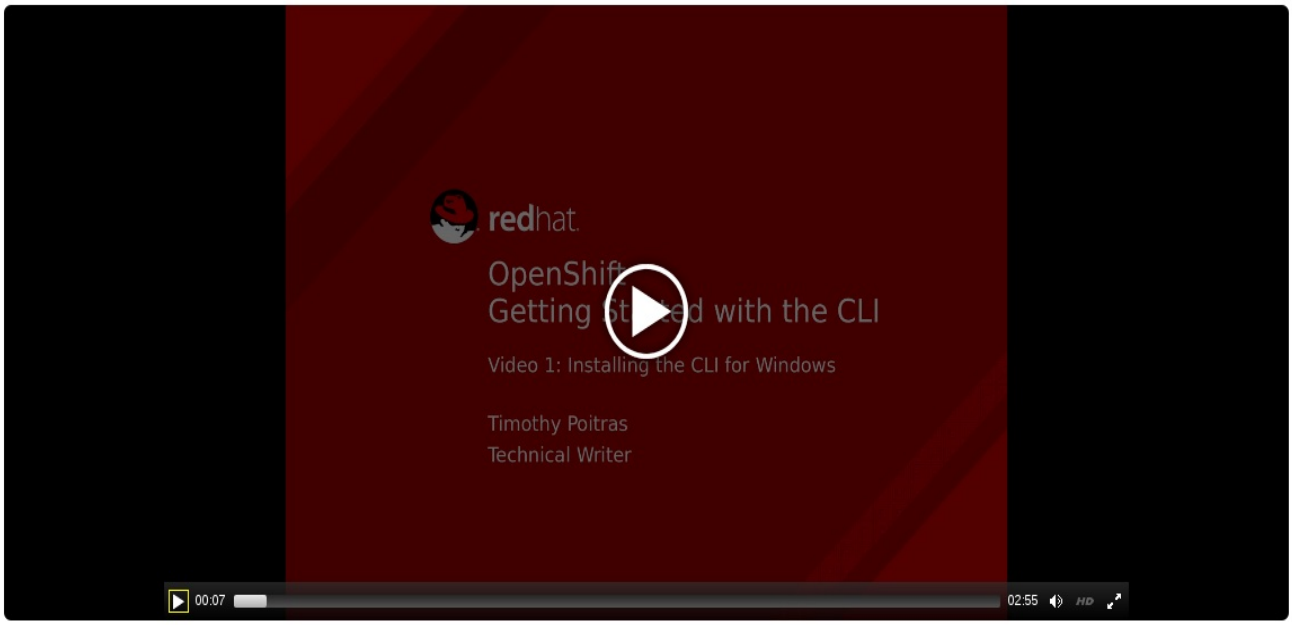
Windows 的 CLI 以 ***zip*** 存档形式提供，您 可以从红帽客户门户下载。使用您的红帽帐户登录后，您必须有一个活跃的 OpenShift Enterprise 订阅才能访问下载页面：

从红帽客户门户网站下载 CLI

另外，如果集群管理员启用了它，您可以在 web 控制台的 **About** 页面中下载并解压缩 CLI。

教程视频：

以下视频将引导您完成此过程： 单击此处观看



然后，使用 ZIP 程序解压存档，并将 **oc** 二进制文件移到 PATH 的目录中。要查看您的 PATH，请打开命令提示并运行：

```
C:\> path
```

2.3.2. 对于 Mac OS X

Mac OS X 的 CLI 以 ***tar.gz*** 存档形式提供，您 可以从红帽客户门户下载。使用您的红帽帐户登录后，您必须有一个活跃的 OpenShift Enterprise 订阅才能访问下载页面：

从红帽客户门户网站下载 CLI

另外，如果集群管理员启用了它，您可以在 web 控制台的 **About** 页面中下载并解压缩 CLI。

教程视频：

以下视频将引导您完成此过程： 单击此处观看

然后，解包存档，并将 **oc** 二进制文件移到 PATH 的目录中。要查看您的 PATH，请打开终端窗口并运行：

```
$ echo $PATH
```

2.3.3. 对于 Linux

对于 Red Hat Enterprise Linux(RHEL)7，如果您的红帽帐户上已有有效的 OpenShift Enterprise 订阅，则可以使用 Red Hat Subscription Management(RHSM)安装 CLI：

1. 使用 Red Hat Subscription Manager 注册：

```
# subscription-manager register
```

2. 获取最新的订阅数据：

```
# subscription-manager refresh
```

3. 在注册的系统中添加订阅：

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

1 活跃的 OpenShift Enterprise 订阅的池 ID

4. 启用 OpenShift Container Platform 3.11 所需的存储库：

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

5. 安装 **atomic-openshift-clients** 软件包：

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

对于 RHEL、Fedora 和其他 Linux 发行版本，您还可以直接从红帽客户门户网站下载 CLI 作为 ***tar.gz*** 存档。https://access.redhat.com/downloads/content/290使用您的红帽帐户登录后，您必须有一个有效的 OpenShift Enterprise 订阅才能访问下载页面。

从红帽客户门户网站下载 CLI

教程视频：

以下视频将引导您完成此过程： 单击此处观看

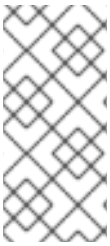
另外，如果集群管理员启用了它，您可以在 web 控制台的 **About** 页面中下载并解压缩 CLI。

然后，解包存档，并将 **oc** 二进制文件移到 PATH 的目录中。要查看路径，请运行：

```
$ echo $PATH
```

解包存档：

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



注意

如果不使用 RHEL 或 Fedora，请确保将 **libc** 安装在库路径的目录中。如果 **libc** 不可用，您在运行 CLI 命令时可能会看到以下错误：

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

2.4. 基本设置和登录

oc login 命令是初始设置 CLI 的最佳方法，它充当大部分用户的入口点。互动流程可帮助您使用提供的凭证建立与 OpenShift Container Platform 服务器的会话。信息自动保存在 CLI 配置文件中，然后用于后续的命令。

以下示例显示了使用 **oc login** 命令进行交互式设置和登录：

例 2.1. 初始 CLI 设置

```
$ oc login
```

输出示例

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

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

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

    $ oc new-project <projectname> (4)

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

1. 该命令提示输入 OpenShift Container Platform 服务器 URL。
2. 命令会提示输入登录凭证：用户名和密码。
3. 会话会根据服务器建立，并接收会话令牌。
4. 如果您没有项目，则会提供关于如何创建项目的信息。

完成 CLI 配置后，后续的命令使用服务器、会话令牌和项目信息的配置文件。

您可以使用 **oc logout** 命令从 CLI 注销：

```
$ oc logout
```

输出示例

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

如果您在创建或被授予了项目访问权限后登录，则您有权访问的项目会自动设置为当前默认值，直到 切换到另一个 项目：

```
$ oc login
```

输出示例

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

Using project "aliceproject".
```

其他选项 也可用于 **oc login** 命令。

注意

如果您可以访问管理员凭据，但不再作为默认系统用户 **system:admin** 登录，只要仍存在于 CLI 配置文件 中，您可以随时以这个用户身份登录。以下命令登录并切换到 **default** 项目 ：

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

2.5. CLI 配置文件

CLI 配置文件会永久存储 **oc** 选项，并包含一系列与 nicknames 相关的 身份验证机制 和 OpenShift Container Platform 服务器连接信息。

如上一节中所述，**oc login** 命令会自动创建和管理 CLI 配置文件。通过该命令收集的所有信息都存储在 *~/.kube/config* 下的配置文件中。使用以下命令可以查看当前的 CLI 配置：

例 2.2. 查看 CLI 配置

```
$ oc config view
```

输出示例

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

CLI 配置文件可用于使用各种 OpenShift Container Platform 服务器、命名空间和用户 设置多个 CLI 配置集，以便可以在它们间轻松切换。CLI 可以支持多个配置文件；它们在运行时加载，并合并在一起，以及从命令行指定的覆盖选项。

2.6. 项目

OpenShift Container Platform 中的项目 包含多个 对象，组成一个逻辑应用程序。

大多数 **oc** 命令 在项目 的上下文中运行。**oc login** 在 初始设置 过程中选择用于后续命令的默认项目。使用以下命令显示当前正在使用的项目：

```
$ oc project
```

如果可以访问多个项目，请使用以下语法通过指定项目名称切换到特定项目：

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

例如：

切换到 Project project02

```
$ oc project project02
```

输出示例

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

切换到 Project project03

```
$ oc project project03
```

输出示例

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

列出当前项目

```
$ oc project
```

输出示例

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

oc status 命令显示当前正在使用的项目的高级概述，及其组件及其关系，如下例所示：

```
$ oc status
```

输出示例

```
In project OpenShift 3 Sample (test)

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

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

```
    #1 deployment waiting on image

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

2.7. 下一步是什么？

登录 后，您可以 创建新应用 并探索一些常见 CLI 操作。

第 3 章 管理 CLI 配置集

3.1. 概述

CLI 配置文件允许您配置不同的配置文件或 *上下文*，以用于 OpenShift CLI。上下文由与 *nickname* 关联的 用户身份验证 和 OpenShift Container Platform 服务器信息组成。

3.2. 在 CLI 配置集间切换

通过上下文，您可以在使用多个 OpenShift Container Platform 服务器或使用 CLI 操作时轻松地切换多个用户。*nicknames 通过提供对上下文、用户凭证和集群详情的简短参考，从而更轻松地管理 CLI 配置。*

*第一次使用 CLI 登录后，OpenShift Container Platform 会创建一个 **~/.kube/config** 文件（如果不存在）。随着更多身份验证和连接详情被提供给 CLI，在 **oc login** 操作或 显式设置 时，更新的信息会存储在配置文件中：*

例 3.1. CLI 配置文件

```
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) ***clusters** 部分定义 OpenShift Container Platform 集群的连接详情，包括其 master 服务器的地址。在本例中，一个集群名为 nicknamed **openshift1.example.com:8443**，另一个是 nicknamed **openshift2.example.com:8443**。*

(2)

此上下文 **部分定义** 两个上下文：一个 nicknamed **alice-project/ openshift1.example.com:8443 /alice**，使用 **alice-project** 项目、openshift1.example.com:8443 集群和 **alice** 用户，另一个

3 **current-context** 参数显示 **joe-project/openshift1.example.com:8443/alice** 上下文当前正在使用中，允许 **alice** 用户在 **openshift1.example.com:8443** 集群上的 **joe-project** 项目中工作。

4 **users** 部分定义用户凭据。在本例中，用户 nickname **alice/openshift1.example.com:8443** 使用访问令牌。

CLI 可以支持多个配置文件；它们在运行时加载，并合并在一起，以及从命令行指定的覆盖选项。

登录后，您可以使用 **oc status** 命令或 **oc project** 命令验证您当前的工作环境：

例 3.2. 验证当前工作环境

```
$ oc status
```

输出示例

```
oc status
In project Joe's Project (joe-project)

service database (172.30.43.12:5434 -> 3306)
  database deploys docker.io/openshift/mysql-55-centos7:latest
    #1 deployed 25 minutes ago - 1 pod

service frontend (172.30.159.137:5432 -> 8080)
  frontend deploys origin-ruby-sample:latest <-
    builds https://github.com/openshift/ruby-hello-world with joe-project/ruby-20-centos7:latest
    #1 deployed 22 minutes ago - 2 pods

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

列出当前项目

```
$ oc project
```

输出示例

```
Using project "joe-project" from context named "joe-project/openshift1.example.com:8443/alice" on server "https://openshift1.example.com:8443".
```

要使用用户凭证和集群详情的其他组合登录，请再次运行 **oc login** 命令并在互动过程中提供相关信息。基于提供的信息构建上下文（如果尚不存在）。

如果您已经登录，并希望切换到当前用户已有权访问的另一个项目，请使用 **oc project** 命令并提供项目的名称：

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

输出示例

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

在任何时候，您可以使用 **oc config view** 命令查看当前的完整 CLI 配置，如输出中所示。

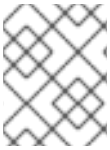
其他 CLI 配置命令也可用于 更高级的使用。

注意

如果您可以访问管理员凭据，但不再作为默认系统用户 **system:admin** 登录，只要仍存在于 CLI 配置文件 中，您可以随时以这个用户身份登录。以下命令登录并切换到 **default** 项目：

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

3.3. 手动配置 CLI 配置集



注意

本节介绍 CLI 配置的更多高级用法。在大多数情况下，您只需使用 **oc login** 和 **oc project** 命令登录并在上下文和项目间切换。

如果要手动配置 CLI 配置文件，您可以使用 **oc config** 命令而不是修改文件本身。**oc config** 命令包括很多有用的子命令来实现这一目的：

表 3.1. CLI 配置子命令

子命令	使用方法
set-cluster	在 CLI 配置文件中设置集群条目。如果引用的 cluster nickname 已存在，则指定的信息将合并到其中。 `$ 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	在 CLI 配置文件中设置上下文条目。如果引用的上下文 nickname 已存在，则指定的信息将合并在. `$ oc config set-context <context_nickname> [--cluster=<cluster_nickname>] [--user=<user_nickname>] [--namespace=<namespace>]`
use-context	使用指定上下文 nickname 设置当前上下文。 `$ oc config use-context <context_nickname>`

子命令	使用方法
set	在 CLI 配置文件中设置单个值。 `$ oc config set <property_name> <property_value>` **<property_name>** 是一个以点分隔的名称，每个令牌代表属性名称或映射键。**<property_value>** 是要设置的新值。
unset	在 CLI 配置文件中取消设置各个值。 `$ oc config unset <property_name>` **<property_name>** 是一个以点分隔的名称，每个令牌代表属性名称或映射键。

view	显示当前正在使用的合并 CLI 配置。
	
	`$ oc config view`
	
	显示指定 CLI 配置文件的结果。
	
	`$ oc config view --config=<specific_filename>`

示例用法

*考虑以下配置工作流：首先，以使用 访问令牌 的用户身份登录。**alice** 用户使用此令牌：*

```
$ oc login https://openshift1.example.com --
token=ns7yVhuRNpDM9cgzfhhxQ7bM5s7N2ZVrkZepSRf4LC0
```

查看自动创建的集群条目：

```
$ oc config view
```

输出示例

```
apiVersion: v1
clusters:
- cluster:
    insecure-skip-tls-verify: true
    server: https://openshift1.example.com
  name: openshift1-example-com
contexts:
- context:
    cluster: openshift1-example-com
    namespace: default
    user: alice/openshift1-example-com
  name: default/openshift1-example-com/alice
current-context: default/openshift1-example-com/alice
kind: Config
preferences: {}
```

```
users:
- name: alice/openshift1.example.com
  user:
    token: ns7yVhuRNpDM9cgzfhhxQ7bM5s7N2ZVrkZepSRf4LC0
```

更新当前上下文以便用户登录到所需的命名空间：

```
$ oc config set-context `oc config current-context` --namespace=<project_name>
```

要确认更改已生效，请检查当前上下文：

```
$ oc whoami -c
```

所有后续 CLI 操作都将使用新的上下文，除非通过覆盖 CLI 选项或直至上下文切换为止。

3.4. 载入和合并规则

在发出 CLI 操作时，CLI 配置的加载和合并顺序遵循这些规则：

1. 使用以下层次结构和合并规则从工作站检索 CLI 配置文件：
   - 如果设置了 **--config** 选项，则只加载该文件。标志可能仅设置为一次，也没有合并发生。
   - 如果设置了 **$KUBECONFIG** 环境变量，则会使用它。变量可以是路径列表，如果将路径合并在一起。修改值后，会在定义该节的文件中对其进行修改。创建值时，会在存在的第一个文件中创建它。如果链中不存在任何文件，则会在列表中创建最后一个文件。
   - 否则，将使用 **~/.kube/config** 文件，且不会发生合并。
2. 要使用的上下文根据以下链中的第一个点击来确定：
   - **--context** 选项的值。
   - CLI 配置文件中的 **current-context** 值。
   - 此阶段允许一个空值。
3. 要使用的用户和集群是决定的。此时，您可能也可能没有上下文；它们基于以下链中的第一个按位置构建，该链为用户运行一次，一次用于集群：
   - 用户名的 **--user** 选项的值以及集群名称的 **--cluster** 选项。
   - 如果存在 **--context** 选项，则使用上下文的值。
   - 此阶段允许一个空值。
4. 要使用的实际集群信息决定。此时，您可能没有集群信息。集群信息的每个信息根据以下链中的第一个点击构建：
   - 以下命令行选项中的任何值：
     - **--server**,

    - ***--api-version***
    - ***--certificate-authority***
    - ***--insecure-skip-tls-verify***
  - *如果集群信息和属性的值存在，则使用它。*
  - *如果您没有服务器位置，则出现错误。*

5. *要使用的实际用户信息是确定的。用户使用与集群相同的规则构建，但每个用户只能有一个身份验证技术；冲突的技术会导致操作失败。命令行选项优先于配置文件值。有效命令行选项包括：*
   - ***--auth-path***
   - ***--client-certificate***
   - ***--client-key***
   - ***--token***

6. *对于仍缺失的任何信息，将使用默认值，并提示提供其他信息。*

第 4 章 开发人员 CLI 操作

4.1. 概述

本节提供有关开发人员 CLI 操作及其语法的信息。您必须先设置 并使用 CLI 登录，然后才能执行这些操作。

开发人员 CLI 使用 **oc** 命令，用于项目级操作。这与管理员 CLI 不同，它使用 **oc adm** 命令进行更高级的管理员操作。

4.2. 常见操作

开发人员 CLI 允许与 OpenShift Container Platform 管理的各种对象交互。许多常见 **oc** 操作都使用以下语法调用：

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

这将指定：

- 要执行的 **<action >**，如 **get** 或 **describe**。
- 要执行 **操作的 <object_type >**，如 **service** 或缩写 **svc**。
- 指定 & **lt;object_ type> 的 <object_name &gt**;。

例如，**oc get** 操作返回当前定义的服务的完整列表：

```
$ oc get svc
```

输出示例

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

然后，**oc describe** 操作可用于返回有关特定对象的详细信息：

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

输出示例

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

> *Endpoints: 10.128.0.2:5000*
> *Session Affinity: None*
> *No events.*

4.3. 对象类型

以下是 CLI 支持的最常用的对象类型列表，其中一些具有简写语法：

对象类型	简写版本
Build	
BuildConfig	**bc**
DeploymentConfig	**dc**
部署	**deploy**
事件	**ev**
ImageStream	**is**
ImageStreamTag	**istag**
ImageStreamImage	**isimage**
Job	
CronJob （技术预览）	**cj**
LimitRange	**limits**
节点	
Pod	**PO**
ResourceQuota	**quota**
ReplicationController	**rc**
replicaSet	**rs**
Secrets	
Service	**svc**
ServiceAccount	**sa**

对象类型	简写版本
StatefulSets	**STS**
PersistentVolume	**pv**
PersistentVolumeClaim	**pvc**

如果要知道服务器支持的资源的完整列表，请使用 ***oc api-resources****。*

4.4. 基本 CLI 操作

下表描述了基本的 ***oc*** *操作及其一般语法：*

4.4.1. 类型

显示一些 OpenShift Container Platform 核心概念的介绍：

```
$ oc types
```

4.4.2. login

登录到 OpenShift Container Platform 服务器：

```
$ oc login
```

4.4.3. logout

结束当前会话：

```
$ oc logout
```

4.4.4. new-project

创建一个新项目

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

4.4.5. new-app

根据当前目录中的源代码创建新应用： https://access.redhat.com/documentation/en-us/openshift_container_platform/3.11/html-single/developer_guide/#dev-guide-new-app

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

根据远程存储库中的源代码创建新应用：

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

根据私有远程存储库中的源代码创建新应用：

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

4.4.6. status

显示当前项目的概述：

```
$ oc status
```

4.4.7. project

切换到另一个项目。运行 （不带选项）以显示当前项目。要查看您有权运行 **oc projects** 的所有项目。

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

4.5. 应用修改操作

4.5.1. get

返回 指定对象类型的 对象列表。如果请求中包含可选的 &lt ;**object_name** >，则会根据该值过滤结果列表。

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

例如，以下命令列出项目的可用镜像：

```
$ oc get images
```

输出示例

```
sha256:f86e02fb8c740b4ed1f59300e94be69783ee51a38cc9ce6ddb73b6f817e173b3
registry.redhat.io/jboss-datagrid-6/datagrid65-
openshift@sha256:f86e02fb8c740b4ed1f59300e94be69783ee51a38cc9ce6ddb73b6f817e173b3
sha256:f98f90938360ab1979f70195a9d518ae87b1089cd42ba5fc279d647b2cb0351b
registry.redhat.io/jboss-fuse-6/fis-karaf-
openshift@sha256:f98f90938360ab1979f70195a9d518ae87b1089cd42ba5fc279d647b2cb0351b
```

您可以使用 **-o** 或 **--output** 选项修改输出格式。

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

输出格式可以是 JSON 或 YAML，也可以是 自定义列、golang 模板 和 jsonpath 等可扩展格式。

例如，以下命令列出了特定项目中运行的 pod 的名称：

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

-

输出示例

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

4.5.2. describe

返回有关查询返回的特定对象的信息。必须提供特定的 **<object_name** >。可用的实际信息因 对象类型 所述而异。

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

4.5.3. 编辑

编辑所需的对象类型：

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

使用指定的文本编辑器编辑所需的对象类型：

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

以指定格式（如 JSON）编辑所需的对象：

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

4.5.4. 卷

修改 卷 ：

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

4.5.5. label

更新对象上的标签：

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

4.5.6. expose

查找服务并将其公开为路由。另外，也可以在指定端口上公开部署配置、复制控制器、服务或 pod 作为新服务。如果没有指定标签，新对象将重新使用其公开对象中的标签。

如果您要公开服务，则默认生成器是 **--generator=route/v1**。对于所有其他情况下，默认为 **--generator=service/v2**，这会留下端口未命名。通常，不需要使用 **oc expose** 命令设置生成器。第三个生成器 **--generator=service/v1** 提供了端口名称 default。

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

4.5.7. 删除

删除指定的对象。对象配置也可通过 STDIN 传递到。**oc delete all -l <label>** 操作将删除与指定 &**lt;label>** 匹配的所有对象，包括 复制控制器，以便 pod 不会重新创建。

```
$ 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

修改指定对象的特定属性。

4.5.8.1. 设置 env

在部署配置或构建配置上设置环境变量：

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

4.5.8.2. 设置 build-secret

在构建配置中设置 secret 的名称。secret 可以是镜像 pull 或 push secret 或源存储库 secret：

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

4.6. 构建和部署操作

OpenShift Container Platform 的一项基本功能是能够从源构建应用程序到容器中。

OpenShift Container Platform 通过标准 **oc** 资源操作（如 **get**、**创建** 和 **describe**）提供 CLI 访问来检查和操作部署配置。

4.6.1. start-build

使用指定的构建配置文件手动启动构建流程：

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

将上一构建的名称指定为起点，手动启动构建流程：

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

通过指定配置文件或上一个构建的名称并检索其构建日志来手动启动构建流程：

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

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

如果构建失败，请等待构建完成，然后以非零返回代码退出：

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

设置或覆盖当前构建的环境变量，而不更改构建配置。或者，使用 **-e**。

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

在构建期间设置或覆盖默认构建日志级别输出：

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

指定构建应使用的源代码提交标识符；需要基于 Git 存储库的构建：

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

使用名称 < **build_name&** gt; 重新运行构建：

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

归档 **<dir_name** > 并将其作为二进制输入进行构建：

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

使用现有存档作为二进制输入；与 **--from-file** 不同，构建过程前由构建器提取存档：

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

使用 **<file_name** > 作为构建的二进制输入。此文件必须是构建源中唯一一个。例如： **pom.xml** 或 **Dockerfile**。

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

使用 HTTP 或 HTTPS 下载二进制文件输入，而不是从文件系统中读取它：

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

下载存档并使用其内容作为构建源：

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

用作构建二进制输入的本地源代码存储库的路径：

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

为要触发的现有构建配置指定 Webhook URL：

-

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

触发构建的 post-receive hook 的内容：

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

post-receive 的 Git 存储库的路径；默认为当前目录：

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

列出指定构建配置或构建的 webhook；**接受所有**、**通用** 或 **github** ：

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

覆盖 source-strategy 构建的 **Spec.Strategy.SourceStrategy.Incremental** 选项：

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

覆盖 docker-strategy 构建的 **Spec.Strategy.DockerStrategy.NoCache** 选项：

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

4.6.2. rollback

执行 回滚 ：

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

4.6.3. new-build

根据当前 Git 存储库中的源代码（具有公共远程）和容器镜像创建构建配置：

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

基于远程 git 存储库创建构建配置：

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

基于私有远程 git 存储库创建构建配置：

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

4.6.4. cancel-build

停止正在进行中的构建：

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

同时取消多个构建：

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

取消从构建配置创建的所有构建：

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

指定要取消的构建：

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

状态的示例值为 new 或 pending。

4.6.5. import-image

从外部镜像存储库导入标签和镜像信息：

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

4.6.6. scale

将 复制控制器或部署配置 所需的副本数设置为指定副本数：

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

4.6.7. tag

从镜像流或容器镜像"pull spec"中提取现有标签或镜像，并将其设置为一个或多个镜像流中标签的最新镜像：

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

4.7. 高级命令

4.7.1. create

解析配置文件，并根据文件内容创建一个或多个 OpenShift Container Platform 对象。**f 标志** 可多次传递，不同的文件或目录路径。当多次传递该标志时，**oc create** 会逐一迭代，创建所有指定文件中描述的对象。任何现有的资源将被忽略。

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

4.7.2. replace

尝试根据指定配置文件的内容修改现有对象。**f 标志** 可多次传递，不同的文件或目录路径。当多次传递该标志时，**oc replace** 会逐一迭代，更新所有指定文件中描述的对象。

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

4.7.3. process

将 项目模板 转换为项目配置文件：

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

4.7.4. run

创建并运行特定的镜像（可能复制）。默认情况下，创建部署配置以管理所创建的容器。您可以选择使用 **--generator** 标志创建不同的资源：

API 资源	--generator 选项
部署配置	**DeploymentConfig/v1** （默认）
Pod	**run-pod/v1**
复制控制器	**run/v1**
使用 **extensions/v1beta1** 端点进行部署	**deployment/v1beta1**
使用 **apps/v1beta1** 端点进行部署	**deployment/apps.v1beta1**
Job	**job/v1**
Cron job	**cronjob/v2alpha1**

您可以选择在前台运行，以进行交互式容器执行。

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

4.7.5. patch

使用策略合并补丁更新对象的一个或多个字段：

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

<changes> 是一个 JSON 或 YAML 表达式，其中包含新字段和值。例如，要将节点 **node1** 的 **spec.unschedulable** 字段更新为值 **true**，json 表达式为：

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

4.7.6. policy

管理授权策略：

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

4.7.7. secrets

配置 secret ：

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

4.7.8. autoscale

为您的应用程序设置 自动扩展器。需要在集群中启用指标。如果需要，请参阅为集群管理员说明 启用集群指标。

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

4.8. 故障排除和调试操作

4.8.1. debug

启动一个 shell 以调试正在运行的应用程序。

```
$ oc debug -h
```

调试镜像并设置问题时，您可以获取正在运行的 pod 配置的确切副本，并使用 shell 进行故障排除。因为一个失败的 pod 可能无法启动且无法被 **rsh** 或 **exec** 访问，运行 **debug** 命令会创建那个设置的 carbon 副本。

默认模式是在引用的 pod、复制控制器或部署配置的第一个容器内启动 shell。启动的 pod 将是源 pod 的副本，其标签会剥离，命令的变为 **/bin/sh**，并且禁用就绪度和存活度检查。如果您只想运行命令，请添加 **--** 和要运行的命令。默认情况下，传递命令将不会创建 TTY 或发送 STDIN。其他标记支持以常见方式更改容器或 pod。

运行容器的常见问题是一个安全策略，它会阻止您在集群中以 root 用户身份运行。您可以使用此命令测试以非 root 运行 pod（使用 **--as-user**）或以 root 身份运行非 root pod（使用 **--as-root**）。

当远程命令完成或中断 shell 时，调试 pod 将被删除。

4.8.1.1. 用法

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

4.8.1.2. 示例

调试当前运行的部署：

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

要测试以非 root 用户身份运行部署：

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

要在 **第二个** 容器中运行 **env** 命令来调试特定的故障容器：

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

查看创建要调试的 pod：

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

4.8.2. logs

检索特定构建、部署或 Pod 的日志输出。此命令适用于构建、构建配置、部署配置和容器集。

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

4.8.3. exec

在已在运行的容器中执行命令。您可以选择指定容器 ID，否则默认为第一个容器。

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

4.8.4. rsh

打开到容器的远程 shell 会话：

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

4.8.5. rsync

将内容复制到已在运行的 pod 容器中的某一目录，或从中复制内容。如果没有指定容器，则默认为 pod 中的第一个容器。

将本地目录中的内容复制到 pod 中的目录中：

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

将 pod 中某一目录的内容复制到本地目录：

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

4.8.6. port-forward

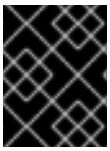
将一个或多个本地端口转发到一个 pod:

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

4.8.7. proxy

运行到 Kubernetes API 服务器的代理：

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



重要

为了安全起见，**oc exec** 命令在访问特权容器时无法工作，除非该命令由 **cluster-admin** 用户执行。管理员可以 SSH 到节点主机，然后在所需容器上使用 **docker exec** 命令。

4.9. OC 故障排除

您可以通过使用 **-v=X** 标志增加日志级别来从任何命令获得更详细的输出。默认情况下，日志级别设置为 0，但您可以将其值从 **0** 设置为 **10**。

每个日志级别概述

- **1-5** - 如果作者决定提供更多有关流的说明，命令通常在内部使用 5。
- **6** - 提供有关客户端与服务器之间 HTTP 流量的基本信息，如 HTTP 操作和 URL。
- **7** - 提供更加彻底的 HTTP 信息，如 HTTP 操作、URL、请求标头和响应状态代码。
- **8** - 提供完整的 HTTP 请求和响应，包括正文。
- **9** - 提供完整的 HTTP 请求和响应，包括正文和示例 **curl** 调用。
- **10** - 提供命令输出。

第 5 章 管理员 CLI 操作

5.1. 概述

本主题提供了管理员 CLI 操作及其语法的信息。您必须先设置 并使用 CLI 登录，然后才能执行这些操作。

openshift 命令用于启动组成 OpenShift Container Platform 集群的服务。例如，openshift 启动 **[master|node]**。但是，它还是一个一体化命令，它可以分别通过 **openshift cli** 和 **openshift admin** 执行 **oc** 和 **oc adm** 命令的所有操作。

管理员 CLI 与开发人员 CLI 下的一组普通命令不同，它使用 **oc** 命令，并将更多用于项目级别操作。

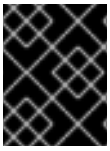
5.2. 常见操作

管理员 CLI 允许与 OpenShift Container Platform 管理的各种对象交互。许多常见 **oc adm** 操作都使用以下语法调用：

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

这将指定：

- 要执行的 **<action >**，如 **new-project** 或 组。
- 一个可用的 & **lt;option >**，用于对 和 选项的值执行操作。选项包括 **--output**。



重要

运行 **oc adm** 命令时，您应该只从 Ansible 主机清单文件中列出的第一个 master 运行它们，默认为 **/etc/ansible/hosts**。

5.3. 基本 CLI 操作

5.3.1. new-project

创建一个新项目

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

5.3.2. policy

管理授权策略：

```
$ oc adm policy
```

5.3.3. groups

管理组：

```
$ oc adm groups
```

5.4. 安装 CLI 操作

5.4.1. 路由器

安装路由器：

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

5.4.2. ipfailover

为一组节点安装 IP 故障转移组：

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

5.4.3. registry

安装集成的容器镜像 registry：

```
$ oc adm registry
```

5.5. 维护 CLI 操作

5.5.1. build-chain

输出构建的输入和依赖项：

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

5.5.2. manage-node

管理节点。例如，列出或撤离 pod，或者标记它们就绪：

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

5.5.3. prune

从服务器中删除旧版本的资源：

```
$ oc adm prune
```

5.6. 设置 CLI 操作

5.6.1. config

更改 kubelet 配置文件：

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

5.6.2. create-kubeconfig

从客户端证书创建基本 .kubeconfig 文件：

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

5.6.3. create-api-client-config

创建用于以用户连接到服务器的配置文件：

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

5.7. 高级 CLI 操作

5.7.1. create-bootstrap-project-template

创建 bootstrap 项目模板：

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

5.7.2. create-bootstrap-policy-file

创建默认 bootstrap 策略：

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

5.7.3. create-login-template

创建登录模板：

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

5.7.4. create-node-config

为节点创建配置捆绑包：

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

5.7.5. ca

管理证书和密钥：

```
$ oc adm ca
```

第 6 章 OC 和 KUBECTL 之间的区别

6.1. 为什么使用 OC OVER KUBECTL ？

Kubernetes 的命令行界面(CLI) **kubectl** 用于针对任何 Kubernetes 集群运行命令。由于 OpenShift Container Platform 在 Kubernetes 集群之上运行，所以 **kubectl** 也会包含在 **oc** 中，OpenShift Container Platform 的命令行界面(CLI)中也包括 kubectl。

虽然这两个客户端之间有多个相似点，但本指南的准备是阐明以后使用的主要原因和场景。

6.2. 使用 OC

oc 二进制文件提供与 **kubectl** 二进制文件相同的功能，但它进一步扩展为原生支持 OpenShift Container Platform 的功能，例如：

完全支持 OpenShift 资源

DeploymentConfig、**BuildConfig s**、Route、**ImageStream** s 和 **ImageStreamTag** 等资源特定于 OpenShift 发行版，而标准 Kubernetes 中不可用。

身份验证

oc 二进制文件提供了一个内置的 **登录** 命令，允许进行身份验证。如需更多信息，请参阅 *开发人员 身份验证和配置身份验证* 。

附加命令

例如，附加命令 **new-app** 可以更轻松地使用现有源代码或预构建镜像来启动新应用程序。

6.3. 使用 KUBECTL

提供 **kubectl** 二进制文件的目的是为来自标准 **Kubernetes** 环境的新 **OpenShift Container Platform** 用户支持现有的工作流和脚本。**kubectl** 的现有用户可以继续使用该二进制文件，而无需更改 **API**，但应该考虑升级到 **oc** 来获取上一节中提到的功能。

因为 **oc** 基于 **kubectl**，所以将 **kubectl** 二进制文件转换为 **oc** 非常简单，就像将二进制名称从 **kubectl** 改为 **oc** 一样简单。

有关安装和设置的信息，请参阅 ***CLI 入门***。

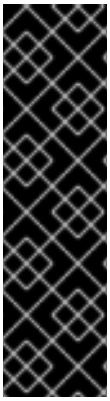
第 7 章 扩展 CLI

7.1. 概述

本节复习如何为 CLI 安装和编写扩展。通常称为 插件（plug-in） 或 二进制扩展（binary extension），这个功能允许您扩展默认可用的 oc 命令集合，以执行新的任务。

插件是一组文件：通常至少有一个 plugin.yaml 描述符以及一个或多个二进制文件、脚本或资产文件。

CLI 插件目前仅在 oc plugin 子命令下可用。



重要

CLI 插件目前还是一个技术预览功能。技术预览功能不包括在红帽生产服务级别协议（SLA）中，且其功能可能并不完善。因此，红帽不建议在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关详细信息，请参阅红帽技术预览功能支持范围 。

7.2. 先决条件

您必须：

- 安装了可正常工作的 oc 二进制文件。

7.3. 安装插件

将插件的 plugin.yaml 描述符、二进制文件、脚本和资产文件复制到 oc 搜索插件的一个位置。

目前，OpenShift Container Platform 不会为插件提供软件包管理器。因此，您的责任将插件文件放在正确的位置。建议每个插件位于其自己的目录中。

要安装作为压缩文件分发的插件，请将其提取到在 Plug-in Loader 部分中指定的其中一个位置。

7.3.1. Plug-in Loader

插件加载程序负责 搜索插件文件，并检查插件是否提供运行所需的最小信息。放置在正确位置的文件不提供最小信息（例如，不完整的 plugin.yaml 描述符）。

7.3.1.1. 搜索顺序

插件加载程序使用以下搜索顺序：

1. ${KUBECTL_PLUGINS_PATH}

   如果指定，则搜索将在此处停止。

   如果 KUBECTL_PLUGINS_PATH 环境变量存在，则加载程序会将其用作查找插件的唯一位置。KUBECTL_PLUGINS_PATH 环境变量是目录列表。在 Linux 和 Mac 中，列表以冒号分隔。在 Windows 中，列表以分号分隔。

   如果 KUBECTL_PLUGINS_PATH 不存在，则加载程序将开始搜索额外位置。

2. ${XDG_DATA_DIRS}/kubectl/plugins

   该插件加载程序根据 XDG 系统目录结构 规格搜索一个或多个指定目录。

   具体来说，加载程序查找 XDG_DATA_DIRS 环境变量指定的目录。该插件加载程序在 XDG_DATA_DIRS 环境变量指定的目录中搜索 kubectl/plugins 目录。如果没有指定 XDG_DATA_DIRS，则默认为 /usr/local/share:/usr/share。

3. ~/.kube/plugins

   用户的 kubeconfig 目录下的 plugins 目录。在大多数情况下，这是 ~/.kube/plugins ：

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

7.4. 编写插件

您可以使用任何编程语言或脚本编写插件，允许您编写 CLI 命令。插件不一定需要有二进制组件。它可完全依赖操作系统实用程序，如 echo、sed 或 grep。或者，它可能依赖 oc 二进制文件。

oc 插件唯一的强要求是 plugin.yaml 描述符文件。此文件至少负责声明注册插件所需的最小属性，并且必须位于 Search Order 部分中指定的其中一个位置。

7.4.1. plugin.yaml Descriptor

描述符文件支持以下属性：

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

上述描述符声明了 great-plugin 插件，它具有一个名为 -f | --flag-name 的标签。它可以通过以下命令调用：

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

调用插件时，它将调用 示例二进制文件或 脚本，该二进制文件位于与描述符文件相同的目录中，传递多个参数和环境变量。Accessing Runtime Attributes 部分论述了 示例 命令如何访问标志值和其他运行时上下文。

7.4.2. 建议的目录结构

建议每个插件在文件系统中都有自己的子目录，最好与插件命令的名称相同。目录必须包含 plugin.yaml 描述符以及它可能需要的任何二进制、脚本、资产或其他依赖项。

例如，great-plugin 插件的目录结构可能类似如下：

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

7.4.3. 访问运行时属性

在大多数用例中，您写入的二进制文件或脚本文件必须有权访问插件框架提供的一些上下文信息。例如，如果您在描述符文件中声明了标记，则插件必须在运行时能够访问用户提供的标志值。

全局标志也是如此。插件框架负责执行此操作，因此插件作者不需要担心解析参数。这也确保插件和常规 oc 命令之间的最佳一致性水平。

插件可以通过环境变量访问运行时上下文属性。例如，若要访问通过标志提供的值，例如，使用针对二进制或脚本的适当函数调用查找正确的环境变量值。

支持的环境变量有：

- KUBECTL_PLUGINS_CALLER ：当前命令调用中使用的 oc 二进制文件的完整路径。作为插件作者，您不必实施逻辑来验证和访问 Kubernetes API。您可以使用此环境变量提供的值来调用 oc 并获取所需信息，如 oc get --raw=/apis。
- KUBECTL_PLUGINS_CURRENT_NAMESPACE ：当前的命名空间是此调用的上下文。这是要在命名空间操作中考虑的实际命名空间，这意味着它已经按照 kubeconfig 提供的优先级、--namespace global 标志、环境变量等之间的优先级处理。
- KUBECTL_PLUGINS_DESCRIPTOR_* ：用于 plugin.yaml 描述符中声明的每个属性的一个环境变量。例如：KUBECTL_PLUGINS_DESCRIPTOR_NAME,KUBECTL_PLUGINS_DESCRIPTOR_COMMAND.
- KUBECTL_PLUGINS_GLOBAL_FLAG_* ：每个 oc 支持的全局标志的环境变量。例如：KUBECTL_PLUGINS_GLOBAL_FLAG_NAMESPACE,KUBECTL_PLUGINS_GLOBAL_FLAG_LOGLEVEL。
- KUBECTL_PLUGINS_LOCAL_FLAG_* ：用于 plugin.yaml 描述符中声明的每个本地标记的一个环境变量。例如，以上 great-plugin 示例中的 KUBECTL_PLUGINS_LOCAL_FLAG_HEAT。