OpenShift Container Platform 4.7

CLI tools

Learning how to use the command-line tools for OpenShift Container Platform
OpenShift Container Platform 4.7 CLI tools

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

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

**CHAPTER 1. OPENSERVICELAb CLI (OC) ................................................................. 10**

1.1. GETTING STARTED WITH THE OPENSERVICELAb CLI ............. 10

1.1.1. About the OpenShift CLI ................................................................. 10

1.1.2. Installing the OpenShift CLI ......................................................... 10

1.1.2.1. Installing the OpenShift CLI by downloading the binary .......... 10

1.1.2.1.1. Installing the OpenShift CLI on Linux .................................... 10

1.1.2.1.2. Installing the OpenShift CLI on Windows ............................... 11

1.1.2.1.3. Installing the OpenShift CLI on macOS ............................... 11

1.1.2.2. Installing the OpenShift CLI by using an RPM ....................... 12

1.1.3. Logging in to the OpenShift CLI .................................................. 12

1.1.4. Using the OpenShift CLI ................................................................. 14

1.1.4.1. Creating a project ................................................................. 14

1.1.4.2. Creating a new app ................................................................. 14

1.1.4.3. Viewing pods ................................................................. 14

1.1.4.4. Viewing pod logs ................................................................. 14

1.1.4.5. Viewing the current project ...................................................... 15

1.1.4.6. Viewing the status for the current project .............................. 15

1.1.4.7. Listing supported API resources .............................................. 15

1.1.5. Getting help ................................................................................. 15

1.1.6. Logging out of the OpenShift CLI ............................................... 17

1.2. CONFIGURING THE OPENSERVICELAb CLI ................................. 17

1.2.1. Enabling tab completion ............................................................. 17

1.3. EXTENDING THE OPENSERVICELAb CLI WITH PLUG-INS ........ 18

1.3.1. Writing CLI plug-ins ................................................................. 18

1.3.2. Installing and using CLI plug-ins .............................................. 19

1.4. OPENSERVICELAb CLI DEVELOPER COMMANDS .................... 20

1.4.1. Basic CLI commands ................................................................. 20

1.4.1.1. explain ................................................................................. 20

1.4.1.2. login ..................................................................................... 20

1.4.1.3. new-app ................................................................................. 20

1.4.1.4. new-project ................................................................. 21

1.4.1.5. project ................................................................. 21

1.4.1.6. projects ................................................................. 21

1.4.1.7. status ................................................................. 21

1.4.2. Build and Deploy CLI commands ........................................... 21

1.4.2.1. cancel-build ................................................................. 21

1.4.2.2. import-image ................................................................. 21

1.4.2.3. new-build ................................................................. 22

1.4.2.4. rollback ................................................................. 22

1.4.2.5. rollout ................................................................. 22

1.4.2.6. start-build ................................................................. 23

1.4.2.7. tag ................................................................. 23

1.4.3. Application management CLI commands ................................. 23

1.4.3.1. annotate ................................................................. 23

1.4.3.2. apply ................................................................. 23

1.4.3.3. autoscale ................................................................. 23

1.4.3.4. create ................................................................. 23

1.4.3.5. delete ................................................................. 24

1.4.3.6. describe ................................................................. 24

1.4.3.7. edit ................................................................. 24

1.4.3.8. expose ................................................................. 24
1.4.3.9. get
1.4.3.10. label
1.4.3.11. scale
1.4.3.12. secrets
1.4.3.13. serviceaccounts
1.4.3.14. set

1.4.4. Troubleshooting and debugging CLI commands
1.4.4.1. attach
1.4.4.2. cp
1.4.4.3. debug
1.4.4.4. exec
1.4.4.5. logs
1.4.4.6. port-forward
1.4.4.7. proxy
1.4.4.8. rsh
1.4.4.9. rsync
1.4.4.10. run
1.4.4.11. wait

1.4.5. Advanced developer CLI commands
1.4.5.1. api-resources
1.4.5.2. api-versions
1.4.5.3. auth
1.4.5.4. cluster-info
1.4.5.5. extract
1.4.5.6. idle
1.4.5.7. image
1.4.5.8. observe
1.4.5.9. patch
1.4.5.10. policy
1.4.5.11. process
1.4.5.12. registry
1.4.5.13. replace

1.4.6. Settings CLI commands
1.4.6.1. completion
1.4.6.2. config
1.4.6.3. logout
1.4.6.4. whoami

1.4.7. Other developer CLI commands
1.4.7.1. help
1.4.7.2. plugin
1.4.7.3. version

1.5. OPENSOURCE CLI ADMINISTRATOR COMMANDS
1.5.1. Cluster management CLI commands
1.5.1.1. inspect
1.5.1.2. must-gather
1.5.1.3. top

1.5.2. Node management CLI commands
1.5.2.1. cordon
1.5.2.2. drain
1.5.2.3. node-logs
1.5.2.4. taint
1.5.2.5. uncordon

1.5.3. Security and policy CLI commands
1.5.3.1. certificate
1.5.3.2. groups
1.5.3.3. new-project
1.5.3.4. pod-network
1.5.3.5. policy
1.5.4. Maintenance CLI commands
1.5.4.1. migrate
1.5.4.2. prune
1.5.5. Configuration CLI commands
1.5.5.1. create-bootstrap-project-template
1.5.5.2. create-error-template
1.5.5.3. create-kubeconfig
1.5.5.4. create-login-template
1.5.5.5. create-provider-selection-template
1.5.6. Other Administrator CLI commands
1.5.6.1. build-chain
1.5.6.2. completion
1.5.6.3. config
1.5.6.4. release
1.5.6.5. verify-image-signature

1.6. USAGE OF OC AND KUBECTL COMMANDS
1.6.1. The oc binary
1.6.2. The kubectl binary

CHAPTER 2. DEVELOPER CLI (ODO) .................................................. 38
2.1. [ODO-TITLE] RELEASE NOTES .............................................. 38
  2.1.1. Notable changes and improvements in odo
  2.1.2. Getting support
  2.1.3. Known issues
2.2. UNDERSTANDING ODO ...................................................... 39
  2.2.1. Key features
  2.2.2. Core concepts
  2.2.2.1. Officially supported languages and corresponding container images
  2.2.2.1.1. Listing available container images
  2.2.2.2. Telemetry in odo
2.3. INSTALLING ODO ............................................................. 41
  2.3.1. Installing odo on Linux
  2.3.1.1. Binary installation
  2.3.1.2. Tarball installation
  2.3.1.3. Installing with yum on Red Hat Enterprise Linux (RHEL)
  2.3.2. Installing odo on Linux on IBM Power
  2.3.2.1. Binary installation
  2.3.2.2. Tarball installation
  2.3.3. Installing odo on Linux on IBM Z and LinuxONE
  2.3.3.1. Binary installation
  2.3.3.2. Tarball installation
  2.3.4. Installing odo on Windows
  2.3.4.1. Binary installation
    Setting the PATH variable for Windows 7/8
    Setting the PATH variable for Windows 10
  2.3.5. Installing odo on macOS
  2.3.5.1. Binary installation
  2.3.5.2. Tarball installation
2.3.6. Installing odo on VS Code

2.4. CREATING AND DEPLOYING APPLICATIONS WITH ODO

2.4.1. Working with projects

2.4.1.1. Creating a project

2.4.2. Creating a single-component application with odo

2.4.2.1. Creating a project

2.4.2.2. Creating a Node.js application with odo

2.4.2.3. Modifying your application code

2.4.2.4. Adding storage to the application components

2.4.2.5. Adding a custom builder to specify a build image

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

2.4.2.7. Deleting an application

2.4.3. Creating a multicomponent application with odo

2.4.3.1. Creating a project

2.4.3.2. Deploying the back-end component

2.4.3.3. Deploying the front-end component

2.4.3.4. Linking both components

2.4.3.5. Exposing components to the public

2.4.3.6. Modifying the running application

2.4.3.7. Deleting an application

2.4.4. Creating an application with a database

2.4.4.1. Creating a project

2.4.4.2. Deploying the front-end component

2.4.4.3. Deploying a database in interactive mode

2.4.4.4. Deploying a database manually

2.4.4.5. Connecting the database to the front-end application

2.4.5. Creating a Java application with a database

2.4.5.1. Creating a project

2.4.5.2. Creating a Java MicroServices JPA application

2.4.5.3. Creating a database with odo

2.4.5.4. Connecting a Java application to a database

2.4.6. Using devfiles in odo

2.4.6.1. About the devfile in odo

2.4.6.2. Creating a Java application by using a devfile

2.4.6.2.1. Creating a project

2.4.6.2.2. Listing available devfile components

2.4.6.2.3. Deploying a Java application using a devfile

2.4.6.3. Converting an S2I component into a devfile component

2.4.7. Working with storage

2.4.7.1. Adding storage to the application components

2.4.7.2. Adding storage to a specific container

2.4.7.3. Switching between ephemeral and persistent storage

2.4.8. Deleting applications

2.4.8.1. Deleting an application

2.4.9. Debugging applications in odo

2.4.9.1. Debugging an application

2.4.9.2. Configuring debugging parameters

2.4.10. Sample applications

2.4.10.1. Examples from Git repositories

2.4.10.1.1. httpd

2.4.10.1.2. java

2.4.10.1.3. nodejs

2.4.10.1.4. perl
5.3.2. Global options
5.3.3. Utility commands
  5.3.3.1. tkn
  5.3.3.2. completion [shell]
  5.3.3.3. version
5.3.4. Pipelines management commands
  5.3.4.1. pipeline
  5.3.4.2. pipeline delete
  5.3.4.3. pipeline describe
  5.3.4.4. pipeline list
  5.3.4.5. pipeline logs
  5.3.4.6. pipeline start
5.3.5. PipelineRun commands
  5.3.5.1. pipelinerun
  5.3.5.2. pipelinerun cancel
  5.3.5.3. pipelinerun delete
  5.3.5.4. pipelinerun describe
  5.3.5.5. pipelinerun list
  5.3.5.6. pipelinerun logs
5.3.6. Task management commands
  5.3.6.1. task
  5.3.6.2. task delete
  5.3.6.3. task describe
  5.3.6.4. task list
  5.3.6.5. task logs
  5.3.6.6. task start
5.3.7. TaskRun commands
  5.3.7.1. taskrun
  5.3.7.2. taskrun cancel
  5.3.7.3. taskrun delete
  5.3.7.4. taskrun describe
  5.3.7.5. taskrun list
  5.3.7.6. taskrun logs
5.3.8. Condition management commands
  5.3.8.1. condition
  5.3.8.2. condition delete
  5.3.8.3. condition describe
  5.3.8.4. condition list
5.3.9. Pipeline Resource management commands
  5.3.9.1. resource
  5.3.9.2. resource create
  5.3.9.3. resource delete
  5.3.9.4. resource describe
  5.3.9.5. resource list
5.3.10. ClusterTask management commands
  5.3.10.1. clustertask
  5.3.10.2. clustertask delete
  5.3.10.3. clustertask describe
  5.3.10.4. clustertask list
  5.3.10.5. clustertask start
5.3.11. Trigger management commands
  5.3.11.1. eventlistener
  5.3.11.2. eventlistener delete
5.3.11.3. eventlistener describe
5.3.11.4. eventlistener list
5.3.11.5. eventlistener logs
5.3.11.6. triggerbinding
5.3.11.7. triggerbinding delete
5.3.11.8. triggerbinding describe
5.3.11.9. triggerbinding list
5.3.11.10. triggertemplate
5.3.11.11. triggertemplate delete
5.3.11.12. triggertemplate describe
5.3.11.13. triggertemplate list
5.3.11.14. clustertriggerbinding
5.3.11.15. clustertriggerbinding delete
5.3.11.16. clustertriggerbinding describe
5.3.11.17. clustertriggerbinding list
5.3.12. Hub interaction commands
5.3.12.1. hub
5.3.12.2. hub downgrade
5.3.12.3. hub get
5.3.12.4. hub info
5.3.12.5. hub install
5.3.12.6. hub reinstall
5.3.12.7. hub search
5.3.12.8. hub upgrade

CHAPTER 6. OPM CLI
6.1. ABOUT OPM
6.2. INSTALLING OPM
6.3. ADDITIONAL RESOURCES

CHAPTER 7. OPERATOR SDK
7.1. INSTALLING THE OPERATOR SDK CLI
7.1.1. Installing the Operator SDK CLI
7.2. OPERATOR SDK CLI REFERENCE
7.2.1. bundle
7.2.1.1. validate
7.2.2. cleanup
7.2.3. completion
7.2.4. create
7.2.4.1. api
7.2.5. generate
7.2.5.1. bundle
7.2.5.2. kustomize
7.2.5.2.1. manifests
7.2.6. init
7.2.7. run
7.2.7.1. bundle
7.2.7.2. bundle-upgrade
7.2.8. scorecard
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**


2. Select the appropriate version in the Version drop-down menu.

3. Click Download Now next to the OpenShift v4.7 Linux Client entry and save the file.

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 OpenShift 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**

2. Select the appropriate version in the Version drop-down menu.
3. Click *Download Now* next to the **OpenShift v4.7 Windows Client** entry and save the file.
4. Unzip the archive with a ZIP program.
5. Move the `oc` binary to a directory that is on your PATH.
   To check your PATH, open the command prompt and execute the following command:
   
   ```
   C:\> path
   ```

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

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

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**

2. Select the appropriate version in the Version drop-down menu.
3. Click *Download Now* next to the **OpenShift v4.7 MacOSX Client** entry and save the file.
4. Unpack and unzip the archive.
5. Move the `oc` binary to a directory on your PATH.
   To check your PATH, open a terminal and execute the following command:
   
   ```
   $ echo $PATH
   ```

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

```
$ oc <command>
```
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:
   ```bash
   # subscription-manager register
   ```

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

3. List the available subscriptions:
   ```bash
   # subscription-manager list --available --matches "OpenShift"
   ```

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

5. Enable the repositories required by OpenShift Container Platform 4.7.
   - For Red Hat Enterprise Linux 8:
     ```bash
     # subscription-manager repos --enable="rhocp-4.7-for-rhel-8-x86_64-rpms"
     ```
   - For Red Hat Enterprise Linux 7:
     ```bash
     # subscription-manager repos --enable="rhel-7-server-ose-4.7-rpms"
     ```

6. Install the `openshift-clients` package:
   ```bash
   # yum install openshift-clients
   ```

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

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

1.1.3. Logging in to the OpenShift CLI

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

Prerequisites
You must have access to an OpenShift Container Platform cluster.

You must have installed the OpenShift CLI (oc).

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

1. Enter the oc login command and pass in a user name:

   $ oc login -u user1

2. When prompted, enter the required information:

   Example output

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

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

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

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

1.1.4.5. Viewing the current project

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

```bash
$ oc project
```

**Example output**

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

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.

```bash
$ oc status
```

**Example output**

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

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

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

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.

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

**Example output**

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

```

1.1.5. Getting help

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

```bash
$ oc logout
```

### Example output

Logged “user1” out on “https://openshift.example.com”

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

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.
- You must have the package `bash-completion` installed.

**Procedure**

The following procedure enables tab completion for Bash.

1. Save the Bash completion code to a file.

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

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

```bash
#!/bin/bash

# optional argument handling
if [[ "$1" == "version" ]]; then
  echo "1.0.0"
  exit 0
fi

# optional argument handling
if [[ "$1" == "config" ]];
```
After you install this plug-in for the OpenShift Container Platform CLI, it can be invoked using the `oc foo` 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.

**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>
   
   then
   echo $KUBECONFIG
   exit 0
   fi
   
   echo "I am a plugin named kubectl-foo"
   ```

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**

---

**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.
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. OPENSSHIFT 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 -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/parksmapi-katadoda --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 parkmap-katacoda-1-qfqz4

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

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

$ oc delete pods -l app=parkmap-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/parkmap-katacoda

Example: Edit a deployment using a different editor

$ OC_EDITOR="nano" oc edit deploymentconfig/parkmap-katacoda

Example: Edit a deployment in JSON format

$ oc edit deploymentconfig/parkmap-katacoda -o json

1.4.3.8. expose
Expose a service externally as a route.

Example: Expose a service
Example: Expose a service and specify the host name

$ oc expose service/parksmap-katacoda

$ 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
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 serviceaccounts get-token default
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. 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=-
```

1.4.5.6. 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
```

1.4.5.7. 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
```

1.4.5.8. observe

Observe changes to resources and take action on them.

Example: Observe changes to services

```
$ oc observe services
```

1.4.5.9. 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.

1.4.5.10. policy

Manage authorization policies.

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

```
$ oc policy add-role-to-user edit user1
```
1.4.5.11. 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 -
```

1.4.5.12. registry

Manage the integrated registry on OpenShift Container Platform.

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

```
$ oc registry info
```

1.4.5.13. 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. OPENSShift 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 sha256:2bba968aedb7dd2aaf6e5fa8c7453f5ac36a0b9639f1bf5b03f95de325238b288 --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 [Telemetry 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 **OpenShift Developer Tools and Services** as a product type and **odo** as a component.

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

<table>
<thead>
<tr>
<th>Language</th>
<th>Container image</th>
<th>Package manager</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node.js</td>
<td>rhscl/nodejs-10-rhel7</td>
<td>NPM</td>
<td>amd64, s390x, ppc64le</td>
</tr>
<tr>
<td></td>
<td>rhscl/nodejs-12-rhel7</td>
<td>NPM</td>
<td>amd64, s390x, ppc64le</td>
</tr>
<tr>
<td>Java</td>
<td>redhat-openjdk-18/openjdk18-openshift</td>
<td>Maven, Gradle</td>
<td>amd64, s390x, ppc64le</td>
</tr>
<tr>
<td></td>
<td>openjdk/openjdk-11-rhel8</td>
<td>Maven, Gradle</td>
<td>amd64, s390x, ppc64le</td>
</tr>
<tr>
<td></td>
<td>openjdk/openjdk-11-rhel7</td>
<td>Maven, Gradle</td>
<td>amd64, s390x, ppc64le</td>
</tr>
</tbody>
</table>

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:

```bash
$ odo catalog list components
```

**Example output**

<table>
<thead>
<tr>
<th>Odo Devfile Components:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>java-maven</td>
<td>Upstream Maven and OpenJDK 11</td>
</tr>
<tr>
<td>java-openliberty</td>
<td>Open Liberty microservice in Java</td>
</tr>
<tr>
<td>java-quarkus</td>
<td>Upstream Quarkus with Java+GraalVM</td>
</tr>
<tr>
<td>java-springboot</td>
<td>Spring Boot® using Java</td>
</tr>
<tr>
<td>nodejs</td>
<td>Stack with NodeJS 12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Odo OpenShift Components:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>PROJECT</td>
</tr>
<tr>
<td>java</td>
<td>openshift</td>
</tr>
<tr>
<td>dotnet</td>
<td>openshift</td>
</tr>
<tr>
<td>golang</td>
<td>openshift</td>
</tr>
<tr>
<td>httpd</td>
<td>openshift</td>
</tr>
<tr>
<td>nginx</td>
<td>openshift</td>
</tr>
<tr>
<td>nodejs</td>
<td>openshift</td>
</tr>
<tr>
<td>perl</td>
<td>openshift</td>
</tr>
<tr>
<td>php</td>
<td>openshift</td>
</tr>
<tr>
<td>python</td>
<td>openshift</td>
</tr>
<tr>
<td>ruby</td>
<td>openshift</td>
</tr>
<tr>
<td>wildfly</td>
<td>openshift</td>
</tr>
</tbody>
</table>

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:

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

2. Change the permissions on the file:

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

2.3.1.3. Installing with yum on Red Hat Enterprise Linux (RHEL)

Procedure

1. Register with Red Hat Subscription Manager:

   ```
   # subscription-manager register
   ```

2. Pull the latest subscription data:

   ```
   # subscription-manager refresh
   ```

3. List the available subscriptions:

   ```
   # subscription-manager list --available --matches "OpenShift Developer Tools and Services"
   ```
4. In the output of the previous command, find the **Pool ID** field for your OpenShift Container Platform subscription and attach the subscription to the registered system:

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

5. Enable the repositories required by **odo**:

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

6. Install the **odo** package:

```
# yum install odo
```

7. Verify that **odo** is now available on your system:

```
$ odo version
```

### 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:

```
```

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:

   ```bash
   ```

2. Change the permissions on the file:

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

#### 2.3.5.2. Tarball installation

**Procedure**

1. Obtain the tarball:

   ```bash
   # 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:

   ```bash
   # 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
   ```
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:
   
   ```bash
   $ mkdir my_components && cd my_components
   ```

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

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

4. Add a component of the type Node.js to your application:
   
   ```bash
   $ odo create nodejs
   ```

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

5. Push the initial source code to the component:
   
   ```bash
   $ 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:
   
   ```bash
   $ odo url create --port 8080
   ```

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

8. List the URLs to check the desired URL for the component.
   
   ```bash
   $ 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
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:

   ```bash
   $ 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:

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

3. Deploy the image with odo:

   ```bash
   $ 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:
To use service catalog-related operations:

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:

   ```bash
   $ 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:

   ```bash
   $ odo component list
   ``

   **Example output**

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

3. Delete the application:

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

   **Example output**

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

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

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
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
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]
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**

     ```
     : 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**
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`:

```bash
$ 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.

```bash
$ 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:

   ```bash
   $ odo list
   ```

Example output

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

```bash
$ 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:

   ```bash
   $ cd frontend
   ```

2. Create an external URL for the application:

   ```bash
   $ 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:

   ```bash
   $ 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-mypersonalproject.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`:
   ```bash
   $ 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.
   ```bash
   $ 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.
   ```bash
   $ 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:
  ```bash
  $ odo service create
  ```
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                    |
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
```

---

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

| Optional Params | |
| MONGODB_ADMIN_PASSWORD, |
| NAMESPACE, MONGODB_PASSWORD, |
| MONGODB_USER | |
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=NCn41tqmx7Rlmfv
```

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:
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 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 -b jpa-sample https://github.com/redhat-developer/application-stack-samples.git

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 **UnknownHostException** 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.java:144)
   [INFO] [err]   ... 86 more
   ```
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

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>URL</th>
<th>PORT</th>
<th>SECURE</th>
<th>KIND</th>
</tr>
</thead>
<tbody>
<tr>
<td>java-application-8080</td>
<td>Pushed</td>
<td><a href="http://java-application-8080.apps-crc.testing">http://java-application-8080.apps-crc.testing</a></td>
<td>8080</td>
<td>false</td>
<td>ingress</td>
</tr>
</tbody>
</table>

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 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.

2.4.5.3. 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

<table>
<thead>
<tr>
<th>Operators available in the cluster</th>
<th>CRDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td></td>
</tr>
<tr>
<td>postgresql-operator.v0.1.1</td>
<td>Backup, Database</td>
</tr>
</tbody>
</table>

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:

   ```yaml
   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:

   ```yaml
   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.

2.4.5.4. 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

<table>
<thead>
<tr>
<th>NAME</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database/sampledatabase</td>
<td>6m31s</td>
</tr>
</tbody>
</table>

2. Connect the database to your application:

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

3. Push the changes to your cluster:

```bash
$ 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:

```bash
$ odo exec -- bash -c 'env | grep DATABASE'
```

```bash
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 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](https://www.redhat.com/).

**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:
   ```bash
   $ odo login -u developer -p developer
   ```
2. Create a project:
   ```bash
   $ odo project create myproject
   ```

**Example output**

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

### 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:
   ```bash
   $ odo catalog list components
   ```

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

   **Odo Devfile Components:**
<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
<th>REGISTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>java-maven</td>
<td>Upstream Maven and OpenJDK 11</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-openliberty</td>
<td>Open Liberty microservice in Java</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-quarkus</td>
<td>Upstream Quarkus with Java+GraalVM</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-springboot</td>
<td>Spring Boot® using Java</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>nodejs</td>
<td>Stack with NodeJS 12</td>
<td>DefaultDevfileRegistry</td>
</tr>
</tbody>
</table>

   **Odo OpenShift Components:**
<table>
<thead>
<tr>
<th>NAME</th>
<th>PROJECT</th>
<th>TAGS</th>
<th>SUPPORTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>java</td>
<td>openshift</td>
<td>11.8,latest</td>
<td>YES</td>
</tr>
<tr>
<td>dotnet</td>
<td>openshift</td>
<td>2.1,3.1,latest</td>
<td>NO</td>
</tr>
</tbody>
</table>

OpenShift Container Platform 4.7 CLI tools
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-springboot myspring --starter
   ```

   The previous command produces the following output:

   ```
   ✓ 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
   
<table>
<thead>
<tr>
<th>NAME</th>
<th>URL</th>
<th>PORT</th>
<th>SECURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>myspring-8080</td>
<td><a href="http://myspring-8080.apps-crc.testing">http://myspring-8080.apps-crc.testing</a></td>
<td>8080</td>
<td>false</td>
</tr>
</tbody>
</table>

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:

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

4. Delete the storage from your component:

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

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

```
$ odo storage list
```

Example output

The component 'nodejs' has the following storage attached:

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

6. Push the changes to the cluster:

```
$ odo push
```

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:

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>SIZE</th>
<th>PATH</th>
<th>CONTAINER</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>store</td>
<td>1Gi</td>
<td>/data</td>
<td>runtime</td>
<td>Not Pushed</td>
</tr>
</tbody>
</table>

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

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

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

   ```bash
   $ odo preference set Ephemeral false
   $ 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:

   ```bash
   $ 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:

   ```bash
   $ 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:

   ```bash
   $ odo component list
   ```

   **Example output**

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

3. Delete the application:

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

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

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

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]
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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>PROJECT</th>
<th>TAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>dotnet</td>
<td>openshift</td>
<td>3.1,latest</td>
</tr>
<tr>
<td>httpd</td>
<td>openshift</td>
<td>2.4,latest</td>
</tr>
<tr>
<td>java</td>
<td>openshift</td>
<td>8,latest</td>
</tr>
<tr>
<td>nginx</td>
<td>openshift</td>
<td>1.10,1.12,1.8,latest</td>
</tr>
<tr>
<td>nodejs</td>
<td>openshift</td>
<td>0.10,4,6,8,latest</td>
</tr>
<tr>
<td>perl</td>
<td>openshift</td>
<td>5.16,5.20,5.24,latest</td>
</tr>
<tr>
<td>php</td>
<td>openshift</td>
<td>5.5,5.6,7.0,7.1,latest</td>
</tr>
<tr>
<td>python</td>
<td>openshift</td>
<td>2.7,3.3,3.4,3.5,3.6,latest</td>
</tr>
<tr>
<td>ruby</td>
<td>openshift</td>
<td>2.0.2,2.2.3,2.4,latest</td>
</tr>
<tr>
<td>wildfly</td>
<td>openshift</td>
<td>10.0,10.1,8.1,9.0,latest</td>
</tr>
</tbody>
</table>

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:

$
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

- Mirroring images for a disconnected installation
- 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:
2. Copy the encoded root CA certificate to the appropriate location:

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

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

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

   $ 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

```bash
$ echo <content_of_additional_ca> | base64 --decode > disconnect-ca.crt
$ sudo cp ./disconnect-ca.crt /etc/pki/ca-trust/source/anchors/<mirror-registry>.crt
$ sudo update-ca-trust enable && sudo systemctl daemon-reload && sudo systemctl restart /docker && docker login <mirror-registry>:5000 -u <username> -p <password>
$ oc image mirror registry.access.redhat.com/openshiftdo/odo-init-image-rhel7:<tag> <mirror-registry>:5000/openshiftdo/odo-init-image-rhel7:<tag>
$ export ODO_BOOTSTRAPPER_IMAGE=<mirror-registry>:5000/openshiftdo/odo-init-image-rhel7:<tag>
```
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
   ```
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

```yaml
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:

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

4. Trust a CA in your client platform:

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

5. Log into the internal registry:

   ```bash
   $ 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:

   ```bash
   $ 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:

   ```bash
   $ 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:

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

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

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

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

Log into the internal registry:

```
PS C:\> oc get route -n openshift-image-registry
```

Push the `odo` init image:

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

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

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

### 2.5.3. 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:
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:

   ```shell
   $ cd <directory_name>
   ```

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

   ```shell
   $ 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:

   ```shell
   $ 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:

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

5. Push the changes. This creates a URL on the cluster.

   ```shell
   $ odo push
   ```

6. List the URLs to check the desired URL for the component.

   ```shell
   $ odo url list
   ```

7. View your deployed application using the generated URL.

   ```shell
   $ 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**

   ```json
   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: run
    isDefault: true

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 the 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]
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:
```
2. Create a Java application:

```bash
$ 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

```yaml
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
```
3. Push the changes to the cluster:

$ odo push

Example output

```
[...]
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" ...
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
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:

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

2. Create a project:

   ```sh
   $ 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:

  ```sh
  $ 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:

<table>
<thead>
<tr>
<th>Operators available in the cluster</th>
<th>CRDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>etcdoperator.v0.9.4</td>
<td>EtcdCluster, EtcdBackup, EtcdRestore</td>
</tr>
<tr>
<td>mongodb-enterprise.v1.4.5</td>
<td>MongoDB, MongoDBUser, MongoDBOpsManager</td>
</tr>
</tbody>
</table>

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

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

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

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

```
$ 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
spec:
  size: 1
  version: 3.2.13
```

1. Change the name from **example** to **my-etcd-cluster**
2. Reduce the size from **3** to **1**
3. Start a service from the YAML file:

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

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

   $ oc get pods | grep my-etcd-cluster

### 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 \texttt{y} when prompted to install the completion hook.

- To install the completion hook manually, add \texttt{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:

     \[
     \texttt{$ odo --uncomplete}
     \]

  2. Press \texttt{y} when prompted to uninstall the completion hook.

\textbf{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 \texttt{.odoignore} file in the root directory of your application. This applies to both odo push and odo watch.

If the \texttt{.odoignore} file does not exist, the \texttt{.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 \texttt{.odoignore} or the \texttt{.gitignore} file:

\[
\begin{align*}
\texttt{.git} \\
\texttt{* .js} \\
\texttt{tests/}
\end{align*}
\]

The \texttt{.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

\[
\begin{align*}
\texttt{# Delete the application} \\
\texttt{odo app delete myapp} \\
\texttt{# Describe 'webapp' application,} \\
\texttt{odo app describe webapp} \\
\texttt{# List all applications in the current project}
\end{align*}
\]
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

<table>
<thead>
<tr>
<th>Application</th>
<th>Application is the name of application the component needs to be part of</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>The minimum and maximum CPU a component can consume</td>
</tr>
<tr>
<td>Ignore</td>
<td>Consider the .odoignore file for push and watch</td>
</tr>
</tbody>
</table>

Table 2.2. Available Local Parameters:

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

### 2.9.15. 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

```bash
# 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
```
2.9.1.6. debug

Debug a component.

Example using debug

```bash
# 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

```bash
# 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
```
2.9.1.8. 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=xxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

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/preferences.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:

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

### 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
2.9.19. storage

Perform storage operations.

Example using storage

```bash
# 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
```

2.9.19. storage

Perform storage operations.

Example using storage

```bash
# 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
```

2.9.19. unlink

Unlink component or a service.

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

Example using unlink

```bash
# 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

```bash
# 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

```bash
# 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
# 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

```bash
# 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

```bash
# 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 the emptyDir volume:

- Binaries:
  - go-init is a minimal init system. It runs as the first process (PID 1) inside the application container. go-init starts the SupervisorD daemon which runs the developer code. go-init is required to handle orphaned processes.
  - SupervisorD is a process control system. It watches over configured processes and ensures that they are running. It also restarts services when necessary. For odo, SupervisorD executes and monitors the developer code.

- Configuration files:
  - supervisor.conf is the configuration file necessary for the SupervisorD daemon to start.

- Scripts:
  - assemble-and-restart is an OpenShift S2I concept to build and deploy user-source code. The assemble-and-restart script first assembles the user source code inside the application container and then restarts SupervisorD for user changes to take effect.
  - Run is an OpenShift S2I concept of executing the assembled source code. The run script executes the assembled code created by the assemble-and-restart script.
  - s2i-setup is a script that creates files and directories which are necessary for the assemble-and-restart and run scripts to execute successfully. The script is executed whenever the application container starts.
• Directories:
  - language-scripts: OpenShift S2I allows custom assemble and run scripts. A few language specific custom scripts are present in the language-scripts directory. The custom scripts provide additional configuration to make odo debug work.

The emptyDir volume is mounted at the /opt/odo mount point for both the Init container and the application container.

2.10.3.1.2. 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.

<table>
<thead>
<tr>
<th>Container</th>
<th>PVC mounted at</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy-files-to-volume</td>
<td>/mnt</td>
</tr>
<tr>
<td>Application container</td>
<td>/opt/app-root</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Container</th>
<th>emptyDir volume mounted at</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy-supervisord</td>
<td>/opt/odo</td>
</tr>
<tr>
<td>Application container</td>
<td>/opt/odo</td>
</tr>
</tbody>
</table>

### 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

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:

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

2. Make the binary file executable:

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

3. Check the installed version:
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:


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:"b31719a91b79f63ac4887a1c1e6d5e53378e34d93", 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

<table>
<thead>
<tr>
<th>NAME</th>
<th>NAMESPACE</th>
<th>REVISION</th>
<th>UPDATED</th>
<th>STATUS</th>
<th>CHART</th>
<th>APP</th>
<th>VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>example-mysql</td>
<td>mysql</td>
<td>1</td>
<td>2019-12-05 15:06:51.379134163 -0500 EST</td>
<td>deployed</td>
<td>mysql-1.5.0</td>
<td></td>
<td>5.7.27</td>
</tr>
</tbody>
</table>

3.1.4. Creating a custom Helm chart on OpenShift Container Platform

Procedure

1. Create a new project:
1. $ 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:

   ```yaml
   apiVersion: v2  ①
   name: nodejs-ex-k  ②
   description: A Helm chart for OpenShift  ③
   icon: https://static.redhat.com/libs/redhat/brand-assets/latest/corp/logo.svg ④
   
   ① The chart API version. It should be **v2** for Helm charts that require at least Helm 3.
   ② The name of your chart.
   ③ The description of your chart.
   ④ The URL to an image to be used as an icon.
   ```

5. Verify that the chart is formatted properly:
   
   $ helm lint

   **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**
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

   ```yaml
   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:

   ```bash
   $ 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.
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>
   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:

```bash
$ 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**

  ```yaml
  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:
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 \texttt{tkn} CLI is provided as a \texttt{tar.gz} archive.

**Procedure**

1. Download the \texttt{CLI}.
2. Unpack and unzip the archive.
3. Move the \texttt{tkn} binary to a directory on your PATH.
4. To check your PATH, open a terminal window and run:
   \[
   \$ \texttt{echo $PATH}
   \]

5.2. CONFIGURING THE OPENSHIFT PIPELINES TKN CLI

Configure the Red Hat OpenShift Pipelines \texttt{tkn} CLI to enable tab completion.

5.2.1. Enabling tab completion

After you install the \texttt{tkn} CLI, you can enable tab completion to automatically complete \texttt{tkn} commands or suggest options when you press Tab.

**Prerequisites**

- You must have the \texttt{tkn} CLI tool installed.
- You must have \texttt{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:
   \[
   \$ \texttt{tkn completion bash > tkn\_bash\_completion}
   \]
2. Copy the file to \texttt{/etc/bash\_completion.d/}:
   \[
   \$ \texttt{sudo cp tkn\_bash\_completion /etc/bash\_completion.d/}
   \]

   Alternatively, you can save the file to a local directory and source it from your \texttt{.bashrc} file instead.

Tab completion is enabled when you open a new terminal.

5.3. OPENSHIFT PIPELINES TKN REFERENCE

This section lists the basic \texttt{tkn} CLI commands.

5.3.1. Basic syntax

\texttt{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**

```bash
$ tkn pipeline describe mypipeline
```

5.3.4.4. pipeline list

List Pipelines.

**Example: Display a list of Pipelines**

```bash
$ tkn pipeline list
```

5.3.4.5. pipeline logs

Display Pipeline logs for a specific Pipeline.

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

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

5.3.4.6. pipeline start

Start a Pipeline.

**Example: Start mypipeline Pipeline**

```bash
$ tkn pipeline start mypipeline
```

5.3.5. PipelineRun commands

5.3.5.1. pipelinerun

Manage PipelineRuns.

**Example: Display help**

```bash
$ tkn pipelinerun -h
```

5.3.5.2. pipelinerun cancel

Cancel a PipelineRun.

**Example: Cancel the mypipelinerun PipelineRun from a namespace**

```bash
$ 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

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

5.3.8.4. condition list

List Conditions.

Example: List Conditions in a namespace

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

5.3.9. Pipeline Resource management commands

5.3.9.1. resource

Manage Pipeline Resources.

Example: Display help

```bash
$ tkn resource -h
```

5.3.9.2. resource create

Create a Pipeline Resource.

Example: Create a Pipeline Resource in a namespace

```bash
$ 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

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

5.3.9.4. resource describe

Describe a Pipeline Resource.

Example: Describe the myresource Pipeline Resource
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 mytask1
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. eventlistener logs
Display logs of an EventListener.

Example: Display the logs of the mylistener EventListener in a namespace

$ tkn eventlistener logs mylistener -n myspace

5.3.11.6. triggerbinding
Manage TriggerBindings.

Example: Display TriggerBindings help

$ tkn clustertask start mytask
5.3.11.7. triggerbinding delete
Delete a TriggerBinding.

Example: Delete mybinding1 and mybinding2 TriggerBindings in a namespace

$ tkn triggerbinding delete mybinding1 mybinding2 -n myspace

5.3.11.8. triggerbinding describe
Describe a TriggerBinding.

Example: Describe the mybinding TriggerBinding in a namespace

$ tkn triggerbinding describe mybinding -n myspace

5.3.11.9. triggerbinding list
List TriggerBindings.

Example: List all the TriggerBindings in a namespace

$ tkn triggerbinding list -n myspace

5.3.11.10. triggertemplate
Manage TriggerTemplates.

Example: Display TriggerTemplate help

$ tkn triggertemplate -h

5.3.11.11. triggertemplate delete
Delete a TriggerTemplate.

Example: Delete mytemplate1 and mytemplate2 TriggerTemplates in a namespace

$ tkn triggertemplate delete mytemplate1 mytemplate2 -n `myspace`

5.3.11.12. triggertemplate describe
Describe a TriggerTemplate.

Example: Describe the mytemplate TriggerTemplate in a namespace

$ tkn triggertemplate describe mytemplate -n `myspace`
5.3.11.13. `triggertemplate list`
List TriggerTemplates.

**Example: List all the TriggerTemplates in a namespace**

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

5.3.11.14. `clustertriggerbinding`
Manage ClusterTriggerBindings.

**Example: Display ClusterTriggerBindings help**

```
$ tkn clustertriggerbinding -h
```

5.3.11.15. `clustertriggerbinding delete`
Delete a ClusterTriggerBinding.

**Example: Delete myclusterbinding1 and myclusterbinding2 ClusterTriggerBindings**

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

5.3.11.16. `clustertriggerbinding describe`
Describe a ClusterTriggerBinding.

**Example: Describe the myclusterbinding ClusterTriggerBinding**

```
$ tkn clustertriggerbinding describe myclusterbinding
```

5.3.11.17. `clustertriggerbinding list`
List ClusterTriggerBindings.

**Example: List all ClusterTriggerBindings**

```
$ tkn clustertriggerbinding list
```

5.3.12. Hub interaction commands
Interact with Tekton Hub for resources such as tasks and pipelines.

5.3.12.1. `hub`
Interact with hub.

**Example: Display help**

```
$ tkn hub -h
```
Example: Interact with a hub API server

$ tkn hub --api-server https://api.hub.tekton.dev

NOTE
For each example, to get the corresponding sub-commands and flags, run tkn hub <command> --help.

5.3.12.2. hub downgrade

Downgrade an installed resource.

Example: Downgrade the mytask task in the mynamespace namespace to its older version

$ tkn hub downgrade task mytask --to version -n mynamespace

5.3.12.3. hub get

Get a resource manifest by its name, kind, catalog, and version.

Example: Get the manifest for a specific version of the myresource pipeline or task from the tekton catalog

$ tkn hub get [pipeline | task] myresource --from tekton --version version

5.3.12.4. hub info

Display information about a resource by its name, kind, catalog, and version.

Example: Display information about a specific version of the mytask task from the tekton catalog

$ tkn hub info task mytask --from tekton --version version

5.3.12.5. hub install

Install a resource from a catalog by its kind, name, and version.

Example: Install a specific version of the mytask task from the tekton catalog in the mynamespace namespace

$ tkn hub install task mytask --from tekton --version version -n mynamespace

5.3.12.6. hub reinstall

Reinstall a resource by its kind and name.

Example: Reinstall a specific version of the mytask task from the tekton catalog in the mynamespace namespace
$ tkn hub reinstall task mytask --from tekton --version version -n mynamespace

5.3.12.7. hub search

Search a resource by a combination of name, kind, and tags.

**Example:** Search a resource with a tag `cli`

$ tkn hub search --tags cli

5.3.12.8. hub upgrade

Upgrade an installed resource.

**Example:** Upgrade the installed `mytask` task in the `mynamespace` namespace to a new version

$ tkn hub upgrade task mytask --to version -n mynamespace
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. RHEL 8 meets these requirements:
  - `podman` version 1.9.3+ (version 2.0+ recommended)
  - `glibc` version 2.28+

Procedure

1. Navigate to the OpenShift mirror site and download the latest version of the tarball that matches your operating system.

2. Unpack the archive.
   - For Linux or macOS:
     ```
     $ 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 Operator 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 4.7.23 directory, download the latest version of the tarball for Linux.
3. Unpack the archive:
   
   ```bash
   $ tar xvf operator-sdk-v1.3.0-ocp-linux-x86_64.tar.gz
   ```
4. Make the file executable:
   
   ```bash
   $ chmod +x operator-sdk
   ```
5. Move the extracted `operator-sdk` binary to a directory that is on your `PATH`.

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

TIP

To check your `PATH`:

```bash
$ echo $PATH
```

```bash
$ 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

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

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
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

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bash</td>
<td>Generate bash completions.</td>
</tr>
<tr>
<td>zsh</td>
<td>Generate zsh completions.</td>
</tr>
</tbody>
</table>

Table 7.4. completion flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Usage help output.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Help output for the <code>run bundle</code> subcommand.</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--apis-dir</code> (string)</td>
<td>Root directory for API type definitions.</td>
</tr>
<tr>
<td><code>-h, --help</code></td>
<td>Help for <code>generate kustomize manifests</code>.</td>
</tr>
<tr>
<td><code>--input-dir</code> (string)</td>
<td>Directory containing existing Kustomize files.</td>
</tr>
<tr>
<td><code>--interactive</code></td>
<td>When set to <code>false</code>, if no Kustomize base exists, an interactive command prompt is presented to accept custom metadata.</td>
</tr>
</tbody>
</table>
### 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

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--help, -h</td>
<td>Help output for the <strong>init</strong> command.</td>
</tr>
<tr>
<td>--plugins (string)</td>
<td>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.</td>
</tr>
<tr>
<td>--project-version</td>
<td>Project version. Available values are 2 and 3-alpha, which is the default.</td>
</tr>
</tbody>
</table>

### 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

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

### 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

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--timeout</code> &lt;duration&gt;</td>
<td>Upgrade timeout. The default value is <code>2m0s</code>.</td>
</tr>
<tr>
<td><code>--kubeconfig</code> (string)</td>
<td>Path to the <code>kubeconfig</code> file to use for CLI requests.</td>
</tr>
<tr>
<td><code>--namespace</code> (string)</td>
<td>If present, namespace in which to run the CLI request.</td>
</tr>
<tr>
<td><code>-h, --help</code></td>
<td>Help output for the <code>run bundle</code> subcommand.</td>
</tr>
</tbody>
</table>

#### 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

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

Additional resources

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