



OpenShift Container Platform 4.4

CLI tools

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

OpenShift Container Platform 4.4 CLI tools

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

Legal Notice

Copyright © 2021 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

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. OPENSIFT CLI (OC)	9
1.1. GETTING STARTED WITH THE CLI	9
1.1.1. About the CLI	9
1.1.2. Installing the CLI	9
1.1.2.1. Installing the CLI by downloading the binary	9
1.1.2.1.1. Installing the CLI on Linux	9
1.1.2.1.2. Installing the CLI on Windows	10
1.1.2.1.3. Installing the CLI on macOS	10
1.1.2.2. Installing the CLI by using an RPM	10
1.1.3. Logging in to the CLI	11
1.1.4. Using the CLI	12
1.1.4.1. Creating a project	12
1.1.4.2. Creating a new app	13
1.1.4.3. Viewing pods	13
1.1.4.4. Viewing pod logs	13
1.1.4.5. Viewing the current project	13
1.1.4.6. Viewing the status for the current project	13
1.1.4.7. Listing supported API resources	14
1.1.5. Getting help	14
1.1.6. Logging out of the CLI	15
1.2. CONFIGURING THE CLI	15
1.2.1. Enabling tab completion	15
1.3. EXTENDING THE CLI WITH PLUG-INS	16
1.3.1. Writing CLI plug-ins	16
1.3.2. Installing and using CLI plug-ins	17
1.4. DEVELOPER CLI COMMANDS	18
1.4.1. Basic CLI commands	18
1.4.1.1. explain	18
1.4.1.2. login	18
1.4.1.3. new-app	18
1.4.1.4. new-project	19
1.4.1.5. project	19
1.4.1.6. projects	19
1.4.1.7. status	19
1.4.2. Build and Deploy CLI commands	19
1.4.2.1. cancel-build	19
1.4.2.2. import-image	20
1.4.2.3. new-build	20
1.4.2.4. rollback	20
1.4.2.5. rollout	20
1.4.2.6. start-build	20
1.4.2.7. tag	21
1.4.3. Application management CLI commands	21
1.4.3.1. annotate	21
1.4.3.2. apply	21
1.4.3.3. autoscale	21
1.4.3.4. create	22
1.4.3.5. delete	22
1.4.3.6. describe	22
1.4.3.7. edit	22
1.4.3.8. expose	22

1.4.3.9. get	23
1.4.3.10. label	23
1.4.3.11. scale	23
1.4.3.12. secrets	23
1.4.3.13. serviceaccounts	23
1.4.3.14. set	24
1.4.4. Troubleshooting and debugging CLI commands	24
1.4.4.1. attach	24
1.4.4.2. cp	24
1.4.4.3. debug	24
1.4.4.4. exec	24
1.4.4.5. logs	24
1.4.4.6. port-forward	25
1.4.4.7. proxy	25
1.4.4.8. rsh	25
1.4.4.9. rsync	25
1.4.4.10. run	25
1.4.4.11. wait	25
1.4.5. Advanced developer CLI commands	26
1.4.5.1. api-resources	26
1.4.5.2. api-versions	26
1.4.5.3. auth	26
1.4.5.4. cluster-info	26
1.4.5.5. convert	26
1.4.5.6. extract	27
1.4.5.7. idle	27
1.4.5.8. image	27
1.4.5.9. observe	27
1.4.5.10. patch	27
1.4.5.11. policy	28
1.4.5.12. process	28
1.4.5.13. registry	28
1.4.5.14. replace	28
1.4.6. Settings CLI commands	28
1.4.6.1. completion	28
1.4.6.2. config	29
1.4.6.3. logout	29
1.4.6.4. whoami	29
1.4.7. Other developer CLI commands	29
1.4.7.1. help	29
1.4.7.2. plugin	29
1.4.7.3. version	30
1.5. ADMINISTRATOR CLI COMMANDS	30
1.5.1. Cluster management CLI commands	30
1.5.1.1. inspect	30
1.5.1.2. must-gather	30
1.5.1.3. top	30
1.5.2. Node management CLI commands	31
1.5.2.1. cordon	31
1.5.2.2. drain	31
1.5.2.3. node-logs	31
1.5.2.4. taint	31
1.5.2.5. uncordon	31

1.5.3. Security and policy CLI commands	31
1.5.3.1. certificate	32
1.5.3.2. groups	32
1.5.3.3. new-project	32
1.5.3.4. pod-network	32
1.5.3.5. policy	32
1.5.4. Maintenance CLI commands	32
1.5.4.1. migrate	32
1.5.4.2. prune	33
1.5.5. Configuration CLI commands	33
1.5.5.1. create-bootstrap-policy-file	33
1.5.5.2. create-bootstrap-project-template	33
1.5.5.3. create-error-template	33
1.5.5.4. create-kubeconfig	33
1.5.5.5. create-login-template	34
1.5.5.6. create-provider-selection-template	34
1.5.6. Other Administrator CLI commands	34
1.5.6.1. build-chain	34
1.5.6.2. completion	34
1.5.6.3. config	34
1.5.6.4. release	35
1.5.6.5. verify-image-signature	35
1.6. USAGE OF OC AND KUBECTL COMMANDS	35
1.6.1. The oc binary	35
1.6.2. The kubectl binary	35
CHAPTER 2. DEVELOPER CLI (ODO)	37
2.1. UNDERSTANDING OPENSIFT DO	37
2.1.1. Key features	37
2.1.2. Core concepts	37
2.1.2.1. Officially supported languages and corresponding container images	37
2.1.2.1.1. Listing available container images	38
2.2. ODO ARCHITECTURE	39
2.2.1. Developer setup	39
2.2.2. OpenShift source-to-image	39
2.2.3. OpenShift cluster objects	39
2.2.3.1. Init Containers	39
2.2.3.1.1. copy-supervisord	39
2.2.3.1.2. copy-files-to-volume	40
2.2.3.2. Application container	40
2.2.3.3. Persistent volumes and persistent volume claims	40
2.2.3.4. emptyDir volume	41
2.2.3.5. Service	41
2.2.4. odo push workflow	41
2.3. INSTALLING ODO	42
2.3.1. Installing odo on Linux	42
2.3.1.1. Binary installation	42
2.3.1.2. Tarball installation	43
2.3.2. Installing odo on Windows	43
2.3.2.1. Binary installation	43
Setting the PATH variable for Windows 7/8	43
Setting the PATH variable for Windows 10	43
2.3.3. Installing odo on macOS	43

2.3.3.1. Binary installation	43
2.3.3.2. Tarball installation	43
2.4. USING ODO IN A RESTRICTED ENVIRONMENT	44
2.4.1. About odo in a restricted environment	44
2.4.2. Pushing the odo init image to the restricted cluster registry	44
2.4.2.1. Prerequisites	44
2.4.2.2. Pushing the odo init image to a mirror registry	44
2.4.2.2.1. Pushing the init image to a mirror registry on Linux	44
2.4.2.2.2. Pushing the init image to a mirror registry on MacOS	45
2.4.2.2.3. Pushing the init image to a mirror registry on Windows	45
2.4.2.3. Pushing the odo init image to an internal registry directly	46
2.4.2.3.1. Pushing the init image directly on Linux	46
2.4.2.3.2. Pushing the init image directly on MacOS	47
2.4.2.3.3. Pushing the init image directly on Windows	48
2.4.3. Creating and deploying a component to the disconnected cluster	49
2.4.3.1. Prerequisites	49
2.4.3.2. Mirroring a supported builder image	49
2.4.3.3. Overwriting the mirror registry	51
2.4.3.4. Creating a Node.js application with odo	51
2.5. CREATING A SINGLE-COMPONENT APPLICATION WITH ODO	52
2.5.1. Prerequisites	52
2.5.2. Creating a project	52
2.5.3. Creating a Node.js application with odo	53
2.5.4. Modifying your application code	54
2.5.5. Adding storage to the application components	54
2.5.6. Adding a custom builder to specify a build image	54
2.5.7. Connecting your application to multiple services using OpenShift Service Catalog	55
2.5.8. Deleting an application	55
2.6. CREATING A MULTICOMPONENT APPLICATION WITH ODO	56
2.6.1. Prerequisites	56
2.6.2. Creating a project	56
2.6.3. Deploying the back-end component	56
2.6.4. Deploying the front-end component	58
2.6.5. Linking both components	59
2.6.6. Exposing components to the public	60
2.6.7. Modifying the running application	61
2.6.8. Deleting an application	61
2.7. CREATING AN APPLICATION WITH A DATABASE	62
2.7.1. Prerequisites	62
2.7.2. Creating a project	62
2.7.3. Deploying the front-end component	63
2.7.4. Deploying a database in interactive mode	64
2.7.5. Deploying a database manually	64
2.7.6. Connecting the database to the front-end application	65
2.7.7. Deleting an application	66
2.8. CREATING APPLICATIONS BY USING DEVFILES	66
2.8.1. About the devfile in odo	67
2.8.2. Creating a Java application by using a devfile	67
2.8.3. Prerequisites	67
2.8.3.1. Creating a project	67
2.8.3.2. Listing available devfile components	68
2.8.3.3. Deploying a Java application using a devfile	68
2.9. USING SAMPLE APPLICATIONS	70

2.9.1. Examples from Git repositories	70
2.9.1.1. httpd	70
2.9.1.2. java	71
2.9.1.3. nodejs	71
2.9.1.4. perl	71
2.9.1.5. php	71
2.9.1.6. python	71
2.9.1.7. ruby	71
2.9.1.8. wildfly	72
2.9.2. Binary examples	72
2.9.2.1. java	72
2.9.2.2. wildfly	72
2.10. CREATING INSTANCES OF SERVICES MANAGED BY OPERATORS	72
2.10.1. Prerequisites	73
2.10.2. Creating a project	73
2.10.3. Listing available services from the Operators installed on the cluster	73
2.10.4. Creating a service from an Operator	74
2.10.5. Creating services from YAML files	74
2.11. DEBUGGING APPLICATIONS IN ODO	75
2.11.1. Debugging an application	75
2.11.2. Configuring debugging parameters	76
2.12. MANAGING ENVIRONMENT VARIABLES	76
2.12.1. Setting and unsetting environment variables	76
2.13. CONFIGURING THE ODO CLI	76
2.13.1. Using command completion	76
2.13.2. Ignoring files or patterns	77
2.14. ODO CLI REFERENCE	77
2.14.1. Basic odo CLI commands	78
2.14.1.1. app	78
2.14.1.2. catalog	78
2.14.1.3. component	78
2.14.1.4. config	79
2.14.1.5. create	80
2.14.1.6. debug	81
2.14.1.7. delete	82
2.14.1.8. describe	82
2.14.1.9. link	82
2.14.1.10. list	82
2.14.1.11. log	83
2.14.1.12. login	83
2.14.1.13. logout	83
2.14.1.14. preference	83
2.14.1.15. project	84
2.14.1.16. push	85
2.14.1.17. service	85
2.14.1.18. storage	85
2.14.1.19. unlink	86
2.14.1.20. update	86
2.14.1.21. url	86
2.14.1.22. utils	87
2.14.1.23. version	87
2.14.1.24. watch	87
2.15. ODO RELEASE NOTES	88

2.15.1. Notable changes and improvements in <code>odo</code>	88
2.15.2. Getting support	89
2.15.3. Fixed issues	89
2.15.4. Known issues	89
2.15.5. Technology Preview features <code>odo</code>	89
CHAPTER 3. HELM CLI	91
3.1. GETTING STARTED WITH HELM 3 ON OPENSIFT CONTAINER PLATFORM	91
3.1.1. Understanding Helm	91
3.1.1.1. Key features	91
3.1.2. Installing Helm	91
3.1.2.1. On Linux	91
3.1.2.2. On Windows 7/8	92
3.1.2.3. On Windows 10	92
3.1.2.4. On MacOS	92
3.1.3. Installing a Helm chart on an OpenShift Container Platform cluster	92
3.1.4. Creating a custom Helm chart on OpenShift Container Platform	93
CHAPTER 4. KNATIVE CLI (KN) FOR USE WITH OPENSIFT SERVERLESS	95
4.1. KEY FEATURES	95
4.2. INSTALLING KN	95
CHAPTER 5. PIPELINES CLI (TKN)	96
5.1. INSTALLING TKN	96
5.1.1. Installing OpenShift Pipelines CLI (<code>tkn</code>) on Linux	96
5.1.2. Installing OpenShift Pipelines CLI (<code>tkn</code>) on Linux using an RPM	96
5.1.3. Installing OpenShift Pipelines CLI (<code>tkn</code>) on Windows	97
5.1.4. Installing OpenShift Pipelines CLI (<code>tkn</code>) on macOS	97
5.2. CONFIGURING THE OPENSIFT PIPELINES TKN CLI	97
5.2.1. Enabling tab completion	98
5.3. OPENSIFT PIPELINES TKN REFERENCE	98
5.3.1. Basic syntax	98
5.3.2. Global options	98
5.3.3. Utility commands	98
5.3.3.1. <code>tkn</code>	98
5.3.3.2. <code>completion [shell]</code>	98
5.3.3.3. <code>version</code>	99
5.3.4. Pipelines management commands	99
5.3.4.1. <code>pipeline</code>	99
5.3.4.2. <code>pipeline create</code>	99
5.3.4.3. <code>pipeline delete</code>	99
5.3.4.4. <code>pipeline describe</code>	99
5.3.4.5. <code>pipeline list</code>	99
5.3.4.6. <code>pipeline logs</code>	100
5.3.4.7. <code>pipeline start</code>	100
5.3.5. PipelineRun commands	100
5.3.5.1. <code>pipelinerun</code>	100
5.3.5.2. <code>pipelinerun cancel</code>	100
5.3.5.3. <code>pipelinerun delete</code>	100
5.3.5.4. <code>pipelinerun describe</code>	100
5.3.5.5. <code>pipelinerun list</code>	101
5.3.5.6. <code>pipelinerun logs</code>	101
5.3.6. Task management commands	101
5.3.6.1. <code>task</code>	101

5.3.6.2. task create	101
5.3.6.3. task delete	101
5.3.6.4. task describe	102
5.3.6.5. task list	102
5.3.6.6. task logs	102
5.3.6.7. task start	102
5.3.7. TaskRun commands	102
5.3.7.1. taskrun	102
5.3.7.2. taskrun cancel	102
5.3.7.3. taskrun delete	103
5.3.7.4. taskrun describe	103
5.3.7.5. taskrun list	103
5.3.7.6. taskrun logs	103
5.3.8. Condition management commands	103
5.3.8.1. condition	103
5.3.8.2. condition delete	103
5.3.8.3. condition describe	104
5.3.8.4. condition list	104
5.3.9. Pipeline Resource management commands	104
5.3.9.1. resource	104
5.3.9.2. resource create	104
5.3.9.3. resource delete	104
5.3.9.4. resource describe	104
5.3.9.5. resource list	105
5.3.10. ClusterTask management commands	105
5.3.10.1. clustertask	105
5.3.10.2. clustertask delete	105
5.3.10.3. clustertask describe	105
5.3.10.4. clustertask list	105
5.3.10.5. clustertask start	105
5.3.11. Trigger management commands	106
5.3.11.1. eventlistener	106
5.3.11.2. eventlistener delete	106
5.3.11.3. eventlistener describe	106
5.3.11.4. eventlistener list	106
5.3.11.5. triggerbinding	106
5.3.11.6. triggerbinding delete	106
5.3.11.7. triggerbinding describe	107
5.3.11.8. triggerbinding list	107
5.3.11.9. triggertemplate	107
5.3.11.10. triggertemplate delete	107
5.3.11.11. triggertemplate describe	107
5.3.11.12. triggertemplate list	107
5.3.11.13. clustertriggerbinding	107
5.3.11.14. clustertriggerbinding delete	108
5.3.11.15. clustertriggerbinding describe	108
5.3.11.16. clustertriggerbinding list	108

CHAPTER 1. OPENSIFT CLI (OC)

1.1. GETTING STARTED WITH THE CLI

1.1.1. About the CLI

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

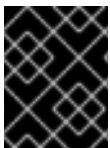
- work directly with project source code.
- script OpenShift Container Platform operations.
- are restricted by bandwidth resources and can not use the web console.

1.1.2. Installing the CLI

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

1.1.2.1. Installing the 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 4.4. Download and install the new version of **oc**.

1.1.2.1.1. Installing the CLI on Linux

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

Procedure

1. Navigate to the [Infrastructure Provider](#) page on the Red Hat OpenShift Cluster Manager site.
2. Select your infrastructure provider, and, if applicable, your installation type.
3. In the **Command-line interface** section, select **Linux** from the drop-down menu and click **Download command-line tools**.
4. Unpack the archive:

```
$ tar xvzf <file>
```

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

```
$ echo $PATH
```

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

```
$ oc <command>
```

1.1.2.1.2. Installing the CLI on Windows

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

Procedure

1. Navigate to the [Infrastructure Provider](#) page on the Red Hat OpenShift Cluster Manager site.
2. Select your infrastructure provider, and, if applicable, your installation type.
3. In the **Command-line interface** section, select **Windows** from the drop-down menu and click **Download command-line tools**.
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 CLI, it is available using the **oc** command:

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

1.1.2.1.3. Installing the CLI on macOS

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

Procedure

1. Navigate to the [Infrastructure Provider](#) page on the Red Hat OpenShift Cluster Manager site.
2. Select your infrastructure provider, and, if applicable, your installation type.
3. In the **Command-line interface** section, select **MacOS** from the drop-down menu and click **Download command-line tools**.
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 CLI, it is available using the **oc** command:

```
$ oc <command>
```

1.1.2.2. Installing the 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:

```
# subscription-manager register
```

2. Pull the latest subscription data:

```
# subscription-manager refresh
```

3. List the available subscriptions:

```
# 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:

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

5. Enable the repositories required by OpenShift Container Platform 4.4.

- For Red Hat Enterprise Linux 8:

```
# subscription-manager repos --enable="rhocp-4.4-for-rhel-8-x86_64-rpms"
```

- For Red Hat Enterprise Linux 7:

```
# subscription-manager repos --enable="rhel-7-server-ose-4.4-rpms"
```

6. Install the **openshift-clients** package:

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

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

```
$ oc <command>
```

1.1.3. Logging in to the 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
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.
```

- Enter the OpenShift Container Platform server URL.
- Enter whether to use insecure connections.
- Enter the user name to log in as.
- Enter the user's password.

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

1.1.4. Using the CLI

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

1.1.4.1. Creating a project

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

```
$ oc new-project my-project
Now using project "my-project" on server "https://openshift.example.com:6443".
```


1.1.4.2. Creating a new app

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

```
$ oc new-app https://github.com/sclorg/cakephp-ex
--> Found image 40de956 (9 days old) in imagestream "openshift/php" under tag "7.2" for "php"
```

...

Run 'oc status' to view your app.

1.1.4.3. Viewing pods

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

```
$ oc get pods -o wide
NAME          READY  STATUS   RESTARTS  AGE  IP             NODE
NOMINATED NODE
cakephp-ex-1-build  0/1   Completed  0         5m45s  10.131.0.10  ip-10-0-141-74.ec2.internal
<none>
cakephp-ex-1-deploy  0/1   Completed  0         3m44s  10.129.2.9   ip-10-0-147-65.ec2.internal
<none>
cakephp-ex-1-ktz97  1/1   Running    0         3m33s  10.128.2.11  ip-10-0-168-105.ec2.internal
<none>
```

1.1.4.4. Viewing pod logs

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

```
$ oc logs cakephp-ex-1-deploy
--> Scaling cakephp-ex-1 to 1
--> Success
```

1.1.4.5. Viewing the current project

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

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

1.1.4.6. 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.

```
$ oc status
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.
```

1.1.4.7. Listing supported API resources

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

```
$ oc api-resources
NAME                SHORTNAMES  APIGROUP  NAMESPACE  KIND
bindings            true        Binding
componentstatuses   cs          false     ComponentStatus
configmaps           cm          true      ConfigMap
...
```

1.1.5. 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

```
$ oc help
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
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
```

```
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.

FIELDS:

```
apiVersion <string>
```

APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info:

<https://git.k8s.io/community/contributors/devel/api-conventions.md#resources>

```
...
```

1.1.6. Logging out of the CLI

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

- Use the **oc logout** command.

```
$ oc logout
```

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

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

1.2. CONFIGURING THE CLI

1.2.1. 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.

Procedure

The following procedure enables tab completion for Bash.

1. Save the Bash completion code to a file.

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

- Copy the file to **/etc/bash_completion.d/**.

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

You can also 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.

1.3. EXTENDING THE 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.

1.3.1. 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.

- 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**.

- Add the following contents to the file.

```
#!/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 0
fi

echo "I am a plugin named kubectl-foo"

```

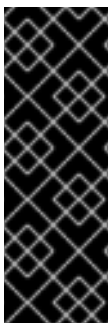
After you install this plug-in for the OpenShift Container Platform CLI, it can be invoked using the **oc foo** command.

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.

1.3.2. 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
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.

1.4. DEVELOPER CLI COMMANDS

1.4.1. Basic CLI commands

1.4.1.1. explain

Display documentation for a certain resource.

Example: Display documentation for pods

```
$ oc explain pods
```

1.4.1.2. 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
```

1.4.1.3. 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
```

1.4.1.4. 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
```

1.4.1.5. project

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

Example: Switch to a different project

```
$ oc project test-project
```

1.4.1.6. projects

Display information about the current active project and existing projects on the server.

Example: List all projects

```
$ oc projects
```

1.4.1.7. status

Show a high-level overview of the current project.

Example: Show the status of the current project

```
$ oc status
```

1.4.2. Build and Deploy CLI commands

1.4.2.1. 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
```

1.4.2.2. import-image

Import the latest tag and image information from an image repository.

Example: Import the latest image information

```
$ oc import-image my-ruby
```

1.4.2.3. 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 https://github.com/sclorg/cakephp-ex
```

1.4.2.4. rollback

Revert an application back to a previous deployment.

Example: Roll back to the last successful deployment

```
$ oc rollback php
```

Example: Roll back to a specific version

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

1.4.2.5. 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

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

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

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

1.4.2.6. start-build

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

Example: Start a build from the specified build config

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

Example: Start a build from a previous build

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

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

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

1.4.2.7. 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

```
$ oc tag ruby:latest ruby:2.0
```

1.4.3. Application management CLI commands

1.4.3.1. 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-
```

1.4.3.2. 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
```

1.4.3.3. autoscale

Autoscale a deployment or replication controller.

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

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

1.4.3.4. 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
```

1.4.3.5. delete

Delete a resource.

Example: Delete a pod named parksmmap-katacoda-1-qfzq4

```
$ oc delete pod/parksmmap-katacoda-1-qfzq4
```

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

```
$ oc delete pods -l app=parksmmap-katacoda
```

1.4.3.6. 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
```

1.4.3.7. edit

Edit a resource.

Example: Edit a deployment using the default editor

```
$ oc edit deploymentconfig/parksmmap-katacoda
```

Example: Edit a deployment using a different editor

```
$ OC_EDITOR="nano" oc edit deploymentconfig/parksmmap-katacoda
```

Example: Edit a deployment in JSON format

```
$ oc edit deploymentconfig/parksmmap-katacoda -o json
```

1.4.3.8. expose

Expose a service externally as a route.

Example: Expose a service

```
$ oc expose service/parksmmap-katacoda
```

Example: Expose a service and specify the host name

```
$ oc expose service/parksmmap-katacoda --hostname=www.my-host.com
```

1.4.3.9. 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

```
$ oc get deploymentconfig/python -o json
```

1.4.3.10. 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
```

1.4.3.11. 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
```

1.4.3.12. 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
```

1.4.3.13. 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
```

1.4.3.14. set

Configure existing application resources.

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

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

1.4.4. Troubleshooting and debugging CLI commands

1.4.4.1. attach

Attach the shell to a running container.

Example: Get output from the python container from pod python-1-mz2rf

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

1.4.4.2. cp

Copy files and directories to and from containers.

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

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

1.4.4.3. debug

Launch a command shell to debug a running application.

Example: Debug the python Deployment

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

1.4.4.4. exec

Execute a command in a container.

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

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

1.4.4.5. logs

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

Example: Stream the latest logs from the `python` deployment

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

1.4.4.6. `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

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

1.4.4.7. `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/`

```
$ oc proxy --port=8011 --www=./local/www/
```

1.4.4.8. `rsh`

Open a remote shell session to a container.

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

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

1.4.4.9. `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

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

1.4.4.10. `run`

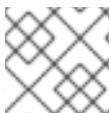
Create and run a particular image. By default, this creates a DeploymentConfig to manage the created containers.

Example: Start an instance of the `perl` image with three replicas

```
$ oc run my-test --image=perl --replicas=3
```

1.4.4.11. `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

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

1.4.5. Advanced developer CLI commands

1.4.5.1. `api-resources`

Display the full list of API resources that the server supports.

Example: List the supported API resources

```
$ oc api-resources
```

1.4.5.2. `api-versions`

Display the full list of API versions that the server supports.

Example: List the supported API versions

```
$ oc api-versions
```

1.4.5.3. `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
```

1.4.5.4. `cluster-info`

Display the address of the master and cluster services.

Example: Display cluster information

```
$ oc cluster-info
```

1.4.5.5. `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
```

1.4.5.6. 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=-
```

1.4.5.7. 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
```

1.4.5.8. 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
```

1.4.5.9. observe

Observe changes to resources and take action on them.

Example: Observe changes to services

```
$ oc observe services
```

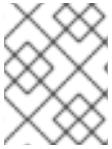
1.4.5.10. patch

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.

1.4.5.11. policy

Manage authorization policies.

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

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

1.4.5.12. process

Process a template into a list of resources.

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

```
$ oc process -f template.json | oc create -f -
```

1.4.5.13. registry

Manage the integrated registry on OpenShift Container Platform.

Example: Display information about the integrated registry

```
$ oc registry info
```

1.4.5.14. 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
```

1.4.6. Settings CLI commands

1.4.6.1. completion

Output shell completion code for the specified shell.

Example: Display completion code for Bash

```
$ oc completion bash
```


1.4.6.2. 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
```

1.4.6.3. logout

Log out of the current session.

Example: End the current session

```
$ oc logout
```

1.4.6.4. whoami

Display information about the current session.

Example: Display the currently authenticated user

```
$ oc whoami
```

1.4.7. Other developer CLI commands

1.4.7.1. 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
```

1.4.7.2. plugin

List the available plug-ins on the user's **PATH**.

Example: List available plug-ins

```
$ oc plugin list
```

1.4.7.3. version

Display the **oc** client and server versions.

Example: Display version information

```
$ oc version
```

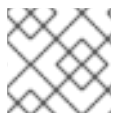
For cluster administrators, the OpenShift Container Platform server version is also displayed.

1.5. ADMINISTRATOR CLI COMMANDS

1.5.1. Cluster management CLI commands

1.5.1.1. inspect

Gather debugging information for a particular resource.



NOTE

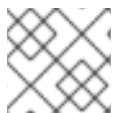
This command is experimental and might change without notice.

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

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

1.5.1.2. 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
```

1.5.1.3. 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
```

1.5.2. Node management CLI commands

1.5.2.1. 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
```

1.5.2.2. drain

Drain a node in preparation for maintenance.

Example: Drain **node1**

```
$ oc adm drain node1
```

1.5.2.3. node-logs

Display and filter node logs.

Example: Get logs for **NetworkManager**

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

1.5.2.4. 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-
```

1.5.2.5. uncordon

Mark a node as schedulable.

Example: Mark **node1** as schedulable

```
$ oc adm uncordon node1
```

1.5.3. Security and policy CLI commands

1.5.3.1. certificate

Approve or reject certificate signing requests (CSRs).

Example: Approve a CSR

```
$ oc adm certificate approve csr-sqgzp
```

1.5.3.2. groups

Manage groups in your cluster.

Example: Create a new group

```
$ oc adm groups new my-group
```

1.5.3.3. 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'
```

1.5.3.4. 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
```

1.5.3.5. 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
```

1.5.4. Maintenance CLI commands

1.5.4.1. 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
```

1.5.4.2. 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
```

1.5.5. Configuration CLI commands**1.5.5.1. create-bootstrap-policy-file**

Create the default bootstrap policy.

Example: Create a file called `policy.json` with the default bootstrap policy

```
$ oc adm create-bootstrap-policy-file --filename=policy.json
```

1.5.5.2. 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
```

1.5.5.3. 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
```

1.5.5.4. 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
```

1.5.5.5. 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
```

1.5.5.6. 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
```

1.5.6. Other Administrator CLI commands

1.5.6.1. build-chain

Output the inputs and dependencies of any builds.

Example: Output dependencies for the perl imagestream

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

1.5.6.2. 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
```

1.5.6.3. 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
```

1.5.6.4. 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:4.4.0-rc.7 \
  quay.io/openshift-release-dev/ocp-release:4.4.0 \
  > changelog.md
```

1.5.6.5. 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
```

1.6. USAGE OF OC AND KUBECTL COMMANDS

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.

1.6.1. 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.

1.6.2. 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 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 2. DEVELOPER CLI (ODO)

2.1. UNDERSTANDING OPENSIFT DO

odo is a CLI tool for creating applications on OpenShift Container Platform and Kubernetes. **odo** allows developers to concentrate on creating applications without the need to administer a cluster itself. Creating deployment configurations, build configurations, service routes and other OpenShift Container Platform or Kubernetes elements are all automated by **odo**.

Existing tools such as **oc** are more operations-focused and require a deep understanding of Kubernetes and OpenShift Container Platform concepts. **odo** abstracts away complex Kubernetes and OpenShift Container Platform concepts allowing developers to focus on what is most important to them: code.

2.1.1. Key features

odo is designed to be simple and concise with the following key features:

- Simple syntax and design centered around concepts familiar to developers, such as projects, applications, and components.
- Completely client based. No additional server other than OpenShift Container Platform is required for deployment.
- Official support for Node.js and Java components.
- Partial compatibility with languages and frameworks such as Ruby, Perl, PHP, and Python.
- Detects changes to local code and deploys it to the cluster automatically, giving instant feedback to validate changes in real time.
- Lists all the available components and services from the cluster.

2.1.2. Core concepts

Project

A project is your source code, tests, and libraries organized in a separate single unit.

Application

An application is a program designed for end users. An application consists of multiple microservices or components that work individually to build the entire application. Examples of applications: a video game, a media player, a web browser.

Component

A component is a set of Kubernetes resources which host code or data. Each component can be run and deployed separately. Examples of components: Node.js, Perl, PHP, Python, Ruby.

Service

A service is software that your component links to or depends on. Examples of services: MariaDB, Jenkins, MySQL. In **odo**, services are provisioned from the OpenShift Service Catalog and must be enabled within your cluster.

2.1.2.1. Officially supported languages and corresponding container images

Table 2.1. Supported languages, container images, and package managers

Language	Container image	Package manager
Node.js	centos/nodejs-8-centos7	NPM
	rhoar-nodejs/nodejs-8	NPM
	bucharestgold/centos7-s2i-nodejs	NPM
	rhsc1/nodejs-8-rhel7	NPM
	rhsc1/nodejs-10-rhel7	NPM
Java	redhat-openjdk-18/openjdk18-openshift	Maven, Gradle
	openjdk/openjdk-11-rhel8	Maven, Gradle
	openjdk/openjdk-11-rhel7	Maven, Gradle

2.1.2.1.1. Listing available container images



NOTE

The list of available container images is sourced from the cluster's internal container registry and external registries associated with the cluster.

To list the available components and associated container images for your cluster:

1. Log in to the cluster with **odo**:

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

2. List the available **odo** supported and unsupported components and corresponding container images:

```
$ odo catalog list components
Odo Supported OpenShift Components:
NAME      PROJECT   TAGS
java      openshift 8,latest
nodejs    openshift 10,8,8-RHOAR,latest

Odo Unsupported OpenShift Components:
NAME      PROJECT   TAGS
dotnet    openshift 1.0,1.1,2.1,2.2,latest
fuse7-eap-openshift openshift 1.3
```

The **TAGS** column represents the available image versions, for example, **10** represents the **rhoar-nodejs/nodejs-10** container image.

2.2. ODO ARCHITECTURE

This section describes **odo** architecture and how **odo** manages resources on a cluster.

2.2.1. 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.

2.2.2. 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.

2.2.3. OpenShift cluster objects

2.2.3.1. 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.

2.2.3.1.1. 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.

2.2.3.1.2. 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.

2.2.3.2. 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.

2.2.3.3. 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`.

Container	PVC mounted at
copy-files-to-volume	/mnt
Application container	/opt/app-root

2.2.3.4. 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.

Container	emptyDir volume mounted at
copy-supervisord	/opt/odo
Application container	/opt/odo

2.2.3.5. 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.

2.2.4. 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.

2.3. INSTALLING ODO

The following section describes how to install **odo** on different platforms using the CLI.



NOTE

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

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**

2.3.1. Installing odo on Linux

2.3.1.1. Binary installation

```
# curl -L https://mirror.openshift.com/pub/openshift-v4/clients/odo/latest/odo-linux-amd64 -o
/usr/local/bin/odo
# chmod +x /usr/local/bin/odo
```

2.3.1.2. Tarball installation

```
# sh -c 'curl -L https://mirror.openshift.com/pub/openshift-v4/clients/odo/latest/odo-linux-amd64.tar.gz
| gzip -d > /usr/local/bin/odo'
# chmod +x /usr/local/bin/odo
```

2.3.2. Installing odo on Windows

2.3.2.1. Binary installation

1. Download the latest [odo.exe](#) file.
2. Add the location of your **odo.exe** to your **GOPATH/bin** directory.

Setting the **PATH** variable for Windows 7/8

The following example demonstrates how to set up a path variable. Your binaries can be located in any location, but this example uses **C:\go-bin** as the location.

1. Create a folder at **C:\go-bin**.
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 the **Advanced systems settings** and click the **Environment Variables** button at the bottom.
5. Select **Path** from the **Variable** section and click **Edit**.
6. Click **New** and type **C:\go-bin** into the field or click **Browse** and select the directory, and click **OK**.

Setting the **PATH** variable for Windows 10

Edit **Environment Variables** using search:

1. Click **Search** and type **env** or **environment**.
2. Select **Edit environment variables for your account**
3. Select **Path** from the **Variable** section and click **Edit**.
4. Click **New** and type **C:\go-bin** into the field or click **Browse** and select the directory, and click **OK**.

2.3.3. Installing odo on macOS

2.3.3.1. Binary installation

```
# curl -L https://mirror.openshift.com/pub/openshift-v4/clients/odo/latest/odo-darwin-amd64 -o
/usr/local/bin/odo
# chmod +x /usr/local/bin/odo
```

2.3.3.2. Tarball installation

■

```
# sh -c 'curl -L https://mirror.openshift.com/pub/openshift-v4/clients/odo/latest/odo-darwin-  
amd64.tar.gz | gzip -d > /usr/local/bin/odo'  
# chmod +x /usr/local/bin/odo
```

2.4. USING ODO IN A RESTRICTED ENVIRONMENT

2.4.1. 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

- [Creating a mirror registry for installation in a restricted network](#)
- [Accessing the registry](#)

2.4.2. 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.

2.4.2.1. 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.

2.4.2.2. 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:

2.4.2.2.1. 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 -d > 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 into the OpenShift Container Platform mirror registry:

```
$ sudo update-ca-trust enable && sudo systemctl daemon-reload && sudo systemctl restart /
docker && 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>
```

2.4.2.2.2. 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:

```
$ echo <content_of_additional_ca> | base64 -d > 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:

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

3. 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>
```

4. 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>
```

2.4.2.2.3. 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:

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

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

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

- Trust a CA in your client platform and log into the OpenShift Container Platform mirror registry:
 - Restart Docker using the Docker UI.
 - Run the following command:

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

- Mirror the **odo** init image:

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

- 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>"
```

2.4.2.3. 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:

2.4.2.3.1. Pushing the **init** image directly on Linux

Procedure

- Enable the default route:

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

- Get a wildcard route CA:

```
$ oc get secret router-certs-default -n openshift-ingress -o yaml
apiVersion: v1
data:
  tls.crt: *****
  tls.key: #####
kind: Secret
metadata:
  [...]
type: kubernetes.io/tls
```

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

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

- 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
```

- Log into 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)
```

- 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>
```

- 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
```

2.4.2.3.2. Pushing the init image directly on MacOS

Procedure

- Enable the default route:

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

- Get a wildcard route CA:

```
$ oc get secret router-certs-default -n openshift-ingress -o yaml
apiVersion: v1
data:
  tls.crt: *****
  tls.key: #####
kind: Secret
metadata:
  [...]
type: kubernetes.io/tls
```

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

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

- Trust a CA in your client platform:

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

- Log into 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)
```

- 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>
```

- 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
```

2.4.2.3.3. Pushing the init image directly on Windows

Procedure

- Enable the default route:

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

- Get a wildcard route CA:

```
PS C:\> oc get secret router-certs-default -n openshift-ingress -o yaml
apiVersion: v1
data:
  tls.crt: *****
  tls.key: #####
kind: Secret
metadata:
  [...]
type: kubernetes.io/tls
```

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

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

- 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
```

- Log into the internal registry:

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

```
PS C:\> docker login <registry_path> -u kubeadmin -p $(oc whoami -t)
```

- Push the **odo** init image:

```
PS C:\> docker pull registry.access.redhat.com/openshiftdo/odo-init-image-rhel7:<tag>
```

```
PS C:\> docker tag registry.access.redhat.com/openshiftdo/odo-init-image-rhel7:<tag>
<registry_path>/openshiftdo/odo-init-image-rhel7:<tag>
```

```
PS C:\> docker push <registry_path>/openshiftdo/odo-init-image-rhel7:<tag>
```

- Override the default **odo** init image path by setting the **ODO_BOOTSTRAPPER_IMAGE** environment variable:

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

2.4.3. 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.

2.4.3.1. 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 .

2.4.3.2. 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
Name:          nodejs
Namespace:     openshift
[...]

10
tagged from <mirror-registry>:<port>/rhoar-nodejs/nodejs-10
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, hidden
Example Repo: https://github.com/sclorg/nodejs-ex.git

! error: Import failed (NotFound): dockerimage.image.openshift.io "<mirror-registry>:
<port>/rhoar-nodejs/nodejs-10:latest" not found
  About an hour ago

10-SCL (latest)
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

! error: Import failed (NotFound): dockerimage.image.openshift.io "<mirror-registry>:
<port>/rhscl/nodejs-10-rhel7:latest" not found
  About an hour ago

[...]
```

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
```

```

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

3 minutes ago

[...]

```

2.4.3.3. 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>
```

2.4.3.4. 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**.

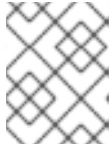
Procedure

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. 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>
```

2.5. CREATING A SINGLE-COMPONENT APPLICATION WITH `odo`

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

2.5.1. Prerequisites

- **odo** is installed.
- You have a running cluster. You can use [CodeReady Containers \(CRC\)](#) to deploy a local cluster quickly.

2.5.2. 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:


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

2.5.3. 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:

```
$ mkdir my_components $$ cd my_components
```

2. Download the example Node.js application:

```
$ git clone https://github.com/openshift/nodejs-ex
```

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

```
$ cd <directory_name>
```

4. 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**.

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>
```

2.5.4. 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.

2.5.5. Adding storage to the application components

Persistent storage keeps data available between restarts of `odo`. You can add storage to your components with the **`odo storage`** command.

Procedure

- Add storage to your components:

```
$ odo storage create nodestorage --path=/opt/app-root/src/storage/ --size=1Gi
```

Your component now has 1 GB storage.

2.5.6. 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`:

```
$ oc annotate istag/openjdk18:latest tags=builder
```

3. Deploy the image with `odo`:

```
$ odo create openjdk18 --git \
https://github.com/openshift-evangelists/Wild-West-Backend
```

2.5.7. 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>
```

2.5.8. Deleting an application



IMPORTANT

Deleting an application will delete all components associated with the application.

Procedure

1. List the applications in the current project:

```
$ odo app list
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
APP   NAME                TYPE   SOURCE   STATE
app   nodejs-nodejs-ex-elyf  nodejs  file:///  Pushed
```

3. Delete the application:

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

2.6. 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.

2.6.1. Prerequisites

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

2.6.2. 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:

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

2.6.3. 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
```

- Run **odo catalog list components** to see the created image:

```
$ odo catalog list components
Odo Supported OpenShift Components:
NAME      PROJECT   TAGS
nodejs    openshift 10,8,8-RHOAR,latest
openjdk18 myproject  latest
```

- Create a directory for your components:

```
$ mkdir my_components $$ cd my_components
```

- Download the example back-end application:

```
$ git clone https://github.com/openshift-evangelists/Wild-West-Backend backend
```

- Change directory to the back-end source directory and check that you have the correct files in the directory:

```
$ cd backend
$ ls
debug.sh pom.xml src
```

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

```
$ mvn package
...
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.635 s
[INFO] Finished at: 2019-09-30T16:11:11-04:00
[INFO] Final Memory: 30M/91M
[INFO] -----
```

- Create a component configuration of Java component-type named **backend**:

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

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

```
$ odo config view
COMPONENT SETTINGS
-----
PARAMETER      CURRENT_VALUE
Type           openjdk18
```

```

Application    app
Project       myproject
SourceType    binary
Ref
SourceLocation target/wildwest-1.0.jar
Ports         8080/TCP,8443/TCP,8778/TCP
Name          backend
MinMemory
MaxMemory
DebugPort
Ignore
MinCPU
MaxCPU

```

10. Push the component to the OpenShift Container Platform cluster.

```

$ odo push
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.

11. Validate:

- The status of the action in odo:

```

odo log -f
2019-09-30 20:14:19.738 INFO 444 --- [          main] c.o.wildwest.WildWestApplication
: Starting WildWestApplication v1.0 onbackend-app-1-9tnhc with PID 444
(/deployments/wildwest-1.0.jar started by jboss in /deployments)

```

- The status of the back-end component:

```

$ odo list
APP  NAME    TYPE    SOURCE                                STATE
app  backend openjdk18  file://target/wildwest-1.0.jar  Pushed

```

2.6.4. 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
```

2. Change the current directory to the front-end directory:

```
$ cd <directory-name>
```

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

```
$ ls
assets bin index.html kwww-frontend.iml package.json package-lock.json playfield.png
README.md server.js
```



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 nodejs frontend
✓ 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
Validation
✓ Checking component [8ms]

Configuration changes
✓ Initializing component
✓ Creating component [83ms]

Pushing to component frontend of type local
✓ Checking files for pushing [2ms]
✓ Waiting for component to start [45s]
✓ Syncing files to the component [3s]
✓ Building component [18s]
✓ Changes successfully pushed to component
```

2.6.5. 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
APP    NAME    TYPE    SOURCE                                STATE
app    backend openjdk18 file://target/wildwest-1.0.jar    Pushed
app    frontend nodejs   file://./                               Pushed
```

2. Link the current front-end component to the back end:

```
$ odo link backend --port 8080
✓ 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.

2.6.6. Exposing components to the public

Procedure

1. Create an external URL for the application:

```
$ cd frontend
$ odo url create frontend --port 8080
✓ URL frontend created for component: frontend
```

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

2. Apply the changes:

```
$ odo push
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.
```

3. 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.

2.6.7. Modifying the running application

Procedure

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

```
$ cd ~/frontend
```

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.

2.6.8. Deleting an application

**IMPORTANT**

Deleting an application will delete all components associated with the application.

Procedure

1. List the applications in the current project:

```
$ odo app list
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
APP   NAME                TYPE   SOURCE   STATE
app   nodejs-nodejs-ex-elyf  nodejs file:///  Pushed
```

3. Delete the application:

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

2.7. CREATING AN APPLICATION WITH A DATABASE

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

2.7.1. 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.
- [Service Catalog](#) is enabled.

2.7.2. 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:

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

2.7.3. 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
```

2. Change the current directory to the front-end directory:

```
$ cd <directory-name>
```

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

```
$ ls
assets bin index.html kwww-frontend.iml package.json package-lock.json playfield.png
README.md server.js
```



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 nodejs frontend
✓ 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
✓ URL myurl created for component: nodejs-nodejs-ex-pmdp
```

6. Push the component to the OpenShift Container Platform cluster.

```
$ odo push
Validation
✓ Checking component [7ms]

Configuration changes
✓ Initializing component
✓ Creating component [134ms]

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

Pushing to component nodejs-nodejs-ex-mhbb of type local
✓ Checking files for pushing [657850ns]
✓ Waiting for component to start [6s]
```

- ✓ Syncing files to the component [408ms]
- ✓ Building component [7s]
- ✓ Changes successfully pushed to component

2.7.4. 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
? 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
? Output the non-interactive version of the selected options 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.

2.7.5. Deploying a database manually

1. List the available services:

```
$ odo catalog list services
NAME                PLANS
django-psql-persistent  default
jenkins-ephemeral     default
jenkins-pipeline-example default
mariadb-persistent    default
mongodb-persistent    default
mysql-persistent      default
nodejs-mongo-persistent default
postgresql-persistent default
rails-pgsq-persistent default
```

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

```

$ odo catalog describe service mongodb-persistent
***** | *****
Name | default
----- | -----
Display Name |
----- | -----
Short Description | Default plan
----- | -----
Required Params without a |
default value |
----- | -----
Required Params with a default | DATABASE_SERVICE_NAME
value | (default: 'mongodb'),
| MEMORY_LIMIT (default:
| '512Mi'), MONGODB_VERSION
| (default: '3.2'),
| MONGODB_DATABASE (default:
| 'sampledb'), VOLUME_CAPACITY
| (default: '1Gi')
----- | -----
Optional Params | MONGODB_ADMIN_PASSWORD,
| NAMESPACE, MONGODB_PASSWORD,
| MONGODB_USER

```

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

```

$ 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

```

2.7.6. Connecting the database to the front-end application

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

```

$ odo link mongodb-persistent
✓ 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:

```

$ oc get pods
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

$ oc rsh nodejs-nodejs-ex-mhbb-app-4-vkn9l
sh-4.2$ env

```

```
uri=mongodb://172.30.126.3:27017
password=dHIOPYneSkX3rTLn
database_name=sampled
username=user43U
admin_password=NCn41tqmx7Rlqmfv
sh-4.2$
```

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

```
$ odo url list
```

```
Request information
Page view count: 24
```

```
DB Connection Info:
Type: MongoDB
URL: mongodb://172.30.126.3:27017/sampled
```

2.7.7. Deleting an application



IMPORTANT

Deleting an application will delete all components associated with the application.

Procedure

- List the applications in the current project:

```
$ odo app list
The project '<project_name>' has the following applications:
NAME
app
```

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

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

- Delete the application:

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

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

2.8. CREATING APPLICATIONS BY USING DEVFILES



IMPORTANT

Creating applications by using devfiles with `odo` is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see <https://access.redhat.com/support/offerings/techpreview/>.

2.8.1. 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.

2.8.2. Creating a Java application by using a devfile

2.8.3. 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](#).
- You have enabled Experimental Mode in **odo**.
 - To enable Experimental Mode in **odo** preferences, run **odo preference set Experimental true** or use the environment variable **odo config set --env ODO_EXPERIMENTAL=true**

2.8.3.1. 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
✓ Project 'myproject' is ready for use
✓ New project created and now using project : myproject
```

2.8.3.2. 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:

```
$ odo catalog list components was
```

The output lists the available **odo** components:

```
Odo OpenShift Components:
NAME          PROJECT   TAGS                SUPPORTED
java          openshift 11,8,latest        YES
nodejs        openshift 10-SCL,8,8-RHOAR,latest YES
dotnet        openshift 2.1,2.2,3.0,latest NO
golang        openshift 1.11.5,latest      NO
httpd         openshift 2.4,latest          NO
modern-webapp openshift 10.x,latest         NO
nginx         openshift 1.10,1.12,latest   NO
perl          openshift 5.24,5.26,latest   NO
php           openshift 7.0,7.1,7.2,latest NO
python        openshift 2.7,3.6,latest     NO
ruby          openshift 2.4,2.5,latest     NO

Odo Devfile Components:
NAME          DESCRIPTION                SUPPORTED
maven         Upstream Maven and OpenJDK 11 YES
nodejs        Stack with NodeJS 10       YES
openLiberty   Open Liberty microservice in Java YES
java-spring-boot Spring Boot® using Java    YES
```

2.8.3.3. 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 --downloadSource
```


The previous command produces the following output:

```
Experimental mode is enabled, use at your own risk

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.

- 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
```

- 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.

- 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:

```
$ odo url list
```

The previous command produces the following output:

```
Found the following URLs for component myspring
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
```

2.9. USING SAMPLE APPLICATIONS

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

```
NAME    PROJECT    TAGS
dotnet  openshift  2.0,latest
httpd   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
perl    openshift  5.16,5.20,5.24,latest
php     openshift  5.5,5.6,7.0,7.1,latest
python  openshift  2.7,3.3,3.4,3.5,3.6,latest
ruby    openshift  2.0,2.2,2.3,2.4,latest
wildfly openshift  10.0,10.1,8.1,9.0,latest
```



NOTE

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**.

2.9.1. Examples from Git repositories

2.9.1.1. 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](#).

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

2.9.1.2. 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](#).

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

2.9.1.3. 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](#).

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

2.9.1.4. 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](#).

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

2.9.1.5. 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
```

2.9.1.6. 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
```

2.9.1.7. 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
```

2.9.1.8. wildfly

This example helps build and run WildFly applications on CentOS 7. For more information about using this builder image, including OpenShift Container Platform considerations, see the [Wildfly - CentOS Docker images for OpenShift](#).

```
$ odo create wildfly --git https://github.com/openshift/openshift-jee-sample.git
```

2.9.2. Binary examples

2.9.2.1. java

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

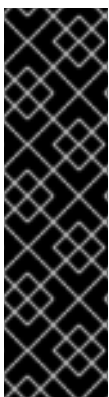
```
$ git clone https://github.com/spring-projects/spring-petclinic.git
$ cd spring-petclinic
$ mvn package
$ odo create java test3 --binary target/*.jar
$ odo push
```

2.9.2.2. wildfly

WildFly can be used to deploy a binary application as follows:

```
$ git clone https://github.com/openshift demos/os-sample-java-web.git
$ cd os-sample-java-web
$ mvn package
$ cd ..
$ mkdir example && cd example
$ mv ../os-sample-java-web/target/ROOT.war example.war
$ odo create wildfly --binary example.war
```

2.10. CREATING INSTANCES OF SERVICES MANAGED BY OPERATORS



IMPORTANT

Creating instances of services managed by Operators in `odo` is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see <https://access.redhat.com/support/offerings/techpreview/>.

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](#).

2.10.1. Prerequisites

- Install the **oc** CLI and log into 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.
- Enable experimental mode. To enable experimental mode in **odo**, run: **odo preference set Experimental true** or use the environment variable **odo config set --env ODO_EXPERIMENTAL=true**

2.10.2. 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:

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

2.10.3. 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:

```
$ 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.

2.10.4. 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:

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

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

```
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:

```
$ odo service create etcdoperator.v0.9.4 --crd EtcdCluster
```

4. Verify that a service has started:

```
$ oc get EtcdCluster
```

2.10.5. 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> --crd <cr-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 --crd EtcdCluster --dry-run
```

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

2. Modify the **etcd.yaml** file:

```
apiVersion: etcd.database.coreos.com/v1beta2
kind: EtcdCluster
metadata:
  name: my-etcd-cluster 1
```

```
spec:
  size: 1 2
  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
```

2.11. DEBUGGING APPLICATIONS IN **odo**



IMPORTANT

Interactive debugging in **odo** is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see <https://access.redhat.com/support/offerings/techpreview/>.

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).

2.11.1. Debugging an application

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

Procedure

1. After an application is deployed, start the port forwarding for your component to debug the application:

```
$ odo debug port-forward
```

2. Attach the debugger bundled in your IDE to the component. Instructions vary depending on your IDE.

2.11.2. 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:

```
$ 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:

```
$ 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.

2.12. 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.

2.12.1. 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
```

2.13. CONFIGURING THE ODO CLI

2.13.1. Using command completion

**NOTE**

Currently command completion is only supported for bash, zsh, and fish shells.

odo provides a smart completion of command parameters based on user input. For this to work, odo needs to integrate with the executing shell.

Procedure

- To install command completion automatically:

1. Run:

```
$ odo --complete
```

2. Press **y** when prompted to install the completion hook.

- To install the completion hook manually, add **complete -o nospace -C <full_path_to_your_odo_binary> odo** to your shell configuration file. After any modification to your shell configuration file, restart your shell.

- To disable completion:

1. Run:

```
$ odo --uncomplete
```

2. Press **y** when prompted to uninstall the completion hook.

**NOTE**

Re-enable command completion if you either rename the odo executable or move it to a different directory.

2.13.2. 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.

2.14. ODO CLI REFERENCE

2.14.1. Basic `odo` CLI commands

2.14.1.1. `app`

Perform application operations related to your OpenShift Container Platform project.

Example using `app`

```
# Delete the application
odo app delete myapp

# Describe 'webapp' application,
odo app describe webapp

# List all applications in the current project
odo app list

# List all applications in the specified project
odo app list --project myproject
```

2.14.1.2. `catalog`

Perform catalog-related operations.

Example using `catalog`

```
# Get the supported components
odo catalog list components

# Get the supported services from service catalog
odo catalog list services

# Search for a component
odo catalog search component python

# Search for a service
odo catalog search service mysql

# Describe a service
odo catalog describe service mysql-persistent
```

2.14.1.3. `component`

Manage components of an application.

Example using `component`

```
# Create a new component
odo component create

# Create a local configuration and create all objects on the cluster
odo component create --now
```

2.14.1.4. config

Modify **odo** specific settings within the **config** file.

Example using config

```
# For viewing the current local configuration
odo config view

# Set a configuration value in the local configuration
odo config set Type java
odo config set Name test
odo config set MinMemory 50M
odo config set MaxMemory 500M
odo config set Memory 250M
odo config set Ignore false
odo config set MinCPU 0.5
odo config set MaxCPU 2
odo config set CPU 1

# Set an environment variable in the local configuration
odo config set --env KAFKA_HOST=kafka --env KAFKA_PORT=6639

# Create a local configuration and apply the changes to the cluster immediately
odo config set --now

# Unset a configuration value in the local config
odo config unset Type
odo config unset Name
odo config unset MinMemory
odo config unset MaxMemory
odo config unset Memory
odo config unset Ignore
odo config unset MinCPU
odo config unset MaxCPU
odo config unset CPU

# Unset an env variable in the local config
odo config unset --env KAFKA_HOST --env KAFKA_PORT
```

Application	Application is the name of application the component needs to be part of
CPU	The minimum and maximum CPU a component can consume
Ignore	Consider the .odoignore file for push and watch

Table 2.2. Available Local Parameters:

Application	The name of application that the component needs to be part of
-------------	--

CPU	The minimum and maximum CPU a component can consume
Ignore	Whether to consider the .odoignore file for push and watch
MaxCPU	The maximum CPU a component can consume
MaxMemory	The maximum memory a component can consume
Memory	The minimum and maximum memory a component can consume
MinCPU	The minimum CPU a component can consume
MinMemory	The minimum memory a component is provided
Name	The name of the component
Ports	Ports to be opened in the component
Project	The name of the project that the component is part of
Ref	Git ref to use for creating component from git source
SourceLocation	The path indicates the location of binary file or git source
SourceType	Type of component source - git/binary/local
Storage	Storage of the component
Type	The type of component
Url	The URL to access the component

2.14.1.5. create

Create a configuration describing a component to be deployed on OpenShift Container Platform. If a component name is not provided, it is autogenerated.

By default, builder images are used from the current namespace. To explicitly supply a namespace, use: **odo create namespace/name:version**. If a version is not specified, the version defaults to **latest**.

Use **odo catalog list** to see a full list of component types that can be deployed.

Example using create

-

```

# Create new Node.js component with the source in current directory.
odo create nodejs

# Create new Node.js component and push it to the cluster immediately.
odo create nodejs --now

# A specific image version may also be specified
odo create nodejs:latest

# Create new Node.js component named 'frontend' with the source in './frontend' directory
odo create nodejs frontend --context ./frontend

# Create a new Node.js component of version 6 from the 'openshift' namespace
odo create openshift/nodejs:6 --context /nodejs-ex

# Create new Wildfly component with binary named sample.war in './downloads' directory
odo create wildfly wildfly --binary ./downloads/sample.war

# Create new Node.js component with source from remote git repository
odo create nodejs --git https://github.com/openshift/nodejs-ex.git

# Create new Node.js git component while specifying a branch, tag or commit ref
odo create nodejs --git https://github.com/openshift/nodejs-ex.git --ref master

# Create new Node.js git component while specifying a tag
odo create nodejs --git https://github.com/openshift/nodejs-ex.git --ref v1.0.1

# Create new Node.js component with the source in current directory and ports 8080-tcp,8100-tcp
and 9100-udp exposed
odo create nodejs --port 8080,8100/tcp,9100/udp

# Create new Node.js component with the source in current directory and env variables key=value
and key1=value1 exposed
odo create nodejs --env key=value,key1=value1

# For more examples, visit: https://github.com/openshift/odo/blob/master/docs/examples.adoc
odo create python --git https://github.com/openshift/django-ex.git

# Passing memory limits
odo create nodejs --memory 150Mi
odo create nodejs --min-memory 150Mi --max-memory 300 Mi

# Passing cpu limits
odo create nodejs --cpu 2
odo create nodejs --min-cpu 200m --max-cpu 2

```

2.14.1.6. debug

Debug a component.

```

# Displaying information about the state of debugging
odo debug info

# Starting the port forwarding for a component to debug the application
odo debug port-forward

```

```
# Setting a local port to port forward
odo debug port-forward --local-port 9292
```

2.14.1.7. delete

Delete an existing component.

Example using delete

```
# Delete component named 'frontend'.
odo delete frontend
odo delete frontend --all-apps
```

2.14.1.8. describe

Describe the given component.

Example using describe

```
# Describe nodejs component
odo describe nodejs
```

2.14.1.9. link

Link a component to a service or component.

Example using link

```
# Link the current component to the 'my-postgresql' service
odo link my-postgresql

# Link component 'nodejs' to the 'my-postgresql' service
odo link my-postgresql --component nodejs

# Link current component to the 'backend' component (backend must have a single exposed port)
odo link backend

# Link component 'nodejs' to the 'backend' component
odo link backend --component nodejs

# Link current component to port 8080 of the 'backend' component (backend must have port 8080
exposed)
odo link backend --port 8080
```

Link adds the appropriate secret to the environment of the source component. The source component can then consume the entries of the secret as environment variables. If the source component is not provided, the current active component is assumed.

2.14.1.10. list

List all the components in the current application.

Example using list

```
# List all components in the application
odo list
```

2.14.1.11. log

Retrieve the log for the given component.

Example using log

```
# Get the logs for the nodejs component
odo log nodejs
```

2.14.1.12. login

Log in to the cluster.

Example using login

```
# Log in interactively
odo login

# Log in to the given server with the given certificate authority file
odo login localhost:8443 --certificate-authority=/path/to/cert.crt

# Log in to the given server with the given credentials (basic auth)
odo login localhost:8443 --username=myuser --password=mypass

# Log in to the given server with the given credentials (token)
odo login localhost:8443 --token=xxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

2.14.1.13. logout

Log out of the current OpenShift Container Platform session.

Example using logout

```
# Log out
odo logout
```

2.14.1.14. preference

Modify **odo** specific configuration settings within the global preference file.

Example using preference

```
# For viewing the current preferences
odo preference view

# Set a preference value in the global preference
odo preference set UpdateNotification false
```

```

odo preference set NamePrefix "app"
odo preference set Timeout 20

# Enable experimental mode
odo preference set experimental true

# Unset a preference value in the global preference
odo preference unset UpdateNotification
odo preference unset NamePrefix
odo preference unset Timeout

# Disable experimental mode
odo preference set experimental false

```



NOTE

By default, the path to the global preference file is `~/.odo/preference.yaml` and it is stored in the environment variable **GLOBALODOCONFIG**. You can set up a custom path by setting the value of the environment variable to a new preference path, for example **GLOBALODOCONFIG="new_path/preference.yaml"**

Table 2.3. Available Parameters:

NamePrefix	The default prefix is the current directory name. Use this value to set a default name prefix.
Timeout	The timeout (in seconds) for OpenShift Container Platform server connection checks.
UpdateNotification	Controls whether an update notification is shown.

2.14.1.15. project

Perform project operations.

Example using project

```

# Set the active project
odo project set

# Create a new project
odo project create myproject

# List all the projects
odo project list

# Delete a project
odo project delete myproject

# Get the active project
odo project get

```


2.14.1.16. push

Push source code to a component.

Example using push

```
# Push source code to the current component
odo push

# Push data to the current component from the original source.
odo push

# Push source code in ~/mycode to component called my-component
odo push my-component --context ~/mycode
```

2.14.1.17. service

Perform service catalog operations.

Example using service

```
# Create new postgresql service from service catalog using dev plan and name my-postgresql-db.
odo service create dh-postgresql-apb my-postgresql-db --plan dev -p postgresql_user=luke -p
postgresql_password=secret

# Delete the service named 'mysql-persistent'
odo service delete mysql-persistent

# List all services in the application
odo service list
```

2.14.1.18. storage

Perform storage operations.

Example using storage

```
# Create storage of size 1Gb to a component
odo storage create mystorage --path=/opt/app-root/src/storage/ --size=1Gi

# Delete storage mystorage from the currently active component
odo storage delete mystorage

# Delete storage mystorage from component 'mongodb'
odo storage delete mystorage --component mongodb

# List all storage attached or mounted to the current component and
# all unattached or unmounted storage in the current application
odo storage list

# Set the `-o json` flag to get a JSON formatted output
odo storage list -o json
```

2.14.1.19. unlink

Unlink component or a service.

For this command to be successful, the service or component must have been linked prior to the invocation using **odo link**.

Example using unlink

```
# Unlink the 'my-postgresql' service from the current component
odo unlink my-postgresql

# Unlink the 'my-postgresql' service from the 'nodejs' component
odo unlink my-postgresql --component nodejs

# Unlink the 'backend' component from the current component (backend must have a single
exposed port)
odo unlink backend

# Unlink the 'backend' service from the 'nodejs' component
odo unlink backend --component nodejs

# Unlink the backend's 8080 port from the current component
odo unlink backend --port 8080
```

2.14.1.20. update

Update the source code path of a component

Example using update

```
# Change the source code path of a currently active component to local (use the current directory as
a source)
odo update --local

# Change the source code path of the frontend component to local with source in ./frontend directory
odo update frontend --local ./frontend

# Change the source code path of a currently active component to git
odo update --git https://github.com/openshift/nodejs-ex.git

# Change the source code path of the component named node-ex to git
odo update node-ex --git https://github.com/openshift/nodejs-ex.git

# Change the source code path of the component named wildfly to a binary named sample.war in
./downloads directory
odo update wildfly --binary ./downloads/sample.war
```

2.14.1.21. url

Expose a component to the outside world.

Example using url

```

# Create a URL for the current component with a specific port
odo url create --port 8080

# Create a URL with a specific name and port
odo url create example --port 8080

# Create a URL with a specific name by automatic detection of port (only for components which
expose only one service port)
odo url create example

# Create a URL with a specific name and port for component frontend
odo url create example --port 8080 --component frontend

# Delete a URL to a component
odo url delete myurl

# List the available URLs
odo url list

# Create a URL in the configuration and apply the changes to the cluster
odo url create --now

# Create an HTTPS URL
odo url create --secure

```

The URLs that are generated using this command can be used to access the deployed components from outside the cluster.

2.14.1.22. utils

Utilities for terminal commands and modifying odo configurations.

Example using utils

```

# Bash terminal PS1 support
source <(odo utils terminal bash)

# Zsh terminal PS1 support
source <(odo utils terminal zsh)

```

2.14.1.23. version

Print the client version information.

Example using version

```

# Print the client version of odo
odo version

```

2.14.1.24. watch

odo starts watching for changes and updates the component upon a change automatically.

Example using watch

```
# Watch for changes in directory for current component
odo watch

# Watch for changes in directory for component called frontend
odo watch frontend
```

2.15. odo RELEASE NOTES

2.15.1. Notable changes and improvements in odo

- A **--now** flag is added for **odo create**, **odo url**, and **odo config**.
Run **odo url create --now** to create the URL in the configuration and propagate all the changes to the cluster.
- An **odo debug info** command is added. The command displays if the debug mode is enabled for a component, if the port-forward process is running, and information about ports used.
- The **odo push** output now shows the exact error from the OpenShift cluster on failure.
- A **--secure** flag is added for **odo url**. It indicates whether a URL is secure.
- **odo storage list** now displays the information about the storage state: **Pushed**, **NotPushed** or **Locally Deleted**.
- **odo debug port-forward** now selects a port automatically if the default port (5858) is occupied.
- The **--all** flag is renamed to **--all-apps**.
- The default PVC size has been increased to 10GiB.
- The JSON output for **odo storage list -o json** has been restructured.
- **nodejs-8** and **nodejs-10** images are no longer supported.
- Experimental mode feature. By default, features under development or in experimental mode are hidden from the user.

- To enable experimental mode, run:

```
$ odo config set --env ODO_EXPERIMENTAL=true
```

or

```
$ odo preference set experimental true
```

- To disable experimental mode, run:

```
$ odo config unset --env ODO_EXPERIMENTAL
```

or

```
$ odo preference set experimental false
```

- **odo** now supports Ingress to create URLs on Kubernetes.

2.15.2. Getting support

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.

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.

2.15.3. Fixed issues

- [Bug 1760575](#) The **odo app delete** command removes application components but not Services.
- [Bug 1760577](#) The **odo push** command does not delete the OpenShift objects when the component name is changed.

2.15.4. Known issues

- [Bug 1760574](#) A deleted namespace is listed in the **odo project get** command.
- [Bug 1760586](#) The **odo delete** command starts an infinite loop after a project is deleted and a component name is set.
- [Bug 1760588](#) The **odo service create** command crashes when run in Cygwin.
- [Bug 1760590](#) In Git BASH for Windows, the **odo login -u developer** command does not hide a typed password when requested.
- [Bug 1783188](#) In a disconnected cluster, the **odo component create** command throws an error **... tag not found...** despite the component being listed in the catalog list.
- [Bug 1761440](#) It is not possible to create two Services of the same type in one project.
- [Bug 1821643](#) **odo push** does not work on the .NET component tag 2.1+.
Workaround: specify your .NET project file by running:

```
$ odo config set --env DOTNET_STARTUP_PROJECT=<path to your project file>
```

2.15.5. Technology Preview features odo

- **odo debug** is a feature that allows users to attach a local debugger to a component running in the Pod. To learn more, see [Debugging applications in odo](#)



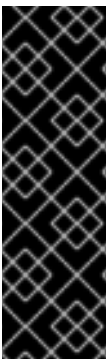
IMPORTANT

`odo debug` is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see <https://access.redhat.com/support/offerings/techpreview/>.

- Devfile support. You can now create and deploy your applications using the devfiles with **odo**. A devfile is a file that defines the development environment: environment variables, image, and so on. To access this feature, you must enable experimental mode with **odo preference set experimental true**.

To see the list of currently supported devfile components, run **odo catalog list components**



IMPORTANT

Devfile support is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see <https://access.redhat.com/support/offerings/techpreview/>.

CHAPTER 3. HELM CLI

3.1. GETTING STARTED WITH HELM 3 ON OPENSIFT CONTAINER PLATFORM

3.1.1. Understanding Helm

Helm is a command-line interface (CLI) tool 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 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.

3.1.1.1. 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.

3.1.2. 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.

3.1.2.1. On Linux

1. Download the Helm binary and add it to your path:

```
# curl -L https://mirror.openshift.com/pub/openshift-v4/clients/helm/latest/helm-linux-amd64 -  
o /usr/local/bin/helm
```

2. Make the binary file executable:

```
# chmod +x /usr/local/bin/helm
```

3. Check the installed version:

```
$ helm version  
version.BuildInfo{Version:"v3.0",  
GitCommit:"b31719aab7963acf4887a1c1e6d5e53378e34d93", GitTreeState:"clean",  
GoVersion:"go1.13.4"}
```

3.1.2.2. 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**.

3.1.2.3. 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**.

3.1.2.4. On MacOS

1. Download the Helm binary and add it to your path:

```
# curl -L https://mirror.openshift.com/pub/openshift-v4/clients/helm/latest/helm-darwin-amd64  
-o /usr/local/bin/helm
```

2. Make the binary file executable:

```
# chmod +x /usr/local/bin/helm
```

3. Check the installed version:

```
$ helm version  
version.BuildInfo{Version:"v3.0",  
GitCommit:"b31719aab7963acf4887a1c1e6d5e53378e34d93", GitTreeState:"clean",  
GoVersion:"go1.13.4"}
```

3.1.3. 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
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
```

3.1.4. Creating a custom Helm chart on OpenShift Container Platform

Procedure

1. Create a new project:

```
$ oc new-project nodejs-ex-k
```

2. Download an example Node.js chart that contains OpenShift Container Platform objects:

```
$ git clone https://github.com/redhat-developer/redhat-helm-charts
```

3. Go to the directory with the sample chart:

```
$ cd redhat-helm-charts/alpha/nodejs-ex-k/
```

4. Edit the **Chart.yaml** file and add a description of your chart:

```
apiVersion: v2 1
name: nodejs-ex-k 2
description: A Helm chart for OpenShift 3
icon: https://static.redhat.com/libs/redhat/brand-assets/latest/corp/logo.svg 4
```

- 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.

5. Verify that the chart is formatted properly:

```
$ helm lint
[INFO] Chart.yaml: icon is recommended

1 chart(s) linted, 0 chart(s) failed
```

6. Install the chart:

```
$ cd ..
$ helm install nodejs-chart nodejs-ex-k
```

7. Verify that the chart has installed successfully:

```
$ helm list
NAME NAMESPACE REVISION UPDATED STATUS CHART APP VERSION
nodejs-chart nodejs-ex-k 1 2019-12-05 15:06:51.379134163 -0500 EST deployed nodejs-
0.1.0 1.16.0
```

CHAPTER 4. KNATIVE CLI (KN) FOR USE WITH OPENSIFT SERVERLESS

kn enables simple interaction with Knative components on OpenShift Container Platform.

You can enable Knative on OpenShift Container Platform by installing OpenShift Serverless. For more information, see the documentation on [Getting started with OpenShift Serverless](#).



NOTE

OpenShift Serverless cannot be installed using the **kn** CLI. A cluster administrator must install the OpenShift Serverless Operator and set up the Knative components, as described in the [Serverless applications](#) documentation for OpenShift Container Platform.

4.1. KEY FEATURES

kn is designed to make serverless computing tasks simple and concise. Key features of **kn** 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.



NOTE

Knative Eventing is currently available as a Technology Preview feature of OpenShift Serverless.

- Create [sink binding](#) to connect existing Kubernetes applications and Knative services.
- Extend **kn** with flexible plugin architecture, similar to **kubectl**.
- Configure [autoscaling](#) parameters for Knative services.
- Scripted usage, such as waiting for the results of an operation, or deploying custom rollout and rollback strategies.

4.2. INSTALLING KN

For information about installing **kn** for use with OpenShift Serverless, see the documentation on [Installing the Knative CLI \(kn\)](#).

CHAPTER 5. PIPELINES CLI (TKN)

5.1. INSTALLING TKN

Use the **tkn** CLI to manage 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**.

5.1.1. Installing OpenShift Pipelines CLI (tkn) on Linux

For Linux distributions, you can download the CLI directly as a **tar.gz** archive.

Procedure

1. Download the [CLI](#).
2. Unpack the archive:

```
$ tar xvfz <file>
```

3. Place the **tkn** binary in a directory that is on your **PATH**.
4. To check your **PATH**, run:

```
$ echo $PATH
```

5.1.2. Installing OpenShift Pipelines CLI (tkn) on Linux using an RPM

For Red Hat Enterprise Linux (RHEL) version 8, you can install the 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:

```
# subscription-manager register
```

2. Pull the latest subscription data:

```
# subscription-manager refresh
```

3. List the available subscriptions:

```
# 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:

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

5. Enable the repositories required by OpenShift Pipelines:

```
# subscription-manager repos --enable="pipelines-1.0-for-rhel-8-x86_64-rpms"
```

6. Install the **openshift-pipelines-client** package:

```
# yum install openshift-pipelines-client
```

After you install the CLI, it is available using the **tkn** command:

```
$ tkn version
```

5.1.3. Installing 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:

```
C:\> path
```

5.1.4. Installing 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:

```
$ echo $PATH
```

5.2. CONFIGURING THE OPENSIFT PIPELINES TKN CLI

Configure the OpenShift Pipelines **tkn** CLI to enable tab completion.

5.2.1. 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.

5.3. OPENSIFT PIPELINES TKN REFERENCE

This section lists the basic **tkn** CLI commands.

5.3.1. Basic syntax

tkn [command or options] [arguments...]

5.3.2. Global options

--help, -h

5.3.3. Utility commands

5.3.3.1. tkn

Parent command for **tkn** CLI.

Example: Display all options

```
$ tkn
```

5.3.3.2. 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
```

5.3.3.3. version

Print version information of the **tkn** CLI.

Example: Check the **tkn** version

```
$ tkn version
```

5.3.4. Pipelines management commands

5.3.4.1. pipeline

Manage Pipelines.

Example: Display help

```
$ tkn pipeline --help
```

5.3.4.2. pipeline create

Create a Pipeline.

Example: Create a Pipeline defined by the **mypipeline.yaml** file in a namespace

```
$ tkn pipeline create -f mypipeline.yaml -n myspace
```

5.3.4.3. pipeline delete

Delete a Pipeline.

Example: Delete the **mypipeline** Pipeline from a namespace

```
$ tkn pipeline delete mypipeline -n myspace
```

5.3.4.4. pipeline describe

Describe a Pipeline.

Example: Describe **mypipeline** Pipeline

```
$ tkn pipeline describe mypipeline
```

5.3.4.5. pipeline list

List Pipelines.

Example: Display a list of Pipelines

```
$ tkn pipeline list
```

5.3.4.6. pipeline logs

Display Pipeline logs for a specific Pipeline.

Example: Stream live logs for the mypipeline Pipeline

```
$ tkn pipeline logs -f mypipeline
```

5.3.4.7. pipeline start

Start a Pipeline.

Example: Start mypipeline Pipeline

```
$ tkn pipeline start mypipeline
```

5.3.5. PipelineRun commands

5.3.5.1. pipelinerun

Manage PipelineRuns.

Example: Display help

```
$ tkn pipelinerun -h
```

5.3.5.2. pipelinerun cancel

Cancel a PipelineRun.

Example: Cancel the mypipelinerun PipelineRun from a namespace

```
$ tkn pipelinerun cancel mypipelinerun -n myspace
```

5.3.5.3. pipelinerun delete

Delete a PipelineRun.

Example: Delete PipelineRuns from a namespace

```
$ tkn pipelinerun delete mypipelinerun1 mypipelinerun2 -n myspace
```

5.3.5.4. pipelinerun describe

Describe a PipelineRun.

Example: Describe the mypipelinerun PipelineRun in a namespace

```
tkn pipelinerun describe mypipelinerun -n myspace
```

5.3.5.5. pipelinerun list

List PipelineRuns.

Example: Display a list of PipelineRuns in a namespace

```
$ tkn pipelinerun list -n myspace
```

5.3.5.6. pipelinerun logs

Display the logs of a PipelineRun.

Example: Display the logs of the mypipelinerun PipelineRun with all tasks and steps in a namespace

```
$ tkn pipelinerun logs mypipelinerun -a -n myspace
```

5.3.6. Task management commands

5.3.6.1. task

Manage Tasks.

Example: Display help

```
$ tkn task -h
```

5.3.6.2. task create

Create a Task.

Example: Create a Task defined by the mytask.yaml file in a namespace

```
$ tkn task create -f mytask.yaml -n myspace
```

5.3.6.3. task delete

Delete a Task.

Example: Delete mytask1 and mytask2 Tasks from a namespace

```
$ tkn task delete mytask1 mytask2 -n myspace
```

5.3.6.4. task describe

Describe a Task.

Example: Describe the mytask Task in a namespace

```
$ tkn task describe mytask -n myspace
```

5.3.6.5. task list

List Tasks.

Example: List all the Tasks in a namespace

```
$ tkn task list -n myspace
```

5.3.6.6. task logs

Display Task logs.

Example: Display logs for the mytaskrun TaskRun of the mytask Task

```
$ tkn task logs mytask mytaskrun -n myspace
```

5.3.6.7. task start

Start a Task.

Example: Start the mytask Task in a namespace

```
$ tkn task start mytask -s <ServiceAccountName> -n myspace
```

5.3.7. TaskRun commands

5.3.7.1. taskrun

Manage TaskRuns.

Example: Display help

```
$ tkn taskrun -h
```

5.3.7.2. taskrun cancel

Cancel a TaskRun.

Example: Cancel the mytaskrun TaskRun from a namespace

```
$ tkn taskrun cancel mytaskrun -n myspace
```

5.3.7.3. taskrun delete

Delete a TaskRun.

Example: Delete mytaskrun1 and mytaskrun2 TaskRuns from a namespace

```
$ tkn taskrun delete mytaskrun1 mytaskrun2 -n myspace
```

5.3.7.4. taskrun describe

Describe a TaskRun.

Example: Describe the mytaskrun TaskRun in a namespace

```
$ tkn taskrun describe mytaskrun -n myspace
```

5.3.7.5. taskrun list

List TaskRuns.

Example: List all TaskRuns in a namespace

```
$ tkn taskrun list -n myspace
```

5.3.7.6. taskrun logs

Display TaskRun logs.

Example: Display live logs for the mytaskrun TaskRun in a namespace

```
$ tkn taskrun logs -f mytaskrun -n myspace
```

5.3.8. Condition management commands

5.3.8.1. condition

Manage Conditions.

Example: Display help

```
$ tkn condition --help
```

5.3.8.2. condition delete

Delete a Condition.

Example: Delete the mycondition1 Condition from a namespace

```
$ tkn condition delete mycondition1 -n myspace
```

5.3.8.3. condition describe

Describe a Condition.

Example: Describe the mycondition1 Condition in a namespace

```
$ tkn condition describe mycondition1 -n myspace
```

5.3.8.4. condition list

List Conditions.

Example: List Conditions in a namespace

```
$ tkn condition list -n myspace
```

5.3.9. Pipeline Resource management commands

5.3.9.1. resource

Manage Pipeline Resources.

Example: Display help

```
$ tkn resource -h
```

5.3.9.2. resource create

Create a Pipeline Resource.

Example: Create Pipeline Resource defined by the myresource.yaml file in a namespace

```
$ tkn resource create -f myresource.yaml -n myspace
```

5.3.9.3. resource delete

Delete a Pipeline Resource.

Example: Delete the myresource Pipeline Resource from a namespace

```
$ tkn resource delete myresource -n myspace
```

5.3.9.4. resource describe

Describe a Pipeline Resource.

Example: Describe the myresource Pipeline Resource

```
$ tkn resource describe myresource -n myspace
```

5.3.9.5. resource list

List Pipeline Resources.

Example: List all Pipeline Resources in a namespace

```
$ tkn resource list -n myspace
```

5.3.10. ClusterTask management commands

5.3.10.1. clustertask

Manage ClusterTasks.

Example: Display help

```
$ tkn clustertask --help
```

5.3.10.2. clustertask delete

Delete a ClusterTask resource in a cluster.

Example: Delete mytask1 and mytask2 ClusterTasks

```
$ tkn clustertask delete mytask1 mytask2
```

5.3.10.3. clustertask describe

Describe a ClusterTask.

Example: Describe the mytask ClusterTask

```
$ tkn clustertask describe mytask1
```

5.3.10.4. clustertask list

List ClusterTasks.

Example: List ClusterTasks

```
$ tkn clustertask list
```

5.3.10.5. clustertask start

Start ClusterTasks.

Example: Start the mytask ClusterTask

```
$ tkn clustertask start mytask
```

5.3.11. Trigger management commands

5.3.11.1. eventlistener

Manage EventListeners.

Example: Display help

```
$ tkn eventlistener -h
```

5.3.11.2. eventlistener delete

Delete an EventListener.

Example: Delete mylistener1 and mylistener2 EventListeners in a namespace

```
$ tkn eventlistener delete mylistener1 mylistener2 -n myspace
```

5.3.11.3. eventlistener describe

Describe an EventListener.

Example: Describe the mylistener EventListener in a namespace

```
$ tkn eventlistener describe mylistener -n myspace
```

5.3.11.4. eventlistener list

List EventListeners.

Example: List all the EventListeners in a namespace

```
$ tkn eventlistener list -n myspace
```

5.3.11.5. triggerbinding

Manage TriggerBindings.

Example: Display TriggerBindings help

```
$ tkn triggerbinding -h
```

5.3.11.6. triggerbinding delete

Delete a TriggerBinding.

Example: Delete mybinding1 and mybinding2 TriggerBindings in a namespace

```
$ tkn triggerbinding delete mybinding1 mybinding2 -n myspace
```

5.3.11.7. triggerbinding describe

Describe a TriggerBinding.

Example: Describe the mybinding TriggerBinding in a namespace

```
$ tkn triggerbinding describe mybinding -n myspace
```

5.3.11.8. triggerbinding list

List TriggerBindings.

Example: List all the TriggerBindings in a namespace

```
$ tkn triggerbinding list -n myspace
```

5.3.11.9. triggertemplate

Manage TriggerTemplates.

Example: Display TriggerTemplate help

```
$ tkn triggertemplate -h
```

5.3.11.10. triggertemplate delete

Delete a TriggerTemplate.

Example: Delete mytemplate1 and mytemplate2 TriggerTemplates in a namespace

```
$ tkn triggertemplate delete mytemplate1 mytemplate2 -n `myspace`
```

5.3.11.11. triggertemplate describe

Describe a TriggerTemplate.

Example: Describe the mytemplate TriggerTemplate in a namespace

```
$ tkn triggertemplate describe mytemplate -n `myspace`
```

5.3.11.12. triggertemplate list

List TriggerTemplates.

Example: List all the TriggerTemplates in a namespace

```
$ tkn triggertemplate list -n myspace
```

5.3.11.13. clustertriggerbinding

Manage ClusterTriggerBindings.

Example: Display ClusterTriggerBindings help

```
$ tkn clustertriggerbinding -h
```

5.3.11.14. clustertriggerbinding delete

Delete a ClusterTriggerBinding.

Example: Delete myclusterbinding1 and myclusterbinding2 ClusterTriggerBindings

```
$ tkn clustertriggerbinding delete myclusterbinding1 myclusterbinding2
```

5.3.11.15. clustertriggerbinding describe

Describe a ClusterTriggerBinding.

Example: Describe the myclusterbinding ClusterTriggerBinding

```
$ tkn clustertriggerbinding describe myclusterbinding
```

5.3.11.16. clustertriggerbinding list

List ClusterTriggerBindings.

Example: List all ClusterTriggerBindings

```
$ tkn clustertriggerbinding list
```