OpenShift Container Platform 4.10

CLI tools

Learning how to use the command-line tools for OpenShift Container Platform
OpenShift Container Platform 4.10 CLI tools

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

Copyright © 2022 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at http://creativecommons.org/licenses/by-sa/3.0/. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

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

## CHAPTER 1. OPENSSHIFT CONTAINER PLATFORM CLI TOOLS OVERVIEW

1. LIST OF CLI TOOLS

## CHAPTER 2. OPENSSHIFT CLI (OC)

2.1. GETTING STARTED WITH THE OPENSSHIFT CLI

2.1.1. About the OpenShift CLI
2.1.2. Installing the OpenShift CLI
   2.1.2.1. Installing the OpenShift CLI by downloading the binary
   2.1.2.2. Installing the OpenShift CLI by using the web console
   2.1.2.2.1. Installing the OpenShift CLI on Linux using the web console
   2.1.2.2.2. Installing the OpenShift CLI on Windows using the web console
   2.1.2.2.3. Installing the OpenShift CLI on macOS using the web console
2.1.2.3. Installing the OpenShift CLI by using an RPM
2.1.3. Logging in to the OpenShift CLI
2.1.4. Using the OpenShift CLI
   2.1.4.1. Creating a project
   2.1.4.2. Creating a new app
   2.1.4.3. Viewing pods
   2.1.4.4. Viewing pod logs
   2.1.4.5. Viewing the current project
   2.1.4.6. Viewing the status for the current project
   2.1.4.7. Listing supported API resources
2.1.5. Getting help
2.1.6. Logging out of the OpenShift CLI

2.2. CONFIGURING THE OPENSHEET CLI

2.2.1. Enabling tab completion

2.3. MANAGING CLI PROFILES

2.3.1. About switches between CLI profiles
2.3.2. Manual configuration of CLI profiles
2.3.3. Load and merge rules

2.4. EXTENDING THE OPENSHEET CLI WITH PLUG-INS

2.4.1. Writing CLI plug-ins
2.4.2. Installing and using CLI plug-ins

2.5. OPENSHEET CLI DEVELOPER COMMAND REFERENCE

2.5.1. OpenShift CLI (oc) developer commands
   2.5.1.1. oc annotate
   2.5.1.2. oc api-resources
   2.5.1.3. oc api-versions
   2.5.1.4. oc apply
   2.5.1.5. oc apply edit-last-applied
   2.5.1.6. oc apply set-last-applied
   2.5.1.7. oc apply view-last-applied
   2.5.1.8. oc attach
   2.5.1.9. oc auth can-i
   2.5.1.10. oc auth reconcile
   2.5.1.11. oc autoscale
   2.5.1.12. oc cancel-build
   2.5.1.13. oc cluster-info
<table>
<thead>
<tr>
<th>Command</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5.1.14. oc cluster-info dump</td>
<td>36</td>
</tr>
<tr>
<td>2.5.1.15. oc completion</td>
<td>37</td>
</tr>
<tr>
<td>2.5.1.16. oc config current-context</td>
<td>38</td>
</tr>
<tr>
<td>2.5.1.17. oc config delete-cluster</td>
<td>38</td>
</tr>
<tr>
<td>2.5.1.18. oc config delete-context</td>
<td>38</td>
</tr>
<tr>
<td>2.5.1.19. oc config delete-user</td>
<td>38</td>
</tr>
<tr>
<td>2.5.1.20. oc config get-clusters</td>
<td>38</td>
</tr>
<tr>
<td>2.5.1.21. oc config get-contexts</td>
<td>38</td>
</tr>
<tr>
<td>2.5.1.22. oc config get-users</td>
<td>39</td>
</tr>
<tr>
<td>2.5.1.23. oc config rename-context</td>
<td>39</td>
</tr>
<tr>
<td>2.5.1.24. oc config set</td>
<td>39</td>
</tr>
<tr>
<td>2.5.1.25. oc config set-cluster</td>
<td>39</td>
</tr>
<tr>
<td>2.5.1.26. oc config set-context</td>
<td>40</td>
</tr>
<tr>
<td>2.5.1.27. oc config set-credentials</td>
<td>40</td>
</tr>
<tr>
<td>2.5.1.28. oc config unset</td>
<td>41</td>
</tr>
<tr>
<td>2.5.1.29. oc config use-context</td>
<td>41</td>
</tr>
<tr>
<td>2.5.1.30. oc config view</td>
<td>41</td>
</tr>
<tr>
<td>2.5.1.31. oc cp</td>
<td>41</td>
</tr>
<tr>
<td>2.5.1.32. oc create</td>
<td>42</td>
</tr>
<tr>
<td>2.5.1.33. oc create build</td>
<td>42</td>
</tr>
<tr>
<td>2.5.1.34. oc create clusterresourcequota</td>
<td>42</td>
</tr>
<tr>
<td>2.5.1.35. oc create clusterrole</td>
<td>43</td>
</tr>
<tr>
<td>2.5.1.36. oc create clusterrolebinding</td>
<td>43</td>
</tr>
<tr>
<td>2.5.1.37. oc create configmap</td>
<td>43</td>
</tr>
<tr>
<td>2.5.1.38. oc create cronjob</td>
<td>44</td>
</tr>
<tr>
<td>2.5.1.39. oc create deployment</td>
<td>44</td>
</tr>
<tr>
<td>2.5.1.40. oc create deploymentconfig</td>
<td>44</td>
</tr>
<tr>
<td>2.5.1.41. oc create identity</td>
<td>44</td>
</tr>
<tr>
<td>2.5.1.42. oc create imagestream</td>
<td>45</td>
</tr>
<tr>
<td>2.5.1.43. oc create imagestreamtag</td>
<td>45</td>
</tr>
<tr>
<td>2.5.1.44. oc create ingress</td>
<td>45</td>
</tr>
<tr>
<td>2.5.1.45. oc create job</td>
<td>46</td>
</tr>
<tr>
<td>2.5.1.46. oc create namespace</td>
<td>46</td>
</tr>
<tr>
<td>2.5.1.47. oc create poddisruptionbudget</td>
<td>46</td>
</tr>
<tr>
<td>2.5.1.48. oc create priorityclass</td>
<td>46</td>
</tr>
<tr>
<td>2.5.1.49. oc create quota</td>
<td>47</td>
</tr>
<tr>
<td>2.5.1.50. oc create role</td>
<td>47</td>
</tr>
<tr>
<td>2.5.1.51. oc create rolebinding</td>
<td>47</td>
</tr>
<tr>
<td>2.5.1.52. oc create route edge</td>
<td>48</td>
</tr>
<tr>
<td>2.5.1.53. oc create route passthrough</td>
<td>48</td>
</tr>
<tr>
<td>2.5.1.54. oc create route reencrypt</td>
<td>48</td>
</tr>
<tr>
<td>2.5.1.55. oc create secret docker-registry</td>
<td>48</td>
</tr>
<tr>
<td>2.5.1.56. oc create secret generic</td>
<td>49</td>
</tr>
<tr>
<td>2.5.1.57. oc create secret tls</td>
<td>49</td>
</tr>
<tr>
<td>2.5.1.58. oc create service clusterip</td>
<td>49</td>
</tr>
<tr>
<td>2.5.1.59. oc create service externalname</td>
<td>49</td>
</tr>
<tr>
<td>2.5.1.60. oc create service loadbalancer</td>
<td>50</td>
</tr>
<tr>
<td>2.5.1.61. oc create service nodeport</td>
<td>50</td>
</tr>
<tr>
<td>2.5.1.62. oc create serviceaccount</td>
<td>50</td>
</tr>
<tr>
<td>2.5.1.63. oc create user</td>
<td>50</td>
</tr>
<tr>
<td>2.5.1.64. oc create useridentitymapping</td>
<td>50</td>
</tr>
<tr>
<td>2.5.1.65. oc debug</td>
<td>51</td>
</tr>
<tr>
<td>2.5.1.66. oc delete</td>
<td>51</td>
</tr>
</tbody>
</table>
2.5.1.120. oc serviceaccounts new-token
2.5.1.121. oc set build-hook
2.5.1.122. oc set build-secret
2.5.1.123. oc set data
2.5.1.124. oc set deployment-hook
2.5.1.125. oc set env
2.5.1.126. oc set image
2.5.1.127. oc set image-lookup
2.5.1.128. oc set probe
2.5.1.129. oc set resources
2.5.1.130. oc set route-backends
2.5.1.131. oc set selector
2.5.1.132. oc set serviceaccount
2.5.1.133. oc set subject
2.5.1.134. oc set triggers
2.5.1.135. oc set volumes
2.5.1.136. oc start-build
2.5.1.137. oc status
2.5.1.138. oc tag
2.5.1.139. oc version
2.5.1.140. oc wait
2.5.1.141. oc whoami

2.5.2. Additional resources

2.6. OPENSHIFT CLI ADMINISTRATOR COMMAND REFERENCE

2.6.1. OpenShift CLI (oc) administrator commands
2.6.1.1. oc adm build-chain
2.6.1.2. oc adm catalog mirror
2.6.1.3. oc adm certificate approve
2.6.1.4. oc adm certificate deny
2.6.1.5. oc adm cordon
2.6.1.6. oc adm create-bootstrap-project-template
2.6.1.7. oc adm create-error-template
2.6.1.8. oc adm create-login-template
2.6.1.9. oc adm create-provider-selection-template
2.6.1.10. oc adm drain
2.6.1.11. oc adm groups add-users
2.6.1.12. oc adm groups new
2.6.1.13. oc adm groups prune
2.6.1.14. oc adm groups remove-users
2.6.1.15. oc adm groups sync
2.6.1.16. oc adm inspect
2.6.1.17. oc adm migrate template-instances
2.6.1.18. oc adm must-gather
2.6.1.19. oc adm new-project
2.6.1.20. oc adm node-logs
2.6.1.21. oc adm pod-network isolate-projects
2.6.1.22. oc adm pod-network join-projects
2.6.1.23. oc adm pod-network make-projects-global
2.6.1.24. oc adm policy add-role-to-user
2.6.1.25. oc adm policy add-scc-to-group
2.6.1.26. oc adm policy add-scc-to-user
2.6.1.27. oc adm policy scc-review
2.6.1.28. oc adm policy scc-subject-review
CHAPTER 3. DEVELOPER CLI (ODO) .................................................. 95
3.1. ODO RELEASE NOTES ...................................................... 95
  3.1.1. Notable changes and improvements in odo version 2.5.0 ....... 95
  3.1.2. Bug fixes .................................................................... 95
  3.1.3. Getting support ......................................................... 95
3.2. UNDERSTANDING ODO .................................................... 96
  3.2.1. odo key features ....................................................... 96
  3.2.2. odo core concepts ................................................... 96
  3.2.3. Listing components in odo ........................................ 97
  3.2.4. Telemetry in odo ....................................................... 98
3.3. INSTALLING ODO .......................................................... 99
  3.3.1. Installing odo on Linux ............................................. 99
  3.3.2. Installing odo on Windows ......................................... 100
  3.3.3. Installing odo on macOS .......................................... 100
  3.3.4. Installing odo on VS Code ....................................... 101
  3.3.5. Installing odo on Red Hat Enterprise Linux (RHEL) using an RPM .............................................. 101
3.4. CREATING AND DEPLOYING APPLICATIONS WITH ODO .... 102
  3.4.1. Working with projects ............................................. 102
  3.4.1.1. Creating a project ............................................... 102
  3.4.2. Creating a single-component application with odo ...... 103
    3.4.2.1. Creating a project ......................................... 103
    3.4.2.2. Creating a Node.js application with odo ............ 103
    3.4.2.3. Modifying your application code ....................... 104
    3.4.2.4. Adding storage to the application components .... 104
    3.4.2.5. Adding a custom builder to specify a build image ... 105
    3.4.2.6. Connecting your application to multiple services using OpenShift Service Catalog .................. 106
  3.4.2.7. Deleting an application ....................................... 106
  3.4.3. Creating a multicomponent application with odo ........ 107
    3.4.3.1. Creating a project ......................................... 107
    3.4.3.2. Deploying the back-end component .................... 108
    3.4.3.3. Deploying the front-end component .................... 111
    3.4.3.4. Linking both components .................................. 112
3.4.3.5. Exposing components to the public
3.4.3.6. Modifying the running application
3.4.3.7. Deleting an application
3.4.4. Creating an application with a database
3.4.4.1. Creating a project
3.4.4.2. Deploying the front-end component
3.4.4.3. Deploying a database in interactive mode
3.4.4.4. Deploying a database manually
3.4.4.5. Connecting the database to the front-end application
3.4.4.6. Creating a Java application with a database
3.4.4.7. Creating a project
3.4.4.8. Creating a Java MicroServices JPA application
3.4.4.9. Creating a database with odo
3.4.4.10. Connecting a Java application to a database
3.4.5. Using devfiles in odo
3.4.5.1. About the devfile in odo
3.4.5.2. Creating a Java application by using a devfile
3.4.5.2.1. Creating a project
3.4.5.2.2. Listing available devfile components
3.4.5.2.3. Deploying a Java application using a devfile
3.4.5.3. Converting an S2I component into a devfile component
3.4.6. Working with storage
3.4.6.1. Adding storage to the application components
3.4.6.2. Adding storage to a specific container
3.4.6.3. Switching between ephemeral and persistent storage
3.4.7. Deleting applications
3.4.8. Debugging applications in odo
3.4.8.1. Debugging an application
3.4.8.2. Configuring debugging parameters
3.4.9. Sample applications
3.4.9.1. Git repository example applications
3.4.9.1.1. httpd
3.4.9.1.2. java
3.4.9.1.3. nodejs
3.4.9.1.4. perl
3.4.9.1.5. php
3.4.9.1.6. python
3.4.9.1.7. ruby
3.4.9.2. Binary example applications
3.4.9.2.1. java
3.5. USING ODO IN A RESTRICTED ENVIRONMENT
3.5.1. About odo in a restricted environment
3.5.2. Pushing the odo init image to the restricted cluster registry
3.5.2.1. Prerequisites
3.5.2.2. Pushing the odo init image to a mirror registry
3.5.2.2.1. Pushing the init image to a mirror registry on Linux
3.5.2.2.2. Pushing the init image to a mirror registry on MacOS
3.5.2.2.3. Pushing the init image to a mirror registry on Windows
3.5.2.3. Pushing the odo init image to an internal registry directly
3.5.2.3.1. Pushing the init image directly on Linux
3.5.2.3.2. Pushing the init image directly on MacOS
3.5.2.3.3. Pushing the init image directly on Windows
3.5.3. Creating and deploying a component to the disconnected cluster
3.5.3.1. Prerequisites
3.5.3.2. Mirroring a supported builder image
3.5.3.3. Overwriting the mirror registry
3.5.3.4. Creating a Node.js application with odo
3.5.4. Creating and deploying devfile components to the disconnected cluster
3.5.4.1. Creating a NodeJS application by using a devfile in a disconnected cluster
3.5.4.2. Creating a Java application by using a devfile in a disconnected cluster
3.6. CREATING INSTANCES OF SERVICES MANAGED BY OPERATORS
3.6.1. Prerequisites
3.6.2. Creating a project
3.6.3. Listing available services from the Operators installed on the cluster
3.6.4. Creating a service from an Operator
3.6.5. Creating services from YAML files
3.7. MANAGING ENVIRONMENT VARIABLES
3.7.1. Setting and unsetting environment variables
3.8. CONFIGURING THE ODO CLI
3.8.1. Viewing the current configuration
3.8.2. Setting a value
3.8.3. Setting a value
3.8.4. Preference key table
3.8.5. Ignoring files or patterns
3.9. ODO CLI REFERENCE
3.9.1. odo build-images
3.9.2. odo catalog
3.9.2.1. Components
3.9.2.1.1. Listing components
3.9.2.1.2. Getting information about a component
3.9.2.2. Services
3.9.2.2.1. Listing services
3.9.2.2.2. Searching services
3.9.2.2.3. Getting information about a service
3.9.3. odo create
3.9.3.1. Creating a component
3.9.3.2. Starter projects
3.9.3.3. Using an existing devfile
3.9.3.4. Interactive creation
3.9.4. odo delete
3.9.4.1. Deleting a component
3.9.4.2. Undeploying devfile Kubernetes components
3.9.4.3. Delete all
3.9.4.4. Available flags
3.9.5. odo deploy
3.9.6. odo link
3.9.6.1. Various linking options
3.9.6.1.1. Default behavior
3.9.6.1.2. The --inlined flag
3.9.6.1.3. The --map flag
3.9.6.1.4. The --bind-as-files flag
3.9.6.2. Examples
3.9.6.2.1. Default odo link
3.9.6.2.2. Using odo link with the --inlined flag
3.9.6.2.3. Custom bindings
3.9.6.2.3.1. To inline or not?
3.9.6.3. Binding as files
3.9.6.4. --bind-as-files examples
  3.9.6.4.1. Using the default odo link
  3.9.6.4.2. Using --inlined
  3.9.6.4.3. Custom bindings
3.9.7. odo registry
  3.9.7.1. Listing the registries
  3.9.7.2. Adding a registry
  3.9.7.3. Deleting a registry
  3.9.7.4. Updating a registry
3.9.8. odo service
  3.9.8.1. Creating a new service
    3.9.8.1.1. Inlining the manifest
    3.9.8.1.2. Configuring the service
      3.9.8.1.2.1. Using command-line arguments
      3.9.8.1.2.2. Using a file
    3.9.8.2. Deleting a service
    3.9.8.3. Listing services
    3.9.8.4. Getting information about a service
3.9.9. odo storage
  3.9.9.1. Adding a storage volume
  3.9.9.2. Listing the storage volumes
  3.9.9.3. Deleting a storage volume
  3.9.9.4. Adding storage to specific container
3.9.10. Common flags
3.9.11. JSON output
3.10. ODO ARCHITECTURE
  3.10.1. Developer setup
  3.10.2. OpenShift source-to-image
  3.10.3. OpenShift cluster objects
    3.10.3.1. Init Containers
      3.10.3.1.1. copy-supervisord
      3.10.3.1.2. copy-files-to-volume
    3.10.3.2. Application container
    3.10.3.3. Persistent volumes and persistent volume claims
    3.10.3.4. emptyDir volume
    3.10.3.5. Service
  3.10.4. odo push workflow

CHAPTER 4. KNATIVE CLI (KN) FOR USE WITH OPENSHIFT SERVERLESS  ........................................... 186
4.1. KEY FEATURES
4.2. INSTALLING THE KNATIVE CLI

CHAPTER 5. PIPELINES CLI (TKN) .............................................................................. 187
5.1. INSTALLING TKN
  5.1.1. Installing Red Hat OpenShift Pipelines CLI (tkn) on Linux
  5.1.2. Installing Red Hat OpenShift Pipelines CLI (tkn) on Linux using an RPM
  5.1.3. Installing Red Hat OpenShift Pipelines CLI (tkn) on Windows
  5.1.4. Installing Red Hat OpenShift Pipelines CLI (tkn) on macOS
5.2. CONFIGURING THE OPENSHIFT PIPELINES TKN CLI
  5.2.1. Enabling tab completion
5.3. OPENSHIFT PIPELINES TKN REFERENCE
5.3.1. Basic syntax
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. Pipeline run 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. Task run 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 197
5.3.11.3. eventlistener describe 197
5.3.11.4. eventlistener list 198
5.3.11.5. eventlistener logs 198
5.3.11.6. triggerbinding 198
5.3.11.7. triggerbinding delete 198
5.3.11.8. triggerbinding describe 198
5.3.11.9. triggerbinding list 198
5.3.11.10. triggertemplate 198
5.3.11.11. triggertemplate delete 199
5.3.11.12. triggertemplate describe 199
5.3.11.13. triggertemplate list 199
5.3.11.14. clustertriggerbinding 199
5.3.11.15. clustertriggerbinding delete 199
5.3.11.16. clustertriggerbinding describe 199
5.3.11.17. clustertriggerbinding list 200
5.3.12. Hub interaction commands 200
5.3.12.1. hub 200
5.3.12.2. hub downgrade 200
5.3.12.3. hub get 200
5.3.12.4. hub info 201
5.3.12.5. hub install 201
5.3.12.6. hub reinstall 201
5.3.12.7. hub search 201
5.3.12.8. hub upgrade 201

CHAPTER 6. OPM CLI .......................................................... 202
6.1. INSTALLING THE OPM CLI 202
6.1.1. About the opm CLI 202
6.1.2. Installing the opm CLI 202
6.1.3. Additional resources 203
6.2. OPM CLI REFERENCE 203
6.2.1. index 204
6.2.1.1. add 204
6.2.1.2. prune 205
6.2.1.3. prune-stranded 206
6.2.1.4. rm 206
6.2.2. init 207
6.2.3. render 208
6.2.4. validate 208
6.2.5. serve 208

CHAPTER 7. OPERATOR SDK .................................................. 210
7.1. INSTALLING THE OPERATOR SDK CLI 210
7.1.1. Installing the Operator SDK CLI 210
7.2. OPERATOR SDK CLI REFERENCE 211
7.2.1. bundle 211
7.2.1.1. validate 211
7.2.2. cleanup 211
7.2.3. completion 212
7.2.4. create 212
7.2.4.1. api 212
7.2.5. generate 213
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
CHAPTER 1. OPENSШIFT CONTAINER PLATFORM CLI TOOLS OVERVIEW

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

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

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

1.1. LIST OF CLI TOOLS

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

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

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

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

- **opm CLI:** The opm CLI tool helps the Operator developers and cluster administrators to create and maintain the catalogs of Operators from the terminal.

- **Operator SDK:** The Operator SDK, a component of the Operator Framework, provides a CLI tool that Operator developers can use to build, test, and deploy an Operator from the terminal. It simplifies the process of building Kubernetes-native applications, which can require deep, application-specific operational knowledge.
CHAPTER 2. OPENSIFT CLI (OC)

2.1. GETTING STARTED WITH THE OPENSIFT CLI

2.1.1. About the OpenShift CLI

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

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

2.1.2. Installing the OpenShift CLI

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

2.1.2.1. Installing the OpenShift CLI by downloading the binary

You can install the OpenShift CLI (`oc`) 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 4.10. Download and install the new version of `oc`.

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 v4.10 Linux Client** entry and save the file.
4. Unpack the archive:

   ```bash
   $ tar xvzf <file>
   ```
5. Place the `oc` binary in a directory that is on your **PATH**. To check your **PATH**, execute the following command:

   ```bash
   $ echo $PATH
   ```

After you install the OpenShift CLI, it is available using the `oc` command:
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 v4.10 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>
```

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 v4.10 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>
```

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

If you installed an earlier version of `oc`, you cannot use it to complete all of the commands in OpenShift Container Platform 4.10. Download and install the new version of `oc`.

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

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

**Procedure**

1. From the web console, click `?`.
2. Click `Command Line Tools`.
3. Select appropriate `oc` binary for your Linux platform, and then click `Download oc for Linux`
4. Save the file.
5. Unpack the archive.
   
   ```bash
   $ tar xvzf <file>
   ```
6. Move the `oc` binary to a directory that is on your `PATH`.
   To check your `PATH`, execute the following command:
   
   ```bash
   $ echo $PATH
   ```
   
   After you install the OpenShift CLI, it is available using the `oc` command:
   
   ```bash
   $ oc <command>
   ```

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

**Procedure**

1. From the web console, click ?.

2. Click **Command Line Tools**.

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

4. Save the file.

5. Unzip the archive with a ZIP program.

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

   ```
   C:\> path
   ```

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

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

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

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

**Procedure**

1. From the web console, click ?.
2. Click **Command Line Tools**

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

4. Save the file.

5. Unpack and unzip the archive.

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

   ```bash
   $ echo $PATH
   ```

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

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

2.1.2.3 Installing the OpenShift CLI by using an RPM

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

**Prerequisites**

- Must have root or sudo privileges.

**Procedure**

1. Register with Red Hat Subscription Manager:

   ```bash
   # subscription-manager register
   ```

2. Pull the latest subscription data:
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 4.10.
   - For Red Hat Enterprise Linux 8:
     ```bash
     # subscription-manager repos --enable="rhocp-4.10-for-rhel-8-x86_64-rpms"
     ```
   - For Red Hat Enterprise Linux 7:
     ```bash
     # subscription-manager repos --enable="rhel-7-server-ose-4.10-rpms"
     ```

6. Install the `openshift-clients` package:

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

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

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

### 2.1.3. Logging in to the OpenShift CLI

You can log in to the 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:

   ```bash
   $ oc login -u user1
   ```
2. When prompted, enter the required information:

Example output

Enter the OpenShift Container Platform server URL.

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.

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

NOTE

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

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

2.1.4. Using the OpenShift CLI

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

2.1.4.1. Creating a project

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

```bash
$ oc new-project my-project
```

Example output

Now using project "my-project" on server "https://openshift.example.com:6443".
2.1.4.2. Creating a new app

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

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

Example output

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

2.1.4.3. Viewing pods

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

```
NOTE
When you run oc inside a pod and do not specify a namespace, the namespace of the pod is used by default.
```

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

Example output

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

2.1.4.4. Viewing pod logs

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

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

Example output

```
--> Scaling cakephp-ex-1 to 1
--> Success
```

2.1.4.5. Viewing the current project

Use the `oc project` command to view the current project.
2.1.4.6. Viewing the status for the current project

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

```bash
$ oc status

Example output
```

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

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

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

2.1.4.7. Listing supported API resources

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

```bash
$ oc api-resources

Example output
```

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

2.1.5. Getting help

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

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

  ```bash
  $ oc help
  
  Example output
  ```
OpenShift Client

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

Usage:
```
oc [flags]
```

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

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

  **Example: Get help for the oc create command**

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

  **Example output**

  Create a resource by filename or stdin

  JSON and YAML formats are accepted.

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

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

  **Example: View documentation for the Pod resource**

  ```
  $ oc explain pods
  ```

  **Example output**

  KIND: Pod

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

  FIELDS:
  apiVersion <string>
  APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal
2.1.6. Logging out of the OpenShift CLI

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

- Use the `oc logout` command.

  ```$ oc logout```

**Example output**

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

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

2.2. CONFIGURING THE OPENSHIFT CLI

2.2.1. Enabling tab completion

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

**Prerequisites**

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

**Procedure**

The following procedure enables tab completion for Bash.

1. Save the Bash completion code to a file.

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

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

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

   You can also save the file to a local directory and source it from your `.bashrc` file instead.

Tab completion is enabled when you open a new terminal.

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

### 2.3.1. About switches between CLI profiles

Contexts allow you to easily switch between multiple users across multiple OpenShift Container Platform servers, or clusters, when using CLI operations. Nicknames make managing CLI configurations easier by providing short-hand references to contexts, user credentials, and cluster details. After logging in with the CLI for the first time, OpenShift Container Platform creates a `~/.kube/config` file if one does not already exist. As more authentication and connection details are provided to the CLI, either automatically during an `oc login` operation or by manually configuring CLI profiles, the updated information is stored in the configuration file:

**CLI config file**

```yaml
apiVersion: v1
clusters: 1
- cluster:
  insecure-skip-tls-verify: true
  server: https://openshift1.example.com:8443
  name: openshift1.example.com:8443
- cluster:
  insecure-skip-tls-verify: true
  server: https://openshift2.example.com:8443
  name: openshift2.example.com:8443
contexts: 2
- context:
  cluster: openshift1.example.com:8443
  namespace: alice-project
  user: alice/openshift1.example.com:8443
  name: alice-project/openshift1.example.com:8443/alice
- context:
  cluster: openshift1.example.com:8443
  namespace: joe-project
  user: alice/openshift1.example.com:8443
  name: joe-project/openshift1/alice
current-context: joe-project/openshift1.example.com:8443/alice
kind: Config
preferences: {}
users: 4
- name: alice/openshift1.example.com:8443
  user:
    token: xZHd2piv5_9vQrg-SKXRJ2DsI9SceNJdhNTl]jEKIb8k
```

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

2. This **contexts** section defines two contexts: one nicknamed `alice-project/openshift1.example.com:8443/alice`, using the `alice-project` project, `openshift1.example.com:8443` cluster, and `alice` user, and another nicknamed `joe-project/openshift1.example.com:8443/alice`, using the `joe-project` project, `openshift1.example.com:8443` cluster and `alice` user.
The **current-context** parameter shows that the `joe-project/openshift1.example.com:8443/alice` context is currently in use, allowing the **alice** user to work in the **joe-project** project on the

The **users** section defines user credentials. In this example, the user nickname `alice/openshift1.example.com:8443` uses an access token.

The CLI can support multiple configuration files which are loaded at runtime and merged together along with any override options specified from the command line. After you are logged in, you can use the **oc status** or **oc project** command to verify your current working environment:

**Verify the current working environment**

```
$ oc status
```

**Example output**

```
oc status
In project Joe's Project (joe-project)

service database (172.30.43.12:5434 -> 3306)
database deploys docker.io/openshift/mysql-55-centos7:latest
  #1 deployed 25 minutes ago - 1 pod

service frontend (172.30.159.137:5432 -> 8080)
frontend deploys origin-ruby-sample:latest <-
built builds https://github.com/openshift/ruby-hello-world with joe-project/ruby-20-centos7:latest
  #1 deployed 22 minutes ago - 2 pods

To see more information about a service or deployment, use 'oc describe service <name>' or 'oc describe dc <name>'.
You can use 'oc get all' to see lists of each of the types described in this example.
```

**List the current project**

```
$ oc project
```

**Example output**

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

You can run the **oc login** command again and supply the required information during the interactive process, to log in using any other combination of user credentials and cluster details. A context is constructed based on the supplied information if one does not already exist. If you are already logged in and want to switch to another project the current user already has access to, use the **oc project** command and enter the name of the project:

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

**Example output**

```
Now using project "alice-project" on server "https://openshift1.example.com:8443".
```
At any time, you can use the `oc config view` command to view your current CLI configuration, as seen in the output. Additional CLI configuration commands are also available for more advanced usage.

**NOTE**

If you have access to administrator credentials but are no longer logged in as the default system user `system:admin`, you can log back in as this user at any time as long as the credentials are still present in your CLI config file. The following command logs in and switches to the default project:

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

### 2.3.2. Manual configuration of CLI profiles

**NOTE**

This section covers more advanced usage of CLI configurations. In most situations, you can use the `oc login` and `oc project` commands to log in and switch between contexts and projects.

If you want to manually configure your CLI config files, you can use the `oc config` command instead of directly modifying the files. The `oc config` command includes a number of helpful sub-commands for this purpose:

**Table 2.1. CLI configuration subcommands**

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>set-cluster</strong></td>
<td>Sets a cluster entry in the CLI config file. If the referenced cluster nickname already exists, the specified information is merged in.</td>
</tr>
</tbody>
</table>
|              | `-server=<master_ip_or_fqdn>`
|              | `-certificate-authority=<path/to/certificate/authority>`
|              | `-api-version=<apiversion>`
<p>|              | <code>--insecure-skip-tls-verify=true</code>                                                                                                     |
|              | <code>$ oc config set-cluster &lt;cluster_nickname&gt; [-server=&lt;master_ip_or_fqdn&gt;] [-certificate-authority=&lt;path/to/certificate/authority&gt;]</code>      |
|              | <code>[--api-version=&lt;apiversion&gt;] --insecure-skip-tls-verify=true</code>                                                                          |
| <strong>set-context</strong> | Sets a context entry in the CLI config file. If the referenced context nickname already exists, the specified information is merged in. |
|              | <code>$ oc config set-context &lt;context_nickname&gt; [-cluster=&lt;cluster_nickname&gt;]</code>                                                             |
|              | <code>[--user=&lt;user_nickname&gt;] [--namespace=&lt;namespace&gt;]</code>                                                                                   |
| <strong>use-context</strong> | Sets the current context using the specified context nickname.                                                                           |
|              | <code>$ oc config use-context &lt;context_nickname&gt;</code>                                                                                           |</p>
<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>set</strong></td>
<td>Sets an individual value in the CLI config file.</td>
</tr>
<tr>
<td></td>
<td>$ oc config set &lt;property_name&gt; &lt;property_value&gt;</td>
</tr>
<tr>
<td></td>
<td>The <code>&lt;property_name&gt;</code> is a dot-delimited name where each token represents either an attribute name or a map key. The <code>&lt;property_value&gt;</code> is the new value being set.</td>
</tr>
<tr>
<td><strong>unset</strong></td>
<td>Unsets individual values in the CLI config file.</td>
</tr>
<tr>
<td></td>
<td>$ oc config unset &lt;property_name&gt;</td>
</tr>
<tr>
<td></td>
<td>The <code>&lt;property_name&gt;</code> is a dot-delimited name where each token represents either an attribute name or a map key.</td>
</tr>
<tr>
<td><strong>view</strong></td>
<td>Displays the merged CLI configuration currently in use.</td>
</tr>
<tr>
<td></td>
<td>$ oc config view</td>
</tr>
<tr>
<td></td>
<td>Displays the result of the specified CLI config file.</td>
</tr>
<tr>
<td></td>
<td>$ oc config view --config=&lt;specific_filename&gt;</td>
</tr>
</tbody>
</table>

**Example usage**

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

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

- View the cluster entry automatically created:

```
$ oc config view
```

**Example output**

```
apiVersion: v1
clusters:
- cluster:
  insecure-skip-tls-verify: true
  server: https://openshift1.example.com
  name: openshift1-example-com
contexts:
- context:
  cluster: openshift1-example-com
  namespace: default
  user: alice/openshift1-example-com
  name: default/openshift1-example-com/alice
```
current-context: default/openshift1-example-com/alice
kind: Config
preferences: {}
users:
- name: alice/openshift1.example.com
  user:
    token: ns7yVhuRNpDM9cgzfhxQ7bM5s7N2ZVrkZepSRf4LC0

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

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

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

  $ oc whoami -c

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

### 2.3.3. Load and merge rules

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

- CLI config files are retrieved from your workstation, using the following hierarchy and merge rules:
  - If the `--config` option is set, then only that file is loaded. The flag is set once and no merging takes place.
  - If the `$KUBECONFIG` environment variable is set, then it is used. The variable can be a list of paths, and if so the paths are merged together. When a value is modified, it is modified in the file that defines the stanza. When a value is created, it is created in the first file that exists. If no files in the chain exist, then it creates the last file in the list.
  - Otherwise, the `~/.kube/config` file is used and no merging takes place.

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

- The user and cluster to use is determined. At this point, you may or may not have a context; they are built based on the first match in the following flow, which is run once for the user and once for the cluster:
  - The value of the `--user` for user name and `--cluster` option for cluster name.
  - If the `--context` option is present, then use the context’s value.
  - An empty value is allowed at this stage.
The actual cluster information to use is determined. At this point, you may or may not have cluster information. Each piece of the cluster information is built based on the first match in the following flow:

- The values of any of the following command line options:
  - `--server`,
  - `--api-version`
  - `--certificate-authority`
  - `--insecure-skip-tls-verify`
  - If cluster information and a value for the attribute is present, then use it.
  - If you do not have a server location, then there is an error.

- The actual user information to use is determined. Users are built using the same rules as clusters, except that you can only have one authentication technique per user; conflicting techniques cause the operation to fail. Command line options take precedence over config file values. Valid command line options are:
  - `--auth-path`
  - `--client-certificate`
  - `--client-key`
  - `--token`
  - For any information that is still missing, default values are used and prompts are given for additional information.

2.4. EXTENDING THE OPENSIFT CLD WITH PLUG-INS

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

2.4.1. Writing CLI plug-ins

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

Procedure

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

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

2. Add the following contents to the file.

```bash
#!/bin/bash

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

# optional argument handling
if [[ "$1" == "config" ]]
then
    echo $KUBECONFIG
    exit 0
fi

echo "I am a plugin named kubectl-foo"
```

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

Additional resources

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

2.4.2. Installing and using CLI plug-ins

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

Prerequisites

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

Procedure

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

   ```
   $ chmod +x <plugin_file>
   ```

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

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

3. Run `oc plugin list` to make sure that the plug-in is listed.
4. Invoke the new command or option introduced by the plug-in.

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

```
$ oc ns
```

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

### 2.5. OPENSHIFT CLI DEVELOPER COMMAND REFERENCE

This reference provides descriptions and example commands for OpenShift CLI (oc) developer commands. For administrator commands, see the OpenShift CLI administrator command reference.

Run `oc help` to list all commands or run `oc <command> --help` to get additional details for a specific command.

#### 2.5.1. OpenShift CLI (oc) developer commands

#### 2.5.1.1. oc annotate

Update the annotations on a resource

**Example usage**

```
# Update pod 'foo' with the annotation 'description' and the value 'my frontend'
# If the same annotation is set multiple times, only the last value will be applied
oc annotate pods foo description=my frontend

# Update a pod identified by type and name in "pod.json"
oc annotate -f pod.json description='my frontend'

# Update pod 'foo' with the annotation 'description' and the value 'my frontend running nginx',
# overwriting any existing value
oc annotate --overwrite pods foo description='my frontend running nginx'

# Update all pods in the namespace
oc annotate pods --all description='my frontend running nginx'

# Update pod 'foo' only if the resource is unchanged from version 1
oc annotate pods foo description='my frontend running nginx' --resource-version=1
```
2.5.1.2. oc api-resources

Print the supported API resources on the server

Example usage

```
# Print the supported API resources
oc api-resources

# Print the supported API resources with more information
oc api-resources -o wide

# Print the supported API resources sorted by a column
oc api-resources --sort-by=name

# Print the supported namespaced resources
oc api-resources --namespaced=true

# Print the supported non-namespaced resources
oc api-resources --namespaced=false

# Print the supported API resources with a specific APIGroup
oc api-resources --api-group=extensions
```

2.5.1.3. oc api-versions

Print the supported API versions on the server, in the form of "group/version"

Example usage

```
# Print the supported API versions
oc api-versions
```

2.5.1.4. oc apply

Apply a configuration to a resource by file name or stdin

Example usage

```
# Apply the configuration in pod.json to a pod
oc apply -f ./pod.json

# Apply resources from a directory containing kustomization.yaml - e.g. dir/kustomization.yaml
oc apply -k dir/

# Apply the JSON passed into stdin to a pod
cat pod.json | oc apply -f -

# Note: --prune is still in Alpha
```
2.5.1.5. oc apply edit-last-applied

Edit latest last-applied-configuration annotations of a resource/object

Example usage

```
# Edit the last-applied-configuration annotations by type/name in YAML
oc apply edit-last-applied deployment/nginx

# Edit the last-applied-configuration annotations by file in JSON
oc apply edit-last-applied -f deploy.yaml -o json
```

2.5.1.6. oc apply set-last-applied

Set the last-applied-configuration annotation on a live object to match the contents of a file

Example usage

```
# Set the last-applied-configuration of a resource to match the contents of a file
oc apply set-last-applied -f deploy.yaml

# Execute set-last-applied against each configuration file in a directory
oc apply set-last-applied -f path/

# Set the last-applied-configuration of a resource to match the contents of a file; will create the annotation if it does not already exist
oc apply set-last-applied -f deploy.yaml --create-annotation=true
```

2.5.1.7. oc apply view-last-applied

View the latest last-applied-configuration annotations of a resource/object

Example usage

```
# View the last-applied-configuration annotations by type/name in YAML
oc apply view-last-applied deployment/nginx

# View the last-applied-configuration annotations by file in JSON
oc apply view-last-applied -f deploy.yaml -o json
```

2.5.1.8. oc attach

Attach to a running container

Example usage

```
```

# Apply the configuration in manifest.yaml that matches label app=nginx and delete all other resources that are not in the file and match label app=nginx
oc apply --prune -f manifest.yaml -l app=nginx

# Apply the configuration in manifest.yaml and delete all the other config maps that are not in the file
oc apply --prune -f manifest.yaml --all --prune-whitelist=core/v1/ConfigMap

# Apply the configuration in manifest.yaml that matches label app=nginx and delete all other resources that are not in the file and match label app=nginx
oc apply --prune -f manifest.yaml -l app=nginx

# Apply the configuration in manifest.yaml and delete all the other config maps that are not in the file
oc apply --prune -f manifest.yaml --all --prune-whitelist=core/v1/ConfigMap
```
2.5.1.9. **oc auth can-i**

Check whether an action is allowed

**Example usage**

```bash
# Check to see if I can create pods in any namespace
oc auth can-i create pods --all-namespaces

# Check to see if I can list deployments in my current namespace
oc auth can-i list deployments.apps

# Check to see if I can do everything in my current namespace ("***" means all)
oc auth can-i ***

# Check to see if I can get the job named "bar" in namespace "foo"
oc auth can-i list jobs.batch/bar -n foo

# Check to see if I can read pod logs
oc auth can-i get pods --subresource=log

# Check to see if I can access the URL /logs/
oc auth can-i get /logs/

# List all allowed actions in namespace "foo"
oc auth can-i --list --namespace=foo
```

2.5.1.10. **oc auth reconcile**

Reconciles rules for RBAC role, role binding, cluster role, and cluster role binding objects

**Example usage**

```bash
# Reconcile RBAC resources from a file
oc auth reconcile -f my-rbac-rules.yaml
```

2.5.1.11. **oc autoscale**

Autoscale a deployment config, deployment, replica set, stateful set, or replication controller
2.5.1.12. oc cancel-build

Cancel running, pending, or new builds

Example usage

# Cancel the build with the given name
oc cancel-build ruby-build-2

# Cancel the named build and print the build logs
oc cancel-build ruby-build-2 --dump-logs

# Cancel the named build and create a new one with the same parameters
oc cancel-build ruby-build-2 --restart

# Cancel multiple builds
oc cancel-build ruby-build-1 ruby-build-2 ruby-build-3

# Cancel all builds created from the 'ruby-build' build config that are in the 'new' state
oc cancel-build bc/ruby-build --state=new

2.5.1.13. oc cluster-info

Display cluster information

Example usage

# Print the address of the control plane and cluster services
oc cluster-info

2.5.1.14. oc cluster-info dump

Dump relevant information for debugging and diagnosis

Example usage

# Dump current cluster state to stdout
oc cluster-info dump

# Dump current cluster state to /path/to/cluster-state
oc cluster-info dump --output-directory=/path/to/cluster-state

# Dump all namespaces to stdout
2.5.1.15. oc completion

Output shell completion code for the specified shell (bash, zsh or fish)

Example usage

# Installing bash completion on macOS using homebrew
## If running Bash 3.2 included with macOS
brew install bash-completion
## or, if running Bash 4.1+
brew install bash-completion@2
## If oc is installed via homebrew, this should start working immediately
## If you’ve installed via other means, you may need add the completion to your completion directory
oc completion bash > $(brew --prefix)/etc/bash_completion.d/oc

# Installing bash completion on Linux
## If bash-completion is not installed on Linux, install the 'bash-completion' package
## via your distribution’s package manager.
## Load the oc completion code for bash into the current shell
source <(oc completion bash)
## Write bash completion code to a file and source it from .bash_profile
oc completion bash > ~/.kube/completion.bash.inc
print "
# Kubectl shell completion
source "$HOME/.kube/completion.bash.inc"
" >> $HOME/.bash_profile
source $HOME/.bash_profile

# Load the oc completion code for zsh[1] into the current shell
source <(oc completion zsh)
# Set the oc completion code for zsh[1] to autoload on startup
oc completion zsh > "${fpath[1]}/_oc"

# Load the oc completion code for fish[2] into the current shell
oc completion fish | source
# To load completions for each session, execute once:
oc completion fish > ~/.config/fish/completions/oc.fish

# Load the oc completion code for powershell into the current shell
oc completion powershell | Out-String | Invoke-Expression
# Set oc completion code for powershell to run on startup
## Save completion code to a script and execute in the profile
oc completion powershell > $HOME\.kube\completion.ps1
Add-Content $PROFILE "$HOME\.kube\completion.ps1"
## Execute completion code in the profile
Add-Content $PROFILE "if (Get-Command oc -ErrorAction SilentlyContinue) {
oc completion powershell | Out-String | Invoke-Expression
### 2.5.1.16. oc config current-context

Display the current-context

**Example usage**

```
# Display the current-context
oc config current-context
```

### 2.5.1.17. oc config delete-cluster

Delete the specified cluster from the kubeconfig

**Example usage**

```
# Delete the minikube cluster
oc config delete-cluster minikube
```

### 2.5.1.18. oc config delete-context

Delete the specified context from the kubeconfig

**Example usage**

```
# Delete the context for the minikube cluster
oc config delete-context minikube
```

### 2.5.1.19. oc config delete-user

Delete the specified user from the kubeconfig

**Example usage**

```
# Delete the minikube user
oc config delete-user minikube
```

### 2.5.1.20. oc config get-clusters

Display clusters defined in the kubeconfig

**Example usage**

```
# List the clusters that oc knows about
oc config get-clusters
```

### 2.5.1.21. oc config get-contexts

## Add completion code directly to the $PROFILE script

```bash
oc completion powershell >> $PROFILE
```

## Add completion code directly to the $PROFILE script
Describe one or many contexts

Example usage

```
# List all the contexts in your kubeconfig file
oc config get-contexts

# Describe one context in your kubeconfig file
oc config get-contexts my-context
```

2.5.1.22. oc config get-users

Display users defined in the kubeconfig

Example usage

```
# List the users that oc knows about
oc config get-users
```

2.5.1.23. oc config rename-context

Rename a context from the kubeconfig file

Example usage

```
# Rename the context 'old-name' to 'new-name' in your kubeconfig file
oc config rename-context old-name new-name
```

2.5.1.24. oc config set

Set an individual value in a kubeconfig file

Example usage

```
# Set the server field on the my-cluster cluster to https://1.2.3.4
oc config set clusters.my-cluster.server https://1.2.3.4

# Set the certificate-authority-data field on the my-cluster cluster
oc config set clusters.my-cluster.certificate-authority-data $(echo "cert_data_here" | base64 -i -)

# Set the cluster field in the my-context context to my-cluster
oc config set contexts.my-context.cluster my-cluster

# Set the client-key-data field in the cluster-admin user using --set-raw-bytes option
oc config set users.cluster-admin.client-key-data cert_data_here --set-raw-bytes=true
```

2.5.1.25. oc config set-cluster

Set a cluster entry in kubeconfig

Example usage
2.5.1.26. oc config set-context

Set a context entry in kubeconfig

Example usage

# Set only the server field on the e2e cluster entry without touching other values
oc config set-cluster e2e --server=https://1.2.3.4

# Embed certificate authority data for the e2e cluster entry
oc config set-cluster e2e --embed-ca-cert --certificate-authority=~/.kube/e2e/kubernetes.ca.crt

# Disable cert checking for the dev cluster entry
oc config set-cluster e2e --insecure-skip-tls-verify=true

# Set custom TLS server name to use for validation for the e2e cluster entry
oc config set-cluster e2e --tls-server-name=my-cluster-name

2.5.1.27. oc config set-credentials

Set a user entry in kubeconfig

Example usage

# Set only the "client-key" field on the "cluster-admin" entry, without touching other values
oc config set-credentials cluster-admin --client-key=~/.kube/admin.key

# Set basic auth for the "cluster-admin" entry
oc config set-credentials cluster-admin --username=admin --password=uXFGweU9135qCif

# Embed client certificate data in the "cluster-admin" entry
oc config set-credentials cluster-admin --client-certificate=~/.kube/admin.crt --embed-ca-cert=true

# Enable the Google Compute Platform auth provider for the "cluster-admin" entry
oc config set-credentials cluster-admin --auth-provider=gcp

# Enable the OpenID Connect auth provider for the "cluster-admin" entry with additional args
oc config set-credentials cluster-admin --auth-provider=oidc --auth-provider-arg=client-id=foo --auth-provider-arg=client-secret=bar

# Remove the "client-secret" config value for the OpenID Connect auth provider for the "cluster-admin" entry
oc config set-credentials cluster-admin --auth-provider=oidc --auth-provider-arg=client-secret=

# Enable new exec auth plugin for the "cluster-admin" entry
oc config set-credentials cluster-admin --exec-command=/path/to/the/executable --exec-api-version=client.authentication.k8s.io/v1beta1

# Define new exec auth plugin args for the "cluster-admin" entry
oc config set-credentials cluster-admin --exec-arg=arg1 --exec-arg=arg2
# Create or update exec auth plugin environment variables for the "cluster-admin" entry
oc config set-credentials cluster-admin --exec-env=key1=val1 --exec-env=key2=val2

# Remove exec auth plugin environment variables for the "cluster-admin" entry
oc config set-credentials cluster-admin --exec-env=var-to-remove-

## 2.5.1.28. oc config unset
Unset an individual value in a kubeconfig file

**Example usage**

```
# Unset the current-context
oc config unset current-context

# Unset namespace in foo context
oc config unset contexts.foo.namespace
```

## 2.5.1.29. oc config use-context
Set the current-context in a kubeconfig file

**Example usage**

```
# Use the context for the minikube cluster
oc config use-context minikube
```

## 2.5.1.30. oc config view
Display merged kubeconfig settings or a specified kubeconfig file

**Example usage**

```
# Show merged kubeconfig settings
oc config view

# Show merged kubeconfig settings and raw certificate data
oc config view --raw

# Get the password for the e2e user
oc config view -o jsonpath='{.users[?(@.name == "e2e")].user.password}'
```

## 2.5.1.31. oc cp
Copy files and directories to and from containers

**Example usage**

```
# !!!Important Note!!!
# Requires that the 'tar' binary is present in your container image. If 'tar' is not present, 'oc cp' will fail.
```
2.5.1.32. **oc create**

Create a resource from a file or from stdin

**Example usage**

```bash
# For advanced use cases, such as symlinks, wildcard expansion or
# file mode preservation, consider using 'oc exec'.

# Copy /tmp/foo local file to /tmp/bar in a remote pod in namespace <some-namespace>
 tar cf - /tmp/foo | oc exec -i -n <some-namespace> <some-pod> -- tar x -C /tmp/bar

# Copy /tmp/foo from a remote pod to /tmp/bar locally
oc exec -n <some-namespace> <some-pod> -- tar x -C /tmp/bar

# Copy /tmp/foo_dir local directory to /tmp/bar_dir in a remote pod in the default namespace
oc cp /tmp/foo_dir <some-pod>:/tmp/bar_dir

# Copy /tmp/foo local file to /tmp/bar in a remote pod in a specific container
oc cp /tmp/foo <some-namespace>/<some-pod>:/tmp/bar -c <specific-container>

# Copy /tmp/foo local file to /tmp/bar in a remote pod in namespace <some-namespace>
oc cp /tmp/foo <some-namespace>/<some-pod>:/tmp/bar

# Copy /tmp/foo from a remote pod to /tmp/bar locally
oc cp <some-namespace>/<some-pod>:/tmp/foo /tmp/bar
```

2.5.1.33. **oc create build**

Create a new build

**Example usage**

```bash
# Create a new build
oc create build myapp
```

2.5.1.34. **oc create clusterresourcequota**

Create a cluster resource quota

**Example usage**
2.5.1.35. oc create clusterrole

Create a cluster role

Example usage

```bash
# Create a cluster role named "pod-reader" that allows user to perform "get", "watch" and "list" on pods
oc create clusterrole pod-reader --verb=get,list,watch --resource=pods

# Create a cluster role named "pod-reader" with ResourceName specified
oc create clusterrole pod-reader --verb=get --resource=pods --resource-name=readablepod --resource-name=anotherpod

# Create a cluster role named "foo" with API Group specified
oc create clusterrole foo --verb=get,list,watch --resource=rs.extensions

# Create a cluster role named "foo" with SubResource specified
oc create clusterrole foo --verb=get,list,watch --resource=pods,pods/status

# Create a cluster role name "foo" with NonResourceURL specified
oc create clusterrole "foo" --verb=get --non-resource-url=/logs/*

# Create a cluster role name "monitoring" with AggregationRule specified
oc create clusterrole monitoring --aggregation-rule="rbac.example.com/aggregate-to-monitoring=true"
```

2.5.1.36. oc create clusterrolebinding

Create a cluster role binding for a particular cluster role

Example usage

```bash
# Create a cluster role binding for user1, user2, and group1 using the cluster-admin cluster role
oc create clusterrolebinding cluster-admin --clusterrole=cluster-admin --user=user1 --user=user2 --group=group1
```

2.5.1.37. oc create configmap

Create a config map from a local file, directory or literal value

Example usage

```bash
# Create a new config map named my-config based on folder bar
oc create configmap my-config --from-file=path/to/bar

# Create a new config map named my-config with specified keys instead of file basenames on disk
oc create configmap my-config --from-file=key1=/path/to/bar/file1.txt --from-file=key2=/path/to/bar/file2.txt
```
2.5.1.38. oc create cronjob

Create a cron job with the specified name

Example usage

```bash
# Create a cron job
oc create cronjob my-job --image=busybox --schedule="*/1 * * * *"

# Create a cron job with a command
oc create cronjob my-job --image=busybox --schedule="*/1 * * * *" -- date
```

2.5.1.39. oc create deployment

Create a deployment with the specified name

Example usage

```bash
# Create a deployment named my-dep that runs the busybox image
oc create deployment my-dep --image=busybox

# Create a deployment with a command
oc create deployment my-dep --image=busybox -- date

# Create a deployment named my-dep that runs the nginx image with 3 replicas
oc create deployment my-dep --image=nginx --replicas=3

# Create a deployment named my-dep that runs the busybox image and expose port 5701
oc create deployment my-dep --image=busybox --port=5701
```

2.5.1.40. oc create deploymentconfig

Create a deployment config with default options that uses a given image

Example usage

```bash
# Create an nginx deployment config named my-nginx
oc create deploymentconfig my-nginx --image=nginx

# Create a deploymentconfig my-nginx --image=nginx -- from env file
oc create deploymentconfig my-nginx --from-env-file=path/to/foo.env --from-env-file=path/to/bar.env
```

2.5.1.41. oc create identity

Manually create an identity (only needed if automatic creation is disabled)
Example usage

```
# Create an identity with identity provider "acme_ldap" and the identity provider username "adamjones"
oc create identity acme_ldap:adamjones
```

2.5.1.42. oc create imagestream

Create a new empty image stream

Example usage

```
# Create a new image stream
oc create imagestream mysql
```

2.5.1.43. oc create imagestreamtag

Create a new image stream tag

Example usage

```
# Create a new image stream tag based on an image in a remote registry
oc create imagestreamtag mysql:latest --from-image=myregistry.local/mysql/mysql:5.0
```

2.5.1.44. oc create ingress

Create an ingress with the specified name

Example usage

```
# Create a single ingress called ‘simple’ that directs requests to foo.com/bar to svc
# svc1:8080 with a tls secret “my-cert”
oc create ingress simple --rule="foo.com/bar=svc1:8080,tls=my-cert"

# Create a catch all ingress of “/path” pointing to service svc:port and Ingress Class as “otheringress”
oc create ingress catch-all --class=otheringress --rule="/path=svc:port"

# Create an ingress with two annotations: ingress.annotation1 and ingress.annotations2
oc create ingress annotated --class=default --rule="foo.com/bar=svc:port" \ 
--annotation ingress.annotation1=foo \ 
--annotation ingress.annotation2=bla

# Create an ingress with the same host and multiple paths
oc create ingress multipath --class=default \ 
--rule="foo.com/=svc:port" \ 
--rule="foo.com/admin/=svcadmin:portadmin"

# Create an ingress with multiple hosts and the pathType as Prefix
oc create ingress ingress1 --class=default \ 
--rule="foo.com/path=svc:8080" \ 
--rule="bar.com/admin=svc2:http"
```
2.5.1.45. oc create job

Create a job with the specified name

Example usage

```bash
# Create a job
oc create job my-job --image=busybox

# Create a job with a command
oc create job my-job --image=busybox -- date

# Create a job from a cron job named "a-cronjob"
oc create job test-job --from=cronjob/a-cronjob
```

2.5.1.46. oc create namespace

Create a namespace with the specified name

Example usage

```bash
# Create a new namespace named my-namespace
oc create namespace my-namespace
```

2.5.1.47. oc create poddisruptionbudget

Create a pod disruption budget with the specified name

Example usage

```bash
# Create a pod disruption budget named my-pdb that will select all pods with the app=rails label
# and require at least one of them being available at any point in time
oc create poddisruptionbudget my-pdb --selector=app=rails --min-available=1

# Create a pod disruption budget named my-pdb that will select all pods with the app=nginx label
# and require at least half of the pods selected to be available at any point in time
oc create pdb my-pdb --selector=app=nginx --min-available=50%
```

2.5.1.48. oc create priorityclass
Create a priority class with the specified name

Example usage

```
# Create a priority class named high-priority
oc create priorityclass high-priority --value=1000 --description="high priority"

# Create a priority class named default-priority that is considered as the global default priority
oc create priorityclass default-priority --value=1000 --global-default=true --description="default priority"

# Create a priority class named high-priority that cannot preempt pods with lower priority
oc create priorityclass high-priority --value=1000 --description="high priority" --preemption-policy="Never"
```

2.5.1.49. oc create quota

Create a quota with the specified name

Example usage

```
# Create a new resource quota named my-quota
oc create quota my-quota --
hard=cpu=1,memory=1G,pods=2,services=3,replicationcontrollers=2,resourcequotas=1,secrets=5,persistentvolumeclaims=10

# Create a new resource quota named best-effort
oc create quota best-effort --hard=pods=100 --scopes=BestEffort
```

2.5.1.50. oc create role

Create a role with single rule

Example usage

```
# Create a role named "pod-reader" that allows user to perform "get", "watch" and "list" on pods
oc create role pod-reader --verb=get --verb=list --verb=watch --resource=pods

# Create a role named "pod-reader" with ResourceName specified
oc create role pod-reader --verb=get --resource=pods --resource-name=readablepod --resource-name=anotherpod

# Create a role named "foo" with API Group specified
oc create role foo --verb=get,list,watch --resource=rs.extensions

# Create a role named "foo" with SubResource specified
oc create role foo --verb=get,list,watch --resource=pods,pods/status
```

2.5.1.51. oc create rolebinding

Create a role binding for a particular role or cluster role

Example usage
# Create a role binding for user1, user2, and group1 using the admin cluster role
oc create rolebinding admin --clusterrole=admin --user=user1 --user=user2 --group=group1

2.5.1.52. oc create route edge
Create a route that uses edge TLS termination

Example usage

# Create an edge route named "my-route" that exposes the frontend service
oc create route edge my-route --service=frontend

# Create an edge route that exposes the frontend service and specify a path
# If the route name is omitted, the service name will be used
oc create route edge --service=frontend --path /assets

2.5.1.53. oc create route passthrough
Create a route that uses passthrough TLS termination

Example usage

# Create a passthrough route named "my-route" that exposes the frontend service
oc create route passthrough my-route --service=frontend

# Create a passthrough route that exposes the frontend service and specify
# a host name. If the route name is omitted, the service name will be used
oc create route passthrough --service=frontend --hostname=www.example.com

2.5.1.54. oc create route reencrypt
Create a route that uses reencrypt TLS termination

Example usage

# Create a route named "my-route" that exposes the frontend service
oc create route reencrypt my-route --service=frontend --dest-ca-cert cert.cert

# Create a reencrypt route that exposes the frontend service, letting the
# route name default to the service name and the destination CA certificate
# default to the service CA
oc create route reencrypt --service=frontend

2.5.1.55. oc create secret docker-registry
Create a secret for use with a Docker registry

Example usage

# If you don’t already have a .dockercfg file, you can create a dockercfg secret directly by using:
oc create secret docker-registry my-secret --docker-server=DOCKER_REGISTRY_SERVER --
docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-
email=DOCKER_EMAIL

# Create a new secret named my-secret from ~/.docker/config.json
oc create secret docker-registry my-secret --from-file=.dockerconfigjson=path/to/.docker/config.json

2.5.1.56. oc create secret generic
Create a secret from a local file, directory, or literal value

Example usage

# Create a new secret named my-secret with keys for each file in folder bar
oc create secret generic my-secret --from-file=path/to/bar

# Create a new secret named my-secret with specified keys instead of names on disk
oc create secret generic my-secret --from-file=ssh-privatekey=path/to/id_rsa --from-file=ssh-publickey=path/to/id_rsa.pub

# Create a new secret named my-secret with key1=supersecret and key2=topsecret
oc create secret generic my-secret --from-literal=key1=supersecret --from-literal=key2=topsecret

# Create a new secret named my-secret using a combination of a file and a literal
oc create secret generic my-secret --from-file=ssh-privatekey=path/to/id_rsa --from-literal=passphrase=topsecret

# Create a new secret named my-secret from env files
oc create secret generic my-secret --from-env-file=path/to/foo.env --from-env-file=path/to/bar.env

2.5.1.57. oc create secret tls
Create a TLS secret

Example usage

# Create a new TLS secret named tls-secret with the given key pair
oc create secret tls tls-secret --cert=path/to/tls.cert --key=path/to/tls.key

2.5.1.58. oc create service clusterip
Create a ClusterIP service

Example usage

# Create a new ClusterIP service named my-cs
oc create service clusterip my-cs --tcp=5678:8080

# Create a new ClusterIP service named my-cs (in headless mode)
oc create service clusterip my-cs --clusterip="None"

2.5.1.59. oc create service externalname
Create an ExternalName service
Example usage

```
# Create a new ExternalName service named my-ns
oc create service externalname my-ns --external-name bar.com
```

2.5.1.60. oc create service loadbalancer

Create a LoadBalancer service

Example usage

```
# Create a new LoadBalancer service named my-lbs
oc create service loadbalancer my-lbs --tcp=5678:8080
```

2.5.1.61. oc create service nodeport

Create a NodePort service

Example usage

```
# Create a new NodePort service named my-ns
oc create service nodeport my-ns --tcp=5678:8080
```

2.5.1.62. oc create serviceaccount

Create a service account with the specified name

Example usage

```
# Create a new service account named my-service-account
oc create serviceaccount my-service-account
```

2.5.1.63. oc create user

Manually create a user (only needed if automatic creation is disabled)

Example usage

```
# Create a user with the username “ajones” and the display name “Adam Jones”
oc create user ajones --full-name="Adam Jones"
```

2.5.1.64. oc create useridentitymapping

Manually map an identity to a user

Example usage

```
# Map the identity “acme_ldap:adamjones” to the user “ajones”
oc create useridentitymapping acme_ldap:adamjones ajones
```
2.5.1.65. oc debug

Launch a new instance of a pod for debugging

Example usage

```bash
# Start a shell session into a pod using the OpenShift tools image
oc debug

# Debug a currently running deployment by creating a new pod
oc debug deploy/test

# Debug a node as an administrator
oc debug node/master-1

# Launch a shell in a pod using the provided image stream tag
oc debug istag/mysql:latest -n openshift

# Test running a job as a non-root user
oc debug job/test --as-user=1000000

# Debug a specific failing container by running the env command in the 'second' container
oc debug daemonset/test -c second -- /bin/env

# See the pod that would be created to debug
oc debug mypod-9xbc -o yaml

# Debug a resource but launch the debug pod in another namespace
# Note: Not all resources can be debugged using --to-namespace without modification. For example,
# volumes and service accounts are namespace-dependent. Add `-o yaml` to output the debug pod
definition
# to disk. If necessary, edit the definition then run `oc debug -f -` or run without --to-namespace
oc debug mypod-9xbc --to-namespace testns
```

2.5.1.66. oc delete

Delete resources by file names, stdin, resources and names, or by resources and label selector

Example usage

```bash
# Delete a pod using the type and name specified in pod.json
oc delete -f ./pod.json

# Delete resources from a directory containing kustomization.yaml - e.g. dir/kustomization.yaml
oc delete -k dir

# Delete a pod based on the type and name in the JSON passed into stdin
cat pod.json | oc delete -f -

# Delete pods and services with same names "baz" and "foo"
oc delete pod,service baz foo

# Delete pods and services with label name=myLabel
oc delete pods,services -l name=myLabel
```
2.5.1.67. oc describe

Show details of a specific resource or group of resources

Example usage

# Describe a node
oc describe nodes kubernetes-node-emt8.c.myproject.internal

# Describe a pod
oc describe pods/nginx

# Describe a pod identified by type and name in "pod.json"
oc describe -f pod.json

# Describe all pods
oc describe pods

# Describe pods by label name=myLabel
oc describe po -l name=myLabel

# Describe all pods managed by the ‘frontend’ replication controller
# (rc-created pods get the name of the rc as a prefix in the pod name)
oc describe pods frontend

2.5.1.68. oc diff

Diff the live version against a would-be applied version

Example usage

# Diff resources included in pod.json
oc diff -f pod.json

# Diff file read from stdin
cat service.yaml | oc diff -f -

2.5.1.69. oc edit

Edit a resource on the server

Example usage
2.5.1.70. **oc exec**

Execute a command in a container

**Example usage**

```powershell
# Get output from running the ‘date’ command from pod mypod, using the first container by default
oc exec mypod -- date

# Get output from running the ‘date’ command in ruby-container from pod mypod
oc exec mypod -c ruby-container -- date

# Switch to raw terminal mode; sends stdin to ‘bash’ in ruby-container from pod mypod
# and sends stdout/stderr from ‘bash’ back to the client
oc exec mypod -c ruby-container -i -t -- bash -il

# List contents of /usr from the first container of pod mypod and sort by modification time
# If the command you want to execute in the pod has any flags in common (e.g. -i),
# you must use two dashes (--) to separate your command's flags/arguments
# Also note, do not surround your command and its flags/arguments with quotes
# unless that is how you would execute it normally (i.e., do ls -t /usr, not "ls -t /usr")
oc exec mypod -i -t -- ls -t /usr

# Get output from running 'date' command from the first pod of the deployment mydeployment, using the first container by default
oc exec deploy/mydeployment -- date

# Get output from running 'date' command from the first pod of the service myservice, using the first container by default
oc exec svc/myservice -- date
```

2.5.1.71. **oc explain**

Get documentation for a resource

**Example usage**

```powershell
# Get the documentation of the resource and its fields
oc explain pods

# Get the documentation of a specific field of a resource
oc explain pods.spec.containers
```
2.5.1.72. oc expose

Expose a replicated application as a service or route

Example usage

```
# Create a route based on service nginx. The new route will reuse nginx's labels
oc expose service nginx

# Create a route and specify your own label and route name
oc expose service nginx -l name=myroute --name=fromdowntown

# Create a route and specify a host name
oc expose service nginx --hostname=www.example.com

# Create a route with a wildcard
oc expose service nginx --hostname=x.example.com --wildcard-policy=Subdomain
# This would be equivalent to *.example.com. NOTE: only hosts are matched by the wildcard;
# subdomains would not be included

# Expose a deployment configuration as a service and use the specified port
oc expose dc ruby-hello-world --port=8080

# Expose a service as a route in the specified path
oc expose service nginx --path=/nginx

# Expose a service using different generators
oc expose service nginx --name=exposed-svc --port=12201 --protocol="TCP" --
generator="service/v2"
oc expose service nginx --name=my-route --port=12201 --generator="route/v1"

# Exposing a service using the "route/v1" generator (default) will create a new exposed route with
# the "--name" provided
# (or the name of the service otherwise). You may not specify a "--protocol" or "--target-port" option
# when using this generator
```

2.5.1.73. oc extract

Extract secrets or config maps to disk

Example usage

```
# Extract the secret "test" to the current directory
oc extract secret/test

# Extract the config map "nginx" to the /tmp directory
oc extract configmap/nginx --to=/tmp

# Extract the config map "nginx" to STDOUT
oc extract configmap/nginx --to=-

# Extract only the key "nginx.conf" from config map "nginx" to the /tmp directory
oc extract configmap/nginx --to=/tmp --keys=nginx.conf
```
2.5.1.74. oc get

Display one or many resources

Example usage

```
# List all pods in ps output format
oc get pods

# List all pods in ps output format with more information (such as node name)
oc get pods -o wide

# List a single replication controller with specified NAME in ps output format
oc get replicationcontroller web

# List deployments in JSON output format, in the "v1" version of the "apps" API group
oc get deployments.v1.apps -o json

# List a single pod in JSON output format
oc get -o json pod web-pod-13je7

# List a pod identified by type and name specified in "pod.yaml" in JSON output format
oc get -f pod.yaml -o json

# List resources from a directory with kustomization.yaml - e.g. dir/kustomization.yaml
oc get -k dir/

# Return only the phase value of the specified pod
oc get -o template pod/web-pod-13je7 --template={{.status.phase}}

# List resource information in custom columns
oc get pod test-pod -o custom-columns=CONTAINER:.spec.containers[0].name,IMAGE:.spec.containers[0].image

# List all replication controllers and services together in ps output format
oc get rc,services

# List one or more resources by their type and names
oc get rc/web service/frontend pods/web-pod-13je7
```

2.5.1.75. oc idle

Idle scalable resources

Example usage

```
# Idle the scalable controllers associated with the services listed in to-idle.txt
$ oc idle --resource-names-file to-idle.txt
```

2.5.1.76. oc image append

Add layers to images and push them to a registry

Example usage
2.5.1.77. oc image extract

Copy files from an image to the file system

Example usage

# Extract the busybox image into the current directory
oc image extract docker.io/library/busybox:latest

# Extract the busybox image into a designated directory (must exist)
oc image extract docker.io/library/busybox:latest --path /tmp/busybox

# Extract the busybox image into the current directory for linux/s390x platform
# Note: Wildcard filter is not supported with extract. Pass a single os/arch to extract
oc image extract docker.io/library/busybox:latest --filter-by-os=linux/s390x

# Extract a single file from the image into the current directory
oc image extract docker.io/library/centos:7 --path /bin/bash:

# Extract all .repo files from the image’s /etc/yum.repos.d/ folder into the current directory
oc image extract docker.io/library/centos:7 --path /etc/yum.repos.d/* .repo:

# Extract all .repo files from the image’s /etc/yum.repos.d/ folder into a designated directory (must exist)
# This results in /tmp/yum.repos.d/* .repo on local system
oc image extract docker.io/library/centos:7 --path /etc/yum.repos.d/* .repo:/tmp/yum.repos.d

# Extract an image stored on disk into the current directory ($pwd)/v2/busybox/blobs,manifests
exists)
# --confirm is required because the current directory is not empty
oc image extract file://busybox:local --confirm

# Extract an image stored on disk in a directory other than $(pwd)/v2 into the current directory
# --confirm is required because the current directory is not empty ($(pwd)/busybox-mirror-dir/v2/busybox exists)
oc image extract file://busybox:local --dir busybox-mirror-dir --confirm

# Extract an image stored on disk in a directory other than $(pwd)/v2 into a designated directory
# (must exist)
oc image extract file://busybox:local --dir busybox-mirror-dir --path /:/tmp/busybox

# Extract the last layer in the image
oc image extract docker.io/library/centos:7[-1]

# Extract the first three layers of the image
oc image extract docker.io/library/centos:7:[3]

# Extract the last three layers of the image
oc image extract docker.io/library/centos:7[-3:]

2.5.1.78. oc image info

Display information about an image

Example usage

# Show information about an image
oc image info quay.io/openshift/cli:latest

# Show information about images matching a wildcard
oc image info quay.io/openshift/cli:4.*

# Show information about a file mirrored to disk under DIR
oc image info --dir=DIR file://library/busybox:latest

# Select which image from a multi-OS image to show
oc image info library/busybox:latest --filter-by-os=linux/arm64

2.5.1.79. oc image mirror

Mirror images from one repository to another

Example usage

# Copy image to another tag
oc image mirror myregistry.com/myimage:latest myregistry.com/myimage:stable

# Copy image to another registry
oc image mirror myregistry.com/myimage:latest docker.io/myrepository/myimage:stable

# Copy all tags starting with mysql to the destination repository
oc image mirror myregistry.com/myimage:mysql* docker.io/myrepository/myimage
2.5.1.80. `oc import-image`

Import images from a container image registry

**Example usage**

```bash
# Copy image to disk, creating a directory structure that can be served as a registry
oc image mirror myregistry.com/myimage:latest file://myrepository/myimage:latest

# Copy image to S3 (pull from `<bucket>.s3.amazonaws.com/image:latest`)  
oc image mirror myregistry.com/myimage:latest  
s3://s3.amazonaws.com/<region>/<bucket>/image:latest

# Copy image to S3 without setting a tag (pull via `@<digest>`)  
oc image mirror myregistry.com/myimage:latest  
s3://s3.amazonaws.com/<region>/<bucket>/image

# Copy image to multiple locations  
oc image mirror myregistry.com/myimage:latest docker.io/myrepository/myimage:stable \  
docker.io/myrepository/myimage:dev

# Copy multiple images  
oc image mirror myregistry.com/myimage:latest=myregistry.com/other:test \  
myregistry.com/myimage:new=myregistry.com/other:target

# Copy manifest list of a multi-architecture image, even if only a single image is found  
oc image mirror myregistry.com/myimage:latest=myregistry.com/other:test \  
--keep-manifest-list=true

# Copy specific os/arch manifest of a multi-architecture image  
# Run `oc image info myregistry.com/myimage:latest` to see available os/arch for multi-arch images  
# Note that with multi-arch images, this results in a new manifest list digest that includes only  
# the filtered manifests  
oc image mirror myregistry.com/myimage:latest=myregistry.com/other:test \  
--filter-by-os=os/arch

# Copy all os/arch manifests of a multi-architecture image  
# Run `oc image info myregistry.com/myimage:latest` to see list of os/arch manifests that will be mirrored  
oc image mirror myregistry.com/myimage:latest=myregistry.com/other:test \  
--keep-manifest-list=true

# Note the above command is equivalent to  
oc image mirror myregistry.com/myimage:latest=myregistry.com/other:test \  
--filter-by-os=.*
```

# Import tag latest into a new image stream  
oc import-image mystream --from=registry.io/repo/image:latest --confirm

# Update imported data for tag latest in an already existing image stream  
oc import-image mystream

# Update imported data for tag stable in an already existing image stream  
oc import-image mystream:stable
2.5.1.81. oc kustomize

Build a kustomization target from a directory or URL.

Example usage

```
# Build the current working directory
oc kustomize

# Build some shared configuration directory
oc kustomize /home/config/production

# Build from github
oc kustomize https://github.com/kubernetes-sigs/kustomize.git/examples/helloWorld?ref=v1.0.6
```

2.5.1.82. oc label

Update the labels on a resource

Example usage

```
# Update pod 'foo' with the label 'unhealthy' and the value 'true'
oc label pods foo unhealthy=true

# Update pod 'foo' with the label 'status' and the value 'unhealthy', overwriting any existing value
oc label --overwrite pods foo status=unhealthy

# Update all pods in the namespace
oc label pods --all status=unhealthy

# Update a pod identified by the type and name in "pod.json"
oc label -f pod.json status=unhealthy

# Update pod 'foo' only if the resource is unchanged from version 1
oc label pods foo status=unhealthy --resource-version=1

# Update pod 'foo' by removing a label named 'bar' if it exists
# Does not require the --overwrite flag
oc label pods foo bar-
```

2.5.1.83. oc login

Log in to a server

Example usage

```
# Update imported data for all tags in an existing image stream
oc import-image mystream --all

# Import all tags into a new image stream
oc import-image mystream --from=registry.io/repo/image --all --confirm

# Import all tags into a new image stream using a custom timeout
oc --request-timeout=5m import-image mystream --from=registry.io/repo/image --all --confirm
```
# Log in interactively
oc login --username=myuser

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

# Log in to the given server with the given credentials (will not prompt interactively)
oc login localhost:8443 --username=myuser --password=mypass

2.5.1.84. oc logout

End the current server session

Example usage

# Log out
oc logout

2.5.1.85. oc logs

Print the logs for a container in a pod

Example usage

# Start streaming the logs of the most recent build of the openldap build config
oc logs -f bc/openldap

# Start streaming the logs of the latest deployment of the mysql deployment config
oc logs -f dc/mysql

# Get the logs of the first deployment for the mysql deployment config. Note that logs
# from older deployments may not exist either because the deployment was successful
# or due to deployment pruning or manual deletion of the deployment
oc logs --version=1 dc/mysql

# Return a snapshot of ruby-container logs from pod backend
oc logs backend -c ruby-container

# Start streaming of ruby-container logs from pod backend
oc logs -f pod/backend -c ruby-container

2.5.1.86. oc new-app

Create a new application

Example usage

# List all local templates and image streams that can be used to create an app
oc new-app --list

# Create an application based on the source code in the current git repository (with a public remote)
and a container image
oc new-app . --image=registry/repo/langimage
Create an application myapp with Docker based build strategy expecting binary input
oc new-app --strategy=docker --binary --name=myapp

Create a Ruby application based on the provided [image]~[source code] combination
oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git

Use the public container registry MySQL image to create an app. Generated artifacts will be labeled with db=mysql
oc new-app mysql MYSQL_USER=user MYSQL_PASSWORD=pass MYSQL_DATABASE=testdb -l db=mysql

Use a MySQL image in a private registry to create an app and override application artifacts’ names
oc new-app --image=myregistry.com/mycompany/mysql --name=private

Create an application from a remote repository using its beta4 branch
oc new-app https://github.com/openshift/ruby-hello-world#beta4

Create an application based on a stored template, explicitly setting a parameter value
oc new-app --template=ruby-helloworld-sample --param=MYSQL_USER=admin

Create an application from a remote repository and specify a context directory
oc new-app https://github.com/youruser/yourgitrepo --context-dir=src/build

Create an application from a remote private repository and specify which existing secret to use
oc new-app https://github.com/youruser/yourgitrepo --source-secret=yoursecret

Create an application based on a template file, explicitly setting a parameter value
oc new-app --file=./example/myapp/template.json --param=MYSQL_USER=admin

Search all templates, image streams, and container images for the ones that match "ruby"
oc new-app --search ruby

Search for "ruby", but only in stored templates (--template, --image-stream and --image
# can be used to filter search results)
oc new-app --search --template=ruby

Search for "ruby" in stored templates and print the output as YAML
oc new-app --search --template=ruby --output=yaml

2.5.1.87. oc new-build

Create a new build configuration

Example usage

# Create a build config based on the source code in the current git repository (with a public
# remote) and a container image
oc new-build --image=repo/langimage

# Create a NodeJS build config based on the provided [image]~[source code] combination
oc new-build centos/nodejs-8-centos7~https://github.com/sclorg/nodejs-ex.git

# Create a build config from a remote repository using its beta2 branch
2.5.1.88. oc new-project

Request a new project

Example usage

# Create a new project with minimal information
oc new-project web-team-dev

# Create a new project with a display name and description
oc new-project web-team-dev --display-name="Web Team Development" --description="Development project for the web team."

2.5.1.89. oc observe

Observe changes to resources and react to them (experimental)

Example usage

# Observe changes to services
oc observe services

# Observe changes to services, including the clusterIP and invoke a script for each
oc observe services --template '{ .spec.clusterIP }' -- register_dns.sh

# Observe changes to services filtered by a label selector
oc observe namespaces -l regist-dns=true --template '{ .spec.clusterIP }' -- register_dns.sh

2.5.1.90. oc patch

Update fields of a resource

Example usage

oc new-build https://github.com/openshift/ruby-hello-world#beta2

# Create a build config using a Dockerfile specified as an argument
oc new-build -D $'FROM centos:7
RUN yum install -y httpd'

# Create a build config from a remote repository and add custom environment variables
oc new-build https://github.com/openshift/ruby-hello-world -e RACK_ENV=development

# Create a build config from a remote private repository and specify which existing secret to use
oc new-build https://github.com/youruser/yourgitrepo --source-secret=yoursecret

# Create a build config from a remote repository and inject the npmrc into a build
oc new-build https://github.com/openshift/ruby-hello-world --build-secret npmrc::npmrc

# Create a build config from a remote repository and inject environment data into a build

# Create a build config that gets its input from a remote repository and another container image

# Create a new project with minimal information
oc new-project web-team-dev

# Create a new project with a display name and description
oc new-project web-team-dev --display-name="Web Team Development" --description="Development project for the web team."
2.5.1.91. oc policy add-role-to-user

Add a role to users or service accounts for the current project

Example usage

```
# Add the 'view' role to user1 for the current project
oc policy add-role-to-user view user1

# Add the 'edit' role to serviceaccount1 for the current project
oc policy add-role-to-user edit -z serviceaccount1
```

2.5.1.92. oc policy scc-review

Check which service account can create a pod

Example usage

```
# Check whether service accounts sa1 and sa2 can admit a pod with a template pod spec specified in my_resource.yaml
# Service Account specified in myresource.yaml file is ignored
oc policy scc-review -z sa1,sa2 -f my_resource.yaml

# Check whether service accounts system:serviceaccount:bob:default can admit a pod with a template pod spec specified in my_resource.yaml
oc policy scc-review -z system:serviceaccount:bob:default -f my_resource.yaml

# Check whether the service account specified in my_resource_with_sa.yaml can admit the pod
oc policy scc-review -f my_resource_with_sa.yaml

# Check whether the default service account can admit the pod; default is taken since no service account is defined in myresource_with_no_sa.yaml
oc policy scc-review -f myresource_with_no_sa.yaml
```

2.5.1.93. oc policy scc-subject-review
Check whether a user or a service account can create a pod

Example usage

```bash
# Check whether user bob can create a pod specified in myresource.yaml
oc policy scc-subject-review -u bob -f myresource.yaml

# Check whether user bob who belongs to projectAdmin group can create a pod specified in myresource.yaml
oc policy scc-subject-review -u bob -g projectAdmin -f myresource.yaml

# Check whether a service account specified in the pod template spec in myresourcewithsa.yaml can create the pod
oc policy scc-subject-review -f myresourcewithsa.yaml
```

2.5.1.94. oc port-forward

Forward one or more local ports to a pod

Example usage

```bash
# Listen on ports 5000 and 6000 locally, forwarding data to/from ports 5000 and 6000 in the pod
oc port-forward pod/mypod 5000 6000

# Listen on ports 5000 and 6000 locally, forwarding data to/from ports 5000 and 6000 in a pod selected by the deployment
oc port-forward deployment/mydeployment 5000 6000

# Listen on port 8443 locally, forwarding to the targetPort of the service's port named "https" in a pod selected by the service
oc port-forward service/myservice 8443:https

# Listen on port 8888 locally, forwarding to 5000 in the pod
oc port-forward pod/mypod 8888:5000

# Listen on port 8888 on all addresses, forwarding to 5000 in the pod
oc port-forward --address 0.0.0.0 pod/mypod 8888:5000

# Listen on port 8888 on localhost and selected IP, forwarding to 5000 in the pod
oc port-forward --address localhost,10.19.21.23 pod/mypod 8888:5000

# Listen on a random port locally, forwarding to 5000 in the pod
oc port-forward pod/mypod :5000
```

2.5.1.95. oc process

Process a template into list of resources

Example usage

```bash
# Convert the template.json file into a resource list and pass to create
oc process -f template.json | oc create -f -

# Process a file locally instead of contacting the server
```
2.5.1.96. oc project

Switch to another project

Example usage

```bash
# Switch to the 'myapp' project
oc project myapp

# Display the project currently in use
oc project
```

2.5.1.97. oc projects

Display existing projects

Example usage

```bash
# List all projects
oc projects
```

2.5.1.98. oc proxy

Run a proxy to the Kubernetes API server

Example usage

```bash
# To proxy all of the Kubernetes API and nothing else
oc proxy --api-prefix=/

# To proxy only part of the Kubernetes API and also some static files
# You can get pods info with 'curl localhost:8001/api/v1/pods'
oc proxy --www=/my/files --www-prefix=/static/ --api-prefix=/api/

# To proxy the entire Kubernetes API at a different root
# You can get pods info with 'curl localhost:8001/custom/api/v1/pods'
```

```
oc process -f template.json --local -o yaml

# Process template while passing a user-defined label
oc process -f template.json -l name=mytemplate

# Convert a stored template into a resource list
oc process foo

# Convert a stored template into a resource list by setting/overriding parameter values
oc process foo PARM1=VALUE1 PARM2=VALUE2

# Convert a template stored in different namespace into a resource list
oc process openshift/foo

# Convert template.json into a resource list
cat template.json | oc process -f -
```
2.5.1.99. oc registry info
Print information about the integrated registry

Example usage

```
# Display information about the integrated registry
oc registry info
```

2.5.1.100. oc registry login
Log in to the integrated registry

Example usage

```
# Log in to the integrated registry
oc registry login

# Log in as the default service account in the current namespace
oc registry login -z default

# Log in to different registry using BASIC auth credentials
oc registry login --registry quay.io/myregistry --auth-basic=USER:PASS
```

2.5.1.101. oc replace
Replace a resource by file name or stdin

Example usage

```
# Replace a pod using the data in pod.json
oc replace -f ./pod.json

# Replace a pod based on the JSON passed into stdin
cat pod.json | oc replace -f -

# Update a single-container pod's image version (tag) to v4
oc get pod mypod -o yaml | sed 's/(image: myimage):.*$/\1:v4/' | oc replace -f -
```
2.5.1.102. oc rollback

Revert part of an application back to a previous deployment

Example usage

```
# Perform a rollback to the last successfully completed deployment for a deployment config
oc rollback frontend

# See what a rollback to version 3 will look like, but do not perform the rollback
oc rollback frontend --to-version=3 --dry-run

# Perform a rollback to a specific deployment
oc rollback frontend-2

# Perform the rollback manually by piping the JSON of the new config back to oc
oc rollback frontend -o json | oc replace dc/frontend -f

# Print the updated deployment configuration in JSON format instead of performing the rollback
oc rollback frontend -o json
```

2.5.1.103. oc rollout cancel

Cancel the in-progress deployment

Example usage

```
# Cancel the in-progress deployment based on 'nginx'
oc rollout cancel dc/nginx
```

2.5.1.104. oc rollout history

View rollout history

Example usage

```
# View the rollout history of a deployment
oc rollout history dc/nginx

# View the details of deployment revision 3
oc rollout history dc/nginx --revision=3
```

2.5.1.105. oc rollout latest

Start a new rollout for a deployment config with the latest state from its triggers

Example usage
2.5.1.106. oc rollout pause
Mark the provided resource as paused

Example usage

```
# Mark the nginx deployment as paused. Any current state of
# the deployment will continue its function, new updates to the deployment will not
# have an effect as long as the deployment is paused
oc rollout pause dc/nginx
```

2.5.1.107. oc rollout restart
Restart a resource

Example usage

```
# Restart a deployment
oc rollout restart deployment/nginx

# Restart a daemon set
oc rollout restart daemonset/abc
```

2.5.1.108. oc rollout resume
Resume a paused resource

Example usage

```
# Resume an already paused deployment
oc rollout resume dc/nginx
```

2.5.1.109. oc rollout retry
Retry the latest failed rollout

Example usage

```
# Retry the latest failed deployment based on 'frontend'
# The deployer pod and any hook pods are deleted for the latest failed deployment
oc rollout retry dc/frontend
```

2.5.1.110. oc rollout status
Show the status of the rollout
Example usage

```
# Watch the status of the latest rollout
oc rollout status dc/nginx
```

2.5.1.111. oc rollout undo

Undo a previous rollout

Example usage

```
# Roll back to the previous deployment
oc rollout undo dc/nginx

# Roll back to deployment revision 3. The replication controller for that version must exist
oc rollout undo dc/nginx --to-revision=3
```

2.5.1.112. oc rsh

Start a shell session in a container

Example usage

```
# Open a shell session on the first container in pod ‘foo’
oc rsh foo

# Open a shell session on the first container in pod ‘foo’ and namespace ‘bar’
# (Note that oc client specific arguments must come before the resource name and its arguments)
oc rsh -n bar foo

# Run the command ‘cat /etc/resolv.conf’ inside pod ‘foo’
oc rsh foo cat /etc/resolv.conf

# See the configuration of your internal registry
oc rsh dc/docker-registry cat config.yml

# Open a shell session on the container named ‘index’ inside a pod of your job
oc rsh -c index job/sheduled
```

2.5.1.113. oc rsync

Copy files between a local file system and a pod

Example usage

```
# Synchronize a local directory with a pod directory
oc rsync ./local/dir/ POD:/remote/dir

# Synchronize a pod directory with a local directory
oc rsync POD:/remote/dir/ ./local/dir
```

2.5.1.114. oc run

# Watch the status of the latest rollout
oc rollout status dc/nginx
Run a particular image on the cluster

Example usage

```yaml
# Start a nginx pod
oc run nginx --image=nginx

# Start a hazelcast pod and let the container expose port 5701
oc run hazelcast --image=hazelcast/hazelcast --port=5701

# Start a hazelcast pod and set environment variables "DNS_DOMAIN=cluster" and "POD_NAMESPACE=default" in the container
oc run hazelcast --image=hazelcast/hazelcast --env="DNS_DOMAIN=cluster" --env="POD_NAMESPACE=default"

# Start a hazelcast pod and set labels "app=hazelcast" and "env=prod" in the container
oc run hazelcast --image=hazelcast/hazelcast --labels="app=hazelcast,env=prod"

# Dry run; print the corresponding API objects without creating them
oc run nginx --image=nginx --dry-run=client

# Start a nginx pod, but overload the spec with a partial set of values parsed from JSON
oc run nginx --image=nginx --overrides='{"apiVersion": "v1", "spec": { ... }}'

# Start a busybox pod and keep it in the foreground, don't restart it if it exits
oc run -i -t busybox --image=busybox --restart=Never

# Start the nginx pod using the default command, but use custom arguments (arg1 .. argN) for that command
oc run nginx --image=nginx -- <arg1> <arg2> ... <argN>

# Start the nginx pod using a different command and custom arguments
oc run nginx --image=nginx --command -- <cmd> <arg1> ... <argN>
```

2.5.1.115. oc scale

Set a new size for a deployment, replica set, or replication controller

Example usage

```yaml
# Scale a replica set named 'foo' to 3
oc scale --replicas=3 rs/foo

# Scale a resource identified by type and name specified in "foo.yaml" to 3
oc scale --replicas=3 -f foo.yaml

# If the deployment named mysql's current size is 2, scale mysql to 3
oc scale --current-replicas=2 --replicas=3 deployment/mysql

# Scale multiple replication controllers
oc scale --replicas=5 rc/foo rc/bar rc/baz

# Scale stateful set named 'web' to 3
oc scale --replicas=3 statefulset/web
```
2.5.116. oc secrets link
Link secrets to a service account

Example usage

```bash
# Add an image pull secret to a service account to automatically use it for pulling pod images
oc secrets link serviceaccount-name pull-secret --for=pull

# Add an image pull secret to a service account to automatically use it for both pulling and pushing build images
oc secrets link builder builder-image-secret --for=pull,mount
```

2.5.117. oc secrets unlink
Detach secrets from a service account

Example usage

```bash
# Unlink a secret currently associated with a service account
oc secrets unlink serviceaccount-name secret-name another-secret-name ...
```

2.5.118. oc serviceaccounts create-kubeconfig
Generate a kubeconfig file for a service account

Example usage

```bash
# Create a kubeconfig file for service account 'default'
oc serviceaccounts create-kubeconfig 'default' > default.kubeconfig
```

2.5.119. oc serviceaccounts get-token
Get a token assigned to a service account

Example usage

```bash
# Get the service account token from service account 'default'
oc serviceaccounts get-token 'default'
```

2.5.120. oc serviceaccounts new-token
Generate a new token for a service account

Example usage

```bash
# Generate a new token for service account 'default'
oc serviceaccounts new-token 'default'

# Generate a new token for service account 'default' and apply
# labels 'foo' and 'bar' to the new token for identification
oc serviceaccounts new-token 'default' --labels foo=foo-value,bar=bar-value
```
2.5.1.121. oc set build-hook

Update a build hook on a build config

Example usage

```bash
# Clear post-commit hook on a build config
oc set build-hook bc/mybuild --post-commit --remove

# Set the post-commit hook to execute a test suite using a new entrypoint
oc set build-hook bc/mybuild --post-commit --command -- /bin/bash -c /var/lib/test-image.sh

# Set the post-commit hook to execute a shell script
oc set build-hook bc/mybuild --post-commit --script="/var/lib/test-image.sh param1 param2 && /var/lib/done.sh"
```

2.5.1.122. oc set build-secret

Update a build secret on a build config

Example usage

```bash
# Clear the push secret on a build config
oc set build-secret --push --remove bc/mybuild

# Set the pull secret on a build config
oc set build-secret --pull bc/mybuild mysecret

# Set the push and pull secret on a build config
oc set build-secret --push --pull bc/mybuild mysecret

# Set the source secret on a set of build configs matching a selector
oc set build-secret --source -l app=myapp gitsecret
```

2.5.1.123. oc set data

Update the data within a config map or secret

Example usage

```bash
# Set the 'password' key of a secret
oc set data secret/foo password=this_is_secret

# Remove the 'password' key from a secret
oc set data secret/foo password-

# Update the 'haproxy.conf' key of a config map from a file on disk
oc set data configmap/bar --from-file=../haproxy.conf

# Update a secret with the contents of a directory, one key per file
oc set data secret/foo --from-file=secret-dir
```

2.5.1.124. oc set deployment-hook
Update a deployment hook on a deployment config

Example usage

```bash
# Clear pre and post hooks on a deployment config
oc set deployment-hook dc/myapp --remove --pre --post

# Set the pre deployment hook to execute a db migration command for an application
# using the data volume from the application
oc set deployment-hook dc/myapp --pre --volumes=data -- /var/lib/migrate-db.sh

# Set a mid deployment hook along with additional environment variables
oc set deployment-hook dc/myapp --mid --volumes=data -e VAR1=value1 -e VAR2=value2 -- /var/lib/prepare-deploy.sh
```

2.5.1.125. oc set env

Update environment variables on a pod template

Example usage

```bash
# Update deployment config 'myapp' with a new environment variable
oc set env dc/myapp STORAGE_DIR=/local

# List the environment variables defined on a build config 'sample-build'
oc set env bc/sample-build --list

# List the environment variables defined on all pods
oc set env pods --all --list

# Output modified build config in YAML
oc set env bc/sample-build STORAGE_DIR=/data -o yaml

# Update all containers in all replication controllers in the project to have ENV=prod
oc set env rc --all ENV=prod

# Import environment from a secret
oc set env --from=secret/mysecret dc/myapp

# Import environment from a config map with a prefix
oc set env --from=configmap/myconfigmap --prefix=MYSQL_ dc/myapp

# Remove the environment variable ENV from container 'c1' in all deployment configs
oc set env dc --all --containers="c1" ENV-

# Remove the environment variable ENV from a deployment config definition on disk and
# update the deployment config on the server
oc set env -f dc.json ENV-

# Set some of the local shell environment into a deployment config on the server
oc set env | grep RAILS_ | oc env -e - dc/myapp
```
Example the image of a pod template

Example usage

```bash
# Set a deployment config's `nginx` container image to `nginx:1.9.1`, and its `busybox` container image to `busybox`.
oc set image dc/nginx busybox=busybox nginx=nginx:1.9.1

# Set a deployment config's `app` container image to the image referenced by the imagestream tag `openshift/ruby:2.3`
oc set image dc/myapp app=openshift/ruby:2.3 --source=imagestreamtag

# Update all deployments' and rc's `nginx` container's image to `nginx:1.9.1`
oc set image deployments,rc nginx=nginx:1.9.1 --all

# Update image of all containers of daemonset `abc` to `nginx:1.9.1`
oc set image daemonset abc *=nginx:1.9.1

# Print result (in yaml format) of updating `nginx` container image from local file, without hitting the server
oc set image -f path/to/file.yaml nginx=nginx:1.9.1 --local -o yaml
```

2.5.127. oc set image-lookup

Change how images are resolved when deploying applications

Example usage

```bash
# Print all of the image streams and whether they resolve local names
oc set image-lookup

# Use local name lookup on image stream `mysql`
oc set image-lookup mysql

# Force a deployment to use local name lookup
oc set image-lookup deploy/mysql

# Show the current status of the deployment lookup
oc set image-lookup deploy/mysql --list

# Disable local name lookup on image stream `mysql`
oc set image-lookup mysql --enabled=false

# Set local name lookup on all image streams
oc set image-lookup --all
```

2.5.128. oc set probe

Update a probe on a pod template

Example usage

```bash
# Clear both readiness and liveness probes off all containers
oc set probe dc/myapp --remove --readiness --liveness
```
# Set an exec action as a liveness probe to run 'echo ok'
oc set probe dc/myapp --liveness -- echo ok

# Set a readiness probe to try to open a TCP socket on 3306
oc set probe rc/mysql --readiness --open-tcp=3306

# Set an HTTP startup probe for port 8080 and path /healthz over HTTP on the pod IP
oc set probe dc/webapp --startup --get-url=http://:8080/healthz

# Set an HTTP readiness probe for port 8080 and path /healthz over HTTP on the pod IP
oc set probe dc/webapp --readiness --get-url=http://:8080/healthz

# Set an HTTP readiness probe over HTTPS on 127.0.0.1:1936 for a hostNetwork pod
oc set probe dc/router --readiness --get-url=https://127.0.0.1:1936/stats

# Set only the initial-delay-seconds field on all deployments
oc set probe dc --all --readiness --initial-delay-seconds=30

2.5.1.129. oc set resources

Update resource requests/limits on objects with pod templates

Example usage

# Set a deployments nginx container CPU limits to "200m and memory to 512Mi"
oc set resources deployment nginx -c=nginx --limits=cpu=200m,memory=512Mi

# Set the resource request and limits for all containers in nginx
oc set resources deployment nginx --limits=cpu=200m,memory=512Mi --requests=cpu=100m,memory=256Mi

# Remove the resource requests for resources on containers in nginx
oc set resources deployment nginx --limits=cpu=0,memory=0 --requests=cpu=0,memory=0

# Print the result (in YAML format) of updating nginx container limits locally, without hitting the server
oc set resources -f path/to/file.yaml --limits=cpu=200m,memory=512Mi --local -o yaml

2.5.1.130. oc set route-backends

Update the backends for a route

Example usage

# Print the backends on the route 'web'
oc set route-backends web

# Set two backend services on route 'web' with 2/3rds of traffic going to 'a'
oc set route-backends web a=2 b=1

# Increase the traffic percentage going to b by 10% relative to a
oc set route-backends web --adjust b=+10%

# Set traffic percentage going to b to 10% of the traffic going to a
2.5.1.131. oc set selector

Set the selector on a resource

Example usage

```
# Set labels and selector before creating a deployment/service pair.
oc create service clusterip my-svc --clusterip="None" -o yaml --dry-run | oc set selector --local -f -environment=qa -o yaml | oc create -f -
oc create deployment my-dep -o yaml --dry-run | oc label --local -f -environment=qa -o yaml | oc create -f -
```

2.5.1.132. oc set serviceaccount

Update the service account of a resource

Example usage

```
# Set deployment nginx-deployment's service account to serviceaccount1
oc set serviceaccount deployment nginx-deployment serviceaccount1

# Print the result (in YAML format) of updated nginx deployment with service account from a local file, without hitting the API server
oc set sa -f nginx-deployment.yaml serviceaccount1 --local --dry-run -o yaml
```

2.5.1.133. oc set subject

Update the user, group, or service account in a role binding or cluster role binding

Example usage

```
# Update a cluster role binding for serviceaccount1
oc set subject clusterrolebinding admin --serviceaccount=namespace:serviceaccount1

# Update a role binding for user1, user2, and group1
oc set subject rolebinding admin --user=user1 --user=user2 --group=group1

# Print the result (in YAML format) of updating role binding subjects locally, without hitting the server
oc create rolebinding admin --role=admin --user=admin -o yaml --dry-run | oc set subject --local -f --user=foo -o yaml
```

2.5.1.134. oc set triggers

Update the triggers on one or more objects
Example usage

```bash
# Print the triggers on the deployment config 'myapp'
oc set triggers dc/myapp

# Set all triggers to manual
oc set triggers dc/myapp --manual

# Enable all automatic triggers
oc set triggers dc/myapp --auto

# Reset the GitHub webhook on a build to a new, generated secret
oc set triggers bc/webapp --from-github
oc set triggers bc/webapp --from-webhook

# Remove all triggers
oc set triggers bc/webapp --remove-all

# Stop triggering on config change
oc set triggers dc/myapp --from-config --remove

# Add an image trigger to a build config
oc set triggers bc/webapp --from-image=namespace1/image:latest

# Add an image trigger to a stateful set on the main container
oc set triggers statefulset/db --from-image=namespace1/image:latest -c main
```

2.5.135. `oc set volumes`

Update volumes on a pod template

Example usage

```bash
# List volumes defined on all deployment configs in the current project
oc set volume dc --all

# Add a new empty dir volume to deployment config (dc) 'myapp' mounted under # /var/lib/myapp
oc set volume dc/myapp --add --mount-path=/var/lib/myapp

# Use an existing persistent volume claim (pvc) to overwrite an existing volume 'v1'
oc set volume dc/myapp --add --name=v1 -t pvc --claim-name=pvc1 --overwrite

# Remove volume 'v1' from deployment config 'myapp'
oc set volume dc/myapp --remove --name=v1

# Create a new persistent volume claim that overwrites an existing volume 'v1'
oc set volume dc/myapp --add --name=v1 -t pvc --claim-size=1G --overwrite

# Change the mount point for volume 'v1' to /data
oc set volume dc/myapp --add --name=v1 -m /data --overwrite

# Modify the deployment config by removing volume mount "v1" from container "c1"
# (and by removing the volume "v1" if no other containers have volume mounts that reference it)
oc set volume dc/myapp --remove --name=v1 --containers=c1
```
2.5.1.136. oc start-build

Start a new build

Example usage

```bash
# Starts build from build config "hello-world"
oc start-build hello-world

# Starts build from a previous build "hello-world-1"
oc start-build --from-build=hello-world-1

# Use the contents of a directory as build input
oc start-build hello-world --from-dir=src/

# Send the contents of a Git repository to the server from tag 'v2'
oc start-build hello-world --from-repo=../hello-world --commit=v2

# Start a new build for build config "hello-world" and watch the logs until the build
# completes or fails
oc start-build hello-world --follow

# Start a new build for build config "hello-world" and wait until the build completes. It
# exits with a non-zero return code if the build fails
oc start-build hello-world --wait
```

2.5.1.137. oc status

Show an overview of the current project

Example usage

```bash
# See an overview of the current project
oc status

# Export the overview of the current project in an svg file
oc status -o dot | dot -T svg -o project.svg

# See an overview of the current project including details for any identified issues
oc status --suggest
```

2.5.1.138. oc tag

Tag existing images into image streams

Example usage

```bash
# Tag the current image for the image stream 'openshift/ruby' and tag '2.0' into the image stream
```
2.5.139. oc version

Print the client and server version information

Example usage

```bash
# Print the OpenShift client, kube-apiserver, and openshift-apiserver version information for the current context
oc version

# Print the OpenShift client, kube-apiserver, and openshift-apiserver version numbers for the current context
oc version --short

# Print the OpenShift client version information for the current context
oc version --client
```

2.5.140. oc wait

Experimental: Wait for a specific condition on one or many resources

Example usage

```bash
# Wait for the pod "busybox1" to contain the status condition of type "Ready"
oc wait --for=condition=Ready pod/busybox1

# The default value of status condition is true; you can set it to false
oc wait --for=condition=Ready=false pod/busybox1

# Wait for the pod "busybox1" to contain the status phase to be "Running"
oc wait --for=jsonpath='{.status.phase}'=Running pod/busybox1

# Wait for the pod "busybox1" to be deleted, with a timeout of 60s, after having issued the "delete" command
oc delete pod/busybox1
oc wait --for=delete pod/busybox1 --timeout=60s
```
2.5.1.141. oc whoami

Return information about the current session

Example usage

# Display the currently authenticated user
oc whoami

2.5.2. Additional resources

- OpenShift CLI administrator command reference

2.6. OPENSIFT CLI ADMINISTRATOR COMMAND REFERENCE

This reference provides descriptions and example commands for OpenShift CLI (oc) administrator commands. You must have cluster-admin or equivalent permissions to use these commands.

For developer commands, see the OpenShift CLI developer command reference.

Run oc adm help to list all administrator commands or run oc <command> --help to get additional details for a specific command.

2.6.1. OpenShift CLI (oc) administrator commands

2.6.1.1. oc adm build-chain

Output the inputs and dependencies of your builds

Example usage

# Build the dependency tree for the 'latest' tag in <image-stream>
oc adm build-chain <image-stream>

# Build the dependency tree for the 'v2' tag in dot format and visualize it via the dot utility
oc adm build-chain <image-stream>:v2 -o dot | dot -T svg -o deps.svg

# Build the dependency tree across all namespaces for the specified image stream tag found in the 'test' namespace
oc adm build-chain <image-stream> -n test --all

2.6.1.2. oc adm catalog mirror

Mirror an operator-registry catalog

Example usage

# Mirror an operator-registry image and its contents to a registry
oc adm catalog mirror quay.io/my/image:latest myregistry.com

# Mirror an operator-registry image and its contents to a particular namespace in a registry
oc adm catalog mirror quay.io/my/image:latest myregistry.com/my-namespace
2.6.1.3. oc adm certificate approve

Approve a certificate signing request

Example usage

```bash
# Approve CSR 'csr-sqgzp'
oc adm certificate approve csr-sqgzp
```

2.6.1.4. oc adm certificate deny

Deny a certificate signing request

Example usage

```bash
# Deny CSR 'csr-sqgzp'
oc adm certificate deny csr-sqgzp
```

2.6.1.5. oc adm cordon

Mark node as unschedulable

Example usage

```bash
# Mark node "foo" as unschedulable
oc adm cordon foo
```

2.6.1.6. oc adm create-bootstrap-project-template

Create a bootstrap project template

Example usage

```bash
# Output a bootstrap project template in YAML format to stdout
oc adm create-bootstrap-project-template -o yaml
```

2.6.1.7. oc adm create-error-template

CHAPTER 2. OPENShift CLI (OC)
Create an error page template

Example usage

```
# Output a template for the error page to stdout
oc adm create-error-template
```

2.6.1.8. oc adm create-login-template

Create a login template

Example usage

```
# Output a template for the login page to stdout
oc adm create-login-template
```

2.6.1.9. oc adm create-provider-selection-template

Create a provider selection template

Example usage

```
# Output a template for the provider selection page to stdout
oc adm create-provider-selection-template
```

2.6.1.10. oc adm drain

Drain node in preparation for maintenance

Example usage

```
# Drain node "foo", even if there are pods not managed by a replication controller, replica set, job, daemon set or stateful set on it
oc adm drain foo --force

# As above, but abort if there are pods not managed by a replication controller, replica set, job, daemon set or stateful set, and use a grace period of 15 minutes
oc adm drain foo --grace-period=900
```

2.6.1.11. oc adm groups add-users

Add users to a group

Example usage

```
# Add user1 and user2 to my-group
oc adm groups add-users my-group user1 user2
```

2.6.1.12. oc adm groups new

Create a new group
Example usage

```bash
# Add a group with no users
oc adm groups new my-group

# Add a group with two users
oc adm groups new my-group user1 user2

# Add a group with one user and shorter output
oc adm groups new my-group user1 -o name
```

2.6.1.13. oc adm groups prune

Remove old OpenShift groups referencing missing records from an external provider

Example usage

```bash
# Prune all orphaned groups
oc adm groups prune --sync-config=/path/to/ldap-sync-config.yaml --confirm

# Prune all orphaned groups except the ones from the blacklist file
oc adm groups prune --blacklist=/path/to/blacklist.txt --sync-config=/path/to/ldap-sync-config.yaml --confirm

# Prune all orphaned groups from a list of specific groups specified in a whitelist file
oc adm groups prune --whitelist=/path/to/whitelist.txt --sync-config=/path/to/ldap-sync-config.yaml --confirm

# Prune all orphaned groups from a list of specific groups specified in a whitelist
oc adm groups prune groups/group_name groups/other_name --sync-config=/path/to/ldap-sync-config.yaml --confirm
```

2.6.1.14. oc adm groups remove-users

Remove users from a group

Example usage

```bash
# Remove user1 and user2 from my-group
oc adm groups remove-users my-group user1 user2
```

2.6.1.15. oc adm groups sync

Sync OpenShift groups with records from an external provider

Example usage

```bash
# Sync all groups with an LDAP server
oc adm groups sync --sync-config=/path/to/ldap-sync-config.yaml --confirm

# Sync all groups except the ones from the blacklist file with an LDAP server
oc adm groups sync --blacklist=/path/to/blacklist.txt --sync-config=/path/to/ldap-sync-config.yaml --confirm
```
2.6.1.16. **oc adm inspect**

Collect debugging data for a given resource

**Example usage**

```bash
# Collect debugging data for the "openshift-apiserver" clusteroperator
oc adm inspect clusteroperator/openshift-apiserver

# Collect debugging data for the "openshift-apiserver" and "kube-apiserver" clusteroperators
oc adm inspect clusteroperator/openshift-apiserver clusteroperator/kube-apiserver

# Collect debugging data for all clusteroperators
oc adm inspect clusteroperator

# Collect debugging data for all clusteroperators and clusterversions
oc adm inspect clusteroperators,clusterversions
```

2.6.1.17. **oc adm migrate template-instances**

Update template instances to point to the latest group-version-kinds

**Example usage**

```bash
# Perform a dry-run of updating all objects
oc adm migrate template-instances

# To actually perform the update, the confirm flag must be appended
oc adm migrate template-instances --confirm
```

2.6.1.18. **oc adm must-gather**

Launch a new instance of a pod for gathering debug information

**Example usage**

```bash
# Gather information using the default plug-in image and command, writing into ./must-gather.local.<rand>
oc adm must-gather

# Gather information with a specific local folder to copy to
```
2.6.19. oc adm new-project

Create a new project

Example usage

# Create a new project using a node selector
oc adm new-project myproject --node-selector='type=user-node,region=east'

2.6.20. oc adm node-logs

Display and filter node logs

Example usage

# Show kubelet logs from all masters
oc adm node-logs --role master -u kubelet

# See what logs are available in masters in /var/logs
oc adm node-logs --role master --path=/

# Display cron log file from all masters
oc adm node-logs --role master --path=cron

2.6.21. oc adm pod-network isolate-projects

Isolate project network

Example usage

# Provide isolation for project p1
oc adm pod-network isolate-projects <p1>

# Allow all projects with label name=top-secret to have their own isolated project network
oc adm pod-network isolate-projects --selector='name=top-secret'

2.6.22. oc adm pod-network join-projects
Join project network

Example usage

```bash
# Allow project p2 to use project p1 network
oc adm pod-network join-projects --to=<p1> <p2>

# Allow all projects with label name=top-secret to use project p1 network
oc adm pod-network join-projects --to=<p1> --selector='name=top-secret'
```

2.6.1.23. oc adm pod-network make-projects-global

Make project network global

Example usage

```bash
# Allow project p1 to access all pods in the cluster and vice versa
oc adm pod-network make-projects-global <p1>

# Allow all projects with label name=share to access all pods in the cluster and vice versa
oc adm pod-network make-projects-global --selector='name=share'
```

2.6.1.24. oc adm policy add-role-to-user

Add a role to users or service accounts for the current project

Example usage

```bash
# Add the 'view' role to user1 for the current project
oc adm policy add-role-to-user view user1

# Add the 'edit' role to serviceaccount1 for the current project
oc adm policy add-role-to-user edit -z serviceaccount1
```

2.6.1.25. oc adm policy add-scc-to-group

Add a security context constraint to groups

Example usage

```bash
# Add the 'restricted' security context constraint to group1 and group2
oc adm policy add-scc-to-group restricted group1 group2
```

2.6.1.26. oc adm policy add-scc-to-user

Add a security context constraint to users or a service account

Example usage

```bash
# Add the 'restricted' security context constraint to user1 and user2
oc adm policy add-scc-to-user restricted user1 user2
```
2.6.1.27. oc adm policy scc-review

Check which service account can create a pod

Example usage

```
# Check whether service accounts sa1 and sa2 can admit a pod with a template pod spec specified in my_resource.yaml
# Service Account specified in myresource.yaml file is ignored
oc adm policy scc-review -z sa1,sa2 -f my_resource.yaml

# Check whether service accounts system:serviceaccount:bob:default can admit a pod with a template pod spec specified in my_resource.yaml
oc adm policy scc-review -z system:serviceaccount:bob:default -f my_resource.yaml

# Check whether the service account specified in my_resource_with_sa.yaml can admit the pod
oc adm policy scc-review -f my_resource_with_sa.yaml

# Check whether the default service account can admit the pod; default is taken since no service account is defined in myresource_with_no_sa.yaml
oc adm policy scc-review -f myresource_with_no_sa.yaml
```

2.6.1.28. oc adm policy scc-subject-review

Check whether a user or a service account can create a pod

Example usage

```
# Check whether user bob can create a pod specified in myresource.yaml
oc adm policy scc-subject-review -u bob -f myresource.yaml

# Check whether user bob who belongs to projectAdmin group can create a pod specified in myresource.yaml
oc adm policy scc-subject-review -u bob -g projectAdmin -f myresource.yaml

# Check whether a service account specified in the pod template spec in myresourcewithsa.yaml can create the pod
oc adm policy scc-subject-review -f myresourcewithsa.yaml
```

2.6.1.29. oc adm prune builds

Remove old completed and failed builds

Example usage

```
# Dry run deleting older completed and failed builds and also including
# all builds whose associated build config no longer exists
oc adm prune builds --orphans
```
2.6.1.30. oc adm prune deployments

Remove old completed and failed deployment configs

Example usage

```
# To actually perform the prune operation, the confirm flag must be appended
oc adm prune deployments --orphans --confirm
```

```
# Dry run deleting all but the last complete deployment for every deployment config
oc adm prune deployments --keep-complete=1

# To actually perform the prune operation, the confirm flag must be appended
oc adm prune deployments --keep-complete=1 --confirm
```

2.6.1.31. oc adm prune groups

Remove old OpenShift groups referencing missing records from an external provider

Example usage

```
# Prune all orphaned groups
oc adm prune groups --sync-config=/path/to/ldap-sync-config.yaml --confirm

# Prune all orphaned groups except the ones from the blacklist file
oc adm prune groups --blacklist=/path/to/blacklist.txt --sync-config=/path/to/ldap-sync-config.yaml --confirm

# Prune all orphaned groups from a list of specific groups specified in a whitelist file
oc adm prune groups --whitelist=/path/to/whitelist.txt --sync-config=/path/to/ldap-sync-config.yaml --confirm

# Prune all orphaned groups from a list of specific groups specified in a whitelist
oc adm prune groups groups/group_name groups/other_name --sync-config=/path/to/ldap-sync-config.yaml --confirm
```

2.6.1.32. oc adm prune images

Remove unreferenced images

Example usage

```
# See what the prune command would delete if only images and their referrers were more than an hour old
# and obsoleted by 3 newer revisions under the same tag were considered
oc adm prune images --keep-tag-revisions=3 --keep-younger-than=60m

# To actually perform the prune operation, the confirm flag must be appended
oc adm prune images --keep-tag-revisions=3 --keep-younger-than=60m --confirm

# See what the prune command would delete if we are interested in removing images exceeding currently set limit ranges ('openshift.io/Image')
```
2.6.1.33. oc adm release extract

Extract the contents of an update payload to disk

Example usage

```
# Use git to check out the source code for the current cluster release to DIR
oc adm release extract --git=DIR

# Extract cloud credential requests for AWS
oc adm release extract --credentials-requests --cloud=aws
```

2.6.1.34. oc adm release info

Display information about a release

Example usage

```
# Show information about the cluster's current release
oc adm release info

# Show the source code that comprises a release
oc adm release info 4.2.2 --commit-urls

# Show the source code difference between two releases
oc adm release info 4.2.0 4.2.2 --commits

# Show where the images referenced by the release are located
oc adm release info quay.io/openshift-release-dev/ocp-release:4.2.2 --pullspecs
```

2.6.1.35. oc adm release mirror

Mirror a release to a different image registry location

Example usage

```
# Perform a dry run showing what would be mirrored, including the mirror objects
ocadm release mirror 4.3.0 --to myregistry.local/openshift/release \   --release-image-signature-to-dir /tmp/releases --dry-run

# Mirror a release into the current directory
```
2.6.36. `oc adm release new`

Create a new OpenShift release

**Example usage**

```
# Create a release from the latest origin images and push to a DockerHub repo
oc adm release new --from-image-stream=4.1 -n origin --to-image
docker.io/mycompany/myrepo:latest

# Create a new release with updated metadata from a previous release
oc adm release new --from-release registry.svc.ci.openshift.org/origin/release:v4.1 --name 4.1.1 \
--previous 4.1.0 --metadata ... --to-image docker.io/mycompany/myrepo:latest

# Create a new release and override a single image
oc adm release new --from-release registry.svc.ci.openshift.org/origin/release:v4.1 \
cli=docker.io/mycompany/cli:latest --to-image docker.io/mycompany/myrepo:latest

# Run a verification pass to ensure the release can be reproduced
oc adm release new --from-release registry.svc.ci.openshift.org/origin/release:v4.1
```

2.6.37. `oc adm taint`

Update the taints on one or more nodes

**Example usage**

```
# Update node 'foo' with a taint with key 'dedicated' and value 'special-user' and effect 'NoSchedule'
# If a taint with that key and effect already exists, its value is replaced as specified
oc adm taint nodes foo dedicated=special-user:NoSchedule

# Remove from node 'foo' the taint with key 'dedicated' and effect 'NoSchedule' if one exists
oc adm taint nodes foo dedicated:NoSchedule- 

# Remove from node 'foo' all the taints with key 'dedicated'
oc adm taint nodes foo dedicated-

# Add a taint with key 'dedicated' on nodes having label mylabel=X
oc adm taint node -l myLabel=X dedicated=foo:PreferNoSchedule
```
# Add to node 'foo' a taint with key 'bar' and no value
oc adm taint nodes foo bar:NoSchedule

2.6.1.38. oc adm top images
Show usage statistics for images

Example usage

  # Show usage statistics for images
  oc adm top images

2.6.1.39. oc adm top imagestreams
Show usage statistics for image streams

Example usage

  # Show usage statistics for image streams
  oc adm top imagestreams

2.6.1.40. oc adm top node
Display resource (CPU/memory) usage of nodes

Example usage

  # Show metrics for all nodes
  oc adm top node

  # Show metrics for a given node
  oc adm top node NODE_NAME

2.6.1.41. oc adm top pod
Display resource (CPU/memory) usage of pods

Example usage

  # Show metrics for all pods in the default namespace
  oc adm top pod

  # Show metrics for all pods in the given namespace
  oc adm top pod --namespace=NAMESPACE

  # Show metrics for a given pod and its containers
  oc adm top pod POD_NAME --containers

  # Show metrics for the pods defined by label name=myLabel
  oc adm top pod -l name=myLabel
2.6.1.42. oc adm uncordon

Mark node as schedulable

Example usage

```bash
# Mark node "foo" as schedulable
oc adm uncordon foo
```

2.6.1.43. oc adm upgrade

Upgrade a cluster

Example usage

```bash
# Review the available cluster updates
oc adm upgrade

# Update to the latest version
oc adm upgrade --to-latest=true
```

2.6.1.44. oc adm verify-image-signature

Verify the image identity contained in the image signature

Example usage

```bash
# Verify the image signature and identity using the local GPG keychain
oc adm verify-image-signature
sha256:c841e9b64e4579bd56c794bdd7c36e1c257110fd2404bebb8b613e4935228c4 \ --expected-identity=registry.local:5000/foo/bar:v1

# Verify the image signature and identity using the local GPG keychain and save the status
oc adm verify-image-signature
sha256:c841e9b64e4579bd56c794bdd7c36e1c257110fd2404bebb8b613e4935228c4 \ --expected-identity=registry.local:5000/foo/bar:v1 --save

# Verify the image signature and identity via exposed registry route
oc adm verify-image-signature
sha256:c841e9b64e4579bd56c794bdd7c36e1c257110fd2404bebb8b613e4935228c4 \ --expected-identity=registry.local:5000/foo/bar:v1 \ --registry-url=docker-registry.foo.com

# Remove all signature verifications from the image
oc adm verify-image-signature
sha256:c841e9b64e4579bd56c794bdd7c36e1c257110fd2404bebb8b613e4935228c4 --remove-all
```

2.6.2. Additional resources

- OpenShift CLI developer command reference

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

### 2.7.1. The oc binary

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

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

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

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

**IMPORTANT**

If you installed an earlier version of the `oc` binary, you cannot use it to complete all of the commands in OpenShift Container Platform 4.10. If you want the latest features, you must download and install the latest version of the `oc` binary corresponding to your OpenShift Container Platform server version.

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

#### Table 2.2. Compatibility Matrix

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

- **1** Fully compatible.
2. oc client might not be able to access server features.

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

2.7.2. The kubectl binary

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

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

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

3.1. odo RELEASE NOTES

3.1.1. Notable changes and improvements in odo version 2.5.0

- Creates unique routes for each component, using `adler32` hashing
- Supports additional fields in the devfile for assigning resources:
  - cpuRequest
  - cpuLimit
  - memoryRequest
  - memoryLimit
- Adds the `--deploy` flag to the `odo delete` command, to remove components deployed using the `odo deploy` command:
  
  ```
  $ odo delete --deploy
  ```
- Adds mapping support to the `odo link` command
- Supports ephemeral volumes using the `ephemeral` field in `volume` components
- Sets the default answer to `yes` when asking for telemetry opt-in
- Improves metrics by sending additional telemetry data to the devfile registry
- Updates the bootstrap image to `registry.access.redhat.com/ocp-tools-4/odo-init-container-rhel8:1.1.11`
- The upstream repository is available at `https://github.com/redhat-developer/odo`

3.1.2. Bug fixes

- Previously, `odo deploy` would fail if the `.odo/env` file did not exist. The command now creates the `.odo/env` file if required.

- Previously, interactive component creation using the `odo create` command would fail if disconnect from the cluster. This issue is fixed in the latest release.

3.1.3. Getting support

**For Product**

If you find an error, encounter a bug, or have suggestions for improving the functionality of odo, file an issue in Bugzilla. Choose [OpenShift Developer Tools and Services](https://github.com/redhat-developer/odo) as a product type and odo as a component.

Provide as many details in the issue description as possible.
For Documentation

If you find an error or have suggestions for improving the documentation, file an issue in Bugzilla. Choose the OpenShift Container Platform product type and the Documentation component type.

3.2. UNDERSTANDING ODO

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

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

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

3.2.1. odo key features

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

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

3.2.2. odo core concepts

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

Application

A typical application, developed with a cloud-native approach, that is used to perform a particular task.

Examples of applications include online video streaming, online shopping, and hotel reservation systems.

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

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

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

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

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

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

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

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

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

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

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

#### Procedure
1. Log in to the cluster with `odo`:
   ```bash
   $ odo login -u developer -p developer
   ```
2. List the available `odo` components:
   ```bash
   $ odo catalog list components
   ```

#### Example output
Odo Devfile Components:
### 3.2.4. Telemetry in odo

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
<th>REGISTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>dotnet50</td>
<td>Stack with .NET 5.0</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>dotnet60</td>
<td>Stack with .NET 6.0</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>dotnetcore31</td>
<td>Stack with .NET Core 3.1</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>go</td>
<td>Stack with the latest Go version</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-maven</td>
<td>Upstream Maven and OpenJDK 11</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-openliberty</td>
<td>Java application Maven-built stack using the Open Liberty runtime</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-openliberty-gradle</td>
<td>Java application Gradle-built stack using the Open Liberty runtime</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-quarkus</td>
<td>Quarkus with Java</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-springboot</td>
<td>Spring Boot® using Java</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-vertx</td>
<td>Upstream Vert.x using Java</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-websphereliberty</td>
<td>Java application Maven-built stack using the WebSphere Liberty</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-websphereliberty-gradle</td>
<td>Java application Gradle-built stack using the WebSphere Liberty</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-wildfly</td>
<td>Upstream WildFly</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-wildfly-bootable-jar</td>
<td>Java stack with WildFly in bootable Jar mode, OpenJDK 11 and...</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>nodejs</td>
<td>Stack with Node.js 14</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>nodejs-angular</td>
<td>Stack with Angular 12</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>nodejs-nextjs</td>
<td>Stack with Next.js 11</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>nodejs-nuxtjs</td>
<td>Stack with Nuxt.js 2</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>nodejs-react</td>
<td>Stack with React 17</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>nodejs-svelte</td>
<td>Stack with Svelte 3</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>nodejs-vue</td>
<td>Stack with Vue 3</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>php-laravel</td>
<td>Stack with Laravel 8</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>python</td>
<td>Python Stack with Python 3.7</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>python-django</td>
<td>Python3.7 with Django</td>
<td>DefaultDevfileRegistry</td>
</tr>
</tbody>
</table>
You can modify your telemetry consent by using the odo preference command:

- `odo preference set ConsentTelemetry true` consents to telemetry.
- `odo preference unset ConsentTelemetry` disables telemetry.
- `odo preference view` shows the current preferences.

3.3. INSTALLING ODO

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

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

3.3.1. Installing odo on Linux

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

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

**Procedure**

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

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

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

2. Change the permissions on the binary:

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

```bash
$ echo $PATH
```

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

```bash
$ odo version
```

### 3.3.2. Installing odo on Windows

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

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

**Procedure**

1. Navigate to the content gateway and download the appropriate file:

   - If you download the binary, rename it to `odo.exe`.
   - If you download the archive, unzip the binary with a ZIP program and then rename it to `odo.exe`.

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

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

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

   ```bash
   C:\> odo version
   ```

### 3.3.3. Installing odo on macOS

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

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

**Procedure**

1. Navigate to the content gateway and download the appropriate file:
If you download the binary, rename it to `odo`:

```bash
```

If you download the tarball, extract the binary:

```bash
$ curl -L https://developers.redhat.com/content-gateway/rest/mirror/pub/openshift-v4/clients/odo/latest/odo-darwin-amd64.tar.gz -o odo.tar.gz
$ tar xvzf odo.tar.gz
```

2. Change the permissions on the binary:

```bash
# chmod +x odo
```

3. Place the `odo` binary in a directory that is on your `PATH`. To check your `PATH`, execute the following command:

```bash
$ echo $PATH
```

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

```bash
$ odo version
```

### 3.3.4. Installing odo on VS Code

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

**Prerequisites**

- You have installed VS Code.

**Procedure**

1. Open VS Code.

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

3. Enter the following command:

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

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

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

**Procedure**

1. Register with Red Hat Subscription Manager:

```bash
$ curl -L https://developers.redhat.com/content-gateway/rest/mirror/pub/openshift-v4/clients/odo/latest/odo-darwin-amd64.tar.gz -o odo.tar.gz
$ tar xvzf odo.tar.gz
```
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-4.10-for-rhel-8-x86_64-rpms"

6. Install the odo package:

   # yum install odo

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

   $ odo version

### 3.4. CREATING AND DEPLOYING APPLICATIONS WITH ODO

#### 3.4.1. Working with projects

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

##### 3.4.1.1. 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**
3.4.2. 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.

3.4.2.1. Creating a project

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

Procedure

1. Log in to an OpenShift Container Platform cluster:

   $ odo login -u developer -p developer

2. Create a project:

   $ odo project create myproject

   **Example output**

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

3.4.2.2. Creating a Node.js application with odo

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

Procedure

1. Create a directory for your components:

   $ mkdir my_components && cd my_components

2. Download the example Node.js application:

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

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

   $ cd <directory_name>
4. Add a component of the type Node.js to your application:

```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.

```bash
$ curl <url>
```

### 3.4.2.3. 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:

```bash
$ odo push
```

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

### 3.4.2.4. 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:
3.4.2.5. 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
   $ odo storage create <storage_name> --path=<path_to_the_directory> --size=<size>
   $ odo push
   $ odo storage list
   Example output
   The component 'nodejs' has the following storage attached:
   NAME   SIZE   PATH      STATE
   mystorage  1Gi   /data   Pushed
   $ odo storage delete <storage_name>
   $ odo storage list
   Example output
   The component 'nodejs' has the following storage attached:
   NAME   SIZE   PATH      STATE
   mystorage  1Gi   /data   Locally Deleted
   $ odo push
   ```
3.4.2.6. Connecting your application to multiple services using OpenShift Service Catalog

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

Prerequisites

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

Procedure

- To list the services:
  
  ```
  $ odo catalog list services
  ```

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

3.4.2.7. Deleting an application

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

Procedure

1. List the applications in the current project:

  ```
  $ odo app list
  ```

Example output

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

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

   $ odo component list

   **Example output**

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

3. Delete the application:

   $ odo app delete <application_name>

   **Example output**

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

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

### 3.4.3. 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.

**3.4.3.1. 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
3.4.3.2. 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:

   ```bash
   $ 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:

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

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

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

Example output

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>
<tr>
<td>golang</td>
<td>openshift</td>
<td>1.13.4-ubi7,1.13.4-ubi8,latest</td>
<td>NO</td>
</tr>
<tr>
<td>httpd</td>
<td>openshift</td>
<td>2.4-el7,2.4-el8,latest</td>
<td>NO</td>
</tr>
<tr>
<td>nginx</td>
<td>openshift</td>
<td>1.14-el7,1.14-el8,1.16-el7,1.16-el8,latest</td>
<td>NO</td>
</tr>
<tr>
<td>nodejs</td>
<td>openshift</td>
<td>10-ubi7,10-ubi8,12-ubi7,12-ubi8,latest</td>
<td>NO</td>
</tr>
<tr>
<td>perl</td>
<td>openshift</td>
<td>5.26-el7,5.26-el8,5.30-el7,latest</td>
<td>NO</td>
</tr>
<tr>
<td>php</td>
<td>openshift</td>
<td>7.2-ubi7,7.2-ubi8,7.3-ubi7,7.3-ubi8,latest</td>
<td>NO</td>
</tr>
<tr>
<td>python</td>
<td>openshift</td>
<td>2.7-ubi7,2.7-ubi8,3.6-ubi7,3.6-ubi8,3.8-ubi7,3.8-ubi8,latest</td>
<td>NO</td>
</tr>
<tr>
<td>ruby</td>
<td>openshift</td>
<td>2.5-ubi7,2.5-ubi8,2.6-ubi7,2.6-ubi8,2.7-ubi7,latest</td>
<td>NO</td>
</tr>
<tr>
<td>wildfly</td>
<td>openshift</td>
<td>10.0,10.1,11.0,12.0,13.0,14.0,15.0,16.0,17.0,18.0,19.0,20.0,8.1,9.0,latest</td>
<td>NO</td>
</tr>
</tbody>
</table>

4. Create a directory for your components:
5. Download the example back-end application:

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

6. Change to the back-end source directory:

   ```bash
   $ cd backend
   ```

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

   ```bash
   $ ls
   ```

   **Example output**

   ```
do.sh  pom.xml  src
   ```

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

   ```bash
   $ mvn package
   ```

   **Example output**

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

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

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

   **Example output**

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

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

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

    ```bash
    $ odo config view
    ```

    **Example output**
11. Push the component to the OpenShift Container Platform cluster.

```
$ odo push
```

**Example output**

- **Validation**
  - ✓ Checking component [6ms]

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

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

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

12. Validate:

- The status of the action in odo:

```
$ odo log -f
```

**Example output**

```
  (/deployments/wildwest-1.0.jar started by jboss in /deployments)
```

- The status of the back-end component:
3.4.3.3. 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:
   
   ```bash
   $ git clone https://github.com/openshift/nodejs-ex frontend
   ```

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

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

   Example output

<table>
<thead>
<tr>
<th>README.md</th>
<th>openshift</th>
<th>server.js</th>
<th>views</th>
<th>tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>helm</td>
<td>package.json</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   **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
3.4.3.4. Linking both components

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

Procedure

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

   $ odo list

   **Example output**

   OpenShift Components:
<table>
<thead>
<tr>
<th>APP</th>
<th>NAME</th>
<th>PROJECT</th>
<th>TYPE</th>
<th>SOURCETYPE</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>app</td>
<td>backend</td>
<td>testpro</td>
<td>openjdk18</td>
<td>binary</td>
<td>Pushed</td>
</tr>
<tr>
<td>app</td>
<td>frontend</td>
<td>testpro</td>
<td>nodejs</td>
<td>local</td>
<td>Pushed</td>
</tr>
</tbody>
</table>

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

   $ odo link backend --port 8080

   **Example output**

   ✓ Component backend has been successfully linked from the component frontend

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

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

3.4.3.5. Exposing components to the public

Procedure

1. Navigate to the **frontend** directory:
2. Create an external URL for the application:

```bash
$ cd frontend
$ 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-myproject.192.168.42.79.nip.io created
- Pushing to component frontend of type local: Checking file changes for pushing [1ms], No file changes detected, skipping build. Use the `-f` flag to force the build.

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

**NOTE**

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

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

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

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

Do not do this on a production cluster.

3.4.3.6. 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.

### 3.4.3.7. 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.

### 3.4.4. 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.

#### 3.4.4.1. 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
```

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

**Procedure**

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

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

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

   **Example output**

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

   **NOTE**

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

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

   **Example output**

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

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

   **Example output**

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

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

   **Example output**
3.4.4.3. Deploying a database in interactive mode

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

**Procedure**

- Run the interactive mode and answer the prompts:

  ```
  $ odo service create
  ```

**Example output**

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

**NOTE**

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

3.4.4.4. Deploying a database manually
1. List the available services:

   $ odo catalog list services

**Example output**

```
NAME                         PLANS
django-psql-persistent       default
jenkins-ephemeral            default
jenkins-pipeline-example     default
mariadb-persistent           default
mongodb-persistent           default
mysql-persistent             default
nodejs-mongo-persistent      default
postgresql-persistent        default
rails-pgsql-persistent       default
```

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

   $ odo catalog describe service mongodb-persistent

**Example output**

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

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

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

3.4.4.5. Connecting the database to the front-end application
1. Link the database to the front-end service:

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

**Example output**

- Service mongodb-persistent has been successfully linked from the component nodejs-nodejs-ex-mhbb
- Following environment variables were added to nodejs-nodejs-ex-mhbb component:
  - database_name
  - password
  - uri
  - username
  - admin_password

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

```
$ oc get pods
```

**Example output**

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

   b. Connect to the pod:

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

   c. Check the environment variables:

```
sh-4.2$ env
```

**Example output**

```
uri=mongodb://172.30.126.3:27017
password=dHIopYneSkX3rTLn
database_name=sampledb
username=user43U
admin_password=NCn41tqmx7rilqmfv
```

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
```
3.4.5. 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](#).

3.4.5.1. 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:

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

2. Create a project:

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

**Example output**

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

3.4.5.2. 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:

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

2. Navigate to the application directory:

   ```bash
   ```

   DB Connection Info:
   Type: MongoDB
   URL: mongodb://172.30.126.3:27017/sampledb
3. Initialize the project:

   $ cd ./application-stack-samples/jpa

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 Container Platform logs to the terminal:

   $ odo log

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

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

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

CHAPTER 3. DEVELOPER CLI (ODO)
6. Create an ingress URL to access the application:
   ```bash
   $ odo url create --port 8080
   ```

7. Push the changes to your cluster:
   ```bash
   $ odo push
   ```

8. Display the created URL:
   ```bash
   $ odo url list
   ```

   **Example output**

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

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

9. Use the URL to navigate to the `CreatePerson.xhtml` data entry page and enter a username and age by using the 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.

**3.4.5.3. 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:
   ```bash
   $ odo catalog list services
   ```

   **Example output**

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

2. Store the YAML of the service in a file:
   ```bash
   $ 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:
   ```
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
   serviceName: "sampledb"
   databasePassword: "samplepwd"
   databaseUser: "sampleuser"
   ```

5. Create a database from the YAML file:

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

   A database instance is now present in your project.

3.4.5.4. 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:

   ```bash
   $ odo service list
   ```

   **Example output**

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

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"'
   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.

### 3.4.6. Using devfiles in odo

#### 3.4.6.1. 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.

#### 3.4.6.2. Creating a Java application by using a devfile

**Prerequisites**

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

**NOTE**

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

#### 3.4.6.2.1. Creating a project

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

**Procedure**

1. Log in to an OpenShift Container Platform cluster:

   $ odo login -u developer -p developer

2. Create a project:

   $ odo project create myproject

**Example output**
3.4.6.2.2. Listing available devfile components

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

Procedure

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

   $ odo catalog list components

   The output lists the available odo components:

<p>|
| Odo Devfile Components: |</p>
<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>

<p>|
| Odo OpenShift Components: |</p>
<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>
<tr>
<td>golang</td>
<td>openshift</td>
<td>1.13.4-ubi7,1.13.4-ubi8,latest</td>
<td>NO</td>
</tr>
<tr>
<td>httpd</td>
<td>openshift</td>
<td>2.4-el7,2.4-el8,latest</td>
<td>NO</td>
</tr>
<tr>
<td>nginx</td>
<td>openshift</td>
<td>1.14-el7,1.14-el8,1.16-el7,1.16-el8,latest</td>
<td>NO</td>
</tr>
<tr>
<td>nodejs</td>
<td>openshift</td>
<td>10-ubi7,10-ubi8,12-ubi7,12-ubi8,latest</td>
<td>NO</td>
</tr>
<tr>
<td>perl</td>
<td>openshift</td>
<td>5.26-el7,5.26-ubi8,5.30-el7,latest</td>
<td>NO</td>
</tr>
<tr>
<td>php</td>
<td>openshift</td>
<td>7.2-ubi7,7.2-ubi8,7.3-ubi7,7.3-ubi8,latest</td>
<td>NO</td>
</tr>
<tr>
<td>python</td>
<td>openshift</td>
<td>2.7-ubi7,2.7-ubi8,3.6-ubi7,3.6-ubi8,3.8-ubi7,3.8-ubi8,latest</td>
<td>NO</td>
</tr>
<tr>
<td>ruby</td>
<td>openshift</td>
<td>2.5-ubi7,2.5-ubi8,2.6-ubi7,2.6-ubi8,2.7-ubi7,latest</td>
<td>NO</td>
</tr>
<tr>
<td>wildfly</td>
<td>openshift</td>
<td>10.0,10.1,11.0,12.0,13.0,14.0,15.0,16.0,17.0,18.0,19.0,20.0,8.1,9.0,latest</td>
<td>NO</td>
</tr>
</tbody>
</table>

3.4.6.2.3. Deploying a Java application using a devfile

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

Procedure

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

   $ mkdir <directory-name>

2. Create a component configuration of Spring Boot component type named myspring and download its sample project:
$ odo create java-springboot myspring --starter

The previous command produces the following output:

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

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

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

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

$ ls

The previous command produces the following output:

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

4. Create a URL to access the deployed component:

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

The previous command produces the following output:

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

To apply the URL configuration changes, please use odo push

**NOTE**

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

5. Push the component to the cluster:

$ odo push

The previous command produces the following output:

Validation
✓ Validating the devfile [81808ns]

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

Applying URL changes
✓ URL myspring-8080: http://myspring-8080.apps-crc.testing created
Syncing to component myspring
✓ Checking files for pushing [2ms]
✓ Syncing files to the component [1s]

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

Pushing devfile component myspring
✓ Changes successfully pushed to component

6. List the URLs of the component to verify that the component was pushed successfully:
   $ odo url list

The previous command produces the following output:

   Found the following URLs for component myspring
   NAME              URL                                       PORT     SECURE
   myspring-8080     http://myspring-8080.apps-crc.testing     8080     false

7. View your deployed application by using the generated URL:
   $ curl http://myspring-8080.apps-crc.testing

3.4.6.3. 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:
3.4.7. Working with storage

Persistent storage keeps data available between restarts of odo.

3.4.7.1. Adding storage to the application components

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

Procedure

1. Add the storage to your component:

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

2. Push the storage to the cluster:

   $ odo push

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

   $ odo storage list

   Example output

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

4. Delete the storage from your component:

   $ odo storage delete <storage_name>

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

   $ odo storage list

   Example output

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

6. Push the changes to the cluster:

   $ odo push

3.4.7.2. Adding storage to a specific container

$ odo delete --s2i
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
   
   The **runtime** container.
   
   The **funtime** container.

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

   ```bash
   $ odo storage create store --path /data --size 1Gi --container runtime
   
   ✓ Added storage store to nodejs-testing-xnfg
   Please use `odo push` command to make the storage accessible to the component
   
   $ odo storage list
   
   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>
   
   $ odo push
   
   3.4.7.3. Switching between ephemeral and persistent storage
You can switch between ephemeral and persistent storage in your project by using the `odo preference` command. `odo preference` modifies the global preference in your cluster.

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

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

Ephemeral storage is enabled by default.

**Procedure**

1. See the preference currently set in your project:

   ```
   $ odo preference view
   ```

   **Example output**

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

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

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

3. To set the ephemeral storage again:

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

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

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

   ```
   $ odo push
   ```

**Additional resources**

- [Understanding ephemeral storage](#)
- [Understanding persistent storage](#)

**3.4.8. Deleting applications**

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

**3.4.8.1. Deleting an application**
Use the `odo app delete` command to delete your application.

**Procedure**

1. List the applications in the current project:

   ```
   $ odo app list
   ```

   **Example output**

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

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

   ```
   $ odo component list
   ```

   **Example output**

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

3. Delete the application:

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

   **Example output**

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

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

**3.4.9. 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).

**3.4.9.1. Debugging an application**

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

**Procedure**

1. Download the sample application that contains the necessary `debugrun` step within its devfile:
$ odo create nodejs --starter

Example output

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

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

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

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

$ odo push --debug

Example output

Validation
✓ Validating the devfile [29916ns]

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

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

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

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

Pushing devfile component nodejs
✓ Changes successfully pushed to component

NOTE

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

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

$ odo debug port-forward

Example output

-
3.4.9.2. 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.

3.4.10. Sample applications

`odo` offers partial compatibility with any language or runtime listed within the OpenShift Container Platform 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>
</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**.

### 3.4.10.1. Git repository example applications

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

#### 3.4.10.1.1. 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.

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

#### 3.4.10.1.2. 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.

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

#### 3.4.10.1.3. 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.

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

#### 3.4.10.1.4. 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.

```bash
$ odo create perl --git https://github.com/openshift/dancer-ex.git
```
3.4.10.1.5. 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

3.4.10.1.6. 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

3.4.10.1.7. 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

3.4.10.2. Binary example applications

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

3.4.10.2.1. java

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

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

3.5. USING ODO IN A RESTRICTED ENVIRONMENT

3.5.1. 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

3.5.2. 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.

3.5.2.1. 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.

3.5.2.2. 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:

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

Procedure

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

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

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

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

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

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

4. Mirror the odo init image:

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

5. Override the default odo init image path by setting the ODO_BOOTSTRAPPER_IMAGE environment variable:
3.5.2.2.2. 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:
   ```bash
   $ 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:
      ```bash
      $ docker login <mirror-registry>:5000 -u <username> -p <password>
      ```

3. Mirror the `odo` init image:
   ```bash
   $ 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:
   ```bash
   $ export ODO_BOOTSTRAPPER_IMAGE=<mirror-registry>:5000/openshiftdo/odo-init-image-rhel7:<tag>
   ```

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

Procedure

1. Use `base64` to encode the root certification authority (CA) content of your mirror registry:
   ```bash
   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:
   ```bash
   PS C:\WINDOWS\system32> certutil -addstore -f "ROOT" disconnect-ca.crt
   ```

3. Trust a CA in your client platform and log in to the OpenShift Container Platform mirror registry:
   
   a. Restart Docker using the Docker UI.
   
   b. Run the following command:
      ```bash
      PS C:\WINDOWS\system32> docker login <mirror-registry>:5000 -u <username> -p <password>
      ```
4. Mirror the odo init image:

```bash
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:

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

3.5.2.3. 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:

3.5.2.3.1. Pushing the init image directly on Linux

**Procedure**

1. Enable the default route:

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

2. Get a wildcard route CA:

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

5. Log in to the internal registry:

   ```bash
   $ oc get route -n openshift-image-registry
   ```
6. Push the `odo` init image:

```bash
$ docker login <registry_path> -u kubeadm -p $(oc whoami -t)
$ 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
```

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

#### Procedure

1. Enable the default route:

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

2. Get a wildcard route CA:

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

#### Example output

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

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 in to the internal registry:

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

$ docker login <registry_path> -u kubeadmin -p $(oc whoami -t)

6. Push the odo init image:

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

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

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

3.5.2.3.3. Pushing the init image directly on Windows

**Procedure**

1. Enable the default route:

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

2. Get a wildcard route CA:

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

   **Example output**

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

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

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

4. As an administrator, trust a CA in your client platform by executing the following command:
5. Log in to the internal registry:

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

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

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

6. Push the `odo` init image:

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

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

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

### 3.5.3. 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.

#### 3.5.3.1. 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.

#### 3.5.3.2. 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
3.5.3.3. 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>
  $ odo component create nodejs --env NPM_MIRROR=<npm_mirror_registry>
  ```

- To overwrite the mirror registry when creating a component:

  ```
  $ odo create nodejs
  ```

3.5.3.4. Creating a Node.js application with odo

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

Procedure

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

  ```
  $ cd <directory_name>
  ```

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

  ```
  $ odo create nodejs
  ```
3. Push the initial source code to the component:

   $ odo push

   Your component is now deployed to OpenShift Container Platform.

4. Create a URL and add an entry in the local configuration file as follows:

   $ odo url create --port 8080

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

   $ odo push

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

   $ odo url list

7. View your deployed application using the generated URL.

   $ curl <url>

### 3.5.4. Creating and deploying devfile components to the disconnected cluster

#### 3.5.4.1. Creating a NodeJS application by using a devfile in a disconnected cluster

**WARNING**

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

**Prerequisites**

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

**Procedure**

1. Define your NodeJS application in a devfile:
Example of a devfile

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

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

Example output

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

3. Create a URL to access your application and push it to 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]

Pushing devfile component nodejs
✓ Changes successfully pushed to component

4. Add the storage to your application

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

Example output

✓ Added storage abcde to nodejs

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

5. Push the changes to the cluster:

$ odo push

3.5.4.2. 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

```yaml
schemaVersion: 2.0.0
metadata:
  name: java-maven
  version: 1.1.0
starterProjects:
  - name: springbootproject
git:
    remotes:
      origin: "https://github.com/odo-devfiles/springboot-ex.git"
components:
  - name: tools
    container:
      image: quay.io/eclipse/che-java11-maven:nightly
      memoryLimit: 512Mi
      mountSources: true
      endpoints:
        - name: 'http-8080'
          targetPort: 8080
      volumeMounts:
        - name: m2
          path: /home/user/.m2
      - name: m2
        volume: {}
    commands:
      - id: mvn-package
        exec:
          component: tools
          commandLine: "mvn -Dmaven.repo.local=/home/user/.m2/repository -Dhttp.proxyHost=<proxy-host> -Dhttps.proxyHost=<proxy-host> -Dhttp.proxyPort=<proxy-port> -Dhttps.proxyPort=<proxy-port> package"
```

GROUP:
2. Create a Java application:

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

**Example output**

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

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

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

3. Push the changes to the cluster:

```
$ odo push
```

**Example output**

```

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] ------------------------------------------------------------------------
```
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
3.6. 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.

3.6.1. Prerequisites

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

- Install the `odo` CLI.

3.6.2. Creating a project

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

Procedure

1. Log in to an OpenShift Container Platform cluster:

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

2. Create a project:
3.6.3. Listing available services from the Operators installed on the cluster

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

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

  ```
  $ odo catalog list services
  ```

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

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

3.6.4. Creating a service from an Operator

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

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

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

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

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

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

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

4. Verify that a service has started:
3.6.5. 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

3.7. 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.

3.7.1. Setting and unsetting environment variables
Procedure

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

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

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

3.8. CONFIGURING THE ODO CLI

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

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

3.8.1. Viewing the current configuration

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

```bash
$ odo preference view
```

Example output

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

3.8.2. Setting a value

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

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

**NOTE**

Preference keys are case-insensitive.
3.8.3. Setting a value

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

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

**NOTE**

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

### Example command

```bash
$ odo preference unset updatenotification
? Do you want to unset updatenotification in the preference (y/N) y
```

**Example output**

Global preference was successfully updated

3.8.4. Preference key table

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

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

<table>
<thead>
<tr>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConsentTelemetry</td>
<td>Controls whether <code>odo</code> can collect telemetry for the user’s <code>odo</code> usage.</td>
</tr>
</tbody>
</table>

### 3.8.5. Ignoring files or patterns

You can configure a list of files or patterns to ignore by modifying the `.odoignore` file in the root directory of your application. This applies to both `odo push` and `odo watch`.

If the `.odoignore` file does not exist, the `.gitignore` file is used instead for ignoring specific files and folders.

To ignore `.git` files, any files with the `.js` extension, and the folder `tests`, add the following to either the `.odoignore` or the `.gitignore` file:

```
.git
*.js
tests/
```

The `.odoignore` file allows any glob expressions.

### 3.9. ODO CLI REFERENCE

#### 3.9.1. odo build-images

`odo` can build container images based on Dockerfiles, and push these images to their registries.

When running the `odo build-images` command, `odo` searches for all components in the `devfile.yaml` with the `image` type, for example:

```
components:
- image:
  imageName: quay.io/myusername/myimage
dockerfile:
  uri: ./Dockerfile
buildContext: ${PROJECTS_ROOT}
name: component-built-from-dockerfile
```

The `uri` field indicates the relative path of the Dockerfile to use, relative to the directory containing the `devfile.yaml`. The devfile specification indicates that `uri` could also be an HTTP URL, but this case is not supported by odo yet. The `buildContext` indicates the directory used as build context. The default value is `${PROJECTS_ROOT}`.

For each image component, odo executes either `podman` or `docker` (the first one found, in this order), to build the image with the specified Dockerfile, build context, and arguments.

If the `--push` flag is passed to the command, the images are pushed to their registries after they are built.

#### 3.9.2. odo catalog
odo uses different catalogs to deploy components and services.

3.9.2.1. Components

odo uses the portable devfile format to describe the components. It can connect to various devfile registries to download devfiles for different languages and frameworks. See odo registry for more information.

3.9.2.1.1. Listing components

To list all the devfiles available on the different registries, run the command:

```
$ odo catalog list components
```

Example output

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
<th>REGISTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>go</td>
<td>Stack with the latest Go version</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>java-maven</td>
<td>Upstream Maven and OpenJDK 11</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>nodejs</td>
<td>Stack with Node.js 14</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>php-laravel</td>
<td>Stack with Laravel 8</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>python</td>
<td>Python Stack with Python 3.7</td>
<td>DefaultDevfileRegistry</td>
</tr>
<tr>
<td>[...]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.9.2.1.2. Getting information about a component

To get more information about a specific component, run the command:

```
$ odo catalog describe component
```

For example, run the command:

```
$ odo catalog describe component nodejs
```

Example output

* Registry: DefaultDevfileRegistry <.>

Starter Projects: <.>
```
---
name: nodejs-starter
attributes: {}
description: ""
subdir: ""
projectsource:
sourcetype: ""
git:
gitlikeprojectsourcet: 
commonprojectsourcet: {}
checkoutfrom: null
remotes:
```
< Registry is the registry from which the devfile is retrieved. Starter projects are sample projects in the same language and framework of the devfile, that can help you start a new project.

See odo create for more information on creating a project from a starter project.

3.9.2.2. Services

odo can deploy services with the help of Operators.

Only Operators deployed with the help of the Operator Lifecycle Manager are supported by odo.

3.9.2.2.1. Listing services

To list the available Operators and their associated services, run the command:

Example output

```
$ odo catalog list services
```

Services available through Operators

<table>
<thead>
<tr>
<th>NAME</th>
<th>CRDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>postgresql-operator.v0.1.1</td>
<td>Backup, Database</td>
</tr>
<tr>
<td>redis-operator.v0.8.0</td>
<td>RedisCluster, Redis</td>
</tr>
</tbody>
</table>

In this example, two Operators are installed in the cluster. The postgresql-operator.v0.1.1 Operator deploys services related to PostgreSQL: Backup and Database. The redis-operator.v0.8.0 Operator deploys services related to Redis: RedisCluster and Redis.

NOTE

To get a list of all the available Operators, odo fetches the ClusterServiceVersion (CSV) resources of the current namespace that are in a Succeeded phase. For Operators that support cluster-wide access, when a new namespace is created, these resources are automatically added to it. However, it may take some time before they are in the Succeeded phase, and odo may return an empty list until the resources are ready.

3.9.2.2.2. Searching services

To search for a specific service by a keyword, run the command:

Example output

```
$ odo catalog search service postgres
```

For example, to retrieve the PostgreSQL services, run the command:

```
$ odo catalog search service postgres
```
You will see a list of Operators that contain the searched keyword in their name.

3.9.2.2.3. Getting information about a service

To get more information about a specific service, run the command:

```
$ odo catalog describe service
```

For example:

```
$ odo catalog describe service postgresql-operator.v0.1.1/Database
```

**Example output**

```
KIND:    Database
VERSION: v1alpha1

DESCRIPTION:
Database is the Schema for the the Database Database API

FIELDS:
awsAccessKeyId (string)
    AWS S3 accessKey/token ID

    Key ID of AWS S3 storage. Default Value: nil Required to create the Secret
    with the data to allow send the backup files to AWS S3 storage.

[...]
```

A service is represented in the cluster by a CustomResourceDefinition (CRD) resource. The previous command displays the details about the CRD such as `kind`, `version`, and the list of fields available to define an instance of this custom resource.

The list of fields is extracted from the OpenAPI schema included in the CRD. This information is optional in a CRD, and if it is not present, it is extracted from the ClusterServiceVersion (CSV) resource representing the service instead.

It is also possible to request the description of an Operator-backed service, without providing CRD type information. To describe the Redis Operator on a cluster, without CRD, run the following command:

```
$ odo catalog describe service redis-operator.v0.8.0
```

**Example output**

```
NAME: redis-operator.v0.8.0
DESCRIPTION:
A Golang based redis operator that will make/oversee Redis
standalone/cluster mode setup on top of the Kubernetes. It can create a
redis cluster setup with best practices on Cloud as well as the Bare metal
```
odo uses a devfile to store the configuration of a component and to describe the component’s resources such as storage and services. The odo create command generates this file.

3.9.3.1. Creating a component

To create a devfile for an existing project, run the odo create command with the name and type of your component (for example, nodejs or go):

```
odo create nodejs mynodejs
```

In the example, nodejs is the type of the component and mynodejs is the name of the component that odo creates for you.

```
NOTE
For a list of all the supported component types, run the command odo catalog list components.
```

If your source code exists outside the current directory, the --context flag can be used to specify the path. For example, if the source for the nodejs component is in a folder called node-backend relative to the current working directory, run the command:

```
odo create nodejs mynodejs --context ./node-backend
```

The --context flag supports relative and absolute paths.

To specify the project or app where your component will be deployed, use the --project and --app flags. For example, to create a component that is part of the myapp app inside the backend project, run the command:

```
odo create nodejs --app myapp --project backend
```

```
NOTE
If these flags are not specified, they will default to the active app and project.
```

3.9.3.2. Starter projects
Use the starter projects if you do not have existing source code but want to get up and running quickly to experiment with devfiles and components. To use a starter project, add the `--starter` flag to the `odo create` command.

To get a list of available starter projects for a component type, run the `odo catalog describe component` command. For example, to get all available starter projects for the nodejs component type, run the command:

```
odo catalog describe component nodejs
```

Then specify the desired project using the `--starter` flag on the `odo create` command:

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

This will download the example template corresponding to the chosen component type, in this instance, `nodejs`. The template is downloaded to your current directory, or to the location specified by the `--context` flag. If a starter project has its own devfile, then this devfile will be preserved.

### 3.9.3.3. Using an existing devfile

If you want to create a new component from an existing devfile, you can do so by specifying the path to the devfile using the `--devfile` flag. For example, to create a component called `mynodejs`, based on a devfile from GitHub, use the following command:

```
odo create mynodejs --devfile https://raw.githubusercontent.com/odo-devfiles/registry/master/devfiles/nodejs/devfile.yaml
```

### 3.9.3.4. Interactive creation

You can also run the `odo create` command interactively, to guide you through the steps needed to create a component:

```
$ odo create

? Which devfile component type do you wish to create go
? What do you wish to name the new devfile component go-api
? What project do you want the devfile component to be created in default
Devfile Object Validation
✓ Checking devfile existence [164258ns]
✓ Creating a devfile component from registry: DefaultDevfileRegistry [246051ns]
Validation
✓ Validating if devfile name is correct [92255ns]
? Do you want to download a starter project Yes

Starter Project
✓ Downloading starter project go-starter from https://github.com/devfile-samples/devfile-stack-go.git [429ms]

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

You are prompted to choose the component type, name, and the project for the component. You can also choose whether or not to download a starter project. Once finished, a new `devfile.yaml` file is created in the working directory.
To deploy these resources to your cluster, run the command `odo push`.

3.9.4. odo delete

The `odo delete` command is useful for deleting resources that are managed by `odo`.

3.9.4.1. Deleting a component

To delete a `devfile` component, run the `odo delete` command:

```
$ odo delete
```

If the component has been pushed to the cluster, the component is deleted from the cluster, along with its dependent storage, URL, secrets, and other resources. If the component has not been pushed, the command exits with an error stating that it could not find the resources on the cluster.

Use the `-f` or `--force` flag to avoid the confirmation questions.

3.9.4.2. Undeploying devfile Kubernetes components

To undeploy the devfile Kubernetes components, that have been deployed with `odo deploy`, execute the `odo delete` command with the `--deploy` flag:

```
$ odo delete --deploy
```

Use the `-f` or `--force` flag to avoid the confirmation questions.

3.9.4.3. Delete all

To delete all artifacts including the following items, run the `odo delete` command with the `--all` flag:

- `devfile` component
- Devfile Kubernetes component that was deployed using the `odo deploy` command
- Devfile
- Local configuration

```
$ odo delete --all
```

3.9.4.4. Available flags

- `-f`, `--force`
  
  Use this flag to avoid the confirmation questions.

- `-w`, `--wait`
  
  Use this flag to wait for component deletion and any dependencies. This flag does not work when undeploying.

The documentation on `Common Flags` provides more information on the flags available for commands.

3.9.5. odo deploy
**odo** can be used to deploy components in a manner similar to how they would be deployed using a CI/CD system. First, **odo** builds the container images, and then it deploys the Kubernetes resources required to deploy the components.

When running the command **odo deploy**, **odo** searches for the default command of kind **deploy** in the devfile, and executes this command. The kind **deploy** is supported by the devfile format starting from version 2.2.0.

The **deploy** command is typically a *composite* command, composed of several **apply** commands:

- A command referencing an **image** component that, when applied, will build the image of the container to deploy, and then push it to its registry.

- A command referencing a **Kubernetes component** that, when applied, will create a Kubernetes resource in the cluster.

With the following example **devfile.yaml** file, a container image is built using the **Dockerfile** present in the directory. The image is pushed to its registry and then a Kubernetes Deployment resource is created in the cluster, using this freshly built image.

```yaml
schemaVersion: 2.2.0

variables:
  CONTAINER_IMAGE: quay.io/phmartin/myimage

commands:
  - id: build-image
    apply:
      component: outerloop-build
  - id: deployk8s
    apply:
      component: outerloop-deploy
  - id: deploy
    composite:
      commands:
        - build-image
        - deployk8s
      group:
        kind: deploy
        isDefault: true

components:
  - name: outerloop-build
    image:
      imageName: "{{CONTAINER_IMAGE}}"
    dockerfile:
      uri: ./Dockerfile
      buildContext: ${PROJECTS_ROOT}
  - name: outerloop-deploy
    kubernetes:
      inlined: |
      kind: Deployment
      apiVersion: apps/v1
      metadata:
        name: my-component
      spec:
        replicas: 1
        selector:
```

---

OpenShift Container Platform 4.10 CLI tools

162
3.9.6. odo link

The odo link command helps link an odo component to an Operator-backed service or another odo component. It does this by using the Service Binding Operator. Currently, odo makes use of the Service Binding library and not the Operator itself to achieve the desired functionality.

3.9.6.1. Various linking options

odo provides various options for linking a component with an Operator-backed service or another odo component. All these options (or flags) can be used whether you are linking a component to a service or to another component.

3.9.6.1.1. Default behavior

By default, the odo link command creates a directory named kubernetes/ in your component directory and stores the information (YAML manifests) about services and links there. When you use odo push, odo compares these manifests with the state of the resources on the Kubernetes cluster and decides whether it needs to create, modify or destroy resources to match what is specified by the user.

3.9.6.1.2. The --inlined flag

If you specify the --inlined flag to the odo link command, odo stores the link information inline in the devfile.yaml in the component directory, instead of creating a file under the kubernetes/ directory. The behavior of the --inlined flag is similar in both the odo link and odo service create commands. This flag is helpful if you want everything stored in a single devfile.yaml. You have to remember to use --inlined flag with each odo link and odo service create command that you execute for the component.

3.9.6.1.3. The --map flag

Sometimes, you might want to add more binding information to the component, in addition to what is available by default. For example, if you are linking the component with a service and would like to bind some information from the service’s spec (short for specification), you could use the --map flag. Note that odo does not do any validation against the spec of the service or component being linked. Using this flag is only recommended if you are comfortable using the Kubernetes YAML manifests.

3.9.6.1.4. The --bind-as-files flag

For all the linking options discussed so far, odo injects the binding information into the component as environment variables. If you would like to mount this information as files instead, you can use the --bind-as-files flag. This will make odo inject the binding information as files into the /bindings location within your component’s Pod. Compared to the environment variables scenario, when you use --bind-as-files, the files are named after the keys and the value of these keys is stored as the contents of these files.
3.9.6.2. Examples

3.9.6.2.1. Default odo link

In the following example, the backend component is linked with the PostgreSQL service using the default odo link command. For the backend component, make sure that your component and service are pushed to the cluster:

$ odo list

Sample output

<table>
<thead>
<tr>
<th>APP</th>
<th>NAME</th>
<th>PROJECT</th>
<th>TYPE</th>
<th>STATE</th>
<th>MANAGED BY ODO</th>
</tr>
</thead>
<tbody>
<tr>
<td>app</td>
<td>backend</td>
<td>myproject</td>
<td>spring</td>
<td>Pushed</td>
<td>Yes</td>
</tr>
</tbody>
</table>

$ odo service list

Sample output

<table>
<thead>
<tr>
<th>NAME</th>
<th>MANAGED BY ODO</th>
<th>STATE</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgresCluster/hippo</td>
<td>Yes (backend)</td>
<td>Pushed</td>
<td>59m41s</td>
</tr>
</tbody>
</table>

Now, run odo link to link the backend component with the PostgreSQL service:

$ odo link PostgresCluster/hippo

Example output

✓ Successfully created link between component "backend" and service "PostgresCluster/hippo"

To apply the link, please use `odo push`

And then run odo push to actually create the link on the Kubernetes cluster.

After a successful odo push, you will see a few outcomes:

1. When you open the URL for the application deployed by backend component, it shows a list of todo items in the database. For example, in the output for the odo url list command, the path where todos are listed is included:

   $ odo url list

Sample output

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>URL</th>
<th>PORT</th>
<th>SECURE</th>
<th>KIND</th>
</tr>
</thead>
</table>

The correct path for the URL would be http://8080-tcp.192.168.39.112.nip.io/api/v1/todos. The exact URL depends on your setup. Also note that there are no todos in the database unless you add some, so the URL might just show an empty JSON object.
2. You can see binding information related to the Postgres service injected into the backend component. This binding information is injected, by default, as environment variables. You can check it using the `odo describe` command from the backend component’s directory:

```sh
$ odo describe
```

**Example output:**

Component Name: backend  
Type: spring  
Environment Variables:  
· PROJECTS_ROOT=/projects  
· PROJECT_SOURCE=/projects  
· DEBUG_PORT=5858  
Storage:  
· m2 of size 3Gi mounted to /home/user/.m2  
URLs:  
· http://8080-tcp.192.168.39.112.nip.io exposed via 8080  
Linked Services:  
· PostgresCluster/hippo  
  Environment Variables:  
· POSTGRESCLUSTER_PGBOUNCER-EMPTY  
· POSTGRESCLUSTER_PGBOUNCER.INI  
· POSTGRESCLUSTER_ROOT.CRT  
· POSTGRESCLUSTER_VERIFIER  
· POSTGRESCLUSTER_ID_ECDSA  
· POSTGRESCLUSTER_PGBOUNCER-VERIFIER  
· POSTGRESCLUSTER_TLS.CRT  
· POSTGRESCLUSTER_PGBOUNCER-URI  
· POSTGRESCLUSTER_PATRONI.CRT-COMBINED  
· POSTGRESCLUSTER_USER  
· pgImage  
· pgVersion  
· POSTGRESCLUSTER_CLUSTERIP  
· POSTGRESCLUSTER_HOST  
· POSTGRESCLUSTER_PGBACKREST_REPO.CONF  
· POSTGRESCLUSTER_PGBOUNCER-USERS.TXT  
· POSTGRESCLUSTER_TLS.KEY  
· POSTGRESCLUSTER_CONFIG-HASH  
· POSTGRESCLUSTER_PASSWORD  
· POSTGRESCLUSTER_PATRONI.CA-ROOTS  
· POSTGRESCLUSTER_DBNAME  
· POSTGRESCLUSTER_PGBOUNCER-PASSWORD  
· POSTGRESCLUSTER_SSHD_CONFIG  
· POSTGRESCLUSTER_PGBOUNCER-FRONTEND.KEY  
· POSTGRESCLUSTER_PGBACKREST_INSTANCE.CONF  
· POSTGRESCLUSTER_PGBOUNCER-FRONTEND.CA-ROOTS  
· POSTGRESCLUSTER_PGBOUNCER-ROOT  
· POSTGRESCLUSTER_PGBOUNCER-FRONTEND.KEY  
· POSTGRESCLUSTER_SSH_KNOWN_HOSTS  
· POSTGRESCLUSTER_PGBOUNCE-HOST  
· POSTGRESCLUSTER_PGBOUNCE-URI  
· POSTGRESCLUSTER_PATRONI.YAML  
· POSTGRESCLUSTER_DNS.CRT
Some of these variables are used in the backend component’s `src/main/resources/application.properties` file so that the Java Spring Boot application can connect to the PostgreSQL database service.

3. Lastly, odo has created a directory called `kubernetes/` in your backend component’s directory that contains the following files:

```
$ ls kubernetes
odo-service-backend-postgrescluster-hippo.yaml  odo-service-hippo.yaml
```

These files contain the information (YAML manifests) for two resources:

a. `odo-service-hippo.yaml` - the Postgres service created using `odo service create --from-file ../postgrescluster.yaml` command.

b. `odo-service-backend-postgrescluster-hippo.yaml` - the link created using `odo link` command.

### 3.9.6.2.2. Using odo link with the `--inlined` flag

Using the `--inlined` flag with the `odo link` command has the same effect as an `odo link` command without the flag, in that it injects binding information. However, the subtle difference is that in the above case, there are two manifest files under `kubernetes/` directory, one for the Postgres service and another for the link between the backend component and this service. However, when you pass the `--inlined` flag, odo does not create a file under the `kubernetes/` directory to store the YAML manifest, but rather stores it inline in the `devfile.yaml` file.

To see this, unlink the component from the PostgreSQL service first:

```
$ odo unlink PostgresCluster/hippo
```

Example output:

```
✓ Successfully unlinked component "backend" from service "PostgresCluster/hippo"
To apply the changes, please use `odo push`
```

To unlink them on the cluster, run `odo push`. Now if you inspect the `kubernetes/` directory, you see only one file:

```
$ ls kubernetes
odo-service-hippo.yaml
```

Next, use the `--inlined` flag to create a link:

```
$ odo link PostgresCluster/hippo --inlined
```
Example output:

✓ Successfully created link between component "backend" and service "PostgresCluster/hippo"

To apply the link, please use `odo push`

You need to run **odo push** for the link to get created on the cluster, like the procedure that omits the **--inlined** flag. **odo** stores the configuration in **devfile.yaml**. In this file, you can see an entry like the following:

```yaml
kubernetes:
  inlined: |
    apiVersion: binding.operators.coreos.com/v1alpha1
    kind: ServiceBinding
    metadata:
      creationTimestamp: null
      name: backend-postgrescluster-hippo
    spec:
      application:
        group: apps
        name: backend-app
        resource: deployments
        version: v1
      bindAsFiles: false
      detectBindingResources: true
      services: |
        - group: postgres-operator.crunchydata.com
          id: hippo
          kind: PostgresCluster
          name: hippo
          version: v1beta1
      status:
        secret: ""
        name: backend-postgrescluster-hippo
```

Now if you were to run **odo unlink PostgresCluster/hippo**, **odo** would first remove the link information from the **devfile.yaml**, and then a subsequent **odo push** would delete the link from the cluster.

### 3.9.6.2.3. Custom bindings

**odo link** accepts the flag **--map** which can inject custom binding information into the component. Such binding information will be fetched from the manifest of the resource that you are linking to your component. For example, in the context of the backend component and PostgreSQL service, you can inject information from the PostgreSQL service’s manifest **postgrescluster.yaml** file into the backend component.

If the name of your **PostgresCluster** service is **hippo** (or the output of **odo service list**, if your PostgresCluster service is named differently), when you want to inject the value of **postgresVersion** from that YAML definition into your backend component, run the command:

```
$ odo link PostgresCluster/hippo --map pgVersion='{{ .hippo.spec.postgresVersion }}'
```

Note that, if the name of your Postgres service is different from **hippo**, you will have to specify that in the above command in the place of **.hippo** in the value for **pgVersion**.
After a link operation, run `odo push` as usual. Upon successful completion of the push operation, you can run the following command from your backend component directory, to validate if the custom mapping got injected properly:

```bash
$ odo exec -- env | grep pgVersion
```

Example output:

```
pgVersion=13
```

Since you might want to inject more than just one piece of custom binding information, `odo link` accepts multiple key-value pairs of mappings. The only constraint is that these should be specified as `--map <key>=<value>`. For example, if you want to also inject PostgreSQL image information along with the version, you could run:

```bash
$ odo link PostgresCluster/hippo --map pgVersion='{{ .hippo.spec.postgresVersion }}' --map pgImage='{{ .hippo.spec.image }}'
```

and then run `odo push`. To validate if both the mappings got injected correctly, run the following command:

```bash
$ odo exec -- env | grep -e "pgVersion\|pgImage"
```

Example output:

```
pgVersion=13
pgImage=registry.developers.crunchydata.com/crunchydata/crunchy-postgres-ha:centos8-13.4-0
```

### 3.9.6.2.3.1. To inline or not?

You can accept the default behavior where `odo link` generate a manifests file for the link under `kubernetes/` directory. Alternatively, you can use the `--inlined` flag if you prefer to store everything in a single `devfile.yaml` file.

### 3.9.6.3. Binding as files

Another helpful flag that `odo link` provides is `--bind-as-files`. When this flag is passed, the binding information is not injected into the component’s Pod as environment variables but is mounted as a filesystem.

Ensure that there are no existing links between the backend component and the PostgreSQL service. You could do this by running `odo describe` in the backend component’s directory and check if you see output similar to the following:

```
Linked Services:
  · PostgresCluster/hippo
```

Unlink the service from the component using:

```bash
$ odo unlink PostgresCluster/hippo
$ odo push
```
3.9.6.4. --bind-as-files examples

3.9.6.4.1. Using the default odo link

By default, `odo` creates the manifest file under the `kubernetes/` directory, for storing the link information. Link the backend component and PostgreSQL service using:

```
$ odo link PostgresCluster/hippo --bind-as-files
$ odo push
```

**Example odo describe output:**

```
$ odo describe

Component Name: backend
Type: spring
Environment Variables:
  · PROJECTS_ROOT=/projects
  · PROJECT_SOURCE=/projects
  · DEBUG_PORT=5858
  · SERVICE_BINDING_ROOT=/bindings
  · SERVICE_BINDING_ROOT=/bindings
Storage:
  · m2 of size 3Gi mounted to /home/user/.m2
URLs:
  · http://8080-tcp.192.168.39.112.nip.io exposed via 8080
Linked Services:
  · PostgresCluster/hippo
Files:
  · /bindings/backend-postgrescluster-hippo/pgbackrest_instance.conf
  · /bindings/backend-postgrescluster-hippo/user
  · /bindings/backend-postgrescluster-hippo/ssh_known_hosts
  · /bindings/backend-postgrescluster-hippo/clusterIP
  · /bindings/backend-postgrescluster-hippo/password
  · /bindings/backend-postgrescluster-hippo/patroni.yaml
  · /bindings/backend-postgrescluster-hippo/pgbouncer-frontend.crt
  · /bindings/backend-postgrescluster-hippo/pgbouncer-host
  · /bindings/backend-postgrescluster-hippo/root.key
  · /bindings/backend-postgrescluster-hippo/pgbouncer-frontend.key
  · /bindings/backend-postgrescluster-hippo/pgbouncer.ini
  · /bindings/backend-postgrescluster-hippo/uri
  · /bindings/backend-postgrescluster-hippo/config-hash
  · /bindings/backend-postgrescluster-hippo/pgbouncer-empty
  · /bindings/backend-postgrescluster-hippo/port
  · /bindings/backend-postgrescluster-hippo/dns.crt
  · /bindings/backend-postgrescluster-hippo/pgbouncer-uri
  · /bindings/backend-postgrescluster-hippo/root.crt
  · /bindings/backend-postgrescluster-hippo/ssh_config
  · /bindings/backend-postgrescluster-hippo/dns.key
  · /bindings/backend-postgrescluster-hippo/host
  · /bindings/backend-postgrescluster-hippo/patroni.crt-combined
  · /bindings/backend-postgrescluster-hippo/pgbouncer-frontend.ca-roots
  · /bindings/backend-postgrescluster-hippo/tls.key
  · /bindings/backend-postgrescluster-hippo/verifier
  · /bindings/backend-postgrescluster-hippo/ca.crt
```
Everything that was an environment variable in the key=value format in the earlier odo describe output is now mounted as a file. Use the cat command to view the contents of some of these files:

Example command:

```sh
$ odo exec -- cat /bindings/backend-postgrescluster-hippo/password
```

Example output:

```sh
q{\{JC:jn^mm/Bw}eu+J.GX{k
```

Example command:

```sh
$ odo exec -- cat /bindings/backend-postgrescluster-hippo/user
```

Example output:

```sh
hippo
```

Example command:

```sh
$ odo exec -- cat /bindings/backend-postgrescluster-hippo/clusterIP
```

Example output:

```sh
10.101.78.56
```

3.9.6.4.2. Using --inlined

The result of using --bind-as-files and --inlined together is similar to using odo link --inlined. The manifest of the link gets stored in the devfile.yaml, instead of being stored in a separate file under kubernetes/ directory. Other than that, the odo describe output would be the same as earlier.

3.9.6.4.3. Custom bindings

When you pass custom bindings while linking the backend component with the PostgreSQL service, these custom bindings are injected not as environment variables but are mounted as files. For example:

- /bindings/backend-postgrescluster-hippo/dbname
- /bindings/backend-postgrescluster-hippo/patroni.ca-roots
- /bindings/backend-postgrescluster-hippo/pgbackrest_repo.conf
- /bindings/backend-postgrescluster-hippo/pgbouncer-port
- /bindings/backend-postgrescluster-hippo/pgbouncer-verifier
- /bindings/backend-postgrescluster-hippo/id_ecdsa
- /bindings/backend-postgrescluster-hippo/id_ecdsa.pub
- /bindings/backend-postgrescluster-hippo/pgbouncer-password
- /bindings/backend-postgrescluster-hippo/pgbouncer-users.txt
- /bindings/backend-postgrescluster-hippo/sshd_config
- /bindings/backend-postgrescluster-hippo/tls.crt
These custom bindings get mounted as files instead of being injected as environment variables. To validate that this worked, run the following command:

Example command:

```
$ odo link PostgresCluster/hippo --map pgVersion='{{ .hippo.spec.postgresVersion }}' --map pgImage='{{ .hippo.spec.image }}' --bind-as-files
$ odo push
```

Example output:

```
3.9.7. odo registry
odo uses the portable devfile format to describe the components. odo can connect to various devfile registries, to download devfiles for different languages and frameworks.
You can connect to publicly available devfile registries, or you can install your own Secure Registry.
You can use the odo registry command to manage the registries that are used by odo to retrieve devfile information.

3.9.7.1. Listing the registries
To list the registries currently contacted by odo, run the command:

```
$ odo registry list
```

Example output:

```
NAME                       URL                             SECURE
DefaultDevfileRegistry     https://registry.devfile.io     No
```

DefaultDevfileRegistry is the default registry used by odo; it is provided by the devfile.io project.

3.9.7.2. Adding a registry
To add a registry, run the command:

```
$ odo registry add
```
**Example output:**

```bash
$ odo registry add StageRegistry https://registry.stage.devfile.io
New registry successfully added

$ odo registry add MyRegistry https://myregistry.example.com --token <access_token>
New registry successfully added

3.9.7.3. Deleting a registry

To delete a registry, run the command:

```bash
$ odo registry delete
```

**Example output:**

```bash
$ odo registry delete StageRegistry
? Are you sure you want to delete registry "StageRegistry" Yes
Successfully deleted registry
```

Use the `--force` (or `-f`) flag to force the deletion of the registry without confirmation.

**3.9.7.4. Updating a registry**

To update the URL or the personal access token of a registry already registered, run the command:

```bash
$ odo registry update
```

**Example output:**

```bash
$ odo registry update MyRegistry https://otherregistry.example.com --token <other_access_token>
? Are you sure you want to update registry "MyRegistry" Yes
Successfully updated registry
```

Use the `--force` (or `-f`) flag to force the update of the registry without confirmation.

**3.9.8. odo service**

`odo` can deploy services with the help of `Operators`.

The list of available Operators and services available for installation can be found using the `odo catalog` command.

Services are created in the context of a `component`, so run the `odo create` command before you deploy services.

A service is deployed using two steps:

1. Define the service and store its definition in the devfile.
2. Deploy the defined service to the cluster, using the `odo push` command.

### 3.9.8.1. Creating a new service

To create a new service, run the command:

```bash
$ odo service create
```

For example, to create an instance of a Redis service named `my-redis-service`, you can run the following command:

**Example output**

```bash
$ odo catalog list services
Services available through Operators
NAME                      CRDs
redis-operator.v0.8.0     RedisCluster, Redis

$ odo service create redis-operator.v0.8.0/Redis my-redis-service
Successfully added service to the configuration; do 'odo push' to create service on the cluster
```

This command creates a Kubernetes manifest in the `kubernetes/` directory, containing the definition of the service, and this file is referenced from the `devfile.yaml` file.

```bash
$ cat kubernetes/odo-service-my-redis-service.yaml
```

**Example output**

```yaml
apiVersion: redis.redis.opstreelabs.in/v1beta1
kind: Redis
metadata:  
  name: my-redis-service
spec:  
  kubernetesConfig:  
    image: quay.io/opstree/redis:v6.2.5  
    imagePullPolicy: IfNotPresent  
    resources:  
      limits:  
        cpu: 101m  
        memory: 128Mi  
      requests:  
        cpu: 101m  
        memory: 128Mi
  serviceType: ClusterIP
  redisExporter:  
    enabled: false  
    image: quay.io/opstree/redis-exporter:1.0
```

```yaml
storage:  
  volumeClaimTemplate:  
    spec:  
      accessModes:  
      - ReadWriteOnce
```
resources:
  requests:
    storage: 1Gi

Example command

$ cat devfile.yaml

Example output

[...]
components:
  - kubernetes:
      name: my-redis-service

Note that the name of the created instance is optional. If you do not provide a name, it will be the lowercase name of the service. For example, the following command creates an instance of a Redis service named *redis*:

$ odo service create redis-operator.v0.8.0/Redis my-redis-service --inlined

Successfully added service to the configuration; do 'odo push' to create service on the cluster

3.9.8.1.1. Inlining the manifest

By default, a new manifest is created in the *kubernetes/* directory, referenced from the *devfile.yaml* file. It is possible to inline the manifest inside the *devfile.yaml* file using the *--inlined* flag:

$ odo service create redis-operator.v0.8.0/Redis my-redis-service --inlined

Successfully added service to the configuration; do 'odo push' to create service on the cluster

Example command

$ cat devfile.yaml

Example output

[...]
components:
  - kubernetes:
      inlined:
        apiVersion: redis.redis.opstreelabs.in/v1beta1
        kind: Redis
        metadata:
          name: my-redis-service
        spec:
          kubernetesConfig:
            image: quay.io/opstree/redis:v6.2.5
            imagePullPolicy: IfNotPresent
            resources:
              limits:
                cpu: 101m
3.9.8.1.2. Configuring the service

Without specific customization, the service will be created with a default configuration. You can use either command-line arguments or a file to specify your own configuration.

3.9.8.1.2.1. Using command-line arguments

Use the `--parameters` (or `-p`) flag to specify your own configuration.

The following example configures the Redis service with three parameters:

```
$ odo service create redis-operator.v0.8.0/Redis my-redis-service \
  -p kubernetesConfig.image=quay.io/opstree/redis:v6.2.5 \
  -p kubernetesConfig.serviceType=ClusterIP \
  -p redisExporter.image=quay.io/opstree/redis-exporter:1.0
Successfully added service to the configuration; do 'odo push' to create service on the cluster
```

Example command

```
$ cat kubernetes/odo-service-my-redis-service.yaml
```

Example output

```
apiVersion: redis.redis.opstreelabs.in/v1beta1
kind: Redis
metadata:
  name: my-redis-service
spec:
  kubernetesConfig:
    image: quay.io/opstree/redis:v6.2.5
  serviceType: ClusterIP
  redisExporter:
    image: quay.io/opstree/redis-exporter:1.0
```
You can obtain the possible parameters for a specific service using the `odo catalog describe service` command.

### 3.9.8.1.2.2. Using a file

Use a YAML manifest to configure your own specification. In the following example, the Redis service is configured with three parameters.

1. **Create a manifest:**

   ```
   $ cat > my-redis.yaml <<EOF
   apiVersion: redis.redis.opstreelabs.in/v1beta1
   kind: Redis
   metadata:
     name: my-redis-service
   spec:
     kubernetesConfig:
       image: quay.io/opstree/redis:v6.2.5
       serviceType: ClusterIP
       redisExporter:
         image: quay.io/opstree/redis-exporter:1.0
   EOF
   ```

2. **Create the service from the manifest:**

   ```
   $ odo service create --from-file my-redis.yaml
   Successfully added service to the configuration; do 'odo push' to create service on the cluster
   ```

### 3.9.8.2. Deleting a service

To delete a service, run the command:

```
$ odo service delete
```

**Example output**

```
$ odo service list
NAME                       MANAGED BY ODO     STATE               AGE
Redis/my-redis-service     Yes (api)          Deleted locally     5m39s

$ odo service delete Redis/my-redis-service
? Are you sure you want to delete Redis/my-redis-service Yes
Service "Redis/my-redis-service" has been successfully deleted; do 'odo push' to delete service from the cluster
```

Use the **--force** (or `-f`) flag to force the deletion of the service without confirmation.

### 3.9.8.3. Listing services

To list the services created for your component, run the command:

```
$ odo service list
```
Example output

$ odo service list
NAME MANAGED BY ODO STATE AGE
Redis/my-redis-service-1 Yes (api) Not pushed
Redis/my-redis-service-2 Yes (api) Pushed 52s
Redis/my-redis-service-3 Yes (api) Deleted locally 1m22s

For each service, **STATE** indicates if the service has been pushed to the cluster using the **odo push** command, or if the service is still running on the cluster but removed from the devfile locally using the **odo service delete** command.

3.9.8.4. Getting information about a service

To get details of a service such as its kind, version, name, and list of configured parameters, run the command:

$ odo service describe

Example output

$ odo service describe Redis/my-redis-service
Version: redis.redis.opstreelabs.in/v1beta1
Kind: Redis
Name: my-redis-service
Parameters:
NAME VALUE
kubernetesConfig.image quay.io/opstree/redis:v6.2.5
kubernetesConfig.serviceType ClusterIP
redisExporter.image quay.io/opstree/redis-exporter:1.0

3.9.9. odo storage

**odo** lets users manage storage volumes that are attached to the components. A storage volume can be either an ephemeral volume using an **emptyDir** Kubernetes volume, or a **Persistent Volume Claim** (PVC). A PVC allows users to claim a persistent volume (such as a GCE PersistentDisk or an iSCSI volume) without understanding the details of the particular cloud environment. The persistent storage volume can be used to persist data across restarts and rebuilds of the component.

3.9.9.1. Adding a storage volume

To add a storage volume to the cluster, run the command:

$ odo storage create

Example output:

$ odo storage create store --path /data --size 1Gi
✓ Added storage store to nodejs-project-ufyy

$ odo storage create tempdir --path /tmp --size 2Gi --ephemeral
In the above example, the first storage volume has been mounted to the /data path and has a size of 1Gi, and the second volume has been mounted to /tmp and is ephemeral.

### 3.9.9.2. Listing the storage volumes

To check the storage volumes currently used by the component, run the command:

```bash
$ odo storage list
```

**Example output:**

```bash
$ odo storage list
The component 'nodejs-project-ufyy' has the following storage attached:
NAME   SIZE   PATH      STATE
store   1Gi    /data     Not Pushed
tempdir 2Gi    /tmp      Not Pushed
```

### 3.9.9.3. Deleting a storage volume

To delete a storage volume, run the command:

```bash
$ odo storage delete
```

**Example output:**

```bash
$ odo storage delete store -f
Deleted storage store from nodejs-project-ufyy

Please use `odo push` command to delete the storage from the cluster
```

In the above example, using the -f flag force deletes the storage without asking user permission.

### 3.9.9.4. Adding storage to specific container

If your devfile has multiple containers, you can specify which container you want the storage to attach to, using the **--container** flag in the odo storage create command.

The following example is an excerpt from a devfile with multiple containers:

```yaml
components:
- name: nodejs1
  container:
    image: registry.access.redhat.com/ubi8/nodejs-12:1-36
    memoryLimit: 1024Mi
    endpoints:
      - name: "3000-tcp"
        targetPort: 3000
    mountSources: true
```
In the example, there are two containers, `nodejs1` and `nodejs2`. To attach storage to the `nodejs2` container, use the following command:

```shell
$ odo storage create --container nodejs2
```

**Example output:**

```
$ odo storage create store --path /data --size 1Gi --container nodejs2
✓ Added storage store to nodejs-testing-xnfg
```

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

You can list the storage resources, using the `odo storage list` command:

```shell
$ odo storage list
```

**Example output:**

```
The component 'nodejs-testing-xnfg' has the following storage attached:
NAME  SIZE  PATH  CONTAINER  STATE
store  1Gi   /data  nodejs2  Not Pushed
```

### 3.9.10. Common flags

The following flags are available with most `odo` commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--context</td>
<td>Set the context directory where the component is defined.</td>
</tr>
<tr>
<td>--project</td>
<td>Set the project for the component. Defaults to the project defined in the local configuration. If none is available, then current project on the cluster.</td>
</tr>
<tr>
<td>--app</td>
<td>Set the application of the component. Defaults to the application defined in the local configuration. If none is available, then <code>app</code>.</td>
</tr>
<tr>
<td>--kubeconfig</td>
<td>Set the path to the <code>kubeconfig</code> value if not using the default configuration.</td>
</tr>
<tr>
<td>--show-log</td>
<td>Use this flag to see the logs.</td>
</tr>
<tr>
<td>-f, --force</td>
<td>Use this flag to tell the command not to prompt the user for confirmation.</td>
</tr>
<tr>
<td>-v, --v</td>
<td>Set the verbosity level. See <code>Logging in odo</code> for more information.</td>
</tr>
</tbody>
</table>
### Command Description

-h, --help

Output the help for a command.

**NOTE**

Some flags might not be available for some commands. Run the command with the `--help` flag to get a list of all the available flags.

#### 3.9.11. JSON output

The odo commands that output content generally accept a `-o json` flag to output this content in JSON format, suitable for other programs to parse this output more easily.

The output structure is similar to Kubernetes resources, with the `kind`, `apiVersion`, `metadata`, `spec`, and `status` fields.

*List* commands return a *List* resource, containing an `items` (or similar) field listing the items of the list, with each item also being similar to Kubernetes resources.

*Delete* commands return a *Status* resource; see the *Status Kubernetes resource*.

Other commands return a resource associated with the command, for example, *Application*, *Storage*, *URL*, and so on.

The full list of commands currently accepting the `-o json` flag is:

<table>
<thead>
<tr>
<th>Commands</th>
<th>Kind (version)</th>
<th>Kind (version) of list items</th>
<th>Complete content?</th>
</tr>
</thead>
<tbody>
<tr>
<td>odo application describe</td>
<td>Application (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>no</td>
</tr>
<tr>
<td>odo application list</td>
<td>List (odo.dev/v1alpha1)</td>
<td>Application (odo.dev/v1alpha1)</td>
<td>?</td>
</tr>
<tr>
<td>odo catalog list components</td>
<td>List (odo.dev/v1alpha1)</td>
<td>missing</td>
<td>yes</td>
</tr>
<tr>
<td>odo catalog list services</td>
<td>List (odo.dev/v1alpha1)</td>
<td>ClusterServiceVersion (operators.coreos.com/v1alpha1)</td>
<td>?</td>
</tr>
<tr>
<td>odo catalog describe component</td>
<td>missing</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo catalog describe service</td>
<td>CRDDescription (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>Commands</td>
<td>Kind (version)</td>
<td>Kind (version) of list items</td>
<td>Complete content?</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------</td>
<td>-------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>odo component create</td>
<td>Component (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo component describe</td>
<td>Component (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo component list</td>
<td>List (odo.dev/v1alpha1)</td>
<td>Component (odo.dev/v1alpha1)</td>
<td>yes</td>
</tr>
<tr>
<td>odo config view</td>
<td>DevfileConfiguration (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo debug info</td>
<td>OdoDebugInfo (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo env view</td>
<td>EnvInfo (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo preference view</td>
<td>PreferenceList (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo project create</td>
<td>Project (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo project delete</td>
<td>Status (v1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo project get</td>
<td>Project (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo project list</td>
<td>List (odo.dev/v1alpha1)</td>
<td>Project (odo.dev/v1alpha1)</td>
<td>yes</td>
</tr>
<tr>
<td>odo registry list</td>
<td>List (odo.dev/v1alpha1)</td>
<td>missing</td>
<td>yes</td>
</tr>
<tr>
<td>odo service create</td>
<td>Service</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo service describe</td>
<td>Service</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo service list</td>
<td>List (odo.dev/v1alpha1)</td>
<td>Service</td>
<td>yes</td>
</tr>
<tr>
<td>odo storage create</td>
<td>Storage (odo.dev/v1alpha1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>odo storage delete</td>
<td>Status (v1)</td>
<td>n/a</td>
<td>yes</td>
</tr>
</tbody>
</table>
### 3.10. ODO ARCHITECTURE

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

#### 3.10.1. Developer setup

With odo you can create and deploy an application on OpenShift Container Platform clusters from a terminal. Code editor plug-ins use odo which allows users to interact with OpenShift Container Platform clusters from their IDE terminals. Examples of plug-ins that use odo: VS Code OpenShift Connector, OpenShift Connector for Intellij, Codewind for Eclipse Che.

odo works on Windows, macOS, and Linux operating systems and from any terminal. odo provides autocompletion for bash and zsh command line shells.

odo supports Node.js and Java components.

#### 3.10.2. 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.

#### 3.10.3. OpenShift cluster objects

##### 3.10.3.1. 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.

##### 3.10.3.1.1. 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.

**3.10.3.1.2. 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.

**3.10.3.2. 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.

### 3.10.3.3. 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>

### 3.10.3.4. 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>

### 3.10.3.5. 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.

### 3.10.4. 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 4. KNATIVE CLI (KN) FOR USE WITH OPENSSHIFT SERVERLESS

The Knative `kn` CLI enables simple interaction with Knative components on 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.
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:
   
   ```bash
   # subscription-manager list --available --matches "pipelines"
   ```

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

5. Enable the repositories required by Red Hat OpenShift Pipelines:
   
   - Linux (x86_64, amd64)
     
     ```bash
     # subscription-manager repos --enable="pipelines-1.7-for-rhel-8-x86_64-rpms"
     ```
   
   - Linux on IBM Z and LinuxONE (s390x)
     
     ```bash
     # subscription-manager repos --enable="pipelines-1.7-for-rhel-8-s390x-rpms"
     ```
   
   - Linux on IBM Power Systems (ppc64le)
     
     ```bash
     # subscription-manager repos --enable="pipelines-1.7-for-rhel-8-ppc64le-rpms"
     ```

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

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

```bash
$ tkn version
```

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

### 5.1.4. Installing Red Hat OpenShift Pipelines CLI (tkn) on macOS
For macOS, the `tkn` CLI is provided as a `tar.gz` archive.

**Procedure**

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

**5.2. CONFIGURING THE OPENSIFT PIPELINES TKN CLI**

Configure the Red Hat OpenShift Pipelines `tkn` CLI to enable tab completion.

**5.2.1. Enabling tab completion**

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

**Prerequisites**

- You must have the `tkn` CLI tool installed.
- You must have `bash-completion` installed on your local system.

**Procedure**

The following procedure enables tab completion for Bash.

1. Save the Bash completion code to a file:
   ```
   $ tkn completion bash > tkn_bash_completion
   ```
2. Copy the file to `/etc/bash_completion.d/`:
   ```
   $ sudo cp tkn_bash_completion /etc/bash_completion.d/
   ```
   Alternatively, you can save the file to a local directory and source it from your `.bashrc` file instead.

Tab completion is enabled when you open a new terminal.

**5.3. OPENSIFT PIPELINES TKN REFERENCE**

This section lists the basic `tkn` CLI commands.

**5.3.1. Basic syntax**

`tkn [command or options] [arguments...]`
5.3.2. Global options

--help, -h

5.3.3. Utility commands

5.3.3.1. tkn

Parent command for tkn CLI.

Example: Display all options

  $ tkn

5.3.3.2. completion [shell]

Print shell completion code which must be evaluated to provide interactive completion. Supported shells are bash and zsh.

Example: Completion code for bash shell

  $ tkn completion bash

5.3.3.3. version

Print version information of the tkn CLI.

Example: Check the tkn version

  $ tkn version

5.3.4. Pipelines management commands

5.3.4.1. pipeline

Manage pipelines.

Example: Display help

  $ tkn pipeline --help

5.3.4.2. pipeline 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 the mypipeline pipeline**

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

5.3.4.4. pipeline list
Display a list of pipelines.

**Example: Display a list of pipelines**

```bash
$ tkn pipeline list
```

5.3.4.5. pipeline logs
Display the logs for a specific pipeline.

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

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

5.3.4.6. pipeline start
Start a pipeline.

**Example: Start the mypipeline pipeline**

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

5.3.5. Pipeline run commands

5.3.5.1. pipelinerun
Manage pipeline runs.

**Example: Display help**

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

5.3.5.2. pipelinerun cancel
Cancel a pipeline run.

**Example: Cancel the mypipelinerun pipeline run from a namespace**

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

5.3.5.3. pipelinerun delete
Delete a pipeline run.

**Example: Delete pipeline runs from a namespace**

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

**Example: Delete all pipeline runs from a namespace, except the five most recently executed pipeline runs**

```bash
$ tkn pipelinerun delete -n myspace --keep 5
```

1. Replace 5 with the number of most recently executed pipeline runs you want to retain.

**Example: Delete all pipelines**

```bash
$ tkn pipelinerun delete --all
```

**NOTE**

Starting with Red Hat OpenShift Pipelines 1.6, the `tkn pipelinerun delete --all` command does not delete any resources that are in the running state.

5.3.5.4. pipelinerun describe

Describe a pipeline run.

**Example: Describe the mypipelinerun pipeline run in a namespace**

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

5.3.5.5. pipelinerun list

List pipeline runs.

**Example: Display a list of pipeline runs in a namespace**

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

5.3.5.6. pipelinerun logs

Display the logs of a pipeline run.

**Example: Display the logs of the mypipelinerun pipeline run with all tasks and steps in a namespace**

```bash
$ 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 task run 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. Task run commands
5.3.7.1. taskrun
Manage task runs.

Example: Display help

$ tkn taskrun -h

5.3.7.2. taskrun cancel
Cancel a task run.

Example: Cancel the mytaskrun task run from a namespace

$ tkn taskrun cancel mytaskrun -n myspace

5.3.7.3. taskrun delete
Delete a TaskRun.

Example: Delete the mytaskrun1 and mytaskrun2 task runs from a namespace

$ tkn taskrun delete mytaskrun1 mytaskrun2 -n myspace

Example: Delete all but the five most recently executed task runs from a namespace

$ tkn taskrun delete -n myspace --keep 5

1 Replace 5 with the number of most recently executed task runs you want to retain.

5.3.7.4. taskrun describe
Describe a task run.

Example: Describe the mytaskrun task run in a namespace

$ tkn taskrun describe mytaskrun -n myspace

5.3.7.5. taskrun list
List task runs.

Example: List all the task runs in a namespace

$ tkn taskrun list -n myspace

5.3.7.6. taskrun logs
Display task run logs.
Example: Display live logs for the mytaskrun task run in a namespace

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

5.3.8. Condition management commands

5.3.8.1. condition

Manage Conditions.

Example: Display help

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

5.3.8.2. condition delete

Delete a Condition.

Example: Delete the mycondition1 Condition from a namespace

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

5.3.8.3. condition describe

Describe a Condition.

Example: Describe the mycondition1 Condition in a namespace

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

5.3.8.4. condition list

List Conditions.

Example: List Conditions in a namespace

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

5.3.9. Pipeline Resource management commands

5.3.9.1. resource

Manage Pipeline Resources.

Example: Display help

```
$ tkn resource -h
```

5.3.9.2. resource create
Create a Pipeline Resource.

**Example: Create a Pipeline Resource in a namespace**

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

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

**5.3.9.3. resource delete**

Delete a Pipeline Resource.

**Example: Delete the myresource Pipeline Resource from a namespace**

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

**5.3.9.4. resource describe**

Describe a Pipeline Resource.

**Example: Describe the myresource Pipeline Resource**

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

**5.3.9.5. resource list**

List Pipeline Resources.

**Example: List all Pipeline Resources in a namespace**

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

**5.3.10. ClusterTask management commands**

**5.3.10.1. clustertask**

Manage ClusterTasks.

**Example: Display help**

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

**5.3.10.2. clustertask delete**

Delete a ClusterTask resource in a cluster.

**Example: Delete mytask1 and mytask2 ClusterTasks**

```
$ tkn clustertask delete mytask1 mytask2
```
5.3.10.3. clustertask describe
Describe a ClusterTask.

Example: Describe the mytask ClusterTask

$ tkn clustertask describe mytask1

5.3.10.4. clustertask list
List ClusterTasks.

Example: List ClusterTasks

$ tkn clustertask list

5.3.10.5. clustertask start
Start ClusterTasks.

Example: Start the mytask ClusterTask

$ tkn clustertask start mytask

5.3.11. Trigger management commands

5.3.11.1. eventlistener
Manage EventListeners.

Example: Display help

$ tkn eventlistener -h

5.3.11.2. eventlistener delete
Delete an EventListener.

Example: Delete mylistener1 and mylistener2 EventListeners in a namespace

$ tkn eventlistener delete mylistener1 mylistener2 -n myspace

5.3.11.3. eventlistener describe
Describe an EventListener.

Example: Describe the mylistener EventListener in a namespace

$ tkn eventlistener describe mylistener -n myspace
5.3.11.4. eventlistener list
List EventListeners.

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

```bash
$ 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**

```bash
$ tkn eventlistener logs mylistener -n myspace
```

5.3.11.6. triggerbinding
Manage TriggerBindings.

**Example: Display TriggerBindings help**

```bash
$ tkn triggerbinding -h
```

5.3.11.7. triggerbinding delete
Delete a TriggerBinding.

**Example: Delete mybinding1 and mybinding2 TriggerBindings in a namespace**

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

5.3.11.8. triggerbinding describe
Describe a TriggerBinding.

**Example: Describe the mybinding TriggerBinding in a namespace**

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

5.3.11.9. triggerbinding list
List TriggerBindings.

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

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

5.3.11.10. triggertemplate
Manage TriggerTemplates.

**Example: Display TriggerTemplate help**

```bash
$ tkn triggertemplate -h
```

5.3.11.11. triggertemplate delete
Delete a TriggerTemplate.

**Example: Delete mytemplate1 and mytemplate2 TriggerTemplates in a namespace**

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

5.3.11.12. triggertemplate describe
Describe a TriggerTemplate.

**Example: Describe the mytemplate TriggerTemplate in a namespace**

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

5.3.11.13. triggertemplate list
List TriggerTemplates.

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

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

5.3.11.14. clustertriggerbinding
Manage ClusterTriggerBindings.

**Example: Display ClusterTriggerBindings help**

```bash
$ tkn clustertriggerbinding -h
```

5.3.11.15. clustertriggerbinding delete
Delete a ClusterTriggerBinding.

**Example: Delete myclusterbinding1 and myclusterbinding2 ClusterTriggerBindings**

```bash
$ 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 it’s 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**

```bash
$ 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**

```bash
$ 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**

```bash
$ 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**

```bash
$ 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**

```bash
$ tkn hub upgrade task mytask --to version -n mynamespace
```
CHAPTER 6. OPM CLI

6.1. INSTALLING THE OPM CLI

6.1.1. About the opm CLI

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 Operator bundles that are similar to software repositories. The result is a container image which can be stored in a container registry and then installed on a cluster.

A catalog 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 reference the image in a catalog source, defined by 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 format for more information about the bundle format.
- To create a bundle image using the Operator SDK, see Working with bundle images.

6.1.2. Installing the opm CLI

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

### 6.1.3. Additional resources

- See [Managing custom catalogs](#) for `opm` procedures including creating, updating, and pruning catalogs.

### 6.2. OPM CLI REFERENCE

The `opm` command-line interface (CLI) is a tool for creating and maintaining Operator catalogs.

**opm CLI syntax**

$ opm <command> [<subcommand>] [<argument>] [<flags>]

**Table 6.1. Global flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--skip-tls</code></td>
<td>Skip TLS certificate verification for container image registries while pulling bundles or indexes.</td>
</tr>
</tbody>
</table>
IMPORTANT

The SQLite-based catalog format, including the related CLI commands, is a deprecated feature. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

For the most recent list of major functionality that has been deprecated or removed within OpenShift Container Platform, refer to the Deprecated and removed features section of the OpenShift Container Platform release notes.

6.2.1. index

Generate Operator index container images from pre-existing Operator bundles.

Command syntax

```bash
$ opm index <subcommand> [<flags>]
```

Table 6.2. index subcommands

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>Add Operator bundles to an index.</td>
</tr>
<tr>
<td>prune</td>
<td>Prune an index of all but specified packages.</td>
</tr>
<tr>
<td>prune-stranded</td>
<td>Prune an index of stranded bundles, which are bundles that are not associated with a particular image.</td>
</tr>
<tr>
<td>rm</td>
<td>Delete an entire Operator from an index.</td>
</tr>
</tbody>
</table>

6.2.1.1. add

Add Operator bundles to an index.

Command syntax

```bash
$ opm index add [<flags>]
```

Table 6.3. index add flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-i, --binary-image</td>
<td>Container image for on-image opm command</td>
</tr>
<tr>
<td>-u, --build-tool (string)</td>
<td>Tool to build container images: podman (the default value) or docker. Overrides part of the --container-tool flag.</td>
</tr>
<tr>
<td>-b, --bundles (strings)</td>
<td>Comma-separated list of bundles to add.</td>
</tr>
<tr>
<td>Flag</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>--container-tool</code></td>
<td>Tool to interact with container images, such as for saving and building: <strong>docker</strong> or <strong>podman</strong>.</td>
</tr>
<tr>
<td><code>--from-index</code></td>
<td>Previous index to add to.</td>
</tr>
<tr>
<td><code>--generate</code></td>
<td>If enabled, only creates the Dockerfile and saves it to local disk.</td>
</tr>
<tr>
<td><code>--mode</code></td>
<td>Graph update mode that defines how channel graphs are updated: <strong>replaces</strong> (the default value), <strong>semver</strong>, or <strong>semver-skippatch</strong>.</td>
</tr>
<tr>
<td><code>--out-dockerfile</code></td>
<td>Optional: If generating the Dockerfile, specify a file name.</td>
</tr>
<tr>
<td><code>--permissive</code></td>
<td>Allow registry load errors.</td>
</tr>
<tr>
<td><code>--pull-tool</code></td>
<td>Tool to pull container images: <strong>none</strong> (the default value), <strong>docker</strong>, or <strong>podman</strong>.</td>
</tr>
<tr>
<td><code>--tag</code></td>
<td>Custom tag for container image being built.</td>
</tr>
</tbody>
</table>

### 6.2.1.2. prune

Prune an index of all but specified packages.

**Command syntax**

```
$ opm index prune [<flags>]
```

**Table 6.4. index prune flags**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--binary-image</code></td>
<td>Container image for on-image <strong>opm</strong> command</td>
</tr>
<tr>
<td><code>--container-tool</code></td>
<td>Tool to interact with container images, such as for saving and building: <strong>docker</strong> or <strong>podman</strong>.</td>
</tr>
<tr>
<td><code>--from-index</code></td>
<td>Index to prune.</td>
</tr>
<tr>
<td><code>--generate</code></td>
<td>If enabled, only creates the Dockerfile and saves it to local disk.</td>
</tr>
<tr>
<td><code>--out-dockerfile</code></td>
<td>Optional: If generating the Dockerfile, specify a file name.</td>
</tr>
<tr>
<td>Flag</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>-p, --packages (strings)</td>
<td>Comma-separated list of packages to keep.</td>
</tr>
<tr>
<td>--permissive</td>
<td>Allow registry load errors.</td>
</tr>
<tr>
<td>-t, --tag (string)</td>
<td>Custom tag for container image being built.</td>
</tr>
</tbody>
</table>

### 6.2.1.3. prune-stranded

Prune an index of stranded bundles, which are bundles that are not associated with a particular image.

#### Command syntax

```
$ opm index prune-stranded [flags]
```

#### Table 6.5. index prune-stranded flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-i, --binary-image</td>
<td>Container image for on-image <code>opm</code> command</td>
</tr>
<tr>
<td>-c, --container-tool (string)</td>
<td>Tool to interact with container images, such as for saving and building: <code>docker</code> or <code>podman</code>.</td>
</tr>
<tr>
<td>-f, --from-index (string)</td>
<td>Index to prune.</td>
</tr>
<tr>
<td>--generate</td>
<td>If enabled, only creates the Dockerfile and saves it to local disk.</td>
</tr>
<tr>
<td>-d, --out-dockerfile (string)</td>
<td>Optional: If generating the Dockerfile, specify a file name.</td>
</tr>
<tr>
<td>-p, --packages (strings)</td>
<td>Comma-separated list of packages to keep.</td>
</tr>
<tr>
<td>--permissive</td>
<td>Allow registry load errors.</td>
</tr>
<tr>
<td>-t, --tag (string)</td>
<td>Custom tag for container image being built.</td>
</tr>
</tbody>
</table>

### 6.2.1.4. rm

Delete an entire Operator from an index.

#### Command syntax
$ opm index rm [<flags>]

Table 6.6. index rm flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-i, --binary-image</td>
<td>Container image for on-image opm command</td>
</tr>
<tr>
<td>-u, --build-tool (string)</td>
<td>Tool to build container images: podman (the default value) or docker. Overrides part of the --container-tool flag.</td>
</tr>
<tr>
<td>-c, --container-tool (string)</td>
<td>Tool to interact with container images, such as for saving and building: docker or podman.</td>
</tr>
<tr>
<td>-f, --from-index (string)</td>
<td>Previous index to delete from.</td>
</tr>
<tr>
<td>--generate</td>
<td>If enabled, only creates the Dockerfile and saves it to local disk.</td>
</tr>
<tr>
<td>-o, --operators (strings)</td>
<td>Comma-separated list of Operators to delete.</td>
</tr>
<tr>
<td>-d, --out-dockerfile (string)</td>
<td>Optional: If generating the Dockerfile, specify a file name.</td>
</tr>
<tr>
<td>-p, --packages (strings)</td>
<td>Comma-separated list of packages to keep.</td>
</tr>
<tr>
<td>--permissive</td>
<td>Allow registry load errors.</td>
</tr>
<tr>
<td>-p, --pull-tool (string)</td>
<td>Tool to pull container images: none (the default value), docker, or podman. Overrides part of the --container-tool flag.</td>
</tr>
<tr>
<td>-t, --tag (string)</td>
<td>Custom tag for container image being built.</td>
</tr>
</tbody>
</table>

6.2.2. init

Generate an olm.package declarative config blob.

Command syntax

$ opm init <package_name> [<flags>]

Table 6.7. init flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c, --default-channel (string)</td>
<td>The channel that subscriptions will default to if unspecified.</td>
</tr>
</tbody>
</table>
Path to the Operator’s README.md or other documentation.

Path to package’s icon.

Output format: json (the default value) or yaml.

### 6.2.3. render

Generate a declarative config blob from the provided index images, bundle images, and SQLite database files.

**Command syntax**

```
$ opm render <index_image | bundle_image | sqlite_file> [flags]
```

### 6.2.4. validate

Validate the declarative config JSON file(s) in a given directory.

**Command syntax**

```
$ opm validate <directory> [flags]
```

### 6.2.5. serve

Serve declarative configs via a GRPC server.

**NOTE**

The declarative config directory is loaded by the `serve` command at startup. Changes made to the declarative config after this command starts are not reflected in the served content.

**Command syntax**

```
$ opm serve <source_path> [flags]
```
<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--debug</code></td>
<td>Enable debug logging.</td>
</tr>
<tr>
<td><code>-p, --port</code> (string)</td>
<td>Port number to serve on. Default: <code>50051</code>.</td>
</tr>
<tr>
<td><code>-t, --termination-log</code> (string)</td>
<td>Path to a container termination log file. Default: <code>/dev/termination-log</code>.</td>
</tr>
</tbody>
</table>
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.

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.

NOTE

OpenShift Container Platform 4.9 and later supports Operator SDK v1.16.0.

7.1.1. Installing the Operator SDK CLI

You can install the OpenShift SDK CLI tool on Linux.

**Prerequisites**

- Go v1.16+
- [docker] v17.03+, [podman] v1.9.3+, or [buildah] v1.7+

**Procedure**

1. Navigate to the OpenShift mirror site.
2. From the latest 4.10 directory, download the latest version of the tarball for Linux.
3. Unpack the archive:
   
   ```bash
   $ tar xvf operator-sdk-v1.16.0-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`.

**TIP**

To check your `PATH`:

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

Verification

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

  $ operator-sdk version

Example output

  operator-sdk version: "v1.16.0-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>]

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

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

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:

```bash
$ 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><code>-h, --help</code></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><code>--channels</code> (string)</td>
<td>Comma-separated list of channels to which the bundle belongs. The default value is <code>alpha</code>.</td>
</tr>
<tr>
<td><code>--crds-dir</code> (string)</td>
<td>Root directory for <code>CustomResoureDefinition</code> manifests.</td>
</tr>
<tr>
<td><code>--default-channel</code> (string)</td>
<td>The default channel for the bundle.</td>
</tr>
<tr>
<td><code>--deploy-dir</code> (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><code>-h, --help</code></td>
<td>Help for <code>generate bundle</code></td>
</tr>
<tr>
<td><code>--input-dir</code> (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><code>--kustomize-dir</code> (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><code>--manifests</code></td>
<td>Generate bundle manifests.</td>
</tr>
<tr>
<td>Flag</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>--metadata</td>
<td>Generate bundle metadata and Dockerfile.</td>
</tr>
<tr>
<td>--output-dir</td>
<td>Directory to write the bundle to.</td>
</tr>
<tr>
<td>--overwrite</td>
<td>Overwrite the bundle metadata and Dockerfile if they exist. The default value is true.</td>
</tr>
<tr>
<td>--package</td>
<td>Package name for the bundle.</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Run in quiet mode.</td>
</tr>
<tr>
<td>--stdout</td>
<td>Write bundle manifest to standard out.</td>
</tr>
<tr>
<td>--version</td>
<td>Semantic version of the Operator in the generated bundle. Set only when creating a new bundle or upgrading the Operator.</td>
</tr>
</tbody>
</table>

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>--apis-dir</td>
<td>Root directory for API type definitions.</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Help for generate kustomize manifests.</td>
</tr>
<tr>
<td>--input-dir</td>
<td>Directory containing existing Kustomize files.</td>
</tr>
<tr>
<td>--interactive</td>
<td>When set to false, 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 an 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>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--output-dir</code></td>
<td>Directory where to write Kustomize files.</td>
</tr>
<tr>
<td><code>--package</code></td>
<td>Package name.</td>
</tr>
<tr>
<td><code>-q</code> <code>--quiet</code></td>
<td>Run in quiet mode.</td>
</tr>
</tbody>
</table>

#### Table 7.8. init flags

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--help, -h</code></td>
<td>Help output for the <code>init</code> command.</td>
</tr>
<tr>
<td><code>--plugins</code></td>
<td>Name and optionally version of the plug-in to initialize the project with. Available plug-ins are <code>ansible.sdk.operatorframework.io/v1</code>, <code>go.kubebuilder.io/v2</code>, <code>go.kubebuilder.io/v3</code>, and <code>helm.sdk.operatorframework.io/v1</code>.</td>
</tr>
<tr>
<td><code>--project-version</code></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 &lt;install_mode_value&gt;</code></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 &lt;duration&gt;</code></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>n. --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 &lt;duration&gt;</code></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>n. --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><strong>-c, --config</strong> (string)</td>
<td>Path to scorecard configuration file. The default path is <code>bundle/tests/scorecard/config.yaml</code>.</td>
</tr>
<tr>
<td><strong>-h, --help</strong></td>
<td>Help output for the scorecard command.</td>
</tr>
<tr>
<td><strong>--kubeconfig</strong> (string)</td>
<td>Path to kubeconfig file.</td>
</tr>
<tr>
<td><strong>-L, --list</strong></td>
<td>List which tests are available to run.</td>
</tr>
<tr>
<td><strong>-n, --namespace</strong> (string)</td>
<td>Namespace in which to run the test images.</td>
</tr>
<tr>
<td><strong>-o, --output</strong> (string)</td>
<td>Output format for results. Available values are text, which is the default, and json.</td>
</tr>
<tr>
<td><strong>-l, --selector</strong> (string)</td>
<td>Label selector to determine which tests are run.</td>
</tr>
<tr>
<td><strong>-s, --service-account</strong> (string)</td>
<td>Service account to use for tests. The default value is default.</td>
</tr>
<tr>
<td><strong>-x, --skip-cleanup</strong></td>
<td>Disable resource cleanup after tests are run.</td>
</tr>
<tr>
<td><strong>-w, --wait-time</strong> &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.