OpenShift Container Platform 4.6

CLI tools

Learning how to use the command-line tools for OpenShift Container Platform
OpenShift Container Platform 4.6 CLI tools

Learning how to use the command-line tools for OpenShift Container Platform
Abstract

This document provides information about installing, configuring, and using the command-line tools for OpenShift Container Platform. It also contains a reference of CLI commands and examples of how to use them.
# Table of Contents

## CHAPTER 1. OPENSIGHT CONTAINER PLATFORM CLI TOOLS OVERVIEW .......................... 10
  1.1. LIST OF CLI TOOLS ...................................................................................... 10

## CHAPTER 2. OPENSIGHT CLI (OC) .......................................................... 11
  2.1. GETTING STARTED WITH THE OPENSIGHT CLI ................................. 11
    2.1.1. About the OPENSIGHT CLI ................................................................. 11
    2.1.2. Installing the OPENSIGHT CLI ......................................................... 11
      2.1.2.1. Installing the OPENSIGHT CLI by downloading the binary ........... 11
      2.1.2.1.1. Installing the OPENSIGHT CLI on Linux ............................... 11
      2.1.2.1.2. Installing the OPENSIGHT CLI on Windows ....................... 12
      2.1.2.1.3. Installing the OPENSIGHT CLI on macOS ........................... 12
      2.1.2.2. Installing the OPENSIGHT CLI by using the web console .......... 13
      2.1.2.2.1. Installing the OPENSIGHT CLI on Linux using the web console ... 13
      2.1.2.2.2. Installing the OPENSIGHT CLI on Windows using the web console 14
      2.1.2.2.3. Installing the OPENSIGHT CLI on macOS using the web console .. 14
      2.1.2.3. Installing the OPENSIGHT CLI by using an RPM ....................... 15
    2.1.3. Logging in to the OPENSIGHT CLI .................................................. 16
    2.1.4. Using the OPENSIGHT CLI ............................................................... 17
      2.1.4.1. Creating a project .................................................................. 17
      2.1.4.2. Creating a new app .................................................................. 18
      2.1.4.3. Viewing pods ......................................................................... 18
      2.1.4.4. Viewing pod logs .................................................................... 18
      2.1.4.5. Viewing the current project ................................................... 19
      2.1.4.6. Viewing the status for the current project ................................. 19
      2.1.4.7. Listing supported API resources ............................................... 19
    2.1.5. Getting help ..................................................................................... 19
    2.1.6. Logging out of the OPENSIGHT CLI .............................................. 21
  2.2. CONFIGURING THE OPENSIGHT CLI ............................................. 21
    2.2.1. Enabling tab completion ................................................................. 21
  2.3. MANAGING CLI PROFILES ........................................................... 22
    2.3.1. About switches between CLI profiles ......................................... 22
    2.3.2. Manual configuration of CLI profiles ......................................... 24
    2.3.3. Load and merge rules ................................................................. 26
  2.4. EXTENDING THE OPENSIGHT CLI WITH PLUG-INS .................... 27
    2.4.1. Writing CLI plug-ins ................................................................. 27
    2.4.2. Installing and using CLI plug-ins .............................................. 28
  2.5. OPENSIGHT CLI DEVELOPER COMMANDS .................................. 29
    2.5.1. Basic CLI commands ................................................................. 29
      2.5.1.1. explain ................................................................................. 30
      2.5.1.2. login .................................................................................. 30
      2.5.1.3. new-app ............................................................................. 30
      2.5.1.4. new-project ................................................................. 30
      2.5.1.5. project ............................................................................. 30
      2.5.1.6. projects ........................................................................... 31
      2.5.1.7. status .............................................................................. 31
    2.5.2. Build and Deploy CLI commands ........................................... 31
      2.5.2.1. cancel-build .................................................................... 31
      2.5.2.2. import-image ................................................................. 31
      2.5.2.3. new-build ........................................................................ 31
      2.5.2.4. rollback ......................................................................... 32
      2.5.2.5. rollout ............................................................................. 32
2.5.2.6. start-build 32
2.5.2.7. tag 32

2.5.3. Application management CLI commands 33
  2.5.3.1. annotate 33
  2.5.3.2. apply 33
  2.5.3.3. autoscale 33
  2.5.3.4. create 33
  2.5.3.5. delete 33
  2.5.3.6. describe 34
  2.5.3.7. edit 34
  2.5.3.8. expose 34
  2.5.3.9. get 34
  2.5.3.10. label 35
  2.5.3.11. scale 35
  2.5.3.12. secrets 35
  2.5.3.13. serviceaccounts 35
  2.5.3.14. set 35

2.5.4. Troubleshooting and debugging CLI commands 35
  2.5.4.1. attach 35
  2.5.4.2. cp 36
  2.5.4.3. debug 36
  2.5.4.4. exec 36
  2.5.4.5. logs 36
  2.5.4.6. port-forward 36
  2.5.4.7. proxy 36
  2.5.4.8. rsh 37
  2.5.4.9. rsync 37
  2.5.4.10. run 37
  2.5.4.11. wait 37

2.5.5. Advanced developer CLI commands 37
  2.5.5.1. api-resources 37
  2.5.5.2. api-versions 38
  2.5.5.3. auth 38
  2.5.5.4. cluster-info 38
  2.5.5.5. convert 38
  2.5.5.6. extract 38
  2.5.5.7. idle 39
  2.5.5.8. image 39
  2.5.5.9. observe 39
  2.5.5.10. patch 39
  2.5.5.11. policy 39
  2.5.5.12. process 39
  2.5.5.13. registry 40
  2.5.5.14. replace 40

2.5.6. Settings CLI commands 40
  2.5.6.1. completion 40
  2.5.6.2. config 40
  2.5.6.3. logout 40
  2.5.6.4. whoami 40

2.5.7. Other developer CLI commands 41
  2.5.7.1. help 41
  2.5.7.2. plugin 41
  2.5.7.3. version 41
2.6. OPENSHIFT CLI ADMINISTRATOR COMMANDS

2.6.1. Cluster management CLI commands
2.6.1.1. inspect
2.6.1.2. must-gather
2.6.1.3. top

2.6.2. Node management CLI commands
2.6.2.1. cordon
2.6.2.2. drain
2.6.2.3. node-logs
2.6.2.4. taint
2.6.2.5. uncordon

2.6.3. Security and policy CLI commands
2.6.3.1. certificate
2.6.3.2. groups
2.6.3.3. new-project
2.6.3.4. pod-network
2.6.3.5. policy

2.6.4. Maintenance CLI commands
2.6.4.1. migrate
2.6.4.2. prune

2.6.5. Configuration CLI commands
2.6.5.1. create-bootstrap-project-template
2.6.5.2. create-error-template
2.6.5.3. create-kubeconfig
2.6.5.4. create-login-template
2.6.5.5. create-provider-selection-template

2.6.6. Other Administrator CLI commands
2.6.6.1. build-chain
2.6.6.2. completion
2.6.6.3. config
2.6.6.4. release
2.6.6.5. verify-image-signature

2.7. USAGE OF OC AND KUBECTL COMMANDS

2.7.1. The oc binary
2.7.2. The kubectl binary

CHAPTER 3. DEVELOPER CLI (ODO)
3.3.2.2. Creating a Node.js application with odo
3.3.2.3. Modifying your application code
3.3.2.4. Adding storage to the application components
3.3.2.5. Adding a custom builder to specify a build image
3.3.2.6. Connecting your application to multiple services using OpenShift Service Catalog
3.3.2.7. Deleting an application
3.3.3. Creating a multicomponent application with odo
3.3.3.1. Creating a project
3.3.3.2. Deploying the back-end component
3.3.3.3. Deploying the front-end component
3.3.3.4. Linking both components
3.3.3.5. Exposing components to the public
3.3.3.6. Modifying the running application
3.3.3.7. Deleting an application
3.3.4. Creating an application with a database
3.3.4.1. Creating a project
3.3.4.2. Deploying the front-end component
3.3.4.3. Deploying a database in interactive mode
3.3.4.4. Deploying a database manually
3.3.4.5. Connecting the database to the front-end application
3.3.4.6. Using devfiles in odo
3.3.4.6.1. About the devfile in odo
3.3.4.6.2. Creating a Java application by using a devfile
3.3.4.6.2.1. Creating a project
3.3.4.6.2.2. Listing available devfile components
3.3.4.6.2.3. Deploying a Java application using a devfile
3.3.4.6.3. Converting an S2I component into a devfile component
3.3.5. Using devfiles in odo
3.3.5.1. About the devfile in odo
3.3.5.2. Creating a Java application by using a devfile
3.3.5.2.1. Creating a project
3.3.5.2.2. Listing available devfile components
3.3.5.2.3. Deploying a Java application using a devfile
3.3.5.3. Converting an S2I component into a devfile component
3.3.6. Working with storage
3.3.6.1. Adding storage to the application components
3.3.6.2. Deleting applications
3.3.6.2.1. Deleting an application
3.3.7. Deleting applications
3.3.8. Debugging applications in odo
3.3.8.1. Debugging an application
3.3.8.2. Configuring debugging parameters
3.3.9. Sample applications
3.3.9.1. Git repository example applications
3.3.9.1.1. httpd
3.3.9.1.2. java
3.3.9.1.3. nodejs
3.3.9.1.4. perl
3.3.9.1.5. php
3.3.9.1.6. python
3.3.9.1.7. ruby
3.3.9.2. Binary example applications
3.3.9.2.1. java
3.4. USING ODO IN A RESTRICTED ENVIRONMENT
3.4.1. About odo in a restricted environment
3.4.2. Pushing the odo init image to the restricted cluster registry
3.4.2.1. Prerequisites
3.4.2.2. Pushing the odo init image to a mirror registry
3.4.2.2.1. Pushing the init image to a mirror registry on Linux
3.4.2.2.2. Pushing the init image to a mirror registry on MacOS
3.4.2.2.3. Pushing the init image to a mirror registry on Windows
3.4.2.3. Pushing the odo init image to an internal registry directly
  3.4.2.3.1. Pushing the init image directly on Linux
  3.4.2.3.2. Pushing the init image directly on MacOS
  3.4.2.3.3. Pushing the init image directly on Windows
3.4.3. Creating and deploying a component to the disconnected cluster
  3.4.3.1. Prerequisites
  3.4.3.2. Mirroring a supported builder image
  3.4.3.3. Overwriting the mirror registry
  3.4.3.4. Creating a Node.js application with odo
3.5. CREATING INSTANCES OF SERVICES MANAGED BY OPERATORS
  3.5.1. Prerequisites
  3.5.2. Creating a project
  3.5.3. Listing available services from the Operators installed on the cluster
  3.5.4. Creating a service from an Operator
  3.5.5. Creating services from YAML files
3.6. MANAGING ENVIRONMENT VARIABLES
  3.6.1. Setting and unsetting environment variables
3.7. CONFIGURING THE ODO CLI
  3.7.1. Viewing the current configuration
  3.7.2. Setting a value
  3.7.3. Setting a value
  3.7.4. Preference key table
  3.7.5. Ignoring files or patterns
3.8. ODO CLI REFERENCE
  3.8.1. odo build-images
  3.8.2. odo catalog
    3.8.2.1. Components
      3.8.2.1.1. Listing components
      3.8.2.1.2. Getting information about a component
    3.8.2.2. Services
      3.8.2.2.1. Listing services
      3.8.2.2.2. Searching services
      3.8.2.2.3. Getting information about a service
  3.8.3. odo create
  3.8.4. odo delete
    3.8.4.1. Deleting a component
    3.8.4.2. Undeploying devfile Kubernetes components
    3.8.4.3. Delete all
    3.8.4.4. Available flags
  3.8.5. odo deploy
  3.8.6. odo link
    3.8.6.1. Various linking options
      3.8.6.1.1. Default behavior
      3.8.6.1.2. The --inlined flag
      3.8.6.1.3. The --map flag
      3.8.6.1.4. The --bind-as-files flag
    3.8.6.2. Examples
      3.8.6.2.1. Default odo link
      3.8.6.2.2. Using odo link with the --inlined flag
3.8.6.2.3. Custom bindings
  3.8.6.2.3.1. To inline or not?
  3.8.6.3. Binding as files
  3.8.6.4. --bind-as-files examples
    3.8.6.4.1. Using the default odo link
    3.8.6.4.2. Using --inlined
    3.8.6.4.3. Custom bindings
3.8.7. odo registry
  3.8.7.1. Listing the registries
  3.8.7.2. Adding a registry
  3.8.7.3. Deleting a registry
  3.8.7.4. Updating a registry
3.8.8. odo service
  3.8.8.1. Creating a new service
    3.8.8.1.1. Inlining the manifest
    3.8.8.1.2. Configuring the service
      3.8.8.1.2.1. Using command-line arguments
      3.8.8.1.2.2. Using a file
    3.8.8.2. Deleting a service
  3.8.8.3. Listing services
  3.8.8.4. Getting information about a service
3.8.9. odo storage
  3.8.9.1. Adding a storage volume
  3.8.9.2. Listing the storage volumes
  3.8.9.3. Deleting a storage volume
  3.8.9.4. Adding storage to specific container
3.8.10. Common flags
3.8.11. JSON output
3.9. ODO ARCHITECTURE
  3.9.1. Developer setup
  3.9.2. OpenShift source-to-image
  3.9.3. OpenShift cluster objects
    3.9.3.1. Init Containers
      3.9.3.1.1. copy-supervisord
      3.9.3.1.2. copy-files-to-volume
    3.9.3.2. Application container
    3.9.3.3. Persistent volumes and persistent volume claims
    3.9.3.4. emptyDir volume
    3.9.3.5. Service
  3.9.4. odo push workflow
3.10. ODO RELEASE NOTES
  3.10.1. Notable changes and improvements in odo version 2.5.0
  3.10.2. Bug fixes
  3.10.3. Getting support

CHAPTER 4. HELM CLI

4.1. GETTING STARTED WITH HELM 3
  4.1.1. Understanding Helm
    4.1.1.1. Key features
4.1.2.4. On MacOS
4.1.3. Installing a Helm chart on an OpenShift Container Platform cluster
4.1.4. Creating a custom Helm chart on OpenShift Container Platform
4.2. CONFIGURING CUSTOM HELM CHART REPOSITORIES
  4.2.1. Adding custom Helm chart repositories
  4.2.2. Creating credentials and CA certificates to add Helm chart repositories

CHAPTER 5. KNATIVE CLI (KN) FOR USE WITH OPENSHIFT SERVERLESS ........................................ 135
  5.1. KEY FEATURES ........................................ 135
  5.2. INSTALLING THE KNATIVE CLI ....................... 135

CHAPTER 6. PIPELINES CLI (TKN) ........................................ 136
  6.1. INSTALLING TKN ........................................ 136
    6.1.1. Installing Red Hat OpenShift Pipelines CLI (tkn) on Linux ........................................ 136
    6.1.2. Installing Red Hat OpenShift Pipelines CLI (tkn) on Linux using an RPM .................... 136
    6.1.3. Installing Red Hat OpenShift Pipelines CLI (tkn) on Windows ..................................... 137
    6.1.4. Installing Red Hat OpenShift Pipelines CLI (tkn) on macOS ..................................... 137
  6.2. CONFIGURING THE OPENSHIFT PIPELINES TKN CLI ...
    6.2.1. Enabling tab completion ............................ 138
  6.3. OPENSHIFT PIPELINES TKN REFERENCE ............ 138
    6.3.1. Basic syntax ........................................ 138
    6.3.2. Global options .................................... 138
    6.3.3. Utility commands .................................. 138
      6.3.3.1. tkn ............................................ 138
      6.3.3.2. completion [shell] ............................ 139
      6.3.3.3. version ....................................... 139
    6.3.4. Pipelines management commands .................. 139
      6.3.4.1. pipeline ...................................... 139
      6.3.4.2. pipeline delete ................................ 139
      6.3.4.3. pipeline describe ............................. 139
      6.3.4.4. pipeline list .................................. 139
      6.3.4.5. pipeline logs ................................ 140
      6.3.4.6. pipeline start ................................ 140
    6.3.5. PipelineRun commands ............................ 140
      6.3.5.1. pipelinerun .................................... 140
      6.3.5.2. pipelinerun cancel ............................ 140
      6.3.5.3. pipelinerun delete ............................ 140
      6.3.5.4. pipelinerun describe ........................ 140
      6.3.5.5. pipelinerun list ................................ 141
      6.3.5.6. pipelinerun logs ................................ 141
    6.3.6. Task management commands ....................... 141
      6.3.6.1. task .......................................... 141
      6.3.6.2. task delete .................................... 141
      6.3.6.3. task describe .................................. 141
      6.3.6.4. task list ....................................... 141
      6.3.6.5. task logs ....................................... 142
      6.3.6.6. task start ...................................... 142
    6.3.7. TaskRun commands ................................ 142
      6.3.7.1. taskrun ........................................ 142
      6.3.7.2. taskrun cancel .................................. 142
      6.3.7.3. taskrun delete .................................. 142
      6.3.7.4. taskrun describe ................................ 142
      6.3.7.5. taskrun list ..................................... 143
6.3.7.6. taskrun logs
6.3.8. Condition management commands
6.3.8.1. condition
6.3.8.2. condition delete
6.3.8.3. condition describe
6.3.8.4. condition list
6.3.9. Pipeline Resource management commands
6.3.9.1. resource
6.3.9.2. resource create
6.3.9.3. resource delete
6.3.9.4. resource describe
6.3.9.5. resource list
6.3.10. ClusterTask management commands
6.3.10.1. clustertask
6.3.10.2. clustertask delete
6.3.10.3. clustertask describe
6.3.10.4. clustertask list
6.3.10.5. clustertask start
6.3.11. Trigger management commands
6.3.11.1. eventlistener
6.3.11.2. eventlistener delete
6.3.11.3. eventlistener describe
6.3.11.4. eventlistener list
6.3.11.5. triggerbinding
6.3.11.6. triggerbinding delete
6.3.11.7. triggerbinding describe
6.3.11.8. triggerbinding list
6.3.11.9. triggertemplate
6.3.11.10. triggertemplate delete
6.3.11.11. triggertemplate describe
6.3.11.12. triggertemplate list
6.3.11.13. clustertriggerbinding
6.3.11.14. clustertriggerbinding delete
6.3.11.15. clustertriggerbinding describe
6.3.11.16. clustertriggerbinding list

CHAPTER 7. OPM CLI

7.1. ABOUT OPM
7.2. INSTALLING OPM
7.3. ADDITIONAL RESOURCES
CHAPTER 1. OPENSOURCE CONTAINER PLATFORM CLI TOOLS

OVERVIEW

A user performs a range of operations while working on OpenShift Container Platform such as the following:

- Managing clusters
- Building, deploying, and managing applications
- Managing deployment processes
- Developing Operators
- Creating and maintaining Operator catalogs

OpenShift Container Platform offers a set of command-line interface (CLI) tools that simplify these tasks by enabling users to perform various administration and development operations from the terminal. These tools expose simple commands to manage the applications, as well as interact with each component of the system.

1.1. LIST OF CLI TOOLS

The following set of CLI tools are available in OpenShift Container Platform:

- **OpenShift CLI (oc)**: This is the most commonly used CLI tool by OpenShift Container Platform users. It helps both cluster administrators and developers to perform end-to-end operations across OpenShift Container Platform using the terminal. Unlike the web console, it allows the user to work directly with the project source code using command scripts.

- **Developer CLI (odo)**: The odo CLI tool helps developers focus on their main goal of creating and maintaining applications on OpenShift Container Platform by abstracting away complex Kubernetes and OpenShift Container Platform concepts. It helps the developers to write, build, and debug applications on a cluster from the terminal without the need to administer the cluster.

- **Helm CLI**: Helm is a package manager for Kubernetes applications which enables defining, installing, and upgrading applications packaged as Helm charts. Helm CLI helps the user deploy applications and services to OpenShift Container Platform clusters using simple commands from the terminal.

- **Knative CLI (kn)**: The kn CLI tool provides simple and intuitive terminal commands that can be used to interact with OpenShift Serverless components, such as Knative Serving and Eventing.

- **Pipelines CLI (tkn)**: OpenShift Pipelines is a continuous integration and continuous delivery (CI/CD) solution in OpenShift Container Platform, which internally uses Tekton. The tkn CLI tool provides simple and intuitive commands to interact with OpenShift Pipelines using the terminal.

- **opm CLI**: The opm CLI tool helps the Operator developers and cluster administrators to create and maintain the catalogs of Operators from the terminal.
2.1. GETTING STARTED WITH THE OPENSHIFT CLI

2.1.1. About the OpenShift CLI

With the OpenShift command-line interface (CLI), the `oc` command, you can create applications and manage OpenShift Container Platform projects from a terminal. The OpenShift CLI is ideal in the following situations:

- Working directly with project source code
- Scripting OpenShift Container Platform operations
- Managing projects while restricted by bandwidth resources and the web console is unavailable

2.1.2. Installing the OpenShift CLI

You can install the OpenShift CLI (oc) either by downloading the binary or by using an RPM.

2.1.2.1. Installing the OpenShift CLI by downloading the binary

You can install the OpenShift CLI (oc) in order to interact with OpenShift Container Platform from a command-line interface. You can install oc on Linux, Windows, or macOS.

**IMPORTANT**

If you installed an earlier version of oc, you cannot use it to complete all of the commands in OpenShift Container Platform 4.6. Download and install the new version of oc.

2.1.2.1.1. Installing the OpenShift CLI on Linux

You can install the OpenShift CLI (oc) binary on Linux by using the following procedure.

**Procedure**

2. Select the appropriate version in the Version drop-down menu.
3. Click Download Now next to the OpenShift v4.6 Linux Client entry and save the file.
4. Unpack the archive:
   
   ```bash
   $ tar xvzf <file>
   ``

5. Place the oc binary in a directory that is on your PATH.
   To check your PATH, execute the following command:

   ```bash
   $ echo $PATH
   ```
After you install the OpenShift CLI, it is available using the `oc` command:

```
$ oc <command>
```

### 2.1.2.1.2. Installing the OpenShift CLI on Windows

You can install the OpenShift CLI (`oc`) binary on Windows by using the following procedure.

**Procedure**

2. Select the appropriate version in the **Version** drop-down menu.
3. Click **Download Now** next to the **OpenShift v4.6 Windows Client** entry and save the file.
4. Unzip the archive with a ZIP program.
5. Move the `oc` binary to a directory that is on your **PATH**.
   To check your **PATH**, open the command prompt and execute the following command:

```
C:\> path
```

After you install the OpenShift CLI, it is available using the `oc` command:

```
C:\> oc <command>
```

### 2.1.2.1.3. Installing the OpenShift CLI on macOS

You can install the OpenShift CLI (`oc`) binary on macOS by using the following procedure.

**Procedure**

2. Select the appropriate version in the **Version** drop-down menu.
3. Click **Download Now** next to the **OpenShift v4.6 MacOSX Client** entry and save the file.
4. Unpack and unzip the archive.
5. Move the `oc` binary to a directory on your **PATH**.
   To check your **PATH**, open a terminal and execute the following command:

```
$ echo $PATH
```

After you install the OpenShift CLI, it is available using the `oc` command:

```
$ oc <command>
```

---

OpenShift Container Platform 4.6 CLI tools

12
2.1.2.2. Installing the OpenShift CLI by using the web console

You can install the OpenShift CLI (oc) to interact with OpenShift Container Platform from a web console. You can install oc on Linux, Windows, or macOS.

IMPORTANT

If you installed an earlier version of oc, you cannot use it to complete all of the commands in OpenShift Container Platform 4.6. Download and install the new version of oc.

2.1.2.2.1. Installing the OpenShift CLI on Linux using the web console

You can install the OpenShift CLI (oc) binary on Linux by using the following procedure.

Procedure

1. From the web console, click ?.

2. Click Command Line Tools

3. Select appropriate oc binary for your Linux platform, and then click Download oc for Linux

4. Save the file.

5. Unpack the archive.

   $ tar xvzf <file>

6. Move the oc binary to a directory that is on your PATH.

   To check your PATH, execute the following command:

   $ echo $PATH

After you install the OpenShift CLI, it is available using the oc command:
2.1.2.2. Installing the OpenShift CLI on Windows using the web console

You can install the OpenShift CLI (oc) binary on Windows by using the following procedure.

Procedure

1. From the web console, click ?.

2. Click Command Line Tools

3. Select the oc binary for Windows platform, and then click Download oc for Windows for x86_64.

4. Save the file.

5. Unzip the archive with a ZIP program.

6. Move the oc binary to a directory that is on your PATH.

To check your PATH, open the command prompt and execute the following command:

```bash
C:\> path
```

After you install the OpenShift CLI, it is available using the oc command:

```bash
C:\> oc <command>
```

2.1.2.2.3. Installing the OpenShift CLI on macOS using the web console

You can install the OpenShift CLI (oc) binary on macOS by using the following procedure.

Procedure
1. From the web console, click ?.

2. Click Command Line Tools.

3. Select the oc binary for macOS platform, and then click Download oc for Mac for x86_64.

4. Save the file.

5. Unpack and unzip the archive.

6. Move the oc binary to a directory on your PATH. To check your PATH, open a terminal and execute the following command:

   ```bash
   $ echo $PATH
   $ oc <command>
   ```

   After you install the OpenShift CLI, it is available using the oc command:

   ```bash
   $ oc <command>
   ```

2.1.2.3. Installing the OpenShift CLI by using an RPM

For Red Hat Enterprise Linux (RHEL), you can install the OpenShift CLI (oc) as an RPM if you have an active OpenShift Container Platform subscription on your Red Hat account.

**Prerequisites**

- Must have root or sudo privileges.

**Procedure**

1. Register with Red Hat Subscription Manager:

   ```bash
   # subscription-manager register
   ```
2. Pull the latest subscription data:

```bash
# subscription-manager refresh
```

3. List the available subscriptions:

```bash
# subscription-manager list --available --matches "OpenShift"
```

4. In the output for the previous command, find the pool ID for an OpenShift Container Platform subscription and attach the subscription to the registered system:

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

5. Enable the repositories required by OpenShift Container Platform 4.6.

   - For Red Hat Enterprise Linux 8:
     ```bash
     # subscription-manager repos --enable="rhocp-4.6-for-rhel-8-x86_64-rpms"
     ```
   - For Red Hat Enterprise Linux 7:
     ```bash
     # subscription-manager repos --enable="rhel-7-server-ose-4.6-rpms"
     ```

6. Install the `openshift-clients` package:

```bash
# yum install openshift-clients
```

After you install the CLI, it is available using the `oc` command:

```bash
$ oc <command>
```

### 2.1.3. Logging in to the OpenShift CLI

You can log in to the `oc` CLI to access and manage your cluster.

**Prerequisites**

- You must have access to an OpenShift Container Platform cluster.
- You must have installed the CLI.

**NOTE**

To access a cluster that is accessible only over an HTTP proxy server, you can set the `HTTP_PROXY`, `HTTPS_PROXY` and `NO_PROXY` variables. These environment variables are respected by the `oc` CLI so that all communication with the cluster goes through the HTTP proxy.

**Procedure**

- Log in to the CLI using the `oc login` command and enter the required information when prompted.
$ oc login

Example output

Server [https://localhost:8443]: https://openshift.example.com:6443 1
The server uses a certificate signed by an unknown authority.
You can bypass the certificate check, but any data you send to the server could be intercepted by others.
Use insecure connections? (y/n): y 2

Authentication required for https://openshift.example.com:6443 (openshift)
Username: user1 3
Password: 4
Login successful.

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

   oc new-project <projectname>

Welcome! See ‘oc help’ to get started.

1. Enter the OpenShift Container Platform server URL.
2. Enter whether to use insecure connections.
3. Enter the user name to log in as.
4. Enter the user’s password.

NOTE

If you are logged in to the web console, you can generate an oc login command that includes your token and server information. You can use the command to log in to the OpenShift Container Platform CLI without the interactive prompts. To generate the command, select Copy login command from the username drop-down menu at the top right of the web console.

You can now create a project or issue other commands for managing your cluster.

2.1.4. Using the OpenShift CLI

Review the following sections to learn how to complete common tasks using the CLI.

2.1.4.1. Creating a project

Use the oc new-project command to create a new project.

   $ oc new-project my-project

Example output
2.1.4.2. Creating a new app

Use the `oc new-app` command to create a new application.

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

Example output

```bash
--> Found image 40de956 (9 days old) in imagestream "openshift/php" under tag "7.2" for "php"
...
Run 'oc status' to view your app.
```

2.1.4.3. Viewing pods

Use the `oc get pods` command to view the pods for the current project.

```bash
$ oc get pods -o wide
```

Example output

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
<th>IP</th>
<th>NODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>cakephp-ex-1-build</td>
<td>0/1</td>
<td>Completed</td>
<td>0</td>
<td>5m45s</td>
<td>10.131.0.10</td>
<td>ip-10-0-141-74.ec2.internal</td>
</tr>
<tr>
<td>cakephp-ex-1-deploy</td>
<td>0/1</td>
<td>Completed</td>
<td>0</td>
<td>3m44s</td>
<td>10.129.2.9</td>
<td>ip-10-0-147-65.ec2.internal</td>
</tr>
<tr>
<td>cakephp-ex-1-ktz97</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>3m33s</td>
<td>10.128.2.11</td>
<td>ip-10-0-168-105.ec2.internal</td>
</tr>
</tbody>
</table>

2.1.4.4. Viewing pod logs

Use the `oc logs` command to view logs for a particular pod.

```bash
$ oc logs cakephp-ex-1-deploy
```

Example output

```bash
--> Scaling cakephp-ex-1 to 1
--> Success
```
2.1.4.5. Viewing the current project

Use the `oc project` command to view the current project.

```bash
$ oc project
```

Example output

Using project "my-project" on server "https://openshift.example.com:6443".

2.1.4.6. Viewing the status for the current project

Use the `oc status` command to view information about the current project, such as services, deployments, and build configs.

```bash
$ oc status
```

Example output

In project my-project on server https://openshift.example.com:6443

svc/cakephp-ex - 172.30.236.80 ports 8080, 8443
    dc/cakephp-ex deploys istag/cakephp-ex:latest <-
    bc/cakephp-ex source builds https://github.com/sclorg/cakephp-ex on openshift/php:7.2
deployment #1 deployed 2 minutes ago - 1 pod

3 infos identified, use 'oc status --suggest' to see details.

2.1.4.7. Listing supported API resources

Use the `oc api-resources` command to view the list of supported API resources on the server.

```bash
$ oc api-resources
```

Example output

<table>
<thead>
<tr>
<th>NAME</th>
<th>SHORTNAMES</th>
<th>APIGROUP</th>
<th>NAMESPA</th>
<th>KIND</th>
</tr>
</thead>
<tbody>
<tr>
<td>bindings</td>
<td></td>
<td></td>
<td>true</td>
<td>Binding</td>
</tr>
<tr>
<td>componentstatuses</td>
<td>cs</td>
<td></td>
<td>false</td>
<td>ComponentStatus</td>
</tr>
<tr>
<td>configmaps</td>
<td>cm</td>
<td></td>
<td>true</td>
<td>ConfigMap</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.1.5. Getting help

You can get help with CLI commands and OpenShift Container Platform resources in the following ways.

- Use `oc help` to get a list and description of all available CLI commands:

  Example: Get general help for the CLI
Example output

OpenShift Client

This client helps you develop, build, deploy, and run your applications on any OpenShift or Kubernetes compatible platform. It also includes the administrative commands for managing a cluster under the ‘adm’ subcommand.

Usage:
  oc [flags]

Basic Commands:
  login          Log in to a server
  new-project    Request a new project
  new-app        Create a new application
  ...

- Use the --help flag to get help about a specific CLI command:

  Example: Get help for the oc create command

  $ oc create --help

  Example output

  Create a resource by filename or stdin
  JSON and YAML formats are accepted.

  Usage:
  oc create -f FILENAME [flags]
  ...

- Use the oc explain command to view the description and fields for a particular resource:

  Example: View documentation for the Pod resource

  $ oc explain pods

  Example output

  KIND:    Pod
  VERSION: v1

  DESCRIPTION:
  Pod is a collection of containers that can run on a host. This resource is created by clients and scheduled onto hosts.
### 2.1.6. Logging out of the OpenShift CLI

You can log out the OpenShift CLI to end your current session.

- Use the `oc logout` command.

```bash
$ oc logout
```

**Example output**

Logged "user1" out on "https://openshift.example.com"

This deletes the saved authentication token from the server and removes it from your configuration file.

### 2.2. CONFIGURING THE OPENSHIFT CLI

#### 2.2.1. Enabling tab completion

After you install the `oc` CLI tool, you can enable tab completion to automatically complete `oc` commands or suggest options when you press Tab.

**Prerequisites**

- You must have the `oc` CLI tool installed.
- You must have the package `bash-completion` installed.

**Procedure**

The following procedure enables tab completion for Bash.

1. Save the Bash completion code to a file.

   ```bash
   $ oc completion bash > oc_bash_completion
   ``

2. Copy the file to `/etc/bash_completion.d/`.

   ```bash
   $ sudo cp oc_bash_completion /etc/bash_completion.d/
   ``

Tab completion is enabled when you open a new terminal.
2.3. MANAGING CLI PROFILES

A CLI configuration file allows you to configure different profiles, or contexts, for use with the CLI tools overview. A context consists of user authentication and OpenShift Container Platform server information associated with a nickname.

2.3.1. About switches between CLI profiles

Contexts allow you to easily switch between multiple users across multiple OpenShift Container Platform servers, or clusters, when using CLI operations. Nicknames make managing CLI configurations easier by providing short-hand references to contexts, user credentials, and cluster details. After logging in with the CLI for the first time, OpenShift Container Platform 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 manually configuring CLI profiles, the updated information is stored in the configuration file:

CLI config file

```yaml
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
kind: Config
preferences: {}
users: 4
  - name: alice/openshift1.example.com:8443
    user:
      token: xZHd2piv5_9vQrg-SKXRJ2DsI9SceNjdhNTljEKTb8k
```

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

2 This contexts section defines two contexts: one nicknamed alice-project/openshift1.example.com:8443/alice, using the alice-project project, openshift1.example.com:8443 cluster, and alice user, and another nicknamed joe-project/openshift1.example.com:8443/alice, using the joe-project project,
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.

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 which 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 or oc project command to verify your current working environment:

Verify the current working environment

$ oc status

Example output

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 in this example.

List the current project

$ oc project

Example output

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

You can run the oc login command again and supply the required information during the interactive process, to log in using any other combination of user credentials and cluster details. 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 enter the name of the project:

$ oc project alice-project
Example output

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 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 config file. The following command logs in and switches to the default project:

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

### 2.3.2. Manual configuration of CLI profiles

**NOTE**

This section covers more advanced usage of CLI configurations. In most situations, you can 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 config files, you can use the **oc config** command instead of directly modifying the files. The **oc config** command includes a number of helpful sub-commands for this purpose:

<table>
<thead>
<tr>
<th>Table 2.1. CLI configuration subcommands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subcommand</td>
</tr>
<tr>
<td>set-cluster</td>
</tr>
<tr>
<td>set-context</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>use-context</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
### Subcommand | Usage
---|---
**set** | Sets an individual value in the CLI config file.

```bash
$ 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 config file.

```bash
$ 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.

```bash
$ oc config view
```

Displays the result of the specified CLI config file.

```bash
$ oc config view --config=<specific_filename>
```

### Example usage

- Log in as a user that uses an access token. This token is used by the **alice** user:

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

- View the cluster entry automatically created:

```bash
$ oc config view
```

### Example output

```yaml
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
```

---

**CHAPTER 2. OPENSHIFT CLI (OC)**

25
*current-context: default/openshift1-example-com/alice*

| kind: Config |
| preferences: {} |
| users: |
| - name: alice/openshift1.example.com |
| user: |
| token: ns7yVhuRNpDM9cgzfhhxQ7bM5s7N2ZVrkZepSRf4LC0 |

- Update the current context to have users log in to the desired namespace:

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

- Examine the current context, to confirm that the changes are implemented:

```bash
$ oc whoami -c
```

All subsequent CLI operations uses the new context, unless otherwise specified by overriding CLI options or until the context is switched.

### 2.3.3. Load and merge rules

You can follow these rules, when issuing CLI operations for the loading and merging order for the CLI configuration:

- CLI config 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 is set once and no merging takes place.
  
  - If the `$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.

- The context to use is determined based on the first match in the following flow:
  
  - The value of the `--context` option.
  
  - The `current-context` value from the CLI config file.
  
  - An empty value is allowed at this stage.

- 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 match in the following flow, which is run once for the user and once for the cluster:
  
  - The value of the `--user` for user name and `--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.
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 match in the following flow:

- 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.

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 config file values. Valid command line options are:

- `--auth-path`
- `--client-certificate`
- `--client-key`
- `--token`
- For any information that is still missing, default values are used and prompts are given for additional information.

### 2.4. EXTENDING THE OPENSHIFT CLI WITH PLUG-INS

You can write and install plug-ins to build on the default `oc` commands, allowing you to perform new and more complex tasks with the OpenShift Container Platform CLI.

#### 2.4.1. Writing CLI plug-ins

You can write a plug-in for the OpenShift Container Platform CLI in any programming language or script that allows you to write command-line commands. Note that you can not use a plug-in to overwrite an existing `oc` command.

**IMPORTANT**

OpenShift CLI plug-ins are currently a Technology Preview feature. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

See the [Red Hat Technology Preview features support scope](#) for more information.

**Procedure**
This procedure creates a simple Bash plug-in that prints a message to the terminal when the `oc foo` command is issued.

1. Create a file called `oc-foo`.
   When naming your plug-in file, keep the following in mind:
   - The file must begin with `oc-` or `kubectl-` in order to be recognized as a plug-in.
   - The file name determines the command that invokes the plug-in. For example, a plug-in with the file name `oc-foo-bar` can be invoked by a command of `oc foo bar`. You can also use underscores if you want the command to contain dashes. For example, a plug-in with the file name `oc-foo_bar` can be invoked by a command of `oc foo-bar`.

2. Add the following contents to the file.

   ```bash
   #!/bin/bash
   # optional argument handling
   if [[ "$1" == "version" ]]
   then
     echo "1.0.0"
     exit 0
   fi
   
   # optional argument handling
   if [[ "$1" == "config" ]]
   then
     echo $KUBECONFIG
     exit $KUBECONFIG
   fi
   
   echo "I am a plugin named kubectl-foo"
   ```

   After you install this plug-in for the OpenShift Container Platform CLI, it can be invoked using the `oc foo` command.

Additional resources

- Review the [Sample plug-in repository](#) for an example of a plug-in written in Go.
- Review the [CLI runtime repository](#) for a set of utilities to assist in writing plug-ins in Go.

2.4.2. Installing and using CLI plug-ins

After you write a custom plug-in for the OpenShift Container Platform CLI, you must install it to use the functionality that it provides.
IMPORTANT

OpenShift CLI plug-ins are currently a Technology Preview feature. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

See the Red Hat Technology Preview features support scope for more information.

Prerequisites

- You must have the oc CLI tool installed.
- You must have a CLI plug-in file that begins with oc- or kubectl-.

Procedure

1. If necessary, update the plug-in file to be executable.

   $ chmod +x <plugin_file>

2. Place the file anywhere in your PATH, such as /usr/local/bin/.

   $ sudo mv <plugin_file> /usr/local/bin/

3. Run oc plugin list to make sure that the plug-in is listed.

   $ oc plugin list

Example output

The following compatible plugins are available:

/usr/local/bin/<plugin_file>

If your plug-in is not listed here, verify that the file begins with oc- or kubectl-, is executable, and is on your PATH.

4. Invoke the new command or option introduced by the plug-in.

   For example, if you built and installed the kubectl-ns plug-in from the Sample plug-in repository, you can use the following command to view the current namespace.

   $ oc ns

   Note that the command to invoke the plug-in depends on the plug-in file name. For example, a plug-in with the file name of oc-foo-bar is invoked by the oc foo bar command.

2.5. OPENSHIFT CLI DEVELOPER COMMANDS

2.5.1. Basic CLI commands
2.5.1.1. explain
Display documentation for a certain resource.

Example: Display documentation for pods

$ oc explain pods

2.5.1.2. login
Log in to the OpenShift Container Platform server and save login information for subsequent use.

Example: Interactive login

$ oc login

Example: Log in specifying a user name

$ oc login -u user1

2.5.1.3. new-app
Create a new application by specifying source code, a template, or an image.

Example: Create a new application from a local Git repository

$ oc new-app .

Example: Create a new application from a remote Git repository

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

Example: Create a new application from a private remote repository

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

2.5.1.4. new-project
Create a new project and switch to it as the default project in your configuration.

Example: Create a new project

$ oc new-project myproject

2.5.1.5. project
Switch to another project and make it the default in your configuration.

Example: Switch to a different project
2.5.1.6. projects
Display information about the current active project and existing projects on the server.

Example: List all projects

$ oc projects

2.5.1.7. status
Show a high-level overview of the current project.

Example: Show the status of the current project

$ oc status

2.5.2. Build and Deploy CLI commands

2.5.2.1. cancel-build
Cancel a running, pending, or new build.

Example: Cancel a build

$ oc cancel-build python-1

Example: Cancel all pending builds from the python build config

$ oc cancel-build buildconfig/python --state=pending

2.5.2.2. import-image
Import the latest tag and image information from an image repository.

Example: Import the latest image information

$ oc import-image my-ruby

2.5.2.3. new-build
Create a new build config from source code.

Example: Create a build config from a local Git repository

$ oc new-build .

Example: Create a build config from a remote Git repository

$ oc new-build .
2.5.2.4. rollback

Revert an application back to a previous deployment.

**Example: Roll back to the last successful deployment**

```bash
$ oc rollback php
```

**Example: Roll back to a specific version**

```bash
$ oc rollback php --to-version=3
```

2.5.2.5. rollout

Start a new rollout, view its status or history, or roll back to a previous revision of your application.

**Example: Roll back to the last successful deployment**

```bash
$ oc rollout undo deploymentconfig/php
```

**Example: Start a new rollout for a deployment with its latest state**

```bash
$ oc rollout latest deploymentconfig/php
```

2.5.2.6. start-build

Start a build from a build config or copy an existing build.

**Example: Start a build from the specified build config**

```bash
$ oc start-build python
```

**Example: Start a build from a previous build**

```bash
$ oc start-build --from-build=python-1
```

**Example: Set an environment variable to use for the current build**

```bash
$ oc start-build python --env=mykey=myvalue
```

2.5.2.7. tag

Tag existing images into image streams.

**Example: Configure the ruby image’s latest tag to refer to the image for the 2.0 tag**

```bash
$ oc tag ruby:latest ruby:2.0
```
2.5.3. Application management CLI commands

2.5.3.1. annotate
Update the annotations on one or more resources.

Example: Add an annotation to a route

$ oc annotate route/test-route haproxy.router.openshift.io/ip_whitelist="192.168.1.10"

Example: Remove the annotation from the route

$ oc annotate route/test-route haproxy.router.openshift.io/ip_whitelist-

2.5.3.2. apply
Apply a configuration to a resource by file name or standard in (stdin) in JSON or YAML format.

Example: Apply the configuration in pod.json to a pod

$ oc apply -f pod.json

2.5.3.3. autoscale
Autoscale a deployment or replication controller.

Example: Autoscale to a minimum of two and maximum of five pods

$ oc autoscale deploymentconfig/parksmap-katacoda --min=2 --max=5

2.5.3.4. create
Create a resource by file name or standard in (stdin) in JSON or YAML format.

Example: Create a pod using the content in pod.json

$ oc create -f pod.json

2.5.3.5. delete
Delete a resource.

Example: Delete a pod named parksmap-katacoda-1-qfqz4

$ oc delete pod/parksmap-katacoda-1-qfqz4

Example: Delete all pods with the app=parksmap-katacoda label

$ oc delete pods -l app=parksmap-katacoda
2.5.3.6. describe
Return detailed information about a specific object.

Example: Describe a deployment named example
$ oc describe deployment/example

Example: Describe all pods
$ oc describe pods

2.5.3.7. edit
Edit a resource.

Example: Edit a deployment using the default editor
$ oc edit deploymentconfig/parksmap-katacoda

Example: Edit a deployment using a different editor
$ OC_EDITOR="nano" oc edit deploymentconfig/parksmap-katacoda

Example: Edit a deployment in JSON format
$ oc edit deploymentconfig/parksmap-katacoda -o json

2.5.3.8. expose
Expose a service externally as a route.

Example: Expose a service
$ oc expose service/parksmap-katacoda

Example: Expose a service and specify the host name
$ oc expose service/parksmap-katacoda --hostname=www.my-host.com

2.5.3.9. get
Display one or more resources.

Example: List pods in the default namespace
$ oc get pods -n default

Example: Get details about the python deployment in JSON format
2.5.3.10. label
Update the labels on one or more resources.

Example: Update the `python-1-mz2rf` pod with the label `status` set to `unhealthy`

```
$ oc label pod/python-1-mz2rf status=unhealthy
```

2.5.3.11. scale
Set the desired number of replicas for a replication controller or a deployment.

Example: Scale the `ruby-app` deployment to three pods

```
$ oc scale deploymentconfig/ruby-app --replicas=3
```

2.5.3.12. secrets
Manage secrets in your project.

Example: Allow `my-pull-secret` to be used as an image pull secret by the `default` service account

```
$ oc secrets link default my-pull-secret --for=pull
```

2.5.3.13. serviceaccounts
Get a token assigned to a service account or create a new token or `kubeconfig` file for a service account.

Example: Get the token assigned to the `default` service account

```
$ oc serviceaccounts get-token default
```

2.5.3.14. set
Configure existing application resources.

Example: Set the name of a secret on a build config

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

2.5.4. Troubleshooting and debugging CLI commands

2.5.4.1. attach
Attach the shell to a running container.
Example: Get output from the python container from pod python-1-mz2rf

```bash
$ oc attach python-1-mz2rf -c python
```

2.5.4.2. cp

Copy files and directories to and from containers.

Example: Copy a file from the python-1-mz2rf pod to the local file system

```bash
$ oc cp default/python-1-mz2rf:/opt/app-root/src/README.md ~/mydirectory/.
```

2.5.4.3. debug

Launch a command shell to debug a running application.

Example: Debug the python deployment

```bash
$ oc debug deploymentconfig/python
```

2.5.4.4. exec

Execute a command in a container.

Example: Execute the ls command in the python container from pod python-1-mz2rf

```bash
$ oc exec python-1-mz2rf -c python ls
```

2.5.4.5. logs

Retrieve the log output for a specific build, build config, deployment, or pod.

Example: Stream the latest logs from the python deployment

```bash
$ oc logs -f deploymentconfig/python
```

2.5.4.6. port-forward

Forward one or more local ports to a pod.

Example: Listen on port 8888 locally and forward to port 5000 in the pod

```bash
$ oc port-forward python-1-mz2rf 8888:5000
```

2.5.4.7. proxy

Run a proxy to the Kubernetes API server.

Example: Run a proxy to the API server on port 8011 serving static content from .local/www/
2.5.4.8. rsh
Open a remote shell session to a container.

Example: Open a shell session on the first container in the python-1-mz2rf pod

```bash
$ oc rsh python-1-mz2rf
```

2.5.4.9. rsync
Copy contents of a directory to or from a running pod container. Only changed files are copied using the `rsync` command from your operating system.

Example: Synchronize files from a local directory with a pod directory

```bash
$ oc rsync ~/mydirectory/ python-1-mz2rf:/opt/app-root/src/
```

2.5.4.10. run
Create a pod running a particular image.

Example: Start a pod running the perl image

```bash
$ oc run my-test --image=perl
```

2.5.4.11. wait
Wait for a specific condition on one or more resources.

**NOTE**
This command is experimental and might change without notice.

Example: Wait for the python-1-mz2rf pod to be deleted

```bash
$ oc wait --for=delete pod/python-1-mz2rf
```

2.5.5. Advanced developer CLI commands

2.5.5.1. api-resources
Display the full list of API resources that the server supports.

Example: List the supported API resources

```bash
$ oc api-resources
```
2.5.5.2. api-versions
Display the full list of API versions that the server supports.

Example: List the supported API versions

$ oc api-versions

2.5.5.3. auth
Inspect permissions and reconcile RBAC roles.

Example: Check whether the current user can read pod logs

$ oc auth can-i get pods --subresource=log

Example: Reconcile RBAC roles and permissions from a file

$ oc auth reconcile -f policy.json

2.5.5.4. cluster-info
Display the address of the master and cluster services.

Example: Display cluster information

$ oc cluster-info

2.5.5.5. convert
Convert a YAML or JSON configuration file to a different API version and print to standard output (stdout).

Example: Convert pod.yaml to the latest version

$ oc convert -f pod.yaml

2.5.5.6. extract
Extract the contents of a config map or secret. Each key in the config map or secret is created as a separate file with the name of the key.

Example: Download the contents of the ruby-1-ca config map to the current directory

$ oc extract configmap/ruby-1-ca

Example: Print the contents of the ruby-1-ca config map to stdout

$ oc extract configmap/ruby-1-ca --to=-
2.5.5.7. idle
Idle scalable resources. An idled service will automatically become unidled when it receives traffic or it can be manually unidled using the `oc scale` command.

Example: Idle the `ruby-app` service

```
$ oc idle ruby-app
```

2.5.5.8. image
Manage images in your OpenShift Container Platform cluster.

Example: Copy an image to another tag

```
$ oc image mirror myregistry.com/myimage:latest myregistry.com/myimage:stable
```

2.5.5.9. observe
Observe changes to resources and take action on them.

Example: Observe changes to services

```
$ oc observe services
```

2.5.5.10. patch
Updates one or more fields of an object using strategic merge patch in JSON or YAML format.

Example: Update the `spec.unschedulable` field for node `node1` to true

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

**NOTE**
If you must patch a custom resource definition, you must include the `--type merge` option in the command.

2.5.5.11. policy
Manage authorization policies.

Example: Add the `edit` role to `user1` for the current project

```
$ oc policy add-role-to-user edit user1
```

2.5.5.12. process
Process a template into a list of resources.

Example: Convert `template.json` to a resource list and pass to `oc create`
2.5.5.13. registry

Manage the integrated registry on OpenShift Container Platform.

**Example: Display information about the integrated registry**

```
$ oc registry info
```

2.5.5.14. replace

Modify an existing object based on the contents of the specified configuration file.

**Example: Update a pod using the content in pod.json**

```
$ oc replace -f pod.json
```

2.5.6. Settings CLI commands

2.5.6.1. completion

Output shell completion code for the specified shell.

**Example: Display completion code for Bash**

```
$ oc completion bash
```

2.5.6.2. config

Manage the client configuration files.

**Example: Display the current configuration**

```
$ oc config view
```

**Example: Switch to a different context**

```
$ oc config use-context test-context
```

2.5.6.3. logout

Log out of the current session.

**Example: End the current session**

```
$ oc logout
```

2.5.6.4. whoami
Display information about the current session.

**Example: Display the currently authenticated user**

```
$ oc whoami
```

### 2.5.7. Other developer CLI commands

#### 2.5.7.1. help

Display general help information for the CLI and a list of available commands.

**Example: Display available commands**

```
$ oc help
```

**Example: Display the help for the `new-project` command**

```
$ oc help new-project
```

#### 2.5.7.2. plugin

List the available plug-ins on the user’s `PATH`.

**Example: List available plug-ins**

```
$ oc plugin list
```

#### 2.5.7.3. version

Display the `oc` client and server versions.

**Example: Display version information**

```
$ oc version
```

For cluster administrators, the OpenShift Container Platform server version is also displayed.

### 2.6. OPENSHIFT CLI ADMINISTRATOR COMMANDS

**NOTE**

You must have `cluster-admin` or equivalent permissions to use these administrator commands.

#### 2.6.1. Cluster management CLI commands

#### 2.6.1.1. inspect

- `$ oc whoami`
- `$ oc help`
- `$ oc help new-project`
- `$ oc plugin list`
- `$ oc version`

For cluster administrators, the OpenShift Container Platform server version is also displayed.
Gather debugging information for a particular resource.

**NOTE**
This command is experimental and might change without notice.

Example: Collect debugging data for the OpenShift API server cluster Operator

```
$ oc adm inspect clusteroperator/openshift-apiserver
```

### 2.6.1.2. must-gather

Bulk collect data about the current state of your cluster to debug issues.

**NOTE**
This command is experimental and might change without notice.

Example: Gather debugging information

```
$ oc adm must-gather
```

### 2.6.1.3. top

Show usage statistics of resources on the server.

#### Example: Show CPU and memory usage for pods

```
$ oc adm top pods
```

#### Example: Show usage statistics for images

```
$ oc adm top images
```

### 2.6.2. Node management CLI commands

#### 2.6.2.1. cordon

Mark a node as unschedulable. Manually marking a node as unschedulable blocks any new pods from being scheduled on the node, but does not affect existing pods on the node.

**Example: Mark node1 as unschedulable**

```
$ oc adm cordon node1
```

#### 2.6.2.2. drain

Drain a node in preparation for maintenance.
Example: Drain node1

$ oc adm drain node1

2.6.2.3. node-logs
Display and filter node logs.

Example: Get logs for NetworkManager

$ oc adm node-logs --role master -u NetworkManager.service

2.6.2.4. taint
Update the taints on one or more nodes.

Example: Add a taint to dedicate a node for a set of users

$ oc adm taint nodes node1 dedicated=groupName:NoSchedule

Example: Remove the taints with key dedicated from node node1

$ oc adm taint nodes node1 dedicated-

2.6.2.5. uncordon
Mark a node as schedulable.

Example: Mark node1 as schedulable

$ oc adm uncordon node1

2.6.3. Security and policy CLI commands

2.6.3.1. certificate
Approve or reject certificate signing requests (CSRs).

Example: Approve a CSR

$ oc adm certificate approve csr-sqgzp

2.6.3.2. groups
Manage groups in your cluster.

Example: Create a new group

$ oc adm groups new my-group
2.6.3.3. new-project

Create a new project and specify administrative options.

**Example: Create a new project using a node selector**

```
$ oc adm new-project myproject --node-selector='type=user-node,region=east'
```

2.6.3.4. pod-network

Manage pod networks in the cluster.

**Example: Isolate project1 and project2 from other non-global projects**

```
$ oc adm pod-network isolate-projects project1 project2
```

2.6.3.5. policy

Manage roles and policies on the cluster.

**Example: Add the edit role to user1 for all projects**

```
$ oc adm policy add-cluster-role-to-user edit user1
```

**Example: Add the privileged security context constraint to a service account**

```
$ oc adm policy add-scc-to-user privileged -z myserviceaccount
```

2.6.4. Maintenance CLI commands

2.6.4.1. migrate

Migrate resources on the cluster to a new version or format depending on the subcommand used.

**Example: Perform an update of all stored objects**

```
$ oc adm migrate storage
```

**Example: Perform an update of only pods**

```
$ oc adm migrate storage --include=pods
```

2.6.4.2. prune

Remove older versions of resources from the server.

**Example: Prune older builds including those whose build configs no longer exist**

```
$ oc adm prune builds --orphans
```
2.6.5. Configuration CLI commands

2.6.5.1. create-bootstrap-project-template
Create a bootstrap project template.

Example: Output a bootstrap project template in YAML format to stdout

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

2.6.5.2. create-error-template
Create a template for customizing the error page.

Example: Output a template for the error page to stdout

```
$ oc adm create-error-template
```

2.6.5.3. create-kubeconfig
Creates a basic .kubeconfig file from client certificates.

Example: Create a .kubeconfig file with the provided client certificates

```
$ oc adm create-kubeconfig
    --client-certificate=/path/to/client.crt
    --client-key=/path/to/client.key
    --certificate-authority=/path/to/ca.crt
```

2.6.5.4. create-login-template
Create a template for customizing the login page.

Example: Output a template for the login page to stdout

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

2.6.5.5. create-provider-selection-template
Create a template for customizing the provider selection page.

Example: Output a template for the provider selection page to stdout

```
$ oc adm create-provider-selection-template
```

2.6.6. Other Administrator CLI commands

2.6.6.1. build-chain
Output the inputs and dependencies of any builds.
Example: Output dependencies for the perl imagestream

```
$ oc adm build-chain perl
```

### 2.6.6.2. completion

Output shell completion code for the **oc adm** commands for the specified shell.

Example: Display **oc adm** completion code for Bash

```
$ oc adm completion bash
```

### 2.6.6.3. config

Manage the client configuration files. This command has the same behavior as the **oc config** command.

Example: Display the current configuration

```
$ oc adm config view
```

Example: Switch to a different context

```
$ oc adm config use-context test-context
```

### 2.6.6.4. release

Manage various aspects of the OpenShift Container Platform release process, such as viewing information about a release or inspecting the contents of a release.

Example: Generate a changelog between two releases and save to **changelog.md**

```
$ oc adm release info --changelog=/tmp/git \ 
quay.io/openshift-release-dev/ocp-release:4.6.0-rc.7-x86_64 \ 
quay.io/openshift-release-dev/ocp-release:4.6.4-x86_64 \ 
> changelog.md
```

### 2.6.6.5. verify-image-signature

Verify the image signature of an image imported to the internal registry using the local public GPG key.

Example: Verify the **nodejs** image signature

```
$ oc adm verify-image-signature \ 
  sha256:2bba968aedb7dd2aafe5fa8c7453f5ac36a0b9639f1bf5b03f95de325238b288 \ 
  --expected-identity 172.30.1.1:5000/openshift/nodejs:latest \ 
  --public-key /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release \ 
  --save
```

### 2.7. USAGE OF OC AND KUBECTL COMMANDS
The Kubernetes command-line interface (CLI), `kubectl`, can be used to run commands against a Kubernetes cluster. Because OpenShift Container Platform is a certified Kubernetes distribution, you can use the supported `kubectl` binaries that ship with OpenShift Container Platform, or you can gain extended functionality by using the `oc` binary.

### 2.7.1. The `oc` binary

The `oc` binary offers the same capabilities as the `kubectl` binary, but it extends to natively support additional OpenShift Container Platform features, including:

- **Full support for OpenShift Container Platform resources**
  Resources such as `DeploymentConfig`, `BuildConfig`, `Route`, `ImageStream`, and `ImageStreamTag` objects are specific to OpenShift Container Platform distributions, and build upon standard Kubernetes primitives.

- **Authentication**
  The `oc` binary offers a built-in `login` command that allows authentication and enables you to work with OpenShift Container Platform projects, which map Kubernetes namespaces to authenticated users. See Understanding authentication for more information.

- **Additional commands**
  The additional command `oc new-app`, for example, makes it easier to get new applications started using existing source code or pre-built images. Similarly, the additional command `oc new-project` makes it easier to start a project that you can switch to as your default.

#### IMPORTANT

If you installed an earlier version of the `oc` binary, you cannot use it to complete all of the commands in OpenShift Container Platform 4.6. If you want the latest features, you must download and install the latest version of the `oc` binary corresponding to your OpenShift Container Platform server version.

Non-security API changes will involve, at minimum, two minor releases (4.1 to 4.2 to 4.3, for example) to allow older `oc` binaries to update. Using new capabilities might require newer `oc` binaries. A 4.3 server might have additional capabilities that a 4.2 `oc` binary cannot use and a 4.3 `oc` binary might have additional capabilities that are unsupported by a 4.2 server.

#### Table 2.2. Compatibility Matrix

<table>
<thead>
<tr>
<th>X.Y (Server)</th>
<th>X.Y (oc Client)</th>
<th>X.Y+N footnote:versionpolicyn[Where N is a number greater than or equal to 1.] (oc Client)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X.Y (Server)</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>X.Y+N footnote:versionpolicyn[] (Server)</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

1. Fully compatible.
oc client might be unable to access server features.

oc client might provide options and features that might not be compatible with the accessed server.

2.7.2. The kubectl binary

The `kubectl` binary is provided as a means to support existing workflows and scripts for new OpenShift Container Platform users coming from a standard Kubernetes environment, or for those who prefer to use the `kubectl` CLI. Existing users of `kubectl` can continue to use the binary to interact with Kubernetes primitives, with no changes required to the OpenShift Container Platform cluster.

You can install the supported `kubectl` binary by following the steps to Install the OpenShift CLI. The `kubectl` binary is included in the archive if you download the binary, or is installed when you install the CLI by using an RPM.

For more information, see the `kubectl documentation`. 
CHAPTER 3. DEVELOPER CLI (ODO)

3.1. UNDERSTANDING ODO

Red Hat OpenShift Developer CLI (odo) is a tool for creating applications on OpenShift Container Platform and Kubernetes. With odo, you can develop, test, debug, and deploy microservices-based applications on a Kubernetes cluster without having a deep understanding of the platform.

odo follows a create and push workflow. As a user, when you create, the information (or manifest) is stored in a configuration file. When you push, the corresponding resources are created on the Kubernetes cluster. All of this configuration is stored in the Kubernetes API for seamless accessibility and functionality.

odo uses service and link commands to link components and services together. odo achieves this by creating and deploying services based on Kubernetes Operators in the cluster. Services can be created using any of the Operators available on the Operator Hub. After linking a service, odo injects the service configuration into the component. Your application can then use this configuration to communicate with the Operator-backed service.

3.1.1. odo key features

odo is designed to be a developer-friendly interface to Kubernetes, with the ability to:

- Quickly deploy applications on a Kubernetes cluster by creating a new manifest or using an existing one
- Use commands to easily create and update the manifest, without the need to understand and maintain Kubernetes configuration files
- Provide secure access to applications running on a Kubernetes cluster
- Add and remove additional storage for applications on a Kubernetes cluster
- Create Operator-backed services and link your application to them
- Create a link between multiple microservices that are deployed as odo components
- Remotely debug applications you deployed using odo in your IDE
- Easily test applications deployed on Kubernetes using odo

3.1.2. odo core concepts

odo abstracts Kubernetes concepts into terminology that is familiar to developers:

**Application**
A typical application, developed with a cloud-native approach, that is used to perform a particular task.
Examples of applications include online video streaming, online shopping, and hotel reservation systems.

**Component**
A set of Kubernetes resources that can run and be deployed separately. A cloud-native application is a collection of small, independent, loosely coupled components.
Examples of components include an API back-end, a web interface, and a payment back-end.

**Project**
A single unit containing your source code, tests, and libraries.

**Context**
A directory that contains the source code, tests, libraries, and odo config files for a single component.

**URL**
A mechanism to expose a component for access from outside the cluster.

**Storage**
Persistent storage in the cluster. It persists the data across restarts and component rebuilds.

**Service**
An external application that provides additional functionality to a component. Examples of services include PostgreSQL, MySQL, Redis, and RabbitMQ.

In odo, services are provisioned from the OpenShift Service Catalog and must be enabled within your cluster.

devfile
An open standard for defining containerized development environments that enables developer tools to simplify and accelerate workflows. For more information, see the documentation at https://devfile.io.

You can connect to publicly available devfile registries, or you can install a Secure Registry.

### 3.1.3. Listing components in odo

odo uses the portable devfile format to describe components and their related URLs, storage, and services. odo can connect to various devfile registries to download devfiles for different languages and frameworks. See the documentation for the odo registry command for more information on how to manage the registries used by odo to retrieve devfile information.

You can list all the devfiles available of the different registries with the odo catalog list components command.

**Procedure**

1. Log in to the cluster with odo:

   ```
   $ odo login -u developer -p developer
   ```

2. List the available odo components:

   ```
   $ odo catalog list components
   ```

**Example output**

<table>
<thead>
<tr>
<th>Odo Devfile Components:</th>
<th>DESCRIPTION</th>
<th>REGISTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>dotnet50</td>
<td>Stack with .NET 5.0</td>
<td></td>
</tr>
</tbody>
</table>
3.1.4. Telemetry in odo

*odo* collects information about how it is being used, including metrics on the operating system, RAM, CPU, number of cores, *odo* version, errors, success/failures, and how long *odo* commands take to complete.

You can modify your telemetry consent by using the *odo preference* command:
• *odo preference set ConsentTelemetry true* consents to telemetry.
• *odo preference unset ConsentTelemetry* disables telemetry.
• *odo preference view* shows the current preferences.

### 3.2. INSTALLING ODO

You can install the `odo` CLI on Linux, Windows, or macOS by downloading a binary. You can also install the OpenShift VS Code extension, which uses both the `odo` and the `oc` binaries to interact with your OpenShift Container Platform cluster. For Red Hat Enterprise Linux (RHEL), you can install the `odo` CLI as an RPM.

**NOTE**

Currently, `odo` does not support installation in a restricted network environment.

#### 3.2.1. Installing odo on Linux

The `odo` CLI is available to download as a binary and as a tarball for multiple operating systems and architectures including:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Binary</th>
<th>Tarball</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td><code>odo-linux-amd64</code></td>
<td><code>odo-linux-amd64.tar.gz</code></td>
</tr>
<tr>
<td>Linux on IBM Power</td>
<td><code>odo-linux-ppc64le</code></td>
<td><code>odo-linux-ppc64le.tar.gz</code></td>
</tr>
<tr>
<td>Linux on IBM Z and LinuxONE</td>
<td><code>odo-linux-s390x</code></td>
<td><code>odo-linux-s390x.tar.gz</code></td>
</tr>
</tbody>
</table>

**Procedure**

1. Navigate to the content gateway and download the appropriate file for your operating system and architecture.

   - If you download the binary, rename it to `odo`:
     ```sh
     $ chmod +x odo
     ```

   - If you download the tarball, extract the binary:
     ```sh
     $ curl -L https://developers.redhat.com/content-gateway/rest/mirror/pub/openshift-v4/clients/odo/latest/odo-linux-amd64.tar.gz -o odo.tar.gz
     $ tar xvzf odo.tar.gz
     ```

2. Change the permissions on the binary:

   ```sh
   $ chmod +x <filename>
   ```
3. Place the odo binary in a directory that is on your PATH.
   To check your PATH, execute the following command:

   $ echo $PATH

4. Verify that odo is now available on your system:

   $ odo version

### 3.2.2. Installing odo on Windows

The odo CLI for Windows is available to download as a binary and as an archive.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Binary</th>
<th>Tarball</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>odo-windows-amd64.exe</td>
<td>odo-windows-amd64.exe.zip</td>
</tr>
</tbody>
</table>

**Procedure**

1. Navigate to the content gateway and download the appropriate file:
   - If you download the binary, rename it to odo.exe.
   - If you download the archive, unzip the binary with a ZIP program and then rename it to odo.exe.

2. Move the odo.exe binary to a directory that is on your PATH.
   To check your PATH, open the command prompt and execute the following command:

   C:\> path

3. Verify that odo is now available on your system:

   C:\> odo version

### 3.2.3. Installing odo on macOS

The odo CLI for macOS is available to download as a binary and as a tarball.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Binary</th>
<th>Tarball</th>
</tr>
</thead>
<tbody>
<tr>
<td>macOS</td>
<td>odo-darwin-amd64</td>
<td>odo-darwin-amd64.tar.gz</td>
</tr>
</tbody>
</table>

**Procedure**

1. Navigate to the content gateway and download the appropriate file:
   - If you download the binary, rename it to odo:
3.2.4. Installing odo on VS Code

The OpenShift VS Code extension uses both odo and the oc binary to interact with your OpenShift Container Platform cluster. To work with these features, install the OpenShift VS Code extension on VS Code.

Prerequisites

- You have installed VS Code.

Procedure

1. Open VS Code.

2. Launch VS Code Quick Open with Ctrl+P.

3. Enter the following command:

   $ ext install redhat.vscode-openshift-connector

3.2.5. Installing odo on Red Hat Enterprise Linux (RHEL) using an RPM

For Red Hat Enterprise Linux (RHEL), you can install the odo CLI as an RPM.

Procedure

1. Register with Red Hat Subscription Manager:

   # subscription-manager register
2. Pull the latest subscription data:
   ```
   # subscription-manager refresh
   ```

3. List the available subscriptions:
   ```
   # subscription-manager list --available --matches "^OpenShift Developer Tools and Services$"
   ```

4. In the output of the previous command, find the Pool ID field for your OpenShift Container Platform subscription and attach the subscription to the registered system:
   ```
   # subscription-manager attach --pool=<pool_id>
   ```

5. Enable the repositories required by odo:
   ```
   # subscription-manager repos --enable="ocp-tools-4.9-for-rhel-8-x86_64-rpms"
   ```

6. Install the odo package:
   ```
   # yum install odo
   ```

7. Verify that odo is now available on your system:
   ```
   $ odo version
   ```

### 3.3. CREATING AND DEPLOYING APPLICATIONS WITH ODO

#### 3.3.1. Working with projects

Project keeps your source code, tests, and libraries organized in a separate single unit.

**3.3.1.1. Creating a project**

Create a project to keep your source code, tests, and libraries organized in a separate single unit.

**Procedure**

1. Log in to an OpenShift Container Platform cluster:
   ```
   $ odo login -u developer -p developer
   ```

2. Create a project:
   ```
   $ odo project create myproject
   ```

**Example output**

- Project 'myproject' is ready for use
- New project created and now using project : myproject
3.3.2. Creating a single-component application with `odo`

With `odo`, you can create and deploy applications on clusters.

**Prerequisites**

- `odo` is installed.
- You have a running cluster. You can use CodeReady Containers (CRC) to deploy a local cluster quickly.

### 3.3.2.1. Creating a project

Create a project to keep your source code, tests, and libraries organized in a separate single unit.

**Procedure**

1. Log in to an OpenShift Container Platform cluster:
   
   ```bash
   $ odo login -u developer -p developer
   ```

2. Create a project:
   
   ```bash
   $ odo project create myproject
   ```

**Example output**

- Project `myproject` is ready for use
- New project created and now using project : myproject

### 3.3.2.2. Creating a Node.js application with `odo`

To create a Node.js component, download the Node.js application and push the source code to your cluster with `odo`.

**Procedure**

1. Create a directory for your components:
   
   ```bash
   $ mkdir my_components && cd my_components
   ```

2. Download the example Node.js application:
   
   ```bash
   $ git clone https://github.com/openshift/nodejs-ex
   ```

3. Change the current directory to the directory with your application:
   
   ```bash
   $ cd <directory_name>
   ```

4. Add a component of the type Node.js to your application:
   
   ```bash
   $ odo create nodejs
   ```
NOTE
By default, the latest image is used. You can also explicitly specify an image version by using `odo create openshift/nodejs:8`.

5. Push the initial source code to the component:
   
   ```$ odo push
   Your component is now deployed to OpenShift Container Platform.
   ```

6. Create a URL and add an entry in the local configuration file as follows:
   
   ```$ odo url create --port 8080
   ```

7. Push the changes. This creates a URL on the cluster.
   
   ```$ odo push
   ```

8. List the URLs to check the desired URL for the component.
   
   ```$ odo url list
   ```

9. View your deployed application using the generated URL.
   
   ```$ curl <url>
   ```

3.3.2.3. Modifying your application code
You can modify your application code and have the changes applied to your application on OpenShift Container Platform.

1. Edit one of the layout files within the Node.js directory with your preferred text editor.

2. Update your component:
   
   ```$ odo push
   ```

3. Refresh your application in the browser to see the changes.

3.3.2.4. Adding storage to the application components
Use the `odo storage` command to add persistent data to your application. Examples of data that must persist include database files, dependencies, and build artifacts, such as a `.m2` Maven directory.

Procedure

1. Add the storage to your component:
   
   ```$ odo storage create <storage_name> --path=<path_to_the_directory> --size=<size>
   ```

2. Push the storage to the cluster:
3. Verify that the storage is now attached to your component by listing all storage in the component:

   $ odo storage list

**Example output**

   The component 'nodejs' has the following storage attached:
<table>
<thead>
<tr>
<th>NAME</th>
<th>SIZE</th>
<th>PATH</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>mystorage</td>
<td>1Gi</td>
<td>/data</td>
<td>Pushed</td>
</tr>
</tbody>
</table>

4. Delete the storage from your component:

   $ odo storage delete <storage_name>

   $ odo storage list

**Example output**

   The component 'nodejs' has the following storage attached:
<table>
<thead>
<tr>
<th>NAME</th>
<th>SIZE</th>
<th>PATH</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>mystorage</td>
<td>1Gi</td>
<td>/data</td>
<td>Locally Deleted</td>
</tr>
</tbody>
</table>

5. List all storage to verify that the storage state is **Locally Deleted**:

   $ odo storage list

   $ odo push

6. Push the changes to the cluster:

   $ odo push

### 3.3.2.5. Adding a custom builder to specify a build image

With OpenShift Container Platform, you can add a custom image to bridge the gap between the creation of custom images.

The following example demonstrates the successful import and use of the `redhat-openjdk-18` image:

**Prerequisites**

- The OpenShift CLI (oc) is installed.

**Procedure**

1. Import the image into OpenShift Container Platform:

   $ oc import-image openjdk18
   --from=registry.access.redhat.com/redhat-openjdk-18/openjdk18-openshift
   --confirm

2. Tag the image to make it accessible to odo:
3.3.2.6. Connecting your application to multiple services using OpenShift Service Catalog

The OpenShift service catalog is an implementation of the Open Service Broker API (OSB API) for Kubernetes. You can use it to connect applications deployed in OpenShift Container Platform to a variety of services.

Prerequisites

- You have a running OpenShift Container Platform cluster.
- The service catalog is installed and enabled on your cluster.

Procedure

- To list the services:
  
  $ odo catalog list services

- To use service catalog-related operations:
  
  $ odo service <verb> <service_name>

3.3.2.7. Deleting an application

Use the `odo app delete` command to delete your application.

Procedure

1. List the applications in the current project:

   $ odo app list

   **Example output**

   The project '<project_name>' has the following applications:
   NAME
   app

2. List the components associated with the applications. These components will be deleted with the application:

   $ odo component list

   **Example output**

   -
3. Delete the application:

```
$ odo app delete <application_name>
```

**Example output**

```
? Are you sure you want to delete the application: <application_name> from project: <project_name>
```

4. Confirm the deletion with Y. You can suppress the confirmation prompt using the `-f` flag.

### 3.3.3. Creating a multicomponent application with odo

**odo** allows you to create a multicomponent application, modify it, and link its components in an easy and automated way.

This example describes how to deploy a multicomponent application – a shooter game. The application consists of a front-end Node.js component and a back-end Java component.

**Prerequisites**

- **odo** is installed.
- You have a running cluster. Developers can use CodeReady Containers (CRC) to deploy a local cluster quickly.
- Maven is installed.

#### 3.3.3.1. Creating a project

Create a project to keep your source code, tests, and libraries organized in a separate single unit.

**Procedure**

1. Log in to an OpenShift Container Platform cluster:

```
$ odo login -u developer -p developer
```

2. Create a project:

```
$ odo project create myproject
```

**Example output**

```
✓ Project 'myproject' is ready for use
✓ New project created and now using project : myproject
```

#### 3.3.3.2. Deploying the back-end component
To create a Java component, import the Java builder image, download the Java application and push the source code to your cluster with odo.

Procedure

1. Import openjdk18 into the cluster:
   
   ```
   $ oc import-image openjdk18 \ 
   --from=registry.access.redhat.com/redhat-openjdk-18/openjdk18-openshift --confirm
   ```

2. Tag the image as builder to make it accessible for odo:
   
   ```
   $ oc annotate istag/openjdk18:latest tags=builder
   ```

3. Run odo catalog list components to see the created image:
   
   ```
   $ odo catalog list components
   ```

Example output

| Odo Devfile Components:                      | NAME                 | DESCRIPTION                                      | REGISTRY                      |
|-----------------------------------------------|----------------------|--------------------------------------------------|
| java-maven     Upstream Maven and OpenJDK 11  | DefaultDevfileRegistry |
| java-openliberty  Open Liberty microservice in Java | DefaultDevfileRegistry |
| java-quarkus  Upstream Quarkus with Java+GraalVM | DefaultDevfileRegistry |
| java-springboot  Spring Boot® using Java           | DefaultDevfileRegistry |
| nodejs       Stack with NodeJS 12                  | DefaultDevfileRegistry |

<table>
<thead>
<tr>
<th>Odo OpenShift Components:</th>
<th>NAME</th>
<th>PROJECT</th>
<th>TAGS</th>
<th>SUPPORTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>java                openshift     11,8,latest</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dotnet              openshift     2.1,3.1,latest</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>golang              openshift     1.13.4-ubi7,1.13.4-ubi8,latest</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>httpd               openshift     2.4-el7,2.4-el8,latest</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nginx               openshift     1.14-el7,1.14-el8,1.16-el7,1.16-el8,latest</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodejs              openshift     10-ubi7,10-ubi8,12-ubi7,12-ubi8,latest</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>perl                openshift     5.26-el7,5.26-ubi8,5.30-el7,latest</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>php                 openshift     7.2-ubi7,7.2-ubi8,7.3-ubi7,7.3-ubi8,latest</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>python              openshift     2.7-ubi7,2.7-ubi8,3.6-ubi7,3.6-ubi8,3.8-ubi7,3.8-ubi8,latest</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ruby                openshift     2.5-ubi7,2.5-ubi8,2.6-ubi7,2.6-ubi8,2.7-ubi7,latest</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wildfly             openshift     10.0,10.1,11.0,12.0,13.0,14.0,15.0,16.0,17.0,18.0,19.0,20.0,8.1,9.0,latest</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. Create a directory for your components:
   
   ```
   $ mkdir my_components && cd my_components
   ```

5. Download the example back-end application:
   
   ```
   $ git clone https://github.com/openshift-evangelists/Wild-West-Backend backend
   ```

6. Change to the back-end source directory:
7. Check that you have the correct files in the directory:

```bash
$ ls
```

**Example output**

```
debug.sh  pom.xml  src
```

8. Build the back-end source files with Maven to create a JAR file:

```bash
$ mvn package
```

**Example output**

```
... 
[INFO] --------------------------------------
[INFO] BUILD SUCCESS
[INFO] -------------------------------------- 
[INFO] Total time: 2.635 s 
[INFO] Final Memory: 30M/91M 
[INFO] --------------------------------------
```

9. Create a component configuration of Java component-type named `backend`:

```bash
$ odo create --s2i openjdk18 backend --binary target/wildwest-1.0.jar
```

**Example output**

```
✓ Validating component [1ms]
Please use `odo push` command to create the component with source deployed
```

Now the configuration file `config.yaml` is in the local directory of the back-end component that contains information about the component for deployment.

10. Check the configuration settings of the back-end component in the `config.yaml` file using:

```bash
$ odo config view
```

**Example output**

```
COMPONENT SETTINGS

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>CURRENT_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>openjdk18</td>
</tr>
<tr>
<td>Application</td>
<td>app</td>
</tr>
<tr>
<td>Project</td>
<td>myproject</td>
</tr>
<tr>
<td>SourceType</td>
<td>binary</td>
</tr>
<tr>
<td>Ref</td>
<td></td>
</tr>
<tr>
<td>SourceLocation</td>
<td>target/wildwest-1.0.jar</td>
</tr>
</tbody>
</table>
```
11. Push the component to the OpenShift Container Platform cluster.

```
$ odo push
```

**Example output**

Validation

✓ Checking component [6ms]

Configuration changes

✓ Initializing component
✓ Creating component [124ms]

Pushing to component backend of type binary

✓ Checking files for pushing [1ms]
✓ Waiting for component to start [48s]
✓ Syncing files to the component [811ms]
✓ Building component [3s]

Using odo push, OpenShift Container Platform creates a container to host the back-end component, deploys the container into a pod running on the OpenShift Container Platform cluster, and starts the **backend** component.

12. Validate:

- The status of the action in odo:

```
$ odo log -f
```

**Example output**

```
```

- The status of the back-end component:

```
$ odo list
```

**Example output**

```
APP   NAME        TYPE          SOURCE                     STATE
app   backend     openjdk18    file://target/wildwest-1.0.jar  Pushed
```
3.3.3.3. Deploying the front-end component

To create and deploy a front-end component, download the Node.js application and push the source code to your cluster with `odo`.

**Procedure**

1. Download the example front-end application:
   
   ```
   $ git clone https://github.com/openshift/nodejs-ex frontend
   ```

2. Change the current directory to the front-end directory:
   
   ```
   $ cd frontend
   ```

3. List the contents of the directory to see that the front end is a Node.js application.
   
   ```
   $ ls
   ```

**Example output**

```
README.md openshift server.js views
helm package.json tests
```

**NOTE**

The front-end component is written in an interpreted language (Node.js); it does not need to be built.

4. Create a component configuration of Node.js component-type named `frontend`:
   
   ```
   $ odo create --s2i nodejs frontend
   ```

**Example output**

```
✓ Validating component [5ms]
Please use `odo push` command to create the component with source deployed
```

5. Push the component to a running container.
   
   ```
   $ odo push
   ```

**Example output**

```
Validation
✓ Checking component [8ms]

Configuration changes
✓ Initializing component
✓ Creating component [83ms]
```
3.3.3.4. Linking both components

Components running on the cluster need to be connected in order to interact. OpenShift Container Platform provides linking mechanisms to publish communication bindings from a program to its clients.

Procedure

1. List all the components that are running on the cluster:
   
   $ odo list
   
   Example output
   
   OpenShift Components:
   APP    NAME     PROJECT     TYPE         SOURCETYPE     STATE
   app    backend  testpro    openjdk18    binary        Pushed
   app    frontend testpro    nodejs       local         Pushed

2. Link the current front-end component to the back end:
   
   $ odo link backend --port 8080
   
   Example output
   
   ✓ Component backend has been successfully linked from the component frontend
   Following environment variables were added to frontend component:
   - COMPONENT_BACKEND_HOST
   - COMPONENT_BACKEND_PORT
   
   The configuration information of the back-end component is added to the front-end component and the front-end component restarts.

3.3.3.5. Exposing components to the public

Procedure

1. Navigate to the frontend directory:
   
   $ cd frontend

2. Create an external URL for the application:
   
   $ odo url create frontend --port 8080
Example output

✓ URL frontend created for component: frontend

To create URL on the OpenShift cluster, use `odo push`

3. Apply the changes:

   $ odo push

Example output

Validation
✓ Checking component [21ms]

Configuration changes
✓ Retrieving component data [35ms]
✓ Applying configuration [29ms]

Applying URL changes
✓ URL frontend: http://frontend-app-myproject.192.168.42.79.nip.io created

Pushing to component frontend of type local
✓ Checking file changes for pushing [1ms]
✓ No file changes detected, skipping build. Use the `-f` flag to force the build.

4. Open the URL in a browser to view the application.

NOTE

If an application requires permissions to the active service account to access the OpenShift Container Platform namespace and delete active pods, the following error may occur when looking at `odo log` from the back-end component:

**Message: Forbidden!Configured service account doesn’t have access. Service account may have been revoked**

To resolve this error, add permissions for the service account role:

   $ oc policy add-role-to-group view system:serviceaccounts -n <project>

   $ oc policy add-role-to-group edit system:serviceaccounts -n <project>

Do not do this on a production cluster.

3.3.3.6. Modifying the running application

Procedure

1. Change the local directory to the front-end directory:

   $ cd frontend

   OpenShift Container Platform 4.6 CLI tools

66
2. Monitor the changes on the file system using:

```
$ odo watch
```

3. Edit the `index.html` file to change the displayed name for the game.

NOTE

A slight delay is possible before odo recognizes the change.

odo pushes the changes to the front-end component and prints its status to the terminal:

```
File /root/frontend/index.html changed
File  changed
Pushing files...
  ✓ Waiting for component to start
  ✓ Copying files to component
  ✓ Building component
```

4. Refresh the application page in the web browser. The new name is now displayed.

3.3.3.7. Deleting an application

Use the `odo app delete` command to delete your application.

Procedure

1. List the applications in the current project:

```
$ odo app list
```

**Example output**

```
The project `<project_name>` has the following applications:
NAME
app
```

2. List the components associated with the applications. These components will be deleted with the application:

```
$ odo component list
```

**Example output**

```
APP NAME TYPE SOURCE STATE
app nodejs-nodejs-ex-elyf nodejs file://./ Pushed
```

3. Delete the application:

```
$ odo app delete <application_name>
```
Example output

$ odo login -u developer -p developer

✓ Project 'myproject' is ready for use
✓ New project created and now using project : myproject

3.3.4. Creating an application with a database

This example describes how to deploy and connect a database to a front-end application.

Prerequisites

- odo is installed.
- oc client is installed.
- You have a running cluster. Developers can use CodeReady Containers (CRC) to deploy a local cluster quickly.
- The Service Catalog is installed and enabled on your cluster.

NOTE

Service Catalog is deprecated on OpenShift Container Platform 4 and later.

3.3.4.1. Creating a project

Create a project to keep your source code, tests, and libraries organized in a separate single unit.

Procedure

1. Log in to an OpenShift Container Platform cluster:

   $ odo login -u developer -p developer

2. Create a project:

   $ odo project create myproject

Example output

✓ Project 'myproject' is ready for use
✓ New project created and now using project : myproject

3.3.4.2. Deploying the front-end component

To create and deploy a front-end component, download the Node.js application and push the source code to your cluster with odo.

Procedure

? Are you sure you want to delete the application: <application_name> from project: <project_name>

4. Confirm the deletion with Y. You can suppress the confirmation prompt using the -f flag.
1. Download the example front-end application:
   
   $ git clone https://github.com/openshift/nodejs-ex frontend

2. Change the current directory to the front-end directory:
   
   $ cd frontend

3. List the contents of the directory to see that the front end is a Node.js application.
   
   $ ls

   **Example output**
   
   README.md  openshift  server.js  views
   helm        package.json  tests

   **NOTE**
   
   The front-end component is written in an interpreted language (Node.js); it does not need to be built.

4. Create a component configuration of Node.js component-type named `frontend`:
   
   $ odo create --s2i nodejs frontend

   **Example output**
   
   ✓ Validating component [5ms]
   Please use `odo push` command to create the component with source deployed

5. Create a URL to access the frontend interface.
   
   $ odo url create myurl

   **Example output**
   
   ✓ URL myurl created for component: nodejs-nodejs-ex-pmdp

6. Push the component to the OpenShift Container Platform cluster.
   
   $ odo push

   **Example output**
   
   Validation
   ✓ Checking component [7ms]

   Configuration changes
   ✓ Initializing component
3.3.4.3. Deploying a database in interactive mode

odo provides a command-line interactive mode which simplifies deployment.

Procedure

- Run the interactive mode and answer the prompts:

  ```
  $ odo service create
  ```

Example output

? Which kind of service do you wish to create database
? Which database service class should we use mongodb-persistent
? Enter a value for string property DATABASE_SERVICE_NAME (Database Service Name): mongodb
? Enter a value for string property MEMORY_LIMIT (Memory Limit): 512Mi
? Enter a value for string property MONGODB_DATABASE (MongoDB Database Name): sampledb
? Enter a value for string property MONGODB_VERSION (Version of MongoDB Image): 3.2
? Enter a value for string property VOLUME_CAPACITY (Volume Capacity): 1Gi
? Provide values for non-required properties No
? How should we name your service mongodb-persistent No
? Wait for the service to be ready No
  ✓ Creating service [32ms]
  ✓ Service 'mongodb-persistent' was created
Progress of the provisioning will not be reported and might take a long time.
You can see the current status by executing 'odo service list'

NOTE

Your password or username will be passed to the front-end application as environment variables.

3.3.4.4. Deploying a database manually

1. List the available services:

  ```
  $ odo catalog list services
  ```
Example output

<table>
<thead>
<tr>
<th>NAME</th>
<th>PLANS</th>
</tr>
</thead>
<tbody>
<tr>
<td>django-psql-persistent</td>
<td>default</td>
</tr>
<tr>
<td>jenkins-ephemeral</td>
<td>default</td>
</tr>
<tr>
<td>jenkins-pipeline-example</td>
<td>default</td>
</tr>
<tr>
<td>mariadb-persistent</td>
<td>default</td>
</tr>
<tr>
<td>mongodb-persistent</td>
<td>default</td>
</tr>
<tr>
<td>mysql-persistent</td>
<td>default</td>
</tr>
<tr>
<td>nodejs-mongo-persistent</td>
<td>default</td>
</tr>
<tr>
<td>postgresql-persistent</td>
<td>default</td>
</tr>
<tr>
<td>rails-pgsql-persistent</td>
<td>default</td>
</tr>
</tbody>
</table>

2. Choose the **mongodb-persistent** type of service and see the required parameters:

```bash
$ odo catalog describe service mongodb-persistent
```

Example output

<table>
<thead>
<tr>
<th>Name</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Name</td>
<td></td>
</tr>
<tr>
<td>Short Description</td>
<td>Default plan</td>
</tr>
<tr>
<td>Required Params without a default value</td>
<td>DATABASE_SERVICE_NAME</td>
</tr>
<tr>
<td>Required Params with a default value</td>
<td>MEMORY_LIMIT (default: '512Mi'), MONGODB_VERSION (default: '3.2'), MONGODB_DATABASE (default: 'sampledb'), VOLUME_CAPACITY (default: '1Gi')</td>
</tr>
<tr>
<td>Optional Params</td>
<td>MONGODB_ADMIN_PASSWORD, *</td>
</tr>
</tbody>
</table>

3. Pass the required parameters as flags and wait for the deployment of the database:

```bash
$ odo service create mongodb-persistent --plan default --wait -p DATABASE_SERVICE_NAME=mongodb -p MEMORY_LIMIT=512Mi -p MONGODB_DATABASE=sampledb -p VOLUME_CAPACITY=1Gi
```

3.3.4.5. Connecting the database to the front-end application

1. Link the database to the front-end service:

```bash
$ odo link mongodb-persistent
```
Example output

✓ Service mongodb-persistent has been successfully linked from the component nodejs-nodejs-ex-mhbb

Following environment variables were added to nodejs-nodejs-ex-mhbb component:
- database_name
- password
- uri
- username
- admin_password

2. See the environment variables of the application and the database in the pod:
   a. Get the pod name:

      $ oc get pods

      Example output

      | NAME                  | READY | STATUS   | RESTARTS | AGE  |
      |-----------------------|-------|----------|----------|------|
      | mongodb-1-gsznc        | 1/1   | Running  | 0        | 28m  |
      | nodejs-nodejs-ex-mhbb-app-4-vkn9l | 1/1   | Running  | 0        | 1m   |

   b. Connect to the pod:

      $ oc rsh nodejs-nodejs-ex-mhbb-app-4-vkn9l

   c. Check the environment variables:

      sh-4.2$ env

      Example output

      uri=mongodb://172.30.126.3:27017
      password=dHIOpYneSkX3rTLn
      database_name=sampledb
      username=user43U
      admin_password=NCn41tqmx7Rlqmfv

3. Open the URL in the browser and notice the database configuration in the bottom right:

   $ odo url list

   Example output

   Request information
   Page view count: 24

   DB Connection Info:
   Type: MongoDB
   URL: mongodb://172.30.126.3:27017/sampledb
3.3.5. Using devfiles in odo

3.3.5.1. About the devfile in odo

The devfile is a portable file that describes your development environment. With the devfile, you can define a portable developmental environment without the need for reconfiguration.

With the devfile, you can describe your development environment, such as the source code, IDE tools, application runtimes, and predefined commands. To learn more about the devfile, see the devfile documentation.

With odo, you can create components from the devfiles. When creating a component by using a devfile, odo transforms the devfile into a workspace consisting of multiple containers that run on OpenShift Container Platform, Kubernetes, or Docker. odo automatically uses the default devfile registry but users can add their own registries.

3.3.5.2. Creating a Java application by using a devfile

Prerequisites

- You have installed odo.
- You must know your ingress domain cluster name. Contact your cluster administrator if you do not know it. For example, apps-crc.testing is the cluster domain name for Red Hat CodeReady Containers.

3.3.5.2.1. Creating a project

Create a project to keep your source code, tests, and libraries organized in a separate single unit.

Procedure

1. Log in to an OpenShift Container Platform cluster:

   $ odo login -u developer -p developer

2. Create a project:

   $ odo project create myproject

   Example output

   ✓ Project ‘myproject’ is ready for use
   ✓ New project created and now using project : myproject

3.3.5.2.2. Listing available devfile components

With odo, you can display all the components that are available for you on the cluster. Components that are available depend on the configuration of your cluster.

Procedure

1. To list available devfile components on your cluster, run:
3.3.5.2.3. Deploying a Java application using a devfile

In this section, you will learn how to deploy a sample Java project that uses Maven and Java 8 JDK using a devfile.

**Procedure**

1. Create a directory to store the source code of your component:

   ```
   $ mkdir <directory-name>
   ```

2. Create a component configuration of Spring Boot component type named `myspring` and download its sample project:

   ```
   $ odo create java-spring-boot myspring --starter
   ```

   The previous command produces the following output:

   **Validation**
   - ✓ Checking devfile compatibility [195728ns]
   - ✓ Creating a devfile component from registry: DefaultDevfileRegistry [170275ns]
   - ✓ Validating devfile component [281940ns]

   Please use `odo push` command to create the component with source deployed
The odo create command downloads the associated devfile.yaml file from the recorded devfile registries.

3. List the contents of the directory to confirm that the devfile and the sample Java application were downloaded:

```
$ ls
```

The previous command produces the following output:

```
README.md  devfile.yaml  pom.xml  src
```

4. Create a URL to access the deployed component:

```
$ odo url create --host apps-crc.testing
```

The previous command produces the following output:

```
✓ URL myspring-8080.apps-crc.testing created for component: myspring
```

To apply the URL configuration changes, please use odo push

```

NOTE
You must use your cluster host domain name when creating the URL.
```

5. Push the component to the cluster:

```
$ odo push
```

The previous command produces the following output:

```
Validation
✓ Validating the devfile [81808ns]

Creating Kubernetes resources for component myspring
✓ Waiting for component to start [5s]

Applying URL changes
✓ URL myspring-8080: http://myspring-8080.apps-crc.testing created

Syncing to component myspring
✓ Checking files for pushing [2ms]
✓ Syncing files to the component [1s]

Executing devfile commands for component myspring
✓ Executing devbuild command "/artifacts/bin/build-container-full.sh" [1m]
✓ Executing devrun command "/artifacts/bin/start-server.sh" [2s]

Pushing devfile component myspring
✓ Changes successfully pushed to component
```

6. List the URLs of the component to verify that the component was pushed successfully:
The previous command produces the following output:

```yaml
NAME       URL                                      PORT   SECURE
myspring-8080 http://myspring-8080.apps-crc.testing 8080   false
```

7. View your deployed application by using the generated URL:

```
$ curl http://myspring-8080.apps-crc.testing
```

### 3.3.5.3. Converting an S2I component into a devfile component

With `odo`, you can create both Source-to-Image (S2I) and devfile components. If you have an existing S2I component, you can convert it into a devfile component using the `odo utils` command.

**Procedure**

Run all the commands from the S2I component directory.

1. Run the `odo utils convert-to-devfile` command, which creates `devfile.yaml` and `env.yaml` based on your component:

```bash
$ odo utils convert-to-devfile
```

2. Push the component to your cluster:

```bash
$ odo push
```

**NOTE**

If the devfile component deployment failed, delete it by running: `odo delete -a`

3. Verify that the devfile component deployed successfully:

```bash
$ odo list
```

4. Delete the S2I component:

```bash
$ odo delete --s2i
```

### 3.3.6. Working with storage

Persistent storage keeps data available between restarts of `odo`.

#### 3.3.6.1. Adding storage to the application components

Use the `odo storage` command to add persistent data to your application. Examples of data that must persist include database files, dependencies, and build artifacts, such as a `.m2` Maven directory.
Procedure

1. Add the storage to your component:

   ```
   $ odo storage create <storage_name> --path=<path_to_the_directory> --size=<size>
   ```

2. Push the storage to the cluster:

   ```
   $ odo push
   ```

3. Verify that the storage is now attached to your component by listing all storage in the component:

   ```
   $ odo storage list
   ```

   **Example output**

   The component 'nodejs' has the following storage attached:

<table>
<thead>
<tr>
<th>NAME</th>
<th>SIZE</th>
<th>PATH</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>mystorage</td>
<td>1Gi</td>
<td>/data</td>
<td>Pushed</td>
</tr>
</tbody>
</table>

4. Delete the storage from your component:

   ```
   $ odo storage delete <storage_name>
   ```

5. List all storage to verify that the storage state is **Locally Deleted**:

   ```
   $ odo storage list
   ```

   **Example output**

   The component 'nodejs' has the following storage attached:

<table>
<thead>
<tr>
<th>NAME</th>
<th>SIZE</th>
<th>PATH</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>mystorage</td>
<td>1Gi</td>
<td>/data</td>
<td>Locally Deleted</td>
</tr>
</tbody>
</table>

6. Push the changes to the cluster:

   ```
   $ odo push
   ```

3.3.7. Deleting applications

You can delete applications and all components associated with the application in your project.

3.3.7.1. Deleting an application

Use the `odo app delete` command to delete your application.

Procedure

1. List the applications in the current project:
2. List the components associated with the applications. These components will be deleted with the application:

```bash
$ odo component list
```

**Example output**

```
<table>
<thead>
<tr>
<th>APP</th>
<th>NAME</th>
<th>TYPE</th>
<th>SOURCE</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>app</td>
<td>nodejs-nodejs-ex-elyf</td>
<td>nodejs</td>
<td>file://./</td>
<td>Pushed</td>
</tr>
</tbody>
</table>
```

3. Delete the application:

```bash
$ odo app delete <application_name>
```

**Example output**

```
? Are you sure you want to delete the application: <application_name> from project: <project_name>
```

4. Confirm the deletion with Y. You can suppress the confirmation prompt using the -f flag.

### 3.3.8. Debugging applications in odo

With **odo**, you can attach a debugger to remotely debug your application. This feature is only supported for NodeJS and Java components.

Components created with **odo** run in the debug mode by default. A debugger agent runs on the component, on a specific port. To start debugging your application, you must start port forwarding and attach the local debugger bundled in your Integrated development environment (IDE).

#### 3.3.8.1. Debugging an application

You can debug your application in **odo** with the **odo debug** command.

**Procedure**

1. Download the sample application that contains the necessary **debugrun** step within its devfile:

```bash
$ odo create nodejs --starter
```

**Example output**

```
Validation
```
2. Push the application with the \texttt{--debug} flag, which is required for all debugging deployments:

\texttt{$odo \text{ push --debug}$}

\textbf{Example output}

- Validation
  ✓ Validating the devfile [29916ns]

- Creating Kubernetes resources for component nodejs
  ✓ Waiting for component to start [38ms]

- Applying URL changes
  ✓ URLs are synced with the cluster, no changes are required.

- Syncing to component nodejs
  ✓ Checking file changes for pushing [1ms]
  ✓ Syncing files to the component [778ms]

- Executing devfile commands for component nodejs
  ✓ Executing install command "npm install" [2s]
  ✓ Executing debug command "npm run debug" [1s]

- Pushing devfile component nodejs
  ✓ Changes successfully pushed to component

\textbf{NOTE}

You can specify a custom debug command by using the \texttt{--debug-command="custom-step"} flag.

3. Port forward to the local port to access the debugging interface:

\texttt{$odo \text{ debug port-forward}$}

\textbf{Example output}

- Started port forwarding at ports - 5858:5858
NOTE
You can specify a port by using the `--local-port` flag.

4. Check that the debug session is running in a separate terminal window:

```bash
$ odo debug info
```

**Example output**

Debug is running for the component on the local port: 5858

5. Attach the debugger that is bundled in your IDE of choice. Instructions vary depending on your IDE, for example: **VSCode debugging interface**.

### 3.3.8.2. Configuring debugging parameters

You can specify a remote port with `odo config` command and a local port with the `odo debug` command.

**Procedure**

- To set a remote port on which the debugging agent should run, run:

  ```bash
  $ odo config set DebugPort 9292
  ```

  **NOTE**
  You must redeploy your component for this value to be reflected on the component.

- To set a local port to port forward, run:

  ```bash
  $ odo debug port-forward --local-port 9292
  ```

  **NOTE**
  The local port value does not persist. You must provide it every time you need to change the port.

### 3.3.9. Sample applications

*odo* offers partial compatibility with any language or runtime listed within the OpenShift catalog of component types. For example:

```bash
$ odo debug info
```

**Example output**

Debug is running for the component on the local port: 5858

**NAME** | **PROJECT** | **TAGS**
---|---|---
```
dotnet | openshift | 2.0,latest
http | openshift | 2.4,latest
java | openshift | 8,latest
nginx | openshift | 1.10,1.12,1.8,latest
nodejs | openshift | 0.10,4,6,8,latest
```
For odo Java and Node.js are the officially supported component types. Run odo catalog list components to verify the officially supported component types.

In order to access the component over the web, create a URL using odo url create.

3.3.9.1. Git repository example applications

Use the following commands to build and run sample applications from a Git repository for a particular runtime.

3.3.9.1.1. httpd

This example helps build and serve static content using httpd on CentOS 7. For more information about using this builder image, including OpenShift Container Platform considerations, see the Apache HTTP Server container image repository.

```bash
$ odo create httpd --git https://github.com/openshift/httpd-ex.git
```

3.3.9.1.2. java

This example helps build and run fat JAR Java applications on CentOS 7. For more information about using this builder image, including OpenShift Container Platform considerations, see the Java S2I Builder image.

```bash
$ odo create java --git https://github.com/spring-projects/spring-petclinic.git
```

3.3.9.1.3. nodejs

Build and run Node.js applications on CentOS 7. For more information about using this builder image, including OpenShift Container Platform considerations, see the Node.js 8 container image.

```bash
$ odo create nodejs --git https://github.com/openshift/nodejs-ex.git
```

3.3.9.1.4. perl

This example helps build and run Perl applications on CentOS 7. For more information about using this builder image, including OpenShift Container Platform considerations, see the Perl 5.26 container image.

```bash
$ odo create perl --git https://github.com/openshift/dancer-ex.git
```

3.3.9.1.5. php
This example helps build and run PHP applications on CentOS 7. For more information about using this builder image, including OpenShift Container Platform considerations, see the PHP 7.1 Docker image.

$ odo create php --git https://github.com/openshift/cakephp-ex.git

3.3.9.1.6. python

This example helps build and run Python applications on CentOS 7. For more information about using this builder image, including OpenShift Container Platform considerations, see the Python 3.6 container image.

$ odo create python --git https://github.com/openshift/django-ex.git

3.3.9.1.7. ruby

This example helps build and run Ruby applications on CentOS 7. For more information about using this builder image, including OpenShift Container Platform considerations, see Ruby 2.5 container image.

$ odo create ruby --git https://github.com/openshift/ruby-ex.git

3.3.9.2. Binary example applications

Use the following commands to build and run sample applications from a binary file for a particular runtime.

3.3.9.2.1. java

Java can be used to deploy a binary artifact as follows:

$ git clone https://github.com/spring-projects/spring-petclinic.git
$ cd spring-petclinic
$ mvn package
$ odo create java test3 --binary target/*.jar
$ odo push

3.4. USING ODO IN A RESTRICTED ENVIRONMENT

3.4.1. About odo in a restricted environment

To run odo in a disconnected cluster or a cluster provisioned in a restricted environment, you must ensure that a cluster administrator has created a cluster with a mirrored registry.

To start working in a disconnected cluster, you must first push the odo init image to the registry of the cluster and then overwrite the odo init image path using the ODO_BOOTSTRAPPER_IMAGE environment variable.

After you push the odo init image, you must mirror a supported builder image from the registry, overwrite a mirror registry and then create your application. A builder image is necessary to configure a runtime environment for your application and also contains the build tool needed to build your application, for example npm for Node.js or Maven for Java. A mirror registry contains all the necessary dependencies for your application.
Additional resources

- Mirroring images for a disconnected installation
- Accessing the registry

3.4.2. Pushing the odo init image to the restricted cluster registry

Depending on the configuration of your cluster and your operating system you can either push the odo init image to a mirror registry or directly to an internal registry.

3.4.2.1. Prerequisites

- Install oc on the client operating system.
- Install odo on the client operating system.
- Access to a restricted cluster with a configured internal registry or a mirror registry.

3.4.2.2. Pushing the odo init image to a mirror registry

Depending on your operating system, you can push the odo init image to a cluster with a mirror registry as follows:

3.4.2.2.1. Pushing the init image to a mirror registry on Linux

Procedure

1. Use base64 to encode the root certification authority (CA) content of your mirror registry:

   $ echo <content_of_additional_ca> | base64 --decode > disconnect-ca.crt

2. Copy the encoded root CA certificate to the appropriate location:

   $ sudo cp ./disconnect-ca.crt /etc/pki/ca-trust/source/anchors/<mirror-registry>.crt

3. Trust a CA in your client platform and log in to the OpenShift Container Platform mirror registry:

   $ sudo update-ca-trust enable && sudo systemctl daemon-reload && sudo systemctl restart /docker && sudo docker login <mirror-registry>:5000 -u <username> -p <password>

4. Mirror the odo init image:

   $ oc image mirror registry.access.redhat.com/openshiftdo/odo-init-image-rhel7:<tag> <mirror-registry>:5000/openshiftdo/odo-init-image-rhel7:<tag>

5. Override the default odo init image path by setting the ODO_BOOTSTRAPPER_IMAGE environment variable:

   $ export ODO_BOOTSTRAPPER_IMAGE=<mirror-registry>:5000/openshiftdo/odo-init-image-rhel7:<tag>
3.4.2.2.2. Pushing the init image to a mirror registry on MacOS

Procedure

1. Use `base64` to encode the root certification authority (CA) content of your mirror registry:

   ```shell
   $ echo <content_of_additional_ca> | base64 --decode > disconnect-ca.crt
   ```

2. Copy the encoded root CA certificate to the appropriate location:

   a. Restart Docker using the Docker UI.

   b. Run the following command:

   ```shell
   $ docker login <mirror-registry>:5000 -u <username> -p <password>
   ```

3. Mirror the `odo` init image:

   ```shell
   $ oc image mirror registry.access.redhat.com/openshiftdo/odo-init-image-rhel7:<tag> <mirror-registry>:5000/openshiftdo/odo-init-image-rhel7:<tag>
   ```

4. Override the default `odo` init image path by setting the `ODO_BOOTSTRAPPER_IMAGE` environment variable:

   ```shell
   $ export ODO_BOOTSTRAPPER_IMAGE=<mirror-registry>:5000/openshiftdo/odo-init-image-rhel7:<tag>
   ```

3.4.2.2.3. Pushing the init image to a mirror registry on Windows

Procedure

1. Use `base64` to encode the root certification authority (CA) content of your mirror registry:

   ```powershell
   PS C:\> echo <content_of_additional_ca> | base64 --decode > disconnect-ca.crt
   ```

2. As an administrator, copy the encoded root CA certificate to the appropriate location by executing the following command:

   ```powershell
   PS C:\WINDOWS\system32> certutil -addstore -f "ROOT" disconnect-ca.crt
   ```

3. Trust a CA in your client platform and log in to the OpenShift Container Platform mirror registry:

   a. Restart Docker using the Docker UI.

   b. Run the following command:

   ```powershell
   PS C:\WINDOWS\system32> docker login <mirror-registry>:5000 -u <username> -p <password>
   ```

4. Mirror the `odo` init image:

   ```powershell
   PS C:\> oc image mirror registry.access.redhat.com/openshiftdo/odo-init-image-rhel7:<tag> <mirror-registry>:5000/openshiftdo/odo-init-image-rhel7:<tag>
   ```
5. Override the default `odo` init image path by setting the `ODO_BOOTSTRAPPER_IMAGE` environment variable:

```
PS C:\> $env:ODO_BOOTSTRAPPER_IMAGE="<mirror-registry>:5000/openshiftdo/odo-init-image-rhel7:<tag>"
```

### 3.4.2.3. Pushing the `odo` init image to an internal registry directly

If your cluster allows images to be pushed to the internal registry directly, push the `odo` init image to the registry as follows:

#### 3.4.2.3.1. Pushing the init image directly on Linux

**Procedure**

1. Enable the default route:

```
$ oc patch configs.imageregistry.operator.openshift.io cluster -p '{"spec":
{"defaultRoute":true}}' --type='merge' -n openshift-image-registry
```

2. Get a wildcard route CA:

```
$ oc get secret router-certs-default -n openshift-ingress -o yaml
```

**Example output**

```
apiVersion: v1
data:
tls.crt: "*************
tls.key: "*************
kind: Secret
metadata:
[...]
type: kubernetes.io/tls
```

3. Use `base64` to encode the root certification authority (CA) content of your mirror registry:

```
$ echo <tls.crt> | base64 --decode > ca.crt
```

4. Trust a CA in your client platform:

```
$ sudo cp ca.crt /etc/pki/ca-trust/source/anchors/externalroute.crt && sudo update-ca-trust enable && sudo systemctl daemon-reload && sudo systemctl restart docker
```

5. Log in to the internal registry:

```
$ oc get route -n openshift-image-registry
NAME     HOST/PORT     PATH   SERVICES     PORT  TERMINATION   WILDCARD
default-route <registry_path> image-registry <all> reencrypt None
```

```
$ docker login <registry_path> -u kubeadmin -p $(oc whoami -t)
```
6. Push the `odo` init image:

   ```
   $ docker pull registry.access.redhat.com/openshiftdo/odo-init-image-rhel7:<tag>
   $ docker tag registry.access.redhat.com/openshiftdo/odo-init-image-rhel7:<tag> <registry_path>/openshiftdo/odo-init-image-rhel7:<tag>
   $ docker push <registry_path>/openshiftdo/odo-init-image-rhel7:<tag>
   ```

7. Override the default `odo` init image path by setting the `ODO_BOOTSTRAPPER_IMAGE` environment variable:

   ```
   $ export ODO_BOOTSTRAPPER_IMAGE=<registry_path>/openshiftdo/odo-init-image-rhel7:1.0.1
   ```

### 3.4.2.3.2. Pushing the init image directly on MacOS

**Procedure**

1. Enable the default route:

   ```
   $ oc patch configs.imageregistry.operator.openshift.io cluster -p '{"spec": {"defaultRoute":true}}' --type=merge -n openshift-image-registry
   ```

2. Get a wildcard route CA:

   ```
   $ oc get secret router-certs-default -n openshift-ingress -o yaml
   ```

   **Example output**

   ```
   apiVersion: v1
data:
tls.crt: *************************
tls.key: ##################
kind: Secret
metadata:
[...]
type: kubernetes.io/tls
   ```

3. Use `base64` to encode the root certification authority (CA) content of your mirror registry:

   ```
   $ echo <tls.crt> | base64 --decode > ca.crt
   ```

4. Trust a CA in your client platform:

   ```
   $ sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.keychain ca.crt
   ```

5. Log in to the internal registry:

   ```
   $ oc get route -n openshift-image-registry
   NAME     HOST/PORT  PATH   SERVICES     PORT  TERMINATION   WILDCARD
   default-route <registry_path> image-registry <all> reencrypt None
   ```
6. Push the odo init image:

   $ docker pull registry.access.redhat.com/openshiftdo/odo-init-image-rhel7:<tag>
   $ docker tag registry.access.redhat.com/openshiftdo/odo-init-image-rhel7:<tag> <registry_path>/openshiftdo/odo-init-image-rhel7:<tag>
   $ docker push <registry_path>/openshiftdo/odo-init-image-rhel7:<tag>

7. Override the default odo init image path by setting the ODO_BOOTSTRAPPER_IMAGE environment variable:

   $ export ODO_BOOTSTRAPPER_IMAGE=<registry_path>/openshiftdo/odo-init-image-rhel7:1.0.1

3.4.2.3.3. Pushing the init image directly on Windows

**Procedure**

1. Enable the default route:

   PS C:\> oc patch configs.imageregistry.operator.openshift.io cluster -p '{"spec":
   {"defaultRoute":true}}' --type=merge -n openshift-image-registry

2. Get a wildcard route CA:

   PS C:\> oc get secret router-certs-default -n openshift-ingress -o yaml

**Example output**

   apiVersion: v1
   data:
     tls.crt: ************************
     tls.key: ##################
   kind: Secret
   metadata:
     [...]  
   type: kubernetes.io/tls

3. Use base64 to encode the root certification authority (CA) content of your mirror registry:

   PS C:\> echo <tls.crt> | base64 --decode > ca.crt

4. As an administrator, trust a CA in your client platform by executing the following command:

   PS C:\WINDOWS\system32> certutil -addstore -f "ROOT" ca.crt

5. Log in to the internal registry:
3.4.3. Creating and deploying a component to the disconnected cluster

After you push the init image to a cluster with a mirrored registry, you must mirror a supported builder image for your application with the oc tool, overwrite the mirror registry using the environment variable, and then create your component.

3.4.3.1. Prerequisites

- Install oc on the client operating system.
- Install odo on the client operating system.
- Access to an restricted cluster with a configured internal registry or a mirror registry.
- Push the odo init image to your cluster registry.

3.4.3.2. Mirroring a supported builder image

To use npm packages for Node.js dependencies and Maven packages for Java dependencies and configure a runtime environment for your application, you must mirror a respective builder image from the mirror registry.

Procedure

1. Verify that the required images tag is not imported:

   ```
   $ oc describe is nodejs -n openshift
   ```

   **Example output**

   ```
   Name: nodejs
   Namespace: openshift
   ```
2. Mirror the supported image tag to the private registry:

```
$ oc image mirror registry.access.redhat.com/rhscl/nodejs-10-rhel7:<tag>
<private_registry>/rhscl/nodejs-10-rhel7:<tag>
```

3. Import the image:

```
$ oc tag <mirror-registry>:<port>/rhscl/nodejs-10-rhel7:<tag> nodejs-10-rhel7:latest --scheduled
```

You must periodically re-import the image. The `--scheduled` flag enables automatic re-import of the image.

4. Verify that the images with the given tag have been imported:

```
$ oc describe is nodejs -n openshift
```

**Example output**

```
Name: nodejs
[...]
10-SCL (latest)
```

CHAPTER 3. DEVELOPER CLI (ODO)
3.4.3.3. Overwriting the mirror registry

To download npm packages for Node.js dependencies and Maven packages for Java dependencies from a private mirror registry, you must create and configure a mirror npm or Maven registry on the cluster. You can then overwrite the mirror registry on an existing component or when you create a new component.

**Procedure**

- To overwrite the mirror registry on an existing component:

  ```
  $ odo config set --env NPM_MIRROR=<npm_mirror_registry>
  ```

- To overwrite the mirror registry when creating a component:

  ```
  $ odo component create nodejs --env NPM_MIRROR=<npm_mirror_registry>
  ```

3.4.3.4. Creating a Node.js application with odo

To create a Node.js component, download the Node.js application and push the source code to your cluster with odo.

**Procedure**

1. Change the current directory to the directory with your application:

   ```
   $ cd <directory_name>
   ```

2. Add a component of the type Node.js to your application:

   ```
   $ odo create nodejs
   ```

**NOTE**

By default, the latest image is used. You can also explicitly specify an image version by using `odo create openshift/nodejs:8`. 

---

```
3 minutes ago
[...]
```

```
tagged from <mirror-registry>:<port>/rhscl/nodejs-10-rhel7
prefer registry pullthrough when referencing this tag

Build and run Node.js 10 applications on RHEL 7. For more information about using this builder image, including OpenShift considerations, see https://github.com/nodeshift/centos7-s2i-nodejs.
Tags: builder, nodejs
Example Repo: https://github.com/sclorg/nodejs-ex.git

* <mirror-registry>:<port>/rhscl/nodejs-10-rhel7@sha256:d669ecbc11ac88293de50219dae8619832c6a0f5b04883b480e073590fab7c54

```

---

90
3. Push the initial source code to the component:
   
   ```
   $ odo push
   ```
   
   Your component is now deployed to OpenShift Container Platform.

4. Create a URL and add an entry in the local configuration file as follows:
   
   ```
   $ odo url create --port 8080
   ```

5. Push the changes. This creates a URL on the cluster.
   
   ```
   $ odo push
   ```

6. List the URLs to check the desired URL for the component.
   
   ```
   $ odo url list
   ```

7. View your deployed application using the generated URL.
   
   ```
   $ curl <url>
   ```

### 3.5. Creating Instances of Services Managed by Operators

Operators are a method of packaging, deploying, and managing Kubernetes services. With `odo`, you can create instances of services from the custom resource definitions (CRDs) provided by the Operators. You can then use these instances in your projects and link them to your components.

To create services from an Operator, you must ensure that the Operator has valid values defined in its `metadata` to start the requested service. `odo` uses the `metadata.annotations.alm-examples` YAML file of an Operator to start the service. If this YAML has placeholder values or sample values, a service cannot start. You can modify the YAML file and start the service with the modified values. To learn how to modify YAML files and start services from it, see [Creating services from YAML files](#).

#### 3.5.1. Prerequisites

- Install the `oc` CLI and log in to the cluster.
  - Note that the configuration of the cluster determines the services available to you. To access the Operator services, a cluster administrator must install the respective Operator on the cluster first. To learn more, see [Adding Operators to the cluster](#).

- Install the `odo` CLI.

#### 3.5.2. Creating a project

Create a project to keep your source code, tests, and libraries organized in a separate single unit.

**Procedure**

1. Log in to an OpenShift Container Platform cluster:
   
   ```
   $ odo login -u developer -p developer
   ```
2. Create a project:

```bash
$ odo project create myproject
```

**Example output**

- Project 'myproject' is ready for use
- New project created and now using project : myproject

### 3.5.3. Listing available services from the Operators installed on the cluster

With `odo`, you can display the list of the Operators installed on your cluster, and the services they provide.

- To list the Operators installed in current project, run:

```bash
$ odo catalog list services
```

The command lists Operators and the CRDs. The output of the command shows the Operators installed on your cluster. For example:

```
Operators available in the cluster
NAME                          CRDs
etcdoperator.v0.9.4           EtcdCluster, EtcdBackup, EtcdRestore
mongodb-enterprise.v1.4.5     MongoDB, MongoDBUser, MongoDBOpsManager
```

**etcdoperator.v0.9.4** is the Operator, **EtcdCluster**, **EtcdBackup** and **EtcdRestore** are the CRDs provided by the Operator.

### 3.5.4. Creating a service from an Operator

If an Operator has valid values defined in its `metadata` to start the requested service, you can use the service with `odo service create`.

1. Print the YAML of the service as a file on your local drive:

```bash
$ oc get csv/etcdoperator.v0.9.4 -o yaml
```

2. Verify that the values of the service are valid:

```yaml
apiVersion: etcd.database.coreos.com/v1beta2
kind: EtcdCluster
metadata:
    name: example
spec:
    size: 3
    version: 3.2.13
```

3. Start an **EtcdCluster** service from the **etcdoperator.v0.9.4** Operator:

```bash
$ odo service create etcdoperator.v0.9.4 EtcdCluster
```
4. Verify that a service has started:

   $ oc get EtcdCluster

3.5.5. Creating services from YAML files

If the YAML definition of the service or custom resource (CR) has invalid or placeholder data, you can use the `--dry-run` flag to get the YAML definition, specify the correct values, and start the service using the corrected YAML definition. Printing and modifying the YAML used to start a service `odo` provides the feature to print the YAML definition of the service or CR provided by the Operator before starting a service.

1. To display the YAML of the service, run:

   $ odo service create <operator-name> --dry-run

   For example, to print YAML definition of `EtcdCluster` provided by the `etcdoperator.v0.9.4` Operator, run:

   $ odo service create etcdoperator.v0.9.4 --dry-run

   The YAML is saved as the `etcd.yaml` file.

2. Modify the `etcd.yaml` file:

   ```yaml
   apiVersion: etcd.database.coreos.com/v1beta2
   kind: EtcdCluster
   metadata:
     name: my-etcd-cluster
   spec:
     size: 1
     version: 3.2.13
   ```

   1 Change the name from `example` to `my-etcd-cluster`
   2 Reduce the size from 3 to 1

3. Start a service from the YAML file:

   $ odo service create --from-file etcd.yaml

4. Verify that the `EtcdCluster` service has started with one pod instead of the pre-configured three pods:

   $ oc get pods | grep my-etcd-cluster

3.6. MANAGING ENVIRONMENT VARIABLES

`odo` stores component-specific configurations and environment variables in the `config` file. You can use the `odo config` command to set, unset, and list environment variables for components without the need to modify the `config` file.
3.6.1. Setting and unsetting environment variables

Procedure

- To set an environment variable in a component:
  
  ```
  $ odo config set --env <variable>=<value>
  ```

- To unset an environment variable in a component:
  
  ```
  $ odo config unset --env <variable>
  ```

- To list all environment variables in a component:
  
  ```
  $ odo config view
  ```

3.7. CONFIGURING THE ODO CLI

You can find the global settings for odo in the `preference.yaml` file which is located by default in your `$HOME/.odo` directory.

You can set a different location for the `preference.yaml` file by exporting the `GLOBALODOCONFIG` variable.

3.7.1. Viewing the current configuration

You can view the current odo CLI configuration by using the following command:

```
$ odo preference view
```

Example output

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>CURRENT_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpdateNotification</td>
<td></td>
</tr>
<tr>
<td>NamePrefix</td>
<td></td>
</tr>
<tr>
<td>Timeout</td>
<td></td>
</tr>
<tr>
<td>BuildTimeout</td>
<td></td>
</tr>
<tr>
<td>PushTimeout</td>
<td></td>
</tr>
<tr>
<td>Ephemeral</td>
<td></td>
</tr>
<tr>
<td>ConsentTelemetry</td>
<td>true</td>
</tr>
</tbody>
</table>

3.7.2. Setting a value

You can set a value for a preference key by using the following command:

```
$ odo preference set <key> <value>
```

**NOTE**

Preference keys are case-insensitive.
### 3.7.3. Setting a value

You can unset a value for a preference key by using the following command:

```bash
$ odo preference unset <key>
```

**NOTE**

You can use the `-f` flag to skip the confirmation.

### 3.7.4. Preference key table

The following table shows the available options for setting preference keys for the `odo` CLI:

<table>
<thead>
<tr>
<th>Preference key</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpdateNotification</td>
<td>Control whether a notification to update <code>odo</code> is shown.</td>
<td>True</td>
</tr>
<tr>
<td>NamePrefix</td>
<td>Set a default name prefix for an <code>odo</code> resource. For example, <code>component</code> or <code>storage</code>.</td>
<td>Current directory name</td>
</tr>
<tr>
<td>Timeout</td>
<td>Timeout for the Kubernetes server connection check.</td>
<td>1 second</td>
</tr>
<tr>
<td>BuildTimeout</td>
<td>Timeout for waiting for a build of the git component to complete.</td>
<td>300 seconds</td>
</tr>
<tr>
<td>PushTimeout</td>
<td>Timeout for waiting for a component to start.</td>
<td>240 seconds</td>
</tr>
<tr>
<td>Ephemeral</td>
<td>Controls whether <code>odo</code> should create an <code>emptyDir</code> volume to store source code.</td>
<td>True</td>
</tr>
<tr>
<td>Preference key</td>
<td>Description</td>
<td>Default value</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>ConsentTelemetry</td>
<td>Controls whether odo can collect telemetry for the user’s odo usage.</td>
<td>False</td>
</tr>
</tbody>
</table>

### 3.7.5. Ignoring files or patterns

You can configure a list of files or patterns to ignore by modifying the `.odoignore` file in the root directory of your application. This applies to both `odo push` and `odo watch`.

If the `.odoignore` file does not exist, the `.gitignore` file is used instead for ignoring specific files and folders.

To ignore `.git` files, any files with the `.js` extension, and the folder `tests`, add the following to either the `.odoignore` or the `.gitignore` file:

```
.git
*.js
tests/
```

The `.odoignore` file allows any glob expressions.

### 3.8. ODO CLI REFERENCE

#### 3.8.1. odo build-images

`odo` can build container images based on Dockerfiles, and push these images to their registries.

When running the `odo build-images` command, `odo` searches for all components in the `devfile.yaml` with the `image` type, for example:

```
components:
  - image:
      imageName: quay.io/myusername/myimage
dockerfile:
        uri: ./Dockerfile
        buildContext: ${PROJECTS_ROOT}
name: component-built-from-dockerfile
```

The `uri` field indicates the relative path of the Dockerfile to use, relative to the directory containing the `devfile.yaml`. The dockerfile specification indicates that `uri` could also be an HTTP URL, but this case is not supported by odo yet. The `buildContext` indicates the directory used as build context. The default value is `${PROJECTS_ROOT}`.

For each image component, odo executes either `podman` or `docker` (the first one found, in this order), to build the image with the specified Dockerfile, build context, and arguments.

If the `--push` flag is passed to the command, the images are pushed to their registries after they are built.

#### 3.8.2. odo catalog

OpenShift Container Platform 4.6 CLI tools
**odo** uses different *catalogs* to deploy *components* and *services*.

### 3.8.2.1. Components

**odo** uses the portable *devfile* format to describe the components. It can connect to various devfile registries to download devfiles for different languages and frameworks. See **odo registry** for more information.

#### 3.8.2.1.1. Listing components

To list all the *devfiles* available on the different registries, run the command:

```
$ odo catalog list components
```

**Example output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
<th>REGISTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>go</td>
<td>Stack with the latest Go version</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-maven</td>
<td>Upstream Maven and OpenJDK 11</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>nodejs</td>
<td>Stack with Node.js 14</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>php-laravel</td>
<td>Stack with Laravel 8</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>python</td>
<td>Python Stack with Python 3.7</td>
<td>DefaultDevfileRegistry</td>
</tr>
</tbody>
</table>

#### 3.8.2.1.2. Getting information about a component

To get more information about a specific component, run the command:

```
$ odo catalog describe component
```

For example, run the command:

```
$ odo catalog describe component nodejs
```

**Example output**

* Registry: DefaultDevfileRegistry <.>

  Starter Projects: <.>
  ```
  ---
  name: nodejs-starter
  attributes: {}  
  description: ""
  subdir: ""
  projectsource:  
  sourcetype: ""
  git:  
  gitlikeprojectsourc:  
  commonprojectsourc: {}  
  checkoutfrom: null
  remotes:  
  ```
REGISTRY is the registry from which the devfile is retrieved. Starter projects are sample projects in the same language and framework of the devfile, that can help you start a new project.

See odo create for more information on creating a project from a starter project.

3.8.2.2. Services

odo can deploy services with the help of Operators.

Only Operators deployed with the help of the Operator Lifecycle Manager are supported by odo.

3.8.2.2.1. Listing services

To list the available Operators and their associated services, run the command:

```
$ odo catalog list services
```

**Example output**

```
Services available through Operators
NAME                                 CRDs
postgresql-operator.v0.1.1           Backup, Database
redis-operator.v0.8.0                RedisCluster, Redis
```

In this example, two Operators are installed in the cluster. The postgresql-operator.v0.1.1 Operator deploys services related to PostgreSQL: Backup and Database. The redis-operator.v0.8.0 Operator deploys services related to Redis: RedisCluster and Redis.

**NOTE**

To get a list of all the available Operators, odo fetches the ClusterServiceVersion (CSV) resources of the current namespace that are in a Succeeded phase. For Operators that support cluster-wide access, when a new namespace is created, these resources are automatically added to it. However, it may take some time before they are in the Succeeded phase, and odo may return an empty list until the resources are ready.

3.8.2.2.2. Searching services

To search for a specific service by a keyword, run the command:

```
$ odo catalog search service
```

For example, to retrieve the PostgreSQL services, run the command:

```
$ odo catalog search service postgres
```

**Example output**
You will see a list of Operators that contain the searched keyword in their name.

### 3.8.2.2.3. Getting information about a service

To get more information about a specific service, run the command:

```
$ odo catalog describe service
```

For example:

```
$ odo catalog describe service postgresql-operator.v0.1.1/Database
```

**Example output**

```
KIND:    Database
VERSION: v1alpha1

DESCRIPTION:
Database is the Schema for the Database Database API

FIELDS:
awsAccessKeyId (string)
AWS S3 accessKey/token ID

Key ID of AWS S3 storage. Default Value: nil Required to create the Secret with the data to allow send the backup files to AWS S3 storage.

[...]
```

A service is represented in the cluster by a CustomResourceDefinition (CRD) resource. The previous command displays the details about the CRD such as `kind`, `version`, and the list of fields available to define an instance of this custom resource.

The list of fields is extracted from the `OpenAPI schema` included in the CRD. This information is optional in a CRD, and if it is not present, it is extracted from the `ClusterServiceVersion (CSV)` resource representing the service instead.

It is also possible to request the description of an Operator-backed service, without providing CRD type information. To describe the Redis Operator on a cluster, without CRD, run the following command:

```
$ odo catalog describe service redis-operator.v0.8.0
```

**Example output**

```
NAME: redis-operator.v0.8.0
DESCRIPTION:
A Golang based redis operator that will make/oversee Redis standalone/cluster mode setup on top of the Kubernetes. It can create a redis cluster setup with best practices on Cloud as well as the Bare metal
```

3.8.3. odo create

odo uses a devfile to store the configuration of a component and to describe the component’s resources such as storage and services. The odo create command generates this file.

3.8.3.1. Creating a component

To create a devfile for an existing project, run the odo create command with the name and type of your component (for example, nodejs or go):

```
odo create nodejs mynodejs
```

In the example, nodejs is the type of the component and mynodejs is the name of the component that odo creates for you.

**NOTE**

For a list of all the supported component types, run the command odo catalog list components.

If your source code exists outside the current directory, the --context flag can be used to specify the path. For example, if the source for the nodejs component is in a folder called node-backend relative to the current working directory, run the command:

```
odo create nodejs mynodejs --context ./node-backend
```

The --context flag supports relative and absolute paths.

To specify the project or app where your component will be deployed, use the --project and --app flags. For example, to create a component that is part of the myapp app inside the backend project, run the command:

```
odo create nodejs --app myapp --project backend
```

**NOTE**

If these flags are not specified, they will default to the active app and project.

3.8.3.2. Starter projects
Use the starter projects if you do not have existing source code but want to get up and running quickly to experiment with devfiles and components. To use a starter project, add the `--starter` flag to the `odo create` command.

To get a list of available starter projects for a component type, run the `odo catalog describe component` command. For example, to get all available starter projects for the nodejs component type, run the command:

```
odo catalog describe component nodejs
```

Then specify the desired project using the `--starter` flag on the `odo create` command:

```
odo create nodejs --starter nodejs-starter
```

This will download the example template corresponding to the chosen component type, in this instance, `nodejs`. The template is downloaded to your current directory, or to the location specified by the `--context` flag. If a starter project has its own devfile, then this devfile will be preserved.

### 3.8.3.3. Using an existing devfile

If you want to create a new component from an existing devfile, you can do so by specifying the path to the devfile using the `--devfile` flag. For example, to create a component called `mynodejs`, based on a devfile from GitHub, use the following command:

```
odo create mynodejs --devfile https://raw.githubusercontent.com/odo-devfiles/registry/master/devfiles/nodejs/devfile.yaml
```

### 3.8.3.4. Interactive creation

You can also run the `odo create` command interactively, to guide you through the steps needed to create a component:

```
$ odo create

? Which devfile component type do you wish to create `go`
? What do you wish to name the new devfile component `go-api`
? What project do you want the devfile component to be created in `default`
  Devfile Object Validation
    ✓ Checking devfile existence [164258ns]
    ✓ Creating a devfile component from registry: DefaultDevfileRegistry [246051ns]
  Validation
    ✓ Validating if devfile name is correct [92255ns]
? Do you want to download a starter project `Yes`

  Starter Project
    ✓ Downloading starter project go-starter from https://github.com/devfile-samples/devfile-stack-go.git [429ms]

Please use `odo push` command to create the component with source deployed
```

You are prompted to choose the component type, name, and the project for the component. You can also choose whether or not to download a starter project. Once finished, a new `devfile.yaml` file is created in the working directory.
To deploy these resources to your cluster, run the command `odo push`.

### 3.8.4. odo delete

The `odo delete` command is useful for deleting resources that are managed by `odo`.

#### 3.8.4.1. Deleting a component

To delete a `devfile` component, run the `odo delete` command:

```
$ odo delete
```

If the component has been pushed to the cluster, the component is deleted from the cluster, along with its dependent storage, URL, secrets, and other resources. If the component has not been pushed, the command exits with an error stating that it could not find the resources on the cluster.

Use the `-f` or `--force` flag to avoid the confirmation questions.

#### 3.8.4.2. Undeploying devfile Kubernetes components

To undeploy the devfile Kubernetes components, that have been deployed with `odo deploy`, execute the `odo delete` command with the `--deploy` flag:

```
$ odo delete --deploy
```

Use the `-f` or `--force` flag to avoid the confirmation questions.

#### 3.8.4.3. Delete all

To delete all artifacts including the following items, run the `odo delete` command with the `--all` flag:

- `devfile` component
- Devfile Kubernetes component that was deployed using the `odo deploy` command
- Devfile
- Local configuration

```
$ odo delete --all
```

#### 3.8.4.4. Available flags

- `-f, --force`
  
  Use this flag to avoid the confirmation questions.

- `-w, --wait`
  
  Use this flag to wait for component deletion and any dependencies. This flag does not work when undeploying.

The documentation on `Common Flags` provides more information on the flags available for commands.

### 3.8.5. odo deploy

...
odo can be used to deploy components in a manner similar to how they would be deployed using a CI/CD system. First, odo builds the container images, and then it deploys the Kubernetes resources required to deploy the components.

When running the command **odo deploy**, odo searches for the default command of kind **deploy** in the devfile, and executes this command. The kind **deploy** is supported by the devfile format starting from version 2.2.0.

The **deploy** command is typically a composite command, composed of several **apply** commands:

- A command referencing an **image** component that, when applied, will build the image of the container to deploy, and then push it to its registry.

- A command referencing a **Kubernetes component** that, when applied, will create a Kubernetes resource in the cluster.

With the following example **devfile.yaml** file, a container image is built using the **Dockerfile** present in the directory. The image is pushed to its registry and then a Kubernetes Deployment resource is created in the cluster, using this freshly built image.

```
schemaVersion: 2.2.0
[...]
variables:
  CONTAINER_IMAGE: quay.io/phmartin/myimage
commands:
  - id: build-image
    apply:
      component: outerloop-build
  - id: deployk8s
    apply:
      component: outerloop-deploy
  - id: deploy
    composite:
      commands:
        - build-image
        - deployk8s
      group:
        kind: deploy
        isDefault: true
    components:
    - name: outerloop-build
      image:
        imageName: "{{CONTAINER_IMAGE}}"
        dockerfile:
          uri: ./Dockerfile
          buildContext: ${PROJECTS_ROOT}
      - name: outerloop-deploy
        kubernetes:
          inlined:
            kind: Deployment
            apiVersion: apps/v1
            metadata:
              name: my-component
            spec:
              replicas: 1
              selector:
```

CHAPTER 3. DEVELOPER CLI (ODO)

103
3.8.6. odo link

The odo link command helps link an odo component to an Operator-backed service or another odo component. It does this by using the Service Binding Operator. Currently, odo makes use of the Service Binding library and not the Operator itself to achieve the desired functionality.

3.8.6.1. Various linking options

odo provides various options for linking a component with an Operator-backed service or another odo component. All these options (or flags) can be used whether you are linking a component to a service or to another component.

3.8.6.1.1. Default behavior

By default, the odo link command creates a directory named kubernetes/ in your component directory and stores the information (YAML manifests) about services and links there. When you use odo push, odo compares these manifests with the state of the resources on the Kubernetes cluster and decides whether it needs to create, modify or destroy resources to match what is specified by the user.

3.8.6.1.2. The --inlined flag

If you specify the --inlined flag to the odo link command, odo stores the link information inline in the devfile.yaml in the component directory, instead of creating a file under the kubernetes/ directory. The behavior of the --inlined flag is similar in both the odo link and odo service create commands. This flag is helpful if you want everything stored in a single devfile.yaml. You have to remember to use --inlined flag with each odo link and odo service create command that you execute for the component.

3.8.6.1.3. The --map flag

Sometimes, you might want to add more binding information to the component, in addition to what is available by default. For example, if you are linking the component with a service and would like to bind some information from the service’s spec (short for specification), you could use the --map flag. Note that odo does not do any validation against the spec of the service or component being linked. Using this flag is only recommended if you are comfortable using the Kubernetes YAML manifests.

3.8.6.1.4. The --bind-as-files flag

For all the linking options discussed so far, odo injects the binding information into the component as environment variables. If you would like to mount this information as files instead, you can use the --bind-as-files flag. This will make odo inject the binding information as files into the /bindings location within your component’s Pod. Compared to the environment variables scenario, when you use --bind-as-files, the files are named after the keys and the value of these keys is stored as the contents of these files.
3.8.6.2. Examples

3.8.6.2.1. Default odo link

In the following example, the backend component is linked with the PostgreSQL service using the default odo link command. For the backend component, make sure that your component and service are pushed to the cluster:

```
$ odo list
```

Sample output

<table>
<thead>
<tr>
<th>APP</th>
<th>NAME</th>
<th>PROJECT</th>
<th>TYPE</th>
<th>STATE</th>
<th>MANAGED BY ODO</th>
</tr>
</thead>
<tbody>
<tr>
<td>app</td>
<td>backend</td>
<td>myproject</td>
<td>spring</td>
<td>Pushed</td>
<td>Yes</td>
</tr>
</tbody>
</table>

```

$ odo service list
```

Sample output

<table>
<thead>
<tr>
<th>NAME</th>
<th>MANAGED BY ODO</th>
<th>STATE</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgresCluster/hippo</td>
<td>Yes (backend)</td>
<td>Pushed</td>
<td>59m41s</td>
</tr>
</tbody>
</table>

Now, run odo link to link the backend component with the PostgreSQL service:

```
$ odo link PostgresCluster/hippo
```

Example output

✓ Successfully created link between component "backend" and service "PostgresCluster/hippo"

To apply the link, please use `odo push`

And then run odo push to actually create the link on the Kubernetes cluster.

After a successful odo push, you will see a few outcomes:

1. When you open the URL for the application deployed by backend component, it shows a list of todo items in the database. For example, in the output for the odo url list command, the path where todos are listed is included:

```
$ odo url list
```

Sample output

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>URL</th>
<th>PORT</th>
<th>SECURE</th>
<th>KIND</th>
</tr>
</thead>
</table>

The correct path for the URL would be http://8080-tcp.192.168.39.112.nip.io/api/v1/todos. The exact URL depends on your setup. Also note that there are no todos in the database unless you add some, so the URL might just show an empty JSON object.
2. You can see binding information related to the Postgres service injected into the backend component. This binding information is injected, by default, as environment variables. You can check it using the `odo describe` command from the backend component’s directory:

```
$  odo describe
```

**Example output:**

Component Name: backend  
Type: spring  
Environment Variables:  
· PROJECTS_ROOT=/projects  
· PROJECT_SOURCE=/projects  
· DEBUG_PORT=5858  
Storage:  
· m2 of size 3Gi mounted to /home/user/.m2  
URLs:  
· http://8080-tcp.192.168.39.112.nip.io exposed via 8080  
Linked Services:  
· PostgresCluster/hippo  
  Environment Variables:  
· POSTGRESCLUSTER_PGBOUNCER-EMPTY  
· POSTGRESCLUSTER_PGBOUNCER.INI  
· POSTGRESCLUSTER_ROOT.CRT  
· POSTGRESCLUSTER_VERIFIER  
· POSTGRESCLUSTER_ID_ECDSA  
· POSTGRESCLUSTER_PGBOUNCER-VERIFIER  
· POSTGRESCLUSTER_TLS.CRT  
· POSTGRESCLUSTER_PGBOUNCER-URI  
· POSTGRESCLUSTER_PATRONI.CRT-COMBINED  
· POSTGRESCLUSTER_USER  
· pgImage  
· pgVersion  
· POSTGRESCLUSTER_CLUSTERIP  
· POSTGRESCLUSTER_HOST  
· POSTGRESCLUSTER_PGBACKREST_REPO.CONF  
· POSTGRESCLUSTER_PGBOUNCER-USERS.TXT  
· POSTGRESCLUSTER_SSH_CONFIG  
· POSTGRESCLUSTER_TLS.KEY  
· POSTGRESCLUSTERIALOG-HASH  
· POSTGRESCLUSTER_PASSWORD  
· POSTGRESCLUSTER_PATRONI.CA-ROOTS  
· POSTGRESCLUSTER_DBNAME  
· POSTGRESCLUSTER_PGBOUNCER-PASSWORD  
· POSTGRESCLUSTER_SSHD_CONFIG  
· POSTGRESCLUSTER_PGBOUNCER-FRONTEND.KEY  
· POSTGRESCLUSTER_PGBACKREST_INSTANCE.CONF  
· POSTGRESCLUSTER_PGBOUNCER-FRONTEND.CA-ROOTS  
· POSTGRESCLUSTER_PGBOUNCER-HOST  
· POSTGRESCLUSTER_PORT  
· POSTGRESCLUSTER_ROOT.KEY  
· POSTGRESCLUSTER_SSH_KNOWN_HOSTS  
· POSTGRESCLUSTER_URI  
· POSTGRESCLUSTER_PATRONI.YAML  
· POSTGRESCLUSTER_DNS.CRT
Some of these variables are used in the backend component’s `src/main/resources/application.properties` file so that the Java Spring Boot application can connect to the PostgreSQL database service.

3. Lastly, odo has created a directory called `kubernetes/` in your backend component’s directory that contains the following files:

   ```
   $ ls kubernetes
   odo-service-backend-postgrescluster-hippo.yaml  odo-service-hippo.yaml
   ```

   These files contain the information (YAML manifests) for two resources:

   a. `odo-service-hippo.yaml` - the Postgres service created using `odo service create --from-file ../postgrescluster.yaml` command.

   b. `odo-service-backend-postgrescluster-hippo.yaml` - the link created using `odo link` command.

### 3.8.6.2.2. Using odo link with the --inlined flag

Using the `--inlined` flag with the `odo link` command has the same effect as an `odo link` command without the flag, in that it injects binding information. However, the subtle difference is that in the above case, there are two manifest files under `kubernetes/` directory, one for the Postgres service and another for the link between the backend component and this service. However, when you pass the `--inlined` flag, odo does not create a file under the `kubernetes/` directory to store the YAML manifest, but rather stores it inline in the `devfile.yaml` file.

To see this, unlink the component from the PostgreSQL service first:

```
$ odo unlink PostgresCluster/hippo
```

**Example output:**

```
✓ Successfully unlinked component "backend" from service "PostgresCluster/hippo"

To apply the changes, please use `odo push`
```

To unlink them on the cluster, run `odo push`. Now if you inspect the `kubernetes/` directory, you see only one file:

```
$ ls kubernetes
odo-service-hippo.yaml
```

Next, use the `--inlined` flag to create a link:

```
$ odo link PostgresCluster/hippo --inlined
```
Example output:

✓ Successfully created link between component "backend" and service "PostgresCluster/hippo"

To apply the link, please use `odo push`

You need to run `odo push` for the link to get created on the cluster, like the procedure that omits the --inlined flag. `odo` stores the configuration in `devfile.yaml`. In this file, you can see an entry like the following:

```yaml
kubernetes:
  inlined: |
    apiVersion: binding.operators.coreos.com/v1alpha1
    kind: ServiceBinding
    metadata:
      creationTimestamp: null
      name: backend-postgrescluster-hippo
    spec:
      application:
        group: apps
        name: backend-app
        resource: deployments
        version: v1
      bindAsFiles: false
      detectBindingResources: true
      services:
        - group: postgres-operator.crunchydata.com
          id: hippo
          kind: PostgresCluster
          name: hippo
          version: v1beta1
        - group: postgres-operator.crunchydata.com
          id: hippo
          kind: PostgresCluster
          name: hippo
          version: v1beta1
      status:
        secret: "" 
        name: backend-postgrescluster-hippo
```

Now if you were to run `odo unlink PostgresCluster/hippo`, `odo` would first remove the link information from the `devfile.yaml`, and then a subsequent `odo push` would delete the link from the cluster.

### 3.8.6.2.3. Custom bindings

`odo link` accepts the flag --map which can inject custom binding information into the component. Such binding information will be fetched from the manifest of the resource that you are linking to your component. For example, in the context of the backend component and PostgreSQL service, you can inject information from the PostgreSQL service’s manifest `postgrescluster.yaml` file into the backend component.

If the name of your `PostgresCluster` service is `hippo` (or the output of `odo service list`, if your PostgresCluster service is named differently), when you want to inject the value of `postgresVersion` from that YAML definition into your backend component, run the command:

```bash
$ odo link PostgresCluster/hippo --map pgVersion='{{ .hippo.spec.postgresVersion }}'
```

Note that, if the name of your Postgres service is different from `hippo`, you will have to specify that in the above command in the place of `hippo` in the value for `pgVersion`. 

OpenShift Container Platform 4.6 CLI tools

108
After a link operation, run **odo push** as usual. Upon successful completion of the push operation, you can run the following command from your backend component directory, to validate if the custom mapping got injected properly:

```
$ odo exec -- env | grep pgVersion
```

**Example output:**

```
pgVersion=13
```

Since you might want to inject more than just one piece of custom binding information, **odo link** accepts multiple key-value pairs of mappings. The only constraint is that these should be specified as `--map <key>=<value>`. For example, if you want to also inject PostgreSQL image information along with the version, you could run:

```
$ odo link PostgresCluster/hippo --map pgVersion='{{ .hippo.spec.postgresVersion }}' --map pgImage='{{ .hippo.spec.image }}'
```

and then run **odo push**. To validate if both the mappings got injected correctly, run the following command:

```
$ odo exec -- env | grep -e "pgVersion\|pgImage"
```

**Example output:**

```
pgVersion=13
pgImage=registry.developers.crunchydata.com/crunchydata.crunchy-postgres-ha:centos8-13.4-0
```

3.8.6.2.3.1. To inline or not?

You can accept the default behavior where **odo link** generate a manifests file for the link under `kubernetes/` directory. Alternatively, you can use the `--inlined` flag if you prefer to store everything in a single **devfile.yaml** file.

3.8.6.3. Binding as files

Another helpful flag that **odo link** provides is `--bind-as-files`. When this flag is passed, the binding information is not injected into the component’s Pod as environment variables but is mounted as a filesystem.

Ensure that there are no existing links between the backend component and the PostgreSQL service. You could do this by running **odo describe** in the backend component’s directory and check if you see output similar to the following:

```
Linked Services:
  · PostgresCluster/hippo
```

Unlink the service from the component using:

```
$ odo unlink PostgresCluster/hippo
$ odo push
```
3.8.6.4. **--bind-as-files examples**

### 3.8.6.4.1. Using the default odo link

By default, odo creates the manifest file under the `kubernetes/` directory, for storing the link information. Link the backend component and PostgreSQL service using:

```
$ odo link PostgresCluster/hippo --bind-as-files
$ odo push
```

**Example odo describe output:**

```
$ odo describe

Component Name: backend
Type: spring
Environment Variables:
  · PROJECTS_ROOT=/projects
  · PROJECT_SOURCE=/projects
  · DEBUG_PORT=5858
  · SERVICE_BINDING_ROOT=/bindings
  · SERVICE_BINDING_ROOT=/bindings
Storage:
  · m2 of size 3Gi mounted to /home/user/.m2
URLs:
  · http://8080-tcp.192.168.39.112.nip.io exposed via 8080
Linked Services:
  · PostgresCluster/hippo
Files:
  · /bindings/backend-postgrescluster-hippo/pgbackrest_instance.conf
  · /bindings/backend-postgrescluster-hippo/user
  · /bindings/backend-postgrescluster-hippo/ssh_known_hosts
  · /bindings/backend-postgrescluster-hippo/clusterIP
  · /bindings/backend-postgrescluster-hippo/password
  · /bindings/backend-postgrescluster-hippo/patroni.yaml
  · /bindings/backend-postgrescluster-hippo/pgbouncer-frontend.crt
  · /bindings/backend-postgrescluster-hippo/pgbouncer-host
  · /bindings/backend-postgrescluster-hippo/root.key
  · /bindings/backend-postgrescluster-hippo/pgbouncer-frontend.key
  · /bindings/backend-postgrescluster-hippo/pgbouncer.ini
  · /bindings/backend-postgrescluster-hippo/pgbouncer-uri
  · /bindings/backend-postgrescluster-hippo/verify
```
Everything that was an environment variable in the `key=value` format in the earlier `odo describe` output is now mounted as a file. Use the `cat` command to view the contents of some of these files:

**Example command:**

```
$ odo exec -- cat /bindings/backend-postgrescluster-hippo/password
```

**Example output:**

```
q({JC:jn^mm/Bw}eu+j.GX{k
```

**Example command:**

```
$ odo exec -- cat /bindings/backend-postgrescluster-hippo/user
```

**Example output:**

```
hippo
```

**Example command:**

```
$ odo exec -- cat /bindings/backend-postgrescluster-hippo/clusterIP
```

**Example output:**

```
10.101.78.56
```

### 3.8.6.4.2. Using `--inlined`

The result of using `--bind-as-files` and `--inlined` together is similar to using `odo link --inlined`. The manifest of the link gets stored in the `devfile.yaml`, instead of being stored in a separate file under `kubernetes/` directory. Other than that, the `odo describe` output would be the same as earlier.

### 3.8.6.4.3. Custom bindings

When you pass custom bindings while linking the backend component with the PostgreSQL service, these custom bindings are injected not as environment variables but are mounted as files. For example:

- `/bindings/backend-postgrescluster-hippo/dbname`
- `/bindings/backend-postgrescluster-hippo/patroni.ca-roots`
- `/bindings/backend-postgrescluster-hippo/pgbackrest_repo.conf`
- `/bindings/backend-postgrescluster-hippo/pgbouncer-port`
- `/bindings/backend-postgrescluster-hippo/pgbouncer-verifier`
- `/bindings/backend-postgrescluster-hippo/id_ecdsa`
- `/bindings/backend-postgrescluster-hippo/id_ecdsa.pub`
- `/bindings/backend-postgrescluster-hippo/pgbouncer-password`
- `/bindings/backend-postgrescluster-hippo/pgbouncer-users.txt`
- `/bindings/backend-postgrescluster-hippo/sshd_config`
- `/bindings/backend-postgrescluster-hippo/tls.crt`
These custom bindings get mounted as files instead of being injected as environment variables. To validate that this worked, run the following command:

**Example command:**

```
$ odo exec -- cat /bindings/backend-postgrescluster-hippo/pgVersion
```

**Example output:**

```
13
```

**Example command:**

```
$ odo exec -- cat /bindings/backend-postgrescluster-hippo/pgImage
```

**Example output:**

```
registry.developers.crunchydata.com/crunchydata/crunchy-postgres-ha:centos8-13.4-0
```

### 3.8.7. **odo registry**

odo uses the portable devfile format to describe the components. odo can connect to various devfile registries, to download devfiles for different languages and frameworks.

You can connect to publicly available devfile registries, or you can install your own Secure Registry.

You can use the odo registry command to manage the registries that are used by odo to retrieve devfile information.

#### 3.8.7.1. Listing the registries

To list the registries currently contacted by odo, run the command:

```
$ odo registry list
```

**Example output:**

```
NAME                              URL                    SECURE
DefaultDevfileRegistry           https://registry.devfile.io  No
```

**DefaultDevfileRegistry** is the default registry used by odo; it is provided by the devfile.io project.

#### 3.8.7.2. Adding a registry

To add a registry, run the command:

```
$ odo registry add
```
Example output:

```bash
$ odo registry add StageRegistry https://registry.stage.devfile.io
New registry successfully added

$ odo registry add MyRegistry https://myregistry.example.com --token <access_token>
New registry successfully added
```

3.8.7.3. Deleting a registry

To delete a registry, run the command:

```bash
$ odo registry delete
```

Example output:

```
$ odo registry delete StageRegistry
? Are you sure you want to delete registry "StageRegistry" Yes
Successfully deleted registry
```

Use the `--force` (or `-f`) flag to force the deletion of the registry without confirmation.

3.8.7.4. Updating a registry

To update the URL or the personal access token of a registry already registered, run the command:

```bash
$ odo registry update
```

Example output:

```bash
$ odo registry update MyRegistry https://otherregistry.example.com --token <other_access_token>
? Are you sure you want to update registry "MyRegistry" Yes
Successfully updated registry
```

Use the `--force` (or `-f`) flag to force the update of the registry without confirmation.

3.8.8. odo service

`odo` can deploy services with the help of Operators.

The list of available Operators and services available for installation can be found using the `odo catalog` command.

Services are created in the context of a component, so run the `odo create` command before you deploy services.

A service is deployed using two steps:

1. Define the service and store its definition in the devfile.
2. Deploy the defined service to the cluster, using the `odo push` command.

### 3.8.8.1. Creating a new service

To create a new service, run the command:

```bash
$ odo service create
```

For example, to create an instance of a Redis service named `my-redis-service`, you can run the following command:

**Example output**

```bash
$ odo catalog list services
Services available through Operators
NAME                      CRDs
redis-operator.v0.8.0     RedisCluster, Redis
```

```bash
$ odo service create redis-operator.v0.8.0/Redis my-redis-service
Successfully added service to the configuration; do 'odo push' to create service on the cluster
```

This command creates a Kubernetes manifest in the `kubernetes/` directory, containing the definition of the service, and this file is referenced from the `devfile.yaml` file.

```bash
$ cat kubernetes/odo-service-my-redis-service.yaml
```

**Example output**

```yaml
apiVersion: redis.redis.opstreelabs.in/v1beta1
class: Redis
metadata:
  name: my-redis-service
spec:
  kubernetesConfig:
    image: quay.io/opstree/redis:v6.2.5
    imagePullPolicy: IfNotPresent
    resources:
      limits:
        cpu: 101m
        memory: 128Mi
      requests:
        cpu: 101m
        memory: 128Mi
  serviceType: ClusterIP
  redisExporter:
    enabled: false
    image: quay.io/opstree/redis-exporter:1.0
  storage:
    volumeClaimTemplate:
      spec:
        accessModes:
        - ReadWriteOnce
```

OpenShift Container Platform 4.6 CLI tools

114
Example command

$ cat devfile.yaml

Example output

```
[...
components:
  - kubernetes:
    uri: kubernetes/odo-service-my-redis-service.yaml
    name: my-redis-service
[...
```

Note that the name of the created instance is optional. If you do not provide a name, it will be the lowercase name of the service. For example, the following command creates an instance of a Redis service named `redis`:

```
$ odo service create redis-operator.v0.8.0/Redis my-redis-service --inlined
```

Successfully added service to the configuration; do ‘odo push’ to create service on the cluster

Example command

```
$ cat devfile.yaml
```

Example output

```
[...
components:
  - kubernetes:
    inlined: |
      apiVersion: redis.redis.opstreelabs.in/v1beta1
      kind: Redis
      metadata:
        name: my-redis-service
      spec:
        kubernetesConfig:
          image: quay.io/opstree/redis:v6.2.5
          imagePullPolicy: IfNotPresent
          resources:
            limits:
              cpu: 101m
```

3.8.8.1.1. Inlining the manifest

By default, a new manifest is created in the `kubernetes/` directory, referenced from the `devfile.yaml` file. It is possible to inline the manifest inside the `devfile.yaml` file using the `--inlined` flag:

```
$ odo service create redis-operator.v0.8.0/Redis my-redis-service --inlined
```

```
```
3.8.8.1.2. Configuring the service

Without specific customization, the service will be created with a default configuration. You can use either command-line arguments or a file to specify your own configuration.

3.8.8.1.2.1. Using command-line arguments

Use the `--parameters` (or `-p`) flag to specify your own configuration.

The following example configures the Redis service with three parameters:

```
$ odo service create redis-operator.v0.8.0/Redis my-redis-service \
   -p kubernetesConfig.image=quay.io/opstree/redis:v6.2.5 \
   -p kubernetesConfig.serviceType=ClusterIP \
   -p redisExporter.image=quay.io/opstree/redis-exporter:1.0
```

Successfully added service to the configuration; do 'odo push' to create service on the cluster

Example command

```
$ cat kubernetes/odo-service-my-redis-service.yaml
```

Example output

```
apiVersion: redis.redis.opstreelabs.in/v1beta1
kind: Redis
metadata:
  name: my-redis-service
spec:
  kubernetesConfig:
    image: quay.io/opstree/redis:v6.2.5
    serviceType: ClusterIP
  redisExporter:
    image: quay.io/opstree/redis-exporter:1.0
```
You can obtain the possible parameters for a specific service using the `odo catalog describe service` command.

### 3.8.8.1.2.2. Using a file

Use a YAML manifest to configure your own specification. In the following example, the Redis service is configured with three parameters.

1. Create a manifest:

   ```
   $ cat > my-redis.yaml <<EOF
   apiVersion: redis.redis.opstreelabs.in/v1beta1
   kind: Redis
   metadata:
     name: my-redis-service
   spec:
     kubernetesConfig:
       image: quay.io/opstree/redis:v6.2.5
       serviceType: ClusterIP
     redisExporter:
       image: quay.io/opstree/redis-exporter:1.0
   EOF
   ```

2. Create the service from the manifest:

   ```
   $ odo service create --from-file my-redis.yaml
   Successfully added service to the configuration; do 'odo push' to create service on the cluster
   ```

### 3.8.8.2. Deleting a service

To delete a service, run the command:

```
$ odo service delete
```

**Example output**

```
$ odo service list
NAME                       MANAGED BY ODO     STATE               AGE
Redis/my-redis-service     Yes (api)          Deleted locally     5m39s

$ odo service delete Redis/my-redis-service
? Are you sure you want to delete Redis/my-redis-service Yes
Service "Redis/my-redis-service" has been successfully deleted; do 'odo push' to delete service from the cluster
```

Use the `--force` (or `-f`) flag to force the deletion of the service without confirmation.

### 3.8.8.3. Listing services

To list the services created for your component, run the command:

```
$ odo service list
```
Example output

$ odo service list
NAME                MANAGED BY ODO STATE             AGE
Redis/my-redis-service-1 Yes (api) Not pushed
Redis/my-redis-service-2 Yes (api) Pushed      52s
Redis/my-redis-service-3 Yes (api) Deleted locally 1m22s

For each service, **STATE** indicates if the service has been pushed to the cluster using the **odo push** command, or if the service is still running on the cluster but removed from the devfile locally using the **odo service delete** command.

3.8.8.4. Getting information about a service

To get details of a service such as its kind, version, name, and list of configured parameters, run the command:

```bash
$ odo service describe
```

Example output

```
$ odo service describe Redis/my-redis-service
Version: redis.redis.opstreelabs.in/v1beta1
Kind: Redis
Name: my-redis-service
Parameters:
NAME                VALUE
kubernetesConfig.image quay.io/opstree/redis:v6.2.5
kubernetesConfig.serviceType ClusterIP
redisExporter.image   quay.io/opstree/redis-exporter:1.0
```

3.8.9. odo storage

**odo** lets users manage storage volumes that are attached to the components. A storage volume can be either an ephemeral volume using an **emptyDir** Kubernetes volume, or a **Persistent Volume Claim** (PVC). A PVC allows users to claim a persistent volume (such as a GCE PersistentDisk or an iSCSI volume) without understanding the details of the particular cloud environment. The persistent storage volume can be used to persist data across restarts and rebuilds of the component.

3.8.9.1. Adding a storage volume

To add a storage volume to the cluster, run the command:

```bash
$ odo storage create
```

Example output:

```
$ odo storage create store --path /data --size 1Gi
✓ Added storage store to nodejs-project-ufyy

$ odo storage create tempdir --path /tmp --size 2Gi --ephemeral
```
In the above example, the first storage volume has been mounted to the `/data` path and has a size of 1Gi, and the second volume has been mounted to `/tmp` and is ephemeral.

### 3.8.9.2. Listing the storage volumes

To check the storage volumes currently used by the component, run the command:

```
$ odo storage list
```

**Example output:**

```
$ odo storage list
The component 'nodejs-project-ufyy' has the following storage attached:
NAME     SIZE     PATH      STATE
store    1Gi      /data     Not Pushed
tempdir  2Gi      /tmp      Not Pushed
```

### 3.8.9.3. Deleting a storage volume

To delete a storage volume, run the command:

```
$ odo storage delete
```

**Example output:**

```
$ odo storage delete store -f
Deleted storage store from nodejs-project-ufyy
Please use `odo push` command to delete the storage from the cluster
```

In the above example, using the `-f` flag force deletes the storage without asking user permission.

### 3.8.9.4. Adding storage to specific container

If your devfile has multiple containers, you can specify which container you want the storage to attach to, using the `--container` flag in the `odo storage create` command.

The following example is an excerpt from a devfile with multiple containers:

```
components:
  - name: nodejs1
    container:
      image: registry.access.redhat.com/ubi8/nodejs-12:1-36
      memoryLimit: 1024Mi
    endpoints:
      - name: "3000-tcp"
        targetPort: 3000
    mountSources: true
```
In the example, there are two containers, `nodejs1` and `nodejs2`. To attach storage to the `nodejs2` container, use the following command:

```
$ odo storage create --container
```

Example output:

```
$ odo storage create store --path /data --size 1Gi --container nodejs2
✓ Added storage store to nodejs-testing-xnfg

Please use `odo push` command to make the storage accessible to the component
```

You can list the storage resources, using the `odo storage list` command:

```
$ odo storage list
```

Example output:

```
The component 'nodejs-testing-xnfg' has the following storage attached:
NAME SIZE PATH CONTAINER STATE
store 1Gi /data nodejs2 Not Pushed
```

### 3.8.10. Common flags

The following flags are available with most `odo` commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--context</td>
<td>Set the context directory where the component is defined.</td>
</tr>
<tr>
<td>--project</td>
<td>Set the project for the component. Defaults to the project defined in the local configuration. If none is available, then current project on the cluster.</td>
</tr>
<tr>
<td>--app</td>
<td>Set the application of the component. Defaults to the application defined in the local configuration. If none is available, then <code>app</code>.</td>
</tr>
<tr>
<td>--kubeconfig</td>
<td>Set the path to the <code>kubeconfig</code> value if not using the default configuration.</td>
</tr>
<tr>
<td>--show-log</td>
<td>Use this flag to see the logs.</td>
</tr>
<tr>
<td>-f, --force</td>
<td>Use this flag to tell the command not to prompt the user for confirmation.</td>
</tr>
<tr>
<td>-v, --v</td>
<td>Set the verbosity level. See <code>Logging in odo</code> for more information.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Output the help for a command.</td>
</tr>
</tbody>
</table>

**NOTE**

Some flags might not be available for some commands. Run the command with the **--help** flag to get a list of all the available flags.

### 3.8.11. JSON output

The **odo** commands that output content generally accept a **-o json** flag to output this content in JSON format, suitable for other programs to parse this output more easily.

The output structure is similar to Kubernetes resources, with the **kind**, **apiVersion**, **metadata**, **spec**, and **status** fields.

*List* commands return a **List** resource, containing an **items** (or similar) field listing the items of the list, with each item also being similar to Kubernetes resources.

*Delete* commands return a **Status** resource; see the **Status Kubernetes resource**.

Other commands return a resource associated with the command, for example, **Application**, **Storage**, **URL**, and so on.

The full list of commands currently accepting the **-o json** flag is:

<table>
<thead>
<tr>
<th>Commands</th>
<th>Kind (version)</th>
<th>Kind (version) of list items</th>
<th>Complete content?</th>
</tr>
</thead>
<tbody>
<tr>
<td>odo application describe</td>
<td>Application (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>no</td>
</tr>
<tr>
<td>odo application list</td>
<td>List (odo.dev/v1alpha1)</td>
<td>Application (odo.dev/v1alpha1)</td>
<td>?</td>
</tr>
<tr>
<td>odo catalog list components</td>
<td>List (odo.dev/v1alpha1)</td>
<td>missing</td>
<td>yes</td>
</tr>
<tr>
<td>odo catalog list services</td>
<td>List (odo.dev/v1alpha1)</td>
<td>ClusterServiceVersion</td>
<td>?</td>
</tr>
<tr>
<td>odo catalog describe component</td>
<td>missing</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo catalog describe service</td>
<td>CRDDescription (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>Commands</td>
<td>Kind (version)</td>
<td>Kind (version) of list items</td>
<td>Complete content?</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------</td>
<td>------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>odo component create</td>
<td>Component (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo component describe</td>
<td>Component (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo component list</td>
<td>List (odo.dev/v1alpha1)</td>
<td>Component (odo.dev/v1alpha1)</td>
<td>yes</td>
</tr>
<tr>
<td>odo config view</td>
<td>DevfileConfiguration (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo debug info</td>
<td>OdoDebugInfo (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo env view</td>
<td>EnvInfo (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo preference view</td>
<td>PreferenceList (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo project create</td>
<td>Project (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo project delete</td>
<td>Status (v1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo project get</td>
<td>Project (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo project list</td>
<td>List (odo.dev/v1alpha1)</td>
<td>Project (odo.dev/v1alpha1)</td>
<td>yes</td>
</tr>
<tr>
<td>odo registry list</td>
<td>List (odo.dev/v1alpha1)</td>
<td>missing</td>
<td>yes</td>
</tr>
<tr>
<td>odo service create</td>
<td>Service</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo service describe</td>
<td>Service</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo service list</td>
<td>List (odo.dev/v1alpha1)</td>
<td>Service</td>
<td>yes</td>
</tr>
<tr>
<td>odo storage create</td>
<td>Storage (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo storage delete</td>
<td>Status (v1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
</tbody>
</table>
3.9. ODO ARCHITECTURE

This section describes odo architecture and how odo manages resources on a cluster.

3.9.1. Developer setup

With odo you can create and deploy application on OpenShift Container Platform clusters from a terminal. Code editor plug-ins use odo which allows users to interact with OpenShift Container Platform clusters from their IDE terminals. Examples of plug-ins that use odo: VS Code OpenShift Connector, OpenShift Connector for Intellij, Codewind for Eclipse Che.

odo works on Windows, macOS, and Linux operating systems and from any terminal. odo provides autocompletion for bash and zsh command line shells.

odo supports Node.js and Java components.

3.9.2. OpenShift source-to-image

OpenShift Source-to-Image (S2I) is an open-source project which helps in building artifacts from source code and injecting these into container images. S2I produces ready-to-run images by building source code without the need of a Dockerfile. odo uses S2I builder image for executing developer source code inside a container.

3.9.3. OpenShift cluster objects

3.9.3.1. Init Containers

Init containers are specialized containers that run before the application container starts and configure the necessary environment for the application containers to run. Init containers can have files that application images do not have, for example setup scripts. Init containers always run to completion and the application container does not start if any of the init containers fails.

The pod created by odo executes two Init Containers:

- The copy-supervisord Init container.
- The copy-files-to-volume Init container.

3.9.3.1.1. copy-supervisord

The copy-supervisord Init container copies necessary files onto an emptyDir volume. The main application container utilizes these files from the emptyDir volume.

Files that are copied onto the emptyDir volume:
**Binaries:**

- **go-init** is a minimal init system. It runs as the first process (PID 1) inside the application container. go-init starts the **SupervisorD** daemon which runs the developer code. go-init is required to handle orphaned processes.

- **SupervisorD** is a process control system. It watches over configured processes and ensures that they are running. It also restarts services when necessary. For odo, **SupervisorD** executes and monitors the developer code.

**Configuration files:**

- **supervisor.conf** is the configuration file necessary for the SupervisorD daemon to start.

**Scripts:**

- **assemble-and-restart** is an OpenShift S2I concept to build and deploy user-source code. The assemble-and-restart script first assembles the user source code inside the application container and then restarts SupervisorD for user changes to take effect.

- **Run** is an OpenShift S2I concept of executing the assembled source code. The **run** script executes the assembled code created by the **assemble-and-restart** script.

- **s2i-setup** is a script that creates files and directories which are necessary for the **assemble-and-restart** and **run** scripts to execute successfully. The script is executed whenever the application container starts.

**Directories:**

- **language-scripts**: OpenShift S2I allows custom **assemble** and **run** scripts. A few language specific custom scripts are present in the **language-scripts** directory. The custom scripts provide additional configuration to make odo debug work.

The **emptyDir** volume is mounted at the **/opt/odo** mount point for both the Init container and the application container.

### 3.9.3.1.2. copy-files-to-volume

The **copy-files-to-volume** Init container copies files that are in **/opt/app-root** in the S2I builder image onto the persistent volume. The volume is then mounted at the same location (**/opt/app-root**) in an application container.

Without the persistent volume on **/opt/app-root** the data in this directory is lost when the persistent volume claim is mounted at the same location.

The PVC is mounted at the **/mnt** mount point inside the Init container.

### 3.9.3.2. Application container

Application container is the main container inside of which the user-source code executes.

Application container is mounted with two volumes:

- **emptyDir** volume mounted at **/opt/odo**

- The persistent volume mounted at **/opt/app-root**
**go-init** is executed as the first process inside the application container. The **go-init** process then starts the **SupervisorD** daemon.

**SupervisorD** executes and monitors the user assembled source code. If the user process crashes, **SupervisorD** restarts it.

### 3.9.3.3. Persistent volumes and persistent volume claims

A persistent volume claim (PVC) is a volume type in Kubernetes which provisions a persistent volume. The life of a persistent volume is independent of a pod lifecycle. The data on the persistent volume persists across pod restarts.

The **copy-files-to-volume** Init container copies necessary files onto the persistent volume. The main application container utilizes these files at runtime for execution.

The naming convention of the persistent volume is `<component_name>-s2idata`.

<table>
<thead>
<tr>
<th>Container</th>
<th>PVC mounted at</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy-files-to-volume</td>
<td>/mnt</td>
</tr>
<tr>
<td>Application container</td>
<td>/opt/app-root</td>
</tr>
</tbody>
</table>

#### 3.9.3.4. emptyDir volume

An **emptyDir** volume is created when a pod is assigned to a node, and exists as long as that pod is running on the node. If the container is restarted or moved, the content of the **emptyDir** is removed, Init container restores the data back to the **emptyDir**. **emptyDir** is initially empty.

The **copy-supervisord** Init container copies necessary files onto the **emptyDir** volume. These files are then utilized by the main application container at runtime for execution.

<table>
<thead>
<tr>
<th>Container</th>
<th>emptyDir volume mounted at</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy-supervisord</td>
<td>/opt/odo</td>
</tr>
<tr>
<td>Application container</td>
<td>/opt/odo</td>
</tr>
</tbody>
</table>

#### 3.9.3.5. Service

A service is a Kubernetes concept of abstracting the way of communicating with a set of pods.

**odo** creates a service for every application pod to make it accessible for communication.

### 3.9.4. odo push workflow

This section describes **odo push** workflow. **odo push** deploys user code on an OpenShift Container Platform cluster with all the necessary OpenShift Container Platform resources.

1. Creating resources
   - If not already created, **odo** push creates the following OpenShift Container Platform resources:
- **DeploymentConfig** object:
  - Two init containers are executed: `copy-supervisord` and `copy-files-to-volume`. The init containers copy files onto the `emptyDir` and the `PersistentVolume` type of volumes respectively.
  - The application container starts. The first process in the application container is the `go-init` process with PID=1.
  - `go-init` process starts the SupervisorD daemon.

  **NOTE**

  The user application code has not been copied into the application container yet, so the SupervisorD daemon does not execute the `run` script.

- **Service** object

- **Secret** objects

- **PersistentVolumeClaim** object

2. Indexing files

  - A file indexer indexes the files in the source code directory. The indexer traverses through the source code directories recursively and finds files which have been created, deleted, or renamed.
  - A file indexer maintains the indexed information in an odo index file inside the `.odo` directory.
  - If the odo index file is not present, it means that the file indexer is being executed for the first time, and creates a new odo index JSON file. The odo index JSON file contains a file map - the relative file paths of the traversed files and the absolute paths of the changed and deleted files.

3. Pushing code

   Local code is copied into the application container, usually under `/tmp/src`.

4. Executing `assemble-and-restart`

   On a successful copy of the source code, the `assemble-and-restart` script is executed inside the running application container.

### 3.10. odo RELEASE NOTES

#### 3.10.1. Notable changes and improvements in odo version 2.5.0

- Creates unique routes for each component, using `adler32` hashing
- Supports additional fields in the devfile for assigning resources:
  - `cpuRequest`
  - `cpuLimit`
- memoryRequest
- memoryLimit

- Adds the `--deploy` flag to the `odo delete` command, to remove components deployed using the `odo deploy` command:

  ```
  $ odo delete --deploy
  ```

- Adds mapping support to the `odo link` command

- Supports ephemeral volumes using the `ephemeral` field in `volume` components

- Sets the default answer to `yes` when asking for telemetry opt-in

- Improves metrics by sending additional telemetry data to the devfile registry

- Updates the bootstrap image to `registry.access.redhat.com/ocp-tools-4/odo-init-container-rhel8:1.1.11`

- The upstream repository is available at `https://github.com/redhat-developer/odo`

### 3.10.2. Bug fixes

- Previously, `odo deploy` would fail if the `.odo/env` file did not exist. The command now creates the `.odo/env` file if required.

- Previously, interactive component creation using the `odo create` command would fail if disconnect from the cluster. This issue is fixed in the latest release.

### 3.10.3. Getting support

#### For Product

If you find an error, encounter a bug, or have suggestions for improving the functionality of `odo`, file an issue in Bugzilla. Choose the Red Hat odo for OpenShift Container Platform product type.

Provide as many details in the issue description as possible.

#### For Documentation

If you find an error or have suggestions for improving the documentation, file an issue in Bugzilla. Choose the OpenShift Container Platform product type and the Documentation component type.
CHAPTER 4. HELM CLI

4.1. GETTING STARTED WITH HELM 3

4.1.1. Understanding Helm

Helm is a software package manager that simplifies deployment of applications and services to OpenShift Container Platform clusters.

Helm uses a packaging format called charts. A Helm chart is a collection of files that describes the OpenShift Container Platform resources.

A running instance of the chart in a cluster is called a release. A new release is created every time a chart is installed on the cluster.

Each time a chart is installed, or a release is upgraded or rolled back, an incremental revision is created.

4.1.1.1. Key features

Helm provides the ability to:

- Search through a large collection of charts stored in the chart repository.
- Modify existing charts.
- Create your own charts with OpenShift Container Platform or Kubernetes resources.
- Package and share your applications as charts.

4.1.2. Installing Helm

The following section describes how to install Helm on different platforms using the CLI.

You can also find the URL to the latest binaries from the OpenShift Container Platform web console by clicking the ? icon in the upper-right corner and selecting Command Line Tools.

Prerequisites

- You have installed Go, version 1.13 or higher.

4.1.2.1. On Linux

1. Download the Helm binary and add it to your path:

   ```
   # chmod +x /usr/local/bin/helm
   ```

2. Make the binary file executable:

   ```
   # chmod +x /usr/local/bin/helm
   ```

3. Check the installed version:
4.1.2.2. On Windows 7/8

1. Download the latest .exe file and put in a directory of your preference.

2. Right click Start and click Control Panel.

3. Select System and Security and then click System.

4. From the menu on the left, select Advanced systems settings and click Environment Variables at the bottom.

5. Select Path from the Variable section and click Edit.

6. Click New and type the path to the folder with the .exe file into the field or click Browse and select the directory, and click OK.

4.1.2.3. On Windows 10

1. Download the latest .exe file and put in a directory of your preference.

2. Click Search and type env or environment.

3. Select Edit environment variables for your account

4. Select Path from the Variable section and click Edit.

5. Click New and type the path to the directory with the exe file into the field or click Browse and select the directory, and click OK.

4.1.2.4. On MacOS

1. Download the Helm binary and add it to your path:


2. Make the binary file executable:

   # chmod +x /usr/local/bin/helm

3. Check the installed version:

   $ helm version

   Example output

   version.BuildInfo{Version:"v3.0",
   GitCommit:"b31719aab7963acf4887a1c1e6d5e53378e34d93", GitTreeState:"clean",
   GoVersion:"go1.13.4"}
4.1.3. Installing a Helm chart on an OpenShift Container Platform cluster

Prerequisites

- You have a running OpenShift Container Platform cluster and you have logged into it.
- You have installed Helm.

Procedure

1. Create a new project:

   ```
   $ oc new-project mysql
   ```

2. Add a repository of Helm charts to your local Helm client:

   ```
   $ helm repo add stable https://kubernetes-charts.storage.googleapis.com/
   "stable" has been added to your repositories
   ```

3. Update the repository:

   ```
   $ helm repo update
   ```

4. Install an example MySQL chart:

   ```
   $ helm install example-mysql stable/mysql
   ```

5. Verify that the chart has installed successfully:

   ```
   $ helm list
   ```

   **Example output**

   ```
   NAME        NAMESPACE REVISION UPDATED STATUS     CHART APP VERSION
   example-mysql mysql 1 2019-12-05 15:06:51.379134163 -0500 EST deployed mysql-1.5.0 5.7.27
   ```

4.1.4. Creating a custom Helm chart on OpenShift Container Platform

Procedure

1. Create a new project:
Download an example Node.js chart that contains OpenShift Container Platform objects:

```
$ git clone https://github.com/redhat-developer/redhat-helm-charts
```

Go to the directory with the sample chart:

```
$ cd redhat-helm-charts,alpha/nodejs-ex-k/
```

Edit the `Chart.yaml` file and add a description of your chart:

```
apiVersion: v2
name: nodejs-ex-k
description: A Helm chart for OpenShift
icon: https://static.redhat.com/libs/redhat/brand-assets/latest/corp/logo.svg
```

1. The chart API version. It should be `v2` for Helm charts that require at least Helm 3.
2. The name of your chart.
3. The description of your chart.
4. The URL to an image to be used as an icon.

Verify that the chart is formatted properly:

```
$ helm lint

Example output

[INFO] Chart.yaml: icon is recommended
1 chart(s) linted, 0 chart(s) failed
```

Navigate to the previous directory level:

```
$ cd ..
```

Install the chart:

```
$ helm install nodejs-chart nodejs-ex-k
```

Verify that the chart has installed successfully:

```
$ helm list
```

Example output
4.2. CONFIGURING CUSTOM HELM CHART REPOSITORIES

The Developer Catalog, in the Developer perspective of the web console, displays the Helm charts available in the cluster. By default, it lists the Helm charts from the Red Hat Helm chart repository. For a list of the charts see the Red Hat Helm index file.

As a cluster administrator, you can add multiple Helm chart repositories, apart from the default one, and display the Helm charts from these repositories in the Developer Catalog.

4.2.1. Adding custom Helm chart repositories

You can add custom Helm chart repositories to your cluster, and enable access to the Helm charts from these repositories in the Developer Catalog.

Procedure

1. To add a new Helm Chart Repository, you must add the Helm Chart Repository custom resource (CR) to your cluster.

Sample Helm Chart Repository CR

```
apiVersion: helm.openshift.io/v1beta1
type: HelmChartRepository
metadata:
  name: <name>
spec:
  # optional name that might be used by console
  # name: <chart-display-name>
  connectionConfig:
    url: <helm-chart-repository-url>
```

For example, to add an Azure sample chart repository, run:

```
$ cat <<EOF | oc apply -f -
apiVersion: helm.openshift.io/v1beta1
type: HelmChartRepository
metadata:
  name: azure-sample-repo
spec:
  name: azure-sample-repo
  connectionConfig:
    url: https://raw.githubusercontent.com/Azure-Samples/helm-charts/master/docs
EOF
```

2. Navigate to the Developer Catalog in the web console to verify that the helm charts from the Azure chart repository are displayed.

4.2.2. Creating credentials and CA certificates to add Helm chart repositories
Some Helm chart repositories need credentials and custom certificate authority (CA) certificates to connect to it. You can use the web console as well as the CLI to add credentials and certificates.

**Procedure**

To configure the credentials and certificates, and then add a Helm chart repository using the CLI:

1. In the `openshift-config` namespace, create a **ConfigMap** object with a custom CA certificate in PEM encoded format, and store it under the `ca-bundle.crt` key within the config map:

   ```
   $ oc create configmap helm-ca-cert \
   --from-file=ca-bundle.crt=/path/to/certs/ca.crt \
   -n openshift-config
   ```

2. In the `openshift-config` namespace, create a **Secret** object to add the client TLS configurations:

   ```
   $ oc create secret generic helm-tls-configs \
   --from-file=tls.crt=/path/to/certs/client.crt \
   --from-file=tls.key=/path/to/certs/client.key \
   -n openshift-config
   ```

   Note that the client certificate and key must be in PEM encoded format and stored under the keys `tls.crt` and `tls.key`, respectively.

3. Add the Helm repository as follows:

   ```
   $ cat <<EOF | oc apply -f -
   apiVersion: helm.openshift.io/v1beta1
   kind: HelmChartRepository
   metadata:
     name: <helm-repository>
   spec:
     name: <helm-repository>
     connectionConfig:
       url: <URL for the Helm repository>
       tlsConfig:
         name: helm-tls-configs
         ca:
           name: helm-ca-cert
   EOF
   ```

   The **ConfigMap** and **Secret** are consumed in the HelmChartRepository CR using the `tlsConfig` and `ca` fields. These certificates are used to connect to the Helm repository URL.

4. By default, all authenticated users have access to all configured charts. However, for chart repositories where certificates are needed, you must provide users with read access to the `helm-ca-cert` config map and `helm-tls-configs` secret in the `openshift-config` namespace, as follows:

   ```
   $ cat <<EOF | kubectl apply -f -
   apiVersion: rbac.authorization.k8s.io/v1
   kind: Role
   metadata:
     namespace: openshift-config
   name: helm-chartrepos-tls-conf-viewer
   ```
rules:
- apiGroups: ['""
  resources: ['"configmaps"
  resourceNames: ['"helm-ca-cert"
  verbs: ['"get"
- apiGroups: ['""
  resources: ['"secrets"
  resourceNames: ['"helm-tls-configs"
  verbs: ['"get"
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: openshift-config
  name: helm-chartrepos-tls-conf-viewer
subjects:
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: 'system:authenticated'
roleRef:
  apiGroup: rbac.authorization.k8s.io
  name: helm-chartrepos-tls-conf-viewer
  kind: Role
EOF
The Knative `kn` CLI enables simple interaction with Knative components on OpenShift Container Platform.

### 5.1. KEY FEATURES

The `kn` CLI is designed to make serverless computing tasks simple and concise. Key features of the `kn` CLI include:

- Deploy serverless applications from the command line.
- Manage features of Knative Serving, such as services, revisions, and traffic-splitting.
- Create and manage Knative Eventing components, such as event sources and triggers.
- Create sink bindings to connect existing Kubernetes applications and Knative services.
- Extend the `kn` CLI with flexible plug-in architecture, similar to the `kubectl` CLI.
- Configure autoscaling parameters for Knative services.
- Scripted usage, such as waiting for the results of an operation, or deploying custom rollout and rollback strategies.

### 5.2. INSTALLING THE KNATIVE CLI

See Installing the Knative CLI.
CHAPTER 6. PIPELINES CLI (TKN)

6.1. INSTALLING TKN

Use the `tkn` CLI to manage Red Hat OpenShift Pipelines from a terminal. The following section describes how to install `tkn` on different platforms.

You can also find the URL to the latest binaries from the OpenShift Container Platform web console by clicking the ? icon in the upper-right corner and selecting Command Line Tools.

6.1.1. Installing Red Hat OpenShift Pipelines CLI (tkn) on Linux

For Linux distributions, you can download the CLI directly as a `tar.gz` archive.

**Procedure**

1. Download the relevant CLI.
   - Linux (x86_64, amd64)
   - Linux on IBM Z and LinuxONE (s390x)
   - Linux on IBM Power Systems (ppc64le)

2. Unpack the archive:
   ```bash
   $ tar xvzf <file>
   ```

3. Place the `tkn` binary in a directory that is on your `PATH`.

4. To check your `PATH`, run:
   ```bash
   $ echo $PATH
   ```

6.1.2. Installing Red Hat OpenShift Pipelines CLI (tkn) on Linux using an RPM

For Red Hat Enterprise Linux (RHEL) version 8, you can install the Red Hat OpenShift Pipelines CLI (`tkn`) as an RPM.

**Prerequisites**

- You have an active OpenShift Container Platform subscription on your Red Hat account.
- You have root or sudo privileges on your local system.

**Procedure**

1. Register with Red Hat Subscription Manager:
   ```bash
   # subscription-manager register
   ```

2. Pull the latest subscription data:
3. List the available subscriptions:

```bash
# subscription-manager list --available --matches "pipelines"
```

4. In the output for the previous command, find the pool ID for your OpenShift Container Platform subscription and attach the subscription to the registered system:

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

5. Enable the repositories required by Red Hat OpenShift Pipelines:

```bash
# subscription-manager repos --enable="pipelines-1.2-for-rhel-8-x86_64-rpms"
```

6. Install the `openshift-pipelines-client` package:

```bash
# yum install openshift-pipelines-client
```

After you install the CLI, it is available using the `tkn` command:

```bash
$ tkn version
```

### 6.1.3. Installing Red Hat OpenShift Pipelines CLI (tkn) on Windows

For Windows, the `tkn` CLI is provided as a `zip` archive.

**Procedure**

1. Download the CLI.

2. Unzip the archive with a ZIP program.

3. Add the location of your `tkn.exe` file to your `PATH` environment variable.

4. To check your `PATH`, open the command prompt and run the command:

```bash
C:\> path
```

### 6.1.4. Installing Red Hat OpenShift Pipelines CLI (tkn) on macOS

For macOS, the `tkn` CLI is provided as a `tar.gz` archive.

**Procedure**

1. Download the CLI.

2. Unpack and unzip the archive.

3. Move the `tkn` binary to a directory on your `PATH`.

4. To check your `PATH`, open a terminal window and run:
6.2. CONFIGURING THE OPENSHIFT PIPELINES TKN CLI

Configure the Red Hat OpenShift Pipelines tkn CLI to enable tab completion.

6.2.1. Enabling tab completion

After you install the tkn CLI, you can enable tab completion to automatically complete tkn commands or suggest options when you press Tab.

Prerequisites

- You must have the tkn CLI tool installed.
- You must have bash-completion installed on your local system.

Procedure

The following procedure enables tab completion for Bash.

1. Save the Bash completion code to a file:

   ```
   $ tkn completion bash > tkn_bash_completion
   ```

2. Copy the file to /etc/bash_completion.d/:

   ```
   $ sudo cp tkn_bash_completion /etc/bash_completion.d/
   ```

   Alternatively, you can save the file to a local directory and source it from your .bashrc file instead.

Tab completion is enabled when you open a new terminal.

6.3. OPENSIFT PIPELINES TKN REFERENCE

This section lists the basic tkn CLI commands.

6.3.1. Basic syntax

   tkn [command or options] [arguments...]

6.3.2. Global options

   --help, -h

6.3.3. Utility commands

6.3.3.1. tkn

Parent command for tkn CLI.
Example: Display all options

$ tkn

6.3.3.2. completion [shell]
Print shell completion code which must be evaluated to provide interactive completion. Supported shells are bash and zsh.

Example: Completion code for bash shell

$ tkn completion bash

6.3.3.3. version
Print version information of the tkn CLI.

Example: Check the tkn version

$ tkn version

6.3.4. Pipelines management commands

6.3.4.1. pipeline
Manage Pipelines.

Example: Display help

$ tkn pipeline --help

6.3.4.2. pipeline delete
Delete a Pipeline.

Example: Delete the mypipeline Pipeline from a namespace

$ tkn pipeline delete mypipeline -n myspace

6.3.4.3. pipeline describe
Describe a Pipeline.

Example: Describe mypipeline Pipeline

$ tkn pipeline describe mypipeline

6.3.4.4. pipeline list
List Pipelines.
Example: Display a list of Pipelines

$ tkn pipeline list

6.3.4.5. pipeline logs
Display Pipeline logs for a specific Pipeline.

Example: Stream live logs for the mypipeline Pipeline

$ tkn pipeline logs -f mypipeline

6.3.4.6. pipeline start
Start a Pipeline.

Example: Start mypipeline Pipeline

$ tkn pipeline start mypipeline

6.3.5. PipelineRun commands

6.3.5.1. pipelinerun
Manage PipelineRuns.

Example: Display help

$ tkn pipelinerun -h

6.3.5.2. pipelinerun cancel
Cancel a PipelineRun.

Example: Cancel the mypipelinerun PipelineRun from a namespace

$ tkn pipelinerun cancel mypipelinerun -n myspace

6.3.5.3. pipelinerun delete
Delete a PipelineRun.

Example: Delete PipelineRuns from a namespace

$ tkn pipelinerun delete mypipelinerun1 mypipelinerun2 -n myspace

6.3.5.4. pipelinerun describe
Describe a PipelineRun.
Example: Describe the `mypipelinerun` PipelineRun in a namespace

```bash
$ tkn pipelinerun describe mypipelinerun -n myspace
```

6.3.5.5. pipelinerun list

List PipelineRuns.

Example: Display a list of PipelineRuns in a namespace

```bash
$ tkn pipelinerun list -n myspace
```

6.3.5.6. pipelinerun logs

Display the logs of a PipelineRun.

Example: Display the logs of the `mypipelinerun` PipelineRun with all tasks and steps in a namespace

```bash
$ tkn pipelinerun logs mypipelinerun -a -n myspace
```

6.3.6. Task management commands

6.3.6.1. task

Manage Tasks.

Example: Display help

```bash
$ tkn task -h
```

6.3.6.2. task delete

Delete a Task.

Example: Delete `mytask1` and `mytask2` Tasks from a namespace

```bash
$ tkn task delete mytask1 mytask2 -n myspace
```

6.3.6.3. task describe

Describe a Task.

Example: Describe the `mytask` Task in a namespace

```bash
$ tkn task describe mytask -n myspace
```

6.3.6.4. task list

List Tasks.
Example: List all the Tasks in a namespace

$ tkn task list -n myspace

6.3.6.5. task logs
Display Task logs.

Example: Display logs for the mytaskrun TaskRun of the mytask Task

$ tkn task logs mytask mytaskrun -n myspace

6.3.6.6. task start
Start a Task.

Example: Start the mytask Task in a namespace

$ tkn task start mytask -s <ServiceAccountName> -n myspace

6.3.7. TaskRun commands

6.3.7.1. taskrun
Manage TaskRuns.

Example: Display help

$ tkn taskrun -h

6.3.7.2. taskrun cancel
Cancel a TaskRun.

Example: Cancel the mytaskrun TaskRun from a namespace

$ tkn taskrun cancel mytaskrun -n myspace

6.3.7.3. taskrun delete
Delete a TaskRun.

Example: Delete mytaskrun1 and mytaskrun2 TaskRuns from a namespace

$ tkn taskrun delete mytaskrun1 mytaskrun2 -n myspace

6.3.7.4. taskrun describe
Describe a TaskRun.
Example: Describe the mytaskrun TaskRun in a namespace

$ tkn taskrun describe mytaskrun -n myspace

6.3.7.5. taskrun list
List TaskRuns.

Example: List all TaskRuns in a namespace

$ tkn taskrun list -n myspace

6.3.7.6. taskrun logs
Display TaskRun logs.

Example: Display live logs for the mytaskrun TaskRun in a namespace

$ tkn taskrun logs -f mytaskrun -n myspace

6.3.8. Condition management commands

6.3.8.1. condition
Manage Conditions.

Example: Display help

$ tkn condition --help

6.3.8.2. condition delete
Delete a Condition.

Example: Delete the mycondition1 Condition from a namespace

$ tkn condition delete mycondition1 -n myspace

6.3.8.3. condition describe
Describe a Condition.

Example: Describe the mycondition1 Condition in a namespace

$ tkn condition describe mycondition1 -n myspace

6.3.8.4. condition list
List Conditions.
Example: List Conditions in a namespace

$ tkn condition list -n myspace

6.3.9. Pipeline Resource management commands

6.3.9.1. resource
Manage Pipeline Resources.

Example: Display help

$ tkn resource -h

6.3.9.2. resource create
Create a Pipeline Resource.

Example: Create a Pipeline Resource in a namespace

$ tkn resource create -n myspace

This is an interactive command that asks for input on the name of the Resource, type of the Resource, and the values based on the type of the Resource.

6.3.9.3. resource delete
Delete a Pipeline Resource.

Example: Delete the myresource Pipeline Resource from a namespace

$ tkn resource delete myresource -n myspace

6.3.9.4. resource describe
Describe a Pipeline Resource.

Example: Describe the myresource Pipeline Resource

$ tkn resource describe myresource -n myspace

6.3.9.5. resource list
List Pipeline Resources.

Example: List all Pipeline Resources in a namespace

$ tkn resource list -n myspace

6.3.10. ClusterTask management commands
6.3.10.1. clustertask
Manage ClusterTasks.

Example: Display help

$ tkn clustertask --help

6.3.10.2. clustertask delete
Delete a ClusterTask resource in a cluster.

Example: Delete mytask1 and mytask2 ClusterTasks

$ tkn clustertask delete mytask1 mytask2

6.3.10.3. clustertask describe
Describe a ClusterTask.

Example: Describe the mytask ClusterTask

$ tkn clustertask describe mytask1

6.3.10.4. clustertask list
List ClusterTasks.

Example: List ClusterTasks

$ tkn clustertask list

6.3.10.5. clustertask start
Start ClusterTasks.

Example: Start the mytask ClusterTask

$ tkn clustertask start mytask

6.3.11. Trigger management commands

6.3.11.1. eventlistener
Manage EventListeners.

Example: Display help

$ tkn eventlistener -h
6.3.11.2. eventlistener delete
Delete an EventListener.

**Example:** Delete mylistener1 and mylistener2 EventListeners in a namespace

```
$ tkn eventlistener delete mylistener1 mylistener2 -n myspace
```

6.3.11.3. eventlistener describe
Describe an EventListener.

**Example:** Describe the mylistener EventListener in a namespace

```
$ tkn eventlistener describe mylistener -n myspace
```

6.3.11.4. eventlistener list
List EventListeners.

**Example:** List all the EventListeners in a namespace

```
$ tkn eventlistener list -n myspace
```

6.3.11.5. triggerbinding
Manage TriggerBindings.

**Example:** Display TriggerBindings help

```
$ tkn triggerbinding -h
```

6.3.11.6. triggerbinding delete
Delete a TriggerBinding.

**Example:** Delete mybinding1 and mybinding2 TriggerBindings in a namespace

```
$ tkn triggerbinding delete mybinding1 mybinding2 -n myspace
```

6.3.11.7. triggerbinding describe
Describe a TriggerBinding.

**Example:** Describe the mybinding TriggerBinding in a namespace

```
$ tkn triggerbinding describe mybinding -n myspace
```

6.3.11.8. triggerbinding list
List TriggerBindings.

Example: List all the TriggerBindings in a namespace

$ tkn triggerbinding list -n myspace

6.3.11.9. triggertemplate

Manage TriggerTemplates.

Example: Display TriggerTemplate help

$ tkn triggertemplate -h

6.3.11.10. triggertemplate delete

Delete a TriggerTemplate.

Example: Delete mytemplate1 and mytemplate2 TriggerTemplates in a namespace

$ tkn triggertemplate delete mytemplate1 mytemplate2 -n `myspace`

6.3.11.11. triggertemplate describe

Describe a TriggerTemplate.

Example: Describe the mytemplate TriggerTemplate in a namespace

$ tkn triggertemplate describe mytemplate -n `myspace`

6.3.11.12. triggertemplate list

List TriggerTemplates.

Example: List all the TriggerTemplates in a namespace

$ tkn triggertemplate list -n myspace

6.3.11.13. clustertriggerbinding

Manage ClusterTriggerBindings.

Example: Display ClusterTriggerBindings help

$ tkn clustertriggerbinding -h

6.3.11.14. clustertriggerbinding delete

Delete a ClusterTriggerBinding.
Example: Delete myclusterbinding1 and myclusterbinding2 ClusterTriggerBindings

$ tkn clustertriggerbinding delete myclusterbinding1 myclusterbinding2

6.3.11.15. clustertriggerbinding describe
Describe a ClusterTriggerBinding.

Example: Describe the myclusterbinding ClusterTriggerBinding

$ tkn clustertriggerbinding describe myclusterbinding

6.3.11.16. clustertriggerbinding list
List ClusterTriggerBindings.

Example: List all ClusterTriggerBindings

$ tkn clustertriggerbinding list
CHAPTER 7. OPM CLI

7.1. ABOUT OPM

The **opm** CLI tool is provided by the Operator Framework for use with the Operator Bundle Format. This tool allows you to create and maintain catalogs of Operators from a list of bundles, called an *index*, that are similar to software repositories. The result is a container image, called an *index image*, which can be stored in a container registry and then installed on a cluster.

An index contains a database of pointers to Operator manifest content that can be queried through an included API that is served when the container image is run. On OpenShift Container Platform, Operator Lifecycle Manager (OLM) can use the index image as a catalog by referencing it in a **CatalogSource** object, which polls the image at regular intervals to enable frequent updates to installed Operators on the cluster.

**Additional resources**

- See [Operator Framework packaging formats](#) for more information about the Bundle Format.
- To create a bundle image using the Operator SDK, see [Working with bundle images](#).

7.2. INSTALLING OPM

You can install the **opm** CLI tool on your Linux, macOS, or Windows workstation.

**Prerequisites**

- For Linux, you must provide the following packages. RHEL 8 meets these requirements:
  
  - **podman** version 1.9.3+ (version 2.0+ recommended)
  - **glibc** version 2.28+

**Procedure**

1. Navigate to the [OpenShift mirror site](#) and download the latest version of the tarball that matches your operating system.

2. Unpack the archive.

   - For Linux or macOS:
     
     ```bash
     $ tar xvf <file>
     ```

   - For Windows, unzip the archive with a ZIP program.

3. Place the file anywhere in your **PATH**.

   - For Linux or macOS:
     
     ```bash
     $ echo $PATH
     ```

   - For Windows, check your **PATH**:
b. Move the file. For example:

```bash
$ sudo mv ./opm /usr/local/bin/
```

- For Windows:
  a. Check your **PATH**:

```bash
C:\> path
```

  b. Move the file:

```bash
C:\> move opm.exe <directory>
```

**Verification**

- After you install the **opm** CLI, verify that it is available:

```bash
$ opm version
```

**Example output**

```bash
Version: version.Version{OpmVersion:"v1.14.3-29-ge82c4649", GitCommit:"e82c4649b208cd0b08dbd5b88da55450f97b3a2d", BuildDate:"2021-02-13T02:20:52Z", GoOs:"linux", GoArch:"amd64"}
```

### 7.3. ADDITIONAL RESOURCES

- See [Managing custom catalogs](#) for **opm** procedures including creating, updating, and pruning index images.