Red Hat

OpenShift Container Platform 4.7

CLI tools

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

Last Updated: 2021-05-05

OpenShift Container Platform 4.7 CLI tools

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

Legal Notice

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. OPENSHIFT CLI (OC)

1.1. GETTING STARTED WITH THE OPENSHIFT CLI

1.1.1. About the OpenShift CLI

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

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

1.1.2. Installing the OpenShift CLI

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

1.1.2.1. Installing the OpenShift CLI by downloading the binary

You can install the OpenShift CLI (**oc**) 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 4.7. Download and install the new version of **oc**.

1.1.2.1.1. Installing the OpenShift CLI on Linux

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

Procedure

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 OpenShift 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 OpenShift 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 OpenShift CLI by using an RPM

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

Prerequisites

- Must have root or sudo privileges.

Procedure

1. Register with Red Hat Subscription Manager:

   ```
   # 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 4.7.

   - For Red Hat Enterprise Linux 8:

     ```
     # subscription-manager repos --enable="rhocp-4.7-for-rhel-8-x86_64-rpms"
     ```

   - For Red Hat Enterprise Linux 7:

     ```
     # subscription-manager repos --enable="rhel-7-server-ose-4.7-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 OpenShift CLI

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

Prerequisites

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

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

Procedure

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

```
$ oc login
```

Example output

```
Server [https://localhost:8443]: https://openshift.example.com:6443 ❶
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 ❷

Authentication required for https://openshift.example.com:6443 (openshift)
Username: user1 ❸
Password: ❹
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.
```

❶ Enter the OpenShift Container Platform server URL.

❷ Enter whether to use insecure connections.

❸ Enter the user name to log in as.

❹ Enter the user’s password.

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

1.1.4. Using the OpenShift 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
```

Example output

```
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
```

Example output

```
--> 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
```

Example output

```
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
```

Example output

```
--> 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
```

Example output

```
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
```

Example output

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

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

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

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
```

Example output

```
NAME                          SHORTNAMES    APIGROUP                 NAMESPACED   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
```

Example output

```
OpenShift Client

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

Usage:
  oc [flags]

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

...
```

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

Example: Get help for the oc create command

```
$ oc create --help
```

Example output

```
Create a resource by filename or stdin

JSON and YAML formats are accepted.

Usage:
  oc create -f FILENAME [flags]

...
```

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

Example: View documentation for the Pod resource

```
$ oc explain pods
```

Example output

```
KIND:     Pod
VERSION:  v1

DESCRIPTION:
     Pod is a collection of containers that can run on a host. This resource is
     created by clients and scheduled onto hosts.
```

```
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 OpenShift CLI

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

- Use the **oc logout** command.

  ```
  $ oc logout
  ```

  Example output

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

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

1.2. CONFIGURING THE OPENSHIFT 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
   ```

2. 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 OPENSHIFT CLI WITH PLUG-INS

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

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.

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

```
#!/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
   ```

   Example output

   ```
   The following compatible plugins are available:

   /usr/local/bin/<plugin_file>
   ```

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

4. Invoke the new command or option introduced by the plug-in.
   For example, if you built and installed the **kubectl-ns** plug-in from the Sample plug-in repository, you can use the following command to view the current namespace.

   ```
   $ oc ns
   ```

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

1.4. OPENSHIFT CLI DEVELOPER 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/parksmap-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 parksmap-katacoda-1-qfqz4

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

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

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

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/parksmap-katacoda
```

Example: Edit a deployment using a different editor

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

Example: Edit a deployment in JSON format

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

1.4.3.8. expose

Expose a service externally as a route.

Example: Expose a service

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

Example: Expose a service and specify the host name

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

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 a pod running a particular image.

Example: Start a pod running the perl image

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

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. OPENSHIFT CLI ADMINISTRATOR 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-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
```

1.5.5.2. create-error-template

Create a template for customizing the error page.

Example: Output a template for the error page to stdout

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

1.5.5.3. create-kubeconfig

Creates a basic **.kubeconfig** file from client certificates.

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

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

1.5.5.4. create-login-template

Create a template for customizing the login page.

Example: Output a template for the login page to stdout

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

1.5.5.5. create-provider-selection-template

Create a template for customizing the provider selection page.

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

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

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:4.7.0-x86_64 \
    quay.io/openshift-release-dev/ocp-release:4.7.1-x86_64 \
    > 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

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

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 OpenShift CLI. The **kubectl** binary is included in the archive if you download the binary, or is installed when you install the CLI by using an RPM.

For more information, see the kubectl documentation.

CHAPTER 2. DEVELOPER CLI (ODO)

2.1. {ODO-TITLE} RELEASE NOTES

2.1.1. Notable changes and improvements in odo

- **odo** now supports Devfile v2.
- **odo create -s2i** now converts an S2I component into a devfile component. When running **odo create --s2i <component-type> odo** now creates a converted Devfile component based on the S2I images of the specified component type.
  Note that this feature introduces many breaking changes, see Known Issues to learn more.
- Operator based service is now created on the cluster only after you run **odo push** and not after **odo service create** anymore.
- You can now use the **--container** flag to specify the container you want to attach storage to when running **odo storage create** command. See Adding storage to a specific container to learn the details.
- **odo catalog component describe** now returns correct JSON if the same name is used for a component in multiple registries.
- Commands that implement changes directly on a cluster now display a message informing a user that **odo push** is not required.
- When creating a component from a devfile, **odo create** now uses a default component name if the name is not specified.
- **odo** now has Telemetry. See Telemtry section to learn how to modify your Telemetry consent preferences.
- With **odo service**, you can now add or remove custom resource definitions and **ServiceInstance** information in your devfile.

2.1.2. 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.

2.1.3. 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>
  ```

- When running **odo url create** after **odo create --s2i**, the command fails. **odo** creates a URL now directly without asking.

- Wildfly and dotnet S2I components cannot be created with **odo create**.

- **odo env set DebugPort** does not work with converted devfile components. Workaround: use **odo config set --env DEBUG_PORT**.

- **odo delete --wait** does not wait for the resources to be terminated for devfile components.

2.2. UNDERSTANDING ODO

odo is a CLI tool for creating applications on OpenShift Container Platform and Kubernetes. With **odo**, you can write, build, and debug applications on a cluster without the need to administer the 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 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.

2.2.1. 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.

2.2.2. 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.

2.2.2.1. Officially supported languages and corresponding container images

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

Language	Container image	Package manager	Platform
Node.js	rhscl/nodejs-10-rhel7	NPM	amd64, s390x, ppc64le
	rhscl/nodejs-12-rhel7	NPM	amd64, s390x, ppc64le
Java	redhat-openjdk-18/openjdk18-openshift	Maven, Gradle	amd64, s390x, ppc64le
	openjdk/openjdk-11-rhel8	Maven, Gradle	amd64, s390x, ppc64le
	openjdk/openjdk-11-rhel7	Maven, Gradle	amd64, s390x, ppc64le

2.2.2.1.1. 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
```

Example output

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

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

The **TAGS** column represents the available image versions, for example, **10** represents the **rhoar-nodejs/nodejs-10** container image. To learn more about CLI commands, go to odo CLI reference.

2.2.2.2. Telemetry in odo

odo collects information about how **odo** is used: operating system, RAM, CPU size, number of cores, version of **odo**, errors, success/failure, and time it took for a command to complete.

You can modify your Telemetry consent by using **odo preference**:

- **odo preference set ConsentTelemetry true** to consent to Telemetry.
- **odo preference unset ConsentTelemetry** to disable Telemetry.
- **odo preference view** to verify the current preferences.

2.3. INSTALLING ODO

The following section describes how to install **odo** on different platforms using the CLI or the Visual Studio Code (VS Code) IDE.

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

Procedure

1. Obtain the binary:

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

2. Change the permissions on the file:

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

2.3.1.2. Tarball installation

Procedure

1. Obtain the tarball:

```
# 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'
```

2. Change the permissions on the file:

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

2.3.2. Installing odo on Linux on IBM Power

2.3.2.1. Binary installation

Procedure

1. Obtain the binary:

```
# curl -L https://mirror.openshift.com/pub/openshift-v4/clients/odo/latest/odo-linux-ppc64le -o
/usr/local/bin/odo
```

2. Change the permissions on the file:

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

2.3.2.2. Tarball installation

Procedure

1. Obtain the tarball:

```
# sh -c 'curl -L https://mirror.openshift.com/pub/openshift-v4/clients/odo/latest/odo-linux-ppc64le.tar.gz | gzip -d > /usr/local/bin/odo'
```

2. Change the permissions on the file:

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

2.3.3. Installing odo on Linux on IBM Z and LinuxONE

2.3.3.1. Binary installation

Procedure

1. Obtain the binary:

```
# curl -L https://mirror.openshift.com/pub/openshift-v4/clients/odo/latest/odo-linux-s390x -o /usr/local/bin/odo
```

2. Change the permissions on the file:

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

2.3.3.2. Tarball installation

Procedure

1. Obtain the tarball:

```
# sh -c 'curl -L https://mirror.openshift.com/pub/openshift-v4/clients/odo/latest/odo-linux-s390x.tar.gz | gzip -d > /usr/local/bin/odo'
```

2. Change the permissions on the file:

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

2.3.4. Installing odo on Windows

2.3.4.1. 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**.

2.3.5. Installing odo on macOS

2.3.5.1. Binary installation

Procedure

1. Obtain the binary:

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

2. Change the permissions on the file:

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

2.3.5.2. Tarball installation

Procedure

1. Obtain the tarball:

```
# 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'
```

2. Change the permissions on the file:

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

2.3.6. Installing odo on VS Code

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

Prerequisites

- You have installed VS Code.

Procedure

1. Open VS Code.
2. Launch VS Code Quick Open with **Ctrl**+**P**.
3. Enter the following command:

```
$ ext install redhat.vscode-openshift-connector
```

2.4. CREATING AND DEPLOYING APPLICATIONS WITH ODO

2.4.1. Working with projects

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

2.4.1.1. Creating a project

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

Procedure

1. Log in to an OpenShift Container Platform cluster:

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

2. Create a project:

```
$ odo project create myproject
```

Example output

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

2.4.2. Creating a single-component application with odo

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

Prerequisites

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

2.4.2.1. Creating a project

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

Procedure

1. Log in to an OpenShift Container Platform cluster:

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

2. Create a project:

```
$ odo project create myproject
```

Example output

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

2.4.2.2. Creating a Node.js application with odo

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

Procedure

1. Create a directory for your components:

```
$ 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>
```

2.4.2.3. Modifying your application code

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

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

2. Update your component:

```
$ odo push
```

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

2.4.2.4. Adding storage to the application components

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

Procedure

1. Add the storage to your component:

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

2. Push the storage to the cluster:

```
$ odo push
```

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

```
$ odo storage list
```

Example output

```
The component 'nodejs' has the following storage attached:
NAME          SIZE     PATH      STATE
mystorage      1Gi      /data     Pushed
```

4. Delete the storage from your component:

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

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

```
$ odo storage list
```

Example output

```
The component 'nodejs' has the following storage attached:
NAME          SIZE     PATH      STATE
mystorage      1Gi      /data     Locally Deleted
```

6. Push the changes to the cluster:

```
$ odo push
```

2.4.2.5. Adding a custom builder to specify a build image

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

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

Prerequisites

- The OpenShift CLI (oc) is installed.

Procedure

1. Import the image into OpenShift Container Platform:

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

2. Tag the image to make it accessible to odo:

```
$ 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
```

2.4.2.6. Connecting your application to multiple services using OpenShift Service Catalog

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

Prerequisites

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

Procedure

- To list the services:

```
$ odo catalog list services
```

- To use service catalog-related operations:

```
$ odo service <verb> <service_name>
```

2.4.2.7. Deleting an application

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

Procedure

1. List the applications in the current project:

```
$ odo app list
```

Example output

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

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

```
$ odo component list
```

Example output

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

3. Delete the application:

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

Example output

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

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

2.4.3. Creating a multicomponent application with odo

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

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

Prerequisites

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

2.4.3.1. Creating a project

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

Procedure

1. Log in to an OpenShift Container Platform cluster:

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

2. Create a project:

```
$ odo project create myproject
```

Example output

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

2.4.3.2. Deploying the back-end component

To create a Java component, import the Java builder image, download the Java application and push the source code to your cluster with **odo**.

Procedure

1. Import **openjdk18** into the cluster:

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

2. Tag the image as **builder** to make it accessible for odo:

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

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

```
$ odo catalog list components
```

Example output

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

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

4. Create a directory for your components:

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

5. Download the example back-end application:

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

6. Change to the back-end source directory:

```
$ cd backend
```

7. Check that you have the correct files in the directory:

```
$ ls
```

Example output

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

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

```
$ mvn package
```

Example output

```
...
[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] -------------------------------------
```

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

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

Example output

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

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

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

```
$ odo config view
```

Example output

```
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
```

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

```
$ odo push
```

Example output

```
Validation
 ✓  Checking component [6ms]

Configuration changes
 ✓  Initializing component
 ✓  Creating component [124ms]

Pushing to component backend of type binary
 ✓  Checking files for pushing [1ms]
 ✓  Waiting for component to start [48s]
 ✓  Syncing files to the component [811ms]
 ✓  Building component [3s]
```

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

12. Validate:

- The status of the action in odo:

```
$ odo log -f
```

Example output

```
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
```

Example output

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

2.4.3.3. Deploying the front-end component

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

Procedure

1. Download the example front-end application:

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

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

```
$ cd frontend
```

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

```
$ ls
```

Example output

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

NOTE

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

4. Create a component configuration of Node.js component-type named **frontend**:

```
$ odo create --s2i nodejs frontend
```

Example output

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

5. Push the component to a running container.

```
$ odo push
```

Example output

```
Validation
 ✓  Checking component [8ms]

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

```
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
```

2.4.3.4. Linking both components

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

Procedure

1. List all the components that are running on the cluster:

   ```
   $ odo list
   ```

   Example output

   ```
   OpenShift Components:
   APP     NAME         PROJECT     TYPE          SOURCETYPE     STATE
   app     backend      testpro     openjdk18     binary         Pushed
   app     frontend     testpro     nodejs        local          Pushed
   ```

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

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

   Example output

   ```
    ✓  Component backend has been successfully linked from the component frontend

   Following environment variables were added to frontend component:
   - COMPONENT_BACKEND_HOST
   - COMPONENT_BACKEND_PORT
   ```

   The configuration information of the back-end component is added to the front-end component and the front-end component restarts.

2.4.3.5. Exposing components to the public

Procedure

1. Navigate to the **frontend** directory:

   ```
   $ cd frontend
   ```

2. Create an external URL for the application:

   ```
   $ odo url create frontend --port 8080
   ```

Example output

```
 ✓  URL frontend created for component: frontend

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

3. Apply the changes:

```
$ odo push
```

Example output

```
Validation
 ✓  Checking component [21ms]

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

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

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

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

> NOTE
>
> If an application requires permissions to the active service account to access the OpenShift Container Platform namespace and delete active pods, the following error may occur when looking at **odo log** from the back-end component:
>
> **Message: Forbidden!Configured service account doesn't have access. Service account may have been revoked**
>
> To resolve this error, add permissions for the service account role:
>
> ```
> $ oc policy add-role-to-group view system:serviceaccounts -n <project>
> ```
>
> ```
> $ oc policy add-role-to-group edit system:serviceaccounts -n <project>
> ```
>
> Do not do this on a production cluster.

2.4.3.6. 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.

2.4.3.7. Deleting an application

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

Procedure

1. List the applications in the current project:

```
$ odo app list
```

Example output

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

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

```
$ odo component list
```

Example output

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

3. Delete the application:

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

Example output

```
    ? 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.

2.4.4. Creating an application with a database

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

Prerequisites

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

NOTE

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

2.4.4.1. Creating a project

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

Procedure

1. Log in to an OpenShift Container Platform cluster:

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

2. Create a project:

```
$ odo project create myproject
```

Example output

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

2.4.4.2. Deploying the front-end component

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

Procedure

1. Download the example front-end application:

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

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

   ```
   $ cd frontend
   ```

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

   ```
   $ ls
   ```

   Example output

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

   NOTE

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

4. Create a component configuration of Node.js component-type named **frontend**:

   ```
   $ odo create --s2i nodejs frontend
   ```

   Example output

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

5. Create a URL to access the frontend interface.

   ```
   $ odo url create myurl
   ```

   Example output

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

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

   ```
   $ odo push
   ```

   Example output

   ```
   Validation
    ✓  Checking component [7ms]

    Configuration changes
    ✓  Initializing component
   ```

```
✓  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
```

2.4.4.3. Deploying a database in interactive mode

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

Procedure

- Run the interactive mode and answer the prompts:

```
$ odo service create
```

Example output

```
? Which kind of service do you wish to create database
? Which database service class should we use mongodb-persistent
? Enter a value for string property DATABASE_SERVICE_NAME (Database Service Name):
mongodb
? Enter a value for string property MEMORY_LIMIT (Memory Limit): 512Mi
? Enter a value for string property MONGODB_DATABASE (MongoDB Database Name):
sampledb
? Enter a value for string property MONGODB_VERSION (Version of MongoDB Image): 3.2
? Enter a value for string property VOLUME_CAPACITY (Volume Capacity): 1Gi
? Provide values for non-required properties No
? How should we name your service  mongodb-persistent
? 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.

2.4.4.4. Deploying a database manually

1. List the available services:

```
$ odo catalog list services
```

Example output

```
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-pgsql-persistent       default
```

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

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

Example output

```
  ***********************        | *****************************************************
  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
```

2.4.4.5. Connecting the database to the front-end application

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

```
$ odo link mongodb-persistent
```

Example output

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

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

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

   a. Get the pod name:

   ```
   $ oc get pods
   ```

   ### Example output

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

   b. Connect to the pod:

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

   c. Check the environment variables:

   ```
   sh-4.2$ env
   ```

   ### Example output

   ```
   uri=mongodb://172.30.126.3:27017
   password=dHIOpYneSkX3rTLn
   database_name=sampledb
   username=user43U
   admin_password=NCn41tqmx7RIqmfv
   ```

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

```
$ odo url list
```

Example output

```
Request information
Page view count: 24

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

2.4.5. Creating a Java application with a database

This example describes how to deploy a Java application by using devfile and connect it to a database service.

Prerequisites

- A running cluster.
- **odo** is installed.
- A Service Binding Operator is installed in your cluster. To learn how to install Operators, contact your cluster administrator or see Installing Operators from OperatorHub .
- A Dev4Devs PostgreSQL Operator Operator is installed in your cluster. To learn how to install Operators, contact your cluster administrator or see Installing Operators from OperatorHub .

2.4.5.1. Creating a project

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

Procedure

1. Log in to an OpenShift Container Platform cluster:

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

2. Create a project:

```
$ odo project create myproject
```

Example output

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

2.4.5.2. Creating a database with odo

To create a database, you must have an access to the database Operator. For this example, Dev4Devs PostgreSQL Operator is used.

Procedure

1. View the list of the services in your project:

```
$ odo catalog list services
```

Example output

```
Operators available in the cluster
NAME                                CRDs
postgresql-operator.v0.1.1                 Backup, Database
```

2. Store the YAML of the service in a file:

```
$ odo service create postgresql-operator.v0.1.1/Database --dry-run > db.yaml
```

3. Add the following values under the **metadata:** section in the **db.yaml** file:

```
  name: sampledatabase
  annotations:
    service.binding/db.name: 'path={.spec.databaseName}'
    service.binding/db.password: 'path={.spec.databasePassword}'
    service.binding/db.user: 'path={.spec.databaseUser}'
```

This configuration ensures that when a database service is started, appropriate annotations are added to it. Annotations help the Service Binding Operator in injecting the values for **databaseName**, **databasePassword**, and **databaseUser** into the application.

4. Change the following values under the **spec:** section of the YAML file:

```
  databaseName: "sampledb"
  databasePassword: "samplepwd"
  databaseUser: "sampleuser"
```

5. Create a database from the YAML file:

```
$ odo service create --from-file db.yaml
```

A database instance is now present in your project.

2.4.5.3. Creating a Java MicroServices JPA application

With **odo**, you can create and manage a sample Java MicroServices JPA application.

Procedure

1. Clone the sample application:

```
$ git clone https://github.com/redhat-developer/application-stack-samples
```

2. Navigate to the application directory:

```
$ cd ./application-stack-samples/jpa
```

3. Initialize the project:

```
$ odo create java-openliberty java-application
```

4. Push the application to the cluster:

```
$ odo push
```

The application is now deployed to the cluster.

5. View the status of the cluster by streaming the OpenShift logs to the terminal:

```
$ odo log
```

Notice the test failures and **UnknownDatabaseHostException** error. This is because your application does not have a database yet:

```
[INFO] [err] java.net.UnknownHostException: ${DATABASE_CLUSTERIP}
[INFO] [err]    at
java.base/java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:220)
[INFO] [err]    at java.base/java.net.SocksSocketImpl.connect(SocksSocketImpl.java:403)
[INFO] [err]    at java.base/java.net.Socket.connect(Socket.java:609)
[INFO] [err]    at org.postgresql.core.PGStream.<init>(PGStream.java:68)
[INFO] [err]    at
org.postgresql.core.v3.ConnectionFactoryImpl.openConnectionImpl(ConnectionFactoryImpl.jav
a:144)
[INFO] [err]    ... 86 more
[ERROR] Tests run: 2, Failures: 1, Errors: 1, Skipped: 0, Time elapsed: 0.706 s <<<
FAILURE! - in org.example.app.it.DatabaseIT
[ERROR] testGetAllPeople  Time elapsed: 0.33 s  <<< FAILURE!
org.opentest4j.AssertionFailedError: Expected at least 2 people to be registered, but there
were only: [] ==> expected: <true> but was: <false>
        at org.example.app.it.DatabaseIT.testGetAllPeople(DatabaseIT.java:57)

[ERROR] testGetPerson  Time elapsed: 0.047 s  <<< ERROR!
java.lang.NullPointerException
        at org.example.app.it.DatabaseIT.testGetPerson(DatabaseIT.java:41)

[INFO]
[INFO] Results:
[INFO]
[ERROR] Failures:
[ERROR]   DatabaseIT.testGetAllPeople:57 Expected at least 2 people to be registered, but
there were only: [] ==> expected: <true> but was: <false>
[ERROR] Errors:
[ERROR]   DatabaseIT.testGetPerson:41 NullPointer
[INFO]
[ERROR] Tests run: 2, Failures: 1, Errors: 1, Skipped: 0
[INFO]
[ERROR] Integration tests failed: There are test failures.
```

6. Create an ingress URL to access the application:

   ```
   $ odo url create --port 8080
   ```

7. Push the changes to your cluster:

   ```
   $ odo push
   ```

8. Display the created URL:

   ```
   $ odo url list
   ```

   Example output

```
Found the following URLs for component mysboproj
NAME                 STATE      URL                                            PORT      SECURE     KIND
java-application-8080      Pushed      http://java-application-8080.apps-crc.testing      8080
false       ingress
```

The application is now deployed to the cluster and you can access it by using the URL that is created.

9. Use the URL to navigate to the **CreatePerson.xhtml** data entry page and enter a username and age by using the the form. Click **Save**.
   Note that you cannot see the data by clicking the **View Persons Record List** link since your application does not have a database connected yet.

2.4.5.4. Connecting a Java application to a database

To connect your Java application to the database, use the **odo link** command.

Procedure

1. Display the list of services:

   ```
   $ odo service list
   ```

   Example output

   ```
   NAME                           AGE
   Database/sampledatabase      6m31s
   ```

2. Connect the database to your application:

   ```
   $ odo link Database/sampledatabase
   ```

3. Push the changes to your cluster:

   ```
   $ odo push
   ```

   After the link has been created and pushed, a secret that contains the database connection data is created.

4. Check the component for values injected from the database service:

   ```
   $ odo exec -- bash -c 'export | grep DATABASE'
   declare -x DATABASE_CLUSTERIP="10.106.182.173"
   declare -x DATABASE_DB_NAME="sampledb"
   declare -x DATABASE_DB_PASSWORD="samplepwd"
   declare -x DATABASE_DB_USER="sampleuser"
   ```

5. Open the URL of your Java application and navigate to the **CreatePerson.xhtml** data entry page. Enter a username and age by using the the form. Click **Save**.
   Note that now you can see the data in the database by clicking the **View Persons Record List** link.

   You can also use a CLI tool such as **psql** to manipulate the database.

2.4.6. Using devfiles in odo

2.4.6.1. 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.

2.4.6.2. Creating a Java application by using a devfile

Prerequisites

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

NOTE

Currently odo does not support creating devfile components with **--git** or **--binary** flags. You can only create S2I components when using these flags.

2.4.6.2.1. Creating a project

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

Procedure

1. Log in to an OpenShift Container Platform cluster:

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

2. Create a project:

```
$ odo project create myproject
```

Example output

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

2.4.6.2.2. Listing available devfile components

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

Procedure

1. To list available devfile components on your cluster, run:

```
$ odo catalog list components
```

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

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

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

2.4.6.2.3. Deploying a Java application using a devfile

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

Procedure

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

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

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

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

The previous command produces the following output:

```
Validation
```

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

Please use `odo push` command to create the component with source deployed
```

The **odo create** command downloads the associated **devfile.yaml** file from the recorded devfile registries.

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

```
$ ls
```

The previous command produces the following output:

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

4. Create a URL to access the deployed component:

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

The previous command produces the following output:

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

To apply the URL configuration changes, please use odo push
```

NOTE

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

5. Push the component to the cluster:

```
$ odo push
```

The previous command produces the following output:

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

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

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

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

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

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

6. List the URLs of the component to verify that the component was pushed successfully:

   ```
   $ 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
   ```

2.4.6.3. Converting an S2I component into a devfile component

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

Procedure

Run all the commands from the S2I component directory.

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

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

2. Push the component to your cluster:

   ```
   $ odo push
   ```

   NOTE

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

3. Verify that the devfile component deployed successfully:

   ```
   $ odo list
   ```

4. Delete the S2I component:

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

2.4.7. Working with storage

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

2.4.7.1. Adding storage to the application components

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

Procedure

1. Add the storage to your component:

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

2. Push the storage to the cluster:

   ```
   $ odo push
   ```

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

   ```
   $ odo storage list
   ```

   Example output

   ```
   The component 'nodejs' has the following storage attached:
   NAME         SIZE     PATH      STATE
   mystorage     1Gi     /data     Pushed
   ```

4. Delete the storage from your component:

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

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

   ```
   $ odo storage list
   ```

   Example output

   ```
   The component 'nodejs' has the following storage attached:
   NAME         SIZE     PATH      STATE
   mystorage     1Gi     /data     Locally Deleted
   ```

6. Push the changes to the cluster:

   ```
   $ odo push
   ```

2.4.7.2. Adding storage to a specific container

If your devfile has multiple containers, you can use the **--container** flag to specify the container you want to attach storage to.

Procedure

1. Create a devfile with multiple containers:

```
components:
 - name: runtime ①
   container:
     image: registry.access.redhat.com/ubi8/nodejs-12:1-36
     memoryLimit: 1024Mi
     endpoints:
       - name: "3000-tcp"
         targetPort: 3000
     mountSources: true
 - name: funtime ②
   container:
     image: registry.access.redhat.com/ubi8/nodejs-12:1-36
     memoryLimit: 1024Mi
```

① The **runtime** container.

② The **funtime** container.

2. To create storage for the **runtime** container:

```
$ odo storage create store --path /data --size 1Gi --container runtime
```

Output of the command:

```
✓  Added storage store to nodejs-testing-xnfg
  Please use `odo push` command to make the storage accessible to the component
```

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

```
$ odo storage list
```

Example output

```
The component 'nodejs-testing-xnfg' has the following storage attached:
 NAME     SIZE     PATH      CONTAINER     STATE
 store     1Gi      /data      runtime        Not Pushed
```

4. Push the changes to the cluster:

```
$ odo push
```

2.4.7.3. Switching between ephemeral and persistent storage

You can switch between ephemeral and persistent storage in your project by using the **odo preference** command. **odo preference** modifies the global preference in your cluster.

When persistent storage is enabled, the cluster stores the information between the restarts.

When ephemeral storage is enabled, the cluster does not store the information between the restarts.

Ephemeral storage is enabled by default.

Procedure

1. See the preference currently set in your project:

   ```
   $ odo preference view
   ```

   Example output

   ```
   PARAMETER           CURRENT_VALUE
   UpdateNotification
   NamePrefix
   Timeout
   BuildTimeout
   PushTimeout
   Experimental
   PushTarget
   Ephemeral           true
   ```

2. To unset the ephemeral storage and set the persistent storage:

   ```
   $ odo preference set Ephemeral false
   ```

3. To set the ephemeral storage again:

   ```
   $ odo preference set Ephemeral true
   ```

   The **odo preference** command changes the global settings of all your currently deployed components as well as ones you will deploy in future.

4. Run **odo push** to make **odo** create a specified storage for your component:

   ```
   $ odo push
   ```

Additional resources

- Understanding ephemeral storage .
- Understanding persistent storage

2.4.8. Deleting applications

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

2.4.8.1. Deleting an application

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

Procedure

1. List the applications in the current project:

   ```
   $ odo app list
   ```

Example output

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

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

```
$ odo component list
```

Example output

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

3. Delete the application:

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

Example output

```
    ? 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.

2.4.9. Debugging applications in odo

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

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

2.4.9.1. Debugging an application

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

Procedure

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

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

Example output

```
Validation
 ✓  Checking devfile existence [11498ns]
 ✓  Checking devfile compatibility [15714ns]
 ✓  Creating a devfile component from registry: DefaultDevfileRegistry [17565ns]
 ✓  Validating devfile component [113876ns]

Starter Project
 ✓  Downloading starter project nodejs-starter from https://github.com/odo-devfiles/nodejs-ex.git [428ms]

Please use `odo push` command to create the component with source deployed
```

2. Push the application with the **--debug** flag, which is required for all debugging deployments:

   ```
   $ odo push --debug
   ```

   Example output

   ```
   Validation
    ✓  Validating the devfile [29916ns]

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

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

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

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

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

   NOTE

   You can specify a custom debug command by using the **--debug-command="custom-step"** flag.

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

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

   Example output

   ```
   Started port forwarding at ports - 5858:5858
   ```

   NOTE

   You can specify a port by using the **--local-port** flag.

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

```
$ odo debug info
```

Example output

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

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

2.4.9.2. 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.

2.4.10. 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.

To access the component over the web, create a URL using **odo url create**.

2.4.10.1. Examples from Git repositories

2.4.10.1.1. 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
```

2.4.10.1.2. 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
```

2.4.10.1.3. 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
```

2.4.10.1.4. 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
```

2.4.10.1.5. 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
```

2.4.10.1.6. 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
```

2.4.10.1.7. 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
```

2.4.10.2. Binary examples

2.4.10.2.1. 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
```

2.5. USING ODO IN A RESTRICTED ENVIRONMENT

2.5.1. 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

2.5.2. 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.

2.5.2.1. 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.

2.5.2.2. 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:

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

Procedure

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

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

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

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

3. Trust a CA in your client platform and log 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>
```

2.5.2.2.2. 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 --decode > disconnect-ca.crt
```

2. Copy the encoded root CA certificate to the appropriate location:
   a. Restart Docker using the Docker UI.

   b. Run the following command:

```
$ 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>
```

2.5.2.2.3. 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 --decode > disconnect-ca.crt
```

2. 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
```

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

   a. Restart Docker using the Docker UI.

   b. Run the following command:

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

4. 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>
```

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

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

2.5.2.3. 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:

2.5.2.3.1. Pushing the init image directly on Linux

Procedure

1. Enable the default route:

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

2. Get a wildcard route CA:

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

   Example output

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

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

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

4. Trust a CA in your client platform:

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

5. Log 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)
   ```

6. Push the **odo** init image:

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

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

   $ docker push <registry_path>/openshiftdo/odo-init-image-rhel7:<tag>
   ```

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

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

2.5.2.3.2. Pushing the init image directly on MacOS

Procedure

1. Enable the default route:

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

2. Get a wildcard route CA:

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

Example output

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

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

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

4. Trust a CA in your client platform:

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

5. Log 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)
```

6. Push the **odo** init image:

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

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

$ docker push <registry_path>/openshiftdo/odo-init-image-rhel7:<tag>
```

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

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

2.5.2.3.3. Pushing the init image directly on Windows

Procedure

1. Enable the default route:

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

2. Get a wildcard route CA:

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

   Example output

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

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

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

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

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

5. Log 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)
   ```

6. 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>
```

7. 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>"
```

2.5.3. 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.

2.5.3.1. 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 .

2.5.3.2. Mirroring a supported builder image

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

Procedure

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

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

Example output

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

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
```

Example output

```
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
[...]
```

2.5.3.3. 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>
  ```

2.5.3.4. 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>
```

2.5.4. Creating and deploying devfile components to the disconnected cluster

2.5.4.1. Creating a NodeJS application by using a devfile in a disconnected cluster

> **WARNING**
>
> This procedure is using external dependencies such as **nodejs-ex.git** application that are not maintained by Red Hat. These dependencies are not maintained with the documentation and their functionality cannot be guaranteed.

Prerequisites

- You have created and logged into a disconnected cluster.
- You have added **raw.githubusercontent.com**, **registry.access.redhat.com**, and **registry.npmjs.org** URLs in your proxy.

Procedure

1. Define your NodeJS application in a devfile:

Example of a devfile

```
schemaVersion: 2.0.0
metadata:
name: nodejs
starterProjects:
- name: nodejs-starter
  git:
    remotes:
      origin: "https://github.com/odo-devfiles/nodejs-ex.git"
components:
- name: runtime
  container:
    image: registry.access.redhat.com/ubi8/nodejs-12:1-36
    memoryLimit: 1024Mi
    endpoints:
      - name: "3000/tcp"
      targetPort: 3000
    env:
      - name: HTTP_PROXY
        value: http://<proxy-host>:<proxy-port>
      - name: HTTPS_PROXY
        value: http://<proxy-host>:<proxy-port>
    mountSources: true
commands:
- id: devbuild
  exec:
    component: runtime
    commandLine: npm install
    workingDir: ${PROJECTS_ROOT}
    group:
      kind: build
      isDefault: true
- id: build
  exec:
    component: runtime
    commandLine: npm install
    workingDir: ${PROJECTS_ROOT}
    group:
      kind: build
- id: devrun
  exec:
    component: runtime
    commandLine: npm start
    workingDir: ${PROJECTS_ROOT}
    group:
      kind: run
      isDefault: true
- id: run
  exec:
    component: runtime
    commandLine: npm start
    workingDir: ${PROJECTS_ROOT}
    group:
      kind: run
```

2. Create the application and push the changes to the cluster:

```
$ odo create nodejs --devfile <path-to-your-devfile> --starter $$ odo push
```

Example output

```
[...]
Pushing devfile component nodejs
 ✓  Changes successfully pushed to component
```

3. Create a URL to access your application and push it to th cluster:

```
$ odo url create url1 --port 3000 --host example.com --ingress && odo push
```

Example output

```
Validation
 ✓  Validating the devfile [145374ns]

Creating Kubernetes resources for component nodejs
 ✓  Waiting for component to start [14s]

Applying URL changes
 ✓  URL url1: http://url1.abcdr.com/ created

Syncing to component nodejs
 ✓  Checking file changes for pushing [2ms]
 ✓  Syncing files to the component [3s]

Executing devfile commands for component nodejs
 ✓  Executing devbuild command "npm install" [4s]
 ✓  Executing devrun command "npm start" [3s]

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

4. Add the storage to your application

```
$ odo storage create <storage-name> --path /data --size 5Gi
```

Example output

```
✓  Added storage abcde to nodejs

Please use `odo push` command to make the storage accessible to the component
```

5. Push the changes to the cluster:

```
$ odo push
```

2.5.4.2. Creating a Java application by using a devfile in a disconnected cluster

WARNING

This procedure is using external dependencies such as **quay.io/eclipse/che-java11-maven:nightly** or an example application **springboot-ex** that are not maintained by Red Hat. These dependencies are not maintained with the documentation and their functionality cannot be guaranteed.

Prerequisites

- You have created and logged into a disconnected cluster.

- You have added **quay.io**, **registry.access.redhat.com**, **apache.org**, **quayio-production-s3.s3.amazonaws.com** URLs in your proxy configuration.

Procedure

1. Define your Java application in a devfile:

Example of a devfile

```
schemaVersion: 2.0.0
metadata:
  name: java-maven
  version: 1.1.0
starterProjects:
  - name: springbootproject
    git:
      remotes:
        origin: "https://github.com/odo-devfiles/springboot-ex.git"
components:
  - name: tools
    container:
      image: quay.io/eclipse/che-java11-maven:nightly
      memoryLimit: 512Mi
      mountSources: true
      endpoints:
        - name: 'http-8080'
          targetPort: 8080
      volumeMounts:
        - name: m2
          path: /home/user/.m2
  - name: m2
    volume: {}
commands:
  - id: mvn-package
    exec:
      component: tools
      commandLine: "mvn -Dmaven.repo.local=/home/user/.m2/repository -Dhttp.proxyHost=
<proxy-host> -Dhttp.proxyPort=<proxy-port> -Dhttps.proxyHost=<proxy-host> -
Dhttps.proxyPort=<proxy-port> package"
      group:
        kind: build
        isDefault: true
  - id: run
    exec:
      component: tools
      commandLine: "java -jar target/*.jar"
      group:
        kind: run
        isDefault: true
  - id: debug
    exec:
      component: tools
      commandLine: "java -Xdebug -
Xrunjdwp:server=y,transport=dt_socket,address=${DEBUG_PORT},suspend=n -jar
target/*.jar"
```

```
      group:
        kind: debug
        isDefault: true
```

2. Create a Java application:

```
$ odo create java-maven --devfile <path-to-your-devfile> --starter
```

Example output

```
Validation
 ✓  Checking devfile existence [87716ns]
 ✓  Creating a devfile component from registry: DefaultDevfileRegistry [107247ns]
 ✓  Validating devfile component [396971ns]

 Starter Project
 ✓  Downloading starter project springbootproject from https://github.com/odo-devfiles/springboot-ex.git [2s]

Please use `odo push` command to create the component with source deployed
```

3. Push the changes to the cluster:

```
$ odo push
```

Example output

```
I0224 14:43:18.802512   34741 util.go:727] HTTPGetRequest:
https://raw.githubusercontent.com/openshift/odo/master/build/VERSION
I0224 14:43:18.833631   34741 context.go:115] absolute devfile path:
'/Users/pkumari/go/src/github.com/openshift/odo/testim/devfile.yaml'
[...]
Downloaded from central:
https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.2.1/plexus-utils-3.2.1.jar (262 kB at 813 kB/s)
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  19.638 s
[INFO] Finished at: 2021-02-24T08:59:30Z
[INFO] ------------------------------------------------------------------------
 ✓  Executing mvn-package command "mvn -Dmaven.repo.local=/home/user/.m2/repository
-Dhttp.proxyHost=<proxy-host> -Dhttp.proxyPort=<proxy-port> -Dhttps.proxyHost=<proxy-host> -Dhttps.proxyPort=<proxy-port> package" [23s]
 •  Executing run command "java -jar target/*.jar"  ...
I0224 14:29:30.557676   34426 exec.go:27] Executing command [/opt/odo/bin/supervisord ctl
start devrun] for pod: java-maven-5b8f99fcdb-9dnk6 in container: tools
devrun: started
 ✓  Executing run command "java -jar target/*.jar" [3s]

Pushing devfile component java-maven
 ✓  Changes successfully pushed to component
```

4. Display the logs to verify that the application has started:

```
$ odo log
```

Example output

```
time="2021-02-24T08:58:58Z" level=info msg="create process:devrun"
time="2021-02-24T08:58:58Z" level=info msg="create process:debugrun"
time="2021-02-24T08:59:32Z" level=debug msg="no auth required"
time="2021-02-24T08:59:32Z" level=debug msg="succeed to find process:devrun"
time="2021-02-24T08:59:32Z" level=info msg="try to start program" program=devrun
time="2021-02-24T08:59:32Z" level=info msg="success to start program" program=devrun
ODO_COMMAND_RUN is java -jar target/*.jar
Executing command  java -jar target/*.jar
[...]
```

5. Create storage for your application:

```
$ odo storage create storage-name --path /data --size 5Gi
```

Example output

```
✓  Added storage storage-name to java-maven

Please use `odo push` command to make the storage accessible to the component
```

6. Push the changes to the cluster:

```
$ odo push
```

Output

```
✓  Waiting for component to start [310ms]

Validation
 ✓  Validating the devfile [100798ns]

Creating Kubernetes resources for component java-maven
 ✓  Waiting for component to start [30s]
 ✓  Waiting for component to start [303ms]

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

Syncing to component java-maven
 ✓  Checking file changes for pushing [5ms]
 ✓  Syncing files to the component [4s]

Executing devfile commands for component java-maven
 ✓  Waiting for component to start [526ms]
 ✓  Executing mvn-package command "mvn -Dmaven.repo.local=/home/user/.m2/repository
-Dhttp.proxyHost=<proxy-host> -Dhttp.proxyPort=<proxy-port> -Dhttps.proxyHost=<proxy-
host> -Dhttps.proxyPort=<proxy-port> package" [10s]
```

```
 ✓  Executing run command "java -jar target/*.jar" [3s]

Pushing devfile component java-maven
 ✓  Changes successfully pushed to component
```

2.6. CREATING INSTANCES OF SERVICES MANAGED BY OPERATORS

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

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

2.6.1. 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.

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
```

Example output

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

2.6.3. 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.

2.6.4. 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 EtcdCluster
```

4. Verify that a service has started:

```
$ oc get EtcdCluster
```

2.6.5. Creating services from YAML files

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

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

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

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

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

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

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

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

❶ Change the name from **example** to **my-etcd-cluster**

❷ 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
```

2.7. 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.

2.7.1. 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
```

2.8. CONFIGURING THE ODO CLI

2.8.1. 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.

2.8.2. 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.

2.9. ODO CLI REFERENCE

2.9.1. Basic odo CLI commands

2.9.1.1. 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
```

2.9.1.2. 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
```

2.9.1.3. 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
```

2.9.1.4. 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

2.9.1.5. 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

  # Create a new Python component with the source in a Git repository
  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
```

2.9.1.6. debug

Debug a component.

Example using debug

```
# 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
```

2.9.1.7. delete

Delete an existing component.

Example using delete

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

2.9.1.8. describe

Describe the given component.

Example using describe

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

2.9.1.9. 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.

2.9.1.10. list

List all the components in the current application and the states of the components.

The states of the components

Pushed

A component is pushed to the cluster.

Not Pushed

A component is not pushed to the cluster.

Unknown

odo is disconnected from the cluster.

Example using list

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

# List all the components in a given path
odo list --path <path_to_your_component>
```

2.9.1.11. log

Retrieve the log for the given component.

Example using log

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

2.9.1.12. 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=xxxxxxxxxxxxxxxxxxxxxxxxx
```

2.9.1.13. logout

Log out of the current OpenShift Container Platform session.

Example using logout

```
# Log out
odo logout
```

2.9.1.14. 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

# Use persistent volumes in the cluster
odo preference set ephemeral false
```

NOTE

By default, the path to the global preference file is **~/.odo/preferece.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.

2.9.1.15. 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
```

2.9.1.16. 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

# Push source code and display event notifications in JSON format.
odo push -o json
```

2.9.1.17. registry

Create and modify custom registries.

Example using registry

```
# Add a registry to the registry list
odo registry add <registry name> <registry URL>

# List a registry in the registry list
odo registry list

# Delete a registry from the registry list
odo registry delete <registry name>

# Update a registry in the registry list
odo registry update <registry name> <registry URL>
```

```
# List a component with a corresponding registry
odo catalog list components

# Create a component that is hosted by a specific registry
odo create <component type> --registry <registry name>
```

2.9.1.18. 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
```

2.9.1.19. 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

  # 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
```

2.9.1.20. 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
```

2.9.1.21. 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
```

2.9.1.22. 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.

2.9.1.23. 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)
```

2.9.1.24. version

Print the client version information.

Example using version

```
# Print the client version of odo
odo version
```

2.9.1.25. 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
```

2.10. ODO ARCHITECTURE

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

2.10.1. 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.

2.10.2. 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.

2.10.3. OpenShift cluster objects

2.10.3.1. 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.

2.10.3.1.1. 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 theemptyDir 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.

2.10.3.1.2. 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.

2.10.3.2. 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.

2.10.3.3. 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**

2.10.3.4. 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**

2.10.3.5. 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.

2.10.4. 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.

CHAPTER 3. HELM CLI

3.1. GETTING STARTED WITH HELM 3 ON OPENSHIFT CONTAINER PLATFORM

3.1.1. Understanding Helm

Helm is a software package manager that simplifies deployment of applications and services to OpenShift Container Platform clusters.

Helm uses a packaging format called *charts*. A Helm chart is a collection of files that describes the OpenShift Container Platform resources.

A running instance of the chart in a cluster is called a *release*. A new release is created every time a chart is installed on the cluster.

Each time a chart is installed, or a release is upgraded or rolled back, an incremental revision is created.

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
```

Example output

```
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
```

Example output

```
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/
   ```

   Example output

   ```
   "stable" has been added to your repositories
   ```

3. Update the repository:

   ```
   $ helm repo update
   ```

4. Install an example MySQL chart:

   ```
   $ helm install example-mysql stable/mysql
   ```

5. Verify that the chart has installed successfully:

   ```
   $ helm list
   ```

   Example output

   ```
   NAME NAMESPACE REVISION UPDATED STATUS CHART APP VERSION
   example-mysql mysql 1 2019-12-05 15:06:51.379134163 -0500 EST deployed mysql-1.5.0
   5.7.27
   ```

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 ❶
name: nodejs-ex-k ❷
description: A Helm chart for OpenShift ❸
icon: https://static.redhat.com/libs/redhat/brand-assets/latest/corp/logo.svg ❹
```

❶ The chart API version. It should be **v2** for Helm charts that require at least Helm 3.

❷ The name of your chart.

❸ The description of your chart.

❹ The URL to an image to be used as an icon.

5. Verify that the chart is formatted properly:

```
$ helm lint
```

Example output

```
[INFO] Chart.yaml: icon is recommended

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

6. Navigate to the previous directory level:

```
$ cd ..
```

7. Install the chart:

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

8. Verify that the chart has installed successfully:

```
$ helm list
```

Example output

```
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
```

3.2. CONFIGURING CUSTOM HELM CHART REPOSITORIES

The **Developer Catalog**, in the **Developer** perspective of the web console, displays the Helm charts available in the cluster. By default, it lists the Helm charts from the Red Hat Helm chart repository. For a list of the charts see the Red Hat **Helm index** file.

As a cluster administrator, you can add multiple Helm chart repositories, apart from the default one, and display the Helm charts from these repositories in the **Developer Catalog**.

3.2.1. Adding custom Helm chart repositories

As a cluster administrator, you can add custom Helm chart repositories to your cluster and enable access to the Helm charts from these repositories in the **Developer Catalog**.

Procedure

1. To add a new Helm Chart Repository, you must add the Helm Chart Repository custom resource (CR) to your cluster.

   Sample Helm Chart Repository CR

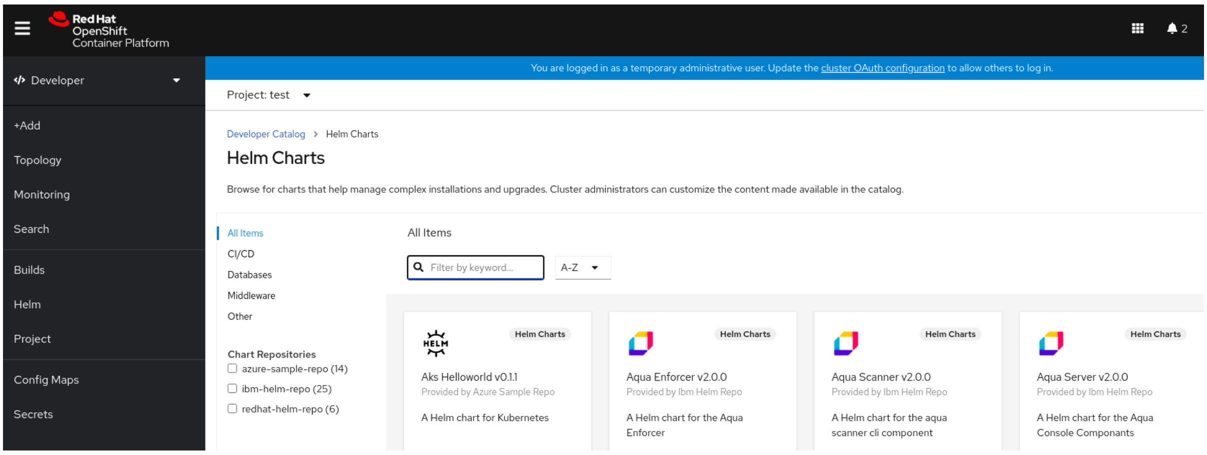
   ```
   apiVersion: helm.openshift.io/v1beta1
   kind: HelmChartRepository
   metadata:
     name: <name>
   spec:
    # optional name that might be used by console
    # name: <chart-display-name>
     connectionConfig:
       url: <helm-chart-repository-url>
   ```

   For example, to add an Azure sample chart repository, run:

   ```
   $ cat <<EOF | oc apply -f -
   apiVersion: helm.openshift.io/v1beta1
   kind: HelmChartRepository
   metadata:
     name: azure-sample-repo
   spec:
     name: azure-sample-repo
     connectionConfig:
       url: https://raw.githubusercontent.com/Azure-Samples/helm-charts/master/docs
   EOF
   ```

2. Navigate to the **Developer Catalog** in the web console to verify that the Helm charts from the chart repository are displayed.
   For example, use the **Chart repositories** filter to search for a Helm chart from the repository.

Figure 3.1. Chart repositories filter



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

Browse for charts that help manage complex installations and upgrades. Cluster administrators can customize the content made available in the catalog.

Aks Helloworld v0.1.1
Provided by Azure Sample Repo
A Helm chart for Kubernetes

Aqua Enforcer v2.0.0
Provided by Ibm Helm Repo
A Helm chart for the Aqua Enforcer

Aqua Scanner v2.0.0
Provided by Ibm Helm Repo
A Helm chart for the aqua scanner cli component

Aqua Server v2.0.0
Provided by Ibm Helm Repo
A Helm chart for the Aqua Console Componants

NOTE

If a cluster administrator removes all of the chart repositories, then you cannot view the Helm option in the **+Add** view, **Developer Catalog**, and left navigation panel.

3.2.2. Creating credentials and CA certificates to add Helm chart repositories

Some Helm chart repositories need credentials and custom certificate authority (CA) certificates to connect to it. You can use the web console as well as the CLI to add credentials and certificates.

Procedure

To configure the credentials and certificates, and then add a Helm chart repository using the CLI:

1. In the **openshift-config** namespace, create a **ConfigMap** object with a custom CA certificate in PEM encoded format, and store it under the **ca-bundle.crt** key within the config map:

```
$ oc create configmap helm-ca-cert \
--from-file=ca-bundle.crt=/path/to/certs/ca.crt \
-n openshift-config
```

2. In the **openshift-config** namespace, create a **Secret** object to add the client TLS configurations:

```
$ oc create secret generic helm-tls-configs \
--from-file=tls.crt=/path/to/certs/client.crt \
--from-file=tls.key=/path/to/certs//client.key \
-n openshift-config
```

Note that the client certificate and key must be in PEM encoded format and stored under the keys **tls.crt** and **tls.key**, respectively.

3. Add the Helm repository as follows:

```
$ cat <<EOF | oc apply -f -
apiVersion: helm.openshift.io/v1beta1
kind: HelmChartRepository
metadata:
  name: <helm-repository>
```

```
spec:
  name: <helm-repository>
  connectionConfig:
    url: <URL for the Helm repository>
    tlsConfig:
        name: helm-tls-configs
    ca:
 name: helm-ca-cert
EOF
```

The **ConfigMap** and **Secret** are consumed in the HelmChartRepository CR using the **tlsConfig** and **ca** fields. These certificates are used to connect to the Helm repository URL.

4. By default, all authenticated users have access to all configured charts. However, for chart repositories where certificates are needed, you must provide users with read access to the **helm-ca-cert** config map and **helm-tls-configs** secret in the **openshift-config** namespace, as follows:

```
$ cat <<EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: openshift-config
  name: helm-chartrepos-tls-conf-viewer
rules:
- apiGroups: [""]
  resources: ["configmaps"]
  resourceNames: ["helm-ca-cert"]
  verbs: ["get"]
- apiGroups: [""]
  resources: ["secrets"]
  resourceNames: ["helm-tls-configs"]
  verbs: ["get"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: openshift-config
  name: helm-chartrepos-tls-conf-viewer
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: 'system:authenticated'
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: helm-chartrepos-tls-conf-viewer
EOF
```

3.3. DISABLING HELM HART REPOSITORIES

As a cluster administrator, you can remove Helm chart repositories in your cluster so they are no longer visible in the **Developer Catalog**.

3.3.1. Disabling Helm Chart repository in the cluster

You can disable Helm Charts in the catalog by adding the **disabled** property in the **HelmChartRepository** custom resource.

Procedure

- To disable a Helm Chart repository by using CLI, add the **disabled: true** flag to the custom resource. For example, to remove an Azure sample chart repository, run:

```
$ cat <<EOF | oc apply -f -
apiVersion: helm.openshift.io/v1beta1
kind: HelmChartRepository
metadata:
  name: azure-sample-repo
spec:
  connectionConfig:
   url:https://raw.githubusercontent.com/Azure-Samples/helm-charts/master/docs
  disabled: true
EOF
```

- To disable a recently added Helm Chart repository by using Web Console:

  1. Go to **Custom Resource Definitions**and search for the **HelmChartRepository** custom resource.

  2. Go to **Instances**, find the repository you want to disable, and click its name.

  3. Go to the **YAML** tab, add the **disabled: true** flag in the **spec** section, and click **Save**.

     Example

     ```
     spec:
       connectionConfig:
         url: <url-of-the-repositoru-to-be-disabled>
       disabled: true
     ```

     The repository is now disabled and will not appear in the catalog.

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

The Knative **kn** CLI 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 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

The **kn** CLI is designed to make serverless computing tasks simple and concise. Key features of the **kn** CLI include:

- Deploy serverless applications from the command line.
- Manage features of Knative Serving, such as services, revisions, and traffic-splitting.
- Create and manage Knative Eventing components, such as event sources and triggers.
- Create sink bindings to connect existing Kubernetes applications and Knative services.
- Extend the **kn** CLI with flexible plug-in architecture, similar to the **kubectl** CLI.
- Configure autoscaling parameters for Knative services.
- Scripted usage, such as waiting for the results of an operation, or deploying custom rollout and rollback strategies.

4.2. INSTALLING THE KNATIVE CLI

See Installing the Knative CLI.

CHAPTER 5. PIPELINES CLI (TKN)

5.1. INSTALLING TKN

Use the **tkn** CLI to manage Red Hat OpenShift Pipelines from a terminal. The following section describes how to install **tkn** on different platforms.

You can also find the URL to the latest binaries from the OpenShift Container Platform web console by clicking the **?** icon in the upper-right corner and selecting **Command Line Tools**.

5.1.1. Installing Red Hat OpenShift Pipelines CLI (tkn) on Linux

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

Procedure

1. Download the relevant CLI.
   - Linux (x86_64, amd64)
   - Linux on IBM Z and LinuxONE (s390x)
   - Linux on IBM Power Systems (ppc64le)
2. Unpack the archive:
   ```
   $ tar xvzf <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 Red Hat OpenShift Pipelines CLI (tkn) on Linux using an RPM

For Red Hat Enterprise Linux (RHEL) version 8, you can install the Red Hat OpenShift Pipelines CLI (**tkn**) as an RPM.

Prerequisites

- You have an active OpenShift Container Platform subscription on your Red Hat account.
- You have root or sudo privileges on your local system.

Procedure

1. Register with Red Hat Subscription Manager:
   ```
   # 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 Red Hat OpenShift Pipelines:
   - Linux (x86_64, amd64)

     ```
     # subscription-manager repos --enable="pipelines-1.4-for-rhel-8-x86_64-rpms"
     ```

   - Linux on IBM Z and LinuxONE (s390x)

     ```
     # subscription-manager repos --enable="pipelines-1.4-for-rhel-8-s390x-rpms"
     ```

   - Linux on IBM Power Systems (ppc64le)

     ```
     # subscription-manager repos --enable="pipelines-1.4-for-rhel-8-ppc64le-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 Red Hat OpenShift Pipelines CLI (tkn) on Windows

For Windows, the **tkn** CLI is provided as a **zip** archive.

Procedure

1. Download the CLI.
2. Unzip the archive with a ZIP program.
3. Add the location of your **tkn.exe** file to your **PATH** environment variable.
4. To check your **PATH**, open the command prompt and run the command:

```
C:\> path
```

5.1.4. Installing Red Hat OpenShift Pipelines CLI (tkn) on macOS

For macOS, the **tkn** CLI is provided as a **tar.gz** archive.

Procedure

1. Download the CLI.
2. Unpack and unzip the archive.
3. Move the **tkn** binary to a directory on your PATH.
4. To check your **PATH**, open a terminal window and run:

   ```
   $ echo $PATH
   ```

5.2. CONFIGURING THE OPENSHIFT PIPELINES TKN CLI

Configure the Red Hat 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. OPENSHIFT 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 delete

Delete a Pipeline.

Example: Delete the mypipeline Pipeline from a namespace

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

5.3.4.3. pipeline describe

Describe a Pipeline.

Example: Describe **mypipeline** Pipeline

```
$ tkn pipeline describe mypipeline
```

5.3.4.4. pipeline list

List Pipelines.

Example: Display a list of Pipelines

```
$ tkn pipeline list
```

5.3.4.5. pipeline logs

Display Pipeline logs for a specific Pipeline.

Example: Stream live logs for the **mypipeline** Pipeline

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

5.3.4.6. 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 delete

Delete a Task.

Example: Delete **mytask1** and **mytask2** Tasks from a namespace

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

5.3.6.3. task describe

Describe a Task.

Example: Describe the mytask Task in a namespace

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

5.3.6.4. task list

List Tasks.

Example: List all the Tasks in a namespace

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

5.3.6.5. task logs

Display Task logs.

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

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

5.3.6.6. 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 a Pipeline Resource in a namespace

```
$ tkn resource create -n myspace
```

This is an interactive command that asks for input on the name of the Resource, type of the Resource, and the values based on the type of the Resource.

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
```

CHAPTER 6. OPM CLI

6.1. ABOUT OPM

The **opm** CLI tool is provided by the Operator Framework for use with the Operator Bundle Format. This tool allows you to create and maintain catalogs of Operators from a list of bundles, called an *index*, that are similar to software repositories. The result is a container image, called an *index image*, which can be stored in a container registry and then installed on a cluster.

An index contains a database of pointers to Operator manifest content that can be queried through an included API that is served when the container image is run. On OpenShift Container Platform, Operator Lifecycle Manager (OLM) can use the index image as a catalog by referencing it in a **CatalogSource** object, which polls the image at regular intervals to enable frequent updates to installed Operators on the cluster.

Additional resources

- See Operator Framework packaging formats for more information about the Bundle Format.
- To create a bundle image using the Operator SDK, see Working with bundle images.

6.2. INSTALLING OPM

You can install the **opm** CLI tool on your Linux, macOS, or Windows workstation.

Prerequisites

- For Linux, you must provide the following packages:
  - **podman** version 1.9.3+ (version 2.0+ recommended)
  - **glibc** version 2.28+

Procedure

1. Navigate to the OpenShift mirror site and download the latest version of the tarball that matches your operating system.
2. Unpack the archive.
   - For Linux or macOS:

     ```
     $ tar xvf <file>
     ```

   - For Windows, unzip the archive with a ZIP program.
3. Place the file anywhere in your **PATH**.
   - For Linux or macOS:
     a. Check your **PATH**:

        ```
        $ echo $PATH
        ```

b. Move the file. For example:

```
$ sudo mv ./opm /usr/local/bin/
```

- For Windows:

a. Check your **PATH**:

```
C:\> path
```

b. Move the file:

```
C:\> move opm.exe <directory>
```

Verification

- After you install the **opm** CLI, verify that it is available:

```
$ opm version
```

Example output

```
Version: version.Version{OpmVersion:"v1.15.4-2-g6183dbb3",
GitCommit:"6183dbb3567397e759f25752011834f86f47a3ea", BuildDate:"2021-02-
13T04:16:08Z", GoOs:"linux", GoArch:"amd64"}
```

6.3. ADDITIONAL RESOURCES

- See Managing custom catalogs for **opm** procedures including creating, updating, and pruning index images.

CHAPTER 7. OPERATOR SDK

7.1. INSTALLING THE OPERATOR SDK CLI

The Operator SDK provides a command-line interface (CLI) tool that Operator developers can use to build, test, and deploy an Operator. You can install the Operator SDK CLI on your workstation so that you are prepared to start authoring your own Operators.

See Developing Operators for full documentation on the Operator SDK.

NOTE

OpenShift Container Platform 4.7 supports Operator SDK v1.3.0.

7.1.1. Installing the Operator SDK CLI

You can install the OpenShift SDK CLI tool on Linux.

Prerequisites

- Go v1.13+
- **docker** v17.03+, **podman** v1.9.3+, or **buildah** v1.7+

Procedure

1. Navigate to the OpenShift mirror site .
2. From the **latest** directory, download the latest version of the tarball for Linux.
3. Unpack the archive:

```
$ tar xvf operator-sdk-v1.3.0-ocp-linux-x86_64.tar.gz
```

4. Make the file executable:

```
$ chmod +x operator-sdk
```

5. Move the extracted **operator-sdk** binary to a directory that is on your **PATH**.

TIP

To check your **PATH**:

```
$ echo $PATH
```

```
$ sudo mv ./operator-sdk /usr/local/bin/operator-sdk
```

Verification

- After you install the Operator SDK CLI, verify that it is available:

```
$ operator-sdk version
```

Example output

```
operator-sdk version: "v1.3.0-ocp", ...
```

7.2. OPERATOR SDK CLI REFERENCE

The Operator SDK command-line interface (CLI) is a development kit designed to make writing Operators easier.

Operator SDK CLI syntax

```
$ operator-sdk <command> [<subcommand>] [<argument>] [<flags>]
```

Operator authors with cluster administrator access to a Kubernetes-based cluster (such as OpenShift Container Platform) can use the Operator SDK CLI to develop their own Operators based on Go, Ansible, or Helm. Kubebuilder is embedded into the Operator SDK as the scaffolding solution for Go-based Operators, which means existing Kubebuilder projects can be used as is with the Operator SDK and continue to work.

See Developing Operators for full documentation on the Operator SDK.

7.2.1. bundle

The **operator-sdk bundle** command manages Operator bundle metadata.

7.2.1.1. validate

The **bundle validate** subcommand validates an Operator bundle.

Table 7.1. **bundle validate** flags

Flag	Description
-h, **--help**	Help output for the **bundle validate** subcommand.
--index-builder (string)	Tool to pull and unpack bundle images. Only used when validating a bundle image. Available options are **docker**, which is the default, **podman**, or **none**.
--list-optional	List all optional validators available. When set, no validators are run.
--select-optional (string)	Label selector to select optional validators to run. When run with the **--list-optional** flag, lists available optional validators.

7.2.2. cleanup

The **operator-sdk cleanup** command destroys and removes resources that were created for an Operator that was deployed with the **run** command.

Table 7.2. **cleanup** flags

Flag	Description
-h, **--help**	Help output for the **run bundle** subcommand.
--kubeconfig (string)	Path to the **kubeconfig** file to use for CLI requests.
n, **--namespace** (string)	If present, namespace in which to run the CLI request.
--timeout <duration>	Time to wait for the command to complete before failing. The default value is **2m0s**.

7.2.3. completion

The **operator-sdk completion** command generates shell completions to make issuing CLI commands quicker and easier.

Table 7.3. completion subcommands

Subcommand	Description
bash	Generate bash completions.
zsh	Generate zsh completions.

Table 7.4. completion flags

Flag	Description
-h, --help	Usage help output.

For example:

```
$ operator-sdk completion bash
```

Example output

```
# bash completion for operator-sdk                         -*- shell-script -*-
...
# ex: ts=4 sw=4 et filetype=sh
```

7.2.4. create

The **operator-sdk create** command is used to create, or *scaffold*, a Kubernetes API.

7.2.4.1. api

The **create api** subcommand scaffolds a Kubernetes API. The subcommand must be run in a project that was initialized with the **init** command.

Table 7.5. **create api** flags

Flag	Description
-h, **--help**	Help output for the **run bundle** subcommand.

7.2.5. generate

The **operator-sdk generate** command invokes a specific generator to generate code or manifests.

7.2.5.1. bundle

The **generate bundle** subcommand generates a set of bundle manifests, metadata, and a **bundle.Dockerfile** file for your Operator project.

NOTE

Typically, you run the **generate kustomize manifests** subcommand first to generate the input Kustomize bases that are used by the **generate bundle** subcommand. However, you can use the **make bundle** command in an initialized project to automate running these commands in sequence.

Table 7.6. **generate bundle** flags

Flag	Description
--channels (string)	Comma-separated list of channels to which the bundle belongs. The default value is **alpha**.
--crds-dir (string)	Root directory for **CustomResoureDefinition** manifests.
--default-channel (string)	The default channel for the bundle.
--deploy-dir (string)	Root directory for Operator manifests, such as deployments and RBAC. This directory is different from the directory passed to the **--input-dir** flag.
-h, **--help**	Help for **generate bundle**
--input-dir (string)	Directory from which to read an existing bundle. This directory is the parent of your bundle **manifests** directory and is different from the**--deploy-dir** directory.
--kustomize-dir (string)	Directory containing Kustomize bases and a **kustomization.yaml** file for bundle manifests. The default path is **config/manifests**.
--manifests	Generate bundle manifests.

Flag	Description
--metadata	Generate bundle metadata and Dockerfile.
--output-dir (string)	Directory to write the bundle to.
--overwrite	Overwrite the bundle metadata and Dockerfile if they exist. The default value is **true**.
--package (string)	Package name for the bundle.
-q, **--quiet**	Run in quiet mode.
--stdout	Write bundle manifest to standard out.
--version (string)	Semantic version of the Operator in the generated bundle. Set only when creating a new bundle or upgrading the Operator.

Additional resources

- See Bundling an Operator and deploying with Operator Lifecycle Manager for a full procedure that includes using the **make bundle** command to call the **generate bundle** subcommand.

7.2.5.2. kustomize

The **generate kustomize** subcommand contains subcommands that generate Kustomize data for the Operator.

7.2.5.2.1. manifests

The **generate kustomize manifests** subcommand generates or regenerates Kustomize bases and a **kustomization.yaml** file in the **config/manifests** directory, which are used to build bundle manifests by other Operator SDK commands. This command interactively asks for UI metadata, an important component of manifest bases, by default unless a base already exists or you set the **--interactive=false** flag.

Table 7.7. **generate kustomize manifests** flags

Flag	Description
--apis-dir (string)	Root directory for API type definitions.
-h, **--help**	Help for **generate kustomize manifests**.
--input-dir (string)	Directory containing existing Kustomize files.
--interactive	When set to **false**, if no Kustomize base exists, an interactive command prompt is presented to accept custom metadata.

Flag	Description
--output-dir (string)	Directory where to write Kustomize files.
--package (string)	Package name.
-q, **--quiet**	Run in quiet mode.

7.2.6. init

The **operator-sdk init** command initializes a Operator project and generates, or *scaffolds*, a default project directory layout for the given plug-in.

This command writes the following files:

- Boilerplate license file
- **PROJECT** file with the domain and repository
- **Makefile** to build the project
- **go.mod** file with project dependencies
- **kustomization.yaml** file for customizing manifests
- Patch file for customizing images for manager manifests
- Patch file for enabling Prometheus metrics
- **main.go** file to run

Table 7.8. **init** flags

Flag	Description
--help, -h	Help output for the **init** command.
--plugins (string)	Name and optionally version of the plug-in to initialize the project with. Available plug-ins are **ansible.sdk.operatorframework.io/v1**, **go.kubebuilder.io/v2**, **go.kubebuilder.io/v3**, and **helm.sdk.operatorframework.io/v1**.
--project-version	Project version. Available values are **2** and **3-alpha**, which is the default.

7.2.7. run

The **operator-sdk run** command provides options that can launch the Operator in various environments.

7.2.7.1. bundle

The **run bundle** subcommand deploys an Operator in the bundle format with Operator Lifecycle Manager (OLM).

Table 7.9. **run bundle** flags

Flag	Description
--index-image (string)	Index image in which to inject a bundle. The default image is **quay.io/operator-framework/upstream-opm-builder:latest**.
--install-mode <install_mode_value>	Install mode supported by the cluster service version (CSV) of the Operator, for example **AllNamespaces** or **SingleNamespace**.
--timeout <duration>	Install timeout. The default value is **2m0s**.
--kubeconfig (string)	Path to the **kubeconfig** file to use for CLI requests.
n, **--namespace** (string)	If present, namespace in which to run the CLI request.
-h, **--help**	Help output for the **run bundle** subcommand.

Additional resources

- See Operator group membership for details on possible install modes.

7.2.7.2. bundle-upgrade

The **run bundle-upgrade** subcommand upgrades an Operator that was previously installed in the bundle format with Operator Lifecycle Manager (OLM).

Table 7.10. **run bundle-upgrade** flags

Flag	Description
--timeout <duration>	Upgrade timeout. The default value is **2m0s**.
--kubeconfig (string)	Path to the **kubeconfig** file to use for CLI requests.
n, **--namespace** (string)	If present, namespace in which to run the CLI request.
-h, **--help**	Help output for the **run bundle** subcommand.

7.2.8. scorecard

The **operator-sdk scorecard** command runs the scorecard tool to validate an Operator bundle and provide suggestions for improvements. The command takes one argument, either a bundle image or directory containing manifests and metadata. If the argument holds an image tag, the image must be present remotely.

Table 7.11. **scorecard** flags

Flag	Description
-c, **--config** (string)	Path to scorecard configuration file. The default path is **bundle/tests/scorecard/config.yaml**.
-h, **--help**	Help output for the **scorecard** command.
--kubeconfig (string)	Path to **kubeconfig** file.
-L, **--list**	List which tests are available to run.
-n, --namespace (string)	Namespace in which to run the test images.
-o, **--output** (string)	Output format for results. Available values are **text**, which is the default, and**json**.
-l, **--selector** (string)	Label selector to determine which tests are run.
-s, **--service-account** (string)	Service account to use for tests. The default value is **default**.
-x, **--skip-cleanup**	Disable resource cleanup after tests are run.
-w, **--wait-time <duration>**	Seconds to wait for tests to complete, for example **35s**. The default value is**30s**.

Additional resources

- See Validating Operators using the scorecard tool for details about running the scorecard tool.