OpenShift Container Platform 3.11

CLI 참조

OpenShift Container Platform 3.11 CLI 참조

Last Updated: 2023-02-20

OpenShift Container Platform 3.11 CLI 참조

OpenShift Container Platform 3.11 CLI 참조

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.

법적 공지

초록

OpenShift Container Platform CLI(명령줄 인터페이스)를 사용하면 터미널에서 애플리케이션을 생성하고 OpenShift 프로젝트를 관리할 수 있습니다. 다음 주제에서는 CLI 사용 방법을 보여줍니다.

차례

1장. 개요

OpenShift Container Platform CLI(명령줄 인터페이스)를 사용하면 터미널에서 애플리케이션을 생성하고 OpenShift Container Platform 프로젝트를 관리할 수 있습니다. CLI는 다음과 같은 경우에 적합합니다.

- 프로젝트 소스 코드로 직접 작업.
- OpenShift Container Platform 작업 스크립팅.
- 대역폭 리소스로 제한되며 웹 콘솔 을 사용할 수 없습니다.

CLI는 **oc** 명령을 사용하여 사용할 수 있습니다.

```
$ oc <command>
```

설치 및 설정 지침은 CLI 시작하기를 참조하십시오.

2장. CLI 시작하기

2.1. 개요

OpenShift Container Platform CLI는 애플리케이션 관리와 시스템의 각 구성 요소와 하위 수준 툴을 관리하는 명령을 표시합니다. 이 주제에서는 설치 및 로그인 등 CLI를 시작하여 첫 번째 프로젝트를 생성하는 방법을 안내합니다.

2.2. 사전 요구 사항

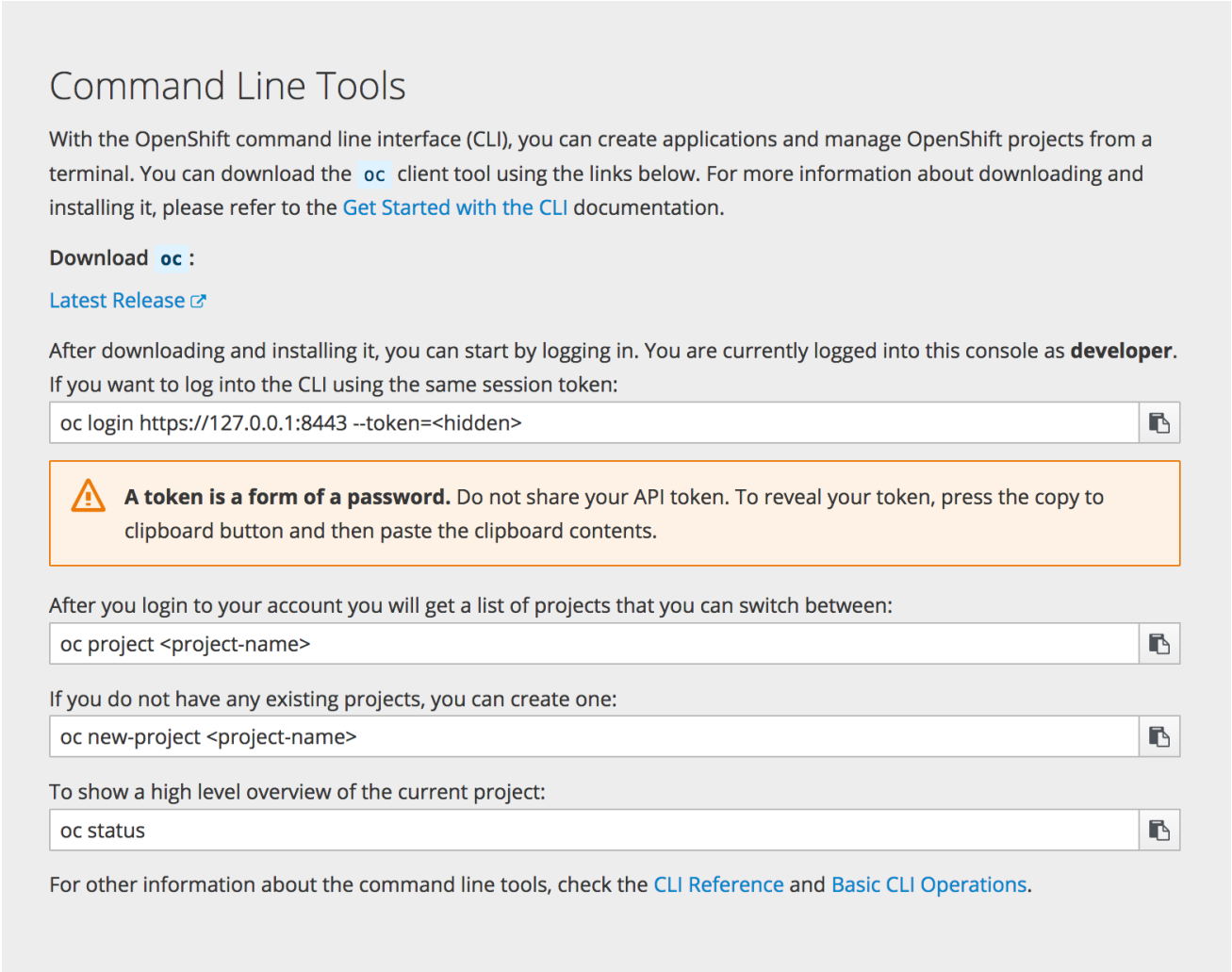
특정 작업을 수행하려면 Git을 클라이언트에 로컬로 설치해야 합니다. 예를 들어 원격 Git 리포지토리를 사용하여 애플리케이션을 생성하는 명령은 다음과 같습니다.

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

진행하기 전에 워크스테이션에 Git을 설치합니다. 워크스테이션 운영 체제별 지침은 공식 Git 설명서 를 참조하십시오.

2.3. CLI 설치

CLI를 다운로드하는 가장 쉬운 방법은 클러스터 관리자가 다운로드 링크를 활성화한 경우 웹 콘솔의 **About** 페이지에 액세스하는 것입니다.



CLI의 설치 옵션은 운영 체제에 따라 다릅니다.

CLI를 사용하여 로그인하려면 **Help** 메뉴의 **명령줄** 메뉴에서 액세스할 수 있는 웹 콘솔의 **명령줄** 페이지에서 토큰을 수집합니다. 토큰이 숨겨져 있으므로 **명령줄 도구** 페이지의 **oc login** line 끝에 있는 **클립보드로 복사** 버튼을 클릭한 다음, 복사된 내용을 복사하여 토큰을 표시해야합니다.

2.3.1. Windows의 경우

Windows용 CLI는 ***zip*** 아카이브로 제공됩니다. Red Hat 고객 포털에서 다운로드할 수 있습니다. Red Hat 계정으로 로그인한 후 다운로드 페이지에 액세스하려면 활성 OpenShift Enterprise 서브스크립션이 있어야 합니다.

Red Hat 고객 포털에서 CLI 다운로드

또는 클러스터 관리자가 이를 활성화한 경우 웹 콘솔의 **정보** 페이지에서 CLI를 다운로드하고 압축을 풀 수 있습니다.

튜토리얼 비디오:

다음 비디오에서는 이 프로세스를 안내합니다. 감시하려면 여기를 클릭하십시오.

그런 다음 ZIP 프로그램으로 아카이브의 압축을 풀고 **oc** 바이너리를 PATH의 디렉터리로 이동합니다. PATH를 확인하려면 명령 프롬프트를 열고 다음을 실행합니다.

```
C:\> path
```

2.3.2. Mac OS X의 경우

Mac OS X용 CLI는 ***tar.gz*** 아카이브로 제공됩니다. Red Hat 고객 포털에서 다운로드할 수 있습니다. Red Hat 계정으로 로그인한 후 다운로드 페이지에 액세스하려면 활성 OpenShift Enterprise 서브스크립션이 있어야 합니다.

Red Hat 고객 포털에서 CLI 다운로드

또는 클러스터 관리자가 이를 활성화한 경우 웹 콘솔의 **정보** 페이지에서 CLI를 다운로드하고 압축을 풀 수 있습니다.

튜토리얼 비디오:

다음 비디오에서는 이 프로세스를 안내합니다. 감시하려면 여기를 클릭하십시오.

그런 다음 아카이브의 압축을 풀고 **oc** 바이너리를 PATH의 디렉터리로 이동합니다. PATH를 확인하려면 터미널 창을 열고 다음을 실행합니다.

```
$ echo $PATH
```

2.3.3. Linux의 경우

RHEL(Red Hat Enterprise Linux) 7의 경우 Red Hat 계정에 활성 OpenShift Enterprise 서브스크립션이 있는 경우 RHSM(Red Hat Subscription Management)을 사용하여 CLI를 RPM으로 설치할 수 있습니다.

1. Red Hat Subscription Manager에 등록합니다.

```
# subscription-manager register
```

2. 최신 서브스크립션 데이터를 가져옵니다.

```
# subscription-manager refresh
```

3. 등록된 시스템에 서브스크립션을 연결합니다.

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

1 활성 OpenShift Enterprise 서브스크립션의 풀 ID

4. OpenShift Container Platform 3.11에 필요한 리포지토리를 활성화합니다.

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

5. **atomic-openshift-clients** 패키지를 설치합니다.

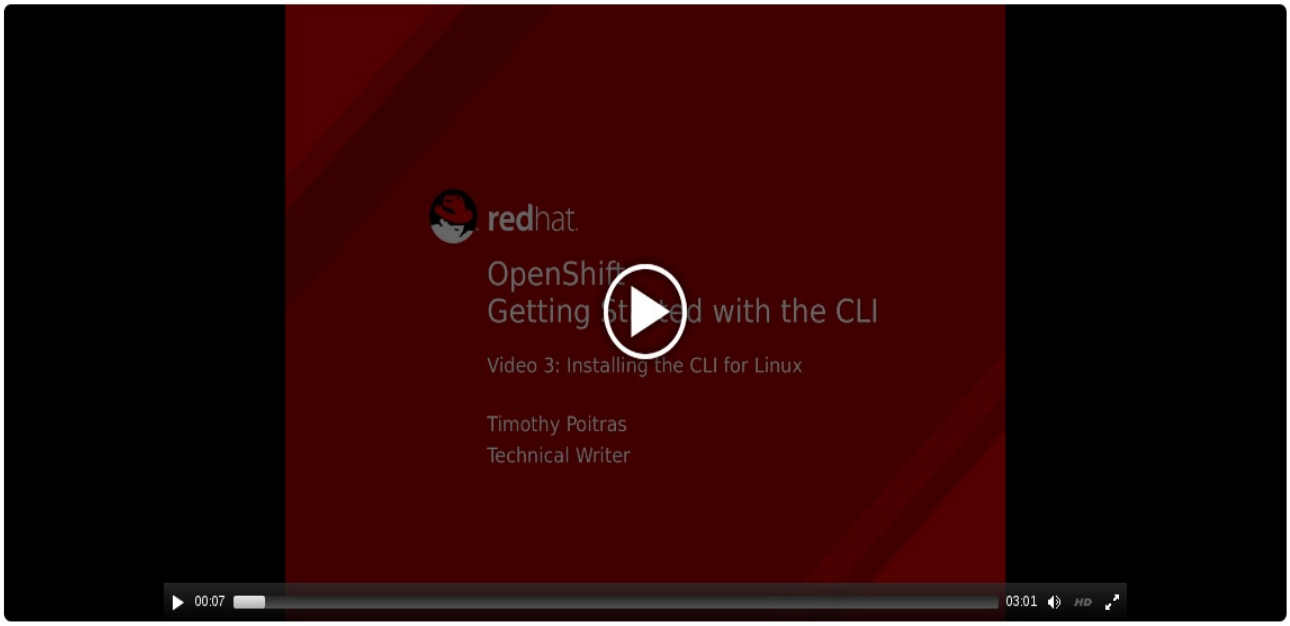
```
# yum install atomic-openshift-clients
```

RHEL, Fedora 및 기타 Linux 배포판의 경우 Red Hat 고객 포털에서 ***tar.gz*** 아카이브로 직접 CLI를 다운로드할 수도 있습니다. Red Hat 계정으로 로그인한 후 다운로드 페이지에 액세스하려면 활성 OpenShift Enterprise 서브스크립션이 있어야 합니다.

Red Hat 고객 포털에서 CLI 다운로드

튜토리얼 비디오:

다음 비디오에서는 이 프로세스를 안내합니다. 감시하려면 여기를 클릭하십시오.



또는 클러스터 관리자가 이를 활성화한 경우 웹 콘솔의 **정보** 페이지에서 CLI를 다운로드하고 압축을 풀 수 있습니다.

그런 다음 아카이브의 압축을 풀고 **oc** 바이너리를 PATH의 디렉터리로 이동합니다. 경로를 확인하려면 다음을 실행하십시오.

```
$ echo $PATH
```

아카이브의 압축을 풀려면 다음을 수행합니다.

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

참고

RHEL 또는 Fedora를 사용하지 않는 경우 **libc**가 라이브러리 경로의 디렉터리에 설치되어 있는지 확인합니다. **libc**를 사용할 수 없는 경우 CLI 명령을 실행할 때 다음과 같은 오류가 표시될 수 있습니다.

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

2.4. 기본 설정 및 로그인

oc login 명령은 처음에 CLI를 설정하는 가장 좋은 방법이며, 대부분의 사용자의 진입점으로 사용됩니다. 대화형 흐름을 사용하면 제공된 인증 정보를 사용하여 OpenShift Container Platform 서버에 대한 세션을 설정할 수 있습니다. 정보는 후속 명령에 사용되는 CLI 구성 파일에 자동으로 저장됩니다.

다음 예제에서는 **oc login** 명령을 사용하여 대화형 설정 및 로그인을 보여줍니다.

예 2.1. 초기 CLI 설정

```
$ oc login
```

출력 예

```
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. 이 명령은 OpenShift Container Platform 서버 URL을 입력하라는 메시지를 표시합니다.
2. 이 명령은 사용자 이름 및 암호와 같은 로그인 자격 증명을 입력하라는 메시지를 표시합니다.
3. 세션이 서버에 설정되며 세션 토큰이 수신됩니다.
4. 프로젝트가 없는 경우 프로젝트를 생성하는 방법에 대한 정보가 제공됩니다.

CLI 구성을 완료한 후 후속 명령에서는 서버, 세션 토큰 및 프로젝트 정보에 구성 파일을 사용합니다.

oc logout 명령을 사용하여 CLI에서 로그아웃할 수 있습니다.

```
$ oc logout
```

출력 예

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

프로젝트에 대한 액세스 권한을 생성하거나 부여한 후 로그인하면 다른 프로젝트로 전환 할 때까지 액세스 권한이 있는 프로젝트가 현재 기본값으로 자동 설정됩니다.

```
$ oc login
```

출력 예

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

```
Login successful.

Using project "aliceproject".
```

oc login 명령에 추가 옵션을 사용할 수도 있습니다.

참고

관리자 자격 증명에 액세스할 수 있지만 더 이상 기본 시스템 사용자 **system**:admin 으로 로그인하지 않은 경우 CLI 구성 파일에 인증 정보가 계속 있는 한 언제든지 이 사용자로 다시 로그인할 수 있습니다. 다음 명령은 **기본** 프로젝트에 로그인하여 전환합니다.

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

2.5. CLI 설정 파일

CLI 구성 파일은 **oc** 옵션을 영구적으로 저장하고 일련의 인증 메커니즘과 닉네임과 관련된 OpenShift Container Platform 서버 연결 정보를 포함합니다.

이전 섹션에 설명된 대로 **oc login** 명령은 CLI 구성 파일을 자동으로 생성하고 관리합니다. 명령으로 수집한 모든 정보는 ***~/.kube/config*** 에 있는 구성 파일에 저장됩니다. 현재 CLI 구성은 다음 명령을 사용하여 볼 수 있습니다.

예 2.2. CLI 구성 보기

```
$ oc config view
```

출력 예

```
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 구성 파일은 쉽게 전환할 수 있도록 다양한 OpenShift Container Platform 서버, 네임스페이스 및 사용자를 사용하여 여러 CLI 프로필을 설정하는 데 사용할 수 있습니다. CLI는 여러 구성 파일을 지원할 수 있습니다. 런타임 시 로드되고 명령줄에서 지정된 덮어쓰기 옵션과 함께 병합됩니다.

2.6. 프로젝트

OpenShift Container Platform의 프로젝트에 는 논리 애플리케이션을 구성하는 여러 오브젝트 가 포함되어 있습니다.

대부분의 **oc** 명령은 프로젝트 의 컨텍스트에서 실행됩니다. **oc login** 은 초기 설정 중에 후속 명령에 사용할 기본 프로젝트를 선택합니다. 다음 명령을 사용하여 현재 사용 중인 프로젝트를 표시합니다.

```
$ oc project
```

여러 프로젝트에 액세스할 수 있는 경우 프로젝트 이름을 지정하여 다음 구문을 사용하여 특정 프로젝트로 전환합니다.

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

예:

Project project02로 전환

```
$ oc project project02
```

출력 예

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

Project 03으로전환

```
$ oc project project03
```

출력 예

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

현재 프로젝트 나열

```
$ oc project
```

출력 예

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

oc status 명령은 다음 예와 같이 현재 사용 중인 프로젝트에 대한 높은 수준 개요와 해당 구성 요소 및 해당 관계를 표시합니다.

```
$ oc status
```

출력 예

```
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. 다음 단계는 무엇입니까?

로그인하면 새 애플리케이션을 생성하고 몇 가지 일반적인 CLI 작업을 살펴볼 수 있습니다.

3장. CLI 프로필 관리

3.1. 개요

CLI 구성 파일을 사용하면 OpenShift CLI 와 함께 사용할 다양한 *프로필 또는 컨텍스트* 를 구성할 수 있습니다. 컨텍스트는 *nickname* 과 관련된 사용자 인증 및 OpenShift Container Platform 서버 정보로 구성됩니다.

3.2. CLI 프로필 간 전환

컨텍스트를 사용하면 CLI 작업을 실행할 때 여러 OpenShift Container Platform 서버 또는 *클러스터에서 여러* 사용자를 쉽게 전환할 수 있습니다. nickname은 컨텍스트, 사용자 인증 정보 및 클러스터 세부 정보에 대한 간단한 참조를 제공하여 CLI 구성을 보다 쉽게 관리할 수 있습니다.

CLI로 처음으로 로그인 한 후 OpenShift Container Platform은 아직 존재하지 않는 경우 ***~/.kube/config*** 파일을 생성합니다. **oc login** 작업 중에 자동으로 또는 명시적으로 설정하여 CLI에 인증 및 연결 세부 정보가 제공되면 업데이트된 정보가 구성 파일에 저장됩니다.

예 3.1. CLI 설정 파일

```
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) cluster **섹션은** 마스터 서버의 주소를 포함하여 OpenShift Container Platform 클러스터에 대한 연결 세부 정보를 정의합니다. 이 예에서는 하나의 클러스터 이름이 **openshift1.example.com:8443** 이고 다른 클러스터는 이름이 **openshift2.example.com:8443** 입니다.

2 이 **context** 섹션은 **alice-project** 프로젝트, **openshift1.example.com:8443** 클러스터, **alice** 사용자, 다른 nick 이름의 **joe-project/openshift1.example.com:8443/alice**, 즉 하나의 컨텍스트 이

3 **current-context** 매개변수는 **joe-project/openshift1.example.com:8443/alice** 컨텍스트가 현재 사용 중이며, **alice** 사용자가 **openshift1.example.com:8443** 클러스터의 **joe-project** 프로젝트에서 작업할 수 있음을 보여줍니다.

4 **users** 섹션은 사용자 자격 증명을 정의합니다. 이 예에서 사용자 nickname **alice/openshift1.example.com:8443** 은 액세스 토큰 을 사용합니다.

CLI는 여러 구성 파일을 지원할 수 있습니다. 런타임 시 로드되고 명령줄에서 지정된 덮어쓰기 옵션과 함께 병합 됩니다.

로그인한 후 **oc status** 명령 또는 **oc project** 명령을 사용하여 현재 작업 환경을 확인할 수 있습니다.

예 3.2. 현재 작업 환경 확인

```
$ oc status
```

출력 예

```
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.
```

현재 프로젝트 나열

```
$ oc project
```

출력 예

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

사용자 자격 증명 및 클러스터 세부 정보 조합을 사용하여 로그인하려면 **oc login** 명령을 다시 실행하고 대화형 프로세스 중에 관련 정보를 제공합니다. 컨텍스트는 아직 존재하지 않는 경우 제공된 정보를 기반으로 구성됩니다.

이미 로그인한 후 현재 사용자가 이미 액세스할 수 있는 다른 프로젝트로 전환하려면 **oc project** 명령을 사용하여 프로젝트 이름을 제공합니다.

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

출력 예

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

언제든지 **oc config view** 명령을 사용하여 출력에 표시된 대로 현재 전체 CLI 구성을 볼 수 있습니다.

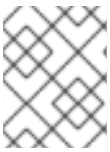
고급 사용을 위해 추가 CLI 구성 명령을 사용할 수도 있습니다.

참고

관리자 자격 증명에 액세스할 수 있지만 더 이상 기본 시스템 사용자 **system**:admin 으로 로그인하지 않은 경우 CLI 구성 파일에 인증 정보가 계속 있는 한 언제든지 이 사용자로 다시 로그인할 수 있습니다. 다음 명령은 **기본** 프로젝트에 로그인하여 전환합니다.

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

3.3. CLI 프로필 수동 구성



참고

이 섹션에서는 CLI 구성의 고급 사용에 대해 설명합니다. 대부분의 경우 **oc login** 및 **oc project** 명령을 사용하여 컨텍스트와 프로젝트 간에 로그인하고 전환할 수 있습니다.

CLI 구성 파일을 수동으로 구성하려면 파일을 수정하는 대신 **oc config** 명령을 사용할 수 있습니다. **oc config** 명령에는 다음과 같은 목적으로 유용한 여러 하위 명령이 포함되어 있습니다.

표 3.1. CLI 구성 하위 명령

하위 명령	사용법
set-cluster	CLI 구성 파일에 클러스터 항목을 설정합니다. 참조된 클러스터 닉네임이 이미 존재하는 경우 지정된 정보가 에 병합됩니다. `$ oc config set-cluster <cluster_nickname> [--server=<master_ip_or_fqdn>] [--certificate-authority=<path/to/certificate/authority>] [--api-version=<apiversion>] [--insecure-skip-tls-verify=true]`
set-context	CLI 구성 파일에 컨텍스트 항목을 설정합니다. 참조된 컨텍스트 닉네임이 이미 있는 경우 지정된 정보가 에 병합됩니다. `$ oc config set-context <context_nickname> [--cluster=<cluster_nickname>] [--user=<user_nickname>] [--namespace=<namespace>]`

하위 명령	사용법
use-context	지정된 컨텍스트 nickname을 사용하여 현재 컨텍스트를 설정합니다. `$ oc config use-context <context_nickname>`
set	CLI 구성 파일에서 개별 값을 설정합니다. `$ oc config set <property_name> <property_value>` & **lt;property_name** >은 각 토큰이 속성 이름 또는 맵 키를 나타내는 점으로 구분된 이름입니다. & **lt;property_value** >는 설정하는 새 값입니다.
unset	CLI 구성 파일에서 개별 값을 설정 해제합니다. `$ oc config unset <property_name>` & **lt;property_name** >은 각 토큰이 속성 이름 또는 맵 키를 나타내는 점으로 구분된 이름입니다.

view	현재 사용 중인 병합된 CLI 구성을 표시합니다.
	`$ oc config view`
	지정된 CLI 구성 파일의 결과를 표시합니다.
	`$ oc config view --config=<specific_filename>`

사용 예

다음 구성 워크플로를 고려하십시오. 먼저 액세스 토큰 을 사용하는 사용자로 로그인합니다. 이 토큰은 **alice** 사용자가 사용합니다.

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

자동으로 생성된 클러스터 항목을 확인합니다.

```
$ oc config view
```

출력 예

```
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
```

사용자가 원하는 네임스페이스에 로그인하도록 현재 컨텍스트를 업데이트합니다.

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

변경 사항이 적용되었는지 확인하려면 현재 컨텍스트를 검사합니다.

```
$ oc whoami -c
```

CLI 옵션을 재정의하거나 컨텍스트가 전환될 때까지 달리 지정하지 않는 한 모든 후속 CLI 작업은 새 컨텍스트를 사용합니다.

3.4. 로드 및 혼합 규칙

CLI 작업을 실행할 때 CLI 구성에 대한 로드 및 병합 순서는 다음 규칙을 따릅니다.

1. CLI 구성 파일은 다음 계층 및 병합 규칙을 사용하여 워크스테이션에서 검색됩니다.
   - **--config** 옵션이 설정되면 해당 파일만 로드됩니다. 플래그는 한 번만 설정할 수 있으며 병합이 수행되지 않습니다.
   - **$KUBECONFIG** 환경 변수가 설정되면 사용됩니다. 변수가 경로 목록일 수 있으며, 경로가 함께 병합되는 경우도 있습니다. 값을 수정하면 스탠자를 정의하는 파일에서 수정됩니다. 값이 생성되면 존재하는 첫 번째 파일에 생성됩니다. 체인에 파일이 없으면 목록에 마지막 파일을 만듭니다.
   - 그렇지 않으면 ***~/.kube/config*** 파일이 사용되며 병합이 수행되지 않습니다.
2. 사용할 컨텍스트는 다음 체인의 첫 번째 히트에 따라 결정됩니다.
   - **--context** 옵션의 값입니다.
   - CLI 구성 파일의 **current-context** 값입니다.
   - 이 단계에서는 빈 값이 허용됩니다.
3. 사용할 사용자 및 클러스터가 결정됩니다. 이 시점에는 컨텍스트가 있거나 없을 수 있습니다. 다음 체인의 첫 번째 히트를 기반으로 빌드되며, 이는 사용자와 클러스터에 대해 한 번 실행됩니다.
   - 사용자 이름에 대한 **--user** 옵션 값 및 클러스터 이름의 **--cluster** 옵션
   - context **옵션이 있는** 경우 컨텍스트의 값을 사용합니다.

- 이 단계에서는 빈 값이 허용됩니다.

4. 사용할 실제 클러스터 정보가 결정됩니다. 이 시점에서 클러스터 정보가 있거나 없을 수 있습니다. 클러스터 정보의 각 부분은 다음 체인의 첫 번째 히트에 따라 빌드됩니다.
   - 다음 명령줄 옵션의 값:
     - **--server**,
     - **--api-version**
     - **--certificate-authority**
     - **--insecure-skip-tls-verify**
   - 클러스터 정보와 속성 값이 있는 경우 이를 사용합니다.
   - 서버 위치가 없는 경우 오류가 발생합니다.

5. 사용할 실제 사용자 정보가 결정됩니다. 사용자는 사용자당 하나의 인증 기술만 가질 수 있다는 점을 제외하고 클러스터와 동일한 규칙을 사용하여 빌드됩니다. 충돌하는 기술은 작업이 실패합니다. 명령줄 옵션은 구성 파일 값보다 우선합니다. 유효한 명령줄 옵션은 다음과 같습니다.
   - **--auth-path**
   - **--client-certificate**
   - **--client-key**
   - **--token**

6. 여전히 누락된 모든 정보에 대해 기본값이 사용되며 추가 정보를 입력하라는 메시지가 표시됩니다.

4장. 개발자 CLI 작업

4.1. 개요

이 주제에서는 개발자 CLI 작업 및 해당 구문에 대한 정보를 제공합니다. 이러한 작업을 수행하려면 CLI를 사용하여 설정하고 로그인해야 합니다.

개발자 CLI는 **oc** 명령을 사용하며 프로젝트 수준 작업에 사용됩니다. 이는 고급 관리자 작업에 **oc adm** 명령을 사용하는 관리자 CLI와 다릅니다.

4.2. COMMON OPERATIONS

개발자 CLI를 사용하면 OpenShift Container Platform에서 관리하는 다양한 오브젝트와 상호 작용할 수 있습니다. 많은 일반적인 **oc** 작업은 다음 구문을 사용하여 호출됩니다.

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

이 명령은 다음을 지정합니다.

- **get** 또는 **describe** 와 같이 수행할 < **action** >입니다.
- 작업을 수행할 &lt **;object_type** >(예: **service** 또는 축약된 **svc**).
- 지정된 & **lt;object_ type>의 <object_name >**.

예를 들어 **oc get** 작업은 현재 정의된 전체 서비스 목록을 반환합니다.

```
$ oc get svc
```

출력 예

```
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
```

그런 다음 **oc describe** 작업을 사용하여 특정 오브젝트에 대한 세부 정보를 반환할 수 있습니다.

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

출력 예

```
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. 오브젝트 유형

다음은 CLI에서 지원하는 가장 일반적인 오브젝트 유형 목록입니다. 일부에는 축약된 구문이 있습니다.

오브젝트 유형	축약된 버전
Build	
BuildConfig	**bc**
DeploymentConfig	**dc**
배포	**deploy**
이벤트	**ev**
ImageStream	**is**
ImageStreamTag	**istag**
ImageStreamImage	**isimage**
Job	
CronJob (기술 프리뷰	**cj**
LimitRange	**limits**
노드	
Pod	**po**
resourceQuota	**quota**
ReplicationController	**rc**
ReplicaSet	**rs**
보안	
서비스	**svc**
ServiceAccount	**sa**

오브젝트 유형	축약된 버전
StatefulSets	**sts**
PersistentVolume	**pv**
PersistentVolumeClaim	**pvc**

서버에서 지원하는 전체 리소스 목록을 알고 싶다면 **oc api-resources** 를 사용하십시오.

4.4. 기본 CLI 작업

다음 표에서는 기본 **oc** 작업 및 일반 구문을 설명합니다.

4.4.1. 유형

몇 가지 핵심 OpenShift Container Platform 개념에 대한 소개를 표시합니다.

```
$ oc types
```

4.4.2. login

OpenShift Container Platform 서버에 로그인합니다.

```
$ oc login
```

4.4.3. logout

현재 세션을 종료합니다.

```
$ oc logout
```

4.4.4. new-project

새 프로젝트를 생성합니다.

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

4.4.5. new-app

현재 디렉터리의 소스 코드를 기반으로 새 애플리케이션을 생성합니다.

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

원격 리포지토리의 소스 코드를 기반으로 새 애플리케이션을 생성합니다.

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

프라이빗 원격 리포지토리에서 소스 코드를 기반으로 새 애플리케이션을 생성합니다.

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

4.4.6. status

현재 프로젝트의 개요를 표시합니다.

```
$ oc status
```

4.4.7. project

다른 프로젝트로 전환합니다. 옵션 없이 를 실행하여 현재 프로젝트를 표시합니다. 모든 프로젝트를 보려면 **oc projects** 를 실행할 수 있습니다.

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

4.5. 애플리케이션 수정 작업

4.5.1. get

지정된 오브젝트 유형의 오브젝트 목록을 반환합니다. 선택 사항인 **<object_name** >이 요청에 포함된 경우 결과 목록은 해당 값에 의해 필터링됩니다.

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

예를 들어 다음 명령은 프로젝트에 사용 가능한 이미지를 나열합니다.

```
$ oc get images
```

출력 예

```
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
```

-o 또는 **--output** 옵션을 사용하여 출력 형식을 수정할 수 있습니다.

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

출력 형식은 JSON 또는 YAML 또는 사용자 지정 열,golang 템플릿, jsonpath 와 같은 확장 가능한 형식일 수 있습니다.

예를 들어 다음 명령은 특정 프로젝트에서 실행 중인 Pod의 이름을 나열합니다.

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

출력 예

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

4.5.2. describe

쿼리에서 반환된 특정 오브젝트에 대한 정보를 반환합니다. 특정 &lt **;object_name&gt**;을 제공해야 합니다. 사용 가능한 실제 정보는 오브젝트 유형에 설명된 대로 다릅니다.

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

4.5.3. edit

원하는 오브젝트 유형을 편집합니다.

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

지정된 텍스트 편집기를 사용하여 원하는 오브젝트 유형을 편집합니다.

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

지정된 형식으로 원하는 오브젝트를 편집합니다(예: JSON).

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

4.5.4. 볼륨

볼륨 수정:

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

4.5.5. label

오브젝트에서 레이블을 업데이트합니다.

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

4.5.6. expose

서비스를 조회하여 경로로 노출합니다. 또한 배포 구성, 복제 컨트롤러, 서비스 또는 Pod를 지정된 포트의 새 서비스로 노출하는 기능도 있습니다. 라벨이 지정되지 않은 경우 새 오브젝트는 표시하는 오브젝트에서 레이블을 다시 사용합니다.

서비스를 노출하는 경우 기본 생성기는 **--generator=route/v1** 입니다. 다른 모든 경우의 기본값은 **--generator=service/v2** 이며, 포트 이름이 지정되지 않은 상태로 유지됩니다. 일반적으로 **oc expose** 명령을 사용하여 생성기를 설정할 필요가 없습니다. 세 번째 생성기 **--generator=service/v1** 은 포트 이름

default와 함께 사용할 수 있습니다.

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

4.5.7. delete

지정된 오브젝트를 삭제합니다. 오브젝트 구성은 STDIN을 통해 전달할 수도 있습니다. **oc delete all -l <label>** 작업은 Pod가 다시 생성되지 않도록 복제 컨트롤러를 포함하여 지정된 < **label** >과 일치하는 모든 오브젝트를 삭제합니다.

```
$ 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

지정된 개체의 특정 속성을 수정합니다.

4.5.8.1. 설정 env

배포 구성 또는 빌드 구성에 환경 변수를 설정합니다.

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

4.5.8.2. build-secret 설정

빌드 구성에 보안 이름을 설정합니다. 보안은 이미지 가져오기 또는 내보내기 보안 또는 소스 리포지토리 보안일 수 있습니다.

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

4.6. 빌드 및 배포 작업

OpenShift Container Platform의 기본 기능 중 하나는 소스에서 애플리케이션을 컨테이너로 빌드하는 기능입니다.

OpenShift Container Platform에서는 **get**,**create**, **describe** 와 같은 표준 **oc** 리소스 작업을 사용하여 배포 구성을 검사하고 조작하는 CLI 액세스 권한을 제공합니다.

4.6.1. start-build

지정된 빌드 구성 파일을 사용하여 빌드 프로세스를 수동으로 시작합니다.

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

이전 빌드의 이름을 시작점으로 지정하여 빌드 프로세스를 수동으로 시작합니다.

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

구성 파일 또는 이전 빌드의 이름을 지정하여 빌드 프로세스를 수동으로 시작하고 해당 빌드 로그를 검색합니다.

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

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

빌드가 실패하면 빌드가 완료되고 0이 아닌 반환 코드로 종료될 때까지 기다립니다.

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

빌드 구성을 변경하지 않고 현재 빌드의 환경 변수를 설정하거나 재정의합니다. 또는 **-e** 를 사용합니다.

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

빌드 중 기본 빌드 로그 수준 출력을 설정하거나 덮어씁니다.

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

빌드에서 사용해야 하는 소스 코드 커밋 식별자를 지정합니다. Git 리포지토리에 따라 빌드가 필요합니다.

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

< **build_name>을 사용하여 빌드를 다시 실행합니다**.

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

& **lt;dir_name** >을 보관하고 바이너리 입력으로 빌드합니다.

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

기존 아카이브를 바이너리 입력으로 사용합니다. **--from-file** 과 달리 빌드 프로세스 전에 빌더에서 아카이브를 추출합니다.

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

& **lt;file_name&** gt;을 빌드의 바이너리 입력으로 사용합니다. 이 파일은 빌드 소스에서 유일한 파일이어야 합니다. 예: ***pom.xml*** 또는 ***Dockerfile***.

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

파일 시스템에서 HTTP 또는 HTTPS를 읽는 대신 HTTP 또는 HTTPS를 사용하여 바이너리 입력을 다운로드합니다.

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

아카이브를 다운로드하고 해당 콘텐츠를 빌드 소스로 사용합니다.

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

빌드에 바이너리 입력으로 사용할 로컬 소스 코드 리포지토리의 경로입니다.

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

트리거할 기존 빌드 구성의 Webhook URL을 지정합니다.

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

빌드를 트리거하기 위한 사후 후크의 내용입니다.

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

post-receive의 Git 리포지토리 경로입니다. 기본값은 현재 디렉터리입니다.

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

지정된 빌드 구성 또는 빌드에 대한 Webhook를 나열합니다. **모든,일반** 또는 **github** 를 허용합니다.

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

source-strategy 빌드의 **Spec.Strategy.SourceStrategy.Incremental** 옵션을 재정의합니다.

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

docker-strategy 빌드의 **Spec.Strategy.DockerStrategy.NoCache** 옵션을 재정의합니다.

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

4.6.2. rollback

롤백 을 수행합니다.

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

4.6.3. new-build

현재 Git 리포지토리(공용 원격 사용) 및 컨테이너 이미지의 소스 코드를 기반으로 빌드 구성을 생성합니다.

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

원격 git 리포지터리를 기반으로 빌드 구성을 생성합니다.

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

개인 원격 git 리포지터리를 기반으로 빌드 구성을 생성합니다.

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

4.6.4. cancel-build

진행 중인 빌드를 중지합니다.

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

동시에 여러 빌드를 취소합니다.

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

빌드 구성에서 생성된 모든 빌드를 취소합니다.

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

취소할 빌드를 지정합니다.

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

상태에 대한 예제 값은 **새** 또는 **보류 중입니다**.

4.6.5. import-image

외부 이미지 리포지토리에서 태그 및 이미지 정보를 가져옵니다.

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

4.6.6. scale

복제 컨트롤러 또는 배포 구성에 대해 원하는 복제본 수를 지정된 복제본 수로 설정합니다.

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

4.6.7. tag

이미지 스트림에서 기존 태그 또는 이미지 또는 컨테이너 이미지 "pull spec"을 가져와서 하나 이상의 다른 이미지 스트림에서 태그의 최신 이미지로 설정합니다.

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

4.7. 고급 명령

4.7.1. create

구성 파일을 구문 분석하고 파일 콘텐츠를 기반으로 하나 이상의 OpenShift Container Platform 오브젝트를 생성합니다. **-f** 플래그는 다른 파일 또는 디렉터리 경로로 여러 번 전달할 수 있습니다. 플래그가 여러 번 전달되면 **oc create** 는 각 파일을 반복하여 표시된 모든 파일에 설명된 오브젝트를 생성합니다. 기존 리소스는 모두 무시됩니다.

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

4.7.2. replace

지정된 구성 파일의 콘텐츠를 기반으로 기존 오브젝트를 수정하려고 합니다. **-f** 플래그는 다른 파일 또는 디렉터리 경로로 여러 번 전달할 수 있습니다. 플래그가 여러 번 전달되면 **oc replace** 가 각 파일을 반복하여 표시된 모든 파일에 설명된 오브젝트를 업데이트합니다.

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

4.7.3. process

프로젝트 템플릿을 프로젝트 구성 파일로 변환합니다.

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

4.7.4. run

특정 이미지를 만들고 실행할 수 있으며, 복제될 수 있습니다. 기본적으로 생성된 컨테이너를 관리할 배포 구성을 생성합니다. **--generator** 플래그를 사용하여 다른 리소스를 생성하도록 선택할 수 있습니다.

API 리소스	--Generator 옵션
배포 구성	**DeploymentConfig/v1** (기본값)
Pod	**run-pod/v1**
복제 컨트롤러	**run/v1**
extensions/v1beta1 끝점을 사용한 배포	**deployment/v1beta1**
apps/v1beta1 끝점을 사용하여 배포	**deployment/apps.v1beta1**
Job	**job/v1**
cron 작업	**cronjob/v2alpha1**

대화형 컨테이너 실행을 위해 foreground에서 실행하도록 선택할 수 있습니다.

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

4.7.5. patch

전략적 병합 패치를 사용하여 오브젝트 필드를 하나 이상 업데이트합니다.

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

<changes>는 새 필드와 값이 포함된 JSON 또는 YAML 표현식입니다. 예를 들어 노드 **node1** 의 **spec.unschedulable** 필드를 **true** 값으로 업데이트하려면 json 표현식은 다음과 같습니다.

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

4.7.6. policy

권한 부여 정책을 관리합니다.

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

4.7.7. secrets

보안 구성:

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

4.7.8. autoscale

애플리케이션 자동 스케일러 를 설정합니다. 클러스터에서 지표를 활성화해야 합니다. 필요한 경우 클러스터 관리자 지침은 Enabling Cluster Metrics for cluster administrator instructions를 참조하십시오.

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

4.8. 문제 해결 및 디버깅

4.8.1. debug

실행 중인 애플리케이션을 디버깅하는 명령 쉘을 시작합니다.

```
$ oc debug -h
```

이미지 및 설정 문제를 디버깅할 때 실행 중인 Pod 구성의 정확한 사본을 가져오고 쉘을 사용하여 문제를 해결할 수 있습니다. 실패한 Pod가 시작되지 않고 **rsh** 또는 **exec** 에 액세스할 수 없으므로 **debug** 명령을 실행하면 해당 설정의 이산적 사본이 생성됩니다.

기본 모드는 참조된 Pod, 복제 컨트롤러 또는 배포 구성의 첫 번째 컨테이너 내에서 쉘을 시작하는 것입니다. 시작된 Pod는 라벨이 제거하고 명령이 **/bin/sh** 로 변경되었으며 준비 및 활성 상태 점검이 비활성화된 소스 포드의 사본입니다. 명령을 실행하려는 경우 **--** 및 명령을 추가하여 실행합니다. 명령을 전달해도 TTY가 생성되지 않거나 기본적으로 STDIN을 보내지 않습니다. 일반적인 방식으로 컨테이너 또는 Pod를 변경하는 데 다른 플래그가 지원됩니다.

컨테이너를 실행하는 일반적인 문제는 클러스터에서 root 사용자로 실행하지 못하도록 하는 보안 정책입니다. 이 명령을 사용하여 Pod를 루트가 아닌(- **as-user**)로 실행하거나 루트가 아닌 Pod를 root로 실행(- **as-root**)으로 실행할 수 있습니다.

원격 명령이 완료되거나 쉘을 중단할 때 디버그 Pod가 삭제됩니다.

4.8.1.1. 사용법

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

4.8.1.2. 예제

현재 실행 중인 배포를 디버깅하려면 다음을 수행합니다.

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

루트가 아닌 사용자로 배포 실행을 테스트하려면 다음을 수행합니다.

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

두 번째 컨테이너에서 **env** 명령을 실행하여 실패한 특정 컨테이너를 디버깅하려면 다음을 수행합니다.

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

디버그로 생성되는 Pod를 보려면 다음을 수행합니다.

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

4.8.2. logs

특정 빌드, 배포 또는 Pod의 로그 출력을 검색합니다. 이 명령은 빌드, 빌드 구성, 배포 구성 및 Pod에 대해 작동합니다.

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

4.8.3. exec

이미 실행 중인 컨테이너에서 명령을 실행합니다. 선택적으로 컨테이너 ID를 지정할 수 있습니다. 그렇지 않으면 기본값은 첫 번째 컨테이너로 설정됩니다.

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

4.8.4. rsh

컨테이너에 대한 원격 쉘 세션을 엽니다.

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

4.8.5. rsync

이미 실행 중인 Pod 컨테이너의 디렉터리 또는 디렉터리의 콘텐츠를 복사합니다. 컨테이너를 지정하지 않으면 기본적으로 Pod의 첫 번째 컨테이너로 설정됩니다.

로컬 디렉터리에서 Pod의 디렉터리로 콘텐츠를 복사하려면 다음을 수행합니다.

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

-

Pod의 디렉터리에서 로컬 디렉터리로 콘텐츠를 복사하려면 다음을 수행합니다.

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

4.8.6. port-forward

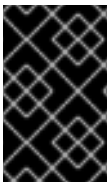
Pod에 하나 이상의 로컬 포트를 전달합니다.

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

4.8.7. proxy

Kubernetes API 서버에 대해 프록시를 실행합니다.

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



중요

보안상의 이유로 **cluster-admin** 사용자가 명령을 실행하는 경우를 제외하고 권한 있는 컨테이너에 액세스하면 **oc exec** 명령이 작동하지 않습니다. 관리자는 노드 호스트에 SSH로 연결한 다음 원하는 컨테이너에서 **docker exec** 명령을 사용할 수 있습니다.

4.9. OC 문제 해결

-v=X 플래그를 사용하여 로그 수준을 늘려 모든 명령에서 더 자세한 출력을 얻을 수 있습니다. 기본적으로 loglevel은 **0** 으로 설정되어 있지만 **0** 에서 **10** 으로 설정할 수 있습니다.

각 로그 수준의 개요

- **1-5** - 일반적으로 작성자가 흐름에 대한 추가 설명을 제공하기로 결정하는 경우 명령에 의해 내부적으로 사용됩니다.
- **6** - 클라이언트와 서버 간의 HTTP 트래픽에 대한 기본 정보를 제공합니다, 이러한 HTTP 작업 및 URL.
- **7** - HTTP 작업, URL, 요청 헤더 및 응답 상태 코드와 같은 보다 상세한 HTTP 정보를 제공합니다.
- **8** - 본문을 포함하여 전체 HTTP 요청 및 응답을 제공합니다.
- **9** - 본문 및 샘플 **curl** 호출을 포함한 전체 HTTP 요청 및 응답을 제공합니다.
- **10** - 명령이 제공하는 모든 출력을 제공합니다.

5장. 관리자 CLI 작업

5.1. 개요

이 주제에서는 관리자 CLI 작업 및 해당 구문에 대한 정보를 제공합니다. 이러한 작업을 수행하려면 CLI를 사용하여 설정하고 로그인해야 합니다.

openshift 명령은 OpenShift Container Platform 클러스터를 구성하는 서비스를 시작하는 데 사용됩니다. 예: **openshift start [master|node]**. 그러나 **openshift cli** 및 **openshift admin** 을 통해 **oc** 및 **oc adm** 명령과 동일한 작업을 모두 수행할 수 있는 all-in-one 명령도 있습니다.

관리자 CLI는 **oc** 명령을 사용하는 개발자 CLI 의 일반 명령 세트와 다르며 프로젝트 수준 작업에 더 많이 사용됩니다.

5.2. COMMON OPERATIONS

관리자 CLI를 사용하면 OpenShift Container Platform에서 관리하는 다양한 오브젝트와 상호 작용할 수 있습니다. 많은 일반적인 **oc adm** 작업은 다음 구문을 사용하여 호출됩니다.

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

이 명령은 다음을 지정합니다.

- 수행할 &lt **;action** >(예: **new-project** 또는 **groups**)입니다.
- 작업을 수행하는 **데 사용할 수 있는 <option** > 및 옵션의 값입니다. 옵션에는 **--output** 이 포함됩니다.

중요

oc adm 명령을 실행하는 경우 기본적으로 */etc/ansible/hosts* 에 의해 Ansible 호스트 인벤토리 파일에 나열된 첫 번째 마스터에서만 실행해야 합니다.

5.3. 기본 CLI 작업

5.3.1. new-project

새 프로젝트를 생성합니다.

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

5.3.2. policy

권한 부여 정책을 관리합니다.

```
$ oc adm policy
```

5.3.3. groups

그룹을 관리합니다.

■

```
$ oc adm groups
```

5.4. CLI 작업 설치

5.4.1. 라우터

라우터를 설치합니다.

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

5.4.2. ipfailover

노드 집합에 대한 IP 페일오버 그룹을 설치합니다.

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

5.4.3. 레지스트리

통합 컨테이너 이미지 레지스트리를 설치합니다.

```
$ oc adm registry
```

5.5. 유지 관리 CLI 작업

5.5.1. build-chain

빌드의 입력 및 종속 항목을 출력합니다.

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

5.5.2. manage-node

노드를 관리합니다. 예를 들어 Pod를 나열하거나 비우거나 준비 상태로 표시합니다.

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

5.5.3. prune

서버에서 이전 버전의 리소스를 제거합니다.

```
$ oc adm prune
```

5.6. 설정 CLI 작업

5.6.1. config

kubelet 구성 파일을 변경합니다.

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

5.6.2. create-kubeconfig

클라이언트 인증서에서 기본 ***.kubeconfig*** 파일을 생성합니다.

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

5.6.3. create-api-client-config

서버에 사용자로 연결하기 위한 구성 파일을 생성합니다.

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

5.7. 고급 CLI 작업

5.7.1. create-bootstrap-project-template

부트스트랩 프로젝트 템플릿을 생성합니다.

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

5.7.2. create-bootstrap-policy-file

기본 부트스트랩 정책을 생성합니다.

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

5.7.3. create-login-template

로그인 템플릿을 생성합니다.

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

5.7.4. create-node-config

노드에 대한 구성 번들을 생성합니다.

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

5.7.5. ca

인증서 및 키를 관리합니다.

```
$ oc adm ca
```

6장. OC와 KUBECTL의 차이점

6.1. OC OVER KUBECTL을 사용하는 이유는 무엇입니까?

Kubernetes의 CLI(명령줄 인터페이스), **kubectl** 는 Kubernetes 클러스터에 대해 명령을 실행하는 데 사용됩니다. OpenShift Container Platform은 Kubernetes 클러스터 상단에 실행되므로 **kubectl** 의 사본은 **oc**, OpenShift Container Platform의 CLI(명령줄 인터페이스)에 포함되어 있습니다.

이 두 클라이언트 사이에 몇 가지 유사점이 있지만 이 가이드의 목표는 다른 고객보다 하나를 사용하기 위한 주요 이유와 시나리오를 명확히하는 것입니다.

6.2. OC 사용

oc 바이너리는 **kubectl** 바이너리와 동일한 기능을 제공하지만 다음과 같은 OpenShift Container Platform 기능을 기본적으로 지원하도록 추가로 확장됩니다.

OpenShift 리소스에 대한 전체 지원

DeploymentConfigs,**BuildConfigs**,**Routes**,**ImageStreams**, **ImageStreamTags** 와 같은 리소스는 OpenShift 배포에 고유하며 표준 Kubernetes에서는 사용할 수 없습니다.

인증

oc 바이너리는 인증을 허용하는 기본 제공 **로그인** 명령을 제공합니다. 자세한 내용은 개발자 인증 및 인증 구성을 참조하십시오.

추가 명령

예를 들어 **new-app** 을 추가로 사용하면 기존 소스 코드 또는 사전 빌드된 이미지를 사용하여 새 애플리케이션을 더 쉽게 시작할 수 있습니다.

6.3. KUBECTL 사용

kubectl 바이너리는 표준 Kubernetes 환경에서 제공되는 새로운 OpenShift Container Platform 사용자에 대한 기존 워크플로우 및 스크립트를 지원하기 위한 수단으로 제공됩니다. **kubectl** 의 기존 사용자는 API에 대한 변경없이 바이너리를 계속 사용할 수 있지만 이전 섹션에 언급된 추가 기능을 얻으려면 **oc** 로 업그레이드해야 합니다.

oc 는 kubectl 위에 내장되어 있기 때문에 **kubectl** 바이너리를 **oc** 로 변환하는 것은 바이너리의 이름을 **kubectl** 에서 **oc** 로 변경하는 것처럼 간단합니다.

설치 및 설정 지침은 CLI 시작하기를 참조하십시오.

7장. CLI 확장

7.1. 개요

이 주제에서는 CLI에 대한 확장 기능을 설치하고 작성하는 방법을 검토합니다. 일반적으로 *플러그인* 또는 *바이너리 확장* 이라고 하는 이 기능을 사용하면 사용 가능한 기본 **oc** 명령 세트를 확장할 수 있으므로 새 작업을 수행할 수 있습니다.

플러그인은 일반적으로 하나 이상의 ***plugin.yaml*** 설명자와 하나 이상의 바이너리, 스크립트 또는 자산 파일 등 파일 세트입니다.

CLI 플러그인은 현재 **oc plugin** 하위 명령에서만 사용할 수 있습니다.

중요

CLI 플러그인은 현재 기술 프리뷰 기능입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원하지 않으며, 기능상 완전하지 않을 수 있어 프로덕션에 사용하지 않는 것이 좋습니다. 고객은 출시 예정된 제품 기능을 프리뷰를 통해 미리 사용해 보면서 테스트하고 개발 과정에서 피드백을 제공할 수 있습니다.

자세한 내용은 Red Hat 기술 프리뷰 기능 지원 범위 를 참조하십시오.

7.2. 사전 요구 사항

다음이 있어야 합니다.

- 작동 중인 **oc** 바이너리가 설치되어 있어야 합니다.

7.3. 플러그인 설치

플러그인의 ***plugin.yaml*** 설명자, 바이너리, 스크립트 및 자산 파일을 **oc** 에서 플러그인을 검색하는 파일 시스템의 위치 중 하나에 복사합니다.

현재 OpenShift Container Platform에서는 플러그인용 패키지 관리자를 제공하지 않습니다. 따라서 플러그인 파일을 올바른 위치에 배치하는 것은 귀하의 책임입니다. 각 플러그인이 자체 디렉터리에 있는 것이 좋습니다.

압축 파일로 배포된 플러그인을 설치하려면 Plug-in Loader 섹션에 지정된 위치 중 하나로 압축을 풉니다.

7.3.1. Plug-in Loader

플러그인 로더는 플러그인 파일을 검색하고 플러그인이 실행하는 데 필요한 최소한의 정보를 제공하는지 확인합니다. 최소 정보를 제공하지 않는 올바른 위치에 배치된 파일은 (예: 불완전한 ***plugin.yaml*** 설명자) 무시됩니다.

7.3.1.1. 검색 순서

플러그인 로더는 다음 검색 순서를 사용합니다.

1. **${KUBECTL_PLUGINS_PATH}**
   지정된 경우 검색은 여기에서 중지됩니다.

KUBECTL_PLUGINS_PATH 환경 변수가 있는 경우 로더는 이를 유일한 위치로 사용하여 플러그인을 찾습니다. **KUBECTL_PLUGINS_PATH** 환경 변수는 디렉터리 목록입니다. Linux 및 Mac에서 목록은 콜론으로 구분됩니다. Windows에서는 list가 intended로 구분됩니다.

KUBECTL_PLUGINS_PATH 가 없으면 로더가 추가 위치 검색을 시작합니다.

2. **${XDG_DATA_DIRS}/kubectl/plugins**
   플러그인 로더는 XDG 시스템 디렉터리 구조 사양에 따라 지정된 하나 이상의 디렉터리를 검색합니다.

   특히 로더는 **XDG_DATA_DIRS** 환경 변수에서 지정한 디렉터리를 찾습니다. 플러그인 로더는 **XDG_DATA_DIRS** 환경 변수에서 지정한 디렉터리 내에서 ***kubectl/plugins*** 디렉터리를 검색합니다. **XDG_DATA_DIRS** 가 지정되지 않은 경우 기본값은 ***/usr/local/share:/usr/share*** 입니다.

3. **~/.kube/plugins**
   사용자의 ***kubeconfig*** 디렉터리 아래에 있는 **plugins** 디렉터리입니다. 대부분의 경우 ***~/.kube/plugins*** 입니다.

```
# 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. 플러그인 작성

CLI 명령을 작성할 수 있는 모든 프로그래밍 언어 또는 스크립트로 플러그인을 작성할 수 있습니다. 플러그인이 반드시 바이너리 구성 요소를 보유할 필요는 없습니다. **echo**,**sed** 또는 **grep** 과 같은 운영 체제 유틸리티에 전적으로 의존할 수 있습니다. 또는 **oc** 바이너리를 사용할 수 있습니다.

oc 플러그인의 유일한 강력한 요구 사항은 ***plugin.yaml*** 설명자 파일입니다. 이 파일은 플러그인 등록에 필요한 최소 특성을 선언해야 하며, Search Order 섹션에 지정된 위치 중 하나에 위치해야 합니다.

7.4.1. plugin.yaml Descriptor

설명자 파일은 다음 속성을 지원합니다.

```
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
```

앞의 설명자는 **-f | --flag-name** 이라는 하나의 플래그가 있는 **great-plugin** 플러그인을 선언합니다. 다음과 같이 호출할 수 있습니다.

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

플러그인이 호출되면 설명자 파일과 동일한 디렉터리에 있는 **예제** 바이너리 또는 스크립트를 호출하여 여러 인수 및 환경 변수를 전달합니다. 액세스 런타임 속성 섹션에서는 **example** 명령이 플래그 값과 기타 런타임 컨텍스트에 액세스하는 방법을 설명합니다.

7.4.2. 권장되는 디렉토리 구조

각 플러그인에는 파일 시스템에 자체 하위 디렉터리가 있으며, 바람직하게는 플러그인 명령과 동일한 이름을 사용하는 것이 좋습니다. 디렉터리에는 ***plugin.yaml*** 설명자와 필요한 바이너리, 스크립트, 자산 또는 기타 종속성이 포함되어야 합니다.

예를 들어 **great-plugin** 플러그인의 디렉터리 구조는 다음과 같습니다.

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

7.4.3. 런타임 속성 액세스

대부분의 사용 사례에서 플러그인을 지원하도록 작성한 바이너리 또는 스크립트 파일은 플러그인 프레임워크에서 제공하는 일부 컨텍스트 정보에 액세스할 수 있어야 합니다. 예를 들어 설명자 파일에서 플래그를 선언한 경우 플러그인은 런타임 시 사용자 제공 플래그 값에 액세스할 수 있어야 합니다.

이는 global flags에서도 마찬가지입니다. 플러그인 프레임워크는 이를 수행해야 하므로 플러그인 작성자는 인수 구문 분석에 대해 걱정할 필요가 없습니다. 또한 플러그인과 일반 **oc** 명령 간의 일관성을 극대화합니다.

플러그인은 환경 변수를 통해 런타임 컨텍스트 속성에 액세스할 수 있습니다. 예를 들어 플래그를 통해 제공되는 값에 액세스하려면 바이너리 또는 스크립트에 대한 적절한 함수 호출을 사용하여 적절한 환경 변수의 값을 찾습니다.

지원되는 환경 변수는 다음과 같습니다.

- **KUBECTL_PLUGINS_CALLER**: 현재 명령 호출에 사용된 **oc** 바이너리의 전체 경로입니다. 플러그인 작성자는 Kubernetes API를 인증하고 액세스하기 위한 논리를 구현할 필요가 없습니다. 대신 이 환경 변수에서 제공하는 값을 사용하여 oc를 호출하고 **oc get --raw=/apis** 를 사용하여 필요한 정보를 가져올 수 있습니다.

- **KUBECTL_PLUGINS_CURRENT_NAMESPACE**:이 호출의 컨텍스트인 현재 네임스페이스입니다. 이는 네임스페이스 작업에서 고려해야 할 실제 네임스페이스입니다. 이는 kubeconfig, **--namespace** 글로벌 플래그, 환경 변수 등을 통해 제공된 우선 순위 측면에서 이미 처리되었습니다.

- **KUBECTL_PLUGINS_DESCRIPTOR_***: ***plugin.yaml*** 설명자에 선언된 모든 속성에 대한 환경 변수 1개 예를 들어 **KUBECTL_PLUGINS_DESCRIPTOR_NAME**,**KUBECTL_PLUGINS_DESCRIPTOR_COMMAND** 입니다.

- **KUBECTL_PLUGINS_GLOBAL_FLAG_***: **oc** 에서 지원하는 모든 글로벌 플래그에 대해 하나의 환경 변수 예를 들어, **KUBECTL_PLUGINS_GLOBAL_FLAG_NAMESPACE**,**KUBECTL_PLUGINS_PLUGINS_LOGLEVEL** 입니다.

- **KUBECTL_PLUGINS_LOCAL_FLAG_***: ***plugin.yaml*** 설명자에 선언된 모든 로컬 플래그의 환경 변수 1개 예를 들어, 위의 **great-plugin** 예제의 **KUBECTL_PLUGINS_LOCAL_FLAG_HEAT** 입니다.