



OpenShift Container Platform 4.12

CI/CD

Contains information on builds, pipelines and GitOps for OpenShift Container Platform

OpenShift Container Platform 4.12 CI/CD

Contains information on builds, pipelines and GitOps for OpenShift Container Platform

Legal Notice

Copyright © 2023 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

CI/CD for the OpenShift Container Platform

Table of Contents

CHAPTER 1. OPENSIFT CONTAINER PLATFORM CI/CD OVERVIEW	14
1.1. OPENSIFT BUILDS	14
1.2. OPENSIFT PIPELINES	14
1.3. OPENSIFT GITOPS	14
1.4. JENKINS	14
CHAPTER 2. BUILDS	15
2.1. UNDERSTANDING IMAGE BUILDS	15
2.1.1. Builds	15
2.1.1.1. Docker build	15
2.1.1.2. Source-to-image build	15
2.1.1.3. Custom build	16
2.1.1.4. Pipeline build	16
2.2. UNDERSTANDING BUILD CONFIGURATIONS	16
2.2.1. BuildConfigs	16
2.3. CREATING BUILD INPUTS	18
2.3.1. Build inputs	18
2.3.2. Dockerfile source	19
2.3.3. Image source	19
2.3.4. Git source	21
2.3.4.1. Using a proxy	21
2.3.4.2. Source Clone Secrets	22
2.3.4.2.1. Automatically adding a source clone secret to a build configuration	22
2.3.4.2.2. Manually adding a source clone secret	24
2.3.4.2.3. Creating a secret from a .gitconfig file	24
2.3.4.2.4. Creating a secret from a .gitconfig file for secured Git	25
2.3.4.2.5. Creating a secret from source code basic authentication	26
2.3.4.2.6. Creating a secret from source code SSH key authentication	26
2.3.4.2.7. Creating a secret from source code trusted certificate authorities	27
2.3.4.2.8. Source secret combinations	28
2.3.4.2.8.1. Creating a SSH-based authentication secret with a .gitconfig file	28
2.3.4.2.8.2. Creating a secret that combines a .gitconfig file and CA certificate	28
2.3.4.2.8.3. Creating a basic authentication secret with a CA certificate	28
2.3.4.2.8.4. Creating a basic authentication secret with a .gitconfig file	29
2.3.4.2.8.5. Creating a basic authentication secret with a .gitconfig file and CA certificate	29
2.3.5. Binary (local) source	30
2.3.6. Input secrets and config maps	31
2.3.6.1. What is a secret?	31
2.3.6.1.1. Properties of secrets	32
2.3.6.1.2. Types of Secrets	32
2.3.6.1.3. Updates to secrets	33
2.3.6.2. Creating secrets	33
2.3.6.3. Using secrets	34
2.3.6.4. Adding input secrets and config maps	36
2.3.6.5. Source-to-image strategy	38
2.3.6.6. Docker strategy	38
2.3.6.7. Custom strategy	39
2.3.7. External artifacts	39
2.3.8. Using docker credentials for private registries	40
2.3.9. Build environments	42
2.3.9.1. Using build fields as environment variables	43

2.3.9.2. Using secrets as environment variables	43
2.3.10. Service serving certificate secrets	44
2.3.11. Secrets restrictions	44
2.4. MANAGING BUILD OUTPUT	45
2.4.1. Build output	45
2.4.2. Output image environment variables	45
2.4.3. Output image labels	46
2.5. USING BUILD STRATEGIES	47
2.5.1. Docker build	47
2.5.1.1. Replacing Dockerfile FROM image	47
2.5.1.2. Using Dockerfile path	47
2.5.1.3. Using docker environment variables	47
2.5.1.4. Adding docker build arguments	48
2.5.1.5. Squashing layers with docker builds	48
2.5.1.6. Using build volumes	49
2.5.2. Source-to-image build	50
2.5.2.1. Performing source-to-image incremental builds	50
2.5.2.2. Overriding source-to-image builder image scripts	51
2.5.2.3. Source-to-image environment variables	51
2.5.2.3.1. Using source-to-image environment files	51
2.5.2.3.2. Using source-to-image build configuration environment	52
2.5.2.4. Ignoring source-to-image source files	52
2.5.2.5. Creating images from source code with source-to-image	52
2.5.2.5.1. Understanding the source-to-image build process	52
2.5.2.5.2. How to write source-to-image scripts	53
2.5.2.6. Using build volumes	55
2.5.3. Custom build	56
2.5.3.1. Using FROM image for custom builds	57
2.5.3.2. Using secrets in custom builds	57
2.5.3.3. Using environment variables for custom builds	57
2.5.3.4. Using custom builder images	58
2.5.3.4.1. Custom builder image	58
2.5.3.4.2. Custom builder workflow	59
2.5.4. Pipeline build	59
2.5.4.1. Understanding OpenShift Container Platform pipelines	59
2.5.4.2. Providing the Jenkins file for pipeline builds	60
2.5.4.3. Using environment variables for pipeline builds	62
2.5.4.3.1. Mapping between BuildConfig environment variables and Jenkins job parameters	62
2.5.4.4. Pipeline build tutorial	63
2.5.5. Adding secrets with web console	67
2.5.6. Enabling pulling and pushing	68
2.6. CUSTOM IMAGE BUILDS WITH BUILDAH	68
2.6.1. Prerequisites	68
2.6.2. Creating custom build artifacts	68
2.6.3. Build custom builder image	69
2.6.4. Use custom builder image	70
2.7. PERFORMING AND CONFIGURING BASIC BUILDS	71
2.7.1. Starting a build	71
2.7.1.1. Re-running a build	71
2.7.1.2. Streaming build logs	72
2.7.1.3. Setting environment variables when starting a build	72
2.7.1.4. Starting a build with source	72
2.7.2. Canceling a build	73

2.7.2.1. Canceling multiple builds	73
2.7.2.2. Canceling all builds	73
2.7.2.3. Canceling all builds in a given state	73
2.7.3. Editing a BuildConfig	73
2.7.4. Deleting a BuildConfig	75
2.7.5. Viewing build details	75
2.7.6. Accessing build logs	76
2.7.6.1. Accessing BuildConfig logs	76
2.7.6.2. Accessing BuildConfig logs for a given version build	76
2.7.6.3. Enabling log verbosity	76
2.8. TRIGGERING AND MODIFYING BUILDS	77
2.8.1. Build triggers	77
2.8.1.1. Webhook triggers	77
2.8.1.1.1. Using GitHub webhooks	78
2.8.1.1.2. Using GitLab webhooks	80
2.8.1.1.3. Using Bitbucket webhooks	80
2.8.1.1.4. Using generic webhooks	81
2.8.1.1.5. Displaying webhook URLs	83
2.8.1.2. Using image change triggers	83
2.8.1.3. Identifying the image change trigger of a build	85
2.8.1.4. Configuration change triggers	87
2.8.1.4.1. Setting triggers manually	87
2.8.2. Build hooks	87
2.8.2.1. Configuring post commit build hooks	88
2.8.2.2. Using the CLI to set post commit build hooks	89
2.9. PERFORMING ADVANCED BUILDS	89
2.9.1. Setting build resources	89
2.9.2. Setting maximum duration	90
2.9.3. Assigning builds to specific nodes	90
2.9.4. Chained builds	91
2.9.5. Pruning builds	93
2.9.6. Build run policy	93
2.10. USING RED HAT SUBSCRIPTIONS IN BUILDS	94
2.10.1. Creating an image stream tag for the Red Hat Universal Base Image	94
2.10.2. Adding subscription entitlements as a build secret	95
2.10.3. Running builds with Subscription Manager	96
2.10.3.1. Docker builds using Subscription Manager	96
2.10.4. Running builds with Red Hat Satellite subscriptions	96
2.10.4.1. Adding Red Hat Satellite configurations to builds	96
2.10.4.2. Docker builds using Red Hat Satellite subscriptions	97
2.10.5. Running entitled builds using SharedSecret objects	97
2.10.6. Additional resources	101
2.11. SECURING BUILDS BY STRATEGY	102
2.11.1. Disabling access to a build strategy globally	102
2.11.2. Restricting build strategies to users globally	104
2.11.3. Restricting build strategies to a user within a project	104
2.12. BUILD CONFIGURATION RESOURCES	105
2.12.1. Build controller configuration parameters	105
2.12.2. Configuring build settings	106
2.13. TROUBLESHOOTING BUILDS	107
2.13.1. Resolving denial for access to resources	107
2.13.2. Service certificate generation failure	107
2.14. SETTING UP ADDITIONAL TRUSTED CERTIFICATE AUTHORITIES FOR BUILDS	108

2.14.1. Adding certificate authorities to the cluster	108
2.14.2. Additional resources	109
CHAPTER 3. PIPELINES	110
3.1. RED HAT OPENSIFT PIPELINES RELEASE NOTES	110
3.1.1. Compatibility and support matrix	110
3.1.2. Making open source more inclusive	111
3.1.3. Release notes for Red Hat OpenShift Pipelines General Availability 1.9	111
3.1.3.1. New features	111
3.1.3.1.1. Pipelines	112
3.1.3.1.2. Triggers	112
3.1.3.1.3. CLI	112
3.1.3.1.4. Operator	113
3.1.3.1.5. Resolvers	113
3.1.3.1.6. Tekton Chains	114
3.1.3.1.7. Tekton Hub	114
3.1.3.1.8. Pipelines as Code	115
3.1.3.2. Breaking changes	117
3.1.3.3. Deprecated and removed features	117
3.1.3.4. Known issues	118
3.1.3.5. Fixed issues	118
3.1.4. Release notes for Red Hat OpenShift Pipelines General Availability 1.8	119
3.1.4.1. New features	120
3.1.4.1.1. Pipelines	120
3.1.4.1.2. Triggers	121
3.1.4.1.3. CLI	121
3.1.4.1.4. Operator	121
3.1.4.1.5. Tekton Chains	122
3.1.4.1.6. Tekton Hub	122
3.1.4.1.7. Pipelines as Code	123
3.1.4.2. Breaking changes	124
3.1.4.3. Deprecated and removed features	125
3.1.4.4. Known issues	126
3.1.4.5. Fixed issues	127
3.1.4.6. Release notes for Red Hat OpenShift Pipelines General Availability 1.8.1	130
3.1.4.6.1. Known issues	130
3.1.4.6.2. Fixed issues	130
3.1.4.7. Release notes for Red Hat OpenShift Pipelines General Availability 1.8.2	131
3.1.4.7.1. Fixed issues	131
3.1.5. Release notes for Red Hat OpenShift Pipelines General Availability 1.7	131
3.1.5.1. New features	131
3.1.5.1.1. Pipelines	131
3.1.5.1.2. Triggers	132
3.1.5.1.3. CLI	133
3.1.5.1.4. Operator	134
3.1.5.1.5. Hub	135
3.1.5.1.6. Chains	135
3.1.5.1.7. Pipelines as Code (PAC)	135
3.1.5.2. Deprecated features	136
3.1.5.3. Known issues	136
3.1.5.4. Fixed issues	137
3.1.5.5. Release notes for Red Hat OpenShift Pipelines General Availability 1.7.1	137
3.1.5.5.1. Fixed issues	137

3.1.5.6. Release notes for Red Hat OpenShift Pipelines General Availability 1.7.2	138
3.1.5.6.1. Known issues	138
3.1.5.6.2. Fixed issues	138
3.1.5.7. Release notes for Red Hat OpenShift Pipelines General Availability 1.7.3	139
3.1.5.7.1. Fixed issues	139
3.1.6. Release notes for Red Hat OpenShift Pipelines General Availability 1.6	139
3.1.6.1. New features	139
3.1.6.2. Deprecated features	142
3.1.6.3. Known issues	144
3.1.6.4. Fixed issues	144
3.1.6.5. Release notes for Red Hat OpenShift Pipelines General Availability 1.6.1	144
3.1.6.5.1. Known issues	145
3.1.6.5.2. Fixed issues	145
3.1.6.6. Release notes for Red Hat OpenShift Pipelines General Availability 1.6.2	145
3.1.6.6.1. Known issues	145
3.1.6.6.2. Fixed issues	145
3.1.6.7. Release notes for Red Hat OpenShift Pipelines General Availability 1.6.3	146
3.1.6.7.1. Fixed issues	146
3.1.6.8. Release notes for Red Hat OpenShift Pipelines General Availability 1.6.4	146
3.1.6.8.1. Known issues	146
3.1.6.8.2. Fixed issues	147
3.1.7. Release notes for Red Hat OpenShift Pipelines General Availability 1.5	147
3.1.7.1. Compatibility and support matrix	148
3.1.7.2. New features	148
3.1.7.3. Deprecated features	151
3.1.7.4. Known issues	153
3.1.7.5. Fixed issues	154
3.1.8. Release notes for Red Hat OpenShift Pipelines General Availability 1.4	154
3.1.8.1. Compatibility and support matrix	155
3.1.8.2. New features	155
3.1.8.3. Deprecated features	157
3.1.8.4. Known issues	157
3.1.8.5. Fixed issues	158
3.1.9. Release notes for Red Hat OpenShift Pipelines Technology Preview 1.3	159
3.1.9.1. New features	160
3.1.9.1.1. Pipelines	160
3.1.9.1.2. Pipelines CLI	161
3.1.9.1.3. Triggers	161
3.1.9.2. Deprecated features	162
3.1.9.3. Known issues	162
3.1.9.4. Fixed issues	162
3.1.10. Release notes for Red Hat OpenShift Pipelines Technology Preview 1.2	163
3.1.10.1. New features	163
3.1.10.1.1. Pipelines	164
3.1.10.1.2. Pipelines CLI	164
3.1.10.1.3. Triggers	165
3.1.10.2. Deprecated features	165
3.1.10.3. Known issues	166
3.1.10.4. Fixed issues	167
3.1.11. Release notes for Red Hat OpenShift Pipelines Technology Preview 1.1	167
3.1.11.1. New features	167
3.1.11.1.1. Pipelines	167
3.1.11.1.2. Pipelines CLI	169

3.1.11.1.3. Triggers	169
3.1.11.2. Deprecated features	170
3.1.11.3. Known issues	170
3.1.11.4. Fixed issues	171
3.1.12. Release notes for Red Hat OpenShift Pipelines Technology Preview 1.0	171
3.1.12.1. New features	171
3.1.12.1.1. Pipelines	171
3.1.12.1.2. Pipelines CLI	172
3.1.12.1.3. Triggers	172
3.1.12.2. Deprecated features	172
3.1.12.3. Known issues	173
3.1.12.4. Fixed issues	174
3.2. UNDERSTANDING OPENSIFT PIPELINES	175
3.2.1. Key features	175
3.2.2. OpenShift Pipeline Concepts	175
3.2.2.1. Tasks	175
3.2.2.2. When expression	176
3.2.2.3. Finally tasks	180
3.2.2.4. TaskRun	181
3.2.2.5. Pipelines	182
3.2.2.6. PipelineRun	184
3.2.2.7. Workspaces	185
3.2.2.8. Triggers	187
3.2.3. Additional resources	191
3.3. INSTALLING OPENSIFT PIPELINES	191
Prerequisites	191
3.3.1. Installing the Red Hat OpenShift Pipelines Operator in web console	191
3.3.2. Installing the OpenShift Pipelines Operator using the CLI	193
3.3.3. Red Hat OpenShift Pipelines Operator in a restricted environment	194
3.3.4. Disabling the automatic creation of RBAC resources	194
3.3.5. Additional resources	195
3.4. UNINSTALLING OPENSIFT PIPELINES	195
3.4.1. Deleting the Red Hat OpenShift Pipelines components and Custom Resources	196
3.4.2. Uninstalling the Red Hat OpenShift Pipelines Operator	196
3.5. CREATING CI/CD SOLUTIONS FOR APPLICATIONS USING OPENSIFT PIPELINES	196
3.5.1. Prerequisites	197
3.5.2. Creating a project and checking your pipeline service account	197
3.5.3. Creating pipeline tasks	198
3.5.4. Assembling a pipeline	199
3.5.5. Mirroring images to run pipelines in a restricted environment	201
3.5.6. Running a pipeline	204
3.5.7. Adding triggers to a pipeline	206
3.5.8. Configuring event listeners to serve multiple namespaces	210
3.5.9. Creating webhooks	212
3.5.10. Triggering a pipeline run	213
3.5.11. Enabling monitoring of event listeners for Triggers for user-defined projects	214
3.5.12. Additional resources	215
3.6. MANAGING NON-VERSIONED AND VERSIONED CLUSTER TASKS	215
3.6.1. Differences between non-versioned and versioned cluster tasks	215
3.6.2. Advantages and disadvantages of non-versioned and versioned cluster tasks	216
3.6.3. Disabling non-versioned and versioned cluster tasks	217
3.7. USING TEKTON HUB WITH OPENSIFT PIPELINES	218
3.7.1. Installing and deploying Tekton Hub on a OpenShift Container Platform cluster	218

3.7.1.1. Installing Tekton Hub without login and rating	219
3.7.1.2. Installing Tekton Hub with login and rating	220
3.7.2. Optional: Adding new users in Tekton Hub configuration	222
3.7.3. Optional: Using a custom database in Tekton Hub	223
3.7.4. Disabling Tekton Hub authorization after upgrading the Red Hat OpenShift Pipelines Operator from 1.7 to 1.8	224
3.7.5. Opting out of Tekton Hub in the Developer perspective	226
3.7.6. Additional resources	227
3.8. USING PIPELINES AS CODE	227
3.8.1. Key features	227
3.8.2. Installing Pipelines as Code on an OpenShift Container Platform	227
3.8.3. Installing Pipelines as Code CLI	228
3.8.4. Configuring Pipelines as Code for a Git repository hosting service provider	228
3.8.4.1. Configuring Pipelines as Code for a GitHub App	228
3.8.4.1.1. Configuring a GitHub App	229
3.8.4.1.2. Configuring Pipelines as Code to access a GitHub App	230
3.8.4.2. Creating a GitHub App in administrator perspective	230
3.8.5. Pipelines as Code command reference	231
3.8.5.1. Basic syntax	232
3.8.5.2. Global options	232
3.8.5.3. Utility commands	232
3.8.5.3.1. bootstrap	232
3.8.5.3.2. repository	232
3.8.5.3.3. generate	233
3.8.5.3.4. resolve	233
3.8.6. Customizing Pipelines as Code configuration	234
3.8.7. Additional resources	236
3.9. WORKING WITH RED HAT OPENSIFT PIPELINES USING THE DEVELOPER PERSPECTIVE	236
Prerequisites	236
3.9.1. Constructing Pipelines using the Pipeline Builder	236
3.9.2. Creating OpenShift Pipelines along with applications	240
3.9.3. Adding a GitHub repository containing pipelines	240
3.9.4. Interacting with pipelines using the Developer perspective	243
3.9.5. Using a custom pipeline template for creating and deploying an application from a Git repository	245
3.9.6. Starting pipelines from Pipelines view	247
3.9.7. Starting pipelines from Topology view	249
3.9.8. Interacting with pipelines from Topology view	250
3.9.9. Editing Pipelines	250
3.9.10. Deleting Pipelines	251
3.9.11. Additional resources	251
3.10. REDUCING RESOURCE CONSUMPTION OF OPENSIFT PIPELINES	251
3.10.1. Understanding resource consumption in pipelines	251
3.10.2. Mitigating extra resource consumption in pipelines	252
3.10.3. Additional resources	253
3.11. SETTING COMPUTE RESOURCE QUOTA FOR OPENSIFT PIPELINES	253
3.11.1. Alternative approaches for limiting compute resource consumption in OpenShift Pipelines	254
3.11.2. Specifying pipelines resource quota using priority class	255
3.11.3. Additional resources	258
3.12. AUTOMATIC PRUNING OF TASK RUN AND PIPELINE RUN	258
3.12.1. Default pruner configuration	258
3.12.2. Annotations for automatically pruning task runs and pipeline runs	259
3.12.3. Additional resources	260
3.13. USING PODS IN A PRIVILEGED SECURITY CONTEXT	260

3.13.1. Running pipeline run and task run pods with privileged security context	260
3.13.2. Running pipeline run and task run by using a custom SCC and a custom service account	261
3.13.3. Additional resources	263
3.14. SECURING WEBHOOKS WITH EVENT LISTENERS	263
3.14.1. Providing secure connection with OpenShift routes	264
3.14.2. Creating a sample EventListener resource using a secure HTTPS connection	265
3.15. AUTHENTICATING PIPELINES USING GIT SECRET	265
3.15.1. Credential selection	266
3.15.2. Configuring basic authentication for Git	266
3.15.3. Configuring SSH authentication for Git	268
3.15.4. Using SSH authentication in git type tasks	270
3.15.5. Using secrets as a non-root user	270
3.15.6. Limiting secret access to specific steps	271
3.16. USING TEKTON CHAINS FOR OPENSIFT PIPELINES SUPPLY CHAIN SECURITY	271
3.16.1. Key features	271
3.16.2. Installing Tekton Chains using the Red Hat OpenShift Pipelines Operator	272
3.16.3. Configuring Tekton Chains	272
3.16.3.1. Supported keys for Tekton Chains configuration	273
3.16.3.1.1. Supported keys for task run	273
3.16.3.1.2. Supported keys for OCI	273
3.16.3.1.3. Supported keys for storage	274
3.16.4. Signing secrets in Tekton Chains	274
3.16.4.1. Signing using x509	274
3.16.4.2. Signing using cosign	274
3.16.4.3. Troubleshooting signing	275
3.16.5. Authenticating to an OCI registry	275
3.16.5.1. Creating and verifying task run signatures without any additional authentication	276
3.16.6. Using Tekton Chains to sign and verify image and provenance	277
3.16.7. Additional resources	280
3.17. VIEWING PIPELINE LOGS USING THE OPENSIFT LOGGING OPERATOR	280
3.17.1. Prerequisites	280
3.17.2. Viewing pipeline logs in Kibana	281
3.17.3. Additional resources	283
3.18. UNPRIVILEGED BUILDING OF CONTAINER IMAGES USING BUILDHAH	283
3.18.1. Configuring custom service account and security context constraint	283
3.18.2. Configuring Buildah to use build user	285
3.18.3. Starting a task run with custom config map, or a pipeline run	287
3.18.4. Limitations of unprivileged builds	289
CHAPTER 4. GITOPS	291
4.1. RED HAT OPENSIFT GITOPS RELEASE NOTES	291
4.1.1. Compatibility and support matrix	291
4.1.2. Making open source more inclusive	292
4.1.3. Release notes for Red Hat OpenShift GitOps 1.7.0	292
4.1.3.1. New features	292
4.1.3.2. Fixed issues	293
4.1.3.3. Known issues	294
4.1.4. Release notes for Red Hat OpenShift GitOps 1.6.2	294
4.1.4.1. Fixed issues	294
4.1.5. Release notes for Red Hat OpenShift GitOps 1.6.1	294
4.1.5.1. Fixed issues	295
4.1.6. Release notes for Red Hat OpenShift GitOps 1.6.0	295
4.1.6.1. New features	296

4.1.6.2. Fixed issues	297
4.1.6.3. Known issues	297
4.1.7. Release notes for Red Hat OpenShift GitOps 1.5.7	297
4.1.7.1. Fixed issues	297
4.1.8. Release notes for Red Hat OpenShift GitOps 1.5.6	298
4.1.8.1. Fixed issues	298
4.1.9. Release notes for Red Hat OpenShift GitOps 1.5.5	299
4.1.9.1. New features	299
4.1.9.2. Fixed issues	299
4.1.9.3. Known issues	299
4.1.10. Release notes for Red Hat OpenShift GitOps 1.5.4	299
4.1.10.1. Fixed issues	299
4.1.11. Release notes for Red Hat OpenShift GitOps 1.5.3	299
4.1.11.1. Fixed issues	300
4.1.12. Release notes for Red Hat OpenShift GitOps 1.5.2	300
4.1.12.1. Fixed issues	300
4.1.13. Release notes for Red Hat OpenShift GitOps 1.5.1	300
4.1.13.1. Fixed issues	300
4.1.14. Release notes for Red Hat OpenShift GitOps 1.5.0	300
4.1.14.1. New features	301
4.1.14.2. Fixed issues	301
4.1.14.3. Known issues	302
4.1.15. Release notes for Red Hat OpenShift GitOps 1.4.13	302
4.1.15.1. Fixed issues	302
4.1.16. Release notes for Red Hat OpenShift GitOps 1.4.12	302
4.1.16.1. Fixed issues	302
4.1.17. Release notes for Red Hat OpenShift GitOps 1.4.11	303
4.1.17.1. New features	303
4.1.17.2. Fixed issues	303
4.1.17.3. Known issues	303
4.1.18. Release notes for Red Hat OpenShift GitOps 1.4.6	304
4.1.18.1. Fixed issues	304
4.1.19. Release notes for Red Hat OpenShift GitOps 1.4.5	304
4.1.19.1. Fixed issues	304
4.1.20. Release notes for Red Hat OpenShift GitOps 1.4.3	305
4.1.20.1. Fixed issues	305
4.1.21. Release notes for Red Hat OpenShift GitOps 1.4.2	305
4.1.21.1. Fixed issues	305
4.1.22. Release notes for Red Hat OpenShift GitOps 1.4.1	305
4.1.22.1. Fixed issues	305
4.1.23. Release notes for Red Hat OpenShift GitOps 1.4.0	306
4.1.23.1. New features	306
4.1.23.2. Fixed issues	306
4.1.23.3. Known issues	307
4.1.24. Release notes for Red Hat OpenShift GitOps 1.3.7	307
4.1.24.1. Fixed issues	307
4.1.25. Release notes for Red Hat OpenShift GitOps 1.3.6	307
4.1.25.1. Fixed issues	307
4.1.26. Release notes for Red Hat OpenShift GitOps 1.3.3	308
4.1.26.1. Fixed issues	308
4.1.27. Release notes for Red Hat OpenShift GitOps 1.3.2	308
4.1.27.1. New features	308
4.1.27.2. Fixed issues	308

4.1.28. Release notes for Red Hat OpenShift GitOps 1.3.1	309
4.1.28.1. Fixed issues	309
4.1.29. Release notes for Red Hat OpenShift GitOps 1.3	309
4.1.29.1. New features	309
4.1.29.2. Fixed issues	309
4.1.29.3. Known issues	310
4.1.30. Release notes for Red Hat OpenShift GitOps 1.2.1	310
4.1.30.1. Support matrix	310
4.1.30.2. Fixed issues	310
4.1.31. Release notes for Red Hat OpenShift GitOps 1.2	311
4.1.31.1. Support matrix	311
4.1.31.2. New features	311
4.1.31.3. Fixed issues	313
4.1.31.4. Known issues	313
4.1.32. Release notes for Red Hat OpenShift GitOps 1.1	313
4.1.32.1. Support matrix	313
4.1.32.2. New features	314
4.1.32.3. Fixed issues	314
4.1.32.4. Known issues	315
4.1.32.5. Breaking Change	315
4.1.32.5.1. Upgrading from Red Hat OpenShift GitOps v1.0.1	315
4.2. UNDERSTANDING OPENSIFT GITOPS	316
4.2.1. About GitOps	317
4.2.2. About Red Hat OpenShift GitOps	317
4.2.2.1. Key features	317
4.3. INSTALLING OPENSIFT GITOPS	318
Prerequisites	318
4.3.1. Installing OpenShift GitOps Operator in web console	318
4.3.2. Installing OpenShift GitOps Operator using CLI	318
4.3.3. Logging in to the Argo CD instance by using the Argo CD admin account	319
4.4. UNINSTALLING OPENSIFT GITOPS	320
4.4.1. Deleting the Argo CD instances	320
4.4.2. Uninstalling the GitOps Operator	321
4.5. SETTING UP A NEW ARGO CD INSTANCE	321
4.5.1. Installing Argo CD	321
4.5.2. Enabling replicas for Argo CD server and repo server	322
4.5.3. Deploying resources to a different namespace	322
4.6. CONFIGURING AN OPENSIFT CLUSTER BY DEPLOYING AN APPLICATION WITH CLUSTER CONFIGURATIONS	323
4.6.1. Running the Argo CD instance at the cluster-level	323
4.6.2. Creating an application by using the Argo CD dashboard	324
4.6.3. Creating an application by using the oc tool	325
4.6.4. Synchronizing your application with your Git repository	325
4.6.5. In-built permissions for cluster configuration	326
4.6.6. Adding permissions for cluster configuration	326
4.7. DEPLOYING A SPRING BOOT APPLICATION WITH ARGO CD	327
4.7.1. Creating an application by using the Argo CD dashboard	327
4.7.2. Creating an application by using the oc tool	329
4.7.3. Verifying Argo CD self-healing behavior	329
4.8. ARGO CD OPERATOR	330
4.8.1. Argo CD CLI tool	330
4.8.2. Argo CD custom resource properties	330
4.8.3. Repo server properties	342

4.8.4. Enabling notifications with Argo CD instance	343
4.9. MONITORING HEALTH INFORMATION FOR APPLICATION RESOURCES AND DEPLOYMENTS	344
4.9.1. Checking health information	344
4.10. CONFIGURING SSO FOR ARGO CD USING DEX	344
4.10.1. Enabling the Dex OpenShift OAuth Connector	345
4.10.1.1. Mapping users to specific roles	345
4.10.2. Disabling Dex	346
4.10.3. Enabling or disabling Dex using .spec.sso	346
4.11. CONFIGURING SSO FOR ARGO CD USING KEYCLOAK	346
4.11.1. Configuring a new client in Keycloak	347
4.11.2. Logging in to Keycloak	347
4.11.3. Uninstalling Keycloak	349
4.12. CONFIGURING ARGO CD RBAC	349
4.12.1. Configuring user level access	349
4.12.2. Modifying RHSSO resource requests/limits	350
4.13. CONFIGURING RESOURCE QUOTA OR REQUESTS	350
4.13.1. Configuring workloads with resource requests and limits	350
4.13.2. Patching Argo CD instance to update the resource requirements	352
4.13.3. Removing resource requests	352
4.14. RUNNING GITOPS CONTROL PLANE WORKLOADS ON INFRASTRUCTURE NODES	352
4.14.1. Moving GitOps workloads to infrastructure nodes	352
4.15. SIZING REQUIREMENTS FOR GITOPS OPERATOR	354
4.15.1. Sizing requirements for GitOps	354
CHAPTER 5. JENKINS	356
5.1. CONFIGURING JENKINS IMAGES	356
5.1.1. Configuration and customization	356
5.1.1.1. OpenShift Container Platform OAuth authentication	356
5.1.1.2. Jenkins authentication	357
5.1.2. Jenkins environment variables	358
5.1.3. Providing Jenkins cross project access	361
5.1.4. Jenkins cross volume mount points	362
5.1.5. Customizing the Jenkins image through source-to-image	362
5.1.6. Configuring the Jenkins Kubernetes plugin	363
5.1.7. Jenkins permissions	367
5.1.8. Creating a Jenkins service from a template	368
5.1.9. Using the Jenkins Kubernetes plugin	369
5.1.10. Jenkins memory requirements	372
5.1.11. Additional resources	372
5.2. JENKINS AGENT	372
5.2.1. Jenkins agent images	373
5.2.2. Jenkins agent environment variables	373
5.2.3. Jenkins agent memory requirements	375
5.2.4. Jenkins agent Gradle builds	375
5.2.5. Jenkins agent pod retention	376
5.3. MIGRATING FROM JENKINS TO OPENSIFT PIPELINES OR TEKTON	376
5.3.1. Comparison of Jenkins and OpenShift Pipelines concepts	377
5.3.1.1. Jenkins terminology	377
5.3.1.2. OpenShift Pipelines terminology	377
5.3.1.3. Mapping of concepts	378
5.3.2. Migrating a sample pipeline from Jenkins to OpenShift Pipelines	378
5.3.2.1. Jenkins pipeline	378
5.3.2.2. OpenShift Pipelines pipeline	379

5.3.3. Migrating from Jenkins plugins to Tekton Hub tasks	380
5.3.4. Extending OpenShift Pipelines capabilities using custom tasks and scripts	381
5.3.5. Comparison of Jenkins and OpenShift Pipelines execution models	381
5.3.6. Examples of common use cases	382
5.3.6.1. Running a Maven pipeline in Jenkins and OpenShift Pipelines	382
5.3.6.2. Extending the core capabilities of Jenkins and OpenShift Pipelines by using plugins	385
5.3.6.3. Sharing reusable code in Jenkins and OpenShift Pipelines	385
5.3.7. Additional resources	385
5.4. IMPORTANT CHANGES TO OPENSIFT JENKINS IMAGES	385
5.4.1. Relocation of OpenShift Jenkins images	385
5.4.2. Customizing the Jenkins image stream tag	387
5.4.3. Additional resources	388

CHAPTER 1. OPENSIFT CONTAINER PLATFORM CI/CD OVERVIEW

OpenShift Container Platform is an enterprise-ready Kubernetes platform for developers, which enables organizations to automate the application delivery process through DevOps practices, such as continuous integration (CI) and continuous delivery (CD). To meet your organizational needs, the OpenShift Container Platform provides the following CI/CD solutions:

- OpenShift Builds
- OpenShift Pipelines
- OpenShift GitOps

1.1. OPENSIFT BUILDS

With OpenShift Builds, you can create cloud-native apps by using a declarative build process. You can define the build process in a YAML file that you use to create a BuildConfig object. This definition includes attributes such as build triggers, input parameters, and source code. When deployed, the BuildConfig object typically builds a runnable image and pushes it to a container image registry.

OpenShift Builds provides the following extensible support for build strategies:

- Docker build
- Source-to-image (S2I) build
- Custom build

For more information, see [Understanding image builds](#)

1.2. OPENSIFT PIPELINES

OpenShift Pipelines provides a Kubernetes-native CI/CD framework to design and run each step of the CI/CD pipeline in its own container. It can scale independently to meet the on-demand pipelines with predictable outcomes.

For more information, see [Understanding OpenShift Pipelines](#)

1.3. OPENSIFT GITOPS

OpenShift GitOps is an Operator that uses Argo CD as the declarative GitOps engine. It enables GitOps workflows across multicluster OpenShift and Kubernetes infrastructure. Using OpenShift GitOps, administrators can consistently configure and deploy Kubernetes-based infrastructure and applications across clusters and development lifecycles.

For more information, see [Understanding OpenShift GitOps](#)

1.4. JENKINS

Jenkins automates the process of building, testing, and deploying applications and projects. OpenShift Developer Tools provides a Jenkins image that integrates directly with the OpenShift Container Platform. Jenkins can be deployed on OpenShift by using the Samples Operator templates or certified Helm chart.

CHAPTER 2. BUILDS

2.1. UNDERSTANDING IMAGE BUILDS

2.1.1. Builds

A build is the process of transforming input parameters into a resulting object. Most often, the process is used to transform input parameters or source code into a runnable image. A **BuildConfig** object is the definition of the entire build process.

OpenShift Container Platform uses Kubernetes by creating containers from build images and pushing them to a container image registry.

Build objects share common characteristics including inputs for a build, the requirement to complete a build process, logging the build process, publishing resources from successful builds, and publishing the final status of the build. Builds take advantage of resource restrictions, specifying limitations on resources such as CPU usage, memory usage, and build or pod execution time.

The OpenShift Container Platform build system provides extensible support for build strategies that are based on selectable types specified in the build API. There are three primary build strategies available:

- Docker build
- Source-to-image (S2I) build
- Custom build

By default, docker builds and S2I builds are supported.

The resulting object of a build depends on the builder used to create it. For docker and S2I builds, the resulting objects are runnable images. For custom builds, the resulting objects are whatever the builder image author has specified.

Additionally, the pipeline build strategy can be used to implement sophisticated workflows:

- Continuous integration
- Continuous deployment

2.1.1.1. Docker build

OpenShift Container Platform uses Buildah to build a container image from a Dockerfile. For more information on building container images with Dockerfiles, see [the Dockerfile reference documentation](#).

TIP

If you set Docker build arguments by using the **buildArgs** array, see [Understand how ARG and FROM interact](#) in the Dockerfile reference documentation.

2.1.1.2. Source-to-image build

Source-to-image (S2I) is a tool for building reproducible container images. It produces ready-to-run images by injecting application source into a container image and assembling a new image. The new image incorporates the base image, the builder, and built source and is ready to use with the **buildah**

run command. S2I supports incremental builds, which re-use previously downloaded dependencies, previously built artifacts, and so on.

2.1.1.3. Custom build

The custom build strategy allows developers to define a specific builder image responsible for the entire build process. Using your own builder image allows you to customize your build process.

A custom builder image is a plain container image embedded with build process logic, for example for building RPMs or base images.

Custom builds run with a high level of privilege and are not available to users by default. Only users who can be trusted with cluster administration permissions should be granted access to run custom builds.

2.1.1.4. Pipeline build



IMPORTANT

The Pipeline build strategy is deprecated in OpenShift Container Platform 4. Equivalent and improved functionality is present in the OpenShift Container Platform Pipelines based on Tekton.

Jenkins images on OpenShift Container Platform are fully supported and users should follow Jenkins user documentation for defining their **jenkinsfile** in a job or store it in a Source Control Management system.

The Pipeline build strategy allows developers to define a Jenkins pipeline for use by the Jenkins pipeline plugin. The build can be started, monitored, and managed by OpenShift Container Platform in the same way as any other build type.

Pipeline workflows are defined in a **jenkinsfile**, either embedded directly in the build configuration, or supplied in a Git repository and referenced by the build configuration.

2.2. UNDERSTANDING BUILD CONFIGURATIONS

The following sections define the concept of a build, build configuration, and outline the primary build strategies available.

2.2.1. BuildConfigs

A build configuration describes a single build definition and a set of triggers for when a new build is created. Build configurations are defined by a **BuildConfig**, which is a REST object that can be used in a POST to the API server to create a new instance.

A build configuration, or **BuildConfig**, is characterized by a build strategy and one or more sources. The strategy determines the process, while the sources provide its input.

Depending on how you choose to create your application using OpenShift Container Platform, a **BuildConfig** is typically generated automatically for you if you use the web console or CLI, and it can be edited at any time. Understanding the parts that make up a **BuildConfig** and their available options can help if you choose to manually change your configuration later.

The following example **BuildConfig** results in a new build every time a container image tag or the source code changes:

BuildConfig object definition

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: "ruby-sample-build" ❶
spec:
  runPolicy: "Serial" ❷
  triggers: ❸
  -
    type: "GitHub"
    github:
      secret: "secret101"
  - type: "Generic"
    generic:
      secret: "secret101"
  -
    type: "ImageChange"
source: ❹
  git:
    uri: "https://github.com/openshift/ruby-hello-world"
strategy: ❺
  sourceStrategy:
    from:
      kind: "ImageStreamTag"
      name: "ruby-20-centos7:latest"
output: ❻
  to:
    kind: "ImageStreamTag"
    name: "origin-ruby-sample:latest"
postCommit: ❼
  script: "bundle exec rake test"
```

- ❶ This specification creates a new **BuildConfig** named **ruby-sample-build**.
- ❷ The **runPolicy** field controls whether builds created from this build configuration can be run simultaneously. The default value is **Serial**, which means new builds run sequentially, not simultaneously.
- ❸ You can specify a list of triggers, which cause a new build to be created.
- ❹ The **source** section defines the source of the build. The source type determines the primary source of input, and can be either **Git**, to point to a code repository location, **Dockerfile**, to build from an inline Dockerfile, or **Binary**, to accept binary payloads. It is possible to have multiple sources at once. See the documentation for each source type for details.
- ❺ The **strategy** section describes the build strategy used to execute the build. You can specify a **Source**, **Docker**, or **Custom** strategy here. This example uses the **ruby-20-centos7** container image that Source-to-image (S2I) uses for the application build.
- ❻ After the container image is successfully built, it is pushed into the repository described in the **output** section.
- ❼ The **postCommit** section defines an optional build hook.

2.3. CREATING BUILD INPUTS

Use the following sections for an overview of build inputs, instructions on how to use inputs to provide source content for builds to operate on, and how to use build environments and create secrets.

2.3.1. Build inputs

A build input provides source content for builds to operate on. You can use the following build inputs to provide sources in OpenShift Container Platform, listed in order of precedence:

- Inline Dockerfile definitions
- Content extracted from existing images
- Git repositories
- Binary (Local) inputs
- Input secrets
- External artifacts

You can combine multiple inputs in a single build. However, as the inline Dockerfile takes precedence, it can overwrite any other file named Dockerfile provided by another input. Binary (local) input and Git repositories are mutually exclusive inputs.

You can use input secrets when you do not want certain resources or credentials used during a build to be available in the final application image produced by the build, or want to consume a value that is defined in a secret resource. External artifacts can be used to pull in additional files that are not available as one of the other build input types.

When you run a build:

1. A working directory is constructed and all input content is placed in the working directory. For example, the input Git repository is cloned into the working directory, and files specified from input images are copied into the working directory using the target path.
2. The build process changes directories into the **contextDir**, if one is defined.
3. The inline Dockerfile, if any, is written to the current directory.
4. The content from the current directory is provided to the build process for reference by the Dockerfile, custom builder logic, or **assemble** script. This means any input content that resides outside the **contextDir** is ignored by the build.

The following example of a source definition includes multiple input types and an explanation of how they are combined. For more details on how each input type is defined, see the specific sections for each input type.

```
source:
  git:
    uri: https://github.com/openshift/ruby-hello-world.git 1
    ref: "master"
  images:
    - from:
        kind: ImageStreamTag
```

```

name: myinputimage:latest
namespace: mynamespace
paths:
- destinationDir: app/dir/injected/dir ❷
  sourcePath: /usr/lib/somefile.jar
contextDir: "app/dir" ❸
dockerfile: "FROM centos:7\nRUN yum install -y httpd" ❹

```

- ❶ The repository to be cloned into the working directory for the build.
- ❷ `/usr/lib/somefile.jar` from **myinputimage** is stored in `<workingdir>/app/dir/injected/dir`.
- ❸ The working directory for the build becomes `<original_workingdir>/app/dir`.
- ❹ A Dockerfile with this content is created in `<original_workingdir>/app/dir`, overwriting any existing file with that name.

2.3.2. Dockerfile source

When you supply a **dockerfile** value, the content of this field is written to disk as a file named **dockerfile**. This is done after other input sources are processed, so if the input source repository contains a Dockerfile in the root directory, it is overwritten with this content.

The source definition is part of the **spec** section in the **BuildConfig**:

```

source:
  dockerfile: "FROM centos:7\nRUN yum install -y httpd" ❶

```

- ❶ The **dockerfile** field contains an inline Dockerfile that is built.

Additional resources

- The typical use for this field is to provide a Dockerfile to a docker strategy build.

2.3.3. Image source

You can add additional files to the build process with images. Input images are referenced in the same way the **From** and **To** image targets are defined. This means both container images and image stream tags can be referenced. In conjunction with the image, you must provide one or more path pairs to indicate the path of the files or directories to copy the image and the destination to place them in the build context.

The source path can be any absolute path within the image specified. The destination must be a relative directory path. At build time, the image is loaded and the indicated files and directories are copied into the context directory of the build process. This is the same directory into which the source repository content is cloned. If the source path ends in `/.` then the content of the directory is copied, but the directory itself is not created at the destination.

Image inputs are specified in the **source** definition of the **BuildConfig**:

```

source:
  git:

```

```

uri: https://github.com/openshift/ruby-hello-world.git
ref: "master"
images: ❶
- from: ❷
  kind: ImageStreamTag
  name: myinputimage:latest
  namespace: mynamespace
paths: ❸
- destinationDir: injected/dir ❹
  sourcePath: /usr/lib/somefile.jar ❺
- from:
  kind: ImageStreamTag
  name: myotherinputimage:latest
  namespace: myothernamespace
pullSecret: mysecret ❻
paths:
- destinationDir: injected/dir
  sourcePath: /usr/lib/somefile.jar

```

- ❶ An array of one or more input images and files.
- ❷ A reference to the image containing the files to be copied.
- ❸ An array of source/destination paths.
- ❹ The directory relative to the build root where the build process can access the file.
- ❺ The location of the file to be copied out of the referenced image.
- ❻ An optional secret provided if credentials are needed to access the input image.



NOTE

If your cluster uses an **ImageContentSourcePolicy** object to configure repository mirroring, you can use only global pull secrets for mirrored registries. You cannot add a pull secret to a project.

Images that require pull secrets

When using an input image that requires a pull secret, you can link the pull secret to the service account used by the build. By default, builds use the **builder** service account. The pull secret is automatically added to the build if the secret contains a credential that matches the repository hosting the input image. To link a pull secret to the service account used by the build, run:

```
$ oc secrets link builder dockerhub
```



NOTE

This feature is not supported for builds using the custom strategy.

Images on mirrored registries that require pull secrets

When using an input image from a mirrored registry, if you get a **build error: failed to pull image** message, you can resolve the error by using either of the following methods:

- Create an input secret that contains the authentication credentials for the builder image's repository and all known mirrors. In this case, create a pull secret for credentials to the image registry and its mirrors.
- Use the input secret as the pull secret on the **BuildConfig** object.

2.3.4. Git source

When specified, source code is fetched from the supplied location.

If you supply an inline Dockerfile, it overwrites the Dockerfile in the **contextDir** of the Git repository.

The source definition is part of the **spec** section in the **BuildConfig**:

```
source:
  git: ❶
    uri: "https://github.com/openshift/ruby-hello-world"
    ref: "master"
  contextDir: "app/dir" ❷
  dockerfile: "FROM openshift/ruby-22-centos7\nUSER example" ❸
```

- ❶ The **git** field contains the URI to the remote Git repository of the source code. Optionally, specify the **ref** field to check out a specific Git reference. A valid **ref** can be a SHA1 tag or a branch name.
- ❷ The **contextDir** field allows you to override the default location inside the source code repository where the build looks for the application source code. If your application exists inside a sub-directory, you can override the default location (the root folder) using this field.
- ❸ If the optional **dockerfile** field is provided, it should be a string containing a Dockerfile that overwrites any Dockerfile that may exist in the source repository.

If the **ref** field denotes a pull request, the system uses a **git fetch** operation and then checkout **FETCH_HEAD**.

When no **ref** value is provided, OpenShift Container Platform performs a shallow clone (**--depth=1**). In this case, only the files associated with the most recent commit on the default branch (typically **master**) are downloaded. This results in repositories downloading faster, but without the full commit history. To perform a full **git clone** of the default branch of a specified repository, set **ref** to the name of the default branch (for example **master**).



WARNING

Git clone operations that go through a proxy that is performing man in the middle (MITM) TLS hijacking or reencrypting of the proxied connection do not work.

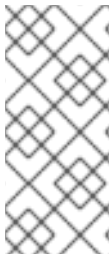
2.3.4.1. Using a proxy

If your Git repository can only be accessed using a proxy, you can define the proxy to use in the **source** section of the build configuration. You can configure both an HTTP and HTTPS proxy to use. Both fields are optional. Domains for which no proxying should be performed can also be specified in the **NoProxy** field.

**NOTE**

Your source URI must use the HTTP or HTTPS protocol for this to work.

```
source:
  git:
    uri: "https://github.com/openshift/ruby-hello-world"
    ref: "master"
  httpProxy: http://proxy.example.com
  httpsProxy: https://proxy.example.com
  noProxy: somedomain.com, otherdomain.com
```

**NOTE**

For Pipeline strategy builds, given the current restrictions with the Git plugin for Jenkins, any Git operations through the Git plugin do not leverage the HTTP or HTTPS proxy defined in the **BuildConfig**. The Git plugin only uses the proxy configured in the Jenkins UI at the Plugin Manager panel. This proxy is then used for all git interactions within Jenkins, across all jobs.

Additional resources

- You can find instructions on how to configure proxies through the Jenkins UI at [JenkinsBehindProxy](#).

2.3.4.2. Source Clone Secrets

Builder pods require access to any Git repositories defined as source for a build. Source clone secrets are used to provide the builder pod with access it would not normally have access to, such as private repositories or repositories with self-signed or untrusted SSL certificates.

The following source clone secret configurations are supported:

- .gitconfig File
- Basic Authentication
- SSH Key Authentication
- Trusted Certificate Authorities

**NOTE**

You can also use combinations of these configurations to meet your specific needs.

2.3.4.2.1. Automatically adding a source clone secret to a build configuration

When a **BuildConfig** is created, OpenShift Container Platform can automatically populate its source clone secret reference. This behavior allows the resulting builds to automatically use the credentials

stored in the referenced secret to authenticate to a remote Git repository, without requiring further configuration.

To use this functionality, a secret containing the Git repository credentials must exist in the namespace in which the **BuildConfig** is later created. This secret must include one or more annotations prefixed with **build.openshift.io/source-secret-match-uri-**. The value of each of these annotations is a Uniform Resource Identifier (URI) pattern, which is defined as follows. When a **BuildConfig** is created without a source clone secret reference and its Git source URI matches a URI pattern in a secret annotation, OpenShift Container Platform automatically inserts a reference to that secret in the **BuildConfig**.

Prerequisites

A URI pattern must consist of:

- A valid scheme: ***://**, **git://**, **http://**, **https://** or **ssh://**
- A host: ***`** or a valid hostname or IP address optionally preceded by *****.
- A path: **/*** or **/** followed by any characters optionally including ***** characters

In all of the above, a ***** character is interpreted as a wildcard.

IMPORTANT

URI patterns must match Git source URIs which are conformant to [RFC3986](#). Do not include a username (or password) component in a URI pattern.

For example, if you use **ssh://git@bitbucket.atlassian.com:7999/ATLASSIAN jira.git** for a git repository URL, the source secret must be specified as **ssh://bitbucket.atlassian.com:7999/*** (and not **ssh://git@bitbucket.atlassian.com:7999/***).

```
$ oc annotate secret mysecret \
    'build.openshift.io/source-secret-match-uri-1=ssh://bitbucket.atlassian.com:7999/*'
```

Procedure

If multiple secrets match the Git URI of a particular **BuildConfig**, OpenShift Container Platform selects the secret with the longest match. This allows for basic overriding, as in the following example.

The following fragment shows two partial source clone secrets, the first matching any server in the domain **mycorp.com** accessed by HTTPS, and the second overriding access to servers **mydev1.mycorp.com** and **mydev2.mycorp.com**:

```
kind: Secret
apiVersion: v1
metadata:
  name: matches-all-corporate-servers-https-only
  annotations:
    build.openshift.io/source-secret-match-uri-1: https://*.mycorp.com/*
data:
  ...
---
kind: Secret
apiVersion: v1
metadata:
```

```

name: override-for-my-dev-servers-https-only
annotations:
  build.openshift.io/source-secret-match-uri-1: https://mydev1.mycorp.com/*
  build.openshift.io/source-secret-match-uri-2: https://mydev2.mycorp.com/*
data:
  ...

```

- Add a **build.openshift.io/source-secret-match-uri-** annotation to a pre-existing secret using:

```

$ oc annotate secret mysecret \
  'build.openshift.io/source-secret-match-uri-1=https://*.mycorp.com/*'

```

2.3.4.2.2. Manually adding a source clone secret

Source clone secrets can be added manually to a build configuration by adding a **sourceSecret** field to the **source** section inside the **BuildConfig** and setting it to the name of the secret that you created. In this example, it is the **basicsecret**.

```

apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  output:
    to:
      kind: "ImageStreamTag"
      name: "sample-image:latest"
  source:
    git:
      uri: "https://github.com/user/app.git"
    sourceSecret:
      name: "basicsecret"
  strategy:
    sourceStrategy:
      from:
        kind: "ImageStreamTag"
        name: "python-33-centos7:latest"

```

Procedure

You can also use the **oc set build-secret** command to set the source clone secret on an existing build configuration.

- To set the source clone secret on an existing build configuration, enter the following command:

```

$ oc set build-secret --source bc/sample-build basicsecret

```

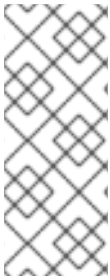
2.3.4.2.3. Creating a secret from a .gitconfig file

If the cloning of your application is dependent on a **.gitconfig** file, then you can create a secret that contains it. Add it to the builder service account and then your **BuildConfig**.

Procedure

- To create a secret from a **.gitconfig** file:

```
$ oc create secret generic <secret_name> --from-file=<path/to/.gitconfig>
```



NOTE

SSL verification can be turned off if **sslVerify=false** is set for the **http** section in your **.gitconfig** file:

```
[http]
    sslVerify=false
```

2.3.4.2.4. Creating a secret from a .gitconfig file for secured Git

If your Git server is secured with two-way SSL and user name with password, you must add the certificate files to your source build and add references to the certificate files in the **.gitconfig** file.

Prerequisites

- You must have Git credentials.

Procedure

Add the certificate files to your source build and add references to the certificate files in the **.gitconfig** file.

1. Add the **client.crt**, **cacert.crt**, and **client.key** files to the **/var/run/secrets/openshift.io/source/** folder in the application source code.
2. In the **.gitconfig** file for the server, add the **[http]** section shown in the following example:

```
# cat .gitconfig
```

Example output

```
[user]
  name = <name>
  email = <email>
[http]
  sslVerify = false
  sslCert = /var/run/secrets/openshift.io/source/client.crt
  sslKey = /var/run/secrets/openshift.io/source/client.key
  sslCaInfo = /var/run/secrets/openshift.io/source/cacert.crt
```

3. Create the secret:

```
$ oc create secret generic <secret_name> \
--from-literal=username=<user_name> \ 1
--from-literal=password=<password> \ 2
--from-file=.gitconfig=.gitconfig \
--from-file=client.crt=/var/run/secrets/openshift.io/source/client.crt \
--from-file=cacert.crt=/var/run/secrets/openshift.io/source/cacert.crt \
--from-file=client.key=/var/run/secrets/openshift.io/source/client.key
```

- 1 The user's Git user name.
- 2 The password for this user.



IMPORTANT

To avoid having to enter your password again, be sure to specify the source-to-image (S2I) image in your builds. However, if you cannot clone the repository, you must still specify your user name and password to promote the build.

Additional resources

- `/var/run/secrets/openshift.io/source/` folder in the application source code.

2.3.4.2.5. Creating a secret from source code basic authentication

Basic authentication requires either a combination of **--username** and **--password**, or a token to authenticate against the software configuration management (SCM) server.

Prerequisites

- User name and password to access the private repository.

Procedure

1. Create the secret first before using the **--username** and **--password** to access the private repository:

```
$ oc create secret generic <secret_name> \
  --from-literal=username=<user_name> \
  --from-literal=password=<password> \
  --type=kubernetes.io/basic-auth
```

2. Create a basic authentication secret with a token:

```
$ oc create secret generic <secret_name> \
  --from-literal=password=<token> \
  --type=kubernetes.io/basic-auth
```

2.3.4.2.6. Creating a secret from source code SSH key authentication

SSH key based authentication requires a private SSH key.

The repository keys are usually located in the `$HOME/.ssh/` directory, and are named **id_dsa.pub**, **id_ecdsa.pub**, **id_ed25519.pub**, or **id_rsa.pub** by default.

Procedure

1. Generate SSH key credentials:

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
```

**NOTE**

Creating a passphrase for the SSH key prevents OpenShift Container Platform from building. When prompted for a passphrase, leave it blank.

Two files are created: the public key and a corresponding private key (one of **id_dsa**, **id_ecdsa**, **id_ed25519**, or **id_rsa**). With both of these in place, consult your source control management (SCM) system's manual on how to upload the public key. The private key is used to access your private repository.

2. Before using the SSH key to access the private repository, create the secret:

```
$ oc create secret generic <secret_name> \
  --from-file=ssh-privatekey=<path/to/ssh/private/key> \
  --from-file=<path/to/known_hosts> \ 1
  --type=kubernetes.io/ssh-auth
```

- 1** Optional: Adding this field enables strict server host key check.

**WARNING**

Skipping the **known_hosts** file while creating the secret makes the build vulnerable to a potential man-in-the-middle (MITM) attack.

**NOTE**

Ensure that the **known_hosts** file includes an entry for the host of your source code.

2.3.4.2.7. Creating a secret from source code trusted certificate authorities

The set of Transport Layer Security (TLS) certificate authorities (CA) that are trusted during a Git clone operation are built into the OpenShift Container Platform infrastructure images. If your Git server uses a self-signed certificate or one signed by an authority not trusted by the image, you can create a secret that contains the certificate or disable TLS verification.

If you create a secret for the CA certificate, OpenShift Container Platform uses it to access your Git server during the Git clone operation. Using this method is significantly more secure than disabling Git SSL verification, which accepts any TLS certificate that is presented.

Procedure

Create a secret with a CA certificate file.

1. If your CA uses Intermediate Certificate Authorities, combine the certificates for all CAs in a **ca.crt** file. Enter the following command:

```
$ cat intermediateCA.crt intermediateCA.crt rootCA.crt > ca.crt
```

- a. Create the secret:

```
$ oc create secret generic mycert --from-file=ca.crt=</path/to/file> 1
```

- 1 You must use the key name **ca.crt**.

2.3.4.2.8. Source secret combinations

You can combine the different methods for creating source clone secrets for your specific needs.

2.3.4.2.8.1. Creating a SSH-based authentication secret with a **gitconfig** file

You can combine the different methods for creating source clone secrets for your specific needs, such as a SSH-based authentication secret with a **.gitconfig** file.

Prerequisites

- SSH authentication
- **.gitconfig** file

Procedure

- To create a SSH-based authentication secret with a **.gitconfig** file, run:

```
$ oc create secret generic <secret_name> \
  --from-file=ssh-privatekey=<path/to/ssh/private/key> \
  --from-file=<path/to/.gitconfig> \
  --type=kubernetes.io/ssh-auth
```

2.3.4.2.8.2. Creating a secret that combines a **.gitconfig** file and CA certificate

You can combine the different methods for creating source clone secrets for your specific needs, such as a secret that combines a **.gitconfig** file and certificate authority (CA) certificate.

Prerequisites

- **.gitconfig** file
- CA certificate

Procedure

- To create a secret that combines a **.gitconfig** file and CA certificate, run:

```
$ oc create secret generic <secret_name> \
  --from-file=ca.crt=<path/to/certificate> \
  --from-file=<path/to/.gitconfig>
```

2.3.4.2.8.3. Creating a basic authentication secret with a CA certificate

You can combine the different methods for creating source clone secrets for your specific needs, such as a secret that combines a basic authentication and certificate authority (CA) certificate.

Prerequisites

- Basic authentication credentials
- CA certificate

Procedure

- Create a basic authentication secret with a CA certificate, run:

```
$ oc create secret generic <secret_name> \
  --from-literal=username=<user_name> \
  --from-literal=password=<password> \
  --from-file=ca-cert=</path/to/file> \
  --type=kubernetes.io/basic-auth
```

2.3.4.2.8.4. Creating a basic authentication secret with a `.gitconfig` file

You can combine the different methods for creating source clone secrets for your specific needs, such as a secret that combines a basic authentication and `.gitconfig` file.

Prerequisites

- Basic authentication credentials
- `.gitconfig` file

Procedure

- To create a basic authentication secret with a `.gitconfig` file, run:

```
$ oc create secret generic <secret_name> \
  --from-literal=username=<user_name> \
  --from-literal=password=<password> \
  --from-file=</path/to/.gitconfig> \
  --type=kubernetes.io/basic-auth
```

2.3.4.2.8.5. Creating a basic authentication secret with a `.gitconfig` file and CA certificate

You can combine the different methods for creating source clone secrets for your specific needs, such as a secret that combines a basic authentication, `.gitconfig` file, and certificate authority (CA) certificate.

Prerequisites

- Basic authentication credentials
- `.gitconfig` file
- CA certificate

Procedure

- To create a basic authentication secret with a **.gitconfig** file and CA certificate, run:

```
$ oc create secret generic <secret_name> \
  --from-literal=username=<user_name> \
  --from-literal=password=<password> \
  --from-file=</path/to/.gitconfig> \
  --from-file=ca-cert=</path/to/file> \
  --type=kubernetes.io/basic-auth
```

2.3.5. Binary (local) source

Streaming content from a local file system to the builder is called a **Binary** type build. The corresponding value of **BuildConfig.spec.source.type** is **Binary** for these builds.

This source type is unique in that it is leveraged solely based on your use of the **oc start-build**.



NOTE

Binary type builds require content to be streamed from the local file system, so automatically triggering a binary type build, like an image change trigger, is not possible. This is because the binary files cannot be provided. Similarly, you cannot launch binary type builds from the web console.

To utilize binary builds, invoke **oc start-build** with one of these options:

- from-file:** The contents of the file you specify are sent as a binary stream to the builder. You can also specify a URL to a file. Then, the builder stores the data in a file with the same name at the top of the build context.
- from-dir** and **--from-repo:** The contents are archived and sent as a binary stream to the builder. Then, the builder extracts the contents of the archive within the build context directory. With **--from-dir**, you can also specify a URL to an archive, which is extracted.
- from-archive:** The archive you specify is sent to the builder, where it is extracted within the build context directory. This option behaves the same as **--from-dir**; an archive is created on your host first, whenever the argument to these options is a directory.

In each of the previously listed cases:

- If your **BuildConfig** already has a **Binary** source type defined, it is effectively ignored and replaced by what the client sends.
- If your **BuildConfig** has a **Git** source type defined, it is dynamically disabled, since **Binary** and **Git** are mutually exclusive, and the data in the binary stream provided to the builder takes precedence.

Instead of a file name, you can pass a URL with HTTP or HTTPS schema to **--from-file** and **--from-archive**. When using **--from-file** with a URL, the name of the file in the builder image is determined by the **Content-Disposition** header sent by the web server, or the last component of the URL path if the header is not present. No form of authentication is supported and it is not possible to use custom TLS certificate or disable certificate validation.

When using **oc new-build --binary=true**, the command ensures that the restrictions associated with

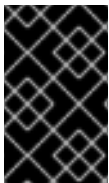
binary builds are enforced. The resulting **BuildConfig** has a source type of **Binary**, meaning that the only valid way to run a build for this **BuildConfig** is to use **oc start-build** with one of the **--from** options to provide the requisite binary data.

The Dockerfile and **contextDir** source options have special meaning with binary builds.

Dockerfile can be used with any binary build source. If Dockerfile is used and the binary stream is an archive, its contents serve as a replacement Dockerfile to any Dockerfile in the archive. If Dockerfile is used with the **--from-file** argument, and the file argument is named Dockerfile, the value from Dockerfile replaces the value from the binary stream.

In the case of the binary stream encapsulating extracted archive content, the value of the **contextDir** field is interpreted as a subdirectory within the archive, and, if valid, the builder changes into that subdirectory before executing the build.

2.3.6. Input secrets and config maps



IMPORTANT

To prevent the contents of input secrets and config maps from appearing in build output container images, use build volumes in your [Docker build](#) and [source-to-image build](#) strategies.

In some scenarios, build operations require credentials or other configuration data to access dependent resources, but it is undesirable for that information to be placed in source control. You can define input secrets and input config maps for this purpose.

For example, when building a Java application with Maven, you can set up a private mirror of Maven Central or JCenter that is accessed by private keys. To download libraries from that private mirror, you have to supply the following:

1. A **settings.xml** file configured with the mirror's URL and connection settings.
2. A private key referenced in the settings file, such as `~/.ssh/id_rsa`.

For security reasons, you do not want to expose your credentials in the application image.

This example describes a Java application, but you can use the same approach for adding SSL certificates into the **/etc/ssl/certs** directory, API keys or tokens, license files, and more.

2.3.6.1. What is a secret?

The **Secret** object type provides a mechanism to hold sensitive information such as passwords, OpenShift Container Platform client configuration files, **dockercfg** files, private source repository credentials, and so on. Secrets decouple sensitive content from the pods. You can mount secrets into containers using a volume plugin or the system can use secrets to perform actions on behalf of a pod.

YAML Secret Object Definition

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
  namespace: my-namespace
type: Opaque 1
```

```
data: ❷
  username: dmFsdWUtMQ0K ❸
  password: dmFsdWUtMg0KDQo=
stringData: ❹
  hostname: myapp.mydomain.com ❺
```

- ❶ Indicates the structure of the secret's key names and values.
- ❷ The allowable format for the keys in the **data** field must meet the guidelines in the **DNS_SUBDOMAIN** value in the Kubernetes identifiers glossary.
- ❸ The value associated with keys in the **data** map must be base64 encoded.
- ❹ Entries in the **stringData** map are converted to base64 and the entry are then moved to the **data** map automatically. This field is write-only. The value is only be returned by the **data** field.
- ❺ The value associated with keys in the **stringData** map is made up of plain text strings.

2.3.6.1.1. Properties of secrets

Key properties include:

- Secret data can be referenced independently from its definition.
- Secret data volumes are backed by temporary file-storage facilities (tmpfs) and never come to rest on a node.
- Secret data can be shared within a namespace.

2.3.6.1.2. Types of Secrets

The value in the **type** field indicates the structure of the secret's key names and values. The type can be used to enforce the presence of user names and keys in the secret object. If you do not want validation, use the **opaque** type, which is the default.

Specify one of the following types to trigger minimal server-side validation to ensure the presence of specific key names in the secret data:

- **kubernetes.io/service-account-token**. Uses a service account token.
- **kubernetes.io/dockercfg**. Uses the **.dockercfg** file for required Docker credentials.
- **kubernetes.io/dockerconfigjson**. Uses the **.docker/config.json** file for required Docker credentials.
- **kubernetes.io/basic-auth**. Use with basic authentication.
- **kubernetes.io/ssh-auth**. Use with SSH key authentication.
- **kubernetes.io/tls**. Use with TLS certificate authorities.

Specify **type= Opaque** if you do not want validation, which means the secret does not claim to conform to any convention for key names or values. An **opaque** secret, allows for unstructured **key:value** pairs that can contain arbitrary values.



NOTE

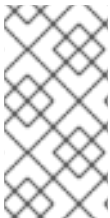
You can specify other arbitrary types, such as **example.com/my-secret-type**. These types are not enforced server-side, but indicate that the creator of the secret intended to conform to the key/value requirements of that type.

2.3.6.1.3. Updates to secrets

When you modify the value of a secret, the value used by an already running pod does not dynamically change. To change a secret, you must delete the original pod and create a new pod, in some cases with an identical **PodSpec**.

Updating a secret follows the same workflow as deploying a new container image. You can use the **kubectrl rolling-update** command.

The **resourceVersion** value in a secret is not specified when it is referenced. Therefore, if a secret is updated at the same time as pods are starting, the version of the secret that is used for the pod is not defined.



NOTE

Currently, it is not possible to check the resource version of a secret object that was used when a pod was created. It is planned that pods report this information, so that a controller could restart ones using an old **resourceVersion**. In the interim, do not update the data of existing secrets, but create new ones with distinct names.

2.3.6.2. Creating secrets

You must create a secret before creating the pods that depend on that secret.

When creating secrets:

- Create a secret object with secret data.
- Update the pod service account to allow the reference to the secret.
- Create a pod, which consumes the secret as an environment variable or as a file using a **secret** volume.

Procedure

- Use the create command to create a secret object from a JSON or YAML file:

```
$ oc create -f <filename>
```

For example, you can create a secret from your local **.docker/config.json** file:

```
$ oc create secret generic dockerhub \
  --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
  --type=kubernetes.io/dockerconfigjson
```

This command generates a JSON specification of the secret named **dockerhub** and creates the object.

YAML Opaque Secret Object Definition

```

apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque ❶
data:
  username: dXNlci1uYW1l
  password: cGFzc3dvcmQ=

```

- ❶ Specifies an *opaque* secret.

Docker Configuration JSON File Secret Object Definition

```

apiVersion: v1
kind: Secret
metadata:
  name: aregistrykey
  namespace: myapps
type: kubernetes.io/dockerconfigjson ❶
data:
  .dockerconfigjson:bm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2cg
  YXV0aCBrZXlzcG== ❷

```

- ❶ Specifies that the secret is using a docker configuration JSON file.
- ❷ The output of a base64-encoded the docker configuration JSON file

2.3.6.3. Using secrets

After creating secrets, you can create a pod to reference your secret, get logs, and delete the pod.

Procedure

1. Create the pod to reference your secret:

```
$ oc create -f <your_yaml_file>.yaml
```

2. Get the logs:

```
$ oc logs secret-example-pod
```

3. Delete the pod:

```
$ oc delete pod secret-example-pod
```

Additional resources

- Example YAML files with secret data:

YAML Secret That Will Create Four Files

```

apiVersion: v1
kind: Secret
metadata:
  name: test-secret
data:
  username: dmFsdWUtMQ0K 1
  password: dmFsdWUtMQ0KDQo= 2
stringData:
  hostname: myapp.mydomain.com 3
secret.properties: |- 4
  property1=valueA
  property2=valueB

```

- 1** File contains decoded values.
- 2** File contains decoded values.
- 3** File contains the provided string.
- 4** File contains the provided data.

YAML of a pod populating files in a volume with secret data

```

apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "cat /etc/secret-volume/*" ]
      volumeMounts:
        # name must match the volume name below
        - name: secret-volume
          mountPath: /etc/secret-volume
          readOnly: true
  volumes:
    - name: secret-volume
      secret:
        secretName: test-secret
  restartPolicy: Never

```

YAML of a pod populating environment variables with secret data

```

apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
    - name: secret-test-container
      image: busybox

```

```

command: [ "/bin/sh", "-c", "export" ]
env:
  - name: TEST_SECRET_USERNAME_ENV_VAR
    valueFrom:
      secretKeyRef:
        name: test-secret
        key: username
restartPolicy: Never

```

YAML of a Build Config Populating Environment Variables with Secret Data

```

apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
        - name: TEST_SECRET_USERNAME_ENV_VAR
          valueFrom:
            secretKeyRef:
              name: test-secret
              key: username

```

2.3.6.4. Adding input secrets and config maps

To provide credentials and other configuration data to a build without placing them in source control, you can define input secrets and input config maps.

In some scenarios, build operations require credentials or other configuration data to access dependent resources. To make that information available without placing it in source control, you can define input secrets and input config maps.

Procedure

To add an input secret, config maps, or both to an existing **BuildConfig** object:

1. Create the **ConfigMap** object, if it does not exist:

```

$ oc create configmap settings-mvn \
  --from-file=settings.xml=<path/to/settings.xml>

```

This creates a new config map named **settings-mvn**, which contains the plain text content of the **settings.xml** file.

TIP

You can alternatively apply the following YAML to create the config map:

```
apiVersion: core/v1
kind: ConfigMap
metadata:
  name: settings-mvn
data:
  settings.xml: |
    <settings>
    ... # Insert maven settings here
    </settings>
```

2. Create the **Secret** object, if it does not exist:

```
$ oc create secret generic secret-mvn \
  --from-file=ssh-privatekey=<path/to/.ssh/id_rsa>
  --type=kubernetes.io/ssh-auth
```

This creates a new secret named **secret-mvn**, which contains the base64 encoded content of the **id_rsa** private key.

TIP

You can alternatively apply the following YAML to create the input secret:

```
apiVersion: core/v1
kind: Secret
metadata:
  name: secret-mvn
type: kubernetes.io/ssh-auth
data:
  ssh-privatekey: |
    # Insert ssh private key, base64 encoded
```

3. Add the config map and secret to the **source** section in the existing **BuildConfig** object:

```
source:
  git:
    uri: https://github.com/wildfly/quickstart.git
    contextDir: helloworld
  configMaps:
    - configMap:
        name: settings-mvn
  secrets:
    - secret:
        name: secret-mvn
```

To include the secret and config map in a new **BuildConfig** object, run the following command:

```
$ oc new-build \
  openshift/wildfly-101-centos7~https://github.com/wildfly/quickstart.git \
```

```
--context-dir helloworld --build-secret "secret-mvn" \
--build-config-map "settings-mvn"
```

During the build, the **settings.xml** and **id_rsa** files are copied into the directory where the source code is located. In OpenShift Container Platform S2I builder images, this is the image working directory, which is set using the **WORKDIR** instruction in the **Dockerfile**. If you want to specify another directory, add a **destinationDir** to the definition:

```
source:
  git:
    uri: https://github.com/wildfly/quickstart.git
  contextDir: helloworld
  configMaps:
    - configMap:
        name: settings-mvn
        destinationDir: ".m2"
  secrets:
    - secret:
        name: secret-mvn
        destinationDir: ".ssh"
```

You can also specify the destination directory when creating a new **BuildConfig** object:

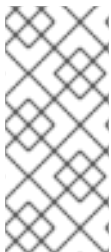
```
$ oc new-build \
  openshift/wildfly-101-centos7~https://github.com/wildfly/quickstart.git \
  --context-dir helloworld --build-secret "secret-mvn:.ssh" \
  --build-config-map "settings-mvn:.m2"
```

In both cases, the **settings.xml** file is added to the **./m2** directory of the build environment, and the **id_rsa** key is added to the **./ssh** directory.

2.3.6.5. Source-to-image strategy

When using a **Source** strategy, all defined input secrets are copied to their respective **destinationDir**. If you left **destinationDir** empty, then the secrets are placed in the working directory of the builder image.

The same rule is used when a **destinationDir** is a relative path. The secrets are placed in the paths that are relative to the working directory of the image. The final directory in the **destinationDir** path is created if it does not exist in the builder image. All preceding directories in the **destinationDir** must exist, or an error will occur.



NOTE

Input secrets are added as world-writable, have **0666** permissions, and are truncated to size zero after executing the **assemble** script. This means that the secret files exist in the resulting image, but they are empty for security reasons.

Input config maps are not truncated after the **assemble** script completes.

2.3.6.6. Docker strategy

When using a **docker** strategy, you can add all defined input secrets into your container image using the **ADD** and **COPY** instructions in your **Dockerfile**.

If you do not specify the **destinationDir** for a secret, then the files are copied into the same directory in which the Dockerfile is located. If you specify a relative path as **destinationDir**, then the secrets are copied into that directory, relative to your Dockerfile location. This makes the secret files available to the Docker build operation as part of the context directory used during the build.

Example of a Dockerfile referencing secret and config map data

```
FROM centos/ruby-22-centos7

USER root
COPY ./secret-dir /secrets
COPY ./config /

# Create a shell script that will output secrets and ConfigMaps when the image is run
RUN echo '#!/bin/sh' > /input_report.sh
RUN echo '(test -f /secrets/secret1 && echo -n "secret1=" && cat /secrets/secret1)' >> /input_report.sh
RUN echo '(test -f /config && echo -n "relative-configMap=" && cat /config)' >> /input_report.sh
RUN chmod 755 /input_report.sh

CMD ["/bin/sh", "-c", "/input_report.sh"]
```

IMPORTANT

Users normally remove their input secrets from the final application image so that the secrets are not present in the container running from that image. However, the secrets still exist in the image itself in the layer where they were added. This removal is part of the Dockerfile itself.

To prevent the contents of input secrets and config maps from appearing in the build output container images and avoid this removal process altogether, [use build volumes](#) in your Docker build strategy instead.

2.3.6.7. Custom strategy

When using a Custom strategy, all the defined input secrets and config maps are available in the builder container in the **/var/run/secrets/openshift.io/build** directory. The custom build image must use these secrets and config maps appropriately. With the Custom strategy, you can define secrets as described in Custom strategy options.

There is no technical difference between existing strategy secrets and the input secrets. However, your builder image can distinguish between them and use them differently, based on your build use case.

The input secrets are always mounted into the **/var/run/secrets/openshift.io/build** directory, or your builder can parse the **\$BUILD** environment variable, which includes the full build object.

IMPORTANT

If a pull secret for the registry exists in both the namespace and the node, builds default to using the pull secret in the namespace.

2.3.7. External artifacts

It is not recommended to store binary files in a source repository. Therefore, you must define a build which pulls additional files, such as Java **.jar** dependencies, during the build process. How this is done depends on the build strategy you are using.

For a Source build strategy, you must put appropriate shell commands into the **assemble** script:

.s2i/bin/assemble File

```
#!/bin/sh
APP_VERSION=1.0
wget http://repository.example.com/app/app-$APP_VERSION.jar -O app.jar
```

.s2i/bin/run File

```
#!/bin/sh
exec java -jar app.jar
```

For a Docker build strategy, you must modify the Dockerfile and invoke shell commands with the **RUN** instruction:

Excerpt of Dockerfile

```
FROM jboss/base-jdk:8

ENV APP_VERSION 1.0
RUN wget http://repository.example.com/app/app-$APP_VERSION.jar -O app.jar

EXPOSE 8080
CMD [ "java", "-jar", "app.jar" ]
```

In practice, you may want to use an environment variable for the file location so that the specific file to be downloaded can be customized using an environment variable defined on the **BuildConfig**, rather than updating the Dockerfile or **assemble** script.

You can choose between different methods of defining environment variables:

- Using the **.s2i/environment** file] (only for a Source build strategy)
- Setting in **BuildConfig**
- Providing explicitly using **oc start-build --env** (only for builds that are triggered manually)

2.3.8. Using docker credentials for private registries

You can supply builds with a **.docker/config.json** file with valid credentials for private container registries. This allows you to push the output image into a private container image registry or pull a builder image from the private container image registry that requires authentication.

You can supply credentials for multiple repositories within the same registry, each with credentials specific to that registry path.



NOTE

For the OpenShift Container Platform container image registry, this is not required because secrets are generated automatically for you by OpenShift Container Platform.

The **.docker/config.json** file is found in your home directory by default and has the following format:

```
auths:
  https://index.docker.io/v1/: 1
    auth: "YWRfbGZhcGU6R2labnRib21ifTE=" 2
    email: "user@example.com" 3
  https://docker.io/my-namespace/my-user/my-image: 4
    auth: "GzhYWRGU6R2fbclabnRgbkSp="
    email: "user@example.com"
  https://docker.io/my-namespace: 5
    auth: "GzhYWRGU6R2deesfrRgbkSp="
    email: "user@example.com"
```

- 1 URL of the registry.
- 2 Encrypted password.
- 3 Email address for the login.
- 4 URL and credentials for a specific image in a namespace.
- 5 URL and credentials for a registry namespace.

You can define multiple container image registries or define multiple repositories in the same registry. Alternatively, you can also add authentication entries to this file by running the **docker login** command. The file will be created if it does not exist.

Kubernetes provides **Secret** objects, which can be used to store configuration and passwords.

Prerequisites

- You must have a **.docker/config.json** file.

Procedure

1. Create the secret from your local **.docker/config.json** file:

```
$ oc create secret generic dockerhub \
  --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
  --type=kubernetes.io/dockerconfigjson
```

This generates a JSON specification of the secret named **dockerhub** and creates the object.

2. Add a **pushSecret** field into the **output** section of the **BuildConfig** and set it to the name of the **secret** that you created, which in the previous example is **dockerhub**:

```
spec:
  output:
    to:
```

```
kind: "DockerImage"
name: "private.registry.com/org/private-image:latest"
pushSecret:
  name: "dockerhub"
```

You can use the **oc set build-secret** command to set the push secret on the build configuration:

```
$ oc set build-secret --push bc/sample-build dockerhub
```

You can also link the push secret to the service account used by the build instead of specifying the **pushSecret** field. By default, builds use the **builder** service account. The push secret is automatically added to the build if the secret contains a credential that matches the repository hosting the build's output image.

```
$ oc secrets link builder dockerhub
```

3. Pull the builder container image from a private container image registry by specifying the **pullSecret** field, which is part of the build strategy definition:

```
strategy:
  sourceStrategy:
    from:
      kind: "DockerImage"
      name: "docker.io/user/private_repository"
    pullSecret:
      name: "dockerhub"
```

You can use the **oc set build-secret** command to set the pull secret on the build configuration:

```
$ oc set build-secret --pull bc/sample-build dockerhub
```



NOTE

This example uses **pullSecret** in a Source build, but it is also applicable in Docker and Custom builds.

You can also link the pull secret to the service account used by the build instead of specifying the **pullSecret** field. By default, builds use the **builder** service account. The pull secret is automatically added to the build if the secret contains a credential that matches the repository hosting the build's input image. To link the pull secret to the service account used by the build instead of specifying the **pullSecret** field, run:

```
$ oc secrets link builder dockerhub
```



NOTE

You must specify a **from** image in the **BuildConfig** spec to take advantage of this feature. Docker strategy builds generated by **oc new-build** or **oc new-app** may not do this in some situations.

2.3.9. Build environments

As with pod environment variables, build environment variables can be defined in terms of references to other resources or variables using the Downward API. There are some exceptions, which are noted.

You can also manage environment variables defined in the **BuildConfig** with the **oc set env** command.



NOTE

Referencing container resources using **valueFrom** in build environment variables is not supported as the references are resolved before the container is created.

2.3.9.1. Using build fields as environment variables

You can inject information about the build object by setting the **fieldPath** environment variable source to the **JsonPath** of the field from which you are interested in obtaining the value.



NOTE

Jenkins Pipeline strategy does not support **valueFrom** syntax for environment variables.

Procedure

- Set the **fieldPath** environment variable source to the **JsonPath** of the field from which you are interested in obtaining the value:

```
env:
  - name: FIELDREF_ENV
    valueFrom:
      fieldRef:
        fieldPath: metadata.name
```

2.3.9.2. Using secrets as environment variables

You can make key values from secrets available as environment variables using the **valueFrom** syntax.



IMPORTANT

This method shows the secrets as plain text in the output of the build pod console. To avoid this, use input secrets and config maps instead.

Procedure

- To use a secret as an environment variable, set the **valueFrom** syntax:

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
        - name: MYVAL
          valueFrom:
```

```
secretKeyRef:
  key: myval
  name: mysecret
```

Additional resources

- [Input secrets and config maps](#)

2.3.10. Service serving certificate secrets

Service serving certificate secrets are intended to support complex middleware applications that need out-of-the-box certificates. It has the same settings as the server certificates generated by the administrator tooling for nodes and masters.

Procedure

To secure communication to your service, have the cluster generate a signed serving certificate/key pair into a secret in your namespace.

- Set the **service.beta.openshift.io/serving-cert-secret-name** annotation on your service with the value set to the name you want to use for your secret. Then, your **PodSpec** can mount that secret. When it is available, your pod runs. The certificate is good for the internal service DNS name, **<service.name>.<service.namespace>.svc**.

The certificate and key are in PEM format, stored in **tls.crt** and **tls.key** respectively. The certificate/key pair is automatically replaced when it gets close to expiration. View the expiration date in the **service.beta.openshift.io/expiry** annotation on the secret, which is in RFC3339 format.



NOTE

In most cases, the service DNS name **<service.name>.<service.namespace>.svc** is not externally routable. The primary use of **<service.name>.<service.namespace>.svc** is for intracluster or intraservice communication, and with re-encrypt routes.

Other pods can trust cluster-created certificates, which are only signed for internal DNS names, by using the certificate authority (CA) bundle in the **/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt** file that is automatically mounted in their pod.

The signature algorithm for this feature is **x509.SHA256WithRSA**. To manually rotate, delete the generated secret. A new certificate is created.

2.3.11. Secrets restrictions

To use a secret, a pod needs to reference the secret. A secret can be used with a pod in three ways:

- To populate environment variables for containers.
- As files in a volume mounted on one or more of its containers.
- By kubelet when pulling images for the pod.

Volume type secrets write data into the container as a file using the volume mechanism.

imagePullSecrets use service accounts for the automatic injection of the secret into all pods in a namespaces.

When a template contains a secret definition, the only way for the template to use the provided secret is to ensure that the secret volume sources are validated and that the specified object reference actually points to an object of type **Secret**. Therefore, a secret needs to be created before any pods that depend on it. The most effective way to ensure this is to have it get injected automatically through the use of a service account.

Secret API objects reside in a namespace. They can only be referenced by pods in that same namespace.

Individual secrets are limited to 1MB in size. This is to discourage the creation of large secrets that would exhaust apiserver and kubelet memory. However, creation of a number of smaller secrets could also exhaust memory.

2.4. MANAGING BUILD OUTPUT

Use the following sections for an overview of and instructions for managing build output.

2.4.1. Build output

Builds that use the docker or source-to-image (S2I) strategy result in the creation of a new container image. The image is then pushed to the container image registry specified in the **output** section of the **Build** specification.

If the output kind is **ImageStreamTag**, then the image will be pushed to the integrated OpenShift Container Platform registry and tagged in the specified imagestream. If the output is of type **DockerImage**, then the name of the output reference will be used as a docker push specification. The specification may contain a registry or will default to DockerHub if no registry is specified. If the output section of the build specification is empty, then the image will not be pushed at the end of the build.

Output to an ImageStreamTag

```
spec:
  output:
    to:
      kind: "ImageStreamTag"
      name: "sample-image:latest"
```

Output to a docker Push Specification

```
spec:
  output:
    to:
      kind: "DockerImage"
      name: "my-registry.mycompany.com:5000/myimages/myimage:tag"
```

2.4.2. Output image environment variables

docker and source-to-image (S2I) strategy builds set the following environment variables on output images:

Variable	Description
OPENSIFT_BUILD_NAME	Name of the build
OPENSIFT_BUILD_NAMESPACE	Namespace of the build
OPENSIFT_BUILD_SOURCE	The source URL of the build
OPENSIFT_BUILD_REFERENCE	The Git reference used in the build
OPENSIFT_BUILD_COMMIT	Source commit used in the build

Additionally, any user-defined environment variable, for example those configured with S2I or docker strategy options, will also be part of the output image environment variable list.

2.4.3. Output image labels

docker and source-to-image (S2I) builds set the following labels on output images:

Label	Description
io.openshift.build.commit.author	Author of the source commit used in the build
io.openshift.build.commit.date	Date of the source commit used in the build
io.openshift.build.commit.id	Hash of the source commit used in the build
io.openshift.build.commit.message	Message of the source commit used in the build
io.openshift.build.commit.ref	Branch or reference specified in the source
io.openshift.build.source-location	Source URL for the build

You can also use the **BuildConfig.spec.output.imageLabels** field to specify a list of custom labels that will be applied to each image built from the build configuration.

Custom Labels to be Applied to Built Images

```
spec:
  output:
    to:
      kind: "ImageStreamTag"
      name: "my-image:latest"
    imageLabels:
      - name: "vendor"
        value: "MyCompany"
      - name: "authoritative-source-url"
        value: "registry.mycompany.com"
```

2.5. USING BUILD STRATEGIES

The following sections define the primary supported build strategies, and how to use them.

2.5.1. Docker build

OpenShift Container Platform uses Buildah to build a container image from a Dockerfile. For more information on building container images with Dockerfiles, see [the Dockerfile reference documentation](#).

TIP

If you set Docker build arguments by using the **buildArgs** array, see [Understand how ARG and FROM interact](#) in the Dockerfile reference documentation.

2.5.1.1. Replacing Dockerfile FROM image

You can replace the **FROM** instruction of the Dockerfile with the **from** of the **BuildConfig** object. If the Dockerfile uses multi-stage builds, the image in the last **FROM** instruction will be replaced.

Procedure

To replace the **FROM** instruction of the Dockerfile with the **from** of the **BuildConfig**,

```
strategy:
  dockerStrategy:
    from:
      kind: "ImageStreamTag"
      name: "debian:latest"
```

2.5.1.2. Using Dockerfile path

By default, docker builds use a Dockerfile located at the root of the context specified in the **BuildConfig.spec.source.contextDir** field.

The **dockerfilePath** field allows the build to use a different path to locate your Dockerfile, relative to the **BuildConfig.spec.source.contextDir** field. It can be a different file name than the default Dockerfile, such as **MyDockerfile**, or a path to a Dockerfile in a subdirectory, such as **dockerfiles/app1/Dockerfile**.

Procedure

To use the **dockerfilePath** field for the build to use a different path to locate your Dockerfile, set:

```
strategy:
  dockerStrategy:
    dockerfilePath: dockerfiles/app1/Dockerfile
```

2.5.1.3. Using docker environment variables

To make environment variables available to the docker build process and resulting image, you can add environment variables to the **dockerStrategy** definition of the build configuration.

The environment variables defined there are inserted as a single **ENV** Dockerfile instruction right after the **FROM** instruction, so that it can be referenced later on within the Dockerfile.

Procedure

The variables are defined during build and stay in the output image, therefore they will be present in any container that runs that image as well.

For example, defining a custom HTTP proxy to be used during build and runtime:

```
dockerStrategy:
...
env:
- name: "HTTP_PROXY"
  value: "http://myproxy.net:5187/"
```

You can also manage environment variables defined in the build configuration with the **oc set env** command.

2.5.1.4. Adding docker build arguments

You can set [docker build arguments](#) using the **buildArgs** array. The build arguments are passed to docker when a build is started.

TIP

See [Understand how ARG and FROM interact](#) in the Dockerfile reference documentation.

Procedure

To set docker build arguments, add entries to the **buildArgs** array, which is located in the **dockerStrategy** definition of the **BuildConfig** object. For example:

```
dockerStrategy:
...
buildArgs:
- name: "foo"
  value: "bar"
```



NOTE

Only the **name** and **value** fields are supported. Any settings on the **valueFrom** field are ignored.

2.5.1.5. Squashing layers with docker builds

Docker builds normally create a layer representing each instruction in a Dockerfile. Setting the **imageOptimizationPolicy** to **SkipLayers** merges all instructions into a single layer on top of the base image.

Procedure

- Set the **imageOptimizationPolicy** to **SkipLayers**:

```
strategy:
  dockerStrategy:
    imageOptimizationPolicy: SkipLayers
```

2.5.1.6. Using build volumes

You can mount build volumes to give running builds access to information that you don't want to persist in the output container image.

Build volumes provide sensitive information, such as repository credentials, that the build environment or configuration only needs at build time. Build volumes are different from [build inputs](#), whose data can persist in the output container image.

The mount points of build volumes, from which the running build reads data, are functionally similar to [pod volume mounts](#).

Prerequisites

- You have [added an input secret, config map, or both to a BuildConfig object](#) .

Procedure

- In the **dockerStrategy** definition of the **BuildConfig** object, add any build volumes to the **volumes** array. For example:

```
spec:
  dockerStrategy:
    volumes:
      - name: secret-mvn 1
        mounts:
          - destinationPath: /opt/app-root/src/.ssh 2
            source:
              type: Secret 3
              secret:
                secretName: my-secret 4
        - name: settings-mvn 5
          mounts:
            - destinationPath: /opt/app-root/src/.m2 6
              source:
                type: ConfigMap 7
                configMap:
                  name: my-config 8
        - name: my-csi-volume 9
          mounts:
            - destinationPath: /opt/app-root/src/some_path 10
              source:
                type: CSI 11
                csi:
                  driver: csi.sharedresource.openshift.io 12
                  readOnly: true 13
                  volumeAttributes: 14
                    attribute: value
```

1 5 9 Required. A unique name.

2 6 10

Required. The absolute path of the mount point. It must not contain `..` or `:` and doesn't collide with the destination path generated by the builder. The `/opt/app-root/src` is the

- 3 7 11** Required. The type of source, **ConfigMap**, **Secret**, or **CSI**.
- 4 8** Required. The name of the source.
- 12** Required. The driver that provides the ephemeral CSI volume.
- 13** Required. This value must be set to **true**. Provides a read-only volume.
- 14** Optional. The volume attributes of the ephemeral CSI volume. Consult the CSI driver's documentation for supported attribute keys and values.



NOTE

The Shared Resource CSI Driver is supported as a Technology Preview feature.

2.5.2. Source-to-image build

Source-to-image (S2I) is a tool for building reproducible container images. It produces ready-to-run images by injecting application source into a container image and assembling a new image. The new image incorporates the base image, the builder, and built source and is ready to use with the **buildah run** command. S2I supports incremental builds, which re-use previously downloaded dependencies, previously built artifacts, and so on.

2.5.2.1. Performing source-to-image incremental builds

Source-to-image (S2I) can perform incremental builds, which means it reuses artifacts from previously-built images.

Procedure

- To create an incremental build, create a with the following modification to the strategy definition:

```
strategy:
  sourceStrategy:
    from:
      kind: "ImageStreamTag"
      name: "incremental-image:latest" 1
    incremental: true 2
```

- 1** Specify an image that supports incremental builds. Consult the documentation of the builder image to determine if it supports this behavior.
- 2** This flag controls whether an incremental build is attempted. If the builder image does not support incremental builds, the build will still succeed, but you will get a log message stating the incremental build was not successful because of a missing **save-artifacts** script.

Additional resources

- See S2I Requirements for information on how to create a builder image supporting incremental builds.

2.5.2.2. Overriding source-to-image builder image scripts

You can override the **assemble**, **run**, and **save-artifacts** source-to-image (S2I) scripts provided by the builder image.

Procedure

To override the **assemble**, **run**, and **save-artifacts** S2I scripts provided by the builder image, either:

- Provide an **assemble**, **run**, or **save-artifacts** script in the **.s2i/bin** directory of your application source repository.
- Provide a URL of a directory containing the scripts as part of the strategy definition. For example:

```
strategy:
  sourceStrategy:
    from:
      kind: "ImageStreamTag"
      name: "builder-image:latest"
      scripts: "http://somehost.com/scripts_directory" 1
```

- 1 This path will have **run**, **assemble**, and **save-artifacts** appended to it. If any or all scripts are found they will be used in place of the same named scripts provided in the image.



NOTE

Files located at the **scripts** URL take precedence over files located in **.s2i/bin** of the source repository.

2.5.2.3. Source-to-image environment variables

There are two ways to make environment variables available to the source build process and resulting image. Environment files and BuildConfig environment values. Variables provided will be present during the build process and in the output image.

2.5.2.3.1. Using source-to-image environment files

Source build enables you to set environment values, one per line, inside your application, by specifying them in a **.s2i/environment** file in the source repository. The environment variables specified in this file are present during the build process and in the output image.

If you provide a **.s2i/environment** file in your source repository, source-to-image (S2I) reads this file during the build. This allows customization of the build behavior as the **assemble** script may use these variables.

Procedure

For example, to disable assets compilation for your Rails application during the build:

- Add **DISABLE_ASSET_COMPILATION=true** in the **.s2i/environment** file.

In addition to builds, the specified environment variables are also available in the running application itself. For example, to cause the Rails application to start in **development** mode instead of **production**:

- Add **RAILS_ENV=development** to the **.s2i/environment** file.

The complete list of supported environment variables is available in the using images section for each image.

2.5.2.3.2. Using source-to-image build configuration environment

You can add environment variables to the **sourceStrategy** definition of the build configuration. The environment variables defined there are visible during the **assemble** script execution and will be defined in the output image, making them also available to the **run** script and application code.

Procedure

- For example, to disable assets compilation for your Rails application:

```
sourceStrategy:
...
env:
  - name: "DISABLE_ASSET_COMPILATION"
    value: "true"
```

Additional resources

- The build environment section provides more advanced instructions.
- You can also manage environment variables defined in the build configuration with the **oc set env** command.

2.5.2.4. Ignoring source-to-image source files

Source-to-image (S2I) supports a **.s2iignore** file, which contains a list of file patterns that should be ignored. Files in the build working directory, as provided by the various input sources, that match a pattern found in the **.s2iignore** file will not be made available to the **assemble** script.

2.5.2.5. Creating images from source code with source-to-image

Source-to-image (S2I) is a framework that makes it easy to write images that take application source code as an input and produce a new image that runs the assembled application as output.

The main advantage of using S2I for building reproducible container images is the ease of use for developers. As a builder image author, you must understand two basic concepts in order for your images to provide the best S2I performance, the build process and S2I scripts.

2.5.2.5.1. Understanding the source-to-image build process

The build process consists of the following three fundamental elements, which are combined into a final container image:

- Sources
- Source-to-image (S2I) scripts

- Builder image

S2I generates a Dockerfile with the builder image as the first **FROM** instruction. The Dockerfile generated by S2I is then passed to Buildah.

2.5.2.5.2. How to write source-to-image scripts

You can write source-to-image (S2I) scripts in any programming language, as long as the scripts are executable inside the builder image. S2I supports multiple options providing **assemble/run/save-artifacts** scripts. All of these locations are checked on each build in the following order:


1. A script specified in the build configuration.
2. A script found in the application source **.s2i/bin** directory.
3. A script found at the default image URL with the **io.openshift.s2i.scripts-url** label.

Both the **io.openshift.s2i.scripts-url** label specified in the image and the script specified in a build configuration can take one of the following forms:

- **image:///path_to_scripts_dir**: absolute path inside the image to a directory where the S2I scripts are located.
- **file:///path_to_scripts_dir**: relative or absolute path to a directory on the host where the S2I scripts are located.
- **http(s)://path_to_scripts_dir**: URL to a directory where the S2I scripts are located.

Table 2.1. S2I scripts

Script	Description
assemble	<p>The assemble script builds the application artifacts from a source and places them into appropriate directories inside the image. This script is required. The workflow for this script is:</p> <ol style="list-style-type: none"> 1. Optional: Restore build artifacts. If you want to support incremental builds, make sure to define save-artifacts as well. 2. Place the application source in the desired location. 3. Build the application artifacts. 4. Install the artifacts into locations appropriate for them to run.
run	The run script executes your application. This script is required.
save-artifacts	<p>The save-artifacts script gathers all dependencies that can speed up the build processes that follow. This script is optional. For example:</p> <ul style="list-style-type: none"> • For Ruby, gems installed by Bundler. • For Java, .m2 contents. <p>These dependencies are gathered into a tar file and streamed to the standard output.</p>

Script	Description
usage	The usage script allows you to inform the user how to properly use your image. This script is optional.
test/run	<p>The test/run script allows you to create a process to check if the image is working correctly. This script is optional. The proposed flow of that process is:</p> <ol style="list-style-type: none"> 1. Build the image. 2. Run the image to verify the usage script. 3. Run s2i build to verify the assemble script. 4. Optional: Run s2i build again to verify the save-artifacts and assemble scripts save and restore artifacts functionality. 5. Run the image to verify the test application is working. <div>  <div> <p>NOTE</p> <p>The suggested location to put the test application built by your test/run script is the test/test-app directory in your image repository.</p> </div> </div>

Example S2I scripts

The following example S2I scripts are written in Bash. Each example assumes its **tar** contents are unpacked into the **/tmp/s2i** directory.

assemble script:

```
#!/bin/bash

# restore build artifacts
if [ "$(ls /tmp/s2i/artifacts/ 2>/dev/null)" ]; then
    mv /tmp/s2i/artifacts/* $HOME/.
fi

# move the application source
mv /tmp/s2i/src $HOME/src

# build application artifacts
pushd ${HOME}
make all

# install the artifacts
make install
popd
```

run script:

```
#!/bin/bash
```

```
# run the application
/opt/application/run.sh
```

save-artifacts script:

```
#!/bin/bash

pushd ${HOME}
if [ -d deps ]; then
    # all deps contents to tar stream
    tar cf - deps
fi
popd
```

usage script:

```
#!/bin/bash

# inform the user how to use the image
cat <<EOF
This is a S2I sample builder image, to use it, install
https://github.com/openshift/source-to-image
EOF
```

Additional resources

- [S2I Image Creation Tutorial](#)

2.5.2.6. Using build volumes

You can mount build volumes to give running builds access to information that you don't want to persist in the output container image.

Build volumes provide sensitive information, such as repository credentials, that the build environment or configuration only needs at build time. Build volumes are different from [build inputs](#), whose data can persist in the output container image.

The mount points of build volumes, from which the running build reads data, are functionally similar to [pod volume mounts](#).

Prerequisites

- You have [added an input secret, config map, or both to a BuildConfig object](#) .

Procedure

- In the **sourceStrategy** definition of the **BuildConfig** object, add any build volumes to the **volumes** array. For example:

```
spec:
  sourceStrategy:
    volumes:
      - name: secret-mvn 1
```

```

mounts:
- destinationPath: /opt/app-root/src/.ssh 2
source:
  type: Secret 3
  secret:
    secretName: my-secret 4
- name: settings-mvn 5
  mounts:
  - destinationPath: /opt/app-root/src/.m2 6
  source:
    type: ConfigMap 7
    configMap:
      name: my-config 8
- name: my-csi-volume 9
  mounts:
  - destinationPath: /opt/app-root/src/some_path 10
  source:
    type: CSI 11
    csi:
      driver: csi.sharedresource.openshift.io 12
      readOnly: true 13
      volumeAttributes: 14
        attribute: value

```

1 5 9 Required. A unique name.

2 6 10 Required. The absolute path of the mount point. It must not contain `..` or `:` and doesn't collide with the destination path generated by the builder. The `/opt/app-root/src` is the default home directory for many Red Hat S2I-enabled images.

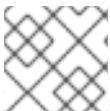
3 7 11 Required. The type of source, **ConfigMap**, **Secret**, or **CSI**.

4 8 Required. The name of the source.

12 Required. The driver that provides the ephemeral CSI volume.

13 Required. This value must be set to **true**. Provides a read-only volume.

14 Optional. The volume attributes of the ephemeral CSI volume. Consult the CSI driver's documentation for supported attribute keys and values.



NOTE

The Shared Resource CSI Driver is supported as a Technology Preview feature.

2.5.3. Custom build

The custom build strategy allows developers to define a specific builder image responsible for the entire build process. Using your own builder image allows you to customize your build process.

A custom builder image is a plain container image embedded with build process logic, for example for building RPMs or base images.

Custom builds run with a high level of privilege and are not available to users by default. Only users who can be trusted with cluster administration permissions should be granted access to run custom builds.

2.5.3.1. Using FROM image for custom builds

You can use the **customStrategy.from** section to indicate the image to use for the custom build

Procedure

- Set the **customStrategy.from** section:

```
strategy:
  customStrategy:
    from:
      kind: "DockerImage"
      name: "openshift/sti-image-builder"
```

2.5.3.2. Using secrets in custom builds

In addition to secrets for source and images that can be added to all build types, custom strategies allow adding an arbitrary list of secrets to the builder pod.

Procedure

- To mount each secret at a specific location, edit the **secretSource** and **mountPath** fields of the **strategy** YAML file:

```
strategy:
  customStrategy:
    secrets:
      - secretSource: 1
        name: "secret1"
        mountPath: "/tmp/secret1" 2
      - secretSource:
        name: "secret2"
        mountPath: "/tmp/secret2"
```

1 **secretSource** is a reference to a secret in the same namespace as the build.

2 **mountPath** is the path inside the custom builder where the secret should be mounted.

2.5.3.3. Using environment variables for custom builds

To make environment variables available to the custom build process, you can add environment variables to the **customStrategy** definition of the build configuration.

The environment variables defined there are passed to the pod that runs the custom build.

Procedure

1. Define a custom HTTP proxy to be used during build:

```
customStrategy:
```

```
...
env:
  - name: "HTTP_PROXY"
    value: "http://myproxy.net:5187/"
```

2. To manage environment variables defined in the build configuration, enter the following command:

```
$ oc set env <enter_variables>
```

2.5.3.4. Using custom builder images

OpenShift Container Platform's custom build strategy enables you to define a specific builder image responsible for the entire build process. When you need a build to produce individual artifacts such as packages, JARs, WARs, installable ZIPs, or base images, use a custom builder image using the custom build strategy.

A custom builder image is a plain container image embedded with build process logic, which is used for building artifacts such as RPMs or base container images.

Additionally, the custom builder allows implementing any extended build process, such as a CI/CD flow that runs unit or integration tests.

2.5.3.4.1. Custom builder image

Upon invocation, a custom builder image receives the following environment variables with the information needed to proceed with the build:

Table 2.2. Custom Builder Environment Variables

Variable Name	Description
BUILD	The entire serialized JSON of the Build object definition. If you must use a specific API version for serialization, you can set the buildAPIVersion parameter in the custom strategy specification of the build configuration.
SOURCE_REPOSITORY	The URL of a Git repository with source to be built.
SOURCE_URI	Uses the same value as SOURCE_REPOSITORY . Either can be used.
SOURCE_CONTEXT_DIR	Specifies the subdirectory of the Git repository to be used when building. Only present if defined.
SOURCE_REF	The Git reference to be built.
ORIGIN_VERSION	The version of the OpenShift Container Platform master that created this build object.
OUTPUT_REGISTRY	The container image registry to push the image to.

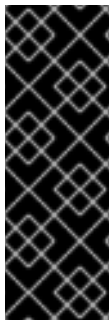
Variable Name	Description
OUTPUT_IMAGE	The container image tag name for the image being built.
PUSH_DOCKERCFG_PATH	The path to the container registry credentials for running a podman push operation.

2.5.3.4.2. Custom builder workflow

Although custom builder image authors have flexibility in defining the build process, your builder image must adhere to the following required steps necessary for running a build inside of OpenShift Container Platform:

1. The **Build** object definition contains all the necessary information about input parameters for the build.
2. Run the build process.
3. If your build produces an image, push it to the output location of the build if it is defined. Other output locations can be passed with environment variables.

2.5.4. Pipeline build



IMPORTANT

The Pipeline build strategy is deprecated in OpenShift Container Platform 4. Equivalent and improved functionality is present in the OpenShift Container Platform Pipelines based on Tekton.

Jenkins images on OpenShift Container Platform are fully supported and users should follow Jenkins user documentation for defining their **jenkinsfile** in a job or store it in a Source Control Management system.

The Pipeline build strategy allows developers to define a Jenkins pipeline for use by the Jenkins pipeline plugin. The build can be started, monitored, and managed by OpenShift Container Platform in the same way as any other build type.

Pipeline workflows are defined in a **jenkinsfile**, either embedded directly in the build configuration, or supplied in a Git repository and referenced by the build configuration.

2.5.4.1. Understanding OpenShift Container Platform pipelines



IMPORTANT

The Pipeline build strategy is deprecated in OpenShift Container Platform 4. Equivalent and improved functionality is present in the OpenShift Container Platform Pipelines based on Tekton.

Jenkins images on OpenShift Container Platform are fully supported and users should follow Jenkins user documentation for defining their **jenkinsfile** in a job or store it in a Source Control Management system.

Pipelines give you control over building, deploying, and promoting your applications on OpenShift Container Platform. Using a combination of the Jenkins Pipeline build strategy, **jenkinsfiles**, and the OpenShift Container Platform Domain Specific Language (DSL) provided by the Jenkins Client Plugin, you can create advanced build, test, deploy, and promote pipelines for any scenario.

OpenShift Container Platform Jenkins Sync Plugin

The OpenShift Container Platform Jenkins Sync Plugin keeps the build configuration and build objects in sync with Jenkins jobs and builds, and provides the following:

- Dynamic job and run creation in Jenkins.
- Dynamic creation of agent pod templates from image streams, image stream tags, or config maps.
- Injection of environment variables.
- Pipeline visualization in the OpenShift Container Platform web console.
- Integration with the Jenkins Git plugin, which passes commit information from OpenShift Container Platform builds to the Jenkins Git plugin.
- Synchronization of secrets into Jenkins credential entries.

OpenShift Container Platform Jenkins Client Plugin

The OpenShift Container Platform Jenkins Client Plugin is a Jenkins plugin which aims to provide a readable, concise, comprehensive, and fluent Jenkins Pipeline syntax for rich interactions with an OpenShift Container Platform API Server. The plugin uses the OpenShift Container Platform command line tool, **oc**, which must be available on the nodes executing the script.

The Jenkins Client Plugin must be installed on your Jenkins master so the OpenShift Container Platform DSL will be available to use within the **jenkinsfile** for your application. This plugin is installed and enabled by default when using the OpenShift Container Platform Jenkins image.

For OpenShift Container Platform Pipelines within your project, you will must use the Jenkins Pipeline Build Strategy. This strategy defaults to using a **jenkinsfile** at the root of your source repository, but also provides the following configuration options:

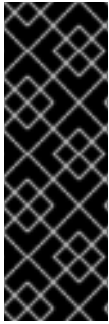
- An inline **jenkinsfile** field within your build configuration.
- A **jenkinsfilePath** field within your build configuration that references the location of the **jenkinsfile** to use relative to the source **contextDir**.



NOTE

The optional **jenkinsfilePath** field specifies the name of the file to use, relative to the source **contextDir**. If **contextDir** is omitted, it defaults to the root of the repository. If **jenkinsfilePath** is omitted, it defaults to **jenkinsfile**.

2.5.4.2. Providing the Jenkins file for pipeline builds



IMPORTANT

The Pipeline build strategy is deprecated in OpenShift Container Platform 4. Equivalent and improved functionality is present in the OpenShift Container Platform Pipelines based on Tekton.

Jenkins images on OpenShift Container Platform are fully supported and users should follow Jenkins user documentation for defining their **jenkinsfile** in a job or store it in a Source Control Management system.

The **jenkinsfile** uses the standard groovy language syntax to allow fine grained control over the configuration, build, and deployment of your application.

You can supply the **jenkinsfile** in one of the following ways:

- A file located within your source code repository.
- Embedded as part of your build configuration using the **jenkinsfile** field.

When using the first option, the **jenkinsfile** must be included in your applications source code repository at one of the following locations:

- A file named **jenkinsfile** at the root of your repository.
- A file named **jenkinsfile** at the root of the source **contextDir** of your repository.
- A file name specified via the **jenkinsfilePath** field of the **JenkinsPipelineStrategy** section of your BuildConfig, which is relative to the source **contextDir** if supplied, otherwise it defaults to the root of the repository.

The **jenkinsfile** is run on the Jenkins agent pod, which must have the OpenShift Container Platform client binaries available if you intend to use the OpenShift Container Platform DSL.

Procedure

To provide the Jenkins file, you can either:

- Embed the Jenkins file in the build configuration.
- Include in the build configuration a reference to the Git repository that contains the Jenkins file.

Embedded Definition

```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "sample-pipeline"
spec:
  strategy:
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        node('agent') {
          stage 'build'
          openshiftBuild(buildConfig: 'ruby-sample-build', showBuildLogs: 'true')
          stage 'deploy'
          openshiftDeploy(deploymentConfig: 'frontend')
        }
```

Reference to Git Repository

```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "sample-pipeline"
spec:
  source:
    git:
      uri: "https://github.com/openshift/ruby-hello-world"
  strategy:
    jenkinsPipelineStrategy:
      jenkinsfilePath: some/repo/dir/filename 1
```

- 1** The optional **jenkinsfilePath** field specifies the name of the file to use, relative to the source **contextDir**. If **contextDir** is omitted, it defaults to the root of the repository. If **jenkinsfilePath** is omitted, it defaults to **jenkinsfile**.

2.5.4.3. Using environment variables for pipeline builds



IMPORTANT

The Pipeline build strategy is deprecated in OpenShift Container Platform 4. Equivalent and improved functionality is present in the OpenShift Container Platform Pipelines based on Tekton.

Jenkins images on OpenShift Container Platform are fully supported and users should follow Jenkins user documentation for defining their **jenkinsfile** in a job or store it in a Source Control Management system.

To make environment variables available to the Pipeline build process, you can add environment variables to the **jenkinsPipelineStrategy** definition of the build configuration.

Once defined, the environment variables will be set as parameters for any Jenkins job associated with the build configuration.

Procedure

- To define environment variables to be used during build, edit the YAML file:

```
jenkinsPipelineStrategy:
...
  env:
    - name: "FOO"
      value: "BAR"
```

You can also manage environment variables defined in the build configuration with the **oc set env** command.

2.5.4.3.1. Mapping between BuildConfig environment variables and Jenkins job parameters

When a Jenkins job is created or updated based on changes to a Pipeline strategy build configuration, any environment variables in the build configuration are mapped to Jenkins job parameters definitions, where the default values for the Jenkins job parameters definitions are the current values of the associated environment variables.

After the Jenkins job's initial creation, you can still add additional parameters to the job from the Jenkins console. The parameter names differ from the names of the environment variables in the build configuration. The parameters are honored when builds are started for those Jenkins jobs.

How you start builds for the Jenkins job dictates how the parameters are set.

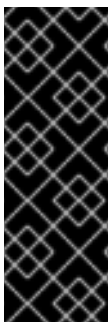
- If you start with **oc start-build**, the values of the environment variables in the build configuration are the parameters set for the corresponding job instance. Any changes you make to the parameters' default values from the Jenkins console are ignored. The build configuration values take precedence.
- If you start with **oc start-build -e**, the values for the environment variables specified in the **-e** option take precedence.
 - If you specify an environment variable not listed in the build configuration, they will be added as a Jenkins job parameter definitions.
 - Any changes you make from the Jenkins console to the parameters corresponding to the environment variables are ignored. The build configuration and what you specify with **oc start-build -e** takes precedence.
- If you start the Jenkins job with the Jenkins console, then you can control the setting of the parameters with the Jenkins console as part of starting a build for the job.



NOTE

It is recommended that you specify in the build configuration all possible environment variables to be associated with job parameters. Doing so reduces disk I/O and improves performance during Jenkins processing.

2.5.4.4. Pipeline build tutorial



IMPORTANT

The Pipeline build strategy is deprecated in OpenShift Container Platform 4. Equivalent and improved functionality is present in the OpenShift Container Platform Pipelines based on Tekton.

Jenkins images on OpenShift Container Platform are fully supported and users should follow Jenkins user documentation for defining their **jenkinsfile** in a job or store it in a Source Control Management system.

This example demonstrates how to create an OpenShift Container Platform Pipeline that will build, deploy, and verify a **Node.js/MongoDB** application using the **nodejs-mongodb.json** template.

Procedure

1. Create the Jenkins master:

```
$ oc project <project_name>
```

Select the project that you want to use or create a new project with **oc new-project <project_name>**.

```
$ oc new-app jenkins-ephemeral ❶
```

If you want to use persistent storage, use **jenkins-persistent** instead.

2. Create a file named **nodejs-sample-pipeline.yaml** with the following content:



NOTE

This creates a **BuildConfig** object that employs the Jenkins pipeline strategy to build, deploy, and scale the **Node.js/MongoDB** example application.

```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "nodejs-sample-pipeline"
spec:
  strategy:
    jenkinsPipelineStrategy:
      jenkinsfile: <pipeline content from below>
      type: JenkinsPipeline
```

3. After you create a **BuildConfig** object with a **jenkinsPipelineStrategy**, tell the pipeline what to do by using an inline **jenkinsfile**:



NOTE

This example does not set up a Git repository for the application.

The following **jenkinsfile** content is written in Groovy using the OpenShift Container Platform DSL. For this example, include inline content in the **BuildConfig** object using the YAML Literal Style, though including a **jenkinsfile** in your source repository is the preferred method.

```
def templatePath = 'https://raw.githubusercontent.com/openshift/nodejs-
ex/master/openshift/templates/nodejs-mongodb.json' ❶
def templateName = 'nodejs-mongodb-example' ❷
pipeline {
  agent {
    node {
      label 'nodejs' ❸
    }
  }
  options {
    timeout(time: 20, unit: 'MINUTES') ❹
  }
  stages {
    stage('preamble') {
      steps {
        script {
          openshift.withCluster() {
```

```

        openshift.withProject() {
            echo "Using project: ${openshift.project()}"
        }
    }
}
}
stage('cleanup') {
    steps {
        script {
            openshift.withCluster() {
                openshift.withProject() {
                    openshift.selector("all", [ template : templateName ]).delete() 5
                    if (openshift.selector("secrets", templateName).exists()) { 6
                        openshift.selector("secrets", templateName).delete()
                    }
                }
            }
        }
    }
}
stage('create') {
    steps {
        script {
            openshift.withCluster() {
                openshift.withProject() {
                    openshift.newApp(templatePath) 7
                }
            }
        }
    }
}
stage('build') {
    steps {
        script {
            openshift.withCluster() {
                openshift.withProject() {
                    def builds = openshift.selector("bc", templateName).related("builds")
                    timeout(5) { 8
                        builds.untilEach(1) {
                            return (it.object().status.phase == "Complete")
                        }
                    }
                }
            }
        }
    }
}
stage('deploy') {
    steps {
        script {
            openshift.withCluster() {
                openshift.withProject() {
                    def rm = openshift.selector("dc", templateName).rollout()
                    timeout(5) { 9

```


- a. If you do not want to create your own file, you can use the sample from the Origin repository by running:

```
$ oc create -f
https://raw.githubusercontent.com/openshift/origin/master/examples/jenkins/pipeline/nodejs-
sample-pipeline.yaml
```

5. Start the Pipeline:

```
$ oc start-build nodejs-sample-pipeline
```



NOTE

Alternatively, you can start your pipeline with the OpenShift Container Platform web console by navigating to the Builds → Pipeline section and clicking **Start Pipeline**, or by visiting the Jenkins Console, navigating to the Pipeline that you created, and clicking **Build Now**.

Once the pipeline is started, you should see the following actions performed within your project:

- A job instance is created on the Jenkins server.
- An agent pod is launched, if your pipeline requires one.
- The pipeline runs on the agent pod, or the master if no agent is required.
 - Any previously created resources with the **template=nodejs-mongodb-example** label will be deleted.
 - A new application, and all of its associated resources, will be created from the **nodejs-mongodb-example** template.
 - A build will be started using the **nodejs-mongodb-example BuildConfig**.
 - The pipeline will wait until the build has completed to trigger the next stage.
 - A deployment will be started using the **nodejs-mongodb-example** deployment configuration.
 - The pipeline will wait until the deployment has completed to trigger the next stage.
 - If the build and deploy are successful, the **nodejs-mongodb-example:latest** image will be tagged as **nodejs-mongodb-example:stage**.
- The agent pod is deleted, if one was required for the pipeline.



NOTE

The best way to visualize the pipeline execution is by viewing it in the OpenShift Container Platform web console. You can view your pipelines by logging in to the web console and navigating to Builds → Pipelines.

2.5.5. Adding secrets with web console

You can add a secret to your build configuration so that it can access a private repository.

Procedure

To add a secret to your build configuration so that it can access a private repository from the OpenShift Container Platform web console:

1. Create a new OpenShift Container Platform project.
2. Create a secret that contains credentials for accessing a private source code repository.
3. Create a build configuration.
4. On the build configuration editor page or in the **create app from builder image** page of the web console, set the **Source Secret**
5. Click **Save**.

2.5.6. Enabling pulling and pushing

You can enable pulling to a private registry by setting the pull secret and pushing by setting the push secret in the build configuration.

Procedure

To enable pulling to a private registry:

- Set the pull secret in the build configuration.

To enable pushing:

- Set the push secret in the build configuration.

2.6. CUSTOM IMAGE BUILDS WITH BUILDDAH

With OpenShift Container Platform 4.12, a docker socket will not be present on the host nodes. This means the *mount docker socket* option of a custom build is not guaranteed to provide an accessible docker socket for use within a custom build image.

If you require this capability in order to build and push images, add the Buildah tool your custom build image and use it to build and push the image within your custom build logic. The following is an example of how to run custom builds with Buildah.



NOTE

Using the custom build strategy requires permissions that normal users do not have by default because it allows the user to execute arbitrary code inside a privileged container running on the cluster. This level of access can be used to compromise the cluster and therefore should be granted only to users who are trusted with administrative privileges on the cluster.

2.6.1. Prerequisites

- Review how to [grant custom build permissions](#).

2.6.2. Creating custom build artifacts

You must create the image you want to use as your custom build image.

Procedure

1. Starting with an empty directory, create a file named **Dockerfile** with the following content:

```
FROM registry.redhat.io/rhel8/buildah
# In this example, `/tmp/build` contains the inputs that build when this
# custom builder image is run. Normally the custom builder image fetches
# this content from some location at build time, by using git clone as an example.
ADD dockerfile.sample /tmp/input/Dockerfile
ADD build.sh /usr/bin
RUN chmod a+x /usr/bin/build.sh
# /usr/bin/build.sh contains the actual custom build logic that will be run when
# this custom builder image is run.
ENTRYPOINT ["/usr/bin/build.sh"]
```

2. In the same directory, create a file named **dockerfile.sample**. This file is included in the custom build image and defines the image that is produced by the custom build:

```
FROM registry.access.redhat.com/ubi8/ubi
RUN touch /tmp/build
```

3. In the same directory, create a file named **build.sh**. This file contains the logic that is run when the custom build runs:

```
#!/bin/sh
# Note that in this case the build inputs are part of the custom builder image, but normally this
# is retrieved from an external source.
cd /tmp/input
# OUTPUT_REGISTRY and OUTPUT_IMAGE are env variables provided by the custom
# build framework
TAG="${OUTPUT_REGISTRY}/${OUTPUT_IMAGE}"

# performs the build of the new image defined by dockerfile.sample
buildah --storage-driver vfs bud --isolation chroot -t ${TAG} .

# buildah requires a slight modification to the push secret provided by the service
# account to use it for pushing the image
cp /var/run/secrets/openshift.io/push/.dockercfg /tmp
(echo '{"auths": " " ; cat /var/run/secrets/openshift.io/push/.dockercfg ; echo "}"') >
/tmp/.dockercfg

# push the new image to the target for the build
buildah --storage-driver vfs push --tls-verify=false --authfile /tmp/.dockercfg ${TAG}
```

2.6.3. Build custom builder image

You can use OpenShift Container Platform to build and push custom builder images to use in a custom strategy.

Prerequisites

- Define all the inputs that will go into creating your new custom builder image.

Procedure

1. Define a **BuildConfig** object that will build your custom builder image:

```
$ oc new-build --binary --strategy=docker --name custom-builder-image
```

2. From the directory in which you created your custom build image, run the build:

```
$ oc start-build custom-builder-image --from-dir . -F
```

After the build completes, your new custom builder image is available in your project in an image stream tag that is named **custom-builder-image:latest**.

2.6.4. Use custom builder image

You can define a **BuildConfig** object that uses the custom strategy in conjunction with your custom builder image to execute your custom build logic.

Prerequisites

- Define all the required inputs for new custom builder image.
- Build your custom builder image.

Procedure

1. Create a file named **buildconfig.yaml**. This file defines the **BuildConfig** object that is created in your project and executed:

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: sample-custom-build
labels:
  name: sample-custom-build
annotations:
  template.alpha.openshift.io/wait-for-ready: 'true'
spec:
  strategy:
    type: Custom
    customStrategy:
      forcePull: true
      from:
        kind: ImageStreamTag
        name: custom-builder-image:latest
        namespace: <yourproject> 1
  output:
    to:
      kind: ImageStreamTag
      name: sample-custom:latest
```

- 1 Specify your project name.

2. Create the **BuildConfig**:

```
$ oc create -f buildconfig.yaml
```

3. Create a file named **imagestream.yaml**. This file defines the image stream to which the build will push the image:

```
kind: ImageStream
apiVersion: image.openshift.io/v1
metadata:
  name: sample-custom
spec: {}
```

4. Create the imagestream:

```
$ oc create -f imagestream.yaml
```

5. Run your custom build:

```
$ oc start-build sample-custom-build -F
```

When the build runs, it launches a pod running the custom builder image that was built earlier. The pod runs the **build.sh** logic that is defined as the entrypoint for the custom builder image. The **build.sh** logic invokes Buildah to build the **dockerfile.sample** that was embedded in the custom builder image, and then uses Buildah to push the new image to the **sample-custom image stream**.

2.7. PERFORMING AND CONFIGURING BASIC BUILDS

The following sections provide instructions for basic build operations, including starting and canceling builds, editing **BuildConfigs**, deleting **BuildConfigs**, viewing build details, and accessing build logs.

2.7.1. Starting a build

You can manually start a new build from an existing build configuration in your current project.

Procedure

To manually start a build, enter the following command:

```
$ oc start-build <buildconfig_name>
```

2.7.1.1. Re-running a build

You can manually re-run a build using the **--from-build** flag.

Procedure

- To manually re-run a build, enter the following command:

```
$ oc start-build --from-build=<build_name>
```

2.7.1.2. Streaming build logs

You can specify the **--follow** flag to stream the build's logs in **stdout**.

Procedure

- To manually stream a build's logs in **stdout**, enter the following command:

```
$ oc start-build <buildconfig_name> --follow
```

2.7.1.3. Setting environment variables when starting a build

You can specify the **--env** flag to set any desired environment variable for the build.

Procedure

- To specify a desired environment variable, enter the following command:

```
$ oc start-build <buildconfig_name> --env=<key>=<value>
```

2.7.1.4. Starting a build with source

Rather than relying on a Git source pull or a Dockerfile for a build, you can also start a build by directly pushing your source, which could be the contents of a Git or SVN working directory, a set of pre-built binary artifacts you want to deploy, or a single file. This can be done by specifying one of the following options for the **start-build** command:

Option	Description
--from-dir=<directory>	Specifies a directory that will be archived and used as a binary input for the build.
--from-file=<file>	Specifies a single file that will be the only file in the build source. The file is placed in the root of an empty directory with the same file name as the original file provided.
--from-repo=<local_source_repo>	Specifies a path to a local repository to use as the binary input for a build. Add the --commit option to control which branch, tag, or commit is used for the build.

When passing any of these options directly to the build, the contents are streamed to the build and override the current build source settings.



NOTE

Builds triggered from binary input will not preserve the source on the server, so rebuilds triggered by base image changes will use the source specified in the build configuration.

Procedure

- Start a build from a source using the following command to send the contents of a local Git repository as an archive from the tag **v2**:

```
$ oc start-build hello-world --from-repo=../hello-world --commit=v2
```

2.7.2. Canceling a build

You can cancel a build using the web console, or with the following CLI command.

Procedure

- To manually cancel a build, enter the following command:

```
$ oc cancel-build <build_name>
```

2.7.2.1. Canceling multiple builds

You can cancel multiple builds with the following CLI command.

Procedure

- To manually cancel multiple builds, enter the following command:

```
$ oc cancel-build <build1_name> <build2_name> <build3_name>
```

2.7.2.2. Canceling all builds

You can cancel all builds from the build configuration with the following CLI command.

Procedure

- To cancel all builds, enter the following command:

```
$ oc cancel-build bc/<buildconfig_name>
```

2.7.2.3. Canceling all builds in a given state

You can cancel all builds in a given state, such as **new** or **pending**, while ignoring the builds in other states.

Procedure

- To cancel all in a given state, enter the following command:

```
$ oc cancel-build bc/<buildconfig_name>
```

2.7.3. Editing a BuildConfig


To edit your build configurations, you use the **Edit BuildConfig** option in the **Builds** view of the **Developer** perspective.

You can use either of the following views to edit a **BuildConfig**:

- The **Form view** enables you to edit your **BuildConfig** using the standard form fields and checkboxes.
- The **YAML view** enables you to edit your **BuildConfig** with full control over the operations.

You can switch between the **Form view** and **YAML view** without losing any data. The data in the **Form view** is transferred to the **YAML view** and vice versa.

Procedure

1. In the **Builds** view of the **Developer** perspective, click the menu  to see the **Edit BuildConfig** option.
2. Click **Edit BuildConfig** to see the **Form view** option.
3. In the **Git** section, enter the Git repository URL for the codebase you want to use to create an application. The URL is then validated.
 - Optional: Click **Show Advanced Git Options** to add details such as:
 - **Git Reference** to specify a branch, tag, or commit that contains code you want to use to build the application.
 - **Context Dir** to specify the subdirectory that contains code you want to use to build the application.
 - **Source Secret** to create a **Secret Name** with credentials for pulling your source code from a private repository.
4. In the **Build from** section, select the option that you would like to build from. You can use the following options:
 - **Image Stream tag** references an image for a given image stream and tag. Enter the project, image stream, and tag of the location you would like to build from and push to.
 - **Image Stream image** references an image for a given image stream and image name. Enter the image stream image you would like to build from. Also enter the project, image stream, and tag to push to.
 - **Docker image**: The Docker image is referenced through a Docker image repository. You will also need to enter the project, image stream, and tag to refer to where you would like to push to.
5. Optional: In the **Environment Variables** section, add the environment variables associated with the project by using the **Name** and **Value** fields. To add more environment variables, use **Add Value**, or **Add from ConfigMap** and **Secret**.
6. Optional: To further customize your application, use the following advanced options:

Trigger

Triggers a new image build when the builder image changes. Add more triggers by clicking **Add Trigger** and selecting the **Type** and **Secret**.

Secrets

Adds secrets for your application. Add more secrets by clicking **Add secret** and selecting the **Secret** and **Mount point**.

Policy

Click **Run policy** to select the build run policy. The selected policy determines the order in which builds created from the build configuration must run.

Hooks

Select **Run build hooks after image is built** to run commands at the end of the build and verify the image. Add **Hook type**, **Command**, and **Arguments** to append to the command.

7. Click **Save** to save the **BuildConfig**.

2.7.4. Deleting a BuildConfig

You can delete a **BuildConfig** using the following command.

Procedure

- To delete a **BuildConfig**, enter the following command:

```
$ oc delete bc <BuildConfigName>
```

This also deletes all builds that were instantiated from this **BuildConfig**.

- To delete a **BuildConfig** and keep the builds instantiated from the **BuildConfig**, specify the **--cascade=false** flag when you enter the following command:

```
$ oc delete --cascade=false bc <BuildConfigName>
```

2.7.5. Viewing build details

You can view build details with the web console or by using the **oc describe** CLI command.

This displays information including:

- The build source.
- The build strategy.
- The output destination.
- Digest of the image in the destination registry.
- How the build was created.

If the build uses the **Docker** or **Source** strategy, the **oc describe** output also includes information about the source revision used for the build, including the commit ID, author, committer, and message.

Procedure

- To view build details, enter the following command:

```
$ oc describe build <build_name>
```

2.7.6. Accessing build logs

You can access build logs using the web console or the CLI.

Procedure

- To stream the logs using the build directly, enter the following command:

```
$ oc describe build <build_name>
```

2.7.6.1. Accessing BuildConfig logs

You can access **BuildConfig** logs using the web console or the CLI.

Procedure

- To stream the logs of the latest build for a **BuildConfig**, enter the following command:

```
$ oc logs -f bc/<buildconfig_name>
```

2.7.6.2. Accessing BuildConfig logs for a given version build

You can access logs for a given version build for a **BuildConfig** using the web console or the CLI.

Procedure

- To stream the logs for a given version build for a **BuildConfig**, enter the following command:

```
$ oc logs --version=<number> bc/<buildconfig_name>
```

2.7.6.3. Enabling log verbosity

You can enable a more verbose output by passing the **BUILD_LOGLEVEL** environment variable as part of the **sourceStrategy** or **dockerStrategy** in a **BuildConfig**.



NOTE

An administrator can set the default build verbosity for the entire OpenShift Container Platform instance by configuring **env/BUILD_LOGLEVEL**. This default can be overridden by specifying **BUILD_LOGLEVEL** in a given **BuildConfig**. You can specify a higher priority override on the command line for non-binary builds by passing **--build-loglevel** to **oc start-build**.

Available log levels for source builds are as follows:

Level 0	Produces output from containers running the assemble script and all encountered errors. This is the default.
Level 1	Produces basic information about the executed process.

Level 2	Produces very detailed information about the executed process.
Level 3	Produces very detailed information about the executed process, and a listing of the archive contents.
Level 4	Currently produces the same information as level 3.
Level 5	Produces everything mentioned on previous levels and additionally provides docker push messages.

Procedure

- To enable more verbose output, pass the **BUILD_LOGLEVEL** environment variable as part of the **sourceStrategy** or **dockerStrategy** in a **BuildConfig**:

```
sourceStrategy:
...
env:
- name: "BUILD_LOGLEVEL"
  value: "2" 1
```

- 1 Adjust this value to the desired log level.

2.8. TRIGGERING AND MODIFYING BUILDS

The following sections outline how to trigger builds and modify builds using build hooks.

2.8.1. Build triggers

When defining a **BuildConfig**, you can define triggers to control the circumstances in which the **BuildConfig** should be run. The following build triggers are available:

- Webhook
- Image change
- Configuration change

2.8.1.1. Webhook triggers

Webhook triggers allow you to trigger a new build by sending a request to the OpenShift Container Platform API endpoint. You can define these triggers using GitHub, GitLab, Bitbucket, or Generic webhooks.

Currently, OpenShift Container Platform webhooks only support the analogous versions of the push event for each of the Git-based Source Code Management (SCM) systems. All other event types are ignored.

When the push events are processed, the OpenShift Container Platform control plane host confirms if the branch reference inside the event matches the branch reference in the corresponding **BuildConfig**. If so, it then checks out the exact commit reference noted in the webhook event on the OpenShift

Container Platform build. If they do not match, no build is triggered.



NOTE

oc new-app and **oc new-build** create GitHub and Generic webhook triggers automatically, but any other needed webhook triggers must be added manually. You can manually add triggers by setting triggers.

For all webhooks, you must define a secret with a key named **WebHookSecretKey** and the value being the value to be supplied when invoking the webhook. The webhook definition must then reference the secret. The secret ensures the uniqueness of the URL, preventing others from triggering the build. The value of the key is compared to the secret provided during the webhook invocation.

For example here is a GitHub webhook with a reference to a secret named **mysecret**:

```
type: "GitHub"
github:
  secretReference:
    name: "mysecret"
```

The secret is then defined as follows. Note that the value of the secret is base64 encoded as is required for any **data** field of a **Secret** object.

```
- kind: Secret
  apiVersion: v1
  metadata:
    name: mysecret
    creationTimestamp:
  data:
    WebHookSecretKey: c2VjcmV0dmFsdWUx
```

2.8.1.1.1. Using GitHub webhooks

GitHub webhooks handle the call made by GitHub when a repository is updated. When defining the trigger, you must specify a secret, which is part of the URL you supply to GitHub when configuring the webhook.

Example GitHub webhook definition:

```
type: "GitHub"
github:
  secretReference:
    name: "mysecret"
```



NOTE

The secret used in the webhook trigger configuration is not the same as **secret** field you encounter when configuring webhook in GitHub UI. The former is to make the webhook URL unique and hard to predict, the latter is an optional string field used to create HMAC hex digest of the body, which is sent as an **X-Hub-Signature** header.

The payload URL is returned as the GitHub Webhook URL by the **oc describe** command (see Displaying Webhook URLs), and is structured as follows:

Example output

```
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/github
```

Prerequisites

- Create a **BuildConfig** from a GitHub repository.

Procedure

1. To configure a GitHub Webhook:
 - a. After creating a **BuildConfig** from a GitHub repository, run:

```
$ oc describe bc/<name-of-your-BuildConfig>
```

This generates a webhook GitHub URL that looks like:

Example output

```
<https://api.starter-us-east-1.openshift.com:443/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/github
```

- b. Cut and paste this URL into GitHub, from the GitHub web console.
- c. In your GitHub repository, select **Add Webhook** from **Settings → Webhooks**.
- d. Paste the URL output into the **Payload URL** field.
- e. Change the **Content Type** from GitHub's default **application/x-www-form-urlencoded** to **application/json**.
- f. Click **Add webhook**.
You should see a message from GitHub stating that your webhook was successfully configured.

Now, when you push a change to your GitHub repository, a new build automatically starts, and upon a successful build a new deployment starts.



NOTE

[Gogs](#) supports the same webhook payload format as GitHub. Therefore, if you are using a Gogs server, you can define a GitHub webhook trigger on your **BuildConfig** and trigger it by your Gogs server as well.

2. Given a file containing a valid JSON payload, such as **payload.json**, you can manually trigger the webhook with **curl**:

```
$ curl -H "X-GitHub-Event: push" -H "Content-Type: application/json" -k -X POST --data-binary @payload.json https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/github
```

-

The **-k** argument is only necessary if your API server does not have a properly signed certificate.

Additional resources

- [Gogs](#)

2.8.1.1.2. Using GitLab webhooks

GitLab webhooks handle the call made by GitLab when a repository is updated. As with the GitHub triggers, you must specify a secret. The following example is a trigger definition YAML within the **BuildConfig**:

```
type: "GitLab"
gitlab:
  secretReference:
    name: "mysecret"
```

The payload URL is returned as the GitLab Webhook URL by the **oc describe** command, and is structured as follows:

Example output

```
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/gitlab
```

Procedure

1. To configure a GitLab Webhook:
 - a. Describe the **BuildConfig** to get the webhook URL:


```
$ oc describe bc <name>
```
 - b. Copy the webhook URL, replacing **<secret>** with your secret value.
 - c. Follow the [GitLab setup instructions](#) to paste the webhook URL into your GitLab repository settings.
2. Given a file containing a valid JSON payload, such as **payload.json**, you can manually trigger the webhook with **curl**:

```
$ curl -H "X-GitLab-Event: Push Hook" -H "Content-Type: application/json" -k -X POST --data-binary @payload.json https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/gitlab
```

The **-k** argument is only necessary if your API server does not have a properly signed certificate.

2.8.1.1.3. Using Bitbucket webhooks

[Bitbucket webhooks](#) handle the call made by Bitbucket when a repository is updated. Similar to the previous triggers, you must specify a secret. The following example is a trigger definition YAML within the **BuildConfig**:

```
type: "Bitbucket"
bitbucket:
  secretReference:
    name: "mysecret"
```

The payload URL is returned as the Bitbucket Webhook URL by the **oc describe** command, and is structured as follows:

Example output

```
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/bitbucket
```

Procedure

1. To configure a Bitbucket Webhook:
 - a. Describe the 'BuildConfig' to get the webhook URL:
 - b. Copy the webhook URL, replacing **<secret>** with your secret value.
 - c. Follow the [Bitbucket setup instructions](#) to paste the webhook URL into your Bitbucket repository settings.
2. Given a file containing a valid JSON payload, such as **payload.json**, you can manually trigger the webhook with **curl**:

```
$ curl -H "X-Event-Key: repo:push" -H "Content-Type: application/json" -k -X POST --data-binary @payload.json
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/bitbucket
```

The **-k** argument is only necessary if your API server does not have a properly signed certificate.

2.8.1.1.4. Using generic webhooks

Generic webhooks are invoked from any system capable of making a web request. As with the other webhooks, you must specify a secret, which is part of the URL that the caller must use to trigger the build. The secret ensures the uniqueness of the URL, preventing others from triggering the build. The following is an example trigger definition YAML within the **BuildConfig**:

```
type: "Generic"
generic:
  secretReference:
    name: "mysecret"
  allowEnv: true 1
```

- 1** Set to **true** to allow a generic webhook to pass in environment variables.

Procedure

1. To set up the caller, supply the calling system with the URL of the generic webhook endpoint for your build:

Example output

```
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildcon
gs/<name>/webhooks/<secret>/generic
```

The caller must invoke the webhook as a **POST** operation.

2. To invoke the webhook manually you can use **curl**:

```
$ curl -X POST -k
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildcon
gs/<name>/webhooks/<secret>/generic
```

The HTTP verb must be set to **POST**. The insecure **-k** flag is specified to ignore certificate validation. This second flag is not necessary if your cluster has properly signed certificates.

The endpoint can accept an optional payload with the following format:

```
git:
  uri: "<url to git repository>"
  ref: "<optional git reference>"
  commit: "<commit hash identifying a specific git commit>"
  author:
    name: "<author name>"
    email: "<author e-mail>"
  committer:
    name: "<committer name>"
    email: "<committer e-mail>"
  message: "<commit message>"
env: ❶
  - name: "<variable name>"
    value: "<variable value>"
```

- ❶ Similar to the **BuildConfig** environment variables, the environment variables defined here are made available to your build. If these variables collide with the **BuildConfig** environment variables, these variables take precedence. By default, environment variables passed by webhook are ignored. Set the **allowEnv** field to **true** on the webhook definition to enable this behavior.

3. To pass this payload using **curl**, define it in a file named **payload_file.yaml** and run:

```
$ curl -H "Content-Type: application/yaml" --data-binary @payload_file.yaml -X POST -k
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildcon
gs/<name>/webhooks/<secret>/generic
```

The arguments are the same as the previous example with the addition of a header and a payload. The **-H** argument sets the **Content-Type** header to **application/yaml** or **application/json** depending on your payload format. The **--data-binary** argument is used to send a binary payload with newlines intact with the **POST** request.



NOTE

OpenShift Container Platform permits builds to be triggered by the generic webhook even if an invalid request payload is presented, for example, invalid content type, unparsable or invalid content, and so on. This behavior is maintained for backwards compatibility. If an invalid request payload is presented, OpenShift Container Platform returns a warning in JSON format as part of its **HTTP 200 OK** response.

2.8.1.1.5. Displaying webhook URLs

You can use the following command to display webhook URLs associated with a build configuration. If the command does not display any webhook URLs, then no webhook trigger is defined for that build configuration.

Procedure

- To display any webhook URLs associated with a **BuildConfig**, run:

```
$ oc describe bc <name>
```

2.8.1.2. Using image change triggers

As a developer, you can configure your build to run automatically every time a base image changes.

You can use image change triggers to automatically invoke your build when a new version of an upstream image is available. For example, if a build is based on a RHEL image, you can trigger that build to run any time the RHEL image changes. As a result, the application image is always running on the latest RHEL base image.



NOTE

Image streams that point to container images in [v1 container registries](#) only trigger a build once when the image stream tag becomes available and not on subsequent image updates. This is due to the lack of uniquely identifiable images in v1 container registries.

Procedure

1. Define an **ImageStream** that points to the upstream image you want to use as a trigger:

```
kind: "ImageStream"
apiVersion: "v1"
metadata:
  name: "ruby-20-centos7"
```

This defines the image stream that is tied to a container image repository located at **<system-registry>/<namespace>/ruby-20-centos7**. The **<system-registry>** is defined as a service with the name **docker-registry** running in OpenShift Container Platform.

2. If an image stream is the base image for the build, set the **from** field in the build strategy to point to the **ImageStream**:

```
strategy:
  sourceStrategy:
    from:
```

```
kind: "ImageStreamTag"
name: "ruby-20-centos7:latest"
```

In this case, the **sourceStrategy** definition is consuming the **latest** tag of the image stream named **ruby-20-centos7** located within this namespace.

3. Define a build with one or more triggers that point to **ImageStreams**:

```
type: "ImageChange" ❶
imageChange: {}
type: "ImageChange" ❷
imageChange:
  from:
    kind: "ImageStreamTag"
    name: "custom-image:latest"
```

- ❶ An image change trigger that monitors the **ImageStream** and **Tag** as defined by the build strategy's **from** field. The **imageChange** object here must be empty.
- ❷ An image change trigger that monitors an arbitrary image stream. The **imageChange** part, in this case, must include a **from** field that references the **ImageStreamTag** to monitor.

When using an image change trigger for the strategy image stream, the generated build is supplied with an immutable docker tag that points to the latest image corresponding to that tag. This new image reference is used by the strategy when it executes for the build.

For other image change triggers that do not reference the strategy image stream, a new build is started, but the build strategy is not updated with a unique image reference.

Since this example has an image change trigger for the strategy, the resulting build is:

```
strategy:
  sourceStrategy:
    from:
      kind: "DockerImage"
      name: "172.30.17.3:5001/mynamespace/ruby-20-centos7:<immutableid>"
```

This ensures that the triggered build uses the new image that was just pushed to the repository, and the build can be re-run any time with the same inputs.

You can pause an image change trigger to allow multiple changes on the referenced image stream before a build is started. You can also set the **paused** attribute to true when initially adding an **ImageChangeTrigger** to a **BuildConfig** to prevent a build from being immediately triggered.

```
type: "ImageChange"
imageChange:
  from:
    kind: "ImageStreamTag"
    name: "custom-image:latest"
  paused: true
```

In addition to setting the image field for all **Strategy** types, for custom builds, the **OPENSIFT_CUSTOM_BUILD_BASE_IMAGE** environment variable is checked. If it does not exist, then it is created with the immutable image reference. If it does exist, then it is updated with the

immutable image reference.

If a build is triggered due to a webhook trigger or manual request, the build that is created uses the **<immutableid>** resolved from the **ImageStream** referenced by the **Strategy**. This ensures that builds are performed using consistent image tags for ease of reproduction.

Additional resources

- [v1 container registries](#)

2.8.1.3. Identifying the image change trigger of a build

As a developer, if you have image change triggers, you can identify which image change initiated the last build. This can be useful for debugging or troubleshooting builds.

Example BuildConfig

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: bc-ict-example
  namespace: bc-ict-example-namespace
spec:
  # ...

  triggers:
  - imageChange:
      from:
        kind: ImageStreamTag
        name: input:latest
        namespace: bc-ict-example-namespace
  - imageChange:
      from:
        kind: ImageStreamTag
        name: input2:latest
        namespace: bc-ict-example-namespace
      type: ImageChange
  status:
    imageChangeTriggers:
      - from:
          name: input:latest
          namespace: bc-ict-example-namespace
        lastTriggerTime: "2021-06-30T13:47:53Z"
        lastTriggeredImageID: image-registry.openshift-image-registry.svc:5000/bc-ict-example-namespace/input@sha256:0f88ffbeb9d25525720bfa3524cb1bf0908b7f791057cf1acfae917b11266a69
      - from:
          name: input2:latest
          namespace: bc-ict-example-namespace
        lastTriggeredImageID: image-registry.openshift-image-registry.svc:5000/bc-ict-example-namespace/input2@sha256:0f88ffbeb9d25525720bfa3524cb2ce0908b7f791057cf1acfae917b11266a69

    lastVersion: 1
```

**NOTE**

This example omits elements that are not related to image change triggers.

Prerequisites

- You have configured multiple image change triggers. These triggers have triggered one or more builds.

Procedure

1. In **buildConfig.status.imageChangeTriggers** to identify the **lastTriggerTime** that has the latest timestamp.

This **ImageChangeTriggerStatus**

Then you use the ``name`` and ``namespace`` from that build to find the corresponding image change trigger in ``buildConfig.spec.triggers``.

2. Under **imageChangeTriggers**, compare timestamps to identify the latest

Image change triggers

In your build configuration, **buildConfig.spec.triggers** is an array of build trigger policies, **BuildTriggerPolicy**.

Each **BuildTriggerPolicy** has a **type** field and set of pointers fields. Each pointer field corresponds to one of the allowed values for the **type** field. As such, you can only set **BuildTriggerPolicy** to only one pointer field.

For image change triggers, the value of **type** is **ImageChange**. Then, the **imageChange** field is the pointer to an **ImageChangeTrigger** object, which has the following fields:

- **lastTriggeredImageID**: This field, which is not shown in the example, is deprecated in OpenShift Container Platform 4.8 and will be ignored in a future release. It contains the resolved image reference for the **ImageStreamTag** when the last build was triggered from this **BuildConfig**.
- **paused**: You can use this field, which is not shown in the example, to temporarily disable this particular image change trigger.
- **from**: You use this field to reference the **ImageStreamTag** that drives this image change trigger. Its type is the core Kubernetes type, **OwnerReference**.

The **from** field has the following fields of note: **kind**: For image change triggers, the only supported value is **ImageStreamTag**. **namespace**: You use this field to specify the namespace of the **ImageStreamTag**. **name**: You use this field to specify the **ImageStreamTag**.

Image change trigger status

In your build configuration, **buildConfig.status.imageChangeTriggers** is an array of **ImageChangeTriggerStatus** elements. Each **ImageChangeTriggerStatus** element includes the **from**, **lastTriggeredImageID**, and **lastTriggerTime** elements shown in the preceding example.

The **ImageChangeTriggerStatus** that has the most recent **lastTriggerTime** triggered the most recent build. You use its **name** and **namespace** to identify the image change trigger in **buildConfig.spec.triggers** that triggered the build.

The **lastTriggerTime** with the most recent timestamp signifies the **ImageChangeTriggerStatus** of the last build. This **ImageChangeTriggerStatus** has the same **name** and **namespace** as the image change trigger in **buildConfig.spec.triggers** that triggered the build.

Additional resources

- [v1 container registries](#)

2.8.1.4. Configuration change triggers

A configuration change trigger allows a build to be automatically invoked as soon as a new **BuildConfig** is created.

The following is an example trigger definition YAML within the **BuildConfig**:

```
type: "ConfigChange"
```



NOTE

Configuration change triggers currently only work when creating a new **BuildConfig**. In a future release, configuration change triggers will also be able to launch a build whenever a **BuildConfig** is updated.

2.8.1.4.1. Setting triggers manually

Triggers can be added to and removed from build configurations with **oc set triggers**.

Procedure

- To set a GitHub webhook trigger on a build configuration, use:

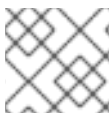
```
$ oc set triggers bc <name> --from-github
```

- To set an imagechange trigger, use:

```
$ oc set triggers bc <name> --from-image='<image>'
```

- To remove a trigger, add **--remove**:

```
$ oc set triggers bc <name> --from-bitbucket --remove
```



NOTE

When a webhook trigger already exists, adding it again regenerates the webhook secret.

For more information, consult the help documentation with by running:

```
$ oc set triggers --help
```

2.8.2. Build hooks

Build hooks allow behavior to be injected into the build process.

The **postCommit** field of a **BuildConfig** object runs commands inside a temporary container that is running the build output image. The hook is run immediately after the last layer of the image has been committed and before the image is pushed to a registry.

The current working directory is set to the image's **WORKDIR**, which is the default working directory of the container image. For most images, this is where the source code is located.

The hook fails if the script or command returns a non-zero exit code or if starting the temporary container fails. When the hook fails it marks the build as failed and the image is not pushed to a registry. The reason for failing can be inspected by looking at the build logs.

Build hooks can be used to run unit tests to verify the image before the build is marked complete and the image is made available in a registry. If all tests pass and the test runner returns with exit code **0**, the build is marked successful. In case of any test failure, the build is marked as failed. In all cases, the build log contains the output of the test runner, which can be used to identify failed tests.

The **postCommit** hook is not only limited to running tests, but can be used for other commands as well. Since it runs in a temporary container, changes made by the hook do not persist, meaning that running the hook cannot affect the final image. This behavior allows for, among other uses, the installation and usage of test dependencies that are automatically discarded and are not present in the final image.

2.8.2.1. Configuring post commit build hooks

There are different ways to configure the post build hook. All forms in the following examples are equivalent and run **bundle exec rake test --verbose**.

Procedure

- Shell script:

```
postCommit:
  script: "bundle exec rake test --verbose"
```

The **script** value is a shell script to be run with **/bin/sh -ic**. Use this when a shell script is appropriate to execute the build hook. For example, for running unit tests as above. To control the image entry point, or if the image does not have **/bin/sh**, use **command** and/or **args**.



NOTE

The additional **-i** flag was introduced to improve the experience working with CentOS and RHEL images, and may be removed in a future release.

- Command as the image entry point:

```
postCommit:
  command: ["/bin/bash", "-c", "bundle exec rake test --verbose"]
```

In this form, **command** is the command to run, which overrides the image entry point in the exec form, as documented in the [Dockerfile reference](#). This is needed if the image does not have **/bin/sh**, or if you do not want to use a shell. In all other cases, using **script** might be more convenient.

- Command with arguments:

```
postCommit:
  command: ["bundle", "exec", "rake", "test"]
  args: ["--verbose"]
```

This form is equivalent to appending the arguments to **command**.



NOTE

Providing both **script** and **command** simultaneously creates an invalid build hook.

2.8.2.2. Using the CLI to set post commit build hooks

The **oc set build-hook** command can be used to set the build hook for a build configuration.

Procedure

1. To set a command as the post-commit build hook:

```
$ oc set build-hook bc/mybc \
  --post-commit \
  --command \
  -- bundle exec rake test --verbose
```

2. To set a script as the post-commit build hook:

```
$ oc set build-hook bc/mybc --post-commit --script="bundle exec rake test --verbose"
```

2.9. PERFORMING ADVANCED BUILDS

The following sections provide instructions for advanced build operations including setting build resources and maximum duration, assigning builds to nodes, chaining builds, build pruning, and build run policies.

2.9.1. Setting build resources

By default, builds are completed by pods using unbound resources, such as memory and CPU. These resources can be limited.

Procedure

You can limit resource use in two ways:

- Limit resource use by specifying resource limits in the default container limits of a project.
- Limit resource use by specifying resource limits as part of the build configuration. ** In the following example, each of the **resources**, **cpu**, and **memory** parameters are optional:

```
apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  resources:
```

```
limits:
```

```
  cpu: "100m" 1
```

```
  memory: "256Mi" 2
```

- ¹ **cpu** is in CPU units: **100m** represents 0.1 CPU units ($100 * 1e-3$).
- ² **memory** is in bytes: **256Mi** represents 268435456 bytes ($256 * 2^{20}$).

However, if a quota has been defined for your project, one of the following two items is required:

- A **resources** section set with an explicit **requests**:

```
resources:
```

```
  requests: 1
```

```
    cpu: "100m"
```

```
    memory: "256Mi"
```

- ¹ The **requests** object contains the list of resources that correspond to the list of resources in the quota.

- A limit range defined in your project, where the defaults from the **LimitRange** object apply to pods created during the build process.
Otherwise, build pod creation will fail, citing a failure to satisfy quota.

2.9.2. Setting maximum duration

When defining a **BuildConfig** object, you can define its maximum duration by setting the **completionDeadlineSeconds** field. It is specified in seconds and is not set by default. When not set, there is no maximum duration enforced.

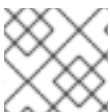
The maximum duration is counted from the time when a build pod gets scheduled in the system, and defines how long it can be active, including the time needed to pull the builder image. After reaching the specified timeout, the build is terminated by OpenShift Container Platform.

Procedure

- To set maximum duration, specify **completionDeadlineSeconds** in your **BuildConfig**. The following example shows the part of a **BuildConfig** specifying **completionDeadlineSeconds** field for 30 minutes:

```
spec:
```

```
  completionDeadlineSeconds: 1800
```



NOTE

This setting is not supported with the Pipeline Strategy option.

2.9.3. Assigning builds to specific nodes

Builds can be targeted to run on specific nodes by specifying labels in the **nodeSelector** field of a build configuration. The **nodeSelector** value is a set of key-value pairs that are matched to **Node** labels when scheduling the build pod.

The **nodeSelector** value can also be controlled by cluster-wide default and override values. Defaults will only be applied if the build configuration does not define any key-value pairs for the **nodeSelector** and also does not define an explicitly empty map value of **nodeSelector: {}**. Override values will replace values in the build configuration on a key by key basis.



NOTE

If the specified **NodeSelector** cannot be matched to a node with those labels, the build still stay in the **Pending** state indefinitely.

Procedure

- Assign builds to run on specific nodes by assigning labels in the **nodeSelector** field of the **BuildConfig**, for example:

```
apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  nodeSelector: 1
    key1: value1
    key2: value2
```

- 1 Builds associated with this build configuration will run only on nodes with the **key1=value2** and **key2=value2** labels.

2.9.4. Chained builds

For compiled languages such as Go, C, C++, and Java, including the dependencies necessary for compilation in the application image might increase the size of the image or introduce vulnerabilities that can be exploited.

To avoid these problems, two builds can be chained together. One build that produces the compiled artifact, and a second build that places that artifact in a separate image that runs the artifact.

In the following example, a source-to-image (S2I) build is combined with a docker build to compile an artifact that is then placed in a separate runtime image.



NOTE

Although this example chains a S2I build and a docker build, the first build can use any strategy that produces an image containing the desired artifacts, and the second build can use any strategy that can consume input content from an image.

The first build takes the application source and produces an image containing a **WAR** file. The image is pushed to the **artifact-image** image stream. The path of the output artifact depends on the **assemble** script of the S2I builder used. In this case, it is output to **/wildfly/standalone/deployments/ROOT.war**.

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: artifact-build
```

```
spec:
  output:
    to:
      kind: ImageStreamTag
      name: artifact-image:latest
  source:
    git:
      uri: https://github.com/openshift/openshift-jee-sample.git
      ref: "master"
  strategy:
    sourceStrategy:
      from:
        kind: ImageStreamTag
        name: wildfly:10.1
        namespace: openshift
```

The second build uses image source with a path to the WAR file inside the output image from the first build. An inline **dockerfile** copies that **WAR** file into a runtime image.

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: image-build
spec:
  output:
    to:
      kind: ImageStreamTag
      name: image-build:latest
  source:
    dockerfile: |-
      FROM jee-runtime:latest
      COPY ROOT.war /deployments/ROOT.war
    images:
      - from: ❶
        kind: ImageStreamTag
        name: artifact-image:latest
      paths: ❷
        - sourcePath: /wildfly/standalone/deployments/ROOT.war
          destinationDir: "."
  strategy:
    dockerStrategy:
      from: ❸
        kind: ImageStreamTag
        name: jee-runtime:latest
  triggers:
    - imageChange: {}
      type: ImageChange
```

❶ **from** specifies that the docker build should include the output of the image from the **artifact-image** image stream, which was the target of the previous build.

❷ **paths** specifies which paths from the target image to include in the current docker build.

❸ The runtime image is used as the source image for the docker build.

The result of this setup is that the output image of the second build does not have to contain any of the build tools that are needed to create the **WAR** file. Also, because the second build contains an image change trigger, whenever the first build is run and produces a new image with the binary artifact, the second build is automatically triggered to produce a runtime image that contains that artifact. Therefore, both builds behave as a single build with two stages.

2.9.5. Pruning builds

By default, builds that have completed their lifecycle are persisted indefinitely. You can limit the number of previous builds that are retained.

Procedure

1. Limit the number of previous builds that are retained by supplying a positive integer value for **successfulBuildsHistoryLimit** or **failedBuildsHistoryLimit** in your **BuildConfig**, for example:

```
apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  successfulBuildsHistoryLimit: 2 1
  failedBuildsHistoryLimit: 2 2
```

- ¹ **successfulBuildsHistoryLimit** will retain up to two builds with a status of **completed**.
- ² **failedBuildsHistoryLimit** will retain up to two builds with a status of **failed**, **canceled**, or **error**.

2. Trigger build pruning by one of the following actions:

- Updating a build configuration.
- Waiting for a build to complete its lifecycle.

Builds are sorted by their creation timestamp with the oldest builds being pruned first.



NOTE

Administrators can manually prune builds using the 'oc adm' object pruning command.

2.9.6. Build run policy

The build run policy describes the order in which the builds created from the build configuration should run. This can be done by changing the value of the **runPolicy** field in the **spec** section of the **Build** specification.

It is also possible to change the **runPolicy** value for existing build configurations, by:

- Changing **Parallel** to **Serial** or **SerialLatestOnly** and triggering a new build from this configuration causes the new build to wait until all parallel builds complete as the serial build can only run alone.

- Changing **Serial** to **SerialLatestOnly** and triggering a new build causes cancellation of all existing builds in queue, except the currently running build and the most recently created build. The newest build runs next.

2.10. USING RED HAT SUBSCRIPTIONS IN BUILDS

Use the following sections to run entitled builds on OpenShift Container Platform.

2.10.1. Creating an image stream tag for the Red Hat Universal Base Image

To use Red Hat subscriptions within a build, you create an image stream tag to reference the Universal Base Image (UBI).

To make the UBI available **in every project** in the cluster, you add the image stream tag to the **openshift** namespace. Otherwise, to make it available **in a specific project**, you add the image stream tag to that project.

The benefit of using image stream tags this way is that doing so grants access to the UBI based on the **registry.redhat.io** credentials in the install pull secret without exposing the pull secret to other users. This is more convenient than requiring each developer to install pull secrets with **registry.redhat.io** credentials in each project.

Procedure

- To create an **ImageStreamTag** in the **openshift** namespace, so it is available to developers in all projects, enter:

```
$ oc tag --source=docker registry.redhat.io/ubi8/ubi:latest ubi:latest -n openshift
```

TIP

You can alternatively apply the following YAML to create an **ImageStreamTag** in the **openshift** namespace:

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  name: ubi
  namespace: openshift
spec:
  tags:
  - from:
    kind: DockerImage
    name: registry.redhat.io/ubi8/ubi:latest
    name: latest
    referencePolicy:
      type: Source
```

- To create an **ImageStreamTag** in a single project, enter:

```
$ oc tag --source=docker registry.redhat.io/ubi8/ubi:latest ubi:latest
```

TIP

You can alternatively apply the following YAML to create an **ImageStreamTag** in a single project:

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  name: ubi
spec:
  tags:
    - from:
        kind: DockerImage
        name: registry.redhat.io/ubi8/ubi:latest
      name: latest
    referencePolicy:
      type: Source
```

2.10.2. Adding subscription entitlements as a build secret

Builds that use Red Hat subscriptions to install content must include the entitlement keys as a build secret.

Prerequisites

You must have access to Red Hat entitlements through your subscription. The entitlement secret is automatically created by the Insights Operator.

TIP

When you perform an Entitlement Build using Red Hat Enterprise Linux (RHEL) 7, you must have the following instructions in your Dockerfile before you run any **yum** commands:

```
RUN rm /etc/rhsm-host
```

Procedure

1. Add the etc-pki-entitlement secret as a build volume in the build configuration's Docker strategy:

```
strategy:
  dockerStrategy:
    from:
      kind: ImageStreamTag
      name: ubi:latest
    volumes:
      - name: etc-pki-entitlement
        mounts:
          - destinationPath: /etc/pki/entitlement
        source:
          type: Secret
          secret:
            secretName: etc-pki-entitlement
```

2.10.3. Running builds with Subscription Manager

2.10.3.1. Docker builds using Subscription Manager

Docker strategy builds can use the Subscription Manager to install subscription content.

Prerequisites

The entitlement keys must be added as build strategy volumes.

Procedure

Use the following as an example Dockerfile to install content with the Subscription Manager:

```
FROM registry.redhat.io/ubi8/ubi:latest
RUN dnf search kernel-devel --showduplicates && \
    dnf install -y kernel-devel
```

2.10.4. Running builds with Red Hat Satellite subscriptions

2.10.4.1. Adding Red Hat Satellite configurations to builds

Builds that use Red Hat Satellite to install content must provide appropriate configurations to obtain content from Satellite repositories.

Prerequisites

- You must provide or create a **yum**-compatible repository configuration file that downloads content from your Satellite instance.

Sample repository configuration

```
[test-<name>]
name=test-<number>
baseurl = https://satellite.../content/dist/rhel/server/7/7Server/x86_64/os
enabled=1
gpgcheck=0
sslverify=0
sslclientkey = /etc/pki/entitlement/...-key.pem
sslclientcert = /etc/pki/entitlement/....pem
```

Procedure

- Create a **ConfigMap** containing the Satellite repository configuration file:

```
$ oc create configmap yum-repos-d --from-file /path/to/satellite.repo
```

- Add the Satellite repository configuration and entitlement key as a build volumes:

```
strategy:
  dockerStrategy:
    from:
      kind: ImageStreamTag
```

```

name: ubi:latest
volumes:
- name: yum-repos-d
  mounts:
  - destinationPath: /etc/yum.repos.d
  source:
    type: ConfigMap
    configMap:
      name: yum-repos-d
- name: etc-pki-entitlement
  mounts:
  - destinationPath: /etc/pki/entitlement
  source:
    type: Secret
    secret:
      secretName: etc-pki-entitlement

```

2.10.4.2. Docker builds using Red Hat Satellite subscriptions

Docker strategy builds can use Red Hat Satellite repositories to install subscription content.

Prerequisites

- You have added the entitlement keys and Satellite repository configurations as build volumes.

Procedure

Use the following as an example Dockerfile to install content with Satellite:

```

FROM registry.redhat.io/ubi8/ubi:latest
RUN dnf search kernel-devel --showduplicates && \
    dnf install -y kernel-devel

```

Additional resources

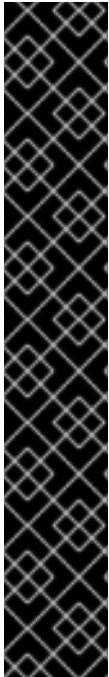
- [How to use builds with Red Hat Satellite subscriptions and which certificate to use](#)

2.10.5. Running entitled builds using SharedSecret objects

You can configure and perform a build in one namespace that securely uses RHEL entitlements from a **Secret** object in another namespace.

You can still access RHEL entitlements from OpenShift Builds by creating a **Secret** object with your subscription credentials in the same namespace as your **Build** object. However, now, in OpenShift Container Platform 4.10 and later, you can access your credentials and certificates from a **Secret** object in one of the OpenShift Container Platform system namespaces. You run entitled builds with a CSI volume mount of a **SharedSecret** custom resource (CR) instance that references the **Secret** object.

This procedure relies on the newly introduced Shared Resources CSI Driver feature, which you can use to declare CSI Volume mounts in OpenShift Container Platform Builds. It also relies on the OpenShift Container Platform Insights Operator.



IMPORTANT

The Shared Resources CSI Driver and The Build CSI Volumes are both Technology Preview features, which are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

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

The Shared Resources CSI Driver and the Build CSI Volumes features also belong to the **TechPreviewNoUpgrade** feature set, which is a subset of the current Technology Preview features. You can enable the **TechPreviewNoUpgrade** feature set on test clusters, where you can fully test them while leaving the features disabled on production clusters. Enabling this feature set cannot be undone and prevents minor version updates. This feature set is not recommended on production clusters. See "Enabling Technology Preview features using feature gates" in the following "Additional resources" section.

Prerequisites

- You have enabled the **TechPreviewNoUpgrade** feature set by using the feature gates.
- You have a **SharedSecret** custom resource (CR) instance that references the **Secret** object where the Insights Operator stores the subscription credentials.
- You must have permission to perform the following actions:
 - Create build configs and start builds.
 - Discover which **SharedSecret** CR instances are available by entering the **oc get sharedsecrets** command and getting a non-empty list back.
 - Determine if the **builder** service account available to you in your namespace is allowed to use the given **SharedSecret** CR instance. In other words, you can run **oc adm policy who-can use <identifier of specific SharedSecret>** to see if the **builder** service account in your namespace is listed.



NOTE

If neither of the last two prerequisites in this list are met, establish, or ask someone to establish, the necessary role-based access control (RBAC) so that you can discover **SharedSecret** CR instances and enable service accounts to use **SharedSecret** CR instances.

Procedure

1. Grant the **builder** service account RBAC permissions to use the **SharedSecret** CR instance by using **oc apply** with YAML content:



NOTE

Currently, **kubectl** and **oc** have hard-coded special case logic restricting the **use** verb to roles centered around pod security. Therefore, you cannot use **oc create role ...** to create the role needed for consuming **SharedSecret** CR instances.

Example `oc apply -f` command with YAML Role object definition

```
$ oc apply -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: shared-resource-my-share
  namespace: my-namespace
rules:
  - apiGroups:
    - sharedresource.openshift.io
    resources:
    - sharedsecrets
    resourceNames:
    - my-share
    verbs:
    - use
EOF
```

2. Create the **RoleBinding** associated with the role by using the **oc** command:

Example `oc create rolebinding` command

```
$ oc create rolebinding shared-resource-my-share --role=shared-resource-my-share --
serviceaccount=my-namespace:builder
```

3. Create a **BuildConfig** object that accesses the RHEL entitlements.

Example YAML BuildConfig object definition

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: my-csi-bc
  namespace: my-csi-app-namespace
spec:
  runPolicy: Serial
  source:
    dockerfile: |
      FROM registry.redhat.io/ubi8/ubi:latest
      RUN ls -la /etc/pki/entitlement
      RUN rm /etc/rhsm-host
      RUN yum repolist --disablerepo=*
      RUN subscription-manager repos --enable rhocp-4.9-for-rhel-8-x86_64-rpms
      RUN yum -y update
      RUN yum install -y openshift-clients.x86_64
  strategy:
    type: Docker
    dockerStrategy:
      volumes:
        - mounts:
          - destinationPath: "/etc/pki/entitlement"
            name: my-csi-shared-secret
            source:
```

```
csi:
  driver: csi.sharedresource.openshift.io
  readOnly: true
  volumeAttributes:
    sharedSecret: my-share-bc
  type: CSI
```

4. Start a build from the **BuildConfig** object and follow the logs with the **oc** command.

Example oc start-build command

```
$ oc start-build my-csi-bc -F
```

Example 2.1. Example output from the oc start-build command



NOTE

Some sections of the following output have been replaced with ...

```
build.build.openshift.io/my-csi-bc-1 started
Caching blobs under "/var/cache/blobs".

Pulling image registry.redhat.io/ubi8/ubi:latest ...
Trying to pull registry.redhat.io/ubi8/ubi:latest...
Getting image source signatures
Copying blob
sha256:5dcdbdc60ea6b60326f98e2b49d6ebcb7771df4b70c6297ddf2d7dede6692df6e
Copying blob
sha256:8671113e1c57d3106acaef2383f9bbfe1c45a26eacb03ec82786a494e15956c3
Copying config
sha256:b81e86a2cb9a001916dc4697d7ed4777a60f757f0b8dcc2c4d8df42f2f7edb3a
Writing manifest to image destination
Storing signatures
Adding transient rw bind mount for /run/secrets/rhsm
STEP 1/9: FROM registry.redhat.io/ubi8/ubi:latest
STEP 2/9: RUN ls -la /etc/pki/entitlement
total 360
drwxrwxrwt. 2 root root 80 Feb 3 20:28 .
drwxr-xr-x. 10 root root 154 Jan 27 15:53 ..
-rw-r--r--. 1 root root 3243 Feb 3 20:28 entitlement-key.pem
-rw-r--r--. 1 root root 362540 Feb 3 20:28 entitlement.pem
time="2022-02-03T20:28:32Z" level=warning msg="Adding metacopy option, configured globally"
--> 1ef7c6d8c1a
STEP 3/9: RUN rm /etc/rhsm-host
time="2022-02-03T20:28:33Z" level=warning msg="Adding metacopy option, configured globally"
--> b1c61f88b39
STEP 4/9: RUN yum repolist --disablerepo=*
Updating Subscription Management repositories.
```

...


```

--> b067f1d63eb
STEP 5/9: RUN subscription-manager repos --enable rhocp-4.9-for-rhel-8-x86_64-rpms
Repository 'rhocp-4.9-for-rhel-8-x86_64-rpms' is enabled for this system.
time="2022-02-03T20:28:40Z" level=warning msg="Adding metacopy option, configured globally"
--> 03927607ebd
STEP 6/9: RUN yum -y update
Updating Subscription Management repositories.

...

Upgraded:
  systemd-239-51.el8_5.3.x86_64      systemd-libs-239-51.el8_5.3.x86_64
  systemd-pam-239-51.el8_5.3.x86_64
Installed:
  diffutils-3.6-6.el8.x86_64      libxkbcommon-0.9.1-1.el8.x86_64
  xkeyboard-config-2.28-1.el8.noarch

Complete!
time="2022-02-03T20:29:05Z" level=warning msg="Adding metacopy option, configured globally"
--> db57e92ff63
STEP 7/9: RUN yum install -y openshift-clients.x86_64
Updating Subscription Management repositories.

...

Installed:
  bash-completion-1:2.7-5.el8.noarch
  libpkgconf-1.4.2-1.el8.x86_64
  openshift-clients-4.9.0-202201211735.p0.g3f16530.assembly.stream.el8.x86_64
  pkgconf-1.4.2-1.el8.x86_64
  pkgconf-m4-1.4.2-1.el8.noarch
  pkgconf-pkg-config-1.4.2-1.el8.x86_64

Complete!
time="2022-02-03T20:29:19Z" level=warning msg="Adding metacopy option, configured globally"
--> 609507b059e
STEP 8/9: ENV "OPENSIFT_BUILD_NAME"="my-csi-bc-1"
"OPENSIFT_BUILD_NAMESPACE"="my-csi-app-namespace"
--> cab2da3efc4
STEP 9/9: LABEL "io.openshift.build.name"="my-csi-bc-1"
"io.openshift.build.namespace"="my-csi-app-namespace"
COMMIT temp.builder.openshift.io/my-csi-app-namespace/my-csi-bc-1:edfe12ca
--> 821b582320b
Successfully tagged temp.builder.openshift.io/my-csi-app-namespace/my-csi-bc-1:edfe12ca
821b582320b41f1d7bab4001395133f86fa9cc99cc0b2b64c5a53f2b6750db91
Build complete, no image push requested

```

2.10.6. Additional resources

- [Importing simple content access certificates with Insights Operator](#)

- [Enabling features using feature gates](#)
- [Managing image streams](#)
- [build strategy](#)

2.11. SECURING BUILDS BY STRATEGY

Builds in OpenShift Container Platform are run in privileged containers. Depending on the build strategy used, if you have privileges, you can run builds to escalate their permissions on the cluster and host nodes. And as a security measure, it limits who can run builds and the strategy that is used for those builds. Custom builds are inherently less safe than source builds, because they can execute any code within a privileged container, and are disabled by default. Grant docker build permissions with caution, because a vulnerability in the Dockerfile processing logic could result in a privileges being granted on the host node.

By default, all users that can create builds are granted permission to use the docker and Source-to-image (S2I) build strategies. Users with cluster administrator privileges can enable the custom build strategy, as referenced in the restricting build strategies to a user globally section.

You can control who can build and which build strategies they can use by using an authorization policy. Each build strategy has a corresponding build subresource. A user must have permission to create a build and permission to create on the build strategy subresource to create builds using that strategy. Default roles are provided that grant the create permission on the build strategy subresource.

Table 2.3. Build Strategy Subresources and Roles

Strategy	Subresource	Role
Docker	builds/docker	system:build-strategy-docker
Source-to-Image	builds/source	system:build-strategy-source
Custom	builds/custom	system:build-strategy-custom
JenkinsPipeline	builds/jenkinspipeline	system:build-strategy-jenkinspipeline

2.11.1. Disabling access to a build strategy globally

To prevent access to a particular build strategy globally, log in as a user with cluster administrator privileges, remove the corresponding role from the **system:authenticated** group, and apply the annotation **rbac.authorization.kubernetes.io/autoupdate: "false"** to protect them from changes between the API restarts. The following example shows disabling the docker build strategy.

Procedure

1. Apply the **rbac.authorization.kubernetes.io/autoupdate** annotation:

```
$ oc edit clusterrolebinding system:build-strategy-docker-binding
```

Example output

```
-
```

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "false" ❶
  creationTimestamp: 2018-08-10T01:24:14Z
  name: system:build-strategy-docker-binding
  resourceVersion: "225"
  selfLink: /apis/rbac.authorization.k8s.io/v1/clusterrolebindings/system%3Abuild-strategy-
  docker-binding
  uid: 17b1f3d4-9c3c-11e8-be62-0800277d20bf
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:build-strategy-docker
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:authenticated

```

- ❶ Change the **rbac.authorization.kubernetes.io/autoupdate** annotation's value to **"false"**.

2. Remove the role:

```

$ oc adm policy remove-cluster-role-from-group system:build-strategy-docker
system:authenticated

```

3. Ensure the build strategy subresources are also removed from these roles:

```

$ oc edit clusterrole admin

```

```

$ oc edit clusterrole edit

```

4. For each role, specify the subresources that correspond to the resource of the strategy to disable.

- a. Disable the docker Build Strategy for **admin**:

```

kind: ClusterRole
metadata:
  name: admin
...
- apiGroups:
  - ""
  - build.openshift.io
resources:
  - buildconfigs
  - buildconfigs/webhooks
  - builds/custom ❶
  - builds/source
verbs:
  - create
  - delete

```

```
- deletecollection
- get
- list
- patch
- update
- watch
...
```

- 1 Add **builds/custom** and **builds/source** to disable docker builds globally for users with the **admin** role.

2.11.2. Restricting build strategies to users globally

You can allow a set of specific users to create builds with a particular strategy.

Prerequisites

- Disable global access to the build strategy.

Procedure

- Assign the role that corresponds to the build strategy to a specific user. For example, to add the **system:build-strategy-docker** cluster role to the user **devuser**:

```
$ oc adm policy add-cluster-role-to-user system:build-strategy-docker devuser
```



WARNING

Granting a user access at the cluster level to the **builds/docker** subresource means that the user can create builds with the docker strategy in any project in which they can create builds.

2.11.3. Restricting build strategies to a user within a project

Similar to granting the build strategy role to a user globally, you can allow a set of specific users within a project to create builds with a particular strategy.

Prerequisites

- Disable global access to the build strategy.

Procedure

- Assign the role that corresponds to the build strategy to a specific user within a project. For example, to add the **system:build-strategy-docker** role within the project **devproject** to the user **devuser**:

```
$ oc adm policy add-role-to-user system:build-strategy-docker devuser -n devproject
```

2.12. BUILD CONFIGURATION RESOURCES

Use the following procedure to configure build settings.

2.12.1. Build controller configuration parameters

The **build.config.openshift.io/cluster** resource offers the following configuration parameters.

Parameter	Description
Build	<p>Holds cluster-wide information on how to handle builds. The canonical, and only valid name is cluster.</p> <p>spec: Holds user-settable values for the build controller configuration.</p>
buildDefaults	<p>Controls the default information for builds.</p> <p>defaultProxy: Contains the default proxy settings for all build operations, including image pull or push and source download.</p> <p>You can override values by setting the HTTP_PROXY, HTTPS_PROXY, and NO_PROXY environment variables in the BuildConfig strategy.</p> <p>gitProxy: Contains the proxy settings for Git operations only. If set, this overrides any proxy settings for all Git commands, such as git clone.</p> <p>Values that are not set here are inherited from DefaultProxy.</p> <p>env: A set of default environment variables that are applied to the build if the specified variables do not exist on the build.</p> <p>imageLabels: A list of labels that are applied to the resulting image. You can override a default label by providing a label with the same name in the BuildConfig.</p> <p>resources: Defines resource requirements to execute the build.</p>
ImageLabel	<p>name: Defines the name of the label. It must have non-zero length.</p>
buildOverrides	<p>Controls override settings for builds.</p> <p>imageLabels: A list of labels that are applied to the resulting image. If you provided a label in the BuildConfig with the same name as one in this table, your label will be overwritten.</p> <p>nodeSelector: A selector which must be true for the build pod to fit on a node.</p> <p>tolerations: A list of tolerations that overrides any existing tolerations set on a build pod.</p>
BuildList	<p>items: Standard object's metadata.</p>

2.12.2. Configuring build settings

You can configure build settings by editing the **build.config.openshift.io/cluster** resource.

Procedure

- Edit the **build.config.openshift.io/cluster** resource:

```
$ oc edit build.config.openshift.io/cluster
```

The following is an example **build.config.openshift.io/cluster** resource:

```
apiVersion: config.openshift.io/v1
kind: Build 1
metadata:
  annotations:
    release.openshift.io/create-only: "true"
    creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 2
  name: cluster
  resourceVersion: "107233"
  selfLink: /apis/config.openshift.io/v1/builds/cluster
  uid: e2e9cc14-78a9-11e9-b92b-06d6c7da38dc
spec:
  buildDefaults: 2
  defaultProxy: 3
    httpProxy: http://proxy.com
    httpsProxy: https://proxy.com
    noProxy: internal.com
  env: 4
    - name: envkey
      value: envvalue
  gitProxy: 5
    httpProxy: http://gitproxy.com
    httpsProxy: https://gitproxy.com
    noProxy: internalgit.com
  imageLabels: 6
    - name: labelkey
      value: labelvalue
  resources: 7
    limits:
      cpu: 100m
      memory: 50Mi
    requests:
      cpu: 10m
      memory: 10Mi
  buildOverrides: 8
  imageLabels: 9
    - name: labelkey
      value: labelvalue
  nodeSelector: 10
    selectorkey: selectorvalue
  tolerations: 11
```

```
- effect: NoSchedule
  key: node-role.kubernetes.io/builds
operator: Exists
```

- 1 **Build:** Holds cluster-wide information on how to handle builds. The canonical, and only valid name is **cluster**.
- 2 **buildDefaults:** Controls the default information for builds.
- 3 **defaultProxy:** Contains the default proxy settings for all build operations, including image pull or push and source download.
- 4 **env:** A set of default environment variables that are applied to the build if the specified variables do not exist on the build.
- 5 **gitProxy:** Contains the proxy settings for Git operations only. If set, this overrides any Proxy settings for all Git commands, such as **git clone**.
- 6 **imageLabels:** A list of labels that are applied to the resulting image. You can override a default label by providing a label with the same name in the **BuildConfig**.
- 7 **resources:** Defines resource requirements to execute the build.
- 8 **buildOverrides:** Controls override settings for builds.
- 9 **imageLabels:** A list of labels that are applied to the resulting image. If you provided a label in the **BuildConfig** with the same name as one in this table, your label will be overwritten.
- 10 **nodeSelector:** A selector which must be true for the build pod to fit on a node.
- 11 **tolerations:** A list of tolerations that overrides any existing tolerations set on a build pod.

2.13. TROUBLESHOOTING BUILDS

Use the following to troubleshoot build issues.

2.13.1. Resolving denial for access to resources

If your request for access to resources is denied:

Issue

A build fails with:

```
requested access to the resource is denied
```

Resolution

You have exceeded one of the image quotas set on your project. Check your current quota and verify the limits applied and storage in use:

```
$ oc describe quota
```

2.13.2. Service certificate generation failure

If your request for access to resources is denied:

Issue

If a service certificate generation fails with (service's **service.beta.openshift.io/serving-cert-generation-error** annotation contains):

Example output

```
secret/ssl-key references serviceUID 62ad25ca-d703-11e6-9d6f-0e9c0057b608, which does not match 77b6dd80-d716-11e6-9d6f-0e9c0057b60
```

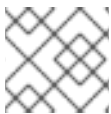
Resolution

The service that generated the certificate no longer exists, or has a different **serviceUID**. You must force certificates regeneration by removing the old secret, and clearing the following annotations on the service: **service.beta.openshift.io/serving-cert-generation-error** and **service.beta.openshift.io/serving-cert-generation-error-num**:

```
$ oc delete secret <secret_name>
```

```
$ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-
```

```
$ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-num-
```



NOTE

The command removing annotation has a - after the annotation name to be removed.

2.14. SETTING UP ADDITIONAL TRUSTED CERTIFICATE AUTHORITIES FOR BUILDS

Use the following sections to set up additional certificate authorities (CA) to be trusted by builds when pulling images from an image registry.

The procedure requires a cluster administrator to create a **ConfigMap** and add additional CAs as keys in the **ConfigMap**.

- The **ConfigMap** must be created in the **openshift-config** namespace.
- **domain** is the key in the **ConfigMap** and **value** is the PEM-encoded certificate.
 - Each CA must be associated with a domain. The domain format is **hostname[..port]**.
- The **ConfigMap** name must be set in the **image.config.openshift.io/cluster** cluster scoped configuration resource's **spec.additionalTrustedCA** field.

2.14.1. Adding certificate authorities to the cluster

You can add certificate authorities (CA) to the cluster for use when pushing and pulling images with the following procedure.

Prerequisites

- You must have access to the public certificates of the registry, usually a **hostname/ca.crt** file located in the **/etc/docker/certs.d/** directory.

Procedure

1. Create a **ConfigMap** in the **openshift-config** namespace containing the trusted certificates for the registries that use self-signed certificates. For each CA file, ensure the key in the **ConfigMap** is the hostname of the registry in the **hostname[..port]** format:

```
$ oc create configmap registry-cas -n openshift-config \
--from-file=myregistry.corp.com..5000=/etc/docker/certs.d/myregistry.corp.com:5000/ca.crt \
--from-file=otherregistry.com=/etc/docker/certs.d/otherregistry.com/ca.crt
```

2. Update the cluster image configuration:

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
{"name":"registry-cas"}}}' --type=merge
```

2.14.2. Additional resources

- [Create a **ConfigMap**](#)
- [Secrets and **ConfigMaps**](#)
- [Configuring a custom PKI](#)

CHAPTER 3. PIPELINES

3.1. RED HAT OPENSIFT PIPELINES RELEASE NOTES

Red Hat OpenShift Pipelines is a cloud-native CI/CD experience based on the Tekton project which provides:

- Standard Kubernetes-native pipeline definitions (CRDs).
- Serverless pipelines with no CI server management overhead.
- Extensibility to build images using any Kubernetes tool, such as S2I, Buildah, JIB, and Kaniko.
- Portability across any Kubernetes distribution.
- Powerful CLI for interacting with pipelines.
- Integrated user experience with the **Developer** perspective of the OpenShift Container Platform web console.

For an overview of Red Hat OpenShift Pipelines, see [Understanding OpenShift Pipelines](#).

3.1.1. Compatibility and support matrix

Some features in this release are currently in [Technology Preview](#). These experimental features are not intended for production use.

In the table, features are marked with the following statuses:

TP	Technology Preview
GA	General Availability

Table 3.1. Compatibility and support matrix

Red Hat OpenShift Pipelines Version	Component Version							OpenShift Version	Support Status
	Operator	Pipelines	Triggers	CLI	Catalog	Chains	Hub	Pipelines as Code	

Red Hat OpenShift Pipelines Version	Component Version							OpenShift Version	Support Status
1.9	0.41.x	0.22.x	0.28.x	NA	0.13.x (TP)	1.11.x	0.15.x (GA)	4.11, 4.12, 4.13 (planned)	GA
1.8	0.37.x	0.20.x	0.24.x	NA	0.9.0 (TP)	1.8.x (TP)	0.10.x (TP)	4.10, 4.11, 4.12 (planned)	GA
1.7	0.33.x	0.19.x	0.23.x	0.33	0.8.0 (TP)	1.7.0 (TP)	0.5.4 (TP)	4.9, 4.10	GA
1.6	0.28.x	0.16.x	0.21.x	0.28	N/A	N/A	N/A	4.9	GA
1.5	0.24.x	0.14.x (TP)	0.19.x	0.24	N/A	N/A	N/A	4.8	GA
1.4	0.22.x	0.12.x (TP)	0.17.x	0.22	N/A	N/A	N/A	4.7	GA

Additionally, support for running Red Hat OpenShift Pipelines on ARM hardware is in [Technology Preview](#).

For questions and feedback, you can send an email to the product team at pipelines-interest@redhat.com.

3.1.2. Making open source more inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

3.1.3. Release notes for Red Hat OpenShift Pipelines General Availability 1.9

With this update, Red Hat OpenShift Pipelines General Availability (GA) 1.9 is available on OpenShift Container Platform 4.11 and later versions.

3.1.3.1. New features

In addition to the fixes and stability improvements, the following sections highlight what is new in Red Hat OpenShift Pipelines 1.9.

3.1.3.1.1. Pipelines

- With this update, you can specify pipeline parameters and results in arrays and object dictionary forms.
- This update provides support for Container Storage Interface (CSI) and projected volumes for your workspace.
- With this update, you can specify the **stdoutConfig** and **stderrConfig** parameters when defining pipeline steps. Defining these parameters helps to capture standard output and standard error, associated with steps, to local files.
- With this update, you can add variables in the **steps[].onError** event handler, for example, **\$(params.CONTINUE)**.
- With this update, you can use the output from the **finally** task in the **PipelineResults** definition. For example, **\$(finally.<pipelinetask-name>.result.<result-name>)**, where **<pipelinetask-name>** denotes the pipeline task name and **<result-name>** denotes the result name.
- This update supports task-level resource requirements for a task run.
- With this update, you do not need to recreate parameters that are shared, based on their names, between a pipeline and the defined tasks. This update is part of a developer preview feature.
- This update adds support for remote resolution, such as built-in git, cluster, bundle, and hub resolvers.

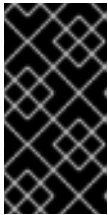
3.1.3.1.2. Triggers

- This update adds the **Interceptor** CRD to define **NamespacedInterceptor**. You can use **NamespacedInterceptor** in the **kind** section of interceptors reference in triggers or in the **EventListener** specification.
- This update enables **CloudEvents**.
- With this update, you can configure the webhook port number when defining a trigger.
- This update supports using trigger **eventID** as input to **TriggerBinding**.
- This update supports validation and rotation of certificates for the **ClusterInterceptor** server.
 - Triggers perform certificate validation for core interceptors and rotate a new certificate to **ClusterInterceptor** when its certificate expires.

3.1.3.1.3. CLI

- This update supports showing annotations in the **describe** command.
- This update supports showing pipeline, tasks, and timeout in the **pr describe** command.
- This update adds flags to provide pipeline, tasks, and timeout in the **pipeline start** command.
- This update supports showing the presence of workspace, optional or mandatory, in the **describe** command of a task and pipeline.

- This update adds the **timestamps** flag to show logs with a timestamp.
- This update adds a new flag **--ignore-running-pipelinerun**, which ignores the deletion of **TaskRun** associated with **PipelineRun**.
- This update adds support for experimental commands. This update also adds experimental subcommands, **sign** and **verify** to the **tkn** CLI tool.
- This update makes the Z shell (Zsh) completion feature usable without generating any files.
- This update introduces a new CLI tool called **opc**. It is anticipated that an upcoming release will replace the **tkn** CLI tool with **opc**.



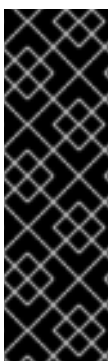
IMPORTANT

- The new CLI tool **opc** is a Technology Preview feature.
- **opc** will be a replacement for **tkn** with additional Red Hat OpenShift Pipelines specific features, which do not necessarily fit in **tkn**.

3.1.3.1.4. Operator

- With this update, you can install Pipelines as Code as a separate component, not as a part of **TektonAddon**. You can configure Pipelines as Code in the **TektonConfig** CRD.
- With this update, you can also modify Pipelines as Code configurations in the **TektonConfig** CRD.
- With this update, if you disable the developer perspective, the Operator does not install developer console related custom resources.
- This update includes **ClusterTriggerBinding** support for Bitbucket Server and Bitbucket Cloud and helps you to reuse a **TriggerBinding** across your entire cluster.

3.1.3.1.5. Resolvers



IMPORTANT

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

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

- With this update, you can configure pipeline resolvers in the **TektonConfig** CRD. You can enable or disable these pipeline resolvers: **enable-bundles-resolver**, **enable-cluster-resolver**, **enable-git-resolver**, and **enable-hub-resolver**.

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
```

```
spec:
  pipeline:
    enable-bundles-resolver: true
    enable-cluster-resolver: true
    enable-git-resolver: true
    enable-hub-resolver: true
  ...
```

You can also provide resolver specific configurations in **TektonConfig**. For example, you can define the following fields in the **map[string]string** format to set configurations for individual resolvers:

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    bundles-resolver-config:
      default-service-account: pipelines
    cluster-resolver-config:
      default-namespace: test
    git-resolver-config:
      server-url: localhost.com
    hub-resolver-config:
      default-tekton-hub-catalog: tekton
  ...
```

3.1.3.1.6. Tekton Chains



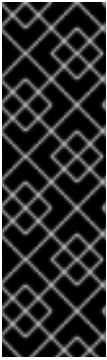
IMPORTANT

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

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

- Before this update, only Open Container Initiative (OCI) images were supported as outputs of **TaskRun** in the in-toto provenance agent. This update adds in-toto provenance metadata as outputs with these suffixes, **ARTIFACT_URI** and **ARTIFACT_DIGEST**.
- Before this update, only **TaskRun** attestations were supported. This update adds support for **PipelineRun** attestations as well.
- This update adds support for Tekton Chains to get the **imgPullSecret** parameter from the pod template. This update helps you to configure repository authentication based on each pipeline run or task run without modifying the service account.

3.1.3.1.7. Tekton Hub



IMPORTANT

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

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

- With this update, as an administrator, you can use an external database, such as Crunchy PostgreSQL with Tekton Hub, instead of using the default Tekton Hub database. This update helps you to perform the following actions:
 - Specify the coordinates of an external database to be used with Tekton Hub
 - Disable the default Tekton Hub database deployed by the Operator
- This update removes the dependency of **config.yaml** from external Git repositories and moves the complete configuration data into the API **ConfigMap**. This update helps an administrator to perform the following actions:
 - Add the configuration data, such as categories, catalogs, scopes, and defaultScopes in the Tekton Hub custom resource.
 - Modify Tekton Hub configuration data on the cluster. All modifications are preserved upon Operator upgrades.
 - Update the list of catalogs for Tekton Hub
 - Change the categories for Tekton Hub



NOTE

If you do not add any configuration data, you can use the default data in the API **ConfigMap** for Tekton Hub configurations.

3.1.3.1.8. Pipelines as Code

- This update adds support for concurrency limit in the **Repository** CRD to define the maximum number of **PipelineRuns** running for a repository at a time. The **PipelineRuns** from a pull request or a push event are queued in alphabetical order.
- This update adds a new command **tkn pac logs** for showing the logs of the latest pipeline run for a repository.
- This update supports advanced event matching on file path for push and pull requests to GitHub and GitLab. For example, you can use the Common Expression Language (CEL) to run a pipeline only if a path has changed for any markdown file in the **docs** directory.

```
...
annotations:
  pipelinesascode.tekton.dev/on-cel-expression: |
    event == "pull_request" && "docs/*.md".pathChanged()
```

- With this update, you can reference a remote pipeline in the **pipelineRef:** object using annotations.
- With this update, you can auto-configure new GitHub repositories with Pipelines as Code, which sets up a namespace and creates a **Repository** CRD for your GitHub repository.
- With this update, Pipelines as Code generates metrics for **PipelineRuns** with provider information.
- This update provides the following enhancements for the **tkn-pac** plugin:
 - Detects running pipelines correctly
 - Fixes showing duration when there is no failure completion time
 - Shows an error snippet and highlights the error regular expression pattern in the **tkn-pac describe** command
 - Adds the **use-real-time** switch to the **tkn-pac ls** and **tkn-pac describe** commands
 - Imports the **tkn-pac** logs documentation
 - Shows **pipelineruntimeout** as a failure in the **tkn-pac ls** and **tkn-pac describe** commands.
 - Show a specific pipeline run failure with the **--target-pipeline-run** option.
- With this update, you can view the errors for your pipeline run in the form of a version control system (VCS) comment or a small snippet in the GitHub checks.
- With this update, Pipelines as Code optionally can detect errors inside the tasks if they are of a simple format and add those tasks as annotations in GitHub. This update is part of a developer preview feature.
- This update adds the following new commands:
 - **tkn-pac webhook add:** Adds a webhook to project repository settings and updates the **webhook.secret** key in the existing **k8s Secret** object without updating the repository.
 - **tkn-pac webhook update-token:** Updates provider token for an existing **k8s Secret** object without updating the repository.
- This update enhances functionality of the **tkn-pac create repo** command, which creates and configures webhooks for GitHub, GitLab, and BitbucketCloud along with creating repositories.
- With this update, the **tkn-pac describe** command shows the latest fifty events in a sorted order.
- This update adds the **--last** option to the **tkn-pac logs** command.
- With this update, the **tkn-pac resolve** command prompts for a token on detecting a **git_auth_secret** in the file template.
- With this update, Pipelines as Code hides secrets from log snippets to avoid exposing secrets in the GitHub interface.
- With this update, the secrets automatically generated for **git_auth_secret** are an owner reference with **PipelineRun**. The secrets get cleaned with the **PipelineRun**, not after the pipeline run execution.

- This update adds support to cancel a pipeline run with the **/cancel** comment.
- Before this update, the GitHub apps token scoping was not defined and tokens would be used on every repository installation. With this update, you can scope the GitHub apps token to the target repository using the following parameters:
 - **secret-github-app-token-scoped**: Scopes the app token to the target repository, not to every repository the app installation has access to.
 - **secret-github-app-scope-extra-repos**: Customizes the scoping of the app token with an additional owner or repository.
- With this update, you can use Pipelines as Code with your own Git repositories that are hosted on GitLab.
- With this update, you can access pipeline execution details in the form of kubernetes events in your namespace. These details help you to troubleshoot pipeline errors without needing access to admin namespaces.
- This update supports authentication of URLs in the Pipelines as Code resolver with the Git provider.
- With this update, you can set the name of the hub catalog by using a setting in the **pipelines-as-code** config map.
- With this update, you can set the maximum and default limits for the **max-keep-run** parameter.
- This update adds documents on how to inject custom Secure Sockets Layer (SSL) certificates in Pipelines as Code to let you connect to provider instance with custom certificates.
- With this update, the **PipelineRun** resource definition has the log URL included as an annotation. For example, the **tkn-pac describe** command shows the log link when describing a **PipelineRun**.
- With this update, **tkn-pac** logs show repository name, instead of **PipelineRun** name.

3.1.3.2. Breaking changes

- With this update, the **Conditions** custom resource definition (CRD) type has been removed. As an alternative, use the **WhenExpressions** instead.
- With this update, support for **tekton.dev/v1alpha1** API pipeline resources, such as Pipeline, PipelineRun, Task, Clustertask, and TaskRun has been removed.
- With this update, the **tkn-pac setup** command has been removed. You can now re-use the functionality provided by the **tkn-pac setup** command by using the **tkn-pac webhook add** and **tkn-pac webhook update-token** commands. You can use the **tkn-pac webhook add** command to re-add a webhook to an existing Git repository. You can use the **tkn-pac webhook update-token** command to update the personal provider access token for an existing Secret object in the Git repository.

3.1.3.3. Deprecated and removed features

- In the Red Hat OpenShift Pipelines 1.9.0 release, **ClusterTasks** are deprecated and planned to be removed in a future release. As an alternative, you can use **Cluster Resolver**.
- In the Red Hat OpenShift Pipelines 1.9.0 release, the use of the **triggers** and the

namespaceSelector fields in a single **EventListener** specification is deprecated and planned to be removed in a future release. You can use these fields in different **EventListener** specifications successfully.

- In the Red Hat OpenShift Pipelines 1.9.0 release, the **tkn pipelinerun describe** command does not display timeouts for the **PipelineRun** resource.
- In the Red Hat OpenShift Pipelines 1.9.0 release, the `PipelineResource`` custom resource (CR) is deprecated. The **PipelineResource** CR was a Tech Preview feature and part of the **tekton.dev/v1alpha1** API.
- In the Red Hat OpenShift Pipelines 1.9.0 release, custom image parameters from cluster tasks are deprecated. As an alternative, you can copy a cluster task and use your custom image in it.

3.1.3.4. Known issues

- The **chains-secret** and **chains-config** config maps are removed after you uninstall the Red Hat OpenShift Pipelines Operator. As they contain user data, they should be preserved and not deleted.
- When running the **tkn pac** set of commands on Windows, you may receive the following error message: **Command finished with error: not supported by Windows.**
Workaround: Set the **NO_COLOR** environment variable to **true**.
- Running the **tkn pac resolve -f <filename> | oc create -f** command may not provide expected results, if the **tkn pac resolve** command uses a templated parameter value to function.
Workaround: To mitigate this issue, save the output of **tkn pac resolve** in a temporary file by running the **tkn pac resolve -f <filename> -o tempfile.yaml** command and then run the **oc create -f tempfile.yaml** command. For example, **tkn pac resolve -f <filename> -o /tmp/pull-request-resolved.yaml && kubectl create -f /tmp/pull-request-resolved.yaml**.

3.1.3.5. Fixed issues

- Before this update, after replacing an empty array, the original array returned an empty string rendering the parameters inside it invalid. With this update, this issue is resolved and the original array returns as empty.
- Before this update, if duplicate secrets were present in a service account for a pipelines run, it resulted in failure in task pod creation. With this update, this issue is resolved and the task pod is created successfully even if duplicate secrets are present in a service account.
- Before this update, by looking at the TaskRun's **spec.StatusMessage** field, users could not distinguish whether the **TaskRun** had been cancelled by the user or by a **PipelineRun** that was part of it. With this update, this issue is resolved and users can distinguish the status of the **TaskRun** by looking at the TaskRun's **spec.StatusMessage** field.
- Before this update, webhook validation was removed on deletion of old versions of invalid objects. With this update, this issue is resolved.
- Before this update, a race condition could occur if another tool updated labels or annotations on a PipelineRun or TaskRun. With this update, this issue is resolved and you can merge labels or annotations.
- Before this update, log keys did not have the same keys as in pipelines controllers. With this update, this issue has been resolved and the log keys have been updated to match the log stream of pipeline controllers. The keys in logs have been changed from "ts" to "timestamp",

from "level" to "severity", and from "message" to "msg".

- Before this update, if a PipelineRun was deleted with an unknown status, an error message was not generated. With this update, this issue is resolved and an error message is generated.
- Before this update, to access bundle commands like **list** and **push**, it was required to use the **kubeconfig** file. With this update, this issue has been resolved and the **kubeconfig** file is not required to access bundle commands.
- Before this update, if the parent pipelinerun was running while deleting TaskRuns, then TaskRuns would be deleted. With this update, this issue is resolved and TaskRuns are not getting deleted if the parent PipelineRun is running.
- Before this update, if the user attempted to build a bundle with more objects than the pipeline controller permitted, the Tekton CLI did not display an error message. With this update, this issue is resolved and the Tekton CLI displays an error message if the user attempts to build a bundle with more objects than the limit permitted in the pipeline controller.
- Before this update, if namespaces were removed from the cluster, then the operator did not remove namespaces from the **ClusterInterceptor ClusterRoleBinding** subjects. With this update, this issue has been resolved, and the operator removes the namespaces from the **ClusterInterceptor ClusterRoleBinding** subjects.
- Before this update, the default installation of the Red Hat OpenShift Pipelines Operator resulted in the **pipelines-scc-rolebinding security context constraint** (SCC) role binding resource remaining in the cluster. With this update, the default installation of the Red Hat OpenShift Pipelines Operator results in the **pipelines-scc-rolebinding security context constraint** (SCC) role binding resource being removed from the cluster.
- Before this update, Pipelines as Code did not get updated values from the Pipelines as Code ConfigMap object. With this update, this issue is fixed and Pipelines as Code ConfigMap object looks for any new changes.
- Before this update, Pipelines as Code controller did not wait for the **tekton.dev/pipeline** label to be updated and added the **checkrun id** label, which would cause race conditions. With this update, the Pipelines as Code controller waits for the **tekton.dev/pipeline** label to be updated and then adds the **checkrun id** label, which helps to avoid race conditions.
- Before this update, the **tkn-pac create repo** command did not override a **PipelineRun** if it already existed in the git repository. With this update, **tkn-pac create** command is fixed to override a **PipelineRun** if it exists in the git repository and this resolves the issue successfully.
- Before this update, the **tkn pac describe** command did not display reasons for every message. With this update, this issue is fixed and the **tkn pac describe** command displays reasons for every message.
- Before this update, a pull request failed if the user in the annotation provided values by using a regex form, for example, **refs/head/rel-***. The pull request failed because it was missing **refs/heads** in its base branch. With this update, the prefix is added and checked that it matches. This resolves the issue and the pull request is successful.

3.1.4. Release notes for Red Hat OpenShift Pipelines General Availability 1.8

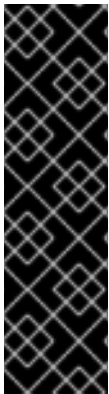
With this update, Red Hat OpenShift Pipelines General Availability (GA) 1.8 is available on OpenShift Container Platform 4.10, and is planned to be available on OpenShift Container Platform 4.11 and 4.12.

3.1.4.1. New features

In addition to the fixes and stability improvements, the following sections highlight what is new in Red Hat OpenShift Pipelines 1.8.

3.1.4.1.1. Pipelines

- With this update, you can run Red Hat OpenShift Pipelines GA 1.8 and later on an OpenShift Container Platform cluster that is running on ARM hardware. This includes support for **ClusterTask** resources and the **tkn** CLI tool.



IMPORTANT

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

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

- This update implements **Step** and **Sidecar** overrides for **TaskRun** resources.
- This update adds minimal **TaskRun** and **Run** statuses within **PipelineRun** statuses. To enable this feature, in the **TektonConfig** custom resource definition, in the **pipeline** section, you must set the **enable-api-fields** field to **alpha**.
- With this update, the graceful termination of pipeline runs feature is promoted from an alpha feature to a stable feature. As a result, the previously deprecated **PipelineRunCancelled** status remains deprecated and is planned to be removed in a future release. Because this feature is available by default, you no longer need to set the **pipeline.enable-api-fields** field to **alpha** in the **TektonConfig** custom resource definition.
- With this update, you can specify the workspace for a pipeline task by using the name of the workspace. This change makes it easier to specify a shared workspace for a pair of **Pipeline** and **PipelineTask** resources. You can also continue to map workspaces explicitly. To enable this feature, in the **TektonConfig** custom resource definition, in the **pipeline** section, you must set the **enable-api-fields** field to **alpha**.
- With this update, parameters in embedded specifications are propagated without mutations.
- With this update, you can specify the required metadata of a **Task** resource referenced by a **PipelineRun** resource by using annotations and labels. This way, **Task** metadata that depends on the execution context is available during the pipeline run.
- This update adds support for object or dictionary types in **params** and **results** values. This change affects backward compatibility and sometimes breaks forward compatibility, such as using an earlier client with a later Red Hat OpenShift Pipelines version. This update changes the **ArrayOfStruct** structure, which affects projects that use the Go language API as a library.
- This update adds a **SkippingReason** value to the **SkippedTasks** field of the **PipelineRun** status fields so that users know why a given PipelineTask was skipped.

- This update supports an alpha feature in which you can use an **array** type for emitting results from a **Task** object. The result type is changed from **string** to **ArrayOfString**. For example, a task can specify a type to produce an array result:

```
kind: Task
apiVersion: tekton.dev/v1beta1
metadata:
  name: write-array
  annotations:
    description: |
      A simple task that writes array
spec:
  results:
    - name: array-results
      type: array
      description: The array results
  ...
```

Additionally, you can run a task script to populate the results with an array:

```
$ echo -n "[\"hello\", \"world\"]" | tee $(results.array-results.path)
```

To enable this feature, in the **TektonConfig** custom resource definition, in the **pipeline** section, you must set the **enable-api-fields** field to **alpha**.

This feature is in progress and is part of TEP-0076.

3.1.4.1.2. Triggers

- This update transitions the **TriggerGroups** field in the **EventListener** specification from an alpha feature to a stable feature. Using this field, you can specify a set of interceptors before selecting and running a group of triggers. Because this feature is available by default, you no longer need to set the **pipeline.enable-api-fields** field to **alpha** in the **TektonConfig** custom resource definition.
- With this update, the **Trigger** resource supports end-to-end secure connections by running the **ClusterInterceptor** server using HTTPS.

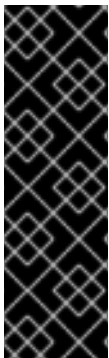
3.1.4.1.3. CLI

- With this update, you can use the **tkn taskrun export** command to export a live task run from a cluster to a YAML file, which you can use to import the task run to another cluster.
- With this update, you can add the **-o name** flag to the **tkn pipeline start** command to print the name of the pipeline run right after it starts.
- This update adds a list of available plugins to the output of the **tkn --help** command.
- With this update, while deleting a pipeline run or task run, you can use both the **--keep** and **keep-since** flags together.
- With this update, you can use **Cancelled** as the value of the **spec.status** field rather than the deprecated **PipelineRunCancelled** value.

3.1.4.1.4. Operator

- With this update, as an administrator, you can configure your local Tekton Hub instance to use a custom database rather than the default database.
- With this update, as a cluster administrator, if you enable your local Tekton Hub instance, it periodically refreshes the database so that changes in the catalog appear in the Tekton Hub web console. You can adjust the period between refreshes.
Previously, to add the tasks and pipelines in the catalog to the database, you performed that task manually or set up a cron job to do it for you.
- With this update, you can install and run a Tekton Hub instance with minimal configuration. This way, you can start working with your teams to decide which additional customizations they might want.
- This update adds **GIT_SSL_CAINFO** to the **git-clone** task so you can clone secured repositories.

3.1.4.1.5. Tekton Chains



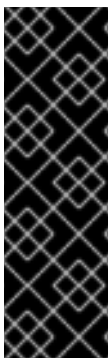
IMPORTANT

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

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

- With this update, you can log in to a vault by using OIDC rather than a static token. This change means that Spire can generate the OIDC credential so that only trusted workloads are allowed to log in to the vault. Additionally, you can pass the vault address as a configuration value rather than inject it as an environment variable.
- The **chains-config** config map for Tekton Chains in the **openshift-pipelines** namespace is automatically reset to default after upgrading the Red Hat OpenShift Pipelines Operator because directly updating the config map is not supported when installed by using the Red Hat OpenShift Pipelines Operator. However, with this update, you can configure Tekton Chains by using the **TektonChain** custom resource. This feature enables your configuration to persist after upgrading, unlike the **chains-config** config map, which gets overwritten during upgrades.

3.1.4.1.6. Tekton Hub



IMPORTANT

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

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

- With this update, if you install a fresh instance of Tekton Hub by using the Operator, the Tekton Hub login is disabled by default. To enable the login and rating features, you must create the Hub API secret while installing Tekton Hub.



NOTE

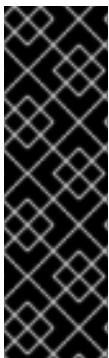
Because Tekton Hub login was enabled by default in Red Hat OpenShift Pipelines 1.7, if you upgrade the Operator, the login is enabled by default in Red Hat OpenShift Pipelines 1.8. To disable this login, see [Disabling Tekton Hub login after upgrading from OpenShift Pipelines 1.7.x --> 1.8.x](#)

- With this update, as an administrator, you can configure your local Tekton Hub instance to use a custom PostgreSQL 13 database rather than the default database. To do so, create a **Secret** resource named **tekton-hub-db**. For example:

```
apiVersion: v1
kind: Secret
metadata:
  name: tekton-hub-db
  labels:
    app: tekton-hub-db
type: Opaque
stringData:
  POSTGRES_HOST: <hostname>
  POSTGRES_DB: <database_name>
  POSTGRES_USER: <user_name>
  POSTGRES_PASSWORD: <user_password>
  POSTGRES_PORT: <listening_port_number>
```

- With this update, you no longer need to log in to the Tekton Hub web console to add resources from the catalog to the database. Now, these resources are automatically added when the Tekton Hub API starts running for the first time.
- This update automatically refreshes the catalog every 30 minutes by calling the catalog refresh API job. This interval is user-configurable.

3.1.4.1.7. Pipelines as Code



IMPORTANT

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

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

- With this update, as a developer, you get a notification from the **tkn-pac** CLI tool if you try to add a duplicate repository to a Pipelines as Code run. When you enter **tkn pac create repository**, each repository must have a unique URL. This notification also helps prevent hijacking exploits.

- With this update, as a developer, you can use the new **tkn-pac setup cli** command to add a Git repository to Pipelines as Code by using the webhook mechanism. This way, you can use Pipelines as Code even when using GitHub Apps is not feasible. This capability includes support for repositories on GitHub, GitLab, and BitBucket.
- With this update, Pipelines as Code supports GitLab integration with features such as the following:
 - ACL (Access Control List) on project or group
 - **/ok-to-test** support from allowed users
 - **/retest** support.
- With this update, you can perform advanced pipeline filtering with Common Expression Language (CEL). With CEL, you can match pipeline runs with different Git provider events by using annotations in the **PipelineRun** resource. For example:

```
...
annotations:
  pipelinesascode.tekton.dev/on-cel-expression: |
    event == "pull_request" && target_branch == "main" && source_branch == "wip"
```

- Previously, as a developer, you could have only one pipeline run in your **.tekton** directory for each Git event, such as a pull request. With this update, you can have multiple pipeline runs in your **.tekton** directory. The web console displays the status and reports of the runs. The pipeline runs operate in parallel and report back to the Git provider interface.
- With this update, you can test or retest a pipeline run by commenting **/test** or **/retest** on a pull request. You can also specify the pipeline run by name. For example, you can enter **/test <pipelinerun_name>** or **/retest <pipelinerun-name>**.
- With this update, you can delete a repository custom resource and its associated secrets by using the new **tkn-pac delete repository** command.

3.1.4.2. Breaking changes

- This update changes the default metrics level of **TaskRun** and **PipelineRun** resources to the following values:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-observability
  namespace: tekton-pipelines
labels:
  app.kubernetes.io/instance: default
  app.kubernetes.io/part-of: tekton-pipelines
data:
  _example: |
    ...
    metrics.taskrun.level: "task"
    metrics.taskrun.duration-type: "histogram"
    metrics.pipelinerun.level: "pipeline"
    metrics.pipelinerun.duration-type: "histogram"
```


- With this update, if an annotation or label is present in both **Pipeline** and **PipelineRun** resources, the value in the **Run** type takes precedence. The same is true if an annotation or label is present in **Task** and **TaskRun** resources.
- In Red Hat OpenShift Pipelines 1.8, the previously deprecated **PipelineRun.Spec.ServiceAccountNames** field has been removed. Use the **PipelineRun.Spec.TaskRunSpecs** field instead.
- In Red Hat OpenShift Pipelines 1.8, the previously deprecated **TaskRun.Status.ResourceResults.ResourceRef** field has been removed. Use the **TaskRun.Status.ResourceResults.ResourceName** field instead.
- In Red Hat OpenShift Pipelines 1.8, the previously deprecated **Conditions** resource type has been removed. Remove the **Conditions** resource from **Pipeline** resource definitions that include it. Use **when** expressions in **PipelineRun** definitions instead.
- For Tekton Chains, the **tekton-provenance** format has been removed in this release. Use the **in-toto** format by setting **"artifacts.taskrun.format": "in-toto"** in the **TektonChain** custom resource instead.
- Red Hat OpenShift Pipelines 1.7.x shipped with Pipelines as Code 0.5.x. The current update ships with Pipelines as Code 0.10.x. This change creates a new route in the **openshift-pipelines** namespace for the new controller. You must update this route in GitHub Apps or webhooks that use Pipelines as Code. To fetch the route, use the following command:

```
$ oc get route -n openshift-pipelines pipelines-as-code-controller \
  --template='https://{ .spec.host }'
```

- With this update, Pipelines as Code renames the default secret keys for the **Repository** custom resource definition (CRD). In your CRD, replace **token** with **provider.token**, and replace **secret** with **webhook.secret**.
- With this update, Pipelines as Code replaces a special template variable with one that supports multiple pipeline runs for private repositories. In your pipeline runs, replace **secret: pac-git-basic-auth-{{repo_owner}}-{{repo_name}}** with **secret: {{ git_auth_secret }}**.
- With this update, Pipelines as Code updates the following commands in the **tkn-pac** CLI tool:
 - Replace **tkn pac repository create** with **tkn pac create repository**.
 - Replace **tkn pac repository delete** with **tkn pac delete repository**.
 - Replace **tkn pac repository list** with **tkn pac list**.

3.1.4.3. Deprecated and removed features

- Starting with OpenShift Container Platform 4.11, the **preview** and **stable** channels for installing and upgrading the Red Hat OpenShift Pipelines Operator are removed. To install and upgrade the Operator, use the appropriate **pipelines-<version>** channel, or the **latest** channel for the most recent stable version. For example, to install the Pipelines Operator version **1.8.x**, use the **pipelines-1.8** channel.



NOTE

In OpenShift Container Platform 4.10 and earlier versions, you can use the **preview** and **stable** channels for installing and upgrading the Operator.

- Support for the **tekton.dev/v1alpha1** API version, which was deprecated in Red Hat OpenShift Pipelines GA 1.6, is planned to be removed in the upcoming Red Hat OpenShift Pipelines GA 1.9 release.

This change affects the pipeline component, which includes the **TaskRun**, **PipelineRun**, **Task**, **Pipeline**, and similar **tekton.dev/v1alpha1** resources. As an alternative, update existing resources to use **apiVersion: tekton.dev/v1beta1** as described in [Migrating From Tekton v1alpha1 to Tekton v1beta1](#).

Bug fixes and support for the **tekton.dev/v1alpha1** API version are provided only through the end of the current GA 1.8 lifecycle.



IMPORTANT

For the **Tekton Operator**, the **operator.tekton.dev/v1alpha1** API version is **not** deprecated. You do not need to make changes to this value.

- In Red Hat OpenShift Pipelines 1.8, the **PipelineResource** custom resource (CR) is available but no longer supported. The **PipelineResource** CR was a Tech Preview feature and part of the **tekton.dev/v1alpha1** API, which had been deprecated and planned to be removed in the upcoming Red Hat OpenShift Pipelines GA 1.9 release.
- In Red Hat OpenShift Pipelines 1.8, the **Condition** custom resource (CR) is removed. The **Condition** CR was part of the **tekton.dev/v1alpha1** API, which has been deprecated and is planned to be removed in the upcoming Red Hat OpenShift Pipelines GA 1.9 release.
- In Red Hat OpenShift Pipelines 1.8, the **gcr.io** image for **gsutil** has been removed. This removal might break clusters with **Pipeline** resources that depend on this image. Bug fixes and support are provided only through the end of the Red Hat OpenShift Pipelines 1.7 lifecycle.
- In Red Hat OpenShift Pipelines 1.8, the **PipelineRun.Status.TaskRuns** and **PipelineRun.Status.Runs** fields are deprecated and are planned to be removed in a future release. See [TEP-0100: Embedded TaskRuns and Runs Status in PipelineRuns](#).
- In Red Hat OpenShift Pipelines 1.8, the **pipelineRunCancelled** state is deprecated and planned to be removed in a future release. Graceful termination of **PipelineRun** objects is now promoted from an alpha feature to a stable feature. (See [TEP-0058: Graceful Pipeline Run Termination](#).) As an alternative, you can use the **Cancelled** state, which replaces the **pipelineRunCancelled** state.

You do not need to make changes to your **Pipeline** and **Task** resources. If you have tools that cancel pipeline runs, you must update tools in the next release. This change also affects tools such as the CLI, IDE extensions, and so on, so that they support the new **PipelineRun** statuses.

Because this feature is available by default, you no longer need to set the **pipeline.enable-api-fields** field to **alpha** in the **TektonConfig** custom resource definition.

- In Red Hat OpenShift Pipelines 1.8, the **timeout** field in **PipelineRun** has been deprecated. Instead, use the **PipelineRun.Timeouts** field, which is now promoted from an alpha feature to a stable feature.
Because this feature is available by default, you no longer need to set the **pipeline.enable-api-fields** field to **alpha** in the **TektonConfig** custom resource definition.
- In Red Hat OpenShift Pipelines 1.8, **init** containers are omitted from the **LimitRange** object's default request calculations.

3.1.4.4. Known issues

- The **s2i-nodejs** pipeline cannot use the **nodejs:14-ubi8-minimal** image stream to perform source-to-image (S2I) builds. Using that image stream produces an **error building at STEP "RUN /usr/libexec/s2i/assemble": exit status 127** message.
Workaround: Use **nodejs:14-ubi8** rather than the **nodejs:14-ubi8-minimal** image stream.
- When you run Maven and Jib-Maven cluster tasks, the default container image is supported only on Intel (x86) architecture. Therefore, tasks will fail on ARM, IBM Power Systems (ppc64le), IBM Z, and LinuxONE (s390x) clusters.
Workaround: Specify a custom image by setting the **MAVEN_IMAGE** parameter value to **maven:3.6.3-adoptopenjdk-11**.

TIP

Before you install tasks that are based on the Tekton Catalog on ARM, IBM Power Systems (ppc64le), IBM Z, and LinuxONE (s390x) using **tkn hub**, verify if the task can be executed on these platforms. To check if **ppc64le** and **s390x** are listed in the "Platforms" section of the task information, you can run the following command: **tkn hub info task <name>**

- On ARM, IBM Power Systems, IBM Z, and LinuxONE, the **s2i-dotnet** cluster task is unsupported.
- Implicit parameter mapping incorrectly passes parameters from the top-level **Pipeline** or **PipelineRun** definitions to the **taskRef** tasks. Mapping should only occur from a top-level resource to tasks with in-line **taskSpec** specifications. This issue only affects clusters where this feature was enabled by setting the **enable-api-fields** field to **alpha** in the **pipeline** section of the **TektonConfig** custom resource definition.

3.1.4.5. Fixed issues

- Before this update, the metrics for pipeline runs in the Developer view of the web console were incomplete and outdated. With this update, the issue has been fixed so that the metrics are correct.
- Before this update, if a pipeline had two parallel tasks that failed and one of them had **retries=2**, the final tasks never ran, and the pipeline timed out and failed to run. For example, the **pipelines-operator-subscription** task failed intermittently with the following error message: **Unable to connect to the server: EOF**. With this update, the issue has been fixed so that the final tasks always run.
- Before this update, if a pipeline run stopped because a task run failed, other task runs might not complete their retries. As a result, no **finally** tasks were scheduled, which caused the pipeline to hang. This update resolves the issue. **TaskRuns** and **Run** objects can retry when a pipeline run has stopped, even by graceful stopping, so that pipeline runs can complete.
- This update changes how resource requirements are calculated when one or more **LimitRange** objects are present in the namespace where a **TaskRun** object exists. The scheduler now considers **step** containers and excludes all other app containers, such as sidecar containers, when factoring requests from **LimitRange** objects.
- Before this update, under specific conditions, the flag package might incorrectly parse a subcommand immediately following a double dash flag terminator, **--**. In that case, it ran the entrypoint subcommand rather than the actual command. This update fixes this flag-parsing issue so that the entrypoint runs the correct command.

- Before this update, the controller might generate multiple panics if pulling an image failed, or its pull status was incomplete. This update fixes the issue by checking the **step.ImageID** value rather than the **status.TaskSpec** value.
- Before this update, canceling a pipeline run that contained an unscheduled custom task produced a **PipelineRunCouldntCancel** error. This update fixes the issue. You can cancel a pipeline run that contains an unscheduled custom task without producing that error.
- Before this update, if the **<NAME>** in **\$params["<NAME>"]** or **\$params['<NAME>']** contained a dot character (**.**), any part of the name to the right of the dot was not extracted. For example, from **\$params["org.ipsum.lorem"]**, only **org** was extracted. This update fixes the issue so that **\$params** fetches the complete value. For example, **\$params["org.ipsum.lorem"]** and **\$params['org.ipsum.lorem']** are valid and the entire value of **<NAME>**, **org.ipsum.lorem**, is extracted.

It also throws an error if **<NAME>** is not enclosed in single or double quotes. For example, **\$params.org.ipsum.lorem** is not valid and generates a validation error.

- With this update, **Trigger** resources support custom interceptors and ensure that the port of the custom interceptor service is the same as the port in the **ClusterInterceptor** definition file.
- Before this update, the **tkn version** command for Tekton Chains and Operator components did not work correctly. This update fixes the issue so that the command works correctly and returns version information for those components.
- Before this update, if you ran a **tkn pr delete --ignore-running** command and a pipeline run did not have a **status.condition** value, the **tkn** CLI tool produced a null-pointer error (NPE). This update fixes the issue so that the CLI tool now generates an error and correctly ignores pipeline runs that are still running.
- Before this update, if you used the **tkn pr delete --keep <value>** or **tkn tr delete --keep <value>** commands, and the number of pipeline runs or task runs was less than the value, the command did not return an error as expected. This update fixes the issue so that the command correctly returns an error under those conditions.
- Before this update, if you used the **tkn pr delete** or **tkn tr delete** commands with the **-p** or **-t** flags together with the **--ignore-running** flag, the commands incorrectly deleted running or pending resources. This update fixes the issue so that these commands correctly ignore running or pending resources.
- With this update, you can configure Tekton Chains by using the **TektonChain** custom resource. This feature enables your configuration to persist after upgrading, unlike the **chains-config** config map, which gets overwritten during upgrades.
- With this update, **ClusterTask** resources no longer run as root by default, except for the **buildah** and **s2i** cluster tasks.
- Before this update, tasks on Red Hat OpenShift Pipelines 1.7.1 failed when using **init** as a first argument followed by two or more arguments. With this update, the flags are parsed correctly, and the task runs are successful.
- Before this update, installation of the Red Hat OpenShift Pipelines Operator on OpenShift Container Platform 4.9 and 4.10 failed due to an invalid role binding, with the following error message:

```
error updating rolebinding openshift-operators-prometheus-k8s-read-binding:
RoleBinding.rbac.authorization.k8s.io
```

```
"openshift-operators-prometheus-k8s-read-binding" is invalid:
roleRef: Invalid value: rbac.RoleRef{APIGroup:"rbac.authorization.k8s.io", Kind:"Role",
Name:"openshift-operator-read"}: cannot change roleRef
```

This update fixes the issue so that the failure no longer occurs.

- Previously, upgrading the Red Hat OpenShift Pipelines Operator caused the **pipeline** service account to be recreated, which meant that the secrets linked to the service account were lost. This update fixes the issue. During upgrades, the Operator no longer recreates the **pipeline** service account. As a result, secrets attached to the **pipeline** service account persist after upgrades, and the resources (tasks and pipelines) continue to work correctly.
- With this update, Pipelines as Code pods run on infrastructure nodes if infrastructure node settings are configured in the **TektonConfig** custom resource (CR).
- Previously, with the resource pruner, each namespace Operator created a command that ran in a separate container. This design consumed too many resources in clusters with a high number of namespaces. For example, to run a single command, a cluster with 1000 namespaces produced 1000 containers in a pod. This update fixes the issue. It passes the namespace-based configuration to the job so that all the commands run in one container in a loop.
- In Tekton Chains, you must define a secret called **signing-secrets** to hold the key used for signing tasks and images. However, before this update, updating the Red Hat OpenShift Pipelines Operator reset or overwrote this secret, and the key was lost. This update fixes the issue. Now, if the secret is configured after installing Tekton Chains through the Operator, the secret persists, and it is not overwritten by upgrades.
- Before this update, all S2I build tasks failed with an error similar to the following message:

```
Error: error writing "0 0 4294967295\n" to /proc/22/uid_map: write /proc/22/uid_map:
operation not permitted
time="2022-03-04T09:47:57Z" level=error msg="error writing \"0 0 4294967295\\n\" to
/proc/22/uid_map: write /proc/22/uid_map: operation not permitted"
time="2022-03-04T09:47:57Z" level=error msg="(unable to determine exit status)"
```

With this update, the **pipelines-scc** security context constraint (SCC) is compatible with the **SETFCAP** capability necessary for **Buildah** and **S2I** cluster tasks. As a result, the **Buildah** and **S2I** build tasks can run successfully.

To successfully run the **Buildah** cluster task and **S2I** build tasks for applications written in various languages and frameworks, add the following snippet for appropriate **steps** objects such as **build** and **push**:

```
securityContext:
  capabilities:
    add: ["SETFCAP"]
```

- Before this update, installing the Red Hat OpenShift Pipelines Operator took longer than expected. This update optimizes some settings to speed up the installation process.
- With this update, Buildah and S2I cluster tasks have fewer steps than in previous versions. Some steps have been combined into a single step so that they work better with **ResourceQuota** and **LimitRange** objects and do not require more resources than necessary.
- This update upgrades the Buildah, **tkn** CLI tool, and **skopeo** CLI tool versions in cluster tasks.

- Before this update, the Operator failed when creating RBAC resources if any namespace was in a **Terminating** state. With this update, the Operator ignores namespaces in a **Terminating** state and creates the RBAC resources.
- Before this update, pods for the prune cronjobs were not scheduled on infrastructure nodes, as expected. Instead, they were scheduled on worker nodes or not scheduled at all. With this update, these types of pods can now be scheduled on infrastructure nodes if configured in the **TektonConfig** custom resource (CR).

3.1.4.6. Release notes for Red Hat OpenShift Pipelines General Availability 1.8.1

With this update, Red Hat OpenShift Pipelines General Availability (GA) 1.8.1 is available on OpenShift Container Platform 4.10 and 4.11.

3.1.4.6.1. Known issues

- By default, the containers have restricted permissions for enhanced security. The restricted permissions apply to all controller pods in the Red Hat OpenShift Pipelines Operator, and to some cluster tasks. Due to restricted permissions, the **git-clone** cluster task fails under certain configurations.
Workaround: None. You can track the issue [SRVKP-2634](#).
- When installer sets are in a failed state, the status of the **TektonConfig** custom resource is incorrectly displayed as **True** instead of **False**.

Example: Failed installer sets

```
$ oc get tektoninstallerset
NAME                                READY  REASON
addon-clustertasks-nx5xz            False  Error
addon-communityclustertasks-cfb2p   True
addon-consolecli-ftrb8              True
addon-openshift-67dj2              True
addon-pac-cf7pz                     True
addon-pipelines-fvllm              True
addon-triggers-b2wtt                True
addon-versioned-clustertasks-1-8-hqhnw  False  Error
pipeline-w75ww                     True
postpipeline-lrs22                  True
prepipeline-ldlhw                   True
rhosp-rbac-4dmgb                    True
trigger-hfg64                       True
validating-mutating-webhook-28rf7   True
```

Example: Incorrect TektonConfig status

```
$ oc get tektonconfig config
NAME  VERSION  READY  REASON
config 1.8.1    True
```

3.1.4.6.2. Fixed issues

- Before this update, the pruner deleted task runs of running pipelines and displayed the following warning: **some tasks were indicated completed without ancestors being done**. With this update, the pruner retains the task runs that are part of running pipelines.
- Before this update, **pipeline-1.8** was the default channel for installing the Red Hat OpenShift Pipelines Operator 1.8.x. With this update, **latest** is the default channel.
- Before this update, the Pipelines as Code controller pods did not have access to certificates exposed by the user. With this update, Pipelines as Code can now access routes and Git repositories guarded by a self-signed or a custom certificate.
- Before this update, the task failed with RBAC errors after upgrading from Red Hat OpenShift Pipelines 1.7.2 to 1.8.0. With this update, the tasks run successfully without any RBAC errors.
- Before this update, using the **tkn** CLI tool, you could not remove task runs and pipeline runs that contained a **result** object whose type was **array**. With this update, you can use the **tkn** CLI tool to remove task runs and pipeline runs that contain a **result** object whose type is **array**.
- Before this update, if a pipeline specification contained a task with an **ENV_VARS** parameter of **array** type, the pipeline run failed with the following error: **invalid input params for task func-buildpacks: param types don't match the user-specified type: [ENV_VARS]**. With this update, pipeline runs with such pipeline and task specifications do not fail.
- Before this update, cluster administrators could not provide a **config.json** file to the **Buildah** cluster task for accessing a container registry. With this update, cluster administrators can provide the **Buildah** cluster task with a **config.json** file by using the **dockerconfig** workspace.

3.1.4.7. Release notes for Red Hat OpenShift Pipelines General Availability 1.8.2

With this update, Red Hat OpenShift Pipelines General Availability (GA) 1.8.2 is available on OpenShift Container Platform 4.10 and 4.11.

3.1.4.7.1. Fixed issues

- Before this update, the **git-clone** task failed when cloning a repository using SSH keys. With this update, the role of the non-root user in the **git-init** task is removed, and the SSH program looks in the **\$HOME/.ssh/** directory for the correct keys.

3.1.5. Release notes for Red Hat OpenShift Pipelines General Availability 1.7

With this update, Red Hat OpenShift Pipelines General Availability (GA) 1.7 is available on OpenShift Container Platform 4.9, 4.10, and 4.11.

3.1.5.1. New features

In addition to the fixes and stability improvements, the following sections highlight what is new in Red Hat OpenShift Pipelines 1.7.

3.1.5.1.1. Pipelines

- With this update, **pipelines-<version>** is the default channel to install the Red Hat OpenShift Pipelines Operator. For example, the default channel to install the Pipelines Operator version **1.7** is **pipelines-1.7**. Cluster administrators can also use the **latest** channel to install the most recent stable version of the Operator.

**NOTE**

The **preview** and **stable** channels will be deprecated and removed in a future release.

- When you run a command in a user namespace, your container runs as **root** (user id **0**) but has user privileges on the host. With this update, to run pods in the user namespace, you must pass the annotations that **CRI-O** expects.
 - To add these annotations for all users, run the **oc edit clustertask buildah** command and edit the **buildah** cluster task.
 - To add the annotations to a specific namespace, export the cluster task as a task to that namespace.
- With this update, the **when** expressions in a **Task** object are scoped to guard the tasks by default. To continue guarding the **Task** object and its dependent tasks, set the **scope-when-expressions-to-task** flag to **true**.

**NOTE**

The **scope-when-expressions-to-task** flag is deprecated and will be removed in a future release. As a best practice for Pipelines, use **when** expressions scoped to the guarded **Task** only.

- With this update, you can use variable substitution in the **subPath** field of a workspace within a task.
- With this update, you can reference parameters and results by using a bracket notation with single or double quotes. Prior to this update, you could only use the dot notation. For example, the following are now equivalent:
 - **\$(param.myparam)**, **\$(param['myparam'])**, and **\$(param["myparam"])**.
You can use single or double quotes to enclose parameter names that contain problematic characters, such as **."**. For example, **\$(param['my.param'])** and **\$(param["my.param"])**.
- With this update, you can include the **onError** parameter of a step in the task definition without enabling the **enable-api-fields** flag.

3.1.5.1.2. Triggers

- With this update, the **feature-flag-triggers** config map has a new field **labels-exclusion-pattern**. You can set the value of this field to a regular expression (regex) pattern. The controller filters out labels that match the regex pattern from propagating from the event listener to the resources created for the event listener.
- With this update, the **TriggerGroups** field is added to the **EventListener** specification. Using this field, you can specify a set of interceptors to run before selecting and running a group of triggers. To enable this feature, in the **TektonConfig** custom resource definition, in the **pipeline** section, you must set the **enable-api-fields** field to **alpha**.
- With this update, **Trigger** resources support custom runs defined by a **TriggerTemplate** template.
- With this update, Triggers support emitting Kubernetes events from an **EventListener** pod.

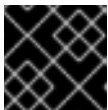
- With this update, count metrics are available for the following objects: **ClusterInteceptor**, **EventListener**, **TriggerTemplate**, **ClusterTriggerBinding**, and **TriggerBinding**.
- This update adds the **ServicePort** specification to Kubernetes resource. You can use this specification to modify which port exposes the event listener service. The default port is **8080**.
- With this update, you can use the **targetURI** field in the **EventListener** specification to send cloud events during trigger processing. To enable this feature, in the **TektonConfig** custom resource definition, in the **pipeline** section, you must set the **enable-api-fields** field to **alpha**.
- With this update, the **tekton-triggers-eventlistener-roles** object now has a **patch** verb, in addition to the **create** verb that already exists.
- With this update, the **securityContext.runAsUser** parameter is removed from event listener deployment.

3.1.5.1.3. CLI

- With this update, the **tkn [pipeline | pipelinerun] export** command exports a pipeline or pipeline run as a YAML file. For example:
 - Export a pipeline named **test_pipeline** in the **openshift-pipelines** namespace:


```
$ tkn pipeline export test_pipeline -n openshift-pipelines
```
 - Export a pipeline run named **test_pipeline_run** in the **openshift-pipelines** namespace:

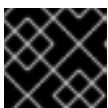

```
$ tkn pipelinerun export test_pipeline_run -n openshift-pipelines
```
- With this update, the **--grace** option is added to the **tkn pipelinerun cancel**. Use the **--grace** option to terminate a pipeline run gracefully instead of forcing the termination. To enable this feature, in the **TektonConfig** custom resource definition, in the **pipeline** section, you must set the **enable-api-fields** field to **alpha**.
- This update adds the Operator and Chains versions to the output of the **tkn version** command.



IMPORTANT

Tekton Chains is a Technology Preview feature.

- With this update, the **tkn pipelinerun describe** command displays all canceled task runs, when you cancel a pipeline run. Before this fix, only one task run was displayed.
- With this update, you can skip supplying the asking specifications for optional workspace when you run the **tkn [t | p | ct] start** command skips with the **--skip-optional-workspace** flag. You can also skip it when running in interactive mode.
- With this update, you can use the **tkn chains** command to manage Tekton Chains. You can also use the **--chains-namespace** option to specify the namespace where you want to install Tekton Chains.

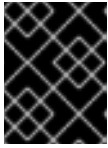


IMPORTANT

Tekton Chains is a Technology Preview feature.

3.1.5.1.4. Operator

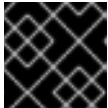
- With this update, you can use the Red Hat OpenShift Pipelines Operator to install and deploy Tekton Hub and Tekton Chains.



IMPORTANT

Tekton Chains and deployment of Tekton Hub on a cluster are Technology Preview features.

- With this update, you can find and use Pipelines as Code (PAC) as an add-on option.



IMPORTANT

Pipelines as Code is a Technology Preview feature.

- With this update, you can now disable the installation of community cluster tasks by setting the **communityClusterTasks** parameter to **false**. For example:

```
...
spec:
  profile: all
  targetNamespace: openshift-pipelines
  addon:
    params:
      - name: clusterTasks
        value: "true"
      - name: pipelineTemplates
        value: "true"
      - name: communityClusterTasks
        value: "false"
  ...
```

- With this update, you can disable the integration of Tekton Hub with the **Developer** perspective by setting the **enable-devconsole-integration** flag in the **TektonConfig** custom resource to **false**. For example:

```
...
hub:
  params:
    - name: enable-devconsole-integration
      value: "true"
  ...
```

- With this update, the **operator-config.yaml** config map enables the output of the **tkn version** command to display of the Operator version.
- With this update, the version of the **argocd-task-sync-and-wait** tasks is modified to **v0.2**.
- With this update to the **TektonConfig** CRD, the **oc get tektonconfig** command displays the OPerator version.
- With this update, service monitor is added to the Triggers metrics.

3.1.5.1.5. Hub



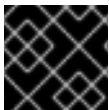
IMPORTANT

Deploying Tekton Hub on a cluster is a Technology Preview feature.

Tekton Hub helps you discover, search, and share reusable tasks and pipelines for your CI/CD workflows. A public instance of Tekton Hub is available at hub.tekton.dev.

Starting with Red Hat OpenShift Pipelines 1.7, cluster administrators can also install and deploy a custom instance of Tekton Hub on enterprise clusters. You can curate a catalog with reusable tasks and pipelines specific to your organization.

3.1.5.1.6. Chains



IMPORTANT

Tekton Chains is a Technology Preview feature.

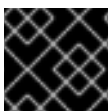
Tekton Chains is a Kubernetes Custom Resource Definition (CRD) controller. You can use it to manage the supply chain security of the tasks and pipelines created using Red Hat OpenShift Pipelines.

By default, Tekton Chains monitors the task runs in your OpenShift Container Platform cluster. Chains takes snapshots of completed task runs, converts them to one or more standard payload formats, and signs and stores all artifacts.

Tekton Chains supports the following features:

- You can sign task runs, task run results, and OCI registry images with cryptographic key types and services such as **cosign**.
- You can use attestation formats such as **in-toto**.
- You can securely store signatures and signed artifacts using OCI repository as a storage backend.

3.1.5.1.7. Pipelines as Code (PAC)



IMPORTANT

Pipelines as Code is a Technology Preview feature.

With Pipelines as Code, cluster administrators and users with the required privileges can define pipeline templates as part of source code Git repositories. When triggered by a source code push or a pull request for the configured Git repository, the feature runs the pipeline and reports status.

Pipelines as Code supports the following features:

- Pull request status. When iterating over a pull request, the status and control of the pull request is exercised on the platform hosting the Git repository.
- GitHub checks the API to set the status of a pipeline run, including rechecks.
- GitHub pull request and commit events.

- Pull request actions in comments, such as **/retest**.
- Git events filtering, and a separate pipeline for each event.
- Automatic task resolution in Pipelines for local tasks, Tekton Hub, and remote URLs.
- Use of GitHub blobs and objects API for retrieving configurations.
- Access Control List (ACL) over a GitHub organization, or using a Prow-style **OWNER** file.
- The **tkn-pac** plugin for the **tkn** CLI tool, which you can use to manage Pipelines as Code repositories and bootstrapping.
- Support for GitHub Application, GitHub Webhook, Bitbucket Server, and Bitbucket Cloud.

3.1.5.2. Deprecated features

- Breaking change: This update removes the **disable-working-directory-overwrite** and **disable-home-env-overwrite** fields from the **TektonConfig** custom resource (CR). As a result, the **TektonConfig** CR no longer automatically sets the **\$HOME** environment variable and **workingDir** parameter. You can still set the **\$HOME** environment variable and **workingDir** parameter by using the **env** and **workingDir** fields in the **Task** custom resource definition (CRD).
- The **Conditions** custom resource definition (CRD) type is deprecated and planned to be removed in a future release. Instead, use the recommended **When** expression.
- Breaking change: The **Triggers** resource validates the templates and generates an error if you do not specify the **EventListener** and **TriggerBinding** values.

3.1.5.3. Known issues

- When you run Maven and Jib-Maven cluster tasks, the default container image is supported only on Intel (x86) architecture. Therefore, tasks will fail on ARM, IBM Power Systems (ppc64le), IBM Z, and LinuxONE (s390x) clusters. As a workaround, you can specify a custom image by setting the **MAVEN_IMAGE** parameter value to **maven:3.6.3-adoptopenjdk-11**.

TIP

Before you install tasks that are based on the Tekton Catalog on ARM, IBM Power Systems (ppc64le), IBM Z, and LinuxONE (s390x) using **tkn hub**, verify if the task can be executed on these platforms. To check if **ppc64le** and **s390x** are listed in the "Platforms" section of the task information, you can run the following command: **tkn hub info task <name>**

- On IBM Power Systems, IBM Z, and LinuxONE, the **s2i-dotnet** cluster task is unsupported.
- You cannot use the **nodejs:14-ubi8-minimal** image stream because doing so generates the following errors:

```
STEP 7: RUN /usr/libexec/s2i/assemble
/bin/sh: /usr/libexec/s2i/assemble: No such file or directory
subprocess exited with status 127
subprocess exited with status 127
error building at STEP "RUN /usr/libexec/s2i/assemble": exit status 127
time="2021-11-04T13:05:26Z" level=error msg="exit status 127"
```

- Implicit parameter mapping incorrectly passes parameters from the top-level **Pipeline** or **PipelineRun** definitions to the **taskRef** tasks. Mapping should only occur from a top-level resource to tasks with in-line **taskSpec** specifications. This issue only affects clusters where this feature was enabled by setting the **enable-api-fields** field to **alpha** in the **pipeline** section of the **TektonConfig** custom resource definition.

3.1.5.4. Fixed issues

- With this update, if metadata such as **labels** and **annotations** are present in both **Pipeline** and **PipelineRun** object definitions, the values in the **PipelineRun** type takes precedence. You can observe similar behavior for **Task** and **TaskRun** objects.
- With this update, if the **timeouts.tasks** field or the **timeouts.finally** field is set to **0**, then the **timeouts.pipeline** is also set to **0**.
- With this update, the **-x** set flag is removed from scripts that do not use a shebang. The fix reduces potential data leak from script execution.
- With this update, any backslash character present in the usernames in Git credentials is escaped with an additional backslash in the **.gitconfig** file.
- With this update, the **finalizer** property of the **EventListener** object is not necessary for cleaning up logging and config maps.
- With this update, the default HTTP client associated with the event listener server is removed, and a custom HTTP client added. As a result, the timeouts have improved.
- With this update, the Triggers cluster role now works with owner references.
- With this update, the race condition in the event listener does not happen when multiple interceptors return extensions.
- With this update, the **tkn pr delete** command does not delete the pipeline runs with the **ignore-running** flag.
- With this update, the Operator pods do not continue restarting when you modify any add-on parameters.
- With this update, the **tkn serve** CLI pod is scheduled on infra nodes, if not configured in the subscription and config custom resources.
- With this update, cluster tasks with specified versions are not deleted during upgrade.

3.1.5.5. Release notes for Red Hat OpenShift Pipelines General Availability 1.7.1

With this update, Red Hat OpenShift Pipelines General Availability (GA) 1.7.1 is available on OpenShift Container Platform 4.9, 4.10, and 4.11.

3.1.5.5.1. Fixed issues

- Before this update, upgrading the Red Hat OpenShift Pipelines Operator deleted the data in the database associated with Tekton Hub and installed a new database. With this update, an Operator upgrade preserves the data.

- Before this update, only cluster administrators could access pipeline metrics in the OpenShift Container Platform console. With this update, users with other cluster roles also can access the pipeline metrics.
- Before this update, pipeline runs failed for pipelines containing tasks that emit large termination messages. The pipeline runs failed because the total size of termination messages of all containers in a pod cannot exceed 12 KB. With this update, the **place-tools** and **step-init** initialization containers that uses the same image are merged to reduce the number of containers running in each task's pod. The solution reduces the chance of failed pipeline runs by minimizing the number of containers running in a task's pod. However, it does not remove the limitation of the maximum allowed size of a termination message.
- Before this update, attempts to access resource URLs directly from the Tekton Hub web console resulted in an Nginx **404** error. With this update, the Tekton Hub web console image is fixed to allow accessing resource URLs directly from the Tekton Hub web console.
- Before this update, for each namespace the resource pruner job created a separate container to prune resources. With this update, the resource pruner job runs commands for all namespaces as a loop in one container.

3.1.5.6. Release notes for Red Hat OpenShift Pipelines General Availability 1.7.2

With this update, Red Hat OpenShift Pipelines General Availability (GA) 1.7.2 is available on OpenShift Container Platform 4.9, 4.10, and the upcoming version.

3.1.5.6.1. Known issues

- The **chains-config** config map for Tekton Chains in the **openshift-pipelines** namespace is automatically reset to default after upgrading the Red Hat OpenShift Pipelines Operator. Currently, there is no workaround for this issue.

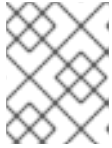
3.1.5.6.2. Fixed issues

- Before this update, tasks on Pipelines 1.7.1 failed on using **init** as the first argument, followed by two or more arguments. With this update, the flags are parsed correctly and the task runs are successful.
- Before this update, installation of the Red Hat OpenShift Pipelines Operator on OpenShift Container Platform 4.9 and 4.10 failed due to invalid role binding, with the following error message:

```
error updating rolebinding openshift-operators-prometheus-k8s-read-binding:
RoleBinding.rbac.authorization.k8s.io "openshift-operators-prometheus-k8s-read-binding" is
invalid: roleRef: Invalid value: rbac.RoleRef{APIGroup:"rbac.authorization.k8s.io",
Kind:"Role", Name:"openshift-operator-read"}: cannot change roleRef
```

With this update, the Red Hat OpenShift Pipelines Operator installs with distinct role binding namespaces to avoid conflict with installation of other Operators.

- Before this update, upgrading the Operator triggered a reset of the **signing-secrets** secret key for Tekton Chains to its default value. With this update, the custom secret key persists after you upgrade the Operator.



NOTE

Upgrading to Red Hat OpenShift Pipelines 1.7.2 resets the key. However, when you upgrade to future releases, the key is expected to persist.

- Before this update, all S2I build tasks failed with an error similar to the following message:

```
Error: error writing "0 0 4294967295\n" to /proc/22/uid_map: write /proc/22/uid_map:
operation not permitted
time="2022-03-04T09:47:57Z" level=error msg="error writing \"0 0 4294967295\\n\" to
/proc/22/uid_map: write /proc/22/uid_map: operation not permitted"
time="2022-03-04T09:47:57Z" level=error msg="(unable to determine exit status)"
```

With this update, the **pipelines-scc** security context constraint (SCC) is compatible with the **SETFCAP** capability necessary for **Buildah** and **S2I** cluster tasks. As a result, the **Buildah** and **S2I** build tasks can run successfully.

To successfully run the **Buildah** cluster task and **S2I** build tasks for applications written in various languages and frameworks, add the following snippet for appropriate **steps** objects such as **build** and **push**:

```
securityContext:
  capabilities:
    add: ["SETFCAP"]
```

3.1.5.7. Release notes for Red Hat OpenShift Pipelines General Availability 1.7.3

With this update, Red Hat OpenShift Pipelines General Availability (GA) 1.7.3 is available on OpenShift Container Platform 4.9, 4.10, and 4.11.

3.1.5.7.1. Fixed issues

- Before this update, the Operator failed when creating RBAC resources if any namespace was in a **Terminating** state. With this update, the Operator ignores namespaces in a **Terminating** state and creates the RBAC resources.
- Previously, upgrading the Red Hat OpenShift Pipelines Operator caused the **pipeline** service account to be recreated, which meant that the secrets linked to the service account were lost. This update fixes the issue. During upgrades, the Operator no longer recreates the **pipeline** service account. As a result, secrets attached to the **pipeline** service account persist after upgrades, and the resources (tasks and pipelines) continue to work correctly.

3.1.6. Release notes for Red Hat OpenShift Pipelines General Availability 1.6

With this update, Red Hat OpenShift Pipelines General Availability (GA) 1.6 is available on OpenShift Container Platform 4.9.

3.1.6.1. New features

In addition to the fixes and stability improvements, the following sections highlight what is new in Red Hat OpenShift Pipelines 1.6.

- With this update, you can configure a pipeline or task **start** command to return a YAML or JSON-formatted string by using the **--output <string>**, where **<string>** is **yaml** or **json**.

Otherwise, without the **--output** option, the **start** command returns a human-friendly message that is hard for other programs to parse. Returning a YAML or JSON-formatted string is useful for continuous integration (CI) environments. For example, after a resource is created, you can use **yq** or **jq** to parse the YAML or JSON-formatted message about the resource and wait until that resource is terminated without using the **showlog** option.

- With this update, you can authenticate to a registry using the **auth.json** authentication file of Podman. For example, you can use **tkn bundle push** to push to a remote registry using Podman instead of Docker CLI.
- With this update, if you use the **tkn [taskrun | pipelinerun] delete --all** command, you can preserve runs that are younger than a specified number of minutes by using the new **--keep-since <minutes>** option. For example, to keep runs that are less than five minutes old, you enter **tkn [taskrun | pipelinerun] delete -all --keep-since 5**.
- With this update, when you delete task runs or pipeline runs, you can use the **--parent-resource** and **--keep-since** options together. For example, the **tkn pipelinerun delete --pipeline pipelinename --keep-since 5** command preserves pipeline runs whose parent resource is named **pipelinename** and whose age is five minutes or less. The **tkn tr delete -t <taskname> --keep-since 5** and **tkn tr delete --clustertask <taskname> --keep-since 5** commands work similarly for task runs.
- This update adds support for the triggers resources to work with **v1beta1** resources.
- This update adds an **ignore-running** option to the **tkn pipelinerun delete** and **tkn taskrun delete** commands.
- This update adds a **create** subcommand to the **tkn task** and **tkn clustertask** commands.
- With this update, when you use the **tkn pipelinerun delete --all** command, you can use the new **--label <string>** option to filter the pipeline runs by label. Optionally, you can use the **--label** option with **=** and **==** as equality operators, or **!=** as an inequality operator. For example, the **tkn pipelinerun delete --all --label asdf** and **tkn pipelinerun delete --all --label==asdf** commands both delete all the pipeline runs that have the **asdf** label.
- With this update, you can fetch the version of installed Tekton components from the config map or, if the config map is not present, from the deployment controller.
- With this update, triggers support the **feature-flags** and **config-defaults** config map to configure feature flags and to set default values respectively.
- This update adds a new metric, **eventlistener_event_count**, that you can use to count events received by the **EventListener** resource.
- This update adds **v1beta1** Go API types. With this update, triggers now support the **v1beta1** API version.
With the current release, the **v1alpha1** features are now deprecated and will be removed in a future release. Begin using the **v1beta1** features instead.
- In the current release, auto-pruning of resources is enabled by default. In addition, you can configure auto-pruning of task run and pipeline run for each namespace separately, by using the following new annotations:
 - **operator.tekton.dev/prune.schedule**: If the value of this annotation is different from the value specified at the **TektonConfig** custom resource definition, a new cron job in that namespace is created.

- **operator.tekton.dev/prune.skip**: When set to **true**, the namespace for which it is configured will not be pruned.
- **operator.tekton.dev/prune.resources**: This annotation accepts a comma-separated list of resources. To prune a single resource such as a pipeline run, set this annotation to **"pipelinerun"**. To prune multiple resources, such as task run and pipeline run, set this annotation to **"taskrun, pipelinerun"**.
- **operator.tekton.dev/prune.keep**: Use this annotation to retain a resource without pruning.
- **operator.tekton.dev/prune.keep-since**: Use this annotation to retain resources based on their age. The value for this annotation must be equal to the age of the resource in minutes. For example, to retain resources which were created not more than five days ago, set **keep-since** to **7200**.



NOTE

The **keep** and **keep-since** annotations are mutually exclusive. For any resource, you must configure only one of them.

- **operator.tekton.dev/prune.strategy**: Set the value of this annotation to either **keep** or **keep-since**.
- Administrators can disable the creation of the **pipeline** service account for the entire cluster, and prevent privilege escalation by misusing the associated SCC, which is very similar to **anyuid**.
- You can now configure feature flags and components by using the **TektonConfig** custom resource (CR) and the CRs for individual components, such as **TektonPipeline** and **TektonTriggers**. This level of granularity helps customize and test alpha features such as the Tekton OCI bundle for individual components.
- You can now configure optional **Timeouts** field for the **PipelineRun** resource. For example, you can configure timeouts separately for a pipeline run, each task run, and the **finally** tasks.
- The pods generated by the **TaskRun** resource now sets the **activeDeadlineSeconds** field of the pods. This enables OpenShift to consider them as terminating, and allows you to use specifically scoped **ResourceQuota** object for the pods.
- You can use configmaps to eliminate metrics tags or labels type on a task run, pipeline run, task, and pipeline. In addition, you can configure different types of metrics for measuring duration, such as a histogram, gauge, or last value.
- You can define requests and limits on a pod coherently, as Tekton now fully supports the **LimitRange** object by considering the **Min**, **Max**, **Default**, and **DefaultRequest** fields.
- The following alpha features are introduced:
 - A pipeline run can now stop after running the **finally** tasks, rather than the previous behavior of stopping the execution of all task run directly. This update adds the following **spec.status** values:
 - **StoppedRunFinally** will stop the currently running tasks after they are completed, and then run the **finally** tasks.
 - **CancelledRunFinally** will immediately cancel the running tasks, and then run the **finally** tasks.

- **Cancelled** will retain the previous behavior provided by the **PipelineRunCancelled** status.



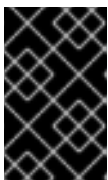
NOTE

The **Cancelled** status replaces the deprecated **PipelineRunCancelled** status, which will be removed in the **v1** version.

- You can now use the **oc debug** command to put a task run into debug mode, which pauses the execution and allows you to inspect specific steps in a pod.
- When you set the **onError** field of a step to **continue**, the exit code for the step is recorded and passed on to subsequent steps. However, the task run does not fail and the execution of the rest of the steps in the task continues. To retain the existing behavior, you can set the value of the **onError** field to **stopAndFail**.
- Tasks can now accept more parameters than are actually used. When the alpha feature flag is enabled, the parameters can implicitly propagate to inlined specs. For example, an inlined task can access parameters of its parent pipeline run, without explicitly defining each parameter for the task.
- If you enable the flag for the alpha features, the conditions under **When** expressions will only apply to the task with which it is directly associated, and not the dependents of the task. To apply the **When** expressions to the associated task and its dependents, you must associate the expression with each dependent task separately. Note that, going forward, this will be the default behavior of the **When** expressions in any new API versions of Tekton. The existing default behavior will be deprecated in favor of this update.
- The current release enables you to configure node selection by specifying the **nodeSelector** and **tolerations** values in the **TektonConfig** custom resource (CR). The Operator adds these values to all the deployments that it creates.
 - To configure node selection for the Operator's controller and webhook deployment, you edit the **config.nodeSelector** and **config.tolerations** fields in the specification for the **Subscription** CR, after installing the Operator.
 - To deploy the rest of the control plane pods of OpenShift Pipelines on an infrastructure node, update the **TektonConfig** CR with the **nodeSelector** and **tolerations** fields. The modifications are then applied to all the pods created by Operator.

3.1.6.2. Deprecated features

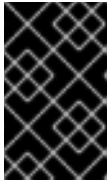
- In CLI 0.21.0, support for all **v1alpha1** resources for **clustertask**, **task**, **taskrun**, **pipeline**, and **pipelinerun** commands are deprecated. These resources are now deprecated and will be removed in a future release.
- In Tekton Triggers v0.16.0, the redundant **status** label is removed from the metrics for the **EventListener** resource.



IMPORTANT

Breaking change: The **status** label has been removed from the **eventlistener_http_duration_seconds_*** metric. Remove queries that are based on the **status** label.

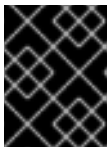
- With the current release, the **v1alpha1** features are now deprecated and will be removed in a future release. With this update, you can begin using the **v1beta1** Go API types instead. Triggers now supports the **v1beta1** API version.
- With the current release, the **EventListener** resource sends a response before the triggers finish processing.



IMPORTANT

Breaking change: With this change, the **EventListener** resource stops responding with a **201 Created** status code when it creates resources. Instead, it responds with a **202 Accepted** response code.

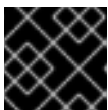
- The current release removes the **podTemplate** field from the **EventListener** resource.



IMPORTANT

Breaking change: The **podTemplate** field, which was deprecated as part of [#1100](#), has been removed.

- The current release removes the deprecated **replicas** field from the specification for the **EventListener** resource.



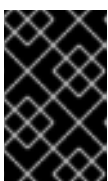
IMPORTANT

Breaking change: The deprecated **replicas** field has been removed.

- In Red Hat OpenShift Pipelines 1.6, the values of **HOME="/tekton/home"** and **workingDir="/workspace"** are removed from the specification of the **Step** objects. Instead, Red Hat OpenShift Pipelines sets **HOME** and **workingDir** to the values defined by the containers running the **Step** objects. You can override these values in the specification of your **Step** objects.

To use the older behavior, you can change the **disable-working-directory-overwrite** and **disable-home-env-overwrite** fields in the **TektonConfig** CR to **false**:

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    disable-working-directory-overwrite: false
    disable-home-env-overwrite: false
...
```



IMPORTANT

The **disable-working-directory-overwrite** and **disable-home-env-overwrite** fields in the **TektonConfig** CR are now deprecated and will be removed in a future release.

3.1.6.3. Known issues

- When you run Maven and Jib-Maven cluster tasks, the default container image is supported only on Intel (x86) architecture. Therefore, tasks will fail on IBM Power Systems (ppc64le), IBM Z, and LinuxONE (s390x) clusters. As a workaround, you can specify a custom image by setting the **MAVEN_IMAGE** parameter value to **maven:3.6.3-adoptopenjdk-11**.
- On IBM Power Systems, IBM Z, and LinuxONE, the **s2i-dotnet** cluster task is unsupported.
- Before you install tasks based on the Tekton Catalog on IBM Power Systems (ppc64le), IBM Z, and LinuxONE (s390x) using **tkn hub**, verify if the task can be executed on these platforms. To check if **ppc64le** and **s390x** are listed in the "Platforms" section of the task information, you can run the following command: **tkn hub info task <name>**
- You cannot use the **nodejs:14-ubi8-minimal** image stream because doing so generates the following errors:

```
STEP 7: RUN /usr/libexec/s2i/assemble
/bin/sh: /usr/libexec/s2i/assemble: No such file or directory
subprocess exited with status 127
subprocess exited with status 127
error building at STEP "RUN /usr/libexec/s2i/assemble": exit status 127
time="2021-11-04T13:05:26Z" level=error msg="exit status 127"
```

3.1.6.4. Fixed issues

- The **tkn hub** command is now supported on IBM Power Systems, IBM Z, and LinuxONE.
- Before this update, the terminal was not available after the user ran a **tkn** command, and the pipeline run was done, even if **retries** were specified. Specifying a timeout in the task run or pipeline run had no effect. This update fixes the issue so that the terminal is available after running the command.
- Before this update, running **tkn pipelinerun delete --all** would delete all resources. This update prevents the resources in the running state from getting deleted.
- Before this update, using the **tkn version --component=<component>** command did not return the component version. This update fixes the issue so that this command returns the component version.
- Before this update, when you used the **tkn pr logs** command, it displayed the pipelines output logs in the wrong task order. This update resolves the issue so that logs of completed **PipelineRuns** are listed in the appropriate **TaskRun** execution order.
- Before this update, editing the specification of a running pipeline might prevent the pipeline run from stopping when it was complete. This update fixes the issue by fetching the definition only once and then using the specification stored in the status for verification. This change reduces the probability of a race condition when a **PipelineRun** or a **TaskRun** refers to a **Pipeline** or **Task** that changes while it is running.
- **When** expression values can now have array parameter references, such as: **values: [\$(params.arrayParam[*])]**.

3.1.6.5. Release notes for Red Hat OpenShift Pipelines General Availability 1.6.1

3.1.6.5.1. Known issues

- After upgrading to Red Hat OpenShift Pipelines 1.6.1 from an older version, Pipelines might enter an inconsistent state where you are unable to perform any operations (create/delete/apply) on Tekton resources (tasks and pipelines). For example, while deleting a resource, you might encounter the following error:

```
Error from server (InternalError): Internal error occurred: failed calling webhook
"validation.webhook.pipeline.tekton.dev": Post "https://tekton-pipelines-webhook.openshift-
pipelines.svc:443/resource-validation?timeout=10s": service "tekton-pipelines-webhook" not
found.
```

3.1.6.5.2. Fixed issues

- The **SSL_CERT_DIR** environment variable (**/tekton-custom-certs**) set by Red Hat OpenShift Pipelines will not override the following default system directories with certificate files:
 - **/etc/pki/tls/certs**
 - **/etc/ssl/certs**
 - **/system/etc/security/cacerts**
- The Horizontal Pod Autoscaler can manage the replica count of deployments controlled by the Red Hat OpenShift Pipelines Operator. From this release onward, if the count is changed by an end user or an on-cluster agent, the Red Hat OpenShift Pipelines Operator will not reset the replica count of deployments managed by it. However, the replicas will be reset when you upgrade the Red Hat OpenShift Pipelines Operator.
- The pod serving the **tkn** CLI will now be scheduled on nodes, based on the node selector and toleration limits specified in the **TektonConfig** custom resource.

3.1.6.6. Release notes for Red Hat OpenShift Pipelines General Availability 1.6.2

3.1.6.6.1. Known issues

- When you create a new project, the creation of the **pipeline** service account is delayed, and removal of existing cluster tasks and pipeline templates takes more than 10 minutes.

3.1.6.6.2. Fixed issues

- Before this update, multiple instances of Tekton installer sets were created for a pipeline after upgrading to Red Hat OpenShift Pipelines 1.6.1 from an older version. With this update, the Operator ensures that only one instance of each type of **TektonInstallerSet** exists after an upgrade.
- Before this update, all the reconcilers in the Operator used the component version to decide resource recreation during an upgrade to Red Hat OpenShift Pipelines 1.6.1 from an older version. As a result, those resources were not recreated whose component versions did not change in the upgrade. With this update, the Operator uses the Operator version instead of the component version to decide resource recreation during an upgrade.
- Before this update, the pipelines webhook service was missing in the cluster after an upgrade. This was due to an upgrade deadlock on the config maps. With this update, a mechanism is added to disable webhook validation if the config maps are absent in the cluster. As a result, the

pipelines webhook service persists in the cluster after an upgrade.

- Before this update, cron jobs for auto-pruning got recreated after any configuration change to the namespace. With this update, cron jobs for auto-pruning get recreated only if there is a relevant annotation change in the namespace.
- The upstream version of Tekton Pipelines is revised to **v0.28.3**, which has the following fixes:
 - Fix **PipelineRun** or **TaskRun** objects to allow label or annotation propagation.
 - For implicit params:
 - Do not apply the **PipelineSpec** parameters to the **TaskRefs** object.
 - Disable implicit param behavior for the **Pipeline** objects.

3.1.6.7. Release notes for Red Hat OpenShift Pipelines General Availability 1.6.3

3.1.6.7.1. Fixed issues

- Before this update, the Red Hat OpenShift Pipelines Operator installed pod security policies from components such as Pipelines and Triggers. However, the pod security policies shipped as part of the components were deprecated in an earlier release. With this update, the Operator stops installing pod security policies from components. As a result, the following upgrade paths are affected:
 - Upgrading from Pipelines 1.6.1 or 1.6.2 to Pipelines 1.6.3 deletes the pod security policies, including those from the Pipelines and Triggers components.
 - Upgrading from Pipelines 1.5.x to 1.6.3 retains the pod security policies installed from components. As a cluster administrator, you can delete them manually.



NOTE

When you upgrade to future releases, the Red Hat OpenShift Pipelines Operator will automatically delete all obsolete pod security policies.

- Before this update, only cluster administrators could access pipeline metrics in the OpenShift Container Platform console. With this update, users with other cluster roles also can access the pipeline metrics.
- Before this update, role-based access control (RBAC) issues with the Pipelines Operator caused problems upgrading or installing components. This update improves the reliability and consistency of installing various Red Hat OpenShift Pipelines components.
- Before this update, setting the **clusterTasks** and **pipelineTemplates** fields to **false** in the **TektonConfig** CR slowed the removal of cluster tasks and pipeline templates. This update improves the speed of lifecycle management of Tekton resources such as cluster tasks and pipeline templates.

3.1.6.8. Release notes for Red Hat OpenShift Pipelines General Availability 1.6.4

3.1.6.8.1. Known issues

- After upgrading from Red Hat OpenShift Pipelines 1.5.2 to 1.6.4, accessing the event listener routes returns a **503** error.

Workaround: Modify the target port in the YAML file for the event listener's route.

1. Extract the route name for the relevant namespace.

```
$ oc get route -n <namespace>
```

2. Edit the route to modify the value of the **targetPort** field.

```
$ oc edit route -n <namespace> <el-route_name>
```

Example: Existing event listener route

```
...
spec:
  host: el-event-listener-q8c3w5-test-upgrade1.apps.ve49aws.aws.ospqa.com
  port:
    targetPort: 8000
  to:
    kind: Service
    name: el-event-listener-q8c3w5
    weight: 100
  wildcardPolicy: None
...
```

Example: Modified event listener route

```
...
spec:
  host: el-event-listener-q8c3w5-test-upgrade1.apps.ve49aws.aws.ospqa.com
  port:
    targetPort: http-listener
  to:
    kind: Service
    name: el-event-listener-q8c3w5
    weight: 100
  wildcardPolicy: None
...
```

3.1.6.8.2. Fixed issues

- Before this update, the Operator failed when creating RBAC resources if any namespace was in a **Terminating** state. With this update, the Operator ignores namespaces in a **Terminating** state and creates the RBAC resources.
- Before this update, the task runs failed or restarted due to absence of annotation specifying the release version of the associated Tekton controller. With this update, the inclusion of the appropriate annotations are automated, and the tasks run without failure or restarts.

3.1.7. Release notes for Red Hat OpenShift Pipelines General Availability 1.5

Red Hat OpenShift Pipelines General Availability (GA) 1.5 is now available on OpenShift Container Platform 4.9.

3.1.7.1. Compatibility and support matrix

Some features in this release are currently in [Technology Preview](#). These experimental features are not intended for production use.

In the table, features are marked with the following statuses:

TP	Technology Preview
GA	General Availability

Note the following scope of support on the Red Hat Customer Portal for these features:

Table 3.2. Compatibility and support matrix

Feature	Version	Support Status
Pipelines	0.24	GA
CLI	0.19	GA
Catalog	0.24	GA
Triggers	0.14	TP
Pipeline resources	-	TP

For questions and feedback, you can send an email to the product team at pipelines-interest@redhat.com.

3.1.7.2. New features

In addition to the fixes and stability improvements, the following sections highlight what is new in Red Hat OpenShift Pipelines 1.5.

- Pipeline run and task runs will be automatically pruned by a cron job in the target namespace. The cron job uses the **IMAGE_JOB_PRUNER_TKN** environment variable to get the value of **tkn image**. With this enhancement, the following fields are introduced to the **TektonConfig** custom resource:

```
...
pruner:
  resources:
    - pipelinerun
    - taskrun
  schedule: "*/5 * * * *" # cron schedule
  keep: 2 # delete all keeping n
...
```


- In OpenShift Container Platform, you can customize the installation of the Tekton Add-ons component by modifying the values of the new parameters **clusterTasks** and **pipelineTemplates** in the **TektonConfig** custom resource:

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  profile: all
  targetNamespace: openshift-pipelines
  addon:
    params:
      - name: clusterTasks
        value: "true"
      - name: pipelineTemplates
        value: "true"
  ...
```

The customization is allowed if you create the add-on using **TektonConfig**, or directly by using Tekton Add-ons. However, if the parameters are not passed, the controller adds parameters with default values.



NOTE

- If add-on is created using the **TektonConfig** custom resource, and you change the parameter values later in the **Addon** custom resource, then the values in the **TektonConfig** custom resource overwrites the changes.
- You can set the value of the **pipelineTemplates** parameter to **true** only when the value of the **clusterTasks** parameter is **true**.

- The **enableMetrics** parameter is added to the **TektonConfig** custom resource. You can use it to disable the service monitor, which is part of Tekton Pipelines for OpenShift Container Platform.

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  profile: all
  targetNamespace: openshift-pipelines
  pipeline:
    params:
      - name: enableMetrics
        value: "true"
  ...
```

- Eventlistener OpenCensus metrics, which captures metrics at process level, is added.
- Triggers now has label selector; you can configure triggers for an event listener using labels.

- The **ClusterInterceptor** custom resource definition for registering interceptors is added, which allows you to register new **Interceptor** types that you can plug in. In addition, the following relevant changes are made:
 - In the trigger specifications, you can configure interceptors using a new API that includes a **ref** field to refer to a cluster interceptor. In addition, you can use the **params** field to add parameters that pass on to the interceptors for processing.
 - The bundled interceptors CEL, GitHub, GitLab, and BitBucket, have been migrated. They are implemented using the new **ClusterInterceptor** custom resource definition.
 - Core interceptors are migrated to the new format, and any new triggers created using the old syntax automatically switch to the new **ref** or **params** based syntax.
- To disable prefixing the name of the task or step while displaying logs, use the **--prefix** option for **log** commands.
- To display the version of a specific component, use the new **--component** flag in the **tkn version** command.
- The **tkn hub check-upgrade** command is added, and other commands are revised to be based on the pipeline version. In addition, catalog names are displayed in the **search** command output.
- Support for optional workspaces are added to the **start** command.
- If the plugins are not present in the **plugins** directory, they are searched in the current path.
- The **tkn start [task | clustertask | pipeline]** command starts interactively and ask for the **params** value, even when you specify the default parameters are specified. To stop the interactive prompts, pass the **--use-param-defaults** flag at the time of invoking the command. For example:

```
$ tkn pipeline start build-and-deploy \
  -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.8/01_pipeline/03_persistent_volume_claim.yaml \
  -p deployment-name=pipelines-vote-api \
  -p git-url=https://github.com/openshift/pipelines-vote-api.git \
  -p IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/pipelines-
vote-api \
  --use-param-defaults
```

- The **version** field is added in the **tkn task describe** command.
- The option to automatically select resources such as **TriggerTemplate**, or **TriggerBinding**, or **ClusterTriggerBinding**, or **EventListener**, is added in the **describe** command, if only one is present.
- In the **tkn pr describe** command, a section for skipped tasks is added.
- Support for the **tkn clustertask logs** is added.
- The YAML merge and variable from **config.yaml** is removed. In addition, the **release.yaml** file can now be more easily consumed by tools such as **kustomize** and **ytt**.
- The support for resource names to contain the dot character (".") is added.

- The **hostAliases** array in the **PodTemplate** specification is added to the pod-level override of hostname resolution. It is achieved by modifying the **/etc/hosts** file.
- A variable **\$(tasks.status)** is introduced to access the aggregate execution status of tasks.
- An entry-point binary build for Windows is added.

3.1.7.3. Deprecated features

- In the **when** expressions, support for fields written in PascalCase is removed. The **when** expressions only support fields written in lowercase.



NOTE

If you had applied a pipeline with **when** expressions in Tekton Pipelines **v0.16** (Operator **v1.2.x**), you have to reapply it.

- When you upgrade the Red Hat OpenShift Pipelines Operator to **v1.5**, the **openshift-client** and the **openshift-client-v-1-5-0** cluster tasks have the **SCRIPT** parameter. However, the **ARGS** parameter and the **git** resource are removed from the specification of the **openshift-client** cluster task. This is a breaking change, and only those cluster tasks that do not have a specific version in the **name** field of the **ClusterTask** resource upgrade seamlessly. To prevent the pipeline runs from breaking, use the **SCRIPT** parameter after the upgrade because it moves the values previously specified in the **ARGS** parameter into the **SCRIPT** parameter of the cluster task. For example:

```
...
- name: deploy
  params:
    - name: SCRIPT
      value: oc rollout status <deployment-name>
  runAfter:
    - build
  taskRef:
    kind: ClusterTask
    name: openshift-client
...
```

- When you upgrade from Red Hat OpenShift Pipelines Operator **v1.4** to **v1.5**, the profile names in which the **TektonConfig** custom resource is installed now change.

Table 3.3. Profiles for TektonConfig custom resource

Profiles in Pipelines 1.5	Corresponding profile in Pipelines 1.4	Installed Tekton components
All (<i>default profile</i>)	All (<i>default profile</i>)	Pipelines, Triggers, Add-ons
Basic	Default	Pipelines, Triggers
Lite	Basic	Pipelines

**NOTE**

If you used **profile: all** in the **config** instance of the **TektonConfig** custom resource, no change is necessary in the resource specification.

However, if the installed Operator is either in the Default or the Basic profile before the upgrade, you must edit the **config** instance of the **TektonConfig** custom resource after the upgrade. For example, if the configuration was **profile: basic** before the upgrade, ensure that it is **profile: lite** after upgrading to Pipelines 1.5.

- The **disable-home-env-overwrite** and **disable-working-dir-overwrite** fields are now deprecated and will be removed in a future release. For this release, the default value of these flags is set to **true** for backward compatibility.

**NOTE**

In the next release (Red Hat OpenShift Pipelines 1.6), the **HOME** environment variable will not be automatically set to **/tekton/home**, and the default working directory will not be set to **/workspace** for task runs. These defaults collide with any value set by image Dockerfile of the step.

- The **ServiceType** and **podTemplate** fields are removed from the **EventListener** spec.
- The controller service account no longer requests cluster-wide permission to list and watch namespaces.
- The status of the **EventListener** resource has a new condition called **Ready**.

**NOTE**

In the future, the other status conditions for the **EventListener** resource will be deprecated in favor of the **Ready** status condition.

- The **eventListener** and **namespace** fields in the **EventListener** response are deprecated. Use the **eventListenerUID** field instead.
- The **replicas** field is deprecated from the **EventListener** spec. Instead, the **spec.replicas** field is moved to **spec.resources.kubernetesResource.replicas** in the **KubernetesResource** spec.

**NOTE**

The **replicas** field will be removed in a future release.

- The old method of configuring the core interceptors is deprecated. However, it continues to work until it is removed in a future release. Instead, interceptors in a **Trigger** resource are now configured using a new **ref** and **params** based syntax. The resulting default webhook automatically switch the usages of the old syntax to the new syntax for new triggers.
- Use **rbac.authorization.k8s.io/v1** instead of the deprecated **rbac.authorization.k8s.io/v1beta1** for the **ClusterRoleBinding** resource.

- In cluster roles, the cluster-wide write access to resources such as **serviceaccounts**, **secrets**, **configmaps**, and **limitranges** are removed. In addition, cluster-wide access to resources such as **deployments**, **statefulsets**, and **deployment/finalizers** are removed.
- The **image** custom resource definition in the **caching.internal.knative.dev** group is not used by Tekton anymore, and is excluded in this release.

3.1.7.4. Known issues

- The **git-cli** cluster task is built off the **alpine/git** base image, which expects **/root** as the user's home directory. However, this is not explicitly set in the **git-cli** cluster task. In Tekton, the default home directory is overwritten with **/tekton/home** for every step of a task, unless otherwise specified. This overwriting of the **\$HOME** environment variable of the base image causes the **git-cli** cluster task to fail.

This issue is expected to be fixed in the upcoming releases. For Red Hat OpenShift Pipelines 1.5 and earlier versions, you can *use any one of the following workarounds* to avoid the failure of the **git-cli** cluster task:

- Set the **\$HOME** environment variable in the steps, so that it is not overwritten.
 1. [OPTIONAL] If you installed Red Hat OpenShift Pipelines using the Operator, then clone the **git-cli** cluster task into a separate task. This approach ensures that the Operator does not overwrite the changes made to the cluster task.
 2. Execute the **oc edit clustertasks git-cli** command.
 3. Add the expected **HOME** environment variable to the YAML of the step:

```
...
steps:
- name: git
  env:
  - name: HOME
    value: /root
  image: $(params.BASE_IMAGE)
  workingDir: $(workspaces.source.path)
...
```



WARNING

For Red Hat OpenShift Pipelines installed by the Operator, if you do not clone the **git-cli** cluster task into a separate task before changing the **HOME** environment variable, then the changes are overwritten during Operator reconciliation.

- Disable overwriting the **HOME** environment variable in the **feature-flags** config map.
 1. Execute the **oc edit -n openshift-pipelines configmap feature-flags** command.
 2. Set the value of the **disable-home-env-overwrite** flag to **true**.

**WARNING**

- If you installed Red Hat OpenShift Pipelines using the Operator, then the changes are overwritten during Operator reconciliation.
- Modifying the default value of the **disable-home-env-overwrite** flag can break other tasks and cluster tasks, as it changes the default behavior for all tasks.

- Use a different service account for the **git-cli** cluster task, as the overwriting of the **HOME** environment variable happens when the default service account for pipelines is used.
 1. Create a new service account.
 2. Link your Git secret to the service account you just created.
 3. Use the service account while executing a task or a pipeline.
- On IBM Power Systems, IBM Z, and LinuxONE, the **s2i-dotnet** cluster task and the **tkn hub** command are unsupported.
- When you run Maven and Jib-Maven cluster tasks, the default container image is supported only on Intel (x86) architecture. Therefore, tasks will fail on IBM Power Systems (ppc64le), IBM Z, and LinuxONE (s390x) clusters. As a workaround, you can specify a custom image by setting the **MAVEN_IMAGE** parameter value to **maven:3.6.3-adoptopenjdk-11**.

3.1.7.5. Fixed issues

- The **when** expressions in **dag** tasks are not allowed to specify the context variable accessing the execution status (**\$(tasks.<pipelineTask>.status)**) of any other task.
- Use Owner UIDs instead of Owner names, as it helps avoid race conditions created by deleting a **volumeClaimTemplate** PVC, in situations where a **PipelineRun** resource is quickly deleted and then recreated.
- A new Dockerfile is added for **pullrequest-init** for **build-base** image triggered by non-root users.
- When a pipeline or task is executed with the **-f** option and the **param** in its definition does not have a **type** defined, a validation error is generated instead of the pipeline or task run failing silently.
- For the **tkn start [task | pipeline | clustertask]** commands, the description of the **--workspace** flag is now consistent.
- While parsing the parameters, if an empty array is encountered, the corresponding interactive help is displayed as an empty string now.

3.1.8. Release notes for Red Hat OpenShift Pipelines General Availability 1.4

Red Hat OpenShift Pipelines General Availability (GA) 1.4 is now available on OpenShift Container Platform 4.7.



NOTE

In addition to the stable and preview Operator channels, the Red Hat OpenShift Pipelines Operator 1.4.0 comes with the ocp-4.6, ocp-4.5, and ocp-4.4 deprecated channels. These deprecated channels and support for them will be removed in the following release of Red Hat OpenShift Pipelines.

3.1.8.1. Compatibility and support matrix

Some features in this release are currently in [Technology Preview](#). These experimental features are not intended for production use.

In the table, features are marked with the following statuses:

TP	Technology Preview
GA	General Availability

Note the following scope of support on the Red Hat Customer Portal for these features:

Table 3.4. Compatibility and support matrix

Feature	Version	Support Status
Pipelines	0.22	GA
CLI	0.17	GA
Catalog	0.22	GA
Triggers	0.12	TP
Pipeline resources	-	TP

For questions and feedback, you can send an email to the product team at pipelines-interest@redhat.com.

3.1.8.2. New features

In addition to the fixes and stability improvements, the following sections highlight what is new in Red Hat OpenShift Pipelines 1.4.

- The custom tasks have the following enhancements:
 - Pipeline results can now refer to results produced by custom tasks.
 - Custom tasks can now use workspaces, service accounts, and pod templates to build more complex custom tasks.

- The **finally** task has the following enhancements:
 - The **when** expressions are supported in **finally** tasks, which provides efficient guarded execution and improved reusability of tasks.
 - A **finally** task can be configured to consume the results of any task within the same pipeline.

**NOTE**

Support for **when** expressions and **finally** tasks are unavailable in the OpenShift Container Platform 4.7 web console.

- Support for multiple secrets of the type **dockercfg** or **dockerconfigjson** is added for authentication at runtime.
- Functionality to support sparse-checkout with the **git-clone** task is added. This enables you to clone only a subset of the repository as your local copy, and helps you to restrict the size of the cloned repositories.
- You can create pipeline runs in a pending state without actually starting them. In clusters that are under heavy load, this allows Operators to have control over the start time of the pipeline runs.
- Ensure that you set the **SYSTEM_NAMESPACE** environment variable manually for the controller; this was previously set by default.
- A non-root user is now added to the build-base image of pipelines so that **git-init** can clone repositories as a non-root user.
- Support to validate dependencies between resolved resources before a pipeline run starts is added. All result variables in the pipeline must be valid, and optional workspaces from a pipeline can only be passed to tasks expecting it for the pipeline to start running.
- The controller and webhook runs as a non-root group, and their superfluous capabilities have been removed to make them more secure.
- You can use the **tkn pr logs** command to see the log streams for retried task runs.
- You can use the **--clustertask** option in the **tkn tr delete** command to delete all the task runs associated with a particular cluster task.
- Support for using Knative service with the **EventListener** resource is added by introducing a new **customResource** field.
- An error message is displayed when an event payload does not use the JSON format.
- The source control interceptors such as GitLab, BitBucket, and GitHub, now use the new **InterceptorRequest** or **InterceptorResponse** type interface.
- A new CEL function **marshalJSON** is implemented so that you can encode a JSON object or an array to a string.
- An HTTP handler for serving the CEL and the source control core interceptors is added. It packages four core interceptors into a single HTTP server that is deployed in the **tekton-pipelines** namespace. The **EventListener** object forwards events over the HTTP server to the interceptor. Each interceptor is available at a different path. For example, the CEL interceptor is available on the **/cel** path.

- The **pipelines-scc** Security Context Constraint (SCC) is used with the default **pipeline** service account for pipelines. This new service account is similar to **anyuid**, but with a minor difference as defined in the YAML for SCC of OpenShift Container Platform 4.7:

```
fsGroup:
  type: MustRunAs
```

3.1.8.3. Deprecated features

- The **build-gcs** sub-type in the pipeline resource storage, and the **gcs-fetcher** image, are not supported.
- In the **taskRun** field of cluster tasks, the label **tekton.dev/task** is removed.
- For webhooks, the value **v1beta1** corresponding to the field **admissionReviewVersions** is removed.
- The **creds-init** helper image for building and deploying is removed.
- In the triggers spec and binding, the deprecated field **template.name** is removed in favor of **template.ref**. You should update all **eventListener** definitions to use the **ref** field.



NOTE

Upgrade from Pipelines 1.3.x and earlier versions to Pipelines 1.4.0 breaks event listeners because of the unavailability of the **template.name** field. For such cases, use Pipelines 1.4.1 to avail the restored **template.name** field.

- For **EventListener** custom resources/objects, the fields **PodTemplate** and **ServiceType** are deprecated in favor of **Resource**.
- The deprecated spec style embedded bindings is removed.
- The **spec** field is removed from the **triggerSpecBinding**.
- The event ID representation is changed from a five-character random string to a UUID.

3.1.8.4. Known issues

- In the **Developer** perspective, the pipeline metrics and triggers features are available only on OpenShift Container Platform 4.7.6 or later versions.
- On IBM Power Systems, IBM Z, and LinuxONE, the **tkn hub** command is not supported.
- When you run Maven and Jib Maven cluster tasks on an IBM Power Systems (ppc64le), IBM Z, and LinuxONE (s390x) clusters, set the **MAVEN_IMAGE** parameter value to **maven:3.6.3-adoptopenjdk-11**.
- Triggers throw error resulting from bad handling of the JSON format, if you have the following configuration in the trigger binding:

```
params:
  - name: github_json
    value: $(body)
```

To resolve the issue:

- If you are using triggers v0.11.0 and above, use the **marshalJSON** CEL function, which takes a JSON object or array and returns the JSON encoding of that object or array as a string.
- If you are using older triggers version, add the following annotation in the trigger template:

```

annotations:
  triggers.tekton.dev/old-escape-quotes: "true"

```

- When upgrading from Pipelines 1.3.x to 1.4.x, you must recreate the routes.

3.1.8.5. Fixed issues

- Previously, the **tekton.dev/task** label was removed from the task runs of cluster tasks, and the **tekton.dev/clusterTask** label was introduced. The problems resulting from that change is resolved by fixing the **clustertask describe** and **delete** commands. In addition, the **lastrun** function for tasks is modified, to fix the issue of the **tekton.dev/task** label being applied to the task runs of both tasks and cluster tasks in older versions of pipelines.
- When doing an interactive **tkn pipeline start pipelinename**, a **PipelineResource** is created interactively. The **tkn p start** command prints the resource status if the resource status is not **nil**.
- Previously, the **tekton.dev/task=name** label was removed from the task runs created from cluster tasks. This fix modifies the **tkn clustertask start** command with the **--last** flag to check for the **tekton.dev/task=name** label in the created task runs.
- When a task uses an inline task specification, the corresponding task run now gets embedded in the pipeline when you run the **tkn pipeline describe** command, and the task name is returned as embedded.
- The **tkn version** command is fixed to display the version of the installed Tekton CLI tool, without a configured **kubeConfiguration namespace** or access to a cluster.
- If an argument is unexpected or more than one arguments are used, the **tkn completion** command gives an error.
- Previously, pipeline runs with the **finally** tasks nested in a pipeline specification would lose those **finally** tasks, when converted to the **v1alpha1** version and restored back to the **v1beta1** version. This error occurring during conversion is fixed to avoid potential data loss. Pipeline runs with the **finally** tasks nested in a pipeline specification is now serialized and stored on the alpha version, only to be deserialized later.
- Previously, there was an error in the pod generation when a service account had the **secrets** field as **{}**. The task runs failed with **CouldntGetTask** because the GET request with an empty secret name returned an error, indicating that the resource name may not be empty. This issue is fixed by avoiding an empty secret name in the **kubeclient** GET request.
- Pipelines with the **v1beta1** API versions can now be requested along with the **v1alpha1** version, without losing the **finally** tasks. Applying the returned **v1alpha1** version will store the resource as **v1beta1**, with the **finally** section restored to its original state.
- Previously, an unset **selfLink** field in the controller caused an error in the Kubernetes v1.20 clusters. As a temporary fix, the **CloudEvent** source field is set to a value that matches the current source URI, without the value of the auto-populated **selfLink** field.

- Previously, a secret name with dots such as **gcr.io** led to a task run creation failure. This happened because of the secret name being used internally as part of a volume mount name. The volume mount name conforms to the RFC1123 DNS label and disallows dots as part of the name. This issue is fixed by replacing the dot with a dash that results in a readable name.
- Context variables are now validated in the **finally** tasks.
- Previously, when the task run reconciler was passed a task run that did not have a previous status update containing the name of the pod it created, the task run reconciler listed the pods associated with the task run. The task run reconciler used the labels of the task run, which were propagated to the pod, to find the pod. Changing these labels while the task run was running, caused the code to not find the existing pod. As a result, duplicate pods were created. This issue is fixed by changing the task run reconciler to only use the **tekton.dev/taskRun** Tekton-controlled label when finding the pod.
- Previously, when a pipeline accepted an optional workspace and passed it to a pipeline task, the pipeline run reconciler stopped with an error if the workspace was not provided, even if a missing workspace binding is a valid state for an optional workspace. This issue is fixed by ensuring that the pipeline run reconciler does not fail to create a task run, even if an optional workspace is not provided.
- The sorted order of step statuses matches the order of step containers.
- Previously, the task run status was set to **unknown** when a pod encountered the **CreateContainerConfigError** reason, which meant that the task and the pipeline ran until the pod timed out. This issue is fixed by setting the task run status to **false**, so that the task is set as failed when the pod encounters the **CreateContainerConfigError** reason.
- Previously, pipeline results were resolved on the first reconciliation, after a pipeline run was completed. This could fail the resolution resulting in the **Succeeded** condition of the pipeline run being overwritten. As a result, the final status information was lost, potentially confusing any services watching the pipeline run conditions. This issue is fixed by moving the resolution of pipeline results to the end of a reconciliation, when the pipeline run is put into a **Succeeded** or **True** condition.
- Execution status variable is now validated. This avoids validating task results while validating context variables to access execution status.
- Previously, a pipeline result that contained an invalid variable would be added to the pipeline run with the literal expression of the variable intact. Therefore, it was difficult to assess whether the results were populated correctly. This issue is fixed by filtering out the pipeline run results that reference failed task runs. Now, a pipeline result that contains an invalid variable will not be emitted by the pipeline run at all.
- The **tkn eventlistener describe** command is fixed to avoid crashing without a template. It also displays the details about trigger references.
- Upgrades from Pipelines 1.3.x and earlier versions to Pipelines 1.4.0 breaks event listeners because of the unavailability of **template.name**. In Pipelines 1.4.1, the **template.name** has been restored to avoid breaking event listeners in triggers.
- In Pipelines 1.4.1, the **ConsoleQuickStart** custom resource has been updated to align with OpenShift Container Platform 4.7 capabilities and behavior.

3.1.9. Release notes for Red Hat OpenShift Pipelines Technology Preview 1.3

3.1.9.1. New features

Red Hat OpenShift Pipelines Technology Preview (TP) 1.3 is now available on OpenShift Container Platform 4.7. Red Hat OpenShift Pipelines TP 1.3 is updated to support:

- Tekton Pipelines 0.19.0
- Tekton **tkn** CLI 0.15.0
- Tekton Triggers 0.10.2
- cluster tasks based on Tekton Catalog 0.19.0
- IBM Power Systems on OpenShift Container Platform 4.7
- IBM Z and LinuxONE on OpenShift Container Platform 4.7

In addition to the fixes and stability improvements, the following sections highlight what is new in Red Hat OpenShift Pipelines 1.3.

3.1.9.1.1. Pipelines

- Tasks that build images, such as S2I and Buildah tasks, now emit a URL of the image built that includes the image SHA.
- Conditions in pipeline tasks that reference custom tasks are disallowed because the **Condition** custom resource definition (CRD) has been deprecated.
- Variable expansion is now added in the **Task** CRD for the following fields: **spec.steps[].imagePullPolicy** and **spec.sidecar[].imagePullPolicy**.
- You can disable the built-in credential mechanism in Tekton by setting the **disable-creds-init** feature-flag to **true**.
- Resolved when expressions are now listed in the **Skipped Tasks** and the **Task Runs** sections in the **Status** field of the **PipelineRun** configuration.
- The **git init** command can now clone recursive submodules.
- A **Task** CR author can now specify a timeout for a step in the **Task** spec.
- You can now base the entry point image on the **distroless/static:nonroot** image and give it a mode to copy itself to the destination, without relying on the **cp** command being present in the base image.
- You can now use the configuration flag **require-git-ssh-secret-known-hosts** to disallow omitting known hosts in the Git SSH secret. When the flag value is set to **true**, you must include the **known_host** field in the Git SSH secret. The default value for the flag is **false**.
- The concept of optional workspaces is now introduced. A task or pipeline might declare a workspace optional and conditionally change their behavior based on its presence. A task run or pipeline run might also omit that workspace, thereby modifying the task or pipeline behavior. The default task run workspaces are not added in place of an omitted optional workspace.
- Credentials initialization in Tekton now detects an SSH credential that is used with a non-SSH URL, and vice versa in Git pipeline resources, and logs a warning in the step containers.

- The task run controller emits a warning event if the affinity specified by the pod template is overwritten by the affinity assistant.
- The task run reconciler now records metrics for cloud events that are emitted once a task run is completed. This includes retries.

3.1.9.1.2. Pipelines CLI

- Support for **--no-headers flag** is now added to the following commands: **tkn condition list**, **tkn triggerbinding list**, **tkn eventlistener list**, **tkn clustertask list**, **tkn clustertriggerbinding list**.
- When used together, the **--last** or **--use** options override the **--prefix-name** and **--timeout** options.
- The **tkn eventlistener logs** command is now added to view the **EventListener** logs.
- The **tekton hub** commands are now integrated into the **tkn** CLI.
- The **--nocolour** option is now changed to **--no-color**.
- The **--all-namespaces** flag is added to the following commands: **tkn triggertemplate list**, **tkn condition list**, **tkn triggerbinding list**, **tkn eventlistener list**.

3.1.9.1.3. Triggers

- You can now specify your resource information in the **EventListener** template.
- It is now mandatory for **EventListener** service accounts to have the **list** and **watch** verbs, in addition to the **get** verb for all the triggers resources. This enables you to use **Listers** to fetch data from **EventListener**, **Trigger**, **TriggerBinding**, **TriggerTemplate**, and **ClusterTriggerBinding** resources. You can use this feature to create a **Sink** object rather than specifying multiple informers, and directly make calls to the API server.
- A new **Interceptor** interface is added to support immutable input event bodies. Interceptors can now add data or fields to a new **extensions** field, and cannot modify the input bodies making them immutable. The CEL interceptor uses this new **Interceptor** interface.
- A **namespaceSelector** field is added to the **EventListener** resource. Use it to specify the namespaces from where the **EventListener** resource can fetch the **Trigger** object for processing events. To use the **namespaceSelector** field, the service account for the **EventListener** resource must have a cluster role.
- The triggers **EventListener** resource now supports end-to-end secure connection to the **eventlistener** pod.
- The escaping parameters behavior in the **TriggerTemplates** resource by replacing **"** with **\"** is now removed.
- A new **resources** field, supporting Kubernetes resources, is introduced as part of the **EventListener** spec.
- A new functionality for the CEL interceptor, with support for upper and lower-casing of ASCII strings, is added.
- You can embed **TriggerBinding** resources by using the **name** and **value** fields in a trigger, or an event listener.

- The **PodSecurityPolicy** configuration is updated to run in restricted environments. It ensures that containers must run as non-root. In addition, the role-based access control for using the pod security policy is moved from cluster-scoped to namespace-scoped. This ensures that the triggers cannot use other pod security policies that are unrelated to a namespace.
- Support for embedded trigger templates is now added. You can either use the **name** field to refer to an embedded template or embed the template inside the **spec** field.

3.1.9.2. Deprecated features

- Pipeline templates that use **PipelineResources** CRDs are now deprecated and will be removed in a future release.
- The **template.name** field is deprecated in favor of the **template.ref** field and will be removed in a future release.
- The **-c** shorthand for the **--check** command has been removed. In addition, global **tkn** flags are added to the **version** command.

3.1.9.3. Known issues

- CEL overlays add fields to a new top-level **extensions** function, instead of modifying the incoming event body. **TriggerBinding** resources can access values within this new **extensions** function using the **\$(extensions.<key>)** syntax. Update your binding to use the **\$(extensions.<key>)** syntax instead of the **\$(body.<overlay-key>)** syntax.
- The escaping parameters behavior by replacing " with \" is now removed. If you need to retain the old escaping parameters behavior add the **tekton.dev/old-escape-quotes: true** annotation to your **TriggerTemplate** specification.
- You can embed **TriggerBinding** resources by using the **name** and **value** fields inside a trigger or an event listener. However, you cannot specify both **name** and **ref** fields for a single binding. Use the **ref** field to refer to a **TriggerBinding** resource and the **name** field for embedded bindings.
- An interceptor cannot attempt to reference a **secret** outside the namespace of an **EventListener** resource. You must include secrets in the namespace of the ``EventListener`` resource.
- In Triggers 0.9.0 and later, if a body or header based **TriggerBinding** parameter is missing or malformed in an event payload, the default values are used instead of displaying an error.
- Tasks and pipelines created with **WhenExpression** objects using Tekton Pipelines 0.16.x must be reapplied to fix their JSON annotations.
- When a pipeline accepts an optional workspace and gives it to a task, the pipeline run stalls if the workspace is not provided.
- To use the Buildah cluster task in a disconnected environment, ensure that the Dockerfile uses an internal image stream as the base image, and then use it in the same manner as any S2I cluster task.

3.1.9.4. Fixed issues

- Extensions added by a CEL Interceptor are passed on to webhook interceptors by adding the **Extensions** field within the event body.

- The activity timeout for log readers is now configurable using the **LogOptions** field. However, the default behavior of timeout in 10 seconds is retained.
- The **log** command ignores the **--follow** flag when a task run or pipeline run is complete, and reads available logs instead of live logs.
- References to the following Tekton resources: **EventListener**, **TriggerBinding**, **ClusterTriggerBinding**, **Condition**, and **TriggerTemplate** are now standardized and made consistent across all user-facing messages in **tkn** commands.
- Previously, if you started a canceled task run or pipeline run with the **--use-taskrun <canceled-task-run-name>**, **--use-pipelinerun <canceled-pipeline-run-name>** or **--last** flags, the new run would be canceled. This bug is now fixed.
- The **tkn pr desc** command is now enhanced to ensure that it does not fail in case of pipeline runs with conditions.
- When you delete a task run using the **tkn tr delete** command with the **--task** option, and a cluster task exists with the same name, the task runs for the cluster task also get deleted. As a workaround, filter the task runs by using the **TaskRefKind** field.
- The **tkn triggertemplate describe** command would display only part of the **apiVersion** value in the output. For example, only **triggers.tekton.dev** was displayed instead of **triggers.tekton.dev/v1alpha1**. This bug is now fixed.
- The webhook, under certain conditions, would fail to acquire a lease and not function correctly. This bug is now fixed.
- Pipelines with when expressions created in v0.16.3 can now be run in v0.17.1 and later. After an upgrade, you do not need to reapply pipeline definitions created in previous versions because both the uppercase and lowercase first letters for the annotations are now supported.
- By default, the **leader-election-ha** field is now enabled for high availability. When the **disable-ha** controller flag is set to **true**, it disables high availability support.
- Issues with duplicate cloud events are now fixed. Cloud events are now sent only when a condition changes the state, reason, or message.
- When a service account name is missing from a **PipelineRun** or **TaskRun** spec, the controller uses the service account name from the **config-defaults** config map. If the service account name is also missing in the **config-defaults** config map, the controller now sets it to **default** in the spec.
- Validation for compatibility with the affinity assistant is now supported when the same persistent volume claim is used for multiple workspaces, but with different subpaths.

3.1.10. Release notes for Red Hat OpenShift Pipelines Technology Preview 1.2

3.1.10.1. New features

Red Hat OpenShift Pipelines Technology Preview (TP) 1.2 is now available on OpenShift Container Platform 4.6. Red Hat OpenShift Pipelines TP 1.2 is updated to support:

- Tekton Pipelines 0.16.3
- Tekton **tkn** CLI 0.13.1

- Tekton Triggers 0.8.1
- cluster tasks based on Tekton Catalog 0.16
- IBM Power Systems on OpenShift Container Platform 4.6
- IBM Z and LinuxONE on OpenShift Container Platform 4.6

In addition to the fixes and stability improvements, the following sections highlight what is new in Red Hat OpenShift Pipelines 1.2.

3.1.10.1.1. Pipelines

- This release of Red Hat OpenShift Pipelines adds support for a disconnected installation.



NOTE

Installations in restricted environments are currently not supported on IBM Power Systems, IBM Z, and LinuxONE.

- You can now use the **when** field, instead of **conditions** resource, to run a task only when certain criteria are met. The key components of **WhenExpression** resources are **Input**, **Operator**, and **Values**. If all the when expressions evaluate to **True**, then the task is run. If any of the when expressions evaluate to **False**, the task is skipped.
- Step statuses are now updated if a task run is canceled or times out.
- Support for Git Large File Storage (LFS) is now available to build the base image used by **git-init**.
- You can now use the **taskSpec** field to specify metadata, such as labels and annotations, when a task is embedded in a pipeline.
- Cloud events are now supported by pipeline runs. Retries with **backoff** are now enabled for cloud events sent by the cloud event pipeline resource.
- You can now set a default **Workspace** configuration for any workspace that a **Task** resource declares, but that a **TaskRun** resource does not explicitly provide.
- Support is available for namespace variable interpolation for the **PipelineRun** namespace and **TaskRun** namespace.
- Validation for **TaskRun** objects is now added to check that not more than one persistent volume claim workspace is used when a **TaskRun** resource is associated with an Affinity Assistant. If more than one persistent volume claim workspace is used, the task run fails with a **TaskRunValidationFailed** condition. Note that by default, the Affinity Assistant is disabled in Red Hat OpenShift Pipelines, so you will need to enable the assistant to use it.

3.1.10.1.2. Pipelines CLI

- The **tkn task describe**, **tkn taskrun describe**, **tkn clustertask describe**, **tkn pipeline describe**, and **tkn pipelinerun describe** commands now:
 - Automatically select the **Task**, **TaskRun**, **ClusterTask**, **Pipeline** and **PipelineRun** resource, respectively, if only one of them is present.

- Display the results of the **Task**, **TaskRun**, **ClusterTask**, **Pipeline** and **PipelineRun** resource in their outputs, respectively.
- Display workspaces declared in the **Task**, **TaskRun**, **ClusterTask**, **Pipeline** and **PipelineRun** resource in their outputs, respectively.
- You can now use the **--prefix-name** option with the **tkn clustertask start** command to specify a prefix for the name of a task run.
- Interactive mode support has now been provided to the **tkn clustertask start** command.
- You can now specify **PodTemplate** properties supported by pipelines using local or remote file definitions for **TaskRun** and **PipelineRun** objects.
- You can now use the **--use-params-defaults** option with the **tkn clustertask start** command to use the default values set in the **ClusterTask** configuration and create the task run.
- The **--use-param-defaults** flag for the **tkn pipeline start** command now prompts the interactive mode if the default values have not been specified for some of the parameters.

3.1.10.1.3. Triggers

- The Common Expression Language (CEL) function named **parseYAML** has been added to parse a YAML string into a map of strings.
- Error messages for parsing CEL expressions have been improved to make them more granular while evaluating expressions and when parsing the hook body for creating the evaluation environment.
- Support is now available for marshaling boolean values and maps if they are used as the values of expressions in a CEL overlay mechanism.
- The following fields have been added to the **EventListener** object:
 - The **replicas** field enables the event listener to run more than one pod by specifying the number of replicas in the YAML file.
 - The **NodeSelector** field enables the **EventListener** object to schedule the event listener pod to a specific node.
- Webhook interceptors can now parse the **EventListener-Request-URL** header to extract parameters from the original request URL being handled by the event listener.
- Annotations from the event listener can now be propagated to the deployment, services, and other pods. Note that custom annotations on services or deployment are overwritten, and hence, must be added to the event listener annotations so that they are propagated.
- Proper validation for replicas in the **EventListener** specification is now available for cases when a user specifies the **spec.replicas** values as **negative** or **zero**.
- You can now specify the **TriggerCRD** object inside the **EventListener** spec as a reference using the **TriggerRef** field to create the **TriggerCRD** object separately and then bind it inside the **EventListener** spec.
- Validation and defaults for the **TriggerCRD** object are now available.

3.1.10.2. Deprecated features

- **\$(params)** parameters are now removed from the **triggertemplate** resource and replaced by **\$(tt.params)** to avoid confusion between the **resourcetemplate** and **triggertemplate** resource parameters.
- The **ServiceAccount** reference of the optional **EventListenerTrigger**-based authentication level has changed from an object reference to a **ServiceAccountName** string. This ensures that the **ServiceAccount** reference is in the same namespace as the **EventListenerTrigger** object.
- The **Conditions** custom resource definition (CRD) is now deprecated; use the **WhenExpressions** CRD instead.
- The **PipelineRun.Spec.ServiceAccountNames** object is being deprecated and replaced by the **PipelineRun.Spec.TaskRunSpec[].ServiceAccountName** object.

3.1.10.3. Known issues

- This release of Red Hat OpenShift Pipelines adds support for a disconnected installation. However, some images used by the cluster tasks must be mirrored for them to work in disconnected clusters.
- Pipelines in the **openshift** namespace are not deleted after you uninstall the Red Hat OpenShift Pipelines Operator. Use the **oc delete pipelines -n openshift --all** command to delete the pipelines.
- Uninstalling the Red Hat OpenShift Pipelines Operator does not remove the event listeners. As a workaround, to remove the **EventListener** and **Pod** CRDs:

1. Edit the **EventListener** object with the **foregroundDeletion** finalizers:

```
$ oc patch el/<eventlistener_name> -p '{"metadata":{"finalizers":["foregroundDeletion"]}}' --type=merge
```

For example:

```
$ oc patch el/github-listener-interceptor -p '{"metadata":{"finalizers":["foregroundDeletion"]}}' --type=merge
```

2. Delete the **EventListener** CRD:

```
$ oc patch crd/eventlisteners.triggers.tekton.dev -p '{"metadata":{"finalizers":[]}}' --type=merge
```

- When you run a multi-arch container image task without command specification on an IBM Power Systems (ppc64le) or IBM Z (s390x) cluster, the **TaskRun** resource fails with the following error:

```
Error executing command: fork/exec /bin/bash: exec format error
```

As a workaround, use an architecture specific container image or specify the sha256 digest to point to the correct architecture. To get the sha256 digest enter:

```
$ skopeo inspect --raw <image_name> | jq '.manifests[] | select(.platform.architecture == "<architecture>") | .digest'
```

3.1.10.4. Fixed issues

- A simple syntax validation to check the CEL filter, overlays in the Webhook validator, and the expressions in the interceptor has now been added.
- Triggers no longer overwrite annotations set on the underlying deployment and service objects.
- Previously, an event listener would stop accepting events. This fix adds an idle timeout of 120 seconds for the **EventListener** sink to resolve this issue.
- Previously, canceling a pipeline run with a **Failed(Canceled)** state gave a success message. This has been fixed to display an error instead.
- The **tkn eventlistener list** command now provides the status of the listed event listeners, thus enabling you to easily identify the available ones.
- Consistent error messages are now displayed for the **triggers list** and **triggers describe** commands when triggers are not installed or when a resource cannot be found.
- Previously, a large number of idle connections would build up during cloud event delivery. The **DisableKeepAlives: true** parameter was added to the **cloudeventclient** config to fix this issue. Thus, a new connection is set up for every cloud event.
- Previously, the **creds-init** code would write empty files to the disk even if credentials of a given type were not provided. This fix modifies the **creds-init** code to write files for only those credentials that have actually been mounted from correctly annotated secrets.

3.1.11. Release notes for Red Hat OpenShift Pipelines Technology Preview 1.1

3.1.11.1. New features

Red Hat OpenShift Pipelines Technology Preview (TP) 1.1 is now available on OpenShift Container Platform 4.5. Red Hat OpenShift Pipelines TP 1.1 is updated to support:

- Tekton Pipelines 0.14.3
- Tekton **tkn** CLI 0.11.0
- Tekton Triggers 0.6.1
- cluster tasks based on Tekton Catalog 0.14

In addition to the fixes and stability improvements, the following sections highlight what is new in Red Hat OpenShift Pipelines 1.1.

3.1.11.1.1. Pipelines

- Workspaces can now be used instead of pipeline resources. It is recommended that you use workspaces in OpenShift Pipelines, as pipeline resources are difficult to debug, limited in scope, and make tasks less reusable. For more details on workspaces, see the Understanding OpenShift Pipelines section.
- Workspace support for volume claim templates has been added:
 - The volume claim template for a pipeline run and task run can now be added as a volume source for workspaces. The tekton-controller then creates a persistent volume claim (PVC)

using the template that is seen as a PVC for all task runs in the pipeline. Thus you do not need to define the PVC configuration every time it binds a workspace that spans multiple tasks.

- Support to find the name of the PVC when a volume claim template is used as a volume source is now available using variable substitution.
- Support for improving audits:
 - The **PipelineRun.Status** field now contains the status of every task run in the pipeline and the pipeline specification used to instantiate a pipeline run to monitor the progress of the pipeline run.
 - Pipeline results have been added to the pipeline specification and **PipelineRun** status.
 - The **TaskRun.Status** field now contains the exact task specification used to instantiate the **TaskRun** resource.
- Support to apply the default parameter to conditions.
- A task run created by referencing a cluster task now adds the **tekton.dev/clusterTask** label instead of the **tekton.dev/task** label.
- The kube config writer now adds the **ClientKeyData** and the **ClientCertificateData** configurations in the resource structure to enable replacement of the pipeline resource type cluster with the kubeconfig-creator task.
- The names of the **feature-flags** and the **config-defaults** config maps are now customizable.
- Support for the host network in the pod template used by the task run is now available.
- An Affinity Assistant is now available to support node affinity in task runs that share workspace volume. By default, this is disabled on OpenShift Pipelines.
- The pod template has been updated to specify **imagePullSecrets** to identify secrets that the container runtime should use to authorize container image pulls when starting a pod.
- Support for emitting warning events from the task run controller if the controller fails to update the task run.
- Standard or recommended k8s labels have been added to all resources to identify resources belonging to an application or component.
- The **Entrypoint** process is now notified for signals and these signals are then propagated using a dedicated PID Group of the **Entrypoint** process.
- The pod template can now be set on a task level at runtime using task run specs.
- Support for emitting Kubernetes events:
 - The controller now emits events for additional task run lifecycle events – **taskrun started** and **taskrun running**.
 - The pipeline run controller now emits an event every time a pipeline starts.
- In addition to the default Kubernetes events, support for cloud events for task runs is now available. The controller can be configured to send any task run events, such as create, started, and failed, as cloud events.

- Support for using the **\$context.<task|taskRun|pipeline|pipelineRun>.name** variable to reference the appropriate name when in pipeline runs and task runs.
- Validation for pipeline run parameters is now available to ensure that all the parameters required by the pipeline are provided by the pipeline run. This also allows pipeline runs to provide extra parameters in addition to the required parameters.
- You can now specify tasks within a pipeline that will always execute before the pipeline exits, either after finishing all tasks successfully or after a task in the pipeline failed, using the **finally** field in the pipeline YAML file.
- The **git-clone** cluster task is now available.

3.1.11.1.2. Pipelines CLI

- Support for embedded trigger binding is now available to the **tkn evenlistener describe** command.
- Support to recommend subcommands and make suggestions if an incorrect subcommand is used.
- The **tkn task describe** command now auto selects the task if only one task is present in the pipeline.
- You can now start a task using default parameter values by specifying the **--use-param-defaults** flag in the **tkn task start** command.
- You can now specify a volume claim template for pipeline runs or task runs using the **--workspace** option with the **tkn pipeline start** or **tkn task start** commands.
- The **tkn pipelinerun logs** command now displays logs for the final tasks listed in the **finally** section.
- Interactive mode support has now been provided to the **tkn task start** command and the **describe** subcommand for the following **tkn** resources: **pipeline**, **pipelinerun**, **task**, **taskrun**, **clustertask**, and **pipelineresource**.
- The **tkn version** command now displays the version of the triggers installed in the cluster.
- The **tkn pipeline describe** command now displays parameter values and timeouts specified for tasks used in the pipeline.
- Support added for the **--last** option for the **tkn pipelinerun describe** and the **tkn taskrun describe** commands to describe the most recent pipeline run or task run, respectively.
- The **tkn pipeline describe** command now displays the conditions applicable to the tasks in the pipeline.
- You can now use the **--no-headers** and **--all-namespaces** flags with the **tkn resource list** command.

3.1.11.1.3. Triggers

- The following Common Expression Language (CEL) functions are now available:
 - **parseURL** to parse and extract portions of a URL

- **parseJSON** to parse JSON value types embedded in a string in the **payload** field of the **deployment** webhook
- A new interceptor for webhooks from Bitbucket has been added.
- Event listeners now display the **Address URL** and the **Available status** as additional fields when listed with the **kubectrl get** command.
- trigger template params now use the **\$(tt.params.<paramName>)** syntax instead of **\$(params.<paramName>)** to reduce the confusion between trigger template and resource templates params.
- You can now add **tolerations** in the **EventListener** CRD to ensure that event listeners are deployed with the same configuration even if all nodes are tainted due to security or management issues.
- You can now add a Readiness Probe for event listener Deployment at **URL/live**.
- Support for embedding **TriggerBinding** specifications in event listener triggers is now added.
- Trigger resources are now annotated with the recommended **app.kubernetes.io** labels.

3.1.11.2. Deprecated features

The following items are deprecated in this release:

- The **--namespace** or **-n** flags for all cluster-wide commands, including the **clustertask** and **clustertriggerbinding** commands, are deprecated. It will be removed in a future release.
- The **name** field in **triggers.bindings** within an event listener has been deprecated in favor of the **ref** field and will be removed in a future release.
- Variable interpolation in trigger templates using **\$(params)** has been deprecated in favor of using **\$(tt.params)** to reduce confusion with the pipeline variable interpolation syntax. The **\$(params.<paramName>)** syntax will be removed in a future release.
- The **tekton.dev/task** label is deprecated on cluster tasks.
- The **TaskRun.Status.ResourceResults.ResourceRef** field is deprecated and will be removed.
- The **tkn pipeline create**, **tkn task create**, and **tkn resource create -f** subcommands have been removed.
- Namespace validation has been removed from **tkn** commands.
- The default timeout of **1h** and the **-t** flag for the **tkn ct start** command have been removed.
- The **s2i** cluster task has been deprecated.

3.1.11.3. Known issues

- Conditions do not support workspaces.
- The **--workspace** option and the interactive mode is not supported for the **tkn clustertask start** command.

- Support of backward compatibility for **\$(params.<paramName>)** syntax forces you to use trigger templates with pipeline specific params as the trigger's webhook is unable to differentiate trigger params from pipelines params.
- Pipeline metrics report incorrect values when you run a promQL query for **tekton_taskrun_count** and **tekton_taskrun_duration_seconds_count**.
- pipeline runs and task runs continue to be in the **Running** and **Running(Pending)** states respectively even when a non existing PVC name is given to a workspace.

3.1.11.4. Fixed issues

- Previously, the **tkn task delete <name> --trs** command would delete both the task and cluster task if the name of the task and cluster task were the same. With this fix, the command deletes only the task runs that are created by the task **<name>**.
- Previously the **tkn pr delete -p <name> --keep 2** command would disregard the **-p** flag when used with the **--keep** flag and would delete all the pipeline runs except the latest two. With this fix, the command deletes only the pipeline runs that are created by the pipeline **<name>**, except for the latest two.
- The **tkn triggertemplate describe** output now displays resource templates in a table format instead of YAML format.
- Previously the **buildah** cluster task failed when a new user was added to a container. With this fix, the issue has been resolved.

3.1.12. Release notes for Red Hat OpenShift Pipelines Technology Preview 1.0

3.1.12.1. New features

Red Hat OpenShift Pipelines Technology Preview (TP) 1.0 is now available on OpenShift Container Platform 4.4. Red Hat OpenShift Pipelines TP 1.0 is updated to support:

- Tekton Pipelines 0.11.3
- Tekton **tkn** CLI 0.9.0
- Tekton Triggers 0.4.0
- cluster tasks based on Tekton Catalog 0.11

In addition to the fixes and stability improvements, the following sections highlight what is new in Red Hat OpenShift Pipelines 1.0.

3.1.12.1.1. Pipelines

- Support for v1beta1 API Version.
- Support for an improved limit range. Previously, limit range was specified exclusively for the task run and the pipeline run. Now there is no need to explicitly specify the limit range. The minimum limit range across the namespace is used.
- Support for sharing data between tasks using task results and task params.

- Pipelines can now be configured to not overwrite the **HOME** environment variable and the working directory of steps.
- Similar to task steps, **sidecars** now support script mode.
- You can now specify a different scheduler name in task run **podTemplate** resource.
- Support for variable substitution using Star Array Notation.
- Tekton controller can now be configured to monitor an individual namespace.
- A new description field is now added to the specification of pipelines, tasks, cluster tasks, resources, and conditions.
- Addition of proxy parameters to Git pipeline resources.

3.1.12.1.2. Pipelines CLI

- The **describe** subcommand is now added for the following **tkn** resources: **EventListener**, **Condition**, **TriggerTemplate**, **ClusterTask**, and **TriggerSBinding**.
- Support added for **v1beta1** to the following resources along with backward compatibility for **v1alpha1**: **ClusterTask**, **Task**, **Pipeline**, **PipelineRun**, and **TaskRun**.
- The following commands can now list output from all namespaces using the **--all-namespaces** flag option: **tkn task list**, **tkn pipeline list**, **tkn taskrun list**, **tkn pipelinerun list**
The output of these commands is also enhanced to display information without headers using the **--no-headers** flag option.
- You can now start a pipeline using default parameter values by specifying **--use-param-defaults** flag in the **tkn pipelines start** command.
- Support for workspace is now added to **tkn pipeline start** and **tkn task start** commands.
- A new **clustertriggerbinding** command is now added with the following subcommands: **describe**, **delete**, and **list**.
- You can now directly start a pipeline run using a local or remote **yaml** file.
- The **describe** subcommand now displays an enhanced and detailed output. With the addition of new fields, such as **description**, **timeout**, **param description**, and **sidecar status**, the command output now provides more detailed information about a specific **tkn** resource.
- The **tkn task log** command now displays logs directly if only one task is present in the namespace.

3.1.12.1.3. Triggers

- Triggers can now create both **v1alpha1** and **v1beta1** pipeline resources.
- Support for new Common Expression Language (CEL) interceptor function - **compareSecret**. This function securely compares strings to secrets in CEL expressions.
- Support for authentication and authorization at the event listener trigger level.

3.1.12.2. Deprecated features

The following items are deprecated in this release:

- The environment variable **\$HOME**, and variable **workingDir** in the **Steps** specification are deprecated and might be changed in a future release. Currently in a **Step** container, the **HOME** and **workingDir** variables are overwritten to **/tekton/home** and **/workspace** variables, respectively.
In a later release, these two fields will not be modified, and will be set to values defined in the container image and the **Task** YAML. For this release, use the **disable-home-env-overwrite** and **disable-working-directory-overwrite** flags to disable overwriting of the **HOME** and **workingDir** variables.
- The following commands are deprecated and might be removed in the future release: **tkn pipeline create**, **tkn task create**.
- The **-f** flag with the **tkn resource create** command is now deprecated. It might be removed in the future release.
- The **-t** flag and the **--timeout** flag (with seconds format) for the **tkn clustertask create** command are now deprecated. Only duration timeout format is now supported, for example **1h30s**. These deprecated flags might be removed in the future release.

3.1.12.3. Known issues

- If you are upgrading from an older version of Red Hat OpenShift Pipelines, you must delete your existing deployments before upgrading to Red Hat OpenShift Pipelines version 1.0. To delete an existing deployment, you must first delete Custom Resources and then uninstall the Red Hat OpenShift Pipelines Operator. For more details, see the uninstalling Red Hat OpenShift Pipelines section.
- Submitting the same **v1alpha1** tasks more than once results in an error. Use the **oc replace** command instead of **oc apply** when re-submitting a **v1alpha1** task.
- The **buildah** cluster task does not work when a new user is added to a container. When the Operator is installed, the **--storage-driver** flag for the **buildah** cluster task is not specified, therefore the flag is set to its default value. In some cases, this causes the storage driver to be set incorrectly. When a new user is added, the incorrect storage-driver results in the failure of the **buildah** cluster task with the following error:

```
useradd: /etc/passwd.8: lock file already used
useradd: cannot lock /etc/passwd; try again later.
```

As a workaround, manually set the **--storage-driver** flag value to **overlay** in the **buildah-task.yaml** file:

1. Login to your cluster as a **cluster-admin**:

```
$ oc login -u <login> -p <password> https://openshift.example.com:6443
```

2. Use the **oc edit** command to edit **buildah** cluster task:

```
$ oc edit clustertask buildah
```

The current version of the **buildah** clustertask YAML file opens in the editor set by your **EDITOR** environment variable.

- Under the **Steps** field, locate the following **command** field:

```
command: ['buildah', 'bud', '--format=$(params.FORMAT)', '--tls-verify=$(params.TLSVERIFY)', '--layers', '-f', '$(params.DOCKERFILE)', '-t', '$(resources.outputs.image.url)', '$(params.CONTEXT)']
```

- Replace the **command** field with the following:

```
command: ['buildah', '--storage-driver=overlay', 'bud', '--format=$(params.FORMAT)', '--tls-verify=$(params.TLSVERIFY)', '--no-cache', '-f', '$(params.DOCKERFILE)', '-t', '$(params.IMAGE)', '$(params.CONTEXT)']
```

- Save the file and exit.

Alternatively, you can also modify the **buildah** cluster task YAML file directly on the web console by navigating to **Pipelines → Cluster Tasks → buildah**. Select **Edit Cluster Task** from the **Actions** menu and replace the **command** field as shown in the previous procedure.

3.1.12.4. Fixed issues

- Previously, the **DeploymentConfig** task triggered a new deployment build even when an image build was already in progress. This caused the deployment of the pipeline to fail. With this fix, the **deploy task** command is now replaced with the **oc rollout status** command which waits for the in-progress deployment to finish.
- Support for **APP_NAME** parameter is now added in pipeline templates.
- Previously, the pipeline template for Java S2I failed to look up the image in the registry. With this fix, the image is looked up using the existing image pipeline resources instead of the user provided **IMAGE_NAME** parameter.
- All the OpenShift Pipelines images are now based on the Red Hat Universal Base Images (UBI).
- Previously, when the pipeline was installed in a namespace other than **tekton-pipelines**, the **tkn version** command displayed the pipeline version as **unknown**. With this fix, the **tkn version** command now displays the correct pipeline version in any namespace.
- The **-c** flag is no longer supported for the **tkn version** command.
- Non-admin users can now list the cluster trigger bindings.
- The event listener **CompareSecret** function is now fixed for the CEL Interceptor.
- The **list**, **describe**, and **start** subcommands for tasks and cluster tasks now correctly display the output in case a task and cluster task have the same name.
- Previously, the OpenShift Pipelines Operator modified the privileged security context constraints (SCCs), which caused an error during cluster upgrade. This error is now fixed.
- In the **tekton-pipelines** namespace, the timeouts of all task runs and pipeline runs are now set to the value of **default-timeout-minutes** field using the config map.
- Previously, the pipelines section in the web console was not displayed for non-admin users. This issue is now resolved.

3.2. UNDERSTANDING OPENSIFT PIPELINES

Red Hat OpenShift Pipelines is a cloud-native, continuous integration and continuous delivery (CI/CD) solution based on Kubernetes resources. It uses Tekton building blocks to automate deployments across multiple platforms by abstracting away the underlying implementation details. Tekton introduces a number of standard custom resource definitions (CRDs) for defining CI/CD pipelines that are portable across Kubernetes distributions.

3.2.1. Key features

- Red Hat OpenShift Pipelines is a serverless CI/CD system that runs pipelines with all the required dependencies in isolated containers.
- Red Hat OpenShift Pipelines are designed for decentralized teams that work on microservice-based architecture.
- Red Hat OpenShift Pipelines use standard CI/CD pipeline definitions that are easy to extend and integrate with the existing Kubernetes tools, enabling you to scale on-demand.
- You can use Red Hat OpenShift Pipelines to build images with Kubernetes tools such as Source-to-Image (S2I), Buildah, Buildpacks, and Kaniko that are portable across any Kubernetes platform.
- You can use the OpenShift Container Platform Developer console to create Tekton resources, view logs of pipeline runs, and manage pipelines in your OpenShift Container Platform namespaces.

3.2.2. OpenShift Pipeline Concepts

This guide provides a detailed view of the various pipeline concepts.

3.2.2.1. Tasks

Tasks are the building blocks of a pipeline and consists of sequentially executed steps. It is essentially a function of inputs and outputs. A task can run individually or as a part of the pipeline. Tasks are reusable and can be used in multiple Pipelines.

Steps are a series of commands that are sequentially executed by the task and achieve a specific goal, such as building an image. Every task runs as a pod, and each step runs as a container within that pod. Because steps run within the same pod, they can access the same volumes for caching files, config maps, and secrets.

The following example shows the **apply-manifests** task.

```
apiVersion: tekton.dev/v1beta1 1
kind: Task 2
metadata:
  name: apply-manifests 3
spec: 4
  workspaces:
    - name: source
  params:
    - name: manifest_dir
      description: The directory in source that contains yaml manifests
      type: string
```

```

    default: "k8s"
  steps:
  - name: apply
    image: image-registry.openshift-image-registry.svc:5000/openshift/cli:latest
    workingDir: /workspace/source
    command: ["/bin/bash", "-c"]
    args:
    - |-
      echo Applying manifests in $(params.manifest_dir) directory
      oc apply -f $(params.manifest_dir)
      echo -----

```

- 1 The task API version, **v1beta1**.
- 2 The type of Kubernetes object, **Task**.
- 3 The unique name of this task.
- 4 The list of parameters and steps in the task and the workspace used by the task.

This task starts the pod and runs a container inside that pod using the specified image to run the specified commands.

NOTE

Starting with Pipelines 1.6, the following defaults from the step YAML file are removed:

- The **HOME** environment variable does not default to the **/tekton/home** directory
- The **workingDir** field does not default to the **/workspace** directory

Instead, the container for the step defines the **HOME** environment variable and the **workingDir** field. However, you can override the default values by specifying the custom values in the YAML file for the step.

As a temporary measure, to maintain backward compatibility with the older Pipelines versions, you can set the following fields in the **TektonConfig** custom resource definition to **false**:

```

spec:
  pipeline:
    disable-working-directory-overwrite: false
    disable-home-env-overwrite: false

```

3.2.2.2. When expression

When expressions guard task execution by setting criteria for the execution of tasks within a pipeline. They contain a list of components that allows a task to run only when certain criteria are met. When expressions are also supported in the final set of tasks that are specified using the **finally** field in the pipeline YAML file.

The key components of a when expression are as follows:

- **input:** Specifies static inputs or variables such as a parameter, task result, and execution status. You must enter a valid input. If you do not enter a valid input, its value defaults to an empty string.
- **operator:** Specifies the relationship of an input to a set of **values**. Enter **in** or **notin** as your operator values.
- **values:** Specifies an array of string values. Enter a non-empty array of static values or variables such as parameters, results, and a bound state of a workspace.

The declared when expressions are evaluated before the task is run. If the value of a when expression is **True**, the task is run. If the value of a when expression is **False**, the task is skipped.

You can use the when expressions in various use cases. For example, whether:

- The result of a previous task is as expected.
- A file in a Git repository has changed in the previous commits.
- An image exists in the registry.
- An optional workspace is available.

The following example shows the when expressions for a pipeline run. The pipeline run will execute the **create-file** task only if the following criteria are met: the **path** parameter is **README.md**, and the **echo-file-exists** task executed only if the **exists** result from the **check-file** task is **yes**.

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun 1
metadata:
  generateName: guarded-pr-
spec:
  serviceAccountName: 'pipeline'
  pipelineSpec:
    params:
      - name: path
        type: string
        description: The path of the file to be created
    workspaces:
      - name: source
        description: |
          This workspace is shared among all the pipeline tasks to read/write common resources
    tasks:
      - name: create-file 2
        when:
          - input: "$(params.path)"
            operator: in
            values: ["README.md"]
        workspaces:
          - name: source
            workspace: source
        taskSpec:
          workspaces:
            - name: source
              description: The workspace to create the readme file in
          steps:
```

```

      - name: write-new-stuff
        image: ubuntu
        script: 'touch ${workspaces.source.path}/README.md'
- name: check-file
  params:
    - name: path
      value: "${params.path}"
  workspaces:
    - name: source
      workspace: source
  runAfter:
    - create-file
  taskSpec:
    params:
      - name: path
    workspaces:
      - name: source
        description: The workspace to check for the file
    results:
      - name: exists
        description: indicates whether the file exists or is missing
    steps:
      - name: check-file
        image: alpine
        script: |
          if test -f ${workspaces.source.path}/${params.path}; then
            printf yes | tee /tekton/results/exists
          else
            printf no | tee /tekton/results/exists
          fi
- name: echo-file-exists
  when: 3
  - input: "${tasks.check-file.results.exists}"
    operator: in
    values: ["yes"]
  taskSpec:
    steps:
      - name: echo
        image: ubuntu
        script: 'echo file exists'
...
- name: task-should-be-skipped-1
  when: 4
  - input: "${params.path}"
    operator: notin
    values: ["README.md"]
  taskSpec:
    steps:
      - name: echo
        image: ubuntu
        script: exit 1
...
finally:
  - name: finally-task-should-be-executed
    when: 5
    - input: "${tasks.echo-file-exists.status}"

```

```

      operator: in
      values: ["Succeeded"]
    - input: "${tasks.status}"
      operator: in
      values: ["Succeeded"]
    - input: "${tasks.check-file.results.exists}"
      operator: in
      values: ["yes"]
    - input: "${params.path}"
      operator: in
      values: ["README.md"]
  taskSpec:
    steps:
      - name: echo
        image: ubuntu
        script: 'echo finally done'
  params:
    - name: path
      value: README.md
  workspaces:
    - name: source
      volumeClaimTemplate:
        spec:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 16Mi

```

- 1 Specifies the type of Kubernetes object. In this example, **PipelineRun**.
- 2 Task **create-file** used in the Pipeline.
- 3 **when** expression that specifies to execute the **echo-file-exists** task only if the **exists** result from the **check-file** task is **yes**.
- 4 **when** expression that specifies to skip the **task-should-be-skipped-1** task only if the **path** parameter is **README.md**.
- 5 **when** expression that specifies to execute the **finally-task-should-be-executed** task only if the execution status of the **echo-file-exists** task and the task status is **Succeeded**, the **exists** result from the **check-file** task is **yes**, and the **path** parameter is **README.md**.

The **Pipeline Run details** page of the OpenShift Container Platform web console shows the status of the tasks and when expressions as follows:

- All the criteria are met: Tasks and the when expression symbol, which is represented by a diamond shape are green.
- Any one of the criteria are not met: Task is skipped. Skipped tasks and the when expression symbol are grey.
- None of the criteria are met: Task is skipped. Skipped tasks and the when expression symbol are grey.
- Task run fails: Failed tasks and the when expression symbol are red.

3.2.2.3. Finally tasks

The **finally** tasks are the final set of tasks specified using the **finally** field in the pipeline YAML file. A **finally** task always executes the tasks within the pipeline, irrespective of whether the pipeline runs are executed successfully. The **finally** tasks are executed in parallel after all the pipeline tasks are run, before the corresponding pipeline exits.

You can configure a **finally** task to consume the results of any task within the same pipeline. This approach does not change the order in which this final task is run. It is executed in parallel with other final tasks after all the non-final tasks are executed.

The following example shows a code snippet of the **clone-cleanup-workspace** pipeline. This code clones the repository into a shared workspace and cleans up the workspace. After executing the pipeline tasks, the **cleanup** task specified in the **finally** section of the pipeline YAML file cleans up the workspace.

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: clone-cleanup-workspace ❶
spec:
  workspaces:
    - name: git-source ❷
  tasks:
    - name: clone-app-repo ❸
      taskRef:
        name: git-clone-from-catalog
      params:
        - name: url
          value: https://github.com/tektoncd/community.git
        - name: subdirectory
          value: application
      workspaces:
        - name: output
          workspace: git-source
  finally:
    - name: cleanup ❹
      taskRef: ❺
        name: cleanup-workspace
      workspaces: ❻
        - name: source
          workspace: git-source
    - name: check-git-commit
      params: ❼
        - name: commit
          value: $(tasks.clone-app-repo.results.commit)
      taskSpec: ❽
        params:
          - name: commit
        steps:
          - name: check-commit-initialized
            image: alpine
            script: |
```



```

    if [[ ! $(params.commit) ]]; then
        exit 1
    fi

```

- 1 Unique name of the Pipeline.
- 2 The shared workspace where the git repository is cloned.
- 3 The task to clone the application repository to the shared workspace.
- 4 The task to clean-up the shared workspace.
- 5 A reference to the task that is to be executed in the TaskRun.
- 6 A shared storage volume that a Task in a Pipeline needs at runtime to receive input or provide output.
- 7 A list of parameters required for a task. If a parameter does not have an implicit default value, you must explicitly set its value.
- 8 Embedded task definition.

3.2.2.4. TaskRun

A *TaskRun* instantiates a Task for execution with specific inputs, outputs, and execution parameters on a cluster. It can be invoked on its own or as part of a *PipelineRun* for each Task in a pipeline.

A Task consists of one or more Steps that execute container images, and each container image performs a specific piece of build work. A TaskRun executes the Steps in a Task in the specified order, until all Steps execute successfully or a failure occurs. A TaskRun is automatically created by a *PipelineRun* for each Task in a Pipeline.

The following example shows a TaskRun that runs the **apply-manifests** Task with the relevant input parameters:

```

apiVersion: tekton.dev/v1beta1 1
kind: TaskRun 2
metadata:
  name: apply-manifests-taskrun 3
spec: 4
  serviceAccountName: pipeline
  taskRef: 5
    kind: Task
    name: apply-manifests
  workspaces: 6
    - name: source
      persistentVolumeClaim:
        claimName: source-pvc

```

- 1 TaskRun API version **v1beta1**.
- 2 Specifies the type of Kubernetes object. In this example, **TaskRun**.
- 3 Unique name to identify this TaskRun.

- 4 Definition of the TaskRun. For this TaskRun, the Task and the required workspace are specified.
- 5 Name of the Task reference used for this TaskRun. This TaskRun executes the **apply-manifests** Task.
- 6 Workspace used by the TaskRun.

3.2.2.5. Pipelines

A *Pipeline* is a collection of **Task** resources arranged in a specific order of execution. They are executed to construct complex workflows that automate the build, deployment and delivery of applications. You can define a CI/CD workflow for your application using pipelines containing one or more tasks.

A **Pipeline** resource definition consists of a number of fields or attributes, which together enable the pipeline to accomplish a specific goal. Each **Pipeline** resource definition must contain at least one **Task** resource, which ingests specific inputs and produces specific outputs. The pipeline definition can also optionally include *Conditions*, *Workspaces*, *Parameters*, or *Resources* depending on the application requirements.

The following example shows the **build-and-deploy** pipeline, which builds an application image from a Git repository using the **buildah ClusterTask** resource:

```
apiVersion: tekton.dev/v1beta1 1
kind: Pipeline 2
metadata:
  name: build-and-deploy 3
spec: 4
  workspaces: 5
  - name: shared-workspace
  params: 6
  - name: deployment-name
    type: string
    description: name of the deployment to be patched
  - name: git-url
    type: string
    description: url of the git repo for the code of deployment
  - name: git-revision
    type: string
    description: revision to be used from repo of the code for deployment
    default: "pipelines-1.8"
  - name: IMAGE
    type: string
    description: image to be built from the code
  tasks: 7
  - name: fetch-repository
    taskRef:
      name: git-clone
      kind: ClusterTask
    workspaces:
      - name: output
        workspace: shared-workspace
    params:
      - name: url
        value: $(params.git-url)
```

```

- name: subdirectory
  value: ""
- name: deleteExisting
  value: "true"
- name: revision
  value: $(params.git-revision)
- name: build-image 8
  taskRef:
    name: buildah
    kind: ClusterTask
  params:
- name: TLSVERIFY
  value: "false"
- name: IMAGE
  value: $(params.IMAGE)
  workspaces:
- name: source
  workspace: shared-workspace
  runAfter:
- fetch-repository
- name: apply-manifests 9
  taskRef:
    name: apply-manifests
  workspaces:
- name: source
  workspace: shared-workspace
  runAfter: 10
- build-image
- name: update-deployment
  taskRef:
    name: update-deployment
  workspaces:
- name: source
  workspace: shared-workspace
  params:
- name: deployment
  value: $(params.deployment-name)
- name: IMAGE
  value: $(params.IMAGE)
  runAfter:
- apply-manifests

```

- 1 Pipeline API version **v1beta1**.
- 2 Specifies the type of Kubernetes object. In this example, **Pipeline**.
- 3 Unique name of this Pipeline.
- 4 Specifies the definition and structure of the Pipeline.
- 5 Workspaces used across all the Tasks in the Pipeline.
- 6 Parameters used across all the Tasks in the Pipeline.
- 7 Specifies the list of Tasks used in the Pipeline.

- 8 Task **build-image**, which uses the **buildah** ClusterTask to build application images from a given Git repository.
- 9 Task **apply-manifests**, which uses a user-defined Task with the same name.
- 10 Specifies the sequence in which Tasks are run in a Pipeline. In this example, the **apply-manifests** Task is run only after the **build-image** Task is completed.



NOTE

The Red Hat OpenShift Pipelines Operator installs the Buildah cluster task and creates the **pipeline** service account with sufficient permission to build and push an image. The Buildah cluster task can fail when associated with a different service account with insufficient permissions.

3.2.2.6. PipelineRun

A **PipelineRun** is a type of resource that binds a pipeline, workspaces, credentials, and a set of parameter values specific to a scenario to run the CI/CD workflow.

A *pipeline run* is the running instance of a pipeline. It instantiates a pipeline for execution with specific inputs, outputs, and execution parameters on a cluster. It also creates a task run for each task in the pipeline run.

The pipeline runs the tasks sequentially until they are complete or a task fails. The **status** field tracks and the progress of each task run and stores it for monitoring and auditing purposes.

The following example runs the **build-and-deploy** pipeline with relevant resources and parameters:

```
apiVersion: tekton.dev/v1beta1 1
kind: PipelineRun 2
metadata:
  name: build-deploy-api-pipelinerun 3
spec:
  pipelineRef:
    name: build-and-deploy 4
  params: 5
  - name: deployment-name
    value: vote-api
  - name: git-url
    value: https://github.com/openshift-pipelines/vote-api.git
  - name: IMAGE
    value: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-api
  workspaces: 6
  - name: shared-workspace
    volumeClaimTemplate:
      spec:
        accessModes:
          - ReadWriteOnce
      resources:
        requests:
          storage: 500Mi
```

- 1 Pipeline run API version **v1beta1**.
- 2 The type of Kubernetes object. In this example, **PipelineRun**.
- 3 Unique name to identify this pipeline run.
- 4 Name of the pipeline to be run. In this example, **build-and-deploy**.
- 5 The list of parameters required to run the pipeline.
- 6 Workspace used by the pipeline run.

Additional resources

- [Authenticating pipelines using git secret](#)

3.2.2.7. Workspaces



NOTE

It is recommended that you use Workspaces instead of PipelineResources in OpenShift Pipelines, as PipelineResources are difficult to debug, limited in scope, and make Tasks less reusable.

Workspaces declare shared storage volumes that a Task in a Pipeline needs at runtime to receive input or provide output. Instead of specifying the actual location of the volumes, Workspaces enable you to declare the filesystem or parts of the filesystem that would be required at runtime. A Task or Pipeline declares the Workspace and you must provide the specific location details of the volume. It is then mounted into that Workspace in a TaskRun or a PipelineRun. This separation of volume declaration from runtime storage volumes makes the Tasks reusable, flexible, and independent of the user environment.

With Workspaces, you can:

- Store Task inputs and outputs
- Share data among Tasks
- Use it as a mount point for credentials held in Secrets
- Use it as a mount point for configurations held in ConfigMaps
- Use it as a mount point for common tools shared by an organization
- Create a cache of build artifacts that speed up jobs

You can specify Workspaces in the TaskRun or PipelineRun using:

- A read-only ConfigMaps or Secret
- An existing PersistentVolumeClaim shared with other Tasks
- A PersistentVolumeClaim from a provided VolumeClaimTemplate
- An emptyDir that is discarded when the TaskRun completes

The following example shows a code snippet of the **build-and-deploy** Pipeline, which declares a **shared-workspace** Workspace for the **build-image** and **apply-manifests** Tasks as defined in the Pipeline.

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  workspaces: ❶
  - name: shared-workspace
  params:
  ...
  tasks: ❷
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
      - name: TLSVERIFY
        value: "false"
      - name: IMAGE
        value: $(params.IMAGE)
    workspaces: ❸
    - name: source ❹
      workspace: shared-workspace ❺
  runAfter:
    - fetch-repository
  - name: apply-manifests
    taskRef:
      name: apply-manifests
    workspaces: ❻
    - name: source
      workspace: shared-workspace
  runAfter:
    - build-image
  ...
```

❶ List of Workspaces shared between the Tasks defined in the Pipeline. A Pipeline can define as many Workspaces as required. In this example, only one Workspace named **shared-workspace** is declared.

❷ Definition of Tasks used in the Pipeline. This snippet defines two Tasks, **build-image** and **apply-manifests**, which share a common Workspace.

❸ List of Workspaces used in the **build-image** Task. A Task definition can include as many Workspaces as it requires. However, it is recommended that a Task uses at most one writable Workspace.

❹ Name that uniquely identifies the Workspace used in the Task. This Task uses one Workspace named **source**.

❺ Name of the Pipeline Workspace used by the Task. Note that the Workspace **source** in turn uses the Pipeline Workspace named **shared-workspace**.

❻

List of Workspaces used in the **apply-manifests** Task. Note that this Task shares the **source** Workspace with the **build-image** Task.

Workspaces help tasks share data, and allow you to specify one or more volumes that each task in the pipeline requires during execution. You can create a persistent volume claim or provide a volume claim template that creates a persistent volume claim for you.

The following code snippet of the **build-deploy-api-pipelinerun** PipelineRun uses a volume claim template to create a persistent volume claim for defining the storage volume for the **shared-workspace** Workspace used in the **build-and-deploy** Pipeline.

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: build-deploy-api-pipelinerun
spec:
  pipelineRef:
    name: build-and-deploy
  params:
  ...

  workspaces: ❶
  - name: shared-workspace ❷
    volumeClaimTemplate: ❸
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 500Mi
```

- ❶ Specifies the list of Pipeline Workspaces for which volume binding will be provided in the PipelineRun.
- ❷ The name of the Workspace in the Pipeline for which the volume is being provided.
- ❸ Specifies a volume claim template that creates a persistent volume claim to define the storage volume for the workspace.

3.2.2.8. Triggers

Use *Triggers* in conjunction with pipelines to create a full-fledged CI/CD system where Kubernetes resources define the entire CI/CD execution. Triggers capture the external events, such as a Git pull request, and process them to extract key pieces of information. Mapping this event data to a set of predefined parameters triggers a series of tasks that can then create and deploy Kubernetes resources and instantiate the pipeline.

For example, you define a CI/CD workflow using Red Hat OpenShift Pipelines for your application. The pipeline must start for any new changes to take effect in the application repository. Triggers automate this process by capturing and processing any change event and by triggering a pipeline run that deploys the new image with the latest changes.

Triggers consist of the following main resources that work together to form a reusable, decoupled, and self-sustaining CI/CD system:

- The **TriggerBinding** resource extracts the fields from an event payload and stores them as parameters.

The following example shows a code snippet of the **TriggerBinding** resource, which extracts the Git repository information from the received event payload:

```
apiVersion: triggers.tekton.dev/v1beta1 1
kind: TriggerBinding 2
metadata:
  name: vote-app 3
spec:
  params: 4
  - name: git-repo-url
    value: $(body.repository.url)
  - name: git-repo-name
    value: $(body.repository.name)
  - name: git-revision
    value: $(body.head_commit.id)
```

- 1** The API version of the **TriggerBinding** resource. In this example, **v1beta1**.
- 2** Specifies the type of Kubernetes object. In this example, **TriggerBinding**.
- 3** Unique name to identify the **TriggerBinding** resource.
- 4** List of parameters which will be extracted from the received event payload and passed to the **TriggerTemplate** resource. In this example, the Git repository URL, name, and revision are extracted from the body of the event payload.

- The **TriggerTemplate** resource acts as a standard for the way resources must be created. It specifies the way parameterized data from the **TriggerBinding** resource should be used. A trigger template receives input from the trigger binding, and then performs a series of actions that results in creation of new pipeline resources, and initiation of a new pipeline run.

The following example shows a code snippet of a **TriggerTemplate** resource, which creates a pipeline run using the Git repository information received from the **TriggerBinding** resource you just created:

```
apiVersion: triggers.tekton.dev/v1beta1 1
kind: TriggerTemplate 2
metadata:
  name: vote-app 3
spec:
  params: 4
  - name: git-repo-url
    description: The git repository url
  - name: git-revision
    description: The git revision
    default: pipelines-1.8
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates: 5
  - apiVersion: tekton.dev/v1beta1
    kind: PipelineRun
```



```

metadata:
  name: build-deploy-${tt.params.git-repo-name}-${uid}
spec:
  serviceAccountName: pipeline
  pipelineRef:
    name: build-and-deploy
  params:
    - name: deployment-name
      value: ${tt.params.git-repo-name}
    - name: git-url
      value: ${tt.params.git-repo-url}
    - name: git-revision
      value: ${tt.params.git-revision}
    - name: IMAGE
      value: image-registry.openshift-image-registry.svc:5000/pipelines-
tutorial/${tt.params.git-repo-name}
  workspaces:
    - name: shared-workspace
  volumeClaimTemplate:
    spec:
      accessModes:
        - ReadWriteOnce
    resources:
      requests:
        storage: 500Mi

```

- ❶ The API version of the **TriggerTemplate** resource. In this example, **v1beta1**.
 - ❷ Specifies the type of Kubernetes object. In this example, **TriggerTemplate**.
 - ❸ Unique name to identify the **TriggerTemplate** resource.
 - ❹ Parameters supplied by the **TriggerBinding** resource.
 - ❺ List of templates that specify the way resources must be created using the parameters received through the **TriggerBinding** or **EventListener** resources.
- The **Trigger** resource combines the **TriggerBinding** and **TriggerTemplate** resources, and optionally, the **interceptors** event processor. Interceptors process all the events for a specific platform that runs before the **TriggerBinding** resource. You can use interceptors to filter the payload, verify events, define and test trigger conditions, and implement other useful processing. Interceptors use secret for event verification. Once the event data passes through an interceptor, it then goes to the trigger before you pass the payload data to the trigger binding. You can also use an interceptor to modify the behavior of the associated trigger referenced in the **EventListener** specification.

The following example shows a code snippet of a **Trigger** resource, named **vote-trigger** that connects the **TriggerBinding** and **TriggerTemplate** resources, and the **interceptors** event processor.

```

apiVersion: triggers.tekton.dev/v1beta1 ❶
kind: Trigger ❷
metadata:
  name: vote-trigger ❸
spec:

```

```

serviceAccountName: pipeline ❹
interceptors:
- ref:
  name: "github" ❺
  params: ❻
  - name: "secretRef"
    value:
      secretName: github-secret
      secretKey: secretToken
  - name: "eventTypes"
    value: ["push"]
bindings:
- ref: vote-app ❼
template: ❸
  ref: vote-app
---
apiVersion: v1
kind: Secret ❾
metadata:
  name: github-secret
type: Opaque
stringData:
  secretToken: "1234567"

```

- ❶ The API version of the **Trigger** resource. In this example, **v1beta1**.
 - ❷ Specifies the type of Kubernetes object. In this example, **Trigger**.
 - ❸ Unique name to identify the **Trigger** resource.
 - ❹ Service account name to be used.
 - ❺ Interceptor name to be referenced. In this example, **github**.
 - ❻ Desired parameters to be specified.
 - ❼ Name of the **TriggerBinding** resource to be connected to the **TriggerTemplate** resource.
 - ❸ Name of the **TriggerTemplate** resource to be connected to the **TriggerBinding** resource.
 - ❾ Secret to be used to verify events.
- The **EventListener** resource provides an endpoint, or an event sink, that listens for incoming HTTP-based events with a JSON payload. It extracts event parameters from each **TriggerBinding** resource, and then processes this data to create Kubernetes resources as specified by the corresponding **TriggerTemplate** resource. The **EventListener** resource also performs lightweight event processing or basic filtering on the payload using event **interceptors**, which identify the type of payload and optionally modify it. Currently, pipeline triggers support five types of interceptors: *Webhook Interceptors*, *GitHub Interceptors*, *GitLab Interceptors*, *Bitbucket Interceptors*, and *Common Expression Language (CEL) Interceptors*. The following example shows an **EventListener** resource, which references the **Trigger** resource named **vote-trigger**.

```
apiVersion: triggers.tekton.dev/v1beta1 ❶
```

```

kind: EventListener 2
metadata:
  name: vote-app 3
spec:
  serviceAccountName: pipeline 4
  triggers:
    - triggerRef: vote-trigger 5

```

- 1** The API version of the **EventListener** resource. In this example, **v1beta1**.
- 2** Specifies the type of Kubernetes object. In this example, **EventListener**.
- 3** Unique name to identify the **EventListener** resource.
- 4** Service account name to be used.
- 5** Name of the **Trigger** resource referenced by the **EventListener** resource.

3.2.3. Additional resources

- For information on installing pipelines, see [Installing OpenShift Pipelines](#).
- For more details on creating custom CI/CD solutions, see [Creating applications with CI/CD Pipelines](#).
- For more details on re-encrypt TLS termination, see [Re-encryption Termination](#).
- For more details on secured routes, see the [Secured routes](#) section.

3.3. INSTALLING OPENSIFT PIPELINES

This guide walks cluster administrators through the process of installing the Red Hat OpenShift Pipelines Operator to an OpenShift Container Platform cluster.

Prerequisites

- You have access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.
- You have installed **oc** CLI.
- You have installed [OpenShift Pipelines \(tkn\) CLI](#) on your local system.
- Your cluster has the [Marketplace capability](#) enabled or the Red Hat Operator catalog source configured manually.

3.3.1. Installing the Red Hat OpenShift Pipelines Operator in web console

You can install Red Hat OpenShift Pipelines using the Operator listed in the OpenShift Container Platform OperatorHub. When you install the Red Hat OpenShift Pipelines Operator, the custom resources (CRs) required for the pipelines configuration are automatically installed along with the Operator.

The default Operator custom resource definition (CRD) **config.operator.tekton.dev** is now replaced by

tektonconfigs.operator.tekton.dev. In addition, the Operator provides the following additional CRDs to individually manage OpenShift Pipelines components: **tektonpipelines.operator.tekton.dev**, **tektontriggers.operator.tekton.dev** and **tektonaddons.operator.tekton.dev**.

If you have OpenShift Pipelines already installed on your cluster, the existing installation is seamlessly upgraded. The Operator will replace the instance of **config.operator.tekton.dev** on your cluster with an instance of **tektonconfigs.operator.tekton.dev** and additional objects of the other CRDs as necessary.



WARNING

If you manually changed your existing installation, such as, changing the target namespace in the **config.operator.tekton.dev** CRD instance by making changes to the **resource name - cluster** field, then the upgrade path is not smooth. In such cases, the recommended workflow is to uninstall your installation and reinstall the Red Hat OpenShift Pipelines Operator.

The Red Hat OpenShift Pipelines Operator now provides the option to choose the components that you want to install by specifying profiles as part of the **TektonConfig** CR. The **TektonConfig** CR is automatically installed when the Operator is installed. The supported profiles are:

- Lite: This installs only Tekton Pipelines.
- Basic: This installs Tekton Pipelines and Tekton Triggers.
- All: This is the default profile used when the **TektonConfig** CR is installed. This profile installs all of the Tekton components: Tekton Pipelines, Tekton Triggers, Tekton Addons (which include **ClusterTasks**, **ClusterTriggerBindings**, **ConsoleCLIDownload**, **ConsoleQuickStart** and **ConsoleYAMLSample** resources).

Procedure

1. In the **Administrator** perspective of the web console, navigate to **Operators → OperatorHub**.
2. Use the **Filter by keyword** box to search for **Red Hat OpenShift Pipelines** Operator in the catalog. Click the **Red Hat OpenShift Pipelines** Operator tile.
3. Read the brief description about the Operator on the **Red Hat OpenShift Pipelines** Operator page. Click **Install**.
4. On the **Install Operator** page:
 - a. Select **All namespaces on the cluster (default)** for the **Installation Mode**. This mode installs the Operator in the default **openshift-operators** namespace, which enables the Operator to watch and be made available to all namespaces in the cluster.
 - b. Select **Automatic** for the **Approval Strategy**. This ensures that the future upgrades to the Operator are handled automatically by the Operator Lifecycle Manager (OLM). If you select the **Manual** approval strategy, OLM creates an update request. As a cluster administrator, you must then manually approve the OLM update request to update the Operator to the new version.

c. Select an **Update Channel**.

- The **latest** channel enables installation of the most recent stable version of the Red Hat OpenShift Pipelines Operator. Currently, it is the default channel for installing the Red Hat OpenShift Pipelines Operator.
- To install a specific version of the Red Hat OpenShift Pipelines Operator, cluster administrators can use the corresponding **pipelines-<version>** channel. For example, to install the Red Hat OpenShift Pipelines Operator version **1.8.x**, you can use the **pipelines-1.8** channel.

**NOTE**

Starting with OpenShift Container Platform 4.11, the **preview** and **stable** channels for installing and upgrading the Red Hat OpenShift Pipelines Operator are not available. However, in OpenShift Container Platform 4.10 and earlier versions, you can use the **preview** and **stable** channels for installing and upgrading the Operator.

5. Click **Install**. You will see the Operator listed on the **Installed Operators** page.

**NOTE**

The Operator is installed automatically into the **openshift-operators** namespace.

6. Verify that the **Status** is set to **Succeeded Up to date** to confirm successful installation of Red Hat OpenShift Pipelines Operator.

3.3.2. Installing the OpenShift Pipelines Operator using the CLI

You can install Red Hat OpenShift Pipelines Operator from the OperatorHub using the CLI.

Procedure

1. Create a Subscription object YAML file to subscribe a namespace to the Red Hat OpenShift Pipelines Operator, for example, **sub.yaml**:

Example Subscription

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-pipelines-operator
  namespace: openshift-operators
spec:
  channel: <channel name> ❶
  name: openshift-pipelines-operator-rh ❷
  source: redhat-operators ❸
  sourceNamespace: openshift-marketplace ❹
```

- ❶ The channel name of the Operator. The **pipelines-<version>** channel is the default channel. For example, the default channel for Red Hat OpenShift Pipelines Operator version **1.7** is **pipelines-1.7**. The **latest** channel enables installation of the most recent

stable version of the Red Hat OpenShift Pipelines Operator.

- 2 Name of the Operator to subscribe to.
- 3 Name of the CatalogSource that provides the Operator.
- 4 Namespace of the CatalogSource. Use **openshift-marketplace** for the default OperatorHub CatalogSources.

2. Create the Subscription object:

```
$ oc apply -f sub.yaml
```

The Red Hat OpenShift Pipelines Operator is now installed in the default target namespace **openshift-operators**.

3.3.3. Red Hat OpenShift Pipelines Operator in a restricted environment

The Red Hat OpenShift Pipelines Operator enables support for installation of pipelines in a restricted network environment.

The Operator installs a proxy webhook that sets the proxy environment variables in the containers of the pod created by tekton-controllers based on the **cluster** proxy object. It also sets the proxy environment variables in the **TektonPipelines, TektonTriggers, Controllers, Webhooks, and Operator Proxy Webhook** resources.

By default, the proxy webhook is disabled for the **openshift-pipelines** namespace. To disable it for any other namespace, you can add the **operator.tekton.dev/disable-proxy: true** label to the **namespace** object.

3.3.4. Disabling the automatic creation of RBAC resources

The default installation of the Red Hat OpenShift Pipelines Operator creates multiple role-based access control (RBAC) resources for all namespaces in the cluster, except the namespaces matching the **^(openshift|kube)-*** regular expression pattern. Among these RBAC resources, the **pipelines-scc-rolebinding** security context constraint (SCC) role binding resource is a potential security issue, because the associated **pipelines-scc** SCC has the **RunAsAny** privilege.

To disable the automatic creation of cluster-wide RBAC resources after the Red Hat OpenShift Pipelines Operator is installed, cluster administrators can set the **createRbacResource** parameter to **false** in the cluster-level **TektonConfig** custom resource (CR).

Example TektonConfig CR

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  params:
    - name: createRbacResource
      value: "false"
  profile: all
  targetNamespace: openshift-pipelines
```

```
addon:
  params:
    - name: clusterTasks
      value: "true"
    - name: pipelineTemplates
      value: "true"
  ...
```



WARNING

As a cluster administrator or an user with appropriate privileges, when you disable the automatic creation of RBAC resources for all namespaces, the default **ClusterTask** resource does not work. For the **ClusterTask** resource to function, you must create the RBAC resources manually for each intended namespace.

3.3.5. Additional resources

- You can learn more about installing Operators on OpenShift Container Platform in the [adding Operators to a cluster](#) section.
- To install Tekton Chains using the Red Hat OpenShift Pipelines Operator, see [Using Tekton Chains for Red Hat OpenShift Pipelines supply chain security](#).
- To install and deploy in-cluster Tekton Hub, see [Using Tekton Hub with Red Hat OpenShift Pipelines](#).
- For more information on using pipelines in a restricted environment, see:
 - [Mirroring images to run pipelines in a restricted environment](#)
 - [Configuring Samples Operator for a restricted cluster](#)
 - [Creating a cluster with a mirrored registry](#)

3.4. UNINSTALLING OPENSIFT PIPELINES

Cluster administrators can uninstall the Red Hat OpenShift Pipelines Operator by performing the following steps:

1. Delete the Custom Resources (CRs) that were added by default when you installed the Red Hat OpenShift Pipelines Operator.
2. Delete the CRs of the optional components, such as Tekton Chains, that are dependent on the Operator.

CAUTION

If you uninstall the Operator without removing the CRs of optional components, you cannot remove them later.

3. Uninstall the Red Hat OpenShift Pipelines Operator.

Uninstalling only the Operator will not remove the Red Hat OpenShift Pipelines components created by default when the Operator is installed.

3.4.1. Deleting the Red Hat OpenShift Pipelines components and Custom Resources

Delete the Custom Resources (CRs) created by default during installation of the Red Hat OpenShift Pipelines Operator.

Procedure

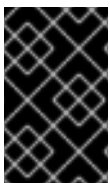
1. In the **Administrator** perspective of the web console, navigate to **Administration → Custom Resource Definition**.
2. Type **config.operator.tekton.dev** in the **Filter by name** box to search for the Red Hat OpenShift Pipelines Operator CRs.
3. Click **CRD Config** to see the **Custom Resource Definition Details** page.
4. Click the **Actions** drop-down menu and select **Delete Custom Resource Definition**.



NOTE

Deleting the CRs will delete the Red Hat OpenShift Pipelines components, and all the Tasks and Pipelines on the cluster will be lost.

5. Click **Delete** to confirm the deletion of the CRs.



IMPORTANT

Repeat the procedure to find and remove CRs of optional components such as Tekton Chains, before uninstalling the Operator. If you uninstall the Operator without removing the CRs of optional components, you cannot remove them later.

3.4.2. Uninstalling the Red Hat OpenShift Pipelines Operator

Procedure

1. From the **Operators → OperatorHub** page, use the **Filter by keyword** box to search for **Red Hat OpenShift Pipelines Operator**.
2. Click the **OpenShift Pipelines Operator** tile. The Operator tile indicates it is installed.
3. In the **OpenShift Pipelines Operator** descriptor page, click **Uninstall**.

Additional resources

- You can learn more about uninstalling Operators on OpenShift Container Platform in the [deleting Operators from a cluster](#) section.

3.5. CREATING CI/CD SOLUTIONS FOR APPLICATIONS USING OPENSHIFT PIPELINES

With Red Hat OpenShift Pipelines, you can create a customized CI/CD solution to build, test, and deploy your application.

To create a full-fledged, self-serving CI/CD pipeline for an application, perform the following tasks:

- Create custom tasks, or install existing reusable tasks.
- Create and define the delivery pipeline for your application.
- Provide a storage volume or filesystem that is attached to a workspace for the pipeline execution, using one of the following approaches:
 - Specify a volume claim template that creates a persistent volume claim
 - Specify a persistent volume claim
- Create a **PipelineRun** object to instantiate and invoke the pipeline.
- Add triggers to capture events in the source repository.

This section uses the **pipelines-tutorial** example to demonstrate the preceding tasks. The example uses a simple application which consists of:

- A front-end interface, **pipelines-vote-ui**, with the source code in the [pipelines-vote-ui](#) Git repository.
- A back-end interface, **pipelines-vote-api**, with the source code in the [pipelines-vote-api](#) Git repository.
- The **apply-manifests** and **update-deployment** tasks in the [pipelines-tutorial](#) Git repository.

3.5.1. Prerequisites

- You have access to an OpenShift Container Platform cluster.
- You have installed [OpenShift Pipelines](#) using the Red Hat OpenShift Pipelines Operator listed in the OpenShift OperatorHub. After it is installed, it is applicable to the entire cluster.
- You have installed [OpenShift Pipelines CLI](#).
- You have forked the front-end [pipelines-vote-ui](#) and back-end [pipelines-vote-api](#) Git repositories using your GitHub ID, and have administrator access to these repositories.
- Optional: You have cloned the [pipelines-tutorial](#) Git repository.

3.5.2. Creating a project and checking your pipeline service account

Procedure

1. Log in to your OpenShift Container Platform cluster:

```
$ oc login -u <login> -p <password> https://openshift.example.com:6443
```

2. Create a project for the sample application. For this example workflow, create the **pipelines-tutorial** project:

```
$ oc new-project pipelines-tutorial
```

**NOTE**

If you create a project with a different name, be sure to update the resource URLs used in the example with your project name.

3. View the **pipeline** service account:

Red Hat OpenShift Pipelines Operator adds and configures a service account named **pipeline** that has sufficient permissions to build and push an image. This service account is used by the **PipelineRun** object.

```
$ oc get serviceaccount pipeline
```

3.5.3. Creating pipeline tasks

Procedure

1. Install the **apply-manifests** and **update-deployment** task resources from the **pipelines-tutorial** repository, which contains a list of reusable tasks for pipelines:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.8/01_pipeline/01_apply_manifest_task.yaml
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.8/01_pipeline/02_update_deployment_task.yaml
```

2. Use the **tkn task list** command to list the tasks you created:

```
$ tkn task list
```

The output verifies that the **apply-manifests** and **update-deployment** task resources were created:

NAME	DESCRIPTION	AGE
apply-manifests		1 minute ago
update-deployment		48 seconds ago

3. Use the **tkn clustertasks list** command to list the Operator-installed additional cluster tasks such as **buildah** and **s2i-python**:

**NOTE**

To use the **buildah** cluster task in a restricted environment, you must ensure that the Dockerfile uses an internal image stream as the base image.

```
$ tkn clustertasks list
```

The output lists the Operator-installed **ClusterTask** resources:

NAME	DESCRIPTION	AGE
------	-------------	-----

buildah	1 day ago
git-clone	1 day ago
s2i-python	1 day ago
tkn	1 day ago

Additional resources

- [Managing non-versioned and versioned cluster tasks](#)

3.5.4. Assembling a pipeline

A pipeline represents a CI/CD flow and is defined by the tasks to be executed. It is designed to be generic and reusable in multiple applications and environments.

A pipeline specifies how the tasks interact with each other and their order of execution using the **from** and **runAfter** parameters. It uses the **workspaces** field to specify one or more volumes that each task in the pipeline requires during execution.

In this section, you will create a pipeline that takes the source code of the application from GitHub, and then builds and deploys it on OpenShift Container Platform.

The pipeline performs the following tasks for the back-end application **pipelines-vote-api** and front-end application **pipelines-vote-ui**:

- Clones the source code of the application from the Git repository by referring to the **git-url** and **git-revision** parameters.
- Builds the container image using the **buildah** cluster task.
- Pushes the image to the internal image registry by referring to the **image** parameter.
- Deploys the new image on OpenShift Container Platform by using the **apply-manifests** and **update-deployment** tasks.

Procedure

1. Copy the contents of the following sample pipeline YAML file and save it:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  workspaces:
    - name: shared-workspace
  params:
    - name: deployment-name
      type: string
      description: name of the deployment to be patched
    - name: git-url
      type: string
      description: url of the git repo for the code of deployment
    - name: git-revision
      type: string
      description: revision to be used from repo of the code for deployment
```

```

    default: "pipelines-1.8"
  - name: IMAGE
    type: string
    description: image to be built from the code
  tasks:
  - name: fetch-repository
    taskRef:
      name: git-clone
      kind: ClusterTask
    workspaces:
    - name: output
      workspace: shared-workspace
    params:
    - name: url
      value: $(params.git-url)
    - name: subdirectory
      value: ""
    - name: deleteExisting
      value: "true"
    - name: revision
      value: $(params.git-revision)
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
    - name: IMAGE
      value: $(params.IMAGE)
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter:
    - fetch-repository
  - name: apply-manifests
    taskRef:
      name: apply-manifests
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter:
    - build-image
  - name: update-deployment
    taskRef:
      name: update-deployment
    params:
    - name: deployment
      value: $(params.deployment-name)
    - name: IMAGE
      value: $(params.IMAGE)
    runAfter:
    - apply-manifests

```

The pipeline definition abstracts away the specifics of the Git source repository and image registries. These details are added as **params** when a pipeline is triggered and executed.

2. Create the pipeline:

```
$ oc create -f <pipeline-yaml-file-name.yaml>
```

Alternatively, you can also execute the YAML file directly from the Git repository:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.8/01_pipeline/04_pipeline.yaml
```

3. Use the **tkn pipeline list** command to verify that the pipeline is added to the application:

```
$ tkn pipeline list
```

The output verifies that the **build-and-deploy** pipeline was created:

NAME	AGE	LAST RUN	STARTED	DURATION	STATUS
build-and-deploy	1 minute ago	---	---	---	---

3.5.5. Mirroring images to run pipelines in a restricted environment

To run OpenShift Pipelines in a disconnected cluster or a cluster provisioned in a restricted environment, ensure that either the Samples Operator is configured for a restricted network, or a cluster administrator has created a cluster with a mirrored registry.

The following procedure uses the **pipelines-tutorial** example to create a pipeline for an application in a restricted environment using a cluster with a mirrored registry. To ensure that the **pipelines-tutorial** example works in a restricted environment, you must mirror the respective builder images from the mirror registry for the front-end interface, **pipelines-vote-ui**; back-end interface, **pipelines-vote-api**; and the **cli**.

Procedure

1. Mirror the builder image from the mirror registry for the front-end interface, **pipelines-vote-ui**.
 - a. Verify that the required images tag is not imported:

```
$ oc describe imagestream python -n openshift
```

Example output

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

```
3.8-ubi8 (latest)
  tagged from registry.redhat.io/ubi8/python-38:latest
  prefer registry pullthrough when referencing this tag
```

```
Build and run Python 3.8 applications on UBI 8. For more information about using this
builder image, including OpenShift considerations, see https://github.com/sclorg/s2i-
python-container/blob/master/3.8/README.md.
```

```
Tags: builder, python
Supports: python:3.8, python
```

Example Repo: <https://github.com/sclorg/django-ex.git>

[...]

- b. Mirror the supported image tag to the private registry:

```
$ oc image mirror registry.redhat.io/ubi8/python-38:latest <mirror-registry>:
<port>/ubi8/python-38
```

- c. Import the image:

```
$ oc tag <mirror-registry>:<port>/ubi8/python-38 python:latest --scheduled -n openshift
```

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

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

```
$ oc describe imagestream python -n openshift
```

Example output

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

latest
updates automatically from registry <mirror-registry>:<port>/ubi8/python-38

* <mirror-registry>:<port>/ubi8/python-38@sha256:3ee3c2e70251e75bfeac25c0c33356add9cc4abcbc9c51d858f39e4dc29c5f58
[...]

[...]
```

2. Mirror the builder image from the mirror registry for the back-end interface, **pipelines-vote-api**.

- a. Verify that the required images tag is not imported:

```
$ oc describe imagestream golang -n openshift
```

Example output

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

1.14.7-ubi8 (latest)
tagged from registry.redhat.io/ubi8/go-toolset:1.14.7
prefer registry pullthrough when referencing this tag

Build and run Go applications on UBI 8. For more information about using this builder image, including OpenShift considerations, see https://github.com/sclorg/golang-container/blob/master/README.md.
```

```
Tags: builder, golang, go
Supports: golang
Example Repo: https://github.com/sclorg/golang-ex.git
```

```
[...]
```

- b. Mirror the supported image tag to the private registry:

```
$ oc image mirror registry.redhat.io/ubi8/go-toolset:1.14.7 <mirror-registry>:
<port>/ubi8/go-toolset
```

- c. Import the image:

```
$ oc tag <mirror-registry>:<port>/ubi8/go-toolset golang:latest --scheduled -n openshift
```

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

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

```
$ oc describe imagestream golang -n openshift
```

Example output

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

latest
updates automatically from registry <mirror-registry>:<port>/ubi8/go-toolset

* <mirror-registry>:<port>/ubi8/go-toolset@sha256:59a74d581df3a2bd63ab55f7ac106677694bf612a1fe9e7e3e1487f55c421b37
[...]

[...]
```

3. Mirror the builder image from the mirror registry for the **cli**.

- a. Verify that the required images tag is not imported:

```
$ oc describe imagestream cli -n openshift
```

Example output

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

latest
updates automatically from registry quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551
```

```
* quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551

[...]
```

- b. Mirror the supported image tag to the private registry:

```
$ oc image mirror quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551
<mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-dev:latest
```

- c. Import the image:

```
$ oc tag <mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-dev cli:latest --
scheduled -n openshift
```

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

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

```
$ oc describe imagestream cli -n openshift
```

Example output

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

latest
  updates automatically from registry <mirror-registry>:<port>/openshift-release-dev/ocp-
v4.0-art-dev

  * <mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551

[...]
```

Additional resources

- [Configuring Samples Operator for a restricted cluster](#)
- [Creating a cluster with a mirrored registry](#)

3.5.6. Running a pipeline

A **PipelineRun** resource starts a pipeline and ties it to the Git and image resources that should be used for the specific invocation. It automatically creates and starts the **TaskRun** resources for each task in the pipeline.

Procedure

1. Start the pipeline for the back-end application:

```
$ tkn pipeline start build-and-deploy \
  -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.8/01_pipeline/03_persistent_volume_claim.yaml \
  -p deployment-name=pipelines-vote-api \
  -p git-url=https://github.com/openshift/pipelines-vote-api.git \
  -p IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/pipelines-
vote-api \
  --use-param-defaults
```

The previous command uses a volume claim template, which creates a persistent volume claim for the pipeline execution.

2. To track the progress of the pipeline run, enter the following command::

```
$ tkn pipelinerun logs <pipelinerun_id> -f
```

The <pipelinerun_id> in the above command is the ID for the **PipelineRun** that was returned in the output of the previous command.

3. Start the pipeline for the front-end application:

```
$ tkn pipeline start build-and-deploy \
  -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.8/01_pipeline/03_persistent_volume_claim.yaml \
  -p deployment-name=pipelines-vote-ui \
  -p git-url=https://github.com/openshift/pipelines-vote-ui.git \
  -p IMAGE=image-registry.openshift-image-
registry.svc:5000/$(context.pipelineRun.namespace)/pipelines-vote-ui \
  --use-param-defaults
```

4. To track the progress of the pipeline run, enter the following command:

```
$ tkn pipelinerun logs <pipelinerun_id> -f
```

The <pipelinerun_id> in the above command is the ID for the **PipelineRun** that was returned in the output of the previous command.

5. After a few minutes, use **tkn pipelinerun list** command to verify that the pipeline ran successfully by listing all the pipeline runs:

```
$ tkn pipelinerun list
```

The output lists the pipeline runs:

NAME	STARTED	DURATION	STATUS
build-and-deploy-run-xy7rw	1 hour ago	2 minutes	Succeeded
build-and-deploy-run-z2rz8	1 hour ago	19 minutes	Succeeded

6. Get the application route:

```
$ oc get route pipelines-vote-ui --template='http://{{.spec.host}}'
```

Note the output of the previous command. You can access the application using this route.

7. To rerun the last pipeline run, using the pipeline resources and service account of the previous pipeline, run:

```
$ tkn pipeline start build-and-deploy --last
```

Additional resources

- [Authenticating pipelines using git secret](#)

3.5.7. Adding triggers to a pipeline

Triggers enable pipelines to respond to external GitHub events, such as push events and pull requests. After you assemble and start a pipeline for the application, add the **TriggerBinding**, **TriggerTemplate**, **Trigger**, and **EventListener** resources to capture the GitHub events.

Procedure

1. Copy the content of the following sample **TriggerBinding** YAML file and save it:

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerBinding
metadata:
  name: vote-app
spec:
  params:
    - name: git-repo-url
      value: $(body.repository.url)
    - name: git-repo-name
      value: $(body.repository.name)
    - name: git-revision
      value: $(body.head_commit.id)
```

2. Create the **TriggerBinding** resource:

```
$ oc create -f <triggerbinding-yaml-file-name.yaml>
```

Alternatively, you can create the **TriggerBinding** resource directly from the **pipelines-tutorial** Git repository:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.8/03_triggers/01_binding.yaml
```

3. Copy the content of the following sample **TriggerTemplate** YAML file and save it:

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerTemplate
metadata:
  name: vote-app
spec:
```

```

params:
- name: git-repo-url
  description: The git repository url
- name: git-revision
  description: The git revision
  default: pipelines-1.8
- name: git-repo-name
  description: The name of the deployment to be created / patched

resourcetemplates:
- apiVersion: tekton.dev/v1beta1
  kind: PipelineRun
  metadata:
    generateName: build-deploy-$(tt.params.git-repo-name)-
  spec:
    serviceAccountName: pipeline
    pipelineRef:
      name: build-and-deploy
    params:
      - name: deployment-name
        value: $(tt.params.git-repo-name)
      - name: git-url
        value: $(tt.params.git-repo-url)
      - name: git-revision
        value: $(tt.params.git-revision)
      - name: IMAGE
        value: image-registry.openshift-image-
registry.svc:5000/$(context.pipelineRun.namespace)/$(tt.params.git-repo-name)
    workspaces:
      - name: shared-workspace
        volumeClaimTemplate:
          spec:
            accessModes:
              - ReadWriteOnce
            resources:
              requests:
                storage: 500Mi

```

The template specifies a volume claim template to create a persistent volume claim for defining the storage volume for the workspace. Therefore, you do not need to create a persistent volume claim to provide data storage.

4. Create the **TriggerTemplate** resource:

```
$ oc create -f <triggertemplate-yaml-file-name.yaml>
```

Alternatively, you can create the **TriggerTemplate** resource directly from the **pipelines-tutorial** Git repository:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.8/03_triggers/02_template.yaml
```

5. Copy the contents of the following sample **Trigger** YAML file and save it:

```
apiVersion: triggers.tekton.dev/v1beta1
```

```
kind: Trigger
metadata:
  name: vote-trigger
spec:
  serviceAccountName: pipeline
  bindings:
    - ref: vote-app
  template:
    ref: vote-app
```

6. Create the **Trigger** resource:

```
$ oc create -f <trigger-yaml-file-name.yaml>
```

Alternatively, you can create the **Trigger** resource directly from the **pipelines-tutorial** Git repository:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.8/03_triggers/03_trigger.yaml
```

7. Copy the contents of the following sample **EventListener** YAML file and save it:

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: vote-app
spec:
  serviceAccountName: pipeline
  triggers:
    - triggerRef: vote-trigger
```

Alternatively, if you have not defined a trigger custom resource, add the binding and template spec to the **EventListener** YAML file, instead of referring to the name of the trigger:

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: vote-app
spec:
  serviceAccountName: pipeline
  triggers:
    - bindings:
        - ref: vote-app
      template:
        ref: vote-app
```

8. Create the **EventListener** resource by performing the following steps:

- To create an **EventListener** resource using a secure HTTPS connection:
 - a. Add a label to enable the secure HTTPS connection to the **EventListener** resource:

```
$ oc label namespace <ns-name> operator.tekton.dev/enable-annotation=enabled
```

- b. Create the **EventListener** resource:

```
$ oc create -f <eventlistener-yaml-file-name.yaml>
```

Alternatively, you can create the **EventListener** resource directly from the **pipelines-tutorial** Git repository:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.8/03_triggers/04_event_listener.yaml
```

- c. Create a route with the re-encrypt TLS termination:

```
$ oc create route reencrypt --service=<svc-name> --cert=tls.crt --key=tls.key --ca-cert=ca.crt --hostname=<hostname>
```

Alternatively, you can create a re-encrypt TLS termination YAML file to create a secured route.

Example Re-encrypt TLS Termination YAML of the Secured Route

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured ❶
spec:
  host: <hostname>
  to:
    kind: Service
    name: frontend ❷
  tls:
    termination: reencrypt ❸
    key: [as in edge termination]
    certificate: [as in edge termination]
    caCertificate: [as in edge termination]
    destinationCACertificate: |- ❹
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

- ❶ ❷ The name of the object, which is limited to 63 characters.
- ❸ The **termination** field is set to **reencrypt**. This is the only required **tls** field.
- ❹ Required for re-encryption. **destinationCACertificate** specifies a CA certificate to validate the endpoint certificate, securing the connection from the router to the destination pods. If the service is using a service signing certificate, or the administrator has specified a default CA certificate for the router and the service has a certificate signed by that CA, this field can be omitted.

See **oc create route reencrypt --help** for more options.

- To create an **EventListener** resource using an insecure HTTP connection:
 - a. Create the **EventListener** resource.

- b. Expose the **EventListener** service as an OpenShift Container Platform route to make it publicly accessible:

```
$ oc expose svc el-vote-app
```

3.5.8. Configuring event listeners to serve multiple namespaces



NOTE

You can skip this section if you want to create a basic CI/CD pipeline. However, if your deployment strategy involves multiple namespaces, you can configure event listeners to serve multiple namespaces.

To increase reusability of **EventListener** objects, cluster administrators can configure and deploy them as multi-tenant event listeners that serve multiple namespaces.

Procedure

1. Configure cluster-wide fetch permission for the event listener.
 - a. Set a service account name to be used in the **ClusterRoleBinding** and **EventListener** objects. For example, **el-sa**.

Example ServiceAccount.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: el-sa
---
```

- b. In the **rules** section of the **ClusterRole.yaml** file, set appropriate permissions for every event listener deployment to function cluster-wide.

Example ClusterRole.yaml

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: el-sel-clusterrole
rules:
- apiGroups: ["triggers.tekton.dev"]
  resources: ["eventlisteners", "clustertriggerbindings", "clusterinterceptors",
"triggerbindings", "triggertemplates", "triggers"]
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources: ["configmaps", "secrets"]
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources: ["serviceaccounts"]
  verbs: ["impersonate"]
...
```

- c. Configure cluster role binding with the appropriate service account name and cluster role name.

Example ClusterRoleBinding.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: el-mul-clusterrolebinding
subjects:
- kind: ServiceAccount
  name: el-sa
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: el-sel-clusterrole
...
```

2. In the **spec** parameter of the event listener, add the service account name, for example **el-sa**. Fill the **namespaceSelector** parameter with names of namespaces where event listener is intended to serve.

Example EventListener.yaml

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: namespace-selector-listener
spec:
  serviceAccountName: el-sa
  namespaceSelector:
    matchNames:
    - default
    - foo
...
```

3. Create a service account with the necessary permissions, for example **foo-trigger-sa**. Use it for role binding the triggers.

Example ServiceAccount.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: foo-trigger-sa
  namespace: foo
...
```

Example RoleBinding.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
```

```

name: triggercr-rolebinding
namespace: foo
subjects:
- kind: ServiceAccount
  name: foo-trigger-sa
  namespace: foo
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tekton-triggers-eventlistener-roles
...

```

4. Create a trigger with the appropriate trigger template, trigger binding, and service account name.

Example Trigger.yaml

```

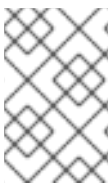
apiVersion: triggers.tekton.dev/v1beta1
kind: Trigger
metadata:
  name: trigger
  namespace: foo
spec:
  serviceAccountName: foo-trigger-sa
  interceptors:
  - ref:
      name: "github"
    params:
    - name: "secretRef"
      value:
        secretName: github-secret
        secretKey: secretToken
    - name: "eventTypes"
      value: ["push"]
  bindings:
  - ref: vote-app
  template:
    ref: vote-app
...

```

3.5.9. Creating webhooks

Webhooks are HTTP POST messages that are received by the event listeners whenever a configured event occurs in your repository. The event payload is then mapped to trigger bindings, and processed by trigger templates. The trigger templates eventually start one or more pipeline runs, leading to the creation and deployment of Kubernetes resources.

In this section, you will configure a webhook URL on your forked Git repositories **pipelines-vote-ui** and **pipelines-vote-api**. This URL points to the publicly accessible **EventListener** service route.



NOTE

Adding webhooks requires administrative privileges to the repository. If you do not have administrative access to your repository, contact your system administrator for adding webhooks.

Procedure

1. Get the webhook URL:

- For a secure HTTPS connection:

```
$ echo "URL: $(oc get route el-vote-app --template='https://{{.spec.host}}')"
```

- For an HTTP (insecure) connection:

```
$ echo "URL: $(oc get route el-vote-app --template='http://{{.spec.host}}')"
```

Note the URL obtained in the output.

2. Configure webhooks manually on the front-end repository:

- a. Open the front-end Git repository **pipelines-vote-ui** in your browser.
- b. Click **Settings** → **Webhooks** → **Add Webhook**
- c. On the **Webhooks/Add Webhook** page:
 - i. Enter the webhook URL from step 1 in **Payload URL** field
 - ii. Select **application/json** for the **Content type**
 - iii. Specify the secret in the **Secret** field
 - iv. Ensure that the **Just the push event** is selected
 - v. Select **Active**
 - vi. Click **Add Webhook**

3. Repeat step 2 for the back-end repository **pipelines-vote-api**.

3.5.10. Triggering a pipeline run

Whenever a **push** event occurs in the Git repository, the configured webhook sends an event payload to the publicly exposed **EventListener** service route. The **EventListener** service of the application processes the payload, and passes it to the relevant **TriggerBinding** and **TriggerTemplate** resource pairs. The **TriggerBinding** resource extracts the parameters, and the **TriggerTemplate** resource uses these parameters and specifies the way the resources must be created. This may rebuild and redeploy the application.

In this section, you push an empty commit to the front-end **pipelines-vote-ui** repository, which then triggers the pipeline run.

Procedure

1. From the terminal, clone your forked Git repository **pipelines-vote-ui**:

```
$ git clone git@github.com:<your GitHub ID>/pipelines-vote-ui.git -b pipelines-1.8
```

2. Push an empty commit:

```
$ git commit -m "empty-commit" --allow-empty && git push origin pipelines-1.8
```

3. Check if the pipeline run was triggered:

```
$ tkn pipelinerun list
```

Notice that a new pipeline run was initiated.

3.5.11. Enabling monitoring of event listeners for Triggers for user-defined projects

As a cluster administrator, to gather event listener metrics for the **Triggers** service in a user-defined project and display them in the OpenShift Container Platform web console, you can create a service monitor for each event listener. On receiving an HTTP request, event listeners for the **Triggers** service return three metrics – **eventlistener_http_duration_seconds**, **eventlistener_event_count**, and **eventlistener_triggered_resources**.

Prerequisites

- You have logged in to the OpenShift Container Platform web console.
- You have installed the Red Hat OpenShift Pipelines Operator.
- You have enabled monitoring for user-defined projects.

Procedure

1. For each event listener, create a service monitor. For example, to view the metrics for the **github-listener** event listener in the **test** namespace, create the following service monitor:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app.kubernetes.io/managed-by: EventListener
    app.kubernetes.io/part-of: Triggers
    eventlistener: github-listener
  annotations:
    networkoperator.openshift.io/ignore-errors: ""
  name: el-monitor
  namespace: test
spec:
  endpoints:
    - interval: 10s
      port: http-metrics
  jobLabel: name
  namespaceSelector:
    matchNames:
      - test
  selector:
    matchLabels:
      app.kubernetes.io/managed-by: EventListener
      app.kubernetes.io/part-of: Triggers
      eventlistener: github-listener
  ...
```

2. Test the service monitor by sending a request to the event listener. For example, push an empty commit:

```
$ git commit -m "empty-commit" --allow-empty && git push origin main
```

3. On the OpenShift Container Platform web console, navigate to **Administrator** → **Observe** → **Metrics**.
4. To view a metric, search by its name. For example, to view the details of the **eventlistener_http_resources** metric for the **github-listener** event listener, search using the **eventlistener_http_resources** keyword.

Additional resources

- [Enabling monitoring for user-defined projects](#)

3.5.12. Additional resources

- To include pipelines as code along with the application source code in the same repository, see [Using Pipelines as code](#).
- For more details on pipelines in the **Developer** perspective, see the [working with pipelines in the Developer perspective](#) section.
- To learn more about Security Context Constraints (SCCs), see the [Managing Security Context Constraints](#) section.
- For more examples of reusable tasks, see the [OpenShift Catalog](#) repository. Additionally, you can also see the Tekton Catalog in the Tekton project.
- To install and deploy a custom instance of Tekton Hub for reusable tasks and pipelines, see [Using Tekton Hub with Red Hat OpenShift Pipelines](#).
- For more details on re-encrypt TLS termination, see [Re-encryption Termination](#).
- For more details on secured routes, see the [Secured routes](#) section.

3.6. MANAGING NON-VERSIONED AND VERSIONED CLUSTER TASKS

As a cluster administrator, installing the Red Hat OpenShift Pipelines Operator creates variants of each default cluster task known as *versioned cluster tasks* (VCT) and *non-versioned cluster tasks* (NVCT). For example, installing the Red Hat OpenShift Pipelines Operator v1.7 creates a **buildah-1-7-0** VCT and a **buildah** NVCT.

Both NVCT and VCT have the same metadata, behavior, and specifications, including **params**, **workspaces**, and **steps**. However, they behave differently when you disable them or upgrade the Operator.

3.6.1. Differences between non-versioned and versioned cluster tasks

Non-versioned and versioned cluster tasks have different naming conventions. And, the Red Hat OpenShift Pipelines Operator upgrades them differently.

Table 3.5. Differences between non-versioned and versioned cluster tasks

	Non-versioned cluster task	Versioned cluster task
Nomenclature	The NVCT only contains the name of the cluster task. For example, the name of the NVCT of Buildah installed with Operator v1.7 is buildah .	The VCT contains the name of the cluster task, followed by the version as a suffix. For example, the name of the VCT of Buildah installed with Operator v1.7 is buildah-1-7-0 .
Upgrade	When you upgrade the Operator, it updates the non-versioned cluster task with the latest changes. The name of the NVCT remains unchanged.	Upgrading the Operator installs the latest version of the VCT and retains the earlier version. The latest version of a VCT corresponds to the upgraded Operator. For example, installing Operator 1.7 installs buildah-1-7-0 and retains buildah-1-6-0 .

3.6.2. Advantages and disadvantages of non-versioned and versioned cluster tasks

Before adopting non-versioned or versioned cluster tasks as a standard in production environments, cluster administrators might consider their advantages and disadvantages.

Table 3.6. Advantages and disadvantages of non-versioned and versioned cluster tasks

Cluster task	Advantages	Disadvantages
Non-versioned cluster task (NVCT)	<ul style="list-style-type: none"> If you prefer deploying pipelines with the latest updates and bug fixes, use the NVCT. Upgrading the Operator upgrades the non-versioned cluster tasks, which consume fewer resources than multiple versioned cluster tasks. 	If you deploy pipelines that use NVCT, they might break after an Operator upgrade if the automatically upgraded cluster tasks are not backward-compatible.

Cluster task	Advantages	Disadvantages
Versioned cluster task (VCT)	<ul style="list-style-type: none"> ● If you prefer stable pipelines in production, use the VCT. ● The earlier version is retained on the cluster even after the later version of a cluster task is installed. You can continue using the earlier cluster tasks. 	<ul style="list-style-type: none"> ● If you continue using an earlier version of a cluster task, you might miss the latest features and critical security updates. ● The earlier versions of cluster tasks that are not operational consume cluster resources. ● * After it is upgraded, the Operator cannot manage the earlier VCT. You can delete the earlier VCT manually by using the oc delete clustertask command, but you cannot restore it.

3.6.3. Disabling non-versioned and versioned cluster tasks

As a cluster administrator, you can disable cluster tasks that the Pipelines Operator installed.

Procedure

1. To delete all non-versioned cluster tasks and latest versioned cluster tasks, edit the **TektonConfig** custom resource definition (CRD) and set the **clusterTasks** parameter in **spec.addon.params** to **false**.

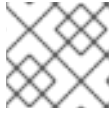
Example TektonConfig CR

```

apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  params:
    - name: createRbacResource
      value: "false"
  profile: all
  targetNamespace: openshift-pipelines
  addon:
    params:
      - name: clusterTasks
        value: "false"
  ...

```

When you disable cluster tasks, the Operator removes all the non-versioned cluster tasks and only the latest version of the versioned cluster tasks from the cluster.

**NOTE**

Re-enabling cluster tasks installs the non-versioned cluster tasks.

2. Optional: To delete earlier versions of the versioned cluster tasks, use any one of the following methods:
 - a. To delete individual earlier versioned cluster tasks, use the **oc delete clustertask** command followed by the versioned cluster task name. For example:

```
$ oc delete clustertask buildah-1-6-0
```

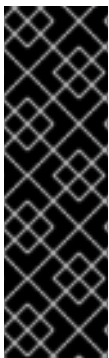
- b. To delete all versioned cluster tasks created by an old version of the Operator, you can delete the corresponding installer set. For example:

```
$ oc delete tektoninstallerset versioned-clustertask-1-6-k98as
```

CAUTION

If you delete an old versioned cluster task, you cannot restore it. You can only restore versioned and non-versioned cluster tasks that the current version of the Operator has created.

3.7. USING TEKTON HUB WITH OPENSIFT PIPELINES

**IMPORTANT**

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

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

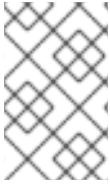
Tekton Hub helps you discover, search, and share reusable tasks and pipelines for your CI/CD workflows. A public instance of Tekton Hub is available at hub.tekton.dev. Cluster administrators can also install and deploy a custom instance of Tekton Hub for enterprise use.

3.7.1. Installing and deploying Tekton Hub on a OpenShift Container Platform cluster

Tekton Hub is an optional component; cluster administrators cannot install it using the **TektonConfig** custom resource (CR). To install and manage Tekton Hub, use the **TektonHub** CR.

You can install Tekton Hub on your cluster using two modes:

- *Without* login authorization and ratings for Tekton Hub artifacts
- *with* login authorization and ratings for Tekton Hub artifacts



NOTE

If you are using Github Enterprise or Gitlab Enterprise, install and deploy Tekton Hub in the same network as the enterprise server. For example, if the enterprise server is running behind a VPN, deploy Tekton Hub on a cluster that is also behind the VPN.

3.7.1.1. Installing Tekton Hub without login and rating

You can install Tekton Hub on your cluster automatically with default configuration. When using the default configuration, Tekton Hub does not support login with authorization and ratings for Tekton Hub artifacts.

Prerequisites

- Ensure that the Red Hat OpenShift Pipelines Operator is installed in the default **openshift-pipelines** namespace on the cluster.

Procedure

1. Create a **TektonHub** CR similar to the following example.

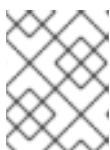
```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonHub
metadata:
  name: hub
spec:
  targetNamespace: openshift-pipelines 1
  api:
    hubConfigUrl: https://raw.githubusercontent.com/tektoncd/hub/main/config.yaml 2
    catalogRefreshInterval: 30m 3
```

- 1** The namespace in which Tekton Hub must be installed; default is **openshift-pipelines**.
- 2** The URL of the **config.yaml** file.
- 3** The time interval after which the catalog refreshes automatically. The supported units of time are seconds (**s**), minutes (**m**), hours (**h**), days (**d**), and weeks (**w**). The default interval is 30 minutes.

2. Apply the **TektonHub** CR.

```
$ oc apply -f <TektonHub>.yaml 1
```

- 1** The file name or path of the **TektonHub** CR.



NOTE

When you apply the **TektonHub** CR, Tekton Hub is installed on the cluster in the **openshift-pipelines** namespace, with upstream Tekton Catalog content.

3. Check the status of the installation. The **TektonHub** CR might take some time to attain steady state.

```
$ oc get tektonhub.operator.tekton.dev
```

Sample output

```
NAME VERSION READY REASON APIURL UIURL
hub v1.8.0 True https://api.route.url/ https://ui.route.url/
```

3.7.1.2. Installing Tekton Hub with login and rating

You can install Tekton Hub on your cluster with custom configuration that supports login with authorization and ratings for Tekton Hub artifacts.

Prerequisites

- Ensure that the Red Hat OpenShift Pipelines Operator is installed in the default **openshift-pipelines** namespace on the cluster.

Procedure

- Create a fork of the [Tekton Hub](#) repository.
- Clone the forked repository.
- Create an OAuth application with your Git repository hosting provider, and note the Client ID and Client Secret. The supported providers are GitHub, GitLab, and BitBucket.
 - For a [GitHub OAuth application](#), set the Homepage URL and the Authorization callback URL as **<auth-route>**.
 - For a [GitLab OAuth application](#), set the **REDIRECT_URI** as **<auth-route>/auth/gitlab/callback**.
 - For a [BitBucket OAuth application](#), set the **Callback URL** as **<auth-route>**.
- Edit the **<tekton_hub_repository>/config/02-api/20-api-secret.yaml** file of your cloned repository to include the Tekton Hub API secrets:

```
apiVersion: v1
kind: Secret
metadata:
  name: tekton-hub-api
  namespace: openshift-pipelines
type: Opaque
stringData:
  GH_CLIENT_ID: 1
  GH_CLIENT_SECRET: 2
  GL_CLIENT_ID: 3
  GL_CLIENT_SECRET: 4
  BB_CLIENT_ID: 5
  BB_CLIENT_SECRET: 6
  JWT_SIGNING_KEY: 7
  ACCESS_JWT_EXPIRES_IN: 8
  REFRESH_JWT_EXPIRES_IN: 9
```


AUTH_BASE_URL: **10**

GHE_URL: **11**

GLE_URL: **12**

- 1** The Client ID from the GitHub OAuth application.
- 2** The Client Secret from the GitHub OAuth application.
- 3** The Client ID from the GitLab OAuth application.
- 4** The Client Secret from the GitLab OAuth application.
- 5** The Client ID from the BitBucket OAuth application.
- 6** The Client Secret from the BitBucket OAuth application.
- 7** A long, random string used to sign the JSON Web Token (JWT) created for users.
- 8** Add the time limit after which the access token expires. For example, **1m**, where **m** denotes minutes. The supported units of time are seconds (**s**), minutes (**m**), hours (**h**), days (**d**), and weeks (**w**).
- 9** Add the time limit after which the refresh token expires. For example, **1m**, where **m** denotes minutes. The supported units of time are seconds (**s**), minutes (**m**), hours (**h**), days (**d**), and weeks (**w**). Ensure that the expiry time set for token refresh is greater than the expiry time set for token access.
- 10** Route URL for the OAuth application.
- 11** GitHub Enterprise URL, if you are authenticating using GitHub Enterprise. Do not provide the URL to the catalog as a value for this field.
- 12** GitLab Enterprise URL, if you are authenticating using GitLab Enterprise. Do not provide the URL to the catalog as a value for this field.



NOTE

You can delete the unused fields for the Git repository hosting service providers that are irrelevant to your deployment.

5. Create a **TektonHub** CR similar to the following example.

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonHub
metadata:
  name: hub
spec:
  targetNamespace: openshift-pipelines 1
  api:
    hubConfigUrl: https://raw.githubusercontent.com/tektoncd/hub/main/config.yaml 2
    catalogRefreshInterval: 30m 3
```

- 1** The namespace in which Tekton Hub must be installed; default is **openshift-pipelines**.

- 2 Substitute with the URL of the **config.yaml** file.
- 3 The time interval after which the catalog refreshes automatically. The supported units of time are seconds (**s**), minutes (**m**), hours (**h**), days (**d**), and weeks (**w**). The default interval is 30 minutes.

6. Apply the **TektonHub** CR.

```
$ oc apply -f <TektonHub>.yaml 1
```

- 1 The file name or path of the **TektonHub** CR.



NOTE

When you apply the **TektonHub** CR, Tekton Hub is installed on the cluster in the **openshift-pipelines** namespace, with upstream Tekton Catalog content.

7. Check the status of the installation. The **TektonHub** CR might take some time to attain steady state.

```
$ oc get tektonhub.operator.tekton.dev
NAME VERSION READY REASON APIURL UIURL
hub v1.8.0 True https://api.route.url/ https://ui.route.url/
```

3.7.2. Optional: Adding new users in Tekton Hub configuration

Procedure

1. Depending on the intended scope, cluster administrators can add new users in the **config.yaml** file.

```
...
scopes:
  - name: agent:create
    users: [<username_1>, <username_2>] 1
  - name: catalog:refresh
    users: [<username_3>, <username_4>]
  - name: config:refresh
    users: [<username_5>, <username_6>]

default:
  scopes:
    - rating:read
    - rating:write
...
```

- 1 The usernames registered with the Git repository hosting service provider.

**NOTE**

When any user logs in for the first time, they will have only the default scope even if they are added in the **config.yaml**. To activate additional scopes, ensure the user has logged in at least once.

2. Ensure that in the **config.yaml** file, you have the **config-refresh** scope.
3. Refresh the configuration.

```
$ curl -X POST -H "Authorization: <access-token>" \ ❶
--header "Content-Type: application/json" \
--data '{"force": true}' \
<api-route>/system/config/refresh
```

❶ The JWT token.

3.7.3. Optional: Using a custom database in Tekton Hub

Cluster administrators can use a custom database with Tekton Hub, instead of the default PostgreSQL database installed by the Operator. You can associate a custom database at the time of installation, and use it with the **db-migration**, **api**, and **ui** interfaces provided by Tekton Hub. Alternatively, you can associate a custom database with Tekton Hub even after the installation with the default database is complete.

Procedure

1. Create a secret named **tekton-hub-db** in the target namespace with the following keys:

- **POSTGRES_HOST**
- **POSTGRES_DB**
- **POSTGRES_USER**
- **POSTGRES_PASSWORD**
- **POSTGRES_PORT**

Example: Custom database secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: tekton-hub-db
labels:
  app: tekton-hub-db
type: Opaque
stringData:
  POSTGRES_HOST: <The name of the host of the database>
  POSTGRES_DB: <Name of the database>
  POSTGRES_USER: <The name of user account>
```

```
POSTGRES_PASSWORD: <The password of user account>
POSTGRES_PORT: <The port that the database is listening on>
```

...

**NOTE**

The default target namespace is **openshift-pipelines**.

2. In the **TektonHub** CR, set the value of the database secret attribute to **tekton-hub-db**.

Example: Adding custom database secret

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonHub
metadata:
  name: hub
spec:
  targetNamespace: openshift-pipelines
  db:
    secret: tekton-hub-db
  api:
    hubConfigUrl: https://raw.githubusercontent.com/tektoncd/hub/main/config.yaml
    catalogRefreshInterval: 30m
...
```

3. Use the updated **TektonHub** CR to associate the custom database with Tekton Hub.
 - a. If you are associating the custom database at the time of installing Tekton Hub on your cluster, apply the updated **TektonHub** CR.

```
$ oc apply -f <tekton-hub-cr>.yaml
```

- b. Alternatively, if you are associating the custom database after the installation of Tekton Hub is complete, replace the existing **TektonHub** CR with the updated **TektonHub** CR.

```
$ oc replace -f <tekton-hub-cr>.yaml
```

4. Check the status of the installation. The **TektonHub** CR might take some time to attain steady state.

```
$ oc get tektonhub.operator.tekton.dev
```

Sample output

```
NAME VERSION READY REASON APIURL UIURL
hub v1.8.0 True https://api.route.url/ https://ui.route.url/
```

3.7.4. Disabling Tekton Hub authorization after upgrading the Red Hat OpenShift Pipelines Operator from 1.7 to 1.8

When you install Tekton Hub with Red Hat OpenShift Pipelines Operator 1.8, the login authorization and ratings for the Tekton Hub artifacts are disabled for the default installation. However, when you upgrade

the Operator from 1.7 to 1.8, the instance of the Tekton Hub on your cluster does not automatically disable the login authorization and ratings.

To disable login authorization and ratings for Tekton Hub after upgrading the Operator from 1.7 to 1.8, perform the steps in the following procedure.

Prerequisites

- Ensure that the Red Hat OpenShift Pipelines Operator is installed in the default **openshift-pipelines** namespace on the cluster.

Procedure

1. Delete the existing Tekton Hub API secret that you created while manually installing Tekton Hub for Operator 1.7.

```
$ oc delete secret tekton-hub-api -n <targetNamespace> 1
```

- 1 The common namespace for the Tekton Hub API secret and the Tekton Hub CR. By default, the target namespace is **openshift-pipelines**.

2. Delete the **TektonInstallerSet** object for the Tekton Hub API.

```
$ oc get tektoninstallerset -o name | grep tekton-hub-api | xargs oc delete
```



NOTE

After deletion, the Operator automatically creates a new Tekton Hub API installer set.

Wait and check the status of the Tekton Hub. Proceed to the next steps when the **READY** column displays **True**.

```
$ oc get tektonhub hub
```

Sample output

```
NAME VERSION    READY REASON  APIURL
UIURL
hub 1.8.0      True      https://tekton-hub-api-openshift-pipelines.apps.example.com
https://tekton-hub-ui-openshift-pipelines.apps.example.com
```

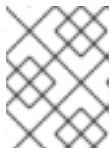
3. Delete the **ConfigMap** object for the Tekton Hub UI.

```
$ oc delete configmap tekton-hub-ui -n <targetNamespace> 1
```

- 1 The common namespace for the Tekton Hub UI and the Tekton Hub CR. By default, the target namespace is **openshift-pipelines**.

4. Delete the **TektonInstallerSet** object for the Tekton Hub UI.

```
$ oc get tektoninstallerset -o name | grep tekton-hub-ui | xargs oc delete
```



NOTE

After deletion, the Operator automatically creates a new Tekton Hub UI installer set.

Wait and check the status of the Tekton Hub. Proceed to the next steps when the **READY** column displays **True**.

```
$ oc get tektonhub hub
```

Sample output

NAME	VERSION	READY	REASON	APIURL	UIURL
hub	1.8.0	True		https://tekton-hub-api-openshift-pipelines.apps.example.com	https://tekton-hub-ui-openshift-pipelines.apps.example.com

3.7.5. Opting out of Tekton Hub in the Developer perspective

Cluster administrators can opt out of displaying Tekton Hub resources, such as tasks and pipelines, in the **Pipeline Builder** view of the **Developer** perspective of an OpenShift Container Platform cluster.

Prerequisite

- Ensure that the Red Hat OpenShift Pipelines Operator is installed on the cluster, and the **oc** command line tool is available.

Procedure

- To opt out of displaying Tekton Hub resources in the **Developer** perspective, set the value of the **enable-devconsole-integration** field in the **TektonConfig** custom resource (CR) to **false**.

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  targetNamespace: openshift-pipelines
  ...
  hub:
    params:
      - name: enable-devconsole-integration
        value: "false"
  ...
```

By default, the **TektonConfig** CR does not include the **enable-devconsole-integration** field, and the Red Hat OpenShift Pipelines Operator assumes that the value is **true**.



NOTE

Instead of opting out of displaying Tekton Hub resources in the **Developer** perspective, if you want to completely disable the Tekton Hub UI, set the **enableUI** field to **false** in the **TektonHub** CR.

3.7.6. Additional resources

- GitHub repository of [Tekton Hub](#)
- [Installing OpenShift Pipelines](#)
- [Red Hat OpenShift Pipelines release notes](#)

3.8. USING PIPELINES AS CODE

With Pipelines as Code, cluster administrators and users with the required privileges can define pipeline templates as part of source code Git repositories. When triggered by a source code push or a pull request for the configured Git repository, the feature runs the pipeline and reports the status.

3.8.1. Key features

Pipelines as Code supports the following features:

- Pull request status and control on the platform hosting the Git repository.
- GitHub Checks API to set the status of a pipeline run, including rechecks.
- GitHub pull request and commit events.
- Pull request actions in comments, such as **/retest**.
- Git events filtering and a separate pipeline for each event.
- Automatic task resolution in Pipelines, including local tasks, Tekton Hub, and remote URLs.
- Retrieval of configurations using GitHub blobs and objects API.
- Access Control List (ACL) over a GitHub organization, or using a Prow style **OWNER** file.
- The **tkn-pac** CLI plugin for managing bootstrapping and Pipelines as Code repositories.
- Support for GitHub App, GitHub Webhook, Bitbucket Server, and Bitbucket Cloud.

3.8.2. Installing Pipelines as Code on an OpenShift Container Platform

Pipelines as Code is installed in the **openshift-pipelines** namespace when you install the Red Hat OpenShift Pipelines Operator. For more details, see *Installing OpenShift Pipelines* in the *Additional resources* section.

To enable the default installation of Pipelines as Code with the Red Hat OpenShift Pipelines Operator, set the value of the **enable** parameter to **true** in the **TektonConfig** custom resource:

```
...
spec:
```

```
platforms:
  openshift:
    pipelinesAsCode:
      enable: true
    settings:
      application-name: Pipelines as Code CI
      auto-configure-new-github-repo: "false"
      bitbucket-cloud-check-source-ip: "true"
      hub-catalog-name: tekton
      hub-url: https://api.hub.tekton.dev/v1
      remote-tasks: "true"
      secret-auto-create: "true"
```

...

If you want to disable the default installation of Pipelines as Code with the Operator, set the value of the **enable** parameter to **false**.

3.8.3. Installing Pipelines as Code CLI

Cluster administrators can use the **tkn-pac** and **opc** CLI tools on local machines or as containers for testing. The **tkn-pac** and **opc** CLI tools are installed automatically when you install the **tkn** CLI for Red Hat OpenShift Pipelines.

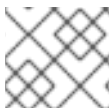
You can install the **tkn-pac** and **opc** version **1.9.0** binaries for the supported platforms:

- [Linux \(x86_64, amd64\)](#)
- [Linux on IBM Z and LinuxONE \(s390x\)](#)
- [Linux on IBM Power \(ppc64le\)](#)
- [macOS](#)
- [Windows](#)

3.8.4. Configuring Pipelines as Code for a Git repository hosting service provider

After installing Pipelines as Code, cluster administrators can configure a Git repository hosting service provider. Currently, the following services are supported:

- GitHub App
- GitHub Webhook
- GitLab
- Bitbucket Server
- Bitbucket Cloud



NOTE

GitHub App is the recommended service for using Pipelines as Code.

3.8.4.1. Configuring Pipelines as Code for a GitHub App

GitHub Apps act as a point of integration with Red Hat OpenShift Pipelines and bring the advantage of Git-based workflows to OpenShift Pipelines. Cluster administrators can configure a single GitHub App for all cluster users. For GitHub Apps to work with Pipelines as Code, ensure that the webhook of the GitHub App points to the Pipelines as Code event listener route (or ingress endpoint) that listens for GitHub events.

3.8.4.1.1. Configuring a GitHub App

Cluster administrators can create a GitHub App by running the following command:

```
$ tkn pac bootstrap github-app
```

If the **tkn pac** CLI plugin is not installed, you can create the GitHub App manually.

Procedure

To create and configure a GitHub App manually for Pipelines as Code, perform the following steps:

1. Sign in to your GitHub account.
2. Go to **Settings** → **Developer settings** → **GitHub Apps**, and click **New GitHub App**.
3. Provide the following information in the GitHub App form:
 - **GitHub Application Name:** **OpenShift Pipelines**
 - **Homepage URL:** OpenShift Console URL
 - **Webhook URL:** The Pipelines as Code route or ingress URL. You can find it by running the command `echo`

NOTE

If Pipelines as Code is installed without using the Red Hat OpenShift Pipelines Operator, use the **pipelines-as-code** namespace instead of **openshift-pipelines**.

- **Webhook secret:** An arbitrary secret. You can generate a secret by executing the command `openssl rand -hex 20`.
4. Select the following **Repository permissions**:
 - **Checks:** **Read & Write**
 - **Contents:** **Read & Write**
 - **Issues:** **Read & Write**
 - **Metadata:** **Read-only**
 - **Pull request:** **Read & Write**
 5. Select the following **Organization permissions**:
 - **Members:** **Readonly**

- **Plan: Readonly**

6. Select the following **User permissions**:

- **Check run**
- **Issue comment**
- **Pull request**
- **Push**

7. Click **Create GitHub App**

8. On the **Details** page of the newly created GitHub App, note the **App ID** displayed at the top.

9. In the **Private keys** section, click **Generate Private key** to automatically generate and download a private key for the GitHub app. Securely store the private key for future reference and usage.

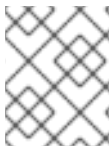
10. Install the created App on a repository that you want to use with Pipelines as Code.

3.8.4.1.2. Configuring Pipelines as Code to access a GitHub App

To configure Pipelines as Code to access the newly created GitHub App, execute the following command:

```
$ oc -n openshift-pipelines create secret generic pipelines-as-code-secret \
  --from-literal github-private-key="$(cat <PATH_PRIVATE_KEY>)" \
  --from-literal github-application-id="<APP_ID>" \
  --from-literal webhook.secret="<WEBHOOK_SECRET>"
```

- 1 If Pipelines as Code is installed without using the Red Hat OpenShift Pipelines Operator, use the **pipelines-as-code** namespace instead of **openshift-pipelines**.
- 2 The path to the private key you downloaded while configuring the GitHub App.
- 3 The **App ID** of the GitHub App.
- 4 The webhook secret provided when you created the GitHub App.



NOTE

Pipelines as Code works automatically with GitHub Enterprise by detecting the header set from GitHub Enterprise and using it for the GitHub Enterprise API authorization URL.

3.8.4.2. Creating a GitHub App in administrator perspective

As a cluster administrator, you can configure your GitHub App with the OpenShift Container Platform cluster to use Pipelines as Code. This configuration allows you to execute a set of tasks required for build deployment.

Prerequisites

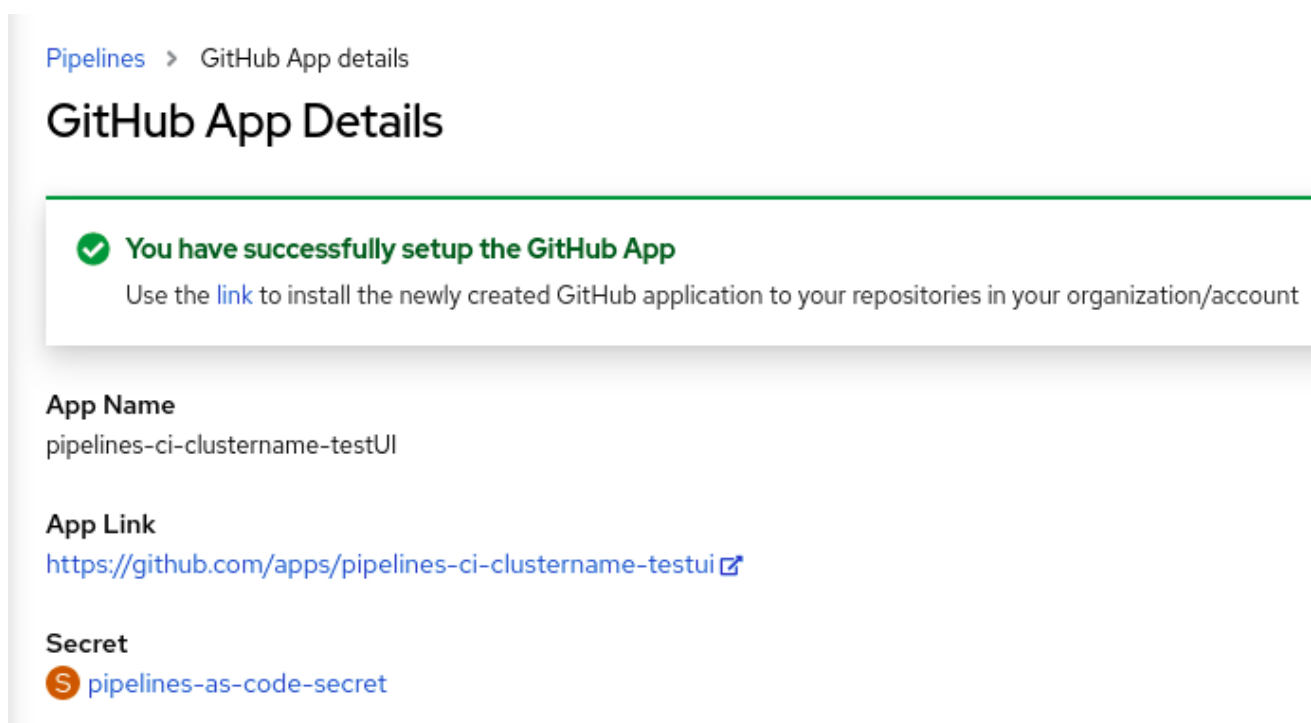
You have installed the Red Hat OpenShift Pipelines **pipelines-1.8** operator from the Operator Hub.

Procedure

1. In the administrator perspective, navigate to **Pipelines** using the navigation pane.
2. Click **Setup GitHub App** on the **Pipelines** page.
3. Enter your GitHub App name. For example, **pipelines-ci-clustername-testui**.
4. Click **Setup**.
5. Enter your Git password when prompted in the browser.
6. Click **Create GitHub App for <username>** where **<username>** is your GitHub user name.

Verification

After successful creation of the GitHub App, the OpenShift Container Platform web console opens and displays the details about the application.



The details of the GitHub App are saved as a secret in the **openShift-pipelines** namespace.

To view details such as name, link, and secret associated with the GitHub applications, navigate to **Pipelines** and click **View GitHub App**.

3.8.5. Pipelines as Code command reference

The **tkn-pac** CLI tool offers the following capabilities:

- Bootstrap Pipelines as Code installation and configuration.
- Create a new Pipelines as Code repository.
- List all Pipelines as Code repositories.
- Describe a Pipelines as Code repository and the associated runs.

- Generate a simple pipeline run to get started.
- Resolve a pipeline run as if it was executed by Pipelines as Code.

TIP

You can use the commands corresponding to the capabilities for testing and experimentation, so that you don't have to make changes to the Git repository containing the application source code.

3.8.5.1. Basic syntax

```
$ tkn pac [command or options] [arguments]
```

3.8.5.2. Global options

```
$ tkn pac --help
```

3.8.5.3. Utility commands

3.8.5.3.1. bootstrap

Table 3.7. Bootstrapping Pipelines as Code installation and configuration

Command	Description
tkn pac bootstrap	Installs and configures Pipelines as Code for Git repository hosting service providers, such as GitHub and GitHub Enterprise.
tkn pac bootstrap --nightly	Installs the nightly build of Pipelines as Code.
tkn pac bootstrap --route-url <public_url_to_ingress_spec>	<p>Overrides the OpenShift route URL.</p> <p>By default, tkn pac bootstrap detects the OpenShift route, which is automatically associated with the Pipelines as Code controller service.</p> <p>If you do not have an OpenShift Container Platform cluster, it asks you for the public URL that points to the ingress endpoint.</p>
tkn pac bootstrap github-app	Create a GitHub application and secrets in the pipelines-as-code namespace.

3.8.5.3.2. repository

Table 3.8. Managing Pipelines as Code repositories

Command	Description
tkn pac create repository	Creates a new Pipelines as Code repository and a namespace based on the pipeline run template.
tkn pac list	Lists all the Pipelines as Code repositories and displays the last status of the associated runs.
tkn pac repo describe	Describes a Pipelines as Code repository and the associated runs.

3.8.5.3.3. generate

Table 3.9. Generating pipeline runs using Pipelines as Code

Command	Description
tkn pac generate	<p>Generates a simple pipeline run.</p> <p>When executed from the directory containing the source code, it automatically detects current Git information.</p> <p>In addition, it uses basic language detection capability and adds extra tasks depending on the language.</p> <p>For example, if it detects a setup.py file at the repository root, the pylint task is automatically added to the generated pipeline run.</p>

3.8.5.3.4. resolve

Table 3.10. Resolving and executing pipeline runs using Pipelines as Code

Command	Description
tkn pac resolve	Executes a pipeline run as if it is owned by the Pipelines as Code on service.
tkn pac resolve -f .tekton/pull-request.yaml oc apply -f -	<p>Displays the status of a live pipeline run that uses the template in .tekton/pull-request.yaml.</p> <p>Combined with a Kubernetes installation running on your local machine, you can observe the pipeline run without generating a new commit.</p> <p>If you run the command from a source code repository, it attempts to detect the current Git information and automatically resolve parameters such as current revision or branch.</p>

Command	Description
tkn pac resolve -f .tekton/pr.yaml -p revision=main -p repo_name=<repository_name>	<p>Executes a pipeline run by overriding default parameter values derived from the Git repository.</p> <p>The -f option can also accept a directory path and apply the tkn pac resolve command on all .yaml or .yml files in that directory. You can also use the -f flag multiple times in the same command.</p> <p>You can override the default information gathered from the Git repository by specifying parameter values using the -p option. For example, you can use a Git branch as a revision and a different repository name.</p>

3.8.6. Customizing Pipelines as Code configuration

To customize Pipelines as Code, cluster administrators can configure the following parameters using the **pipelines-as-code** config map in the **pipelines-as-code** namespace:

Table 3.11. Customizing Pipelines as Code configuration

Parameter	Description	Default
application-name	The name of the application. For example, the name displayed in the GitHub Checks labels.	"Pipelines as Code CI"
max-keep-days	<p>The number of the days for which the executed pipeline runs are kept in the pipelines-as-code namespace.</p> <p>Note that this ConfigMap setting does not affect the cleanups of a user's pipeline runs, which are controlled by the annotations on the pipeline run definition in the user's GitHub repository.</p>	NA
secret-auto-create	Indicates whether or not a secret should be automatically created using the token generated in the GitHub application. This secret can then be used with private repositories.	enabled

Parameter	Description	Default
remote-tasks	When enabled, allows remote tasks from pipeline run annotations.	enabled
hub-url	The base URL for the Tekton Hub API .	https://hub.tekton.dev/
hub-catalog-name	The Tekton Hub catalog name.	tekton
tekton-dashboard-url	The URL of the Tekton Hub dashboard. Pipelines as Code uses this URL to generate a PipelineRun URL on the Tekton Hub dashboard.	NA
bitbucket-cloud-check-source-ip	Indicates whether to secure the service requests by querying IP ranges for a public bitbucket. Changing the parameter's default value might result into a security issue.	enabled
bitbucket-cloud-additional-source-ip	Indicates whether to provide an additional set of IP ranges or networks, which are separated by commas.	NA
max-keep-run-upper-limit	A maximum limit for the max-keep-run value for a pipeline run.	NA
default-max-keep-runs	A default limit for the max-keep-run value for a pipeline run. If defined, the value is applied to all pipeline runs that do not have a max-keep-run annotation.	NA
auto-configure-new-github-repo	Configures new GitHub repositories automatically. Pipelines as Code sets up a namespace and creates a custom resource for your repository. This parameter is only supported with GitHub applications.	disabled
auto-configure-repo-namespace-template	Configures a template to automatically generate the namespace for your new repository, if auto-configure-new-github-repo is enabled.	{repo_name}-pipelines

Parameter	Description	Default
error-log-snippet	Enables or disables the view of a log snippet for the failed tasks, with an error in a pipeline. You can disable this parameter in the case of data leakage from your pipeline.	enabled

3.8.7. Additional resources

- [Installing OpenShift Pipelines](#)
- [Installing tkn](#)
- [Red Hat OpenShift Pipelines release notes](#)

3.9. WORKING WITH RED HAT OPENSIFT PIPELINES USING THE DEVELOPER PERSPECTIVE

You can use the **Developer** perspective of the OpenShift Container Platform web console to create CI/CD pipelines for your software delivery process.

In the **Developer** perspective:

- Use the **Add → Pipeline → Pipeline Builder** option to create customized pipelines for your application.
- Use the **Add → From Git** option to create pipelines using operator-installed pipeline templates and resources while creating an application on OpenShift Container Platform.

After you create the pipelines for your application, you can view and visually interact with the deployed pipelines in the **Pipelines** view. You can also use the **Topology** view to interact with the pipelines created using the **From Git** option. You need to apply custom labels to a pipeline created using the **Pipeline Builder** to see it in the **Topology** view.

Prerequisites

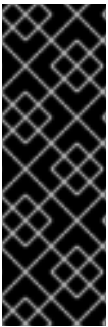
- You have access to an OpenShift Container Platform cluster and have switched to [the Developer perspective](#).
- You have the [OpenShift Pipelines Operator installed](#) in your cluster.
- You are a cluster administrator or a user with create and edit permissions.
- You have created a project.

3.9.1. Constructing Pipelines using the Pipeline Builder

In the **Developer** perspective of the console, you can use the **+Add → Pipeline → Pipeline builder** option to:

- Configure pipelines using either the **Pipeline builder** or the **YAML view**.

- Construct a pipeline flow using existing tasks and cluster tasks. When you install the OpenShift Pipelines Operator, it adds reusable pipeline cluster tasks to your cluster.
- Specify the type of resources required for the pipeline run, and if required, add additional parameters to the pipeline.
- Reference these pipeline resources in each of the tasks in the pipeline as input and output resources.
- If required, reference any additional parameters added to the pipeline in the task. The parameters for a task are prepopulated based on the specifications of the task.
- Use the Operator-installed, reusable snippets and samples to create detailed pipelines.
- Search and add tasks from your configured local Tekton Hub instance.



IMPORTANT

In the developer perspective, you can create a customized pipeline using your own set of curated tasks. To search, install, and upgrade your tasks directly from the developer console, your cluster administrator needs to install and deploy a local Tekton Hub instance and link that hub to the OpenShift Container Platform cluster. For more details, see *Using Tekton Hub with OpenShift Pipelines* in the *Additional resources* section. If you do not deploy any local Tekton Hub instance, by default, you can only access the cluster tasks, namespace tasks and public Tekton Hub tasks.

Procedure

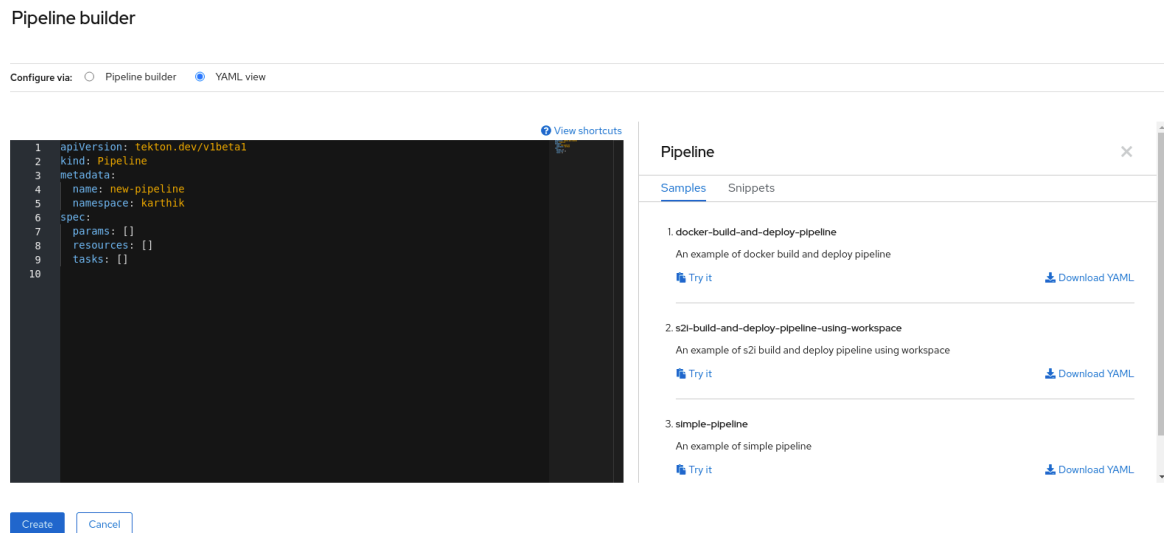
1. In the **+Add** view of the **Developer** perspective, click the **Pipeline** tile to see the **Pipeline builder** page.
2. Configure the pipeline using either the **Pipeline builder** view or the **YAML view**.



NOTE

The **Pipeline builder** view supports a limited number of fields whereas the **YAML view** supports all available fields. Optionally, you can also use the Operator-installed, reusable snippets and samples to create detailed Pipelines.

Figure 3.1. YAML view



Configure your pipeline using the **Pipeline Builder**:

- a. In the **Name** field, enter a unique name for the pipeline.
- b. In the **Tasks** section:
 - i. Click **Add task**.
 - ii. Search for a task using the quick search field and select the required task from the displayed list.
 - iii. Click **Add** or **Install and add**. In this example, use the **s2i-nodejs** task.

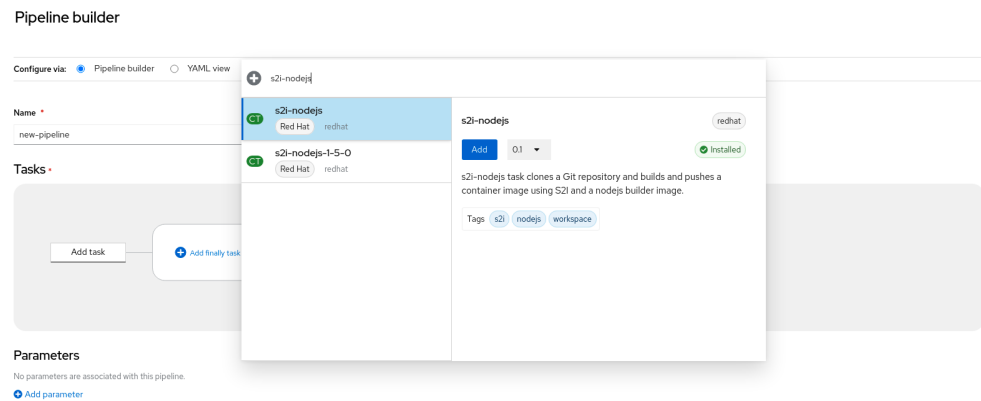


NOTE

The search list contains all the Tekton Hub tasks and tasks available in the cluster. Also, if a task is already installed it will show **Add** to add the task whereas it will show **Install and add** to install and add the task. It will show **Update and add** when you add the same task with an updated version.

- To add sequential tasks to the pipeline:
 - Click the plus icon to the right or left of the task → click **Add task**.
 - Search for a task using the quick search field and select the required task from the displayed list.
 - Click **Add** or **Install and add**

Figure 3.2. Pipeline Builder



- To add a final task:
 - Click the **Add finally task** → Click **Add task**.
 - Search for a task using the quick search field and select the required task from the displayed list.
 - Click **Add** or **Install and add**.
- c. In the **Resources** section, click **Add Resources** to specify the name and type of resources for the pipeline run. These resources are then used by the tasks in the pipeline as inputs and outputs. For this example:
 - i. Add an input resource. In the **Name** field, enter **Source**, and then from the **Resource Type** drop-down list, select **Git**.
 - ii. Add an output resource. In the **Name** field, enter **Img**, and then from the **Resource Type** drop-down list, select **Image**.

**NOTE**

A red icon appears next to the task if a resource is missing.

- d. Optional: The **Parameters** for a task are pre-populated based on the specifications of the task. If required, use the **Add Parameters** link in the **Parameters** section to add additional parameters.
- e. In the **Workspaces** section, click **Add workspace** and enter a unique workspace name in the **Name** field. You can add multiple workspaces to the pipeline.
- f. In the **Tasks** section, click the **s2i-nodejs** task to see the side panel with details for the task. In the task side panel, specify the resources and parameters for the **s2i-nodejs** task:
 - i. If required, in the **Parameters** section, add more parameters to the default ones, by using the `$(params.<param-name>)` syntax.
 - ii. In the **Image** section, enter **Img** as specified in the **Resources** section.
 - iii. Select a workspace from the **source** drop-down under **Workspaces** section.
- g. Add resources, parameters, and workspaces to the **openshift-client** task.

3. Click **Create** to create and view the pipeline in the **Pipeline Details** page.
4. Click the **Actions** drop-down menu then click **Start**, to see the **Start Pipeline** page.
5. The **Workspaces** section lists the workspaces you created earlier. Use the respective drop-down to specify the volume source for your workspace. You have the following options: **Empty Directory**, **Config Map**, **Secret**, **PersistentVolumeClaim**, or **VolumeClaimTemplate**.

3.9.2. Creating OpenShift Pipelines along with applications

To create pipelines along with applications, use the **From Git** option in the **Add+** view of the **Developer** perspective. You can view all of your available pipelines and select the pipelines you want to use to create applications while importing your code or deploying an image.

The Tekton Hub Integration is enabled by default and you can see tasks from the Tekton Hub that are supported by your cluster. Administrators can opt out of the Tekton Hub Integration and the Tekton Hub tasks will no longer be displayed. You can also check whether a webhook URL exists for a generated pipeline. Default webhooks are added for the pipelines that are created using the **+Add** flow and the URL is visible in the side panel of the selected resources in the Topology view.

For more information, see [Creating applications using the Developer perspective](#).

3.9.3. Adding a GitHub repository containing pipelines

In the developer perspective, you can add your GitHub repository containing pipelines to the OpenShift Container Platform cluster. This allows you to run pipelines and tasks from your GitHub repository on the cluster when relevant Git events, such as push or pull requests are triggered.



NOTE

You can add both public and private GitHub repositories.

Prerequisites

- Ensure that your cluster administrator has configured the required GitHub applications in the administrator perspective.

Procedure

1. In the developer perspective, choose the namespace or project in which you want to add your GitHub repository.
2. Navigate to **Pipelines** using the left navigation pane.
3. Click **Create → Repository** on the right side of the Pipelines page.
4. Enter your **Git Repo URL** and the console automatically fetches the repository name.
5. Click **Show configuration options**. By default, you see only one option **Setup a webhook**. If you have a GitHub application configured, you see two options:
 - **Use GitHub App**: Select this option to install your GitHub application in your repository.
 - **Setup a webhook**: Select this option to add a webhook to your GitHub application.
6. Set up a webhook using one of the following options in the **Secret** section:

- Setup a webhook using **Git access token**
 - Enter your personal access token.
 - Click **Generate** corresponding to the **Webhook secret** field to generate a new webhook secret.

Project: openshift-pipelines ▼

Add Git Repository

Git Repo URL *

https://github.com/apps/pipelines-ci-clustername-ss-test

Name *

git-pipelines-ci-clustername-ss-test

▼ Hide configuration options

Secret


☒ Git access token

ghp_Z9eb6i5LrR3cxEPtOngDR1laoZeaj3uN28o


Use your GitHub Personal token. Use this [link](#) to create a token with repo, public_repo & admin:repo_hook scopes and give your token an expiration, i.e 30d.

☐ Git access token secret

Webhook secret

64bdd2115bab0219c2ac82fc13fbac63da3d9bb  **Generate**

› [See GitHub permissions](#)

[Read more about setting up webhook](#) 

Add **Cancel**



NOTE

You can click the link below the **Git access token** field if you do not have a personal access token and want to create a new one.

- Setup a webhook using **Git access token secret**
 - Select a secret in your namespace from the dropdown list. Depending on the secret you selected, a webhook secret is automatically generated.

Project: openshift-pipelines ▼

Add Git Repository

Git Repo URL *

`https://github.com/apps/pipelines-ci-clustername-ss-test`

Name *


`git-pipelines-ci-clustername-ss-test`

▼ Hide configuration options

Secret


☐ Git access token

☒ Git access token secret

 pipelines-as-code-secret ▼

Secret with the Git access token for pulling pipeline and tasks from your Git repository.

Webhook secret

`64bdd2115bab0219c2ac82fc13fbbac63da3d9bb`  [Generate](#)

► [See GitHub permissions](#)

[Read more about setting up webhook](#)

[Add](#) [Cancel](#)

7. Add the webhook secret details to your GitHub repository:
 - a. Copy the **webhook URL** and navigate to your GitHub repository settings.
 - b. Click **Webhooks → Add webhook**.
 - c. Copy the **Webhook URL** from the developer console and paste it in the **Payload URL** field of the GitHub repository settings.
 - d. Select the **Content type**.
 - e. Copy the **Webhook secret** from the developer console and paste it in the **Secret** field of the GitHub repository settings.
 - f. Select one of the **SSL verification** options.
 - g. Select the events to trigger this webhook.
 - h. Click **Add webhook**.
8. Navigate back to the developer console and click **Add**.
9. Read the details of the steps that you have to perform and click **Close**.
10. View the details of the repository you just created.

3.9.4. Interacting with pipelines using the Developer perspective

The **Pipelines** view in the **Developer** perspective lists all the pipelines in a project, along with the following details:

- The namespace in which the pipeline was created
- The last pipeline run
- The status of the tasks in the pipeline run
- The status of the pipeline run
- The creation time of the last pipeline run

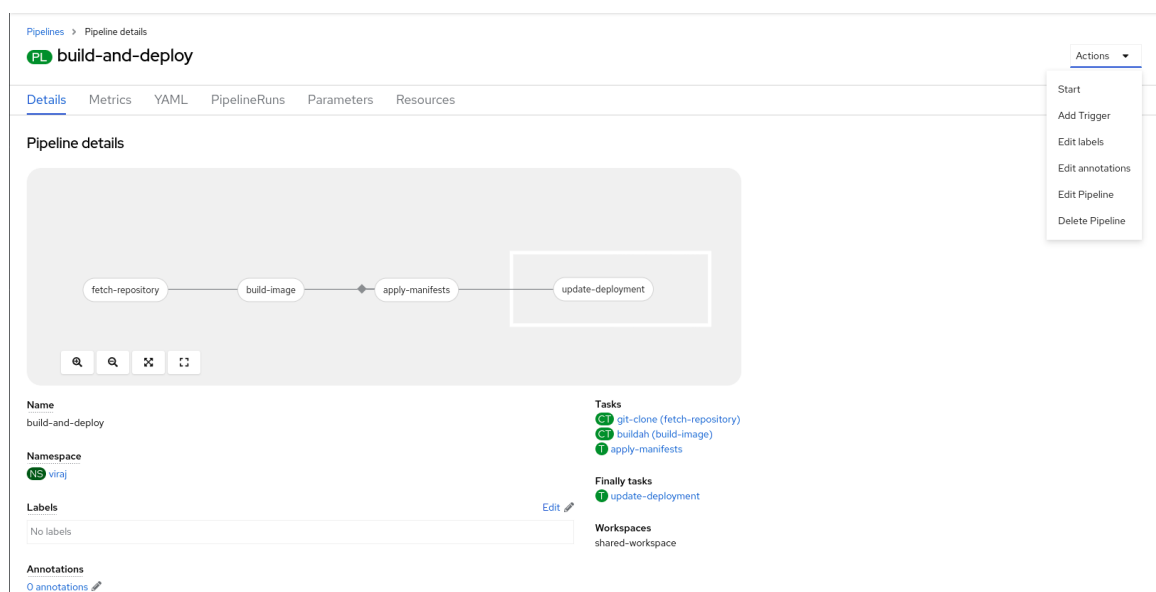
Procedure

1. In the **Pipelines** view of the **Developer** perspective, select a project from the **Project** drop-down list to see the pipelines in that project.
2. Click the required pipeline to see the **Pipeline details** page.
By default, the **Details** tab displays a visual representation of all the **serial** tasks, **parallel** tasks, **finally** tasks, and **when** expressions in the pipeline. The tasks and the **finally** tasks are listed in the lower right portion of the page.

To view the task details, click the listed **Tasks** and **Finally** tasks. In addition, you can do the following:

- Use the zoom in, zoom out, fit to screen, and reset view features using the standard icons displayed in the lower left corner of the **Pipeline details** visualization.
- Change the zoom factor of the pipeline visualization using the mouse wheel.
- Hover over the tasks and see the task details.

Figure 3.3. Pipeline details



3. Optional: On the **Pipeline details** page, click the **Metrics** tab to see the following information about pipelines:

• Pipeline Success Ratio

- **Pipeline Success Ratio**
- **Number of Pipeline Runs**
- **Pipeline Run Duration**
- **Task Run Duration**

You can use this information to improve the pipeline workflow and eliminate issues early in the pipeline lifecycle.

- Optional: Click the **YAML** tab to edit the YAML file for the pipeline.
- Optional: Click the **Pipeline Runs** tab to see the completed, running, or failed runs for the pipeline.

The **Pipeline Runs** tab provides details about the pipeline run, the status of the task, and a link



to debug failed pipeline runs. Use the Options menu to stop a running pipeline, to rerun a pipeline using the same parameters and resources as that of the previous pipeline execution, or to delete a pipeline run.

- Click the required pipeline run to see the **Pipeline Run details** page. By default, the **Details** tab displays a visual representation of all the serial tasks, parallel tasks, **finally** tasks, and when expressions in the pipeline run. The results for successful runs are displayed under the **Pipeline Run results** pane at the bottom of the page. Additionally, you would only be able to see tasks from Tekton Hub which are supported by the cluster. While looking at a task, you can click the link beside it to jump to the task documentation.



NOTE

The **Details** section of the **Pipeline Run Details** page displays a **Log Snippet** of the failed pipeline run. **Log Snippet** provides a general error message and a snippet of the log. A link to the **Logs** section provides quick access to the details about the failed run.

- On the **Pipeline Run details** page, click the **Task Runs** tab to see the completed, running, and failed runs for the task.

The **Task Runs** tab provides information about the task run along with the links to its task



and pod, and also the status and duration of the task run. Use the Options menu to delete a task run.

- Click the required task run to see the **Task Run details** page. The results for successful runs are displayed under the **Task Run results** pane at the bottom of the page.



NOTE

The **Details** section of the **Task Run details** page displays a **Log Snippet** of the failed task run. **Log Snippet** provides a general error message and a snippet of the log. A link to the **Logs** section provides quick access to the details about the failed task run.

- Click the **Parameters** tab to see the parameters defined in the pipeline. You can also add or edit additional parameters, as required.

7. Click the **Resources** tab to see the resources defined in the pipeline. You can also add or edit additional resources, as required.

3.9.5. Using a custom pipeline template for creating and deploying an application from a Git repository

As a cluster administrator, to create and deploy an application from a Git repository, you can use custom pipeline templates that override the default pipeline templates provided by Red Hat OpenShift Pipelines 1.5 and later.



NOTE

This feature is unavailable in Red Hat OpenShift Pipelines 1.4 and earlier versions.

Prerequisites

Ensure that the Red Hat OpenShift Pipelines 1.5 or later is installed and available in all namespaces.

Procedure

1. Log in to the OpenShift Container Platform web console as a cluster administrator.
2. In the **Administrator** perspective, use the left navigation panel to go to the *Pipelines* section.
 - a. From the **Project** drop-down, select the **openshift** project. This ensures that the subsequent steps are performed in the **openshift** namespace.
 - b. From the list of available pipelines, select a pipeline that is appropriate for building and deploying your application. For example, if your application requires a **node.js** runtime environment, select the **s2i-nodejs** pipeline.



NOTE

Do not edit the default pipeline template. It may become incompatible with the UI and the back-end.

- c. Under the **YAML** tab of the selected pipeline, click **Download** and save the YAML file to your local machine. If your custom configuration file fails, you can use this copy to restore a working configuration.
3. Disable (delete) the default pipeline templates:
 - a. Use the left navigation panel to go to **Operators** → **Installed Operators**.
 - b. Click **Red Hat OpenShift Pipelines** → **Tekton Configuration** tab → **config** → **YAML** tab.
 - c. To disable (delete) the default pipeline templates in the **openshift** namespace, set the **pipelineTemplates** parameter to **false** in the **TektonConfig** custom resource YAML, and save it.

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
```

```

profile: all
targetNamespace: openshift-pipelines
addon:
  params:
    - name: clusterTasks
      value: "true"
    - name: pipelineTemplates
      value: "false"
  ...

```

**NOTE**

If you manually delete the default pipeline templates, the operator will restore the defaults during an upgrade.

**WARNING**

As a cluster admin, you can disable installation of the default pipeline templates in the operator configuration. However, such a configuration will disable (delete) all default pipeline templates, and not only the one you want to customize.

4. Create a custom pipeline template:

- a. Use the left navigation panel to go to the *Pipelines* section.
- b. From the **Create** drop-down, select **Pipeline**.
- c. Create the required pipeline in the **openshift** namespace. Give it a different name than the default one (for example, **custom-nodejs**). You can use the downloaded default pipeline template as a starting point and customize it.

**NOTE**

Because **openshift** is the default namespace used by the operator-installed pipeline templates, you must create the custom pipeline template in the **openshift** namespace. When an application uses a pipeline template, the template is automatically copied to the respective project's namespace.

- d. Under the **Details** tab of the created pipeline, ensure that the **Labels** in the custom template match the labels in the default pipeline. The custom pipeline template must have the correct labels for the runtime, type, and strategy of the application. For example, the required labels for a **node.js** application deployed on OpenShift Container Platform are as follows:

```

...
pipeline.openshift.io/runtime: nodejs
pipeline.openshift.io/type: openshift
...

```

**NOTE**

You can use only one pipeline template for each combination of runtime environment and deployment type.


5. In the **Developer** perspective, use the **+Add → Git Repository → From Git** option to select the kind of application you want to create and deploy. Based on the required runtime and type of the application, your custom template is automatically selected.

3.9.6. Starting pipelines from Pipelines view

After you create a pipeline, you need to start it to execute the included tasks in the defined sequence. You can start a pipeline from the **Pipelines** view, the **Pipeline Details** page, or the **Topology** view.

Procedure

To start a pipeline using the **Pipelines** view:

1. In the **Pipelines** view of the **Developer** perspective, click the **Options**  menu adjoining a pipeline, and select **Start**.
2. The **Start Pipeline** dialog box displays the **Git Resources** and the **Image Resources** based on the pipeline definition.

**NOTE**

For pipelines created using the **From Git** option, the **Start Pipeline** dialog box also displays an **APP_NAME** field in the **Parameters** section, and all the fields in the dialog box are prepopulated by the pipeline template.

- a. If you have resources in your namespace, the **Git Resources** and the **Image Resources** fields are prepopulated with those resources. If required, use the drop-downs to select or create the required resources and customize the pipeline run instance.
3. Optional: Modify the **Advanced Options** to add the credentials that authenticate the specified private Git server or the image registry.
 - a. Under **Advanced Options**, click **Show Credentials Options** and select **Add Secret**.
 - b. In the **Create Source Secret** section, specify the following:
 - i. A unique **Secret Name** for the secret.
 - ii. In the **Designated provider to be authenticated** section, specify the provider to be authenticated in the **Access to** field, and the base **Server URL**.
 - iii. Select the **Authentication Type** and provide the credentials:
 - For the **Authentication Type Image Registry Credentials**, specify the **Registry Server Address** that you want to authenticate, and provide your credentials in the **Username**, **Password**, and **Email** fields. Select **Add Credentials** if you want to specify an additional **Registry Server Address**.

- For the **Authentication Type Basic Authentication**, specify the values for the **UserName** and **Password or Token** fields.
- For the **Authentication Type SSH Keys**, specify the value of the **SSH Private Key** field.



NOTE

For basic authentication and SSH authentication, you can use annotations such as:

- **tekton.dev/git-0:** <https://github.com>
- **tekton.dev/git-1:** <https://gitlab.com>.

iv. Select the check mark to add the secret.

You can add multiple secrets based upon the number of resources in your pipeline.

4. Click **Start** to start the pipeline.

5. The **PipelineRun details** page displays the pipeline being executed. After the pipeline starts, the tasks and steps within each task are executed. You can:

- Use the zoom in, zoom out, fit to screen, and reset view features using the standard icons, which are in the lower left corner of the **PipelineRun details** page visualization.
- Change the zoom factor of the pipelinerun visualization using the mouse wheel. At specific zoom factors, the background color of the tasks changes to indicate the error or warning status.
- Hover over the tasks to see the details, such as the time taken to execute each step, task name, and task status.
- Hover over the tasks badge to see the total number of tasks and tasks completed.
- Click on a task to see the logs for each step in the task.
- Click the **Logs** tab to see the logs relating to the execution sequence of the tasks. You can also expand the pane and download the logs individually or in bulk, by using the relevant button.
- Click the **Events** tab to see the stream of events generated by a pipeline run. You can use the **Task Runs**, **Logs**, and **Events** tabs to assist in debugging a failed pipeline run or a failed task run.

Figure 3.4. Pipeline run details

PipelineRun details

Name: django-ex-git-oylqln

Namespace: ns-viraj

Labels: app.kubernetes.io/instance=django-ex-git, app.kubernetes.io/name=django-ex-git, operator.tekton.dev/operand-name=openshift-pipelines-addons, pipeline.openshift.io/runtime=python, pipeline.openshift.io/nrmls-version=3.9-ubi8, pipeline.openshift.io/type=kubernetes, tekton.dev/pipeline=django-ex-git

Status: Running

Pipeline: django-ex-git

Triggered by: kubecadmin

Workspace Resources: pvc-9996cf3221 (workspace)

Created at: 6 Oct 2022, 14:25

3.9.7. Starting pipelines from Topology view

For pipelines created using the **From Git** option, you can use the **Topology** view to interact with pipelines after you start them:



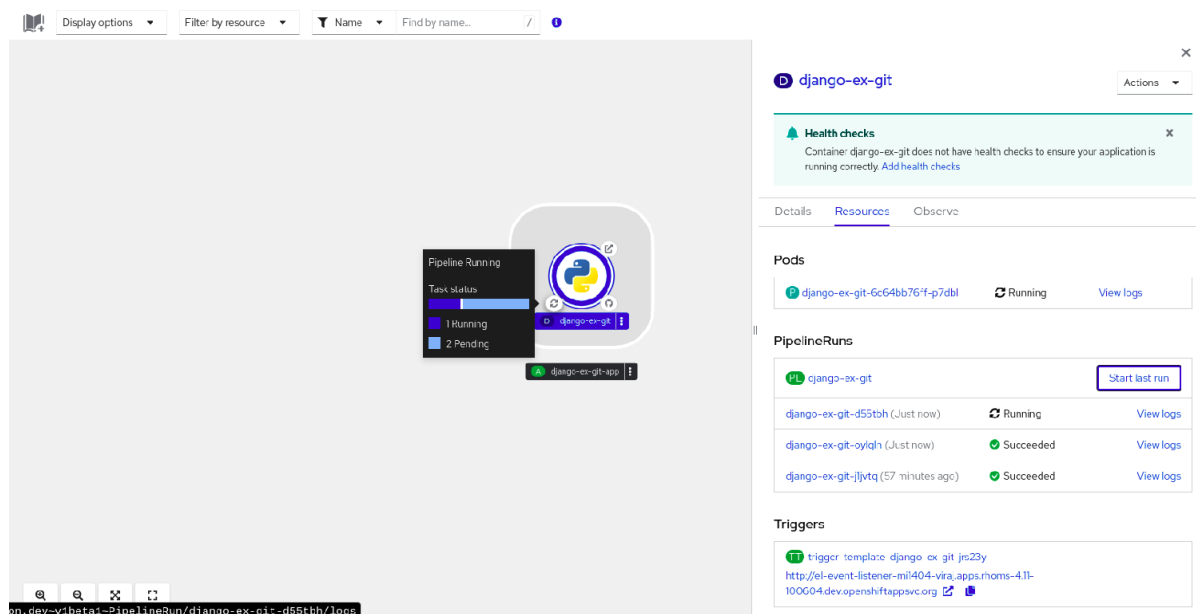
NOTE

To see the pipelines created using the **Pipeline Builder** in the **Topology** view, customize the pipeline labels to link the pipeline with the application workload.

Procedure

1. Click **Topology** in the left navigation panel.
2. Click the application to display **Pipeline Runs** in the side panel.
3. In **Pipeline Runs**, click **Start Last Run** to start a new pipeline run with the same parameters and resources as the previous one. This option is disabled if a pipeline run has not been initiated. You can also start a pipeline run when you create it.

Figure 3.5. Pipelines in Topology view



In the **Topology** page, hover to the left of the application to see the status of its pipeline run. After a pipeline is added, a bottom left icon indicates that there is an associated pipeline.

3.9.8. Interacting with pipelines from Topology view

The side panel of the application node in the **Topology** page displays the status of a pipeline run and you can interact with it.

- If a pipeline run does not start automatically, the side panel displays a message that the pipeline cannot be automatically started, hence it would need to be started manually.
- If a pipeline is created but the user has not started the pipeline, its status is not started. When the user clicks the **Not started** status icon, the start dialog box opens in the **Topology** view.
- If the pipeline has no build or build config, the **Builds** section is not visible. If there is a pipeline and build config, the **Builds** section is visible.
- The side panel displays a **Log Snippet** when a pipeline run fails on a specific task run. You can view the **Log Snippet** in the **Pipeline Runs** section, under the **Resources** tab. It provides a general error message and a snippet of the log. A link to the **Logs** section provides quick access to the details about the failed run.

3.9.9. Editing Pipelines

You can edit the Pipelines in your cluster using the **Developer** perspective of the web console:

Procedure

1. In the **Pipelines** view of the **Developer** perspective, select the Pipeline you want to edit to see the details of the Pipeline. In the **Pipeline Details** page, click **Actions** and select **Edit Pipeline**.
2. In the **Pipeline Builder** page:
 - You can add additional Tasks, parameters, or resources to the Pipeline.


- You can click the Task you want to modify to see the Task details in the side panel and modify the required Task details, such as the display name, parameters and resources.
- Alternatively, to delete the Task, click the Task, and in the side panel, click **Actions** and select **Remove Task**.

3. Click **Save** to save the modified Pipeline.

3.9.10. Deleting Pipelines

You can delete the Pipelines in your cluster using the **Developer** perspective of the web console.

Procedure

1. In the **Pipelines** view of the **Developer** perspective, click the **Options**  menu adjoining a Pipeline, and select **Delete Pipeline**.
2. In the **Delete Pipeline** confirmation prompt, click **Delete** to confirm the deletion.

3.9.11. Additional resources

- [Using Tekton Hub with OpenShift Pipelines](#)

3.10. REDUCING RESOURCE CONSUMPTION OF OPENSIFT PIPELINES

If you use clusters in multi-tenant environments you must control the consumption of CPU, memory, and storage resources for each project and Kubernetes object. This helps prevent any one application from consuming too many resources and affecting other applications.

To define the final resource limits that are set on the resulting pods, Red Hat OpenShift Pipelines use resource quota limits and limit ranges of the project in which they are executed.

To restrict resource consumption in your project, you can:

- [Set and manage resource quotas](#) to limit the aggregate resource consumption.
- Use [limit ranges to restrict resource consumption](#) for specific objects, such as pods, images, image streams, and persistent volume claims.

3.10.1. Understanding resource consumption in pipelines

Each task consists of a number of required steps to be executed in a particular order defined in the **steps** field of the **Task** resource. Every task runs as a pod, and each step runs as a container within that pod.

Steps are executed one at a time. The pod that executes the task only requests enough resources to run a single container image (step) in the task at a time, and thus does not store resources for all the steps in the task.

The **Resources** field in the **steps** spec specifies the limits for resource consumption. By default, the resource requests for the CPU, memory, and ephemeral storage are set to **BestEffort** (zero) values or to the minimums set through limit ranges in that project.

Example configuration of resource requests and limits for a step

```
spec:
  steps:
  - name: <step_name>
    resources:
      requests:
        memory: 2Gi
        cpu: 600m
      limits:
        memory: 4Gi
        cpu: 900m
```

When the **LimitRange** parameter and the minimum values for container resource requests are specified in the project in which the pipeline and task runs are executed, Red Hat OpenShift Pipelines looks at all the **LimitRange** values in the project and uses the minimum values instead of zero.

Example configuration of limit range parameters at a project level

```
apiVersion: v1
kind: LimitRange
metadata:
  name: <limit_container_resource>
spec:
  limits:
  - max:
      cpu: "600m"
      memory: "2Gi"
    min:
      cpu: "200m"
      memory: "100Mi"
    default:
      cpu: "500m"
      memory: "800Mi"
    defaultRequest:
      cpu: "100m"
      memory: "100Mi"
  type: Container
...
```

3.10.2. Mitigating extra resource consumption in pipelines

When you have resource limits set on the containers in your pod, OpenShift Container Platform sums up the resource limits requested as all containers run simultaneously.

To consume the minimum amount of resources needed to execute one step at a time in the invoked task, Red Hat OpenShift Pipelines requests the maximum CPU, memory, and ephemeral storage as specified in the step that requires the most amount of resources. This ensures that the resource requirements of all the steps are met. Requests other than the maximum values are set to zero.

However, this behavior can lead to higher resource usage than required. If you use resource quotas, this could also lead to unschedulable pods.

For example, consider a task with two steps that uses scripts, and that does not define any resource limits and requests. The resulting pod has two init containers (one for entrypoint copy, the other for writing scripts) and two containers, one for each step.

OpenShift Container Platform uses the limit range set up for the project to compute required resource requests and limits. For this example, set the following limit range in the project:

```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-min-max-demo-lr
spec:
  limits:
  - max:
      memory: 1Gi
    min:
      memory: 500Mi
    type: Container
```

In this scenario, each init container uses a request memory of 1Gi (the max limit of the limit range), and each container uses a request memory of 500Mi. Thus, the total memory request for the pod is 2Gi.

If the same limit range is used with a task of ten steps, the final memory request is 5Gi, which is higher than what each step actually needs, that is 500Mi (since each step runs after the other).

Thus, to reduce resource consumption of resources, you can:

- Reduce the number of steps in a given task by grouping different steps into one bigger step, using the script feature, and the same image. This reduces the minimum requested resource.
- Distribute steps that are relatively independent of each other and can run on their own to multiple tasks instead of a single task. This lowers the number of steps in each task, making the request for each task smaller, and the scheduler can then run them when the resources are available.

3.10.3. Additional resources

- [Setting compute resource quota for OpenShift Pipelines](#)
- [Resource quotas per project](#)
- [Restricting resource consumption using limit ranges](#)
- [Resource requests and limits in Kubernetes](#)

3.11. SETTING COMPUTE RESOURCE QUOTA FOR OPENSIFT PIPELINES

A **ResourceQuota** object in Red Hat OpenShift Pipelines controls the total resource consumption per namespace. You can use it to limit the quantity of objects created in a namespace, based on the type of the object. In addition, you can specify a compute resource quota to restrict the total amount of compute resources consumed in a namespace.

However, you might want to limit the amount of compute resources consumed by pods resulting from a pipeline run, rather than setting quotas for the entire namespace. Currently, Red Hat OpenShift Pipelines does not enable you to directly specify the compute resource quota for a pipeline.

3.11.1. Alternative approaches for limiting compute resource consumption in OpenShift Pipelines

To attain some degree of control over the usage of compute resources by a pipeline, consider the following alternative approaches:

- Set resource requests and limits for each step in a task.

Example: Set resource requests and limits for each step in a task.

```
...
spec:
  steps:
    - name: step-with-limits
      resources:
        requests:
          memory: 1Gi
          cpu: 500m
        limits:
          memory: 2Gi
          cpu: 800m
...
```

- Set resource limits by specifying values for the **LimitRange** object. For more information on **LimitRange**, refer to [Restrict resource consumption with limit ranges](#).
- [Reduce pipeline resource consumption](#).
- Set and manage [resource quotas per project](#).
- Ideally, the compute resource quota for a pipeline should be same as the total amount of compute resources consumed by the concurrently running pods in a pipeline run. However, the pods running the tasks consume compute resources based on the use case. For example, a Maven build task might require different compute resources for different applications that it builds. As a result, you cannot predetermine the compute resource quotas for tasks in a generic pipeline. For greater predictability and control over usage of compute resources, use customized pipelines for different applications.

NOTE

When using Red Hat OpenShift Pipelines in a namespace configured with a **ResourceQuota** object, the pods resulting from task runs and pipeline runs might fail with an error, such as: **failed quota: <quota name> must specify cpu, memory**.

To avoid this error, do any one of the following:

- (Recommended) Specify a limit range for the namespace.
- Explicitly define requests and limits for all containers.

For more information, refer to the [issue](#) and the [resolution](#).

If your use case is not addressed by these approaches, you can implement a workaround by using a resource quota for a priority class.

3.11.2. Specifying pipelines resource quota using priority class

A **PriorityClass** object maps priority class names to the integer values that indicates their relative priorities. Higher values increase the priority of a class. After you create a priority class, you can create pods that specify the priority class name in their specifications. In addition, you can control a pod's consumption of system resources based on the pod's priority.

Specifying resource quota for a pipeline is similar to setting a resource quota for the subset of pods created by a pipeline run. The following steps provide an example of the workaround by specifying resource quota based on priority class.

Procedure

1. Create a priority class for a pipeline.

Example: Priority class for a pipeline

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: pipeline1-pc
  value: 1000000
  description: "Priority class for pipeline1"
```

2. Create a resource quota for a pipeline.

Example: Resource quota for a pipeline

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: pipeline1-rq
spec:
  hard:
    cpu: "1000"
    memory: 200Gi
    pods: "10"
  scopeSelector:
    matchExpressions:
      - operator: In
        scopeName: PriorityClass
        values: ["pipeline1-pc"]
```

3. Verify the resource quota usage for the pipeline.

Example: Verify resource quota usage for the pipeline

```
$ oc describe quota
```

Sample output

```

Name:      pipeline1-rq
Namespace: default
Resource   Used   Hard
-----
cpu        0      1k
memory     0      200Gi
pods       0      10

```

Because pods are not running, the quota is unused.

4. Create the pipelines and tasks.

Example: YAML for the pipeline

```

apiVersion: tekton.dev/v1alpha1
kind: Pipeline
metadata:
  name: maven-build
spec:
  workspaces:
    - name: local-maven-repo
  resources:
    - name: app-git
      type: git
  tasks:
    - name: build
      taskRef:
        name: mvn
      resources:
        inputs:
          - name: source
            resource: app-git
      params:
        - name: GOALS
          value: ["package"]
    - name: maven-repo
      workspace: local-maven-repo
    - name: int-test
      taskRef:
        name: mvn
      runAfter: ["build"]
      resources:
        inputs:
          - name: source
            resource: app-git
      params:
        - name: GOALS
          value: ["verify"]
    - name: maven-repo
      workspace: local-maven-repo
    - name: gen-report
      taskRef:
        name: mvn

```

```

runAfter: ["build"]
resources:
  inputs:
    - name: source
      resource: app-git
  params:
    - name: GOALS
      value: ["site"]
  workspaces:
    - name: maven-repo
      workspace: local-maven-repo

```

Example: YAML for a task in the pipeline

```

apiVersion: tekton.dev/v1alpha1
kind: Task
metadata:
  name: mvn
spec:
  workspaces:
    - name: maven-repo
  inputs:
    params:
      - name: GOALS
        description: The Maven goals to run
        type: array
        default: ["package"]
  resources:
    - name: source
      type: git
  steps:
    - name: mvn
      image: gcr.io/cloud-builders/mvn
      workingDir: /workspace/source
      command: ["/usr/bin/mvn"]
      args:
        - -Dmaven.repo.local=$(workspaces.maven-repo.path)
        - "${inputs.params.GOALS}"
      priorityClassName: pipeline1-pc

```



NOTE

Ensure that all tasks in the pipeline belongs to the same priority class.

5. Create and start the pipeline run.

Example: YAML for a pipeline run

```

apiVersion: tekton.dev/v1alpha1
kind: PipelineRun
metadata:
  generateName: petclinic-run-
spec:
  pipelineRef:

```

```

name: maven-build
resources:
- name: app-git
  resourceSpec:
    type: git
    params:
    - name: url
      value: https://github.com/spring-projects/spring-petclinic

```

6. After the pods are created, verify the resource quota usage for the pipeline run.

Example: Verify resource quota usage for the pipeline

```
$ oc describe quota
```

Sample output

```

Name:      pipeline1-rq
Namespace: default
Resource   Used Hard
-----
cpu        500m 1k
memory     10Gi 200Gi
pods       1   10

```

The output indicates that you can manage the combined resource quota for all concurrent running pods belonging to a priority class, by specifying the resource quota per priority class.

3.11.3. Additional resources

- [Resource quotas in Kubernetes](#)
- [Limit ranges in Kubernetes](#)
- [Resource requests and limits in Kubernetes](#)

3.12. AUTOMATIC PRUNING OF TASK RUN AND PIPELINE RUN

Stale **TaskRun** and **PipelineRun** objects and their executed instances occupy physical resources that can be used for the active runs. To prevent this waste, Red Hat OpenShift Pipelines provides annotations that cluster administrators can use to automatically prune the unused objects and their instances in various namespaces.



NOTE

- Starting with Red Hat OpenShift Pipelines 1.6, auto-pruning is enabled by default.
- Configuring automatic pruning by specifying annotations affects the entire namespace. You cannot selectively auto-prune individual task runs and pipeline runs in a namespace.

3.12.1. Default pruner configuration

The default configuration for periodic pruning of resources associated with pipeline runs is as follows:

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
...
spec:
  pruner:
    keep: 100
    resources:
      - pipelinerun
    schedule: 0 8 * * *
  ...
```

You can override the default pruner configuration for a namespace by using annotations on namespace.

3.12.2. Annotations for automatically pruning task runs and pipeline runs

To automatically prune the resources associated with task runs and pipeline runs in a namespace, you can set the following annotations in the namespace:

- **operator.tekton.dev/prune.schedule:** If the value of this annotation is different from the value specified in the **TektonConfig** custom resource definition, a new cron job in that namespace is created.
- **operator.tekton.dev/prune.skip:** When set to **true**, the namespace for which it is configured is not pruned.
- **operator.tekton.dev/prune.resources:** This annotation accepts a comma-separated list of resources. To prune a single resource such as a pipeline run, set this annotation to **"pipelinerun"**. To prune multiple resources, such as task run and pipeline run, set this annotation to **"taskrun, pipelinerun"**.
- **operator.tekton.dev/prune.keep:** Use this annotation to retain a resource without pruning.
- **operator.tekton.dev/prune.keep-since:** Use this annotation to retain resources based on their age. The value for this annotation must be equal to the age of the resource in minutes. For example, to retain resources which were created not more than five days ago, set **keep-since** to **7200**.



NOTE

The **keep** and **keep-since** annotations are mutually exclusive. For any resource, you must configure only one of them.

- **operator.tekton.dev/prune.strategy:** Set the value of this annotation to either **keep** or **keep-since**.

For example, consider the following annotations that retain all task runs and pipeline runs created in the last five days, and deletes the older resources:

Example of auto-pruning annotations

...

```

annotations:
  operator.tekton.dev/prune.resources: "taskrun, pipelinerun"
  operator.tekton.dev/prune.keep-since: 7200
...

```

3.12.3. Additional resources

- For information on manual pruning of various objects, see [Pruning objects to reclaim resources](#).

3.13. USING PODS IN A PRIVILEGED SECURITY CONTEXT

The default configuration of OpenShift Pipelines 1.3.x and later versions does not allow you to run pods with privileged security context, if the pods result from pipeline run or task run. For such pods, the default service account is **pipeline**, and the security context constraint (SCC) associated with the **pipeline** service account is **pipelines-scc**. The **pipelines-scc** SCC is similar to the **anyuid** SCC, but with minor differences as defined in the YAML file for the SCC of pipelines:

Example pipelines-scc.yaml snippet

```

apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
...
allowedCapabilities:
  - SETFCAP
...
fsGroup:
  type: MustRunAs
...

```

In addition, the **Buildah** cluster task, shipped as part of the OpenShift Pipelines, uses **vfs** as the default storage driver.

3.13.1. Running pipeline run and task run pods with privileged security context

Procedure

To run a pod (resulting from pipeline run or task run) with the **privileged** security context, do the following modifications:

- Configure the associated user account or service account to have an explicit SCC. You can perform the configuration using any of the following methods:
 - Run the following command:


```
$ oc adm policy add-scc-to-user <scc-name> -z <service-account-name>
```
 - Alternatively, modify the YAML files for **RoleBinding**, and **Role** or **ClusterRole**:

Example RoleBinding object

```

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: service-account-name 1

```



```

namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pipelines-scc-clusterrole ❷
subjects:
- kind: ServiceAccount
  name: pipeline
  namespace: default

```

- ❶ Substitute with an appropriate service account name.
- ❷ Substitute with an appropriate cluster role based on the role binding you use.

Example ClusterRole object

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pipelines-scc-clusterrole ❶
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - nonroot
  resources:
  - securitycontextconstraints
  verbs:
  - use

```

- ❶ Substitute with an appropriate cluster role based on the role binding you use.



NOTE

As a best practice, create a copy of the default YAML files and make changes in the duplicate file.

- If you do not use the **vfs** storage driver, configure the service account associated with the task run or the pipeline run to have a privileged SCC, and set the security context as **privileged: true**.

3.13.2. Running pipeline run and task run by using a custom SCC and a custom service account

When using the **pipelines-scc** security context constraint (SCC) associated with the default **pipelines** service account, the pipeline run and task run pods may face timeouts. This happens because in the default **pipelines-scc** SCC, the **fsGroup.type** parameter is set to **MustRunAs**.



NOTE

For more information about pod timeouts, see [BZ#1995779](#).

To avoid pod timeouts, you can create a custom SCC with the **fsGroup.type** parameter set to **RunAsAny**, and associate it with a custom service account.



NOTE

As a best practice, use a custom SCC and a custom service account for pipeline runs and task runs. This approach allows greater flexibility and does not break the runs when the defaults are modified during an upgrade.

Procedure

1. Define a custom SCC with the **fsGroup.type** parameter set to **RunAsAny**:

Example: Custom SCC

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: my-scc is a close replica of anyuid scc. pipelines-scc has
fsGroup - RunAsAny.
  name: my-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups:
  - system:cluster-admins
priority: 10
readOnlyRootFilesystem: false
requiredDropCapabilities:
  - MKNOD
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
  - configMap
  - downwardAPI
  - emptyDir
  - persistentVolumeClaim
  - projected
  - secret
```

2. Create the custom SCC:

Example: Create the my-scc SCC

Example: Create the my-scc SCC

```
$ oc create -f my-scc.yaml
```

3. Create a custom service account:

Example: Create a fsgroup-runasany service account

```
$ oc create serviceaccount fsgroup-runasany
```

4. Associate the custom SCC with the custom service account:

Example: Associate the my-scc SCC with the fsgroup-runasany service account

```
$ oc adm policy add-scc-to-user my-scc -z fsgroup-runasany
```

If you want to use the custom service account for privileged tasks, you can associate the **privileged** SCC with the custom service account by running the following command:

Example: Associate the privileged SCC with the fsgroup-runasany service account

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

5. Use the custom service account in the pipeline run and task run:

Example: Pipeline run YAML with fsgroup-runasany custom service account

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: <pipeline-run-name>
spec:
  pipelineRef:
    name: <pipeline-cluster-task-name>
  serviceAccountName: 'fsgroup-runasany'
```

Example: Task run YAML with fsgroup-runasany custom service account

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: <task-run-name>
spec:
  taskRef:
    name: <cluster-task-name>
  serviceAccountName: 'fsgroup-runasany'
```

3.13.3. Additional resources

- For information on managing SCCs, refer to [Managing security context constraints](#).

3.14. SECURING WEBHOOKS WITH EVENT LISTENERS

As an administrator, you can secure webhooks with event listeners. After creating a namespace, you enable HTTPS for the **EventListener** resource by adding the **operator.tekton.dev/enable-annotation=enabled** label to the namespace. Then, you create a **Trigger** resource and a secured route using the re-encrypted TLS termination.

Triggers in Red Hat OpenShift Pipelines support insecure HTTP and secure HTTPS connections to the **EventListener** resource. HTTPS secures connections within and outside the cluster.

Red Hat OpenShift Pipelines runs a **tekton-operator-proxy-webhook** pod that watches for the labels in the namespace. When you add the label to the namespace, the webhook sets the **service.beta.openshift.io/serving-cert-secret-name=<secret_name>** annotation on the **EventListener** object. This, in turn, creates secrets and the required certificates.

```
service.beta.openshift.io/serving-cert-secret-name=<secret_name>
```

In addition, you can mount the created secret into the **EventListener** pod to secure the request.

3.14.1. Providing secure connection with OpenShift routes

To create a route with the re-encrypted TLS termination, run:

```
$ oc create route reencrypt --service=<svc-name> --cert=tls.crt --key=tls.key --ca-cert=ca.crt --hostname=<hostname>
```

Alternatively, you can create a re-encrypted TLS termination YAML file to create a secure route.

Example re-encrypt TLS termination YAML to create a secure route

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured ❶
spec:
  host: <hostname>
  to:
    kind: Service
    name: frontend ❷
  tls:
    termination: reencrypt ❸
    key: [as in edge termination]
    certificate: [as in edge termination]
    caCertificate: [as in edge termination]
    destinationCACertificate: |- ❹
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

❶ ❷ The name of the object, which is limited to only 63 characters.

❸ The termination field is set to **reencrypt**. This is the only required TLS field.

❹ This is required for re-encryption. The **destinationCACertificate** field specifies a CA certificate to validate the endpoint certificate, thus securing the connection from the router to the destination pods. You can omit this field in either of the following scenarios:

- The service uses a service signing certificate.
- The administrator specifies a default CA certificate for the router, and the service has a certificate signed by that CA.

You can run the **oc create route reencrypt --help** command to display more options.

3.14.2. Creating a sample EventListener resource using a secure HTTPS connection

This section uses the [pipelines-tutorial](#) example to demonstrate creation of a sample EventListener resource using a secure HTTPS connection.

Procedure

1. Create the **TriggerBinding** resource from the YAML file available in the pipelines-tutorial repository:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/03_triggers/01_binding.yaml
```

2. Create the **TriggerTemplate** resource from the YAML file available in the pipelines-tutorial repository:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/03_triggers/02_template.yaml
```

3. Create the **Trigger** resource directly from the pipelines-tutorial repository:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/03_triggers/03_trigger.yaml
```

4. Create an **EventListener** resource using a secure HTTPS connection:

- a. Add a label to enable the secure HTTPS connection to the **EventListener** resource:

```
$ oc label namespace <ns-name> operator.tekton.dev/enable-annotation=enabled
```

- b. Create the **EventListener** resource from the YAML file available in the pipelines-tutorial repository:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/03_triggers/04_event_listener.yaml
```

- c. Create a route with the re-encrypted TLS termination:

```
$ oc create route reencrypt --service=<svc-name> --cert=tls.crt --key=tls.key --ca-cert=ca.crt --hostname=<hostname>
```

3.15. AUTHENTICATING PIPELINES USING GIT SECRET

A Git secret consists of credentials to securely interact with a Git repository, and is often used to automate authentication. In Red Hat OpenShift Pipelines, you can use Git secrets to authenticate pipeline runs and task runs that interact with a Git repository during execution.

A pipeline run or a task run gains access to the secrets through the associated service account. Pipelines support the use of Git secrets as annotations (key-value pairs) for basic authentication and SSH-based authentication.

3.15.1. Credential selection

A pipeline run or task run might require multiple authentications to access different Git repositories. Annotate each secret with the domains where Pipelines can use its credentials.

A credential annotation key for Git secrets must begin with **tekton.dev/git-**, and its value is the URL of the host for which you want Pipelines to use that credential.

In the following example, Pipelines uses a **basic-auth** secret, which relies on a username and password, to access repositories at **github.com** and **gitlab.com**.

Example: Multiple credentials for basic authentication

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    tekton.dev/git-0: github.com
    tekton.dev/git-1: gitlab.com
type: kubernetes.io/basic-auth
stringData:
  username: 1
  password: 2
```

- 1 Username for the repository
- 2 Password or personal access token for the repository

You can also use an **ssh-auth** secret (private key) to access a Git repository.

Example: Private key for SSH based authentication

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    tekton.dev/git-0: https://github.com
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: 1
```

- 1 Name of the file containing the SSH private key string.

3.15.2. Configuring basic authentication for Git

For a pipeline to retrieve resources from password-protected repositories, you must configure the basic authentication for that pipeline.

To configure basic authentication for a pipeline, update the **secret.yaml**, **serviceaccount.yaml**, and **run.yaml** files with the credentials from the Git secret for the specified repository. When you complete this process, Pipelines can use that information to retrieve the specified pipeline resources.



NOTE

For GitHub, authentication using plain password is deprecated. Instead, use a [personal access token](#).

Procedure

1. In the **secret.yaml** file, specify the username and password or [GitHub personal access token](#) to access the target Git repository.

```
apiVersion: v1
kind: Secret
metadata:
  name: basic-user-pass ❶
  annotations:
    tekton.dev/git-0: https://github.com
type: kubernetes.io/basic-auth
stringData:
  username: ❷
  password: ❸
```

- ❶ Name of the secret. In this example, **basic-user-pass**.
- ❷ Username for the Git repository.
- ❸ Password for the Git repository.

2. In the **serviceaccount.yaml** file, associate the secret with the appropriate service account.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-bot ❶
secrets:
  - name: basic-user-pass ❷
```

- ❶ Name of the service account. In this example, **build-bot**.
- ❷ Name of the secret. In this example, **basic-user-pass**.

3. In the **run.yaml** file, associate the service account with a task run or a pipeline run.

- Associate the service account with a task run:

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
```

```

metadata:
  name: build-push-task-run-2 1
spec:
  serviceAccountName: build-bot 2
  taskRef:
    name: build-push 3

```

- 1** Name of the task run. In this example, **build-push-task-run-2**.
- 2** Name of the service account. In this example, **build-bot**.
- 3** Name of the task. In this example, **build-push**.

- Associate the service account with a **PipelineRun** resource:

```

apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: demo-pipeline 1
  namespace: default
spec:
  serviceAccountName: build-bot 2
  pipelineRef:
    name: demo-pipeline 3

```

- 1** Name of the pipeline run. In this example, **demo-pipeline**.
- 2** Name of the service account. In this example, **build-bot**.
- 3** Name of the pipeline. In this example, **demo-pipeline**.

4. Apply the changes.

```
$ oc apply --filename secret.yaml,serviceaccount.yaml,run.yaml
```

3.15.3. Configuring SSH authentication for Git

For a pipeline to retrieve resources from repositories configured with SSH keys, you must configure the SSH-based authentication for that pipeline.

To configure SSH-based authentication for a pipeline, update the **secret.yaml**, **serviceaccount.yaml**, and **run.yaml** files with the credentials from the SSH private key for the specified repository. When you complete this process, Pipelines can use that information to retrieve the specified pipeline resources.



NOTE

Consider using SSH-based authentication rather than basic authentication.

Procedure

1. Generate an [SSH private key](#), or copy an existing private key, which is usually available in the `~/.ssh/id_rsa` file.

2. In the **secret.yaml** file, set the value of **ssh-privatekey** to the name of the SSH private key file, and set the value of **known_hosts** to the name of the known hosts file.

```
apiVersion: v1
kind: Secret
metadata:
  name: ssh-key ❶
  annotations:
    tekton.dev/git-0: github.com
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: ❷
  known_hosts: ❸
```

- ❶ Name of the secret containing the SSH private key. In this example, **ssh-key**.
- ❷ Name of the file containing the SSH private key string.
- ❸ Name of the file containing a list of known hosts.

CAUTION

If you omit the private key, Pipelines accepts the public key of any server.

3. Optional: To specify a custom SSH port, add **:<port number>** to the end of the **annotation** value. For example, **tekton.dev/git-0: github.com:2222**.
4. In the **serviceaccount.yaml** file, associate the **ssh-key** secret with the **build-bot** service account.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-bot ❶
secrets:
  - name: ssh-key ❷
```

- ❶ Name of the service account. In this example, **build-bot**.
- ❷ Name of the secret containing the SSH private key. In this example, **ssh-key**.

5. In the **run.yaml** file, associate the service account with a task run or a pipeline run.

- Associate the service account with a task run:

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: build-push-task-run-2 ❶
spec:
```

```
serviceAccountName: build-bot ❷
taskRef:
  name: build-push ❸
```

- ❶ Name of the task run. In this example, **build-push-task-run-2**.
- ❷ Name of the service account. In this example, **build-bot**.
- ❸ Name of the task. In this example, **build-push**.

- Associate the service account with a pipeline run:

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: demo-pipeline ❶
  namespace: default
spec:
  serviceAccountName: build-bot ❷
  pipelineRef:
    name: demo-pipeline ❸
```

- ❶ Name of the pipeline run. In this example, **demo-pipeline**.
- ❷ Name of the service account. In this example, **build-bot**.
- ❸ Name of the pipeline. In this example, **demo-pipeline**.

6. Apply the changes.

```
$ oc apply --filename secret.yaml,serviceaccount.yaml,run.yaml
```

3.15.4. Using SSH authentication in git type tasks

When invoking Git commands, you can use SSH authentication directly in the steps of a task. SSH authentication ignores the **\$HOME** variable and only uses the user's home directory specified in the **/etc/passwd** file. So each step in a task must symlink the **/tekton/home/.ssh** directory to the home directory of the associated user.

However, explicit symlinks are not necessary when you use a pipeline resource of the **git** type, or the **git-clone** task available in the Tekton catalog.

As an example of using SSH authentication in **git** type tasks, refer to [authenticating-git-commands.yaml](#).

3.15.5. Using secrets as a non-root user

You might need to use secrets as a non-root user in certain scenarios, such as:

- The users and groups that the containers use to execute runs are randomized by the platform.
- The steps in a task define a non-root security context.
- A task specifies a global non-root security context, which applies to all steps in a task.

In such scenarios, consider the following aspects of executing task runs and pipeline runs as a non-root user:

- SSH authentication for Git requires the user to have a valid home directory configured in the **/etc/passwd** directory. Specifying a UID that has no valid home directory results in authentication failure.
- SSH authentication ignores the **\$HOME** environment variable. So you must or symlink the appropriate secret files from the **\$HOME** directory defined by Pipelines (**/tekton/home**), to the non-root user's valid home directory.

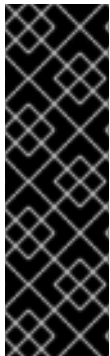
In addition, to configure SSH authentication in a non-root security context, refer to the [example for authenticating git commands](#).

3.15.6. Limiting secret access to specific steps

By default, the secrets for Pipelines are stored in the **\$HOME/tekton/home** directory, and are available for all the steps in a task.

To limit a secret to specific steps, use the secret definition to specify a volume, and mount the volume in specific steps.

3.16. USING TEKTON CHAINS FOR OPENSIFT PIPELINES SUPPLY CHAIN SECURITY



IMPORTANT

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

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

Tekton Chains is a Kubernetes Custom Resource Definition (CRD) controller. You can use it to manage the supply chain security of the tasks and pipelines created using Red Hat OpenShift Pipelines.

By default, Tekton Chains observes all task run executions in your OpenShift Container Platform cluster. When the task runs complete, Tekton Chains takes a snapshot of the task runs. It then converts the snapshot to one or more standard payload formats, and finally signs and stores all artifacts.

To capture information about task runs, Tekton Chains uses the **Result** and **PipelineResource** objects. When the objects are unavailable, Tekton Chains the URLs and qualified digests of the OCI images.



NOTE

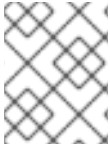
The **PipelineResource** object is deprecated and will be removed in a future release; for manual use, the **Results** object is recommended.

3.16.1. Key features

- You can sign task runs, task run results, and OCI registry images with cryptographic key types and services such as **cosign**.
- You can use attestation formats such as **in-toto**.
- You can securely store signatures and signed artifacts using OCI repository as a storage backend.

3.16.2. Installing Tekton Chains using the Red Hat OpenShift Pipelines Operator

Cluster administrators can use the **TektonChain** custom resource (CR) to install and manage Tekton Chains.



NOTE

Tekton Chains is an optional component of Red Hat OpenShift Pipelines. Currently, you cannot install it using the **TektonConfig** CR.

Prerequisites

- Ensure that the Red Hat OpenShift Pipelines Operator is installed in the **openshift-pipelines** namespace on your cluster.

Procedure

1. Create the **TektonChain** CR for your OpenShift Container Platform cluster.

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonChain
metadata:
  name: chain
spec:
  targetNamespace: openshift-pipelines
```

2. Apply the **TektonChain** CR.

```
$ oc apply -f TektonChain.yaml 1
```

- 1** Substitute with the file name of the **TektonChain** CR.

3. Check the status of the installation.

```
$ oc get tektonchains.operator.tekton.dev
```

3.16.3. Configuring Tekton Chains

Tekton Chains uses a **ConfigMap** object named **chains-config** in the **openshift-pipelines** namespace for configuration.

To configure Tekton Chains, use the following example:

Example: Configuring Tekton Chains

```
$ oc patch configmap chains-config -n openshift-pipelines -p='{ "data": { "artifacts.oci.storage": "",  
"artifacts.taskrun.format": "tekton", "artifacts.taskrun.storage": "tekton" } }'
```

- 1 Use a combination of supported key-value pairs in the JSON payload.

3.16.3.1. Supported keys for Tekton Chains configuration

Cluster administrators can use various supported keys and values to configure specifications about task runs, OCI images, and storage.

3.16.3.1.1. Supported keys for task run

Table 3.12. Chains configuration: Supported keys for task run

Supported keys	Description	Supported values	Default values
artifacts.taskrun.format	The format to store task run payloads.	tekton, in-toto	tekton
artifacts.taskrun.storage	The storage backend for task run signatures. You can specify multiple backends as a comma-separated list, such as "tekton,oci" . To disable this artifact, provide an empty string " " .	tekton, oci	tekton
artifacts.taskrun.signer	The signature backend to sign task run payloads.	x509	x509

3.16.3.1.2. Supported keys for OCI

Table 3.13. Chains configuration: Supported keys for OCI

Supported keys	Description	Supported values	Default values
artifacts.oci.format	The format to store OCI payloads.	simplesigning	simplesigning

Supported keys	Description	Supported values	Default values
artifacts.oci.storage	The storage backend to for OCI signatures. You can specify multiple backends as a comma-separated list, such as “ oci,tekton ”. To disable the OCI artifact, provide an empty string “”.	tekton, oci	oci
artifacts.oci.signer	The signature backend to sign OCI payloads.	x509, cosign	x509

3.16.3.1.3. Supported keys for storage

Table 3.14. Chains configuration: Supported keys for storage

Supported keys	Description	Supported values	Default values
artifacts.oci.repository	The OCI repository to store OCI signatures.	Currently, Chains support only the internal OpenShift OCI registry; other popular options such as Quay is not supported.	

3.16.4. Signing secrets in Tekton Chains

Cluster administrators can generate a key pair and use Tekton Chains to sign artifacts using a Kubernetes secret. For Tekton Chains to work, a private key and a password for encrypted keys must exist as part of the **signing-secrets** Kubernetes secret, in the **openshift-pipelines** namespace.

Currently, Tekton Chains supports the **x509** and **cosign** signature schemes.



NOTE

Use only one of the supported signature schemes.

3.16.4.1. Signing using x509

To use the **x509** signing scheme with Tekton Chains, store the **x509.pem** private key of the **ed25519** or **ecdsa** type in the **signing-secrets** Kubernetes secret. Ensure that the key is stored as an unencrypted PKCS8 PEM file (**BEGIN PRIVATE KEY**).

3.16.4.2. Signing using cosign

To use the **cosign** signing scheme with Tekton Chains:

1. Install [cosign](#).

2. Generate the **cosign.key** and **cosign.pub** key pairs.

```
$ cosign generate-key-pair k8s://openshift-pipelines/signing-secrets
```

Cosign prompts you for a password, and creates a Kubernetes secret.

3. Store the encrypted **cosign.key** private key and the **cosign.password** decryption password in the **signing-secrets** Kubernetes secret. Ensure that the private key is stored as an encrypted PEM file of the **ENCRYPTED COSIGN PRIVATE KEY** type.

3.16.4.3. Troubleshooting signing

If the signing secrets are already populated, you might get the following error:

```
Error from server (AlreadyExists): secrets "signing-secrets" already exists
```

To resolve the error:

1. Delete the secrets:

```
$ oc delete secret signing-secrets -n openshift-pipelines
```

2. Recreate the key pairs and store them in the secrets using your preferred signing scheme.

3.16.5. Authenticating to an OCI registry

Before pushing signatures to an OCI registry, cluster administrators must configure Tekton Chains to authenticate with the registry. The Tekton Chains controller uses the same service account under which the task runs execute. To set up a service account with the necessary credentials for pushing signatures to an OCI registry, perform the following steps:

Procedure

1. Set the namespace and name of the Kubernetes service account.

```
$ export NAMESPACE=<namespace> ❶
$ export SERVICE_ACCOUNT_NAME=<service_account> ❷
```

- ❶ The namespace associated with the service account.
- ❷ The name of the service account.

2. Create a Kubernetes secret.

```
$ oc create secret registry-credentials \
  --from-file=.dockerconfigjson \ ❶
  --type=kubernetes.io/dockerconfigjson \
  -n $NAMESPACE
```

- ❶ Substitute with the path to your Docker config file. Default path is **~/docker/config.json**.

3. Give the service account access to the secret.

```
$ oc patch serviceaccount $SERVICE_ACCOUNT_NAME \
-p "{\"imagePullSecrets\": [{\"name\": \"registry-credentials\"}]}" -n $NAMESPACE
```

If you patch the default **pipeline** service account that Red Hat OpenShift Pipelines assigns to all task runs, the Red Hat OpenShift Pipelines Operator will override the service account. As a best practice, you can perform the following steps:

- a. Create a separate service account to assign to user's task runs.

```
$ oc create serviceaccount <service_account_name>
```

- b. Associate the service account to the task runs by setting the value of the **serviceaccountname** field in the task run template.

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: build-push-task-run-2
spec:
  serviceAccountName: build-bot 1
  taskRef:
    name: build-push
  ...
```

- 1** Substitute with the name of the newly created service account.

3.16.5.1. Creating and verifying task run signatures without any additional authentication

To verify signatures of task runs using Tekton Chains with any additional authentication, perform the following tasks:

- Create an encrypted x509 key pair and save it as a Kubernetes secret.
- Configure the Tekton Chains backend storage.
- Create a task run, sign it, and store the signature and the payload as annotations on the task run itself.
- Retrieve the signature and payload from the signed task run.
- Verify the signature of the task run.

Prerequisites

Ensure that the following are installed on the cluster:

- Red Hat OpenShift Pipelines Operator
- Tekton Chains
- [Cosign](#)

Procedure

1. Create an encrypted x509 key pair and save it as a Kubernetes secret:

```
$ cosign generate-key-pair k8s://openshift-pipelines/signing-secrets
```

Provide a password when prompted. Cosign stores the resulting private key as part of the **signing-secrets** Kubernetes secret in the **openshift-pipelines** namespace.

2. In the Tekton Chains configuration, disable the OCI storage, and set the task run storage and format to **tekton**.

```
$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":{"artifacts.oci.storage":"","artifacts.taskrun.format":"tekton","artifacts.taskrun.storage":"tekton"}}'
```

3. Restart the Tekton Chains controller to ensure that the modified configuration is applied.

```
$ oc delete po -n openshift-pipelines -l app=tekton-chains-controller
```

4. Create a task run.

```
$ oc create -f
https://raw.githubusercontent.com/tektoncd/chains/main/examples/taskruns/task-output-
image.yaml 1
taskrun.tekton.dev/build-push-run-output-image-qbjvh created
```

- 1** Substitute with the URI or file path pointing to your task run.

5. Check the status of the steps, and wait till the process finishes.

```
$ tkn tr describe --last
[...truncated output...]
NAME                                STATUS
· create-dir-builtimage-9467f    Completed
· git-source-sourcerepo-p2sk8    Completed
· build-and-push                  Completed
· echo                            Completed
· image-digest-exporter-xlkn7     Completed
```

6. Retrieve the signature and payload from the object stored as **base64** encoded annotations:

```
$ export TASKRUN_UID=$(tkn tr describe --last -o jsonpath='{.metadata.uid}')
$ tkn tr describe --last -o jsonpath="{.metadata.annotations.chains\tekton\dev/signature-
taskrun-$TASKRUN_UID}" > signature
$ tkn tr describe --last -o jsonpath="{.metadata.annotations.chains\tekton\dev/payload-
taskrun-$TASKRUN_UID}" | base64 -d > payload
```

7. Verify the signature.

```
$ cosign verify-blob --key k8s://openshift-pipelines/signing-secrets --signature ./signature
./payload
Verified OK
```

3.16.6. Using Tekton Chains to sign and verify image and provenance

Cluster administrators can use Tekton Chains to sign and verify images and provenances, by performing the following tasks:

- Create an encrypted x509 key pair and save it as a Kubernetes secret.
- Set up authentication for the OCI registry to store images, image signatures, and signed image attestations.
- Configure Tekton Chains to generate and sign provenance.
- Create an image with Kaniko in a task run.
- Verify the signed image and the signed provenance.

Prerequisites

Ensure that the following are installed on the cluster:

- Red Hat OpenShift Pipelines Operator
- Tekton Chains
- [Cosign](#)
- [Rekor](#)
- [jq](#)

Procedure

1. Create an encrypted x509 key pair and save it as a Kubernetes secret:

```
$ cosign generate-key-pair k8s://openshift-pipelines/signing-secrets
```

Provide a password when prompted. Cosign stores the resulting private key as part of the **signing-secrets** Kubernetes secret in the **openshift-pipelines** namespace, and writes the public key to the **cosign.pub** local file.

2. Configure authentication for the image registry.
 - a. To configure the Tekton Chains controller for pushing signature to an OCI registry, use the credentials associated with the service account of the task run. For detailed information, see the "Authenticating to an OCI registry" section.
 - b. To configure authentication for a Kaniko task that builds and pushes image to the registry, create a Kubernetes secret of the docker **config.json** file containing the required credentials.

```
$ oc create secret generic <docker_config_secret_name> \ 1
--from-file <path_to_config.json> 2
```

1 Substitute with the name of the docker config secret.

2 Substitute with the path to docker **config.json** file.

3. Configure Tekton Chains by setting the **artifacts.taskrun.format**, **artifacts.taskrun.storage**, and **transparency.enabled** parameters in the **chains-config** object:

```
$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":
{"artifacts.taskrun.format": "in-toto"}}'

$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":
{"artifacts.taskrun.storage": "oci"}}'

$ oc patch configmap chains-config -n openshift-pipelines -p='{"data":
{"transparency.enabled": "true"}}'
```

4. Start the Kaniko task.

- a. Apply the Kaniko task to the cluster.

```
$ oc apply -f examples/kaniko/kaniko.yaml ❶
```

- ❶ Substitute with the URI or file path to your Kaniko task.

- b. Set the appropriate environment variables.

```
$ export REGISTRY=<url_of_registry> ❶

$ export DOCKERCONFIG_SECRET_NAME=
<name_of_the_secret_in_docker_config_json> ❷
```

- ❶ Substitute with the URL of the registry where you want to push the image.

- ❷ Substitute with the name of the secret in the docker **config.json** file.

- c. Start the Kaniko task.

```
$ tkn task start --param IMAGE=$REGISTRY/kaniko-chains --use-param-defaults --
workspace name=source,emptyDir="" --workspace
name=dockerconfig,secret=$DOCKERCONFIG_SECRET_NAME kaniko-chains
```

Observe the logs of this task until all steps are complete. On successful authentication, the final image will be pushed to **\$REGISTRY/kaniko-chains**.

5. Wait for a minute to allow Tekton Chains to generate the provenance and sign it, and then check the availability of the **chains.tekton.dev/signed=true** annotation on the task run.

```
$ oc get tr <task_run_name> \ ❶
-o json | jq -r .metadata.annotations

{
  "chains.tekton.dev/signed": "true",
  ...
}
```

- ❶ Substitute with the name of the task run.

6. Verify the image and the attestation.

```
$ cosign verify --key cosign.pub $REGISTRY/kaniko-chains
$ cosign verify-attestation --key cosign.pub $REGISTRY/kaniko-chains
```

7. Find the provenance for the image in Rekor.

- a. Get the digest of the \$REGISTRY/kaniko-chains image. You can search for it in the task run, or pull the image to extract the digest.
- b. Search Rekor to find all entries that match the **sha256** digest of the image.

```
$ rekor-cli search --sha <image_digest> ❶
<uuid_1> ❷
<uuid_2> ❸
...
```

- ❶ Substitute with the **sha256** digest of the image.
- ❷ The first matching universally unique identifier (UUID).
- ❸ The second matching UUID.

The search result displays UUIDs of the matching entries. One of those UUIDs holds the attestation.

c. Check the attestation.

```
$ rekor-cli get --uuid <uuid> --format json | jq -r .Attestation | base64 --decode | jq
```

3.16.7. Additional resources

- [Installing OpenShift Pipelines](#)

3.17. VIEWING PIPELINE LOGS USING THE OPENSIFT LOGGING OPERATOR

The logs generated by pipeline runs, task runs, and event listeners are stored in their respective pods. It is useful to review and analyze logs for troubleshooting and audits.

However, retaining the pods indefinitely leads to unnecessary resource consumption and cluttered namespaces.

To eliminate any dependency on the pods for viewing pipeline logs, you can use the OpenShift Elasticsearch Operator and the OpenShift Logging Operator. These Operators help you to view pipeline logs by using the [Elasticsearch Kibana](#) stack, even after you have deleted the pods that contained the logs.

3.17.1. Prerequisites

Before trying to view pipeline logs in a Kibana dashboard, ensure the following:

- The steps are performed by a cluster administrator.
- Logs for pipeline runs and task runs are available.
- The OpenShift Elasticsearch Operator and the OpenShift Logging Operator are installed.

3.17.2. Viewing pipeline logs in Kibana

To view pipeline logs in the Kibana web console:

Procedure

1. Log in to OpenShift Container Platform web console as a cluster administrator.
2. In the top right of the menu bar, click the **grid icon** → **Observability** → **Logging**. The Kibana web console is displayed.
3. Create an index pattern:
 - a. On the left navigation panel of the **Kibana** web console, click **Management**.
 - b. Click **Create index pattern**.
 - c. Under **Step 1 of 2: Define index pattern**→ **Index pattern**, enter a ***** pattern and click **Next Step**.
 - d. Under **Step 2 of 2: Configure settings**→ **Time filter field name**, select **@timestamp** from the drop-down menu, and click **Create index pattern**.
4. Add a filter:
 - a. On the left navigation panel of the **Kibana** web console, click **Discover**.
 - b. Click **Add a filter** +→ **Edit Query DSL**.



NOTE

- For each of the example filters that follows, edit the query and click **Save**.
- The filters are applied one after another.

- i. Filter the containers related to pipelines:

Example query to filter pipelines containers

```
{
  "query": {
    "match": {
      "kubernetes.flat_labels": {
        "query": "app_kubernetes_io/managed-by=tekton-pipelines",
        "type": "phrase"
      }
    }
  }
}
```

- ii. Filter all containers that are not **place-tools** container. As an illustration of using the graphical drop-down menus instead of editing the query DSL, consider the following approach:

Figure 3.6. Example of filtering using the drop-down fields

The screenshot shows a web-based interface for adding a filter. The title bar says 'Add filter' with a close button 'X'. Below the title bar is a horizontal line. Underneath, there's a 'Filter' section with a blue link 'Edit Query DSL'. The filter is configured as 'kubernetes.container_name' (dropdown), 'is not' (operator), and 'place-tools' (text input). Below the filter section is a 'Label' section with a text input 'Optional'. At the bottom right are 'Cancel' and 'Save' buttons.

- iii. Filter **pipelinerun** in labels for highlighting:

Example query to filter **pipelinerun** in labels for highlighting

```
{
  "query": {
    "match": {
      "kubernetes.flat_labels": {
        "query": "tekton_dev/pipelineRun=",
        "type": "phrase"
      }
    }
  }
}
```

- iv. Filter **pipeline** in labels for highlighting:

Example query to filter **pipeline** in labels for highlighting

```
{
  "query": {
    "match": {
      "kubernetes.flat_labels": {
        "query": "tekton_dev/pipeline=",
        "type": "phrase"
      }
    }
  }
}
```

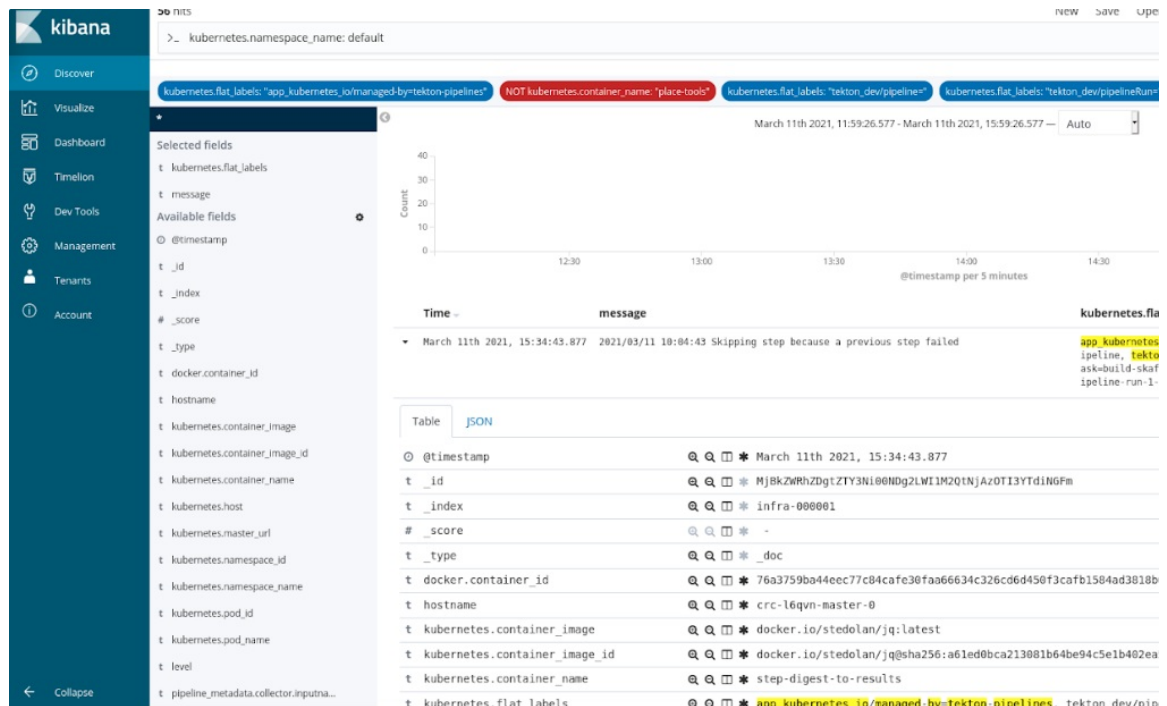
c. From the **Available fields** list, select the following fields:

- **kubernetes.flat_labels**
- **message**

Ensure that the selected fields are displayed under the **Selected fields** list.

d. The logs are displayed under the **message** field.

Figure 3.7. Filtered messages



3.17.3. Additional resources

- [Installing OpenShift Logging](#)
- [Viewing logs for a resource](#)
- [Viewing cluster logs by using Kibana](#)

3.18. UNPRIVILEGED BUILDING OF CONTAINER IMAGES USING BUILDHAH

Running Pipelines as the root user on a container can expose the container processes and the host to other potentially malicious resources. You can reduce this type of exposure by running the workload as a specific non-root user in the container. For secure unprivileged builds of container images using Buildah, you can perform the following steps:

- Define custom service account (SA) and security context constraint (SCC).
- Configure Buildah to use the **build** user with id **1000**.
- Start a task run with a custom config map, or integrate it with a pipeline run.

3.18.1. Configuring custom service account and security context constraint

The default **pipeline** SA allows using a user id outside of the namespace range. To reduce dependency on the default SA, you can define a custom SA and SCC with necessary cluster role and role bindings for the **build** user with user id **1000**.

Procedure

- Create a custom SA and SCC with necessary cluster role and role bindings.

Example: Custom SA and SCC for used id 1000

```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: pipelines-sa-userid-1000 1
---
kind: SecurityContextConstraints
metadata:
  annotations:
    name: pipelines-scc-userid-1000 2
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
groups:
- system:cluster-admins
priority: 10
readOnlyRootFilesystem: false
requiredDropCapabilities:
- MKNOD
runAsUser: 3
  type: MustRunAs
  uid: 1000
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
users: []
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole

```



```

metadata:
  name: pipelines-scc-userid-1000-clusterrole 4
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - pipelines-scc-userid-1000
  resources:
  - securitycontextconstraints
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pipelines-scc-userid-1000-rolebinding 5
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pipelines-scc-userid-1000-clusterrole
subjects:
- kind: ServiceAccount
  name: pipelines-sa-userid-1000

```

- 1 Define a custom SA.
- 2 Define a custom SCC created based on restricted privileges, with modified **runAsUser** field.
- 3 Restrict any pod that gets attached with the custom SCC through the custom SA to run as user id **1000**.
- 4 Define a cluster role that uses the custom SCC.
- 5 Bind the cluster role that uses the custom SCC to the custom SA.

3.18.2. Configuring Buildah to use build user

You can define a Buildah task to use the **build** user with user id **1000**.

Procedure

1. Create a copy of the **buildah** cluster task as an ordinary task.

```
$ tkn task create --from=buildah
```

2. Edit the copied **buildah** task.

```
$ oc edit task buildah
```

Example: Modified Buildah task with build user

```

apiVersion: tekton.dev/v1beta1
kind: Task

```

```

metadata:
  name: buildah-as-user
spec:
  description: >-
    Buildah task builds source into a container image and
    then pushes it to a container registry.
    Buildah Task builds source into a container image using Project Atomic's
    Buildah build tool. It uses Buildah's support for building from Dockerfiles,
    using its buildah bud command. This command executes the directives in the
    Dockerfile to assemble a container image, then pushes that image to a
    container registry.
  params:
    - name: IMAGE
      description: Reference of the image buildah will produce.
    - name: BUILDER_IMAGE
      description: The location of the buildah builder image.
      default:
registry.redhat.io/rhel8/buildah@sha256:99cae35f40c7ec050fed3765b2b27e0b8bbea2aa2da7
c16408e2ca13c60ff8ee
    - name: STORAGE_DRIVER
      description: Set buildah storage driver
      default: vfs
    - name: DOCKERFILE
      description: Path to the Dockerfile to build.
      default: ./Dockerfile
    - name: CONTEXT
      description: Path to the directory to use as context.
      default: .
    - name: TLSVERIFY
      description: Verify the TLS on the registry endpoint (for push/pull to a non-TLS registry)
      default: "true"
    - name: FORMAT
      description: The format of the built container, oci or docker
      default: "oci"
    - name: BUILD_EXTRA_ARGS
      description: Extra parameters passed for the build command when building images.
      default: ""
    - description: Extra parameters passed for the push command when pushing images.
      name: PUSH_EXTRA_ARGS
      type: string
      default: ""
    - description: Skip pushing the built image
      name: SKIP_PUSH
      type: string
      default: "false"
  results:
    - description: Digest of the image just built.
      name: IMAGE_DIGEST
      type: string
  workspaces:
    - name: source
  steps:
    - name: build
      securityContext:
        runAsUser: 1000 1
      image: $(params.BUILDER_IMAGE)

```

```

workingDir: $(workspaces.source.path)
script: |
  echo "Running as USER ID `id`" ❷
  buildah --storage-driver=$(params.STORAGE_DRIVER) bud \
    $(params.BUILD_EXTRA_ARGS) --format=$(params.FORMAT) \
    --tls-verify=$(params.TLSVERIFY) --no-cache \
    -f $(params.DOCKERFILE) -t $(params.IMAGE) $(params.CONTEXT)
  [[ "$(params.SKIP_PUSH)" == "true" ]] && echo "Push skipped" && exit 0
  buildah --storage-driver=$(params.STORAGE_DRIVER) push \
    $(params.PUSH_EXTRA_ARGS) --tls-verify=$(params.TLSVERIFY) \
    --digestfile $(workspaces.source.path)/image-digest $(params.IMAGE) \
    docker://$(params.IMAGE)
  cat $(workspaces.source.path)/image-digest | tee /tekton/results/IMAGE_DIGEST
volumeMounts:
- name: varlibcontainers
  mountPath: /home/build/.local/share/containers
volumeMounts:
- name: varlibcontainers
  mountPath: /home/build/.local/share/containers
volumes:
- name: varlibcontainers
  emptyDir: {}

```

- ❶ Run the container explicitly as the user id **1000**, which corresponds to the **build** user in the Buildah image.
- ❷ Display the user id to confirm that the process is running as user id **1000**.

3.18.3. Starting a task run with custom config map, or a pipeline run

After defining the custom Buildah cluster task, you can create a **TaskRun** object that builds an image as a **build** user with user id **1000**. In addition, you can integrate the **TaskRun** object as part of a **PipelineRun** object.

Procedure

1. Create a **TaskRun** object with a custom **ConfigMap** and **Dockerfile** objects.

Example: A task run that runs Buildah as user id 1000

```

apiVersion: v1
data:
  Dockerfile: |
    ARG BASE_IMG=registry.access.redhat.com/ubi8/ubi
    FROM $BASE_IMG AS buildah-runner
    RUN dnf -y update && \
      dnf -y install git && \
      dnf clean all
    CMD git
kind: ConfigMap
metadata:
  name: dockerfile ❶
---
apiVersion: tekton.dev/v1beta1

```

```

kind: TaskRun
metadata:
  name: buildah-as-user-1000
spec:
  serviceAccountName: pipelines-sa-userid-1000
  params:
    - name: IMAGE
      value: image-registry.openshift-image-registry.svc:5000/test/buildahuser
  taskRef:
    kind: Task
    name: buildah-as-user
  workspaces:
    - configMap:
        name: dockerfile 2
      name: source

```

- 1 Use a config map because the focus is on the task run, without any prior task that fetches some sources with a Dockerfile.
- 2 Mount a config map as the source workspace for the **buildah-as-user** task.

2. (Optional) Create a pipeline and a corresponding pipeline run.

Example: A pipeline and corresponding pipeline run

```

apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: pipeline-buildah-as-user-1000
spec:
  params:
    - name: IMAGE
    - name: URL
  workspaces:
    - name: shared-workspace
    - name: sslcertdir
      optional: true
  tasks:
    - name: fetch-repository 1
      taskRef:
        name: git-clone
        kind: ClusterTask
      workspaces:
        - name: output
          workspace: shared-workspace
      params:
        - name: url
          value: $(params.URL)
        - name: subdirectory
          value: ""
        - name: deleteExisting
          value: "true"
    - name: buildah
      taskRef:

```

```

    name: buildah-as-user ❷
  runAfter:
  - fetch-repository
  workspaces:
  - name: source
    workspace: shared-workspace
  - name: sslcertdir
    workspace: sslcertdir
  params:
  - name: IMAGE
    value: $(params.IMAGE)
---
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: pipelinerun-buildah-as-user-1000
spec:
  serviceAccountName: pipelines-sa-userid-1000
  params:
  - name: URL
    value: https://github.com/openshift/pipelines-vote-api
  - name: IMAGE
    value: image-registry.openshift-image-registry.svc:5000/test/buildahuser
  taskRef:
    kind: Pipeline
    name: pipeline-buildah-as-user-1000
  workspaces:
  - name: shared-workspace ❸
    volumeClaimTemplate:
      spec:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 100Mi

```

- ❶ Use the **git-clone** cluster task to fetch the source containing a Dockerfile and build it using the modified Buildah task.
- ❷ Refer to the modified Buildah task.
- ❸ Share data between the **git-clone** task and the modified Buildah task using a persistent volume claim (PVC) created automatically by the controller.

3. Start the task run or the pipeline run.

3.18.4. Limitations of unprivileged builds

The process for unprivileged builds works with most **Dockerfile** objects. However, there are some known limitations might cause a build to fail:

- Using the **--mount=type=cache** option might fail due to lack of necessary permissions issues. For more information, see this [article](#).

- Using the **--mount=type=secret** option fails because mounting resources requires additional capabilities that are not provided by the custom SCC.

Additional resources

- [Managing security context constraints \(SCCs\)](#)

CHAPTER 4. GITOPS

4.1. RED HAT OPENSIFT GITOPS RELEASE NOTES

Red Hat OpenShift GitOps is a declarative way to implement continuous deployment for cloud native applications. Red Hat OpenShift GitOps ensures consistency in applications when you deploy them to different clusters in different environments, such as: development, staging, and production. Red Hat OpenShift GitOps helps you automate the following tasks:

- Ensure that the clusters have similar states for configuration, monitoring, and storage
- Recover or recreate clusters from a known state
- Apply or revert configuration changes to multiple OpenShift Container Platform clusters
- Associate templated configuration with different environments
- Promote applications across clusters, from staging to production

For an overview of Red Hat OpenShift GitOps, see [Understanding OpenShift GitOps](#).

4.1.1. Compatibility and support matrix

Some features in this release are currently in [Technology Preview](#). These experimental features are not intended for production use.

In the table, features are marked with the following statuses:

- **TP:** *Technology Preview*
- **GA:** *General Availability*

OpenShift GitOps	Component Versions								OpenShift Versions
Version	kam	Helm	Kustomize	Argo CD	ApplicationSet	Dex	RH SSO	Notifications Controller	
1.7.0	0.0.46 TP	3.10.0 GA	4.5.7 GA	2.5.4 GA	2.4.5 GA	2.35.1 GA	7.5.1 GA	2.4.5 TP	4.8-4.11
1.6.0	0.0.46 TP	3.8.1 GA	4.4.1 GA	2.4.5 GA	2.4.5 GA	2.30.3 GA	7.5.1 GA	2.4.5 TP	4.8-4.10
1.5.0	0.0.42 TP	3.8.0 GA	4.4.1 GA	2.3.3 GA	0.4.1 TP	2.30.3 GA	7.5.1 GA		4.8-4.10

OpenShift GitOps	Component Versions								OpenShift Versions
1.4.0	0.0.41 TP	3.7.1 GA	4.2.0 GA	2.2.2 GA	0.2.0 TP	2.30.0 GA	7.4.0 GA		4.7-4.9
1.3.0	0.0.40 TP	3.6.0 GA	4.2.0 GA	2.1.2 GA	0.2.0 TP	2.28.0 GA	7.4.0 GA		4.7-4.9

- "kam" is an abbreviation for Red Hat OpenShift GitOps Application Manager (kam).
- "RH SSO" is an abbreviation for Red Hat SSO.
- The **Environments** page in the **Developer** perspective of the OpenShift Container Platform web console is also in Technology Preview.

4.1.2. Making open source more inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

4.1.3. Release notes for Red Hat OpenShift GitOps 1.7.0

Red Hat OpenShift GitOps 1.7.0 is now available on OpenShift Container Platform 4.8, 4.9, 4.10, and 4.11.

4.1.3.1. New features

The current release adds the following improvements:

- With this update, you can add environment variables to the Notifications controller. [GITOPS-2313](#)
- With this update, the default nodeSelector **"kubernetes.io/os": "linux"** key-value pair is added to all workloads such that they only schedule on Linux nodes. In addition, any custom node selectors are added to the default and take precedence if they have the same key. [GITOPS-2215](#)
- With this update, you can set custom node selectors in the Operator workloads by editing their **GitopsService** custom resource. [GITOPS-2164](#)
- With this update, you can use the RBAC policy matcher mode to select from the following options: **glob** (default) and **regex**. [GITOPS-1975](#)
- With this update, you can customize resource behavior using the following additional subkeys:

Subkey	Key form	Mapped field in argocd-cm

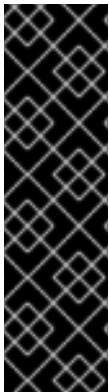
Subkey	Key form	Mapped field in argocd-cm
resourceHealthChecks	resource.customizations.health.<group_kind>	resource.customizations.health
resourceIgnoreDifferences	resource.customizations.ignoreDifferences.<group_kind>	resource.customizations.ignoreDifferences
resourceActions	resource.customizations.actions.<group_kind>	resource.customizations.actions

GITOPS-1561

**NOTE**

In future releases, there is a possibility to deprecate the old method of customizing resource behavior by using only resourceCustomization and not subkeys.

- With this update, to use the **Environments** feature on the **Developer** tab you must upgrade if you are using a Red Hat OpenShift GitOps version prior to 1.7 and OpenShift Container Platform 4.15 or above. [GITOPS-2415](#)
- With this update, applications can be created in any namespace in the same cluster and still managed by the same control-plane's ArgoCD instance. This is done by adding a new label **argocd.argoproj.io/managed-by-cluster-argocd** to the namespace added in **spec.sourceNamespaces** of the Argo CD custom resource. [GITOPS-2341](#)

**IMPORTANT**

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

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

4.1.3.2. Fixed issues

The following issues have been resolved in the current release:

- Before this update, Red Hat OpenShift GitOps releases were affected by an issue of Dex pods failing with **CreateContainerConfigError** error when the **anyuid** SCC was assigned to the Dex service account. This update fixes the issue by assigning a default user id to the Dex container. [GITOPS-2235](#)
- Before this update, Red Hat OpenShift GitOps used the RHSSO (Keycloak) through OIDC in addition to Dex. However, with a recent security fix, the certificate of RHSSO could not be validated when configured with a certificate not signed by one of the well-known certificate

authorities. This update fixes the issue; you can now provide a custom certificate to verify the KeyCloak's TLS certificate while communicating with it. In addition, you can add **rootCA** to the Argo CD custom resource **.spec.keycloak.rootCA** field. The Operator reconciles such changes and updates the **oidc.config in argocd-cm** config map with the PEM encoded root certificate. [GITOPS-2214](#)

Example Argo CD with Keycloak configuration:

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  sso:
    keycloak:
      rootCA: '<PEM encoded root certificate>'
    provider: keycloak
  .....
  .....
```

- Before this update, the application controllers restarted multiple times due to the unresponsiveness of liveness probes. This update fixes the issue by removing the liveness probe in the **statefulset** application controller. [GITOPS-2153](#)

4.1.3.3. Known issues

- Before this update, the Operator did not reconcile the **mountsatoken** and **ServiceAccount** settings for the repository server. While this has been fixed, deletion of the service account does not revert to the default. [GITOPS-1873](#)
- Workaround: Manually set the **spec.repo.serviceaccountfield to thedefault** service account. [GITOPS-2452](#)

4.1.4. Release notes for Red Hat OpenShift GitOps 1.6.2

Red Hat OpenShift GitOps 1.6.2 is now available on OpenShift Container Platform 4.8, 4.9, 4.10 and 4.11.

4.1.4.1. Fixed issues

The following issues have been resolved in the current release:

- Before this update, the subscription health check was marked **degraded** for missing **InstallPlan** when more than 5 Operators were installed in a project. This update fixes the issue. [GITOPS-2018](#)
- Before this update, the Red Hat OpenShift GitOps Operator would spam the cluster with a deprecation notice warning whenever it detected that an Argo CD instance used deprecated fields. This update fixes this issue and shows only one warning event for each instance that detects a field. [GITOPS-2230](#)
- From OpenShift Container Platform 4.12, it is optional to install the console. This fix updates the Red Hat OpenShift GitOps Operator to prevent errors with the Operator if the console is not installed. [GITOPS-2352](#)

4.1.5. Release notes for Red Hat OpenShift GitOps 1.6.1

Red Hat OpenShift GitOps 1.6.1 is now available on OpenShift Container Platform 4.8, 4.9, and 4.10.

4.1.5.1. Fixed issues

The following issues have been resolved in the current release:

- Before this update, in a large set of applications the application controllers were restarted multiple times due to the unresponsiveness of liveness probes. This update fixes the issue by removing the liveness probe in the application controller **StatefulSet** object. [GITOPS-2153](#)
- Before this update, the RHSSO certificate cannot be validated when it is set up with a certificate which is not signed by certificate authorities. This update fixes the issue and now you can provide a custom certificate which will be used in verifying the Keycloak's TLS certificate when communicating with it. You can add the **rootCA** to the Argo CD custom resource **.spec.keycloak.rootCA** field. The Operator reconciles this change and updates the **oidc.config** field in the **argocd-cm ConfigMap** with the PEM-encoded root certificate. [GITOPS-2214](#)



NOTE

Restart the Argo CD server pod after updating the **.spec.keycloak.rootCA** field.

For example:

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
    keycloak:
      rootCA: |
        ---- BEGIN CERTIFICATE ----
        This is a dummy certificate
        Please place this section with appropriate rootCA
        ---- END CERTIFICATE ----
  server:
    route:
      enabled: true
```

- Before this update, a terminating namespace that was managed by Argo CD would block the creation of roles and other configuration of other managed namespaces. This update fixes this issue. [GITOPS-2277](#)
- Before this update, the Dex pods failed to start with **CreateContainerConfigError** when an SCC of **anyuid** was assigned to the Dex **ServiceAccount** resource. This update fixes this issue by assigning a default user id to the Dex container. [GITOPS-2235](#)

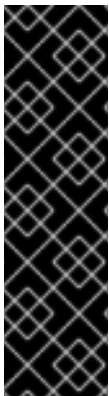
4.1.6. Release notes for Red Hat OpenShift GitOps 1.6.0

Red Hat OpenShift GitOps 1.6.0 is now available on OpenShift Container Platform 4.8, 4.9, and 4.10.

4.1.6.1. New features

The current release adds the following improvements:

- Previously, the Argo CD **ApplicationSet** controller was a technology preview (TP) feature. With this update, it is a general availability (GA) feature. [GITOPS-1958](#)
- With this update, the latest releases of the Red Hat OpenShift GitOps are available in **latest** and version-based channels. To get these upgrades, update the **channel** parameter in the **Subscription** object YAML file: change its value from **stable** to **latest** or a version-based channel such as **gitops-1.6**. [GITOPS-1791](#)
- With this update, the parameters of the **spec.sso** field that controlled the keycloak configurations are moved to **.spec.sso.keycloak**. The parameters of the **.spec.dex** field have been added to **.spec.sso.dex**. Start using **.spec.sso.provider** to enable or disable Dex. The **.spec.dex** parameters are deprecated and planned to be removed in version 1.9, along with the **DISABLE_DEX** and **.spec.sso** fields for keycloak configuration. [GITOPS-1983](#)
- With this update, the Argo CD Notifications controller is available as an optional workload that can be enabled or disabled by using the **.spec.notifications.enabled** parameter in the Argo CD custom resource. The Argo CD Notifications controller is available as a Technical Preview feature. [GITOPS-1917](#)



IMPORTANT

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

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

- With this update, resource exclusions for Tekton pipeline runs and tasks runs are added by default. Argo CD, prunes these resources by default. These resource exclusions are added to the new Argo CD instances that are created from the OpenShift Container Platform. If the instances are created from the CLI, the resources are not added. [GITOPS-1876](#)
- With this update, you can select the tracking method that by Argo CD uses by setting the **resourceTrackingMethod** parameter in the Operand's specification. [GITOPS-1862](#)
- With this update, you can add entries to the **argocd-cm** configMap using the **extraConfig** field of Red Hat OpenShift GitOps Argo CD custom resource. The entries specified are reconciled to the live **config-cm** configMap without validations. [GITOPS-1964](#)
- With this update, on OpenShift Container Platform 4.11, the Red Hat OpenShift GitOps **Environments Details** page in the Red Hat OpenShift GitOps developer perspective shows history of the successful deployments of the application environments, along with links to the revision for each deployment. [GITOPS-1269](#)
- With this update, you can manage resources with Argo CD that are also being used as template resources or "source" by an Operator. [GITOPS-982](#)

- With this update, the Operator will now configure the Argo CD workloads with the correct permissions to satisfy the Pod Security Admission that has been enabled for Kubernetes 1.24. [GITOPS-2026](#)
- With this update, Config Management Plugins 2.0 is supported. You can use the Argo CD custom resource to specify sidebar containers for the repo server. [GITOPS-776](#)
- With this update, all communication between the Argo CD components and the Redis cache are properly secured using modern TLS encryption. [GITOPS-720](#)
- This release of Red Hat OpenShift GitOps adds support for IBM Z and IBM Power on OpenShift Container Platform 4.10. Currently, installations in restricted environments are not supported on IBM Z and IBM Power.

4.1.6.2. Fixed issues

The following issues have been resolved in the current release:

- Before this update, the **system:serviceaccount:argocd:gitops-argocd-application-controller** cannot create resource "prometheusrules" in API group **monitoring.coreos.com** in the namespace **webapps-dev**. This update fixes this issue and Red Hat OpenShift GitOps is now able to manage all resources from the **monitoring.coreos.com** API group. [GITOPS-1638](#)
- Before this update, while reconciling cluster permissions, if a secret belonged to a cluster config instance it was deleted. This update fixes this issue. Now, the **namespaces** field from the secret is deleted instead of the secret. [GITOPS-1777](#)
- Before this update, if you installed the HA variant of Argo CD through the Operator, the Operator created the Redis **StatefulSet** object with **podAffinity** rules instead of **podAntiAffinity** rules. This update fixes this issue and now the Operator creates the Redis **StatefulSet** with **podAntiAffinity** rules. [GITOPS-1645](#)
- Before this update, Argo CD **ApplicationSet** had too many **ssh** Zombie processes. This update fixes this issue: it adds tini, a simple init daemon that spawns processes and reaps zombies, to the **ApplicationSet** controller. This ensures that a **SIGTERM** signal is properly passed to the running process, preventing it from being a zombie process. [GITOPS-2108](#)

4.1.6.3. Known issues

- Red Hat OpenShift GitOps Operator can make use of RHSSO (KeyCloak) through OIDC in addition to Dex. However, with a recent security fix applied, the certificate of RHSSO cannot be validated in some scenarios. [GITOPS-2214](#)
As a workaround, disable TLS validation for the OIDC (Keycloak/RHSSO) endpoint in the ArgoCD specification.

```
spec:
  extraConfig:
    oidc.tls.insecure.skip.verify: "true"
  ...
```

4.1.7. Release notes for Red Hat OpenShift GitOps 1.5.7

Red Hat OpenShift GitOps 1.5.7 is now available on OpenShift Container Platform 4.8, 4.9, 4.10 and 4.11.

4.1.7.1. Fixed issues

The following issues have been resolved in the current release:

- From OpenShift Container Platform 4.12, it is optional to install the console. This fix updates the Red Hat OpenShift GitOps Operator to prevent errors with the Operator if the console is not installed. [GITOPS-2353](#)

4.1.8. Release notes for Red Hat OpenShift GitOps 1.5.6

Red Hat OpenShift GitOps 1.5.6 is now available on OpenShift Container Platform 4.8, 4.9, and 4.10.

4.1.8.1. Fixed issues

The following issues have been resolved in the current release:

- Before this update, in a large set of applications the application controllers were restarted multiple times due to the unresponsiveness of liveness probes. This update fixes the issue by removing the liveness probe in the application controller **StatefulSet** object. [GITOPS-2153](#)
- Before this update, the RHSSO certificate cannot be validated when it is set up with a certificate which is not signed by certificate authorities. This update fixes the issue and now you can provide a custom certificate which will be used in verifying the Keycloak's TLS certificate when communicating with it. You can add the **rootCA** to the Argo CD custom resource **.spec.keycloak.rootCA** field. The Operator reconciles this change and updates the **oidc.config** field in the **argocd-cm ConfigMap** with the PEM-encoded root certificate. [GITOPS-2214](#)



NOTE

Restart the Argo CD server pod after updating the **.spec.keycloak.rootCA** field.

For example:

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
    keycloak:
      rootCA: |
        ---- BEGIN CERTIFICATE ----
        This is a dummy certificate
        Please place this section with appropriate rootCA
        ---- END CERTIFICATE ----
  server:
    route:
      enabled: true
```

- Before this update, a terminating namespace that was managed by Argo CD would block the creation of roles and other configuration of other managed namespaces. This update fixes this issue. [GITOPS-2277](#)

- Before this update, the Dex pods failed to start with **CreateContainerConfigError** when an SCC of **anyuid** was assigned to the Dex **ServiceAccount** resource. This update fixes this issue by assigning a default user id to the Dex container. [GITOPS-2235](#)

4.1.9. Release notes for Red Hat OpenShift GitOps 1.5.5

Red Hat OpenShift GitOps 1.5.5 is now available on OpenShift Container Platform 4.8, 4.9, and 4.10.

4.1.9.1. New features

The current release adds the following improvements:

- With this update, the bundled Argo CD has been updated to version 2.3.7.

4.1.9.2. Fixed issues

The following issues have been resolved in the current release:

- Before this update, the **redis-ha-haproxy** pods of an ArgoCD instance failed when more restrictive SCCs were present in the cluster. This update fixes the issue by updating the security context in workloads. [GITOPS-2034](#)

4.1.9.3. Known issues

- Red Hat OpenShift GitOps Operator can use RHSSO (KeyCloak) with OIDC and Dex. However, with a recent security fix applied, the Operator cannot validate the RHSSO certificate in some scenarios. [GITOPS-2214](#)

As a workaround, disable TLS validation for the OIDC (Keycloak/RHSSO) endpoint in the ArgoCD specification.

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  extraConfig:
    "admin.enabled": "true"
  ...
```

4.1.10. Release notes for Red Hat OpenShift GitOps 1.5.4

Red Hat OpenShift GitOps 1.5.4 is now available on OpenShift Container Platform 4.8, 4.9, and 4.10.

4.1.10.1. Fixed issues

The following issues have been resolved in the current release:

- Before this update, the Red Hat OpenShift GitOps was using an older version of the **REDIS 5** image tag. This update fixes the issue and upgrades the **rhel8/redis-5** image tag. [GITOPS-2037](#)

4.1.11. Release notes for Red Hat OpenShift GitOps 1.5.3

Red Hat OpenShift GitOps 1.5.3 is now available on OpenShift Container Platform 4.8, 4.9, and 4.10.

4.1.11.1. Fixed issues

The following issues have been resolved in the current release:

- Before this update, all unpatched versions of Argo CD v1.0.0 and later were vulnerable to a cross-site scripting bug. As a result, an unauthorized user would be able to inject a javascript link in the UI. This issue is now fixed. [CVE-2022-31035](#)
- Before this update, all versions of Argo CD v0.11.0 and later were vulnerable to multiple attacks when SSO login was initiated from the Argo CD CLI or the UI. This issue is now fixed. [CVE-2022-31034](#)
- Before this update, all unpatched versions of Argo CD v0.7 and later were vulnerable to a memory consumption bug. As a result, an unauthorized user would be able to crash the Argo CD's repo-server. This issue is now fixed. [CVE-2022-31016](#)
- Before this update, all unpatched versions of Argo CD v1.3.0 and later were vulnerable to a symlink-following bug. As a result, an unauthorized user with repository write access would be able to leak sensitive YAML files from Argo CD's repo-server. This issue is now fixed. [CVE-2022-31036](#)

4.1.12. Release notes for Red Hat OpenShift GitOps 1.5.2

Red Hat OpenShift GitOps 1.5.2 is now available on OpenShift Container Platform 4.8, 4.9, and 4.10.

4.1.12.1. Fixed issues

The following issues have been resolved in the current release:

- Before this update, images referenced by the **redhat-operator-index** were missing. This issue is now fixed. [GITOPS-2036](#)

4.1.13. Release notes for Red Hat OpenShift GitOps 1.5.1

Red Hat OpenShift GitOps 1.5.1 is now available on OpenShift Container Platform 4.8, 4.9, and 4.10.

4.1.13.1. Fixed issues

The following issues have been resolved in the current release:

- Before this update, if Argo CD's anonymous access was enabled, an unauthenticated user was able to craft a JWT token and get full access to the Argo CD instance. This issue is fixed now. [CVE-2022-29165](#)
- Before this update, an unauthenticated user was able to display error messages on the login screen while SSO was enabled. This issue is now fixed. [CVE-2022-24905](#)
- Before this update, all unpatched versions of Argo CD v0.7.0 and later were vulnerable to a symlink-following bug. As a result, an unauthorized user with repository write access would be able to leak sensitive files from Argo CD's repo-server. This issue is now fixed. [CVE-2022-24904](#)

4.1.14. Release notes for Red Hat OpenShift GitOps 1.5.0

Red Hat OpenShift GitOps 1.5.0 is now available on OpenShift Container Platform 4.8, 4.9, and 4.10.

4.1.14.1. New features

The current release adds the following improvements:

- This enhancement upgrades Argo CD to version **2.3.3**. [GITOPS-1708](#)
- This enhancement upgrades Dex to version **2.30.3**. [GITOPS-1850](#)
- This enhancement upgrades Helm to version **3.8.0**. [GITOPS-1709](#)
- This enhancement upgrades Kustomize to version **4.4.1**. [GITOPS-1710](#)
- This enhancement upgrades Application Set to version **0.4.1**.
- With this update, a new channel by the name **latest** has been added that provides the latest release of the Red Hat OpenShift GitOps. For GitOps v1.5.0, the Operator is pushed to **gitops-1.5**, **latest** channel, and the existing **stable** channel. From GitOps v1.6 all the latest releases will be pushed only to the **latest** channel and not the **stable** channel. [GITOPS-1791](#)
- With this update, the new CSV adds the **olm.skipRange: '>=1.0.0 <1.5.0'** annotation. As a result, all the previous release versions will be skipped. The Operator upgrades to v1.5.0 directly. [GITOPS-1787](#)
- With this update, the Operator updates the Red Hat Single Sign-On (RH-SSO) to version v7.5.1 including the following enhancements:
 - You can log in to Argo CD using the OpenShift credentials including the **kube:admin** credential.
 - The RH-SSO supports and configures Argo CD instances for Role-based Access Control (RBAC) using OpenShift groups.
 - The RH-SSO honors the **HTTP_Proxy** environment variables. You can use the RH-SSO as an SSO for Argo CD running behind a proxy. [GITOPS-1330](#)
- With this update, a new **.host** URL field is added to the **.status** field of the Argo CD operand. When a route or ingress is enabled with the priority given to route, then the new URL field displays the route. If no URL is provided from the route or ingress, the **.host** field is not displayed.
 When the route or ingress is configured, but the corresponding controller is not set up properly and is not in the **Ready** state or does not propagate its URL, the value of the **.status.host** field in the operand indicates as **Pending** instead of displaying the URL. This affects the overall status of the operand by making it **Pending** instead of **Available**. [GITOPS-654](#)

4.1.14.2. Fixed issues

The following issues have been resolved in the current release:

- Before this update, RBAC rules specific to **AppProjects** would not allow the use of commas for the subject field of the role, thus preventing bindings to the LDAP account. This update fixes the issue and you can now specify complex role bindings in **AppProject** specific RBAC rules. [GITOPS-1771](#)
- Before this update, when a **DeploymentConfig** resource is scaled to **0**, Argo CD displayed it in a **progressing** state with a health status message as **"replication controller is waiting for pods to run"**. This update fixes the edge case and the health check now reports the correct health

status of the **DeploymentConfig** resource. [GITOPS-1738](#)

- Before this update, the TLS certificate in the **argocd-tls-certs-cm** configuration map was deleted by the Red Hat OpenShift GitOps unless the certificate was configured in the **ArgoCD** CR specification **tls.initialCerts** field. This issue is fixed now. [GITOPS-1725](#)
- Before this update, while creating a namespace with the **managed-by** label it created a lot of **RoleBinding** resources on the new namespace. This update fixes the issue and now Red Hat OpenShift GitOps removes the irrelevant **Role** and **RoleBinding** resources created by the previous versions. [GITOPS-1550](#)
- Before this update, the TLS certificate of the route in pass-through mode did not have a CA name. As a result, Firefox 94 and later failed to connect to Argo CD UI with error code **SEC_ERROR_BAD_DER**. This update fixes the issue. You must delete the **<openshift-gitops-ca>** secrets and let it recreate. Then, you must delete the **<openshift-gitops-tls>** secrets. After the Red Hat OpenShift GitOps recreates it, the Argo CD UI is accessible by Firefox again. [GITOPS-1548](#)

4.1.14.3. Known issues

- Argo CD **.status.host** field is not updated when an **Ingress** resource is in use instead of a **Route** resource on OpenShift clusters. [GITOPS-1920](#)

4.1.15. Release notes for Red Hat OpenShift GitOps 1.4.13

Red Hat OpenShift GitOps 1.4.13 is now available on OpenShift Container Platform 4.7, 4.8, 4.9, 4.10 and 4.11.

4.1.15.1. Fixed issues

The following issues have been resolved in the current release:

- From OpenShift Container Platform 4.12, it is optional to install the console. This fix updates the Red Hat OpenShift GitOps Operator to prevent errors with the Operator if the console is not installed. [GITOPS-2354](#)

4.1.16. Release notes for Red Hat OpenShift GitOps 1.4.12

Red Hat OpenShift GitOps 1.4.12 is now available on OpenShift Container Platform 4.8, 4.9, and 4.10.

4.1.16.1. Fixed issues

The following issues have been resolved in the current release:

- Before this update, in a large set of applications the application controllers were restarted multiple times due to the unresponsiveness of liveness probes. This update fixes the issue by removing the liveness probe in the application controller **StatefulSet** object. [GITOPS-2153](#)
- Before this update, the RHSSO certificate cannot be validated when it is set up with a certificate which is not signed by certificate authorities. This update fixes the issue and now you can provide a custom certificate which will be used in verifying the Keycloak's TLS certificate when communicating with it. You can add the **rootCA** to the Argo CD custom resource **.spec.keycloak.rootCA** field. The Operator reconciles this change and updates the **oidc.config** field in the **argocd-cm ConfigMap** with the PEM-encoded root certificate. [GITOPS-2214](#)

**NOTE**

Restart the Argo CD server pod after updating the **.spec.keycloak.rootCA** field.

For example:

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
    keycloak:
      rootCA: |
        ---- BEGIN CERTIFICATE ----
        This is a dummy certificate
        Please place this section with appropriate rootCA
        ---- END CERTIFICATE ----
  server:
    route:
      enabled: true
```

- Before this update, a terminating namespace that was managed by Argo CD would block the creation of roles and other configuration of other managed namespaces. This update fixes this issue. [GITOPS-2277](#)
- Before this update, the Dex pods failed to start with **CreateContainerConfigError** when an SCC of **anyuid** was assigned to the Dex **ServiceAccount** resource. This update fixes this issue by assigning a default user id to the Dex container. [GITOPS-2235](#)

4.1.17. Release notes for Red Hat OpenShift GitOps 1.4.11

Red Hat OpenShift GitOps 1.4.11 is now available on OpenShift Container Platform 4.8, 4.9, and 4.10.

4.1.17.1. New features

The current release adds the following improvements:

- With this update, the bundled Argo CD has been updated to version 2.2.12.

4.1.17.2. Fixed issues

The following issues have been resolved in the current release:

- Before this update, the **redis-ha-haproxy** pods of an ArgoCD instance failed when more restrictive SCCs were present in the cluster. This update fixes the issue by updating the security context in workloads. [GITOPS-2034](#)

4.1.17.3. Known issues

- Red Hat OpenShift GitOps Operator can use RHSSO (KeyCloak) with OIDC and Dex. However, with a recent security fix applied, the Operator cannot validate the RHSSO certificate in some scenarios. [GITOPS-2214](#)

As a workaround, disable TLS validation for the OIDC (Keycloak/RHSSO) endpoint in the ArgoCD specification.

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  extraConfig:
    "admin.enabled": "true"
  ...
```

4.1.18. Release notes for Red Hat OpenShift GitOps 1.4.6

Red Hat OpenShift GitOps 1.4.6 is now available on OpenShift Container Platform 4.7, 4.8, 4.9, and 4.10.

4.1.18.1. Fixed issues

The following issue has been resolved in the current release:

- The base images are updated to the latest version to avoid OpenSSL flaw link: ([CVE-2022-0778](#)).



NOTE

To install the current release of Red Hat OpenShift GitOps 1.4 and receive further updates during its product life cycle, switch to the **GitOps-1.4** channel.

4.1.19. Release notes for Red Hat OpenShift GitOps 1.4.5

Red Hat OpenShift GitOps 1.4.5 is now available on OpenShift Container Platform 4.7, 4.8, 4.9 and 4.10.

4.1.19.1. Fixed issues



WARNING

You should directly upgrade to Red Hat OpenShift GitOps v1.4.5 from Red Hat OpenShift GitOps v1.4.3. Do not use Red Hat OpenShift GitOps v1.4.4 in a production environment. Major issues that affected Red Hat OpenShift GitOps v1.4.4 are fixed in Red Hat OpenShift GitOps 1.4.5.

The following issue has been resolved in the current release:

- Before this update, Argo CD pods were stuck in the **ErrImagePullBackOff** state. The following error message was shown:

■

```

reason: ErrImagePull
  message: >-
    rpc error: code = Unknown desc = reading manifest
    sha256:ff4ad30752cf0d321cd6c2c6fd4490b716607ea2960558347440f2f370a586a8
    in registry.redhat.io/openshift-gitops-1/argocd-rhel8: StatusCode:
    404, <HTML><HEAD><TITLE>Error</TITLE></HEAD><BODY>

```

This issue is now fixed. [GITOPS-1848](#)

4.1.20. Release notes for Red Hat OpenShift GitOps 1.4.3

Red Hat OpenShift GitOps 1.4.3 is now available on OpenShift Container Platform 4.7, 4.8, and 4.9.

4.1.20.1. Fixed issues

The following issue has been resolved in the current release:

- Before this update, the TLS certificate in the **argocd-tls-certs-cm** configuration map was deleted by the Red Hat OpenShift GitOps unless the certificate was configured in the ArgoCD CR specification **tls.initialCerts** field. This update fixes this issue. [GITOPS-1725](#)

4.1.21. Release notes for Red Hat OpenShift GitOps 1.4.2

Red Hat OpenShift GitOps 1.4.2 is now available on OpenShift Container Platform 4.7, 4.8, and 4.9.

4.1.21.1. Fixed issues

The following issue has been resolved in the current release:

- All versions of Argo CD are vulnerable to a path traversal bug that allows to pass arbitrary values to be consumed by Helm charts. This update fixes the **CVE-2022-24348 gitops** error, path traversal and dereference of symlinks when passing Helm value files. [GITOPS-1756](#)
- Before this update, the **Route** resources got stuck in **Progressing** Health status if more than one **Ingress** were attached to the route. This update fixes the health check and reports the correct health status of the **Route** resources. [GITOPS-1751](#)

4.1.22. Release notes for Red Hat OpenShift GitOps 1.4.1

Red Hat OpenShift GitOps 1.4.1 is now available on OpenShift Container Platform 4.7, 4.8, and 4.9.

4.1.22.1. Fixed issues

The following issue has been resolved in the current release:

- Red Hat OpenShift GitOps Operator v1.4.0 introduced a regression which removes the description fields from **spec** for the following CRDs:
 - **argoproj.io_applications.yaml**
 - **argoproj.io_appprojects.yaml**
 - **argoproj.io_argocds.yaml**

Before this update, when you created an **AppProject** resource using the **oc create** command, the resource failed to synchronize due to the missing description fields. This update restores the missing description fields in the preceding CRDs. [GITOPS-1721](#)

4.1.23. Release notes for Red Hat OpenShift GitOps 1.4.0

Red Hat OpenShift GitOps 1.4.0 is now available on OpenShift Container Platform 4.7, 4.8, and 4.9.

4.1.23.1. New features

The current release adds the following improvements.

- This enhancement upgrades Red Hat OpenShift GitOps Application Manager (kam) to version **0.0.41**. [GITOPS-1669](#)
- This enhancement upgrades Argo CD to version **2.2.2**. [GITOPS-1532](#)
- This enhancement upgrades Helm to version **3.7.1**. [GITOPS-1530](#)
- This enhancement adds the health status of the **DeploymentConfig**, **Route**, and **OLM Operator** items to the Argo CD Dashboard and OpenShift Container Platform web console. This information helps you monitor the overall health status of your application. [GITOPS-655](#), [GITOPS-915](#), [GITOPS-916](#), [GITOPS-1110](#)
- With this update, you can specify the number of desired replicas for the **argocd-server** and **argocd-repo-server** components by setting the **.spec.server.replicas** and **.spec.repo.replicas** attributes in the Argo CD custom resource, respectively. If you configure the horizontal pod autoscaler (HPA) for the **argocd-server** components, it takes precedence over the Argo CD custom resource attributes. [GITOPS-1245](#)
- As an administrative user, when you give Argo CD access to a namespace by using the **argocd.argoproj.io/managed-by** label, it assumes namespace-admin privileges. These privileges are an issue for administrators who provide namespaces to non-administrators, such as development teams, because the privileges enable non-administrators to modify objects such as network policies.
With this update, administrators can configure a common cluster role for all the managed namespaces. In role bindings for the Argo CD application controller, the Operator refers to the **CONTROLLER_CLUSTER_ROLE** environment variable. In role bindings for the Argo CD server, the Operator refers to the **SERVER_CLUSTER_ROLE** environment variable. If these environment variables contain custom roles, the Operator doesn't create the default admin role. Instead, it uses the existing custom role for all managed namespaces. [GITOPS-1290](#)
- With this update, the Environment page in the OpenShift Container Platform Developer Console displays a broken heart icon to indicate degraded resources, excluding ones whose status is Progressing, Missing, and Unknown. The console displays a yellow yield sign icon to indicate out-of-sync resources. [GITOPS-1307](#)

4.1.23.2. Fixed issues

The following issues have been resolved in the current release:

- Before this update, when the Route to the Red Hat OpenShift GitOps Application Manager (kam) was accessed without specifying a path in the URL, a default page without any helpful information was displayed to the user. This update fixes the issue so that the default page displays download links for kam. [GITOPS-923](#)

- Before this update, setting a resource quota in the namespace of the Argo CD custom resource might cause the setup of the Red Hat SSO (RH SSO) instance to fail. This update fixes this issue by setting a minimum resource request for the RH SSO deployment pods. [GITOPS-1297](#)
- Before this update, if you changed the log level for the **argocd-repo-server** workload, the Operator didn't reconcile this setting. The workaround was to delete the deployment resource so that the Operator recreated it with the new log level. With this update, the log level is correctly reconciled for existing **argocd-repo-server** workloads. [GITOPS-1387](#)
- Before this update, if the Operator managed an Argo CD instance that lacked the **.data** field in the **argocd-secret** Secret, the Operator on that instance crashed. This update fixes the issue so that the Operator doesn't crash when the **.data** field is missing. Instead, the secret regenerates and the **gitops-operator-controller-manager** resource is redeployed. [GITOPS-1402](#)
- Before this update, the **gitopsservice** service was annotated as an internal object. This update removes the annotation so you can update or delete the default Argo CD instance and run GitOps workloads on infrastructure nodes by using the UI. [GITOPS-1429](#)

4.1.23.3. Known issues

These are the known issues in the current release:

- If you migrate from the Dex authentication provider to the Keycloak provider, you might experience login issues with Keycloak.
To prevent this issue, when migrating, uninstall Dex by removing the **.spec.dex** section from the Argo CD custom resource. Allow a few minutes for Dex to uninstall completely. Then, install Keycloak by adding **.spec.sso.provider: keycloak** to the Argo CD custom resource.

As a workaround, uninstall Keycloak by removing **.spec.sso.provider: keycloak**. Then, re-install it. [GITOPS-1450](#), [GITOPS-1331](#)

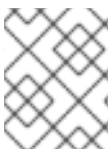
4.1.24. Release notes for Red Hat OpenShift GitOps 1.3.7

Red Hat OpenShift GitOps 1.3.7 is now available on OpenShift Container Platform 4.7, 4.8, 4.9, and 4.10.

4.1.24.1. Fixed issues

The following issue has been resolved in the current release:

- Before this update, a flaw was found in OpenSSL. This update fixes the issue by updating the base images to the latest version to avoid the OpenSSL flaw. ([CVE-2022-0778](#)).



NOTE

To install the current release of Red Hat OpenShift GitOps 1.3 and receive further updates during its product life cycle, switch to the **GitOps-1.3** channel.

4.1.25. Release notes for Red Hat OpenShift GitOps 1.3.6

Red Hat OpenShift GitOps 1.3.6 is now available on OpenShift Container Platform 4.7, 4.8, 4.9, and 4.10.

4.1.25.1. Fixed issues

The following issues have been resolved in the current release:

- In Red Hat OpenShift GitOps, improper access control allows admin privilege escalation ([CVE-2022-1025](#)). This update fixes the issue.
- A path traversal flaw allows leaking of out-of-bound files ([CVE-2022-24731](#)). This update fixes the issue.
- A path traversal flaw and improper access control allows leaking of out-of-bound files ([CVE-2022-24730](#)). This update fixes the issue.

4.1.26. Release notes for Red Hat OpenShift GitOps 1.3.3

Red Hat OpenShift GitOps 1.3.3 is now available on OpenShift Container Platform 4.7, 4.8, and 4.9.

4.1.26.1. Fixed issues

The following issue has been resolved in the current release:

- All versions of Argo CD are vulnerable to a path traversal bug that allows to pass arbitrary values to be consumed by Helm charts. This update fixes the **CVE-2022-24348 gitops** error, path traversal and dereference of symlinks when passing Helm value files. [GITOPS-1756](#)

4.1.27. Release notes for Red Hat OpenShift GitOps 1.3.2

Red Hat OpenShift GitOps 1.3.2 is now available on OpenShift Container Platform 4.6, 4.7, 4.8, and 4.9.

4.1.27.1. New features

In addition to the fixes and stability improvements, the following sections highlight what is new in Red Hat OpenShift GitOps 1.3.2:

- Upgraded Argo CD to version **2.1.8**
- Upgraded Dex to version **2.30.0**

4.1.27.2. Fixed issues

The following issues have been resolved in the current release:

- Previously, in the OperatorHub UI under the **Infrastructure Features** section, when you filtered by **Disconnected** the Red Hat OpenShift GitOps Operator did not show in the search results, as the Operator did not have the related annotation set in its CSV file. With this update, the **Disconnected Cluster** annotation has been added to the Red Hat OpenShift GitOps Operator as an infrastructure feature. [GITOPS-1539](#)
- When using an **Namespace-scoped** Argo CD instance, for example, an Argo CD instance that is not scoped to **All Namespaces** in a cluster, Red Hat OpenShift GitOps dynamically maintains a list of managed namespaces. These namespaces include the **argocd.argoproj.io/managed-by** label. This list of namespaces is stored in a cache in **Argo CD → Settings → Clusters → "in-cluster" → NAMESPACES**. Before this update, if you deleted one of these namespaces, the Operator ignored that, and the namespace remained in the list. This behavior broke the **CONNECTION STATE** in that cluster configuration, and all sync attempts resulted in errors. For example:

Argo service account does not have <random_verb> on <random_resource_type> in namespace <the_namespace_you_deleted>.

This bug is fixed. [GITOPS-1521](#)

- With this update, the Red Hat OpenShift GitOps Operator has been annotated with the **Deep Insights** capability level. [GITOPS-1519](#)
- Previously, the Argo CD Operator managed the **resource.exclusion** field by itself but ignored the **resource.inclusion** field. This prevented the **resource.inclusion** field configured in the **Argo CD** CR to generate in the **argocd-cm** configuration map. This bug is fixed. [GITOPS-1518](#)

4.1.28. Release notes for Red Hat OpenShift GitOps 1.3.1

Red Hat OpenShift GitOps 1.3.1 is now available on OpenShift Container Platform 4.6, 4.7, 4.8, and 4.9.

4.1.28.1. Fixed issues

- If you upgrade to v1.3.0, the Operator does not return an ordered slice of environment variables. As a result, the reconciler fails causing the frequent recreation of Argo CD pods in OpenShift Container Platform clusters running behind a proxy. This update fixes the issue so that Argo CD pods are not recreated. [GITOPS-1489](#)

4.1.29. Release notes for Red Hat OpenShift GitOps 1.3

Red Hat OpenShift GitOps 1.3 is now available on OpenShift Container Platform 4.7, 4.8, and 4.9.

4.1.29.1. New features

In addition to the fixes and stability improvements, the following sections highlight what is new in Red Hat OpenShift GitOps 1.3.0:

- For a fresh install of v1.3.0, Dex is automatically configured. You can log into the default Argo CD instance in the **openshift-gitops** namespace using the OpenShift or **kubeadmin** credentials. As an admin you can disable the Dex installation after the Operator is installed which will remove the Dex deployment from the **openshift-gitops** namespace.
- The default Argo CD instance installed by the Operator as well as accompanying controllers can now run on the infrastructure nodes of the cluster by setting a simple configuration toggle.
- Internal communications in Argo CD can now be secured using the TLS and the OpenShift cluster certificates. The Argo CD routes can now leverage the OpenShift cluster certificates in addition to using external certificate managers such as the cert-manager.
- Use the improved **environment details** view in the **Developer** perspective of the console 4.9 to gain insights into the GitOps environments.
- You can now access custom health checks in Argo CD for **DeploymentConfig** resources, **Route** resources, and Operators installed using OLM.
- The GitOps Operator now conforms to the naming conventions recommended by the latest Operator-SDK:
 - The prefix **gitops-operator-** is added to all resources
 - Service account is renamed to **gitops-operator-controller-manager**

4.1.29.2. Fixed issues

The following issues were resolved in the current release:

- Previously, if you set up a new namespace to be managed by a new instance of Argo CD, it would immediately be **Out Of Sync** due to the new roles and bindings that the Operator creates to manage that new namespace. This behavior is fixed. [GITOPS-1384](#)

4.1.29.3. Known issues

- While migrating from the Dex authentication provider to the Keycloak provider, you may experience login issues with Keycloak. [GITOPS-1450](#)
To prevent the above issue, when migrating, uninstall Dex by removing the **.spec.dex** section found in the Argo CD custom resource. Allow a few minutes for Dex to uninstall completely, and then proceed to install Keycloak by adding **.spec.sso.provider: keycloak** to the Argo CD custom resource.

As a workaround, uninstall Keycloak by removing **.spec.sso.provider: keycloak** and then re-install.

4.1.30. Release notes for Red Hat OpenShift GitOps 1.2.1

Red Hat OpenShift GitOps 1.2.1 is now available on OpenShift Container Platform 4.7 and 4.8.

4.1.30.1. Support matrix

Some features in this release are currently in Technology Preview. These experimental features are not intended for production use.

Technology Preview Features Support Scope

In the table below, features are marked with the following statuses:

- TP:** *Technology Preview*
- GA:** *General Availability*

Note the following scope of support on the Red Hat Customer Portal for these features:

Table 4.1. Support matrix

Feature	Red Hat OpenShift GitOps 1.2.1
Argo CD	GA
Argo CD ApplicationSet	TP
Red Hat OpenShift GitOps Application Manager (kam)	TP

4.1.30.2. Fixed issues

The following issues were resolved in the current release:

- Previously, huge memory spikes were observed on the application controller on startup. The flag

--kubectl-parallelism-limit for the application controller is now set to 10 by default, however this value can be overridden by specifying a number for **.spec.controller.kubeParallelismLimit** in the Argo CD CR specification. [GITOPS-1255](#)

- The latest Triggers APIs caused Kubernetes build failure due to duplicate entries in the `kustomization.yaml` when using the **kam bootstrap** command. The Pipelines and Tekton triggers components have now been updated to v0.24.2 and v0.14.2, respectively, to address this issue. [GITOPS-1273](#)
- Persisting RBAC roles and bindings are now automatically removed from the target namespace when the Argo CD instance from the source namespace is deleted. [GITOPS-1228](#)
- Previously, when deploying an Argo CD instance into a namespace, the Argo CD instance would change the "managed-by" label to be its own namespace. This fix would make namespaces unlabelled while also making sure the required RBAC roles and bindings are created and deleted for the namespace. [GITOPS-1247](#)
- Previously, the default resource request limits on Argo CD workloads, specifically for the repo-server and application controller, were found to be very restrictive. The existing resource quota has now been removed and the default memory limit has been increased to 1024M in the repo server. Please note that this change will only affect new installations; existing Argo CD instance workloads will not be affected. [GITOPS-1274](#)

4.1.31. Release notes for Red Hat OpenShift GitOps 1.2

Red Hat OpenShift GitOps 1.2 is now available on OpenShift Container Platform 4.7, 4.8, and 4.9.

4.1.31.1. Support matrix

Some features in this release are currently in Technology Preview. These experimental features are not intended for production use.

Technology Preview Features Support Scope

In the table below, features are marked with the following statuses:

- **TP:** *Technology Preview*
- **GA:** *General Availability*

Note the following scope of support on the Red Hat Customer Portal for these features:

Table 4.2. Support matrix

Feature	Red Hat OpenShift GitOps 1.2
Argo CD	GA
Argo CD ApplicationSet	TP
Red Hat OpenShift GitOps Application Manager (kam)	TP

4.1.31.2. New features

In addition to the fixes and stability improvements, the following sections highlight what is new in Red Hat OpenShift GitOps 1.2:

- If you do not have read or write access to the `openshift-gitops` namespace, you can now use the **DISABLE_DEFAULT_ARGOCD_INSTANCE** environment variable in the GitOps Operator and set the value to **TRUE** to prevent the default Argo CD instance from starting in the **openshift-gitops** namespace.
- Resource requests and limits are now configured in Argo CD workloads. Resource quota is enabled in the **openshift-gitops** namespace. As a result, out-of-band workloads deployed manually in the `openshift-gitops` namespace must be configured with resource requests and limits and the resource quota may need to be increased.
- Argo CD authentication is now integrated with Red Hat SSO and it is automatically configured with OpenShift 4 Identity Provider on the cluster. This feature is disabled by default. To enable Red Hat SSO, add SSO configuration in **ArgoCD** CR as shown below. Currently, **keycloak** is the only supported provider.

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
  server:
    route:
      enabled: true
```

- You can now define hostnames using route labels to support router sharding. Support for setting labels on the **server** (argocd server), **grafana**, and **prometheus** routes is now available. To set labels on a route, add **labels** under the route configuration for a server in the **ArgoCD** CR.

Example ArgoCD CR YAML to set labels on argocd server

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  server:
    route:
      enabled: true
      labels:
        key1: value1
        key2: value2
```

- The GitOps Operator now automatically grants permissions to Argo CD instances to manage resources in target namespaces by applying labels. Users can label the target namespace with the label **argocd.argoproj.io/managed-by: <source-namespace>**, where the **source-**

namespace is the namespace where the argocd instance is deployed.

4.1.31.3. Fixed issues

The following issues were resolved in the current release:

- Previously, if a user created additional instances of Argo CD managed by the default cluster instance in the `openshift-gitops` namespace, the application responsible for the new Argo CD instance would get stuck in an **OutOfSync** status. This issue has now been resolved by adding an owner reference to the cluster secret. [GITOPS-1025](#)

4.1.31.4. Known issues

These are the known issues in Red Hat OpenShift GitOps 1.2:

- When an Argo CD instance is deleted from the source namespace, the **argocd.argoproj.io/managed-by** labels in the target namespaces are not removed. [GITOPS-1228](#)
- Resource quota has been enabled in the `openshift-gitops` namespace in Red Hat OpenShift GitOps 1.2. This can affect out-of-band workloads deployed manually and workloads deployed by the default Argo CD instance in the **openshift-gitops** namespace. When you upgrade from Red Hat OpenShift GitOps **v1.1.2** to **v1.2** such workloads must be configured with resource requests and limits. If there are any additional workloads, the resource quota in the `openshift-gitops` namespace must be increased.

Current Resource Quota for **openshift-gitops** namespace.

Resource	Requests	Limits
CPU	6688m	13750m
Memory	4544Mi	9070Mi

You can use the below command to update the CPU limits.

```
$ oc patch resourcequota openshift-gitops-compute-resources -n openshift-gitops --
type=json -p='[{"op": "replace", "path": "/spec/hard/limits.cpu", "value": "9000m"}]'
```

You can use the below command to update the CPU requests.

```
$ oc patch resourcequota openshift-gitops-compute-resources -n openshift-gitops --
type=json -p='[{"op": "replace", "path": "/spec/hard/cpu", "value": "7000m"}]'
```

You can replace the path in the above commands from **cpu** to **memory** to update the memory.

4.1.32. Release notes for Red Hat OpenShift GitOps 1.1

Red Hat OpenShift GitOps 1.1 is now available on OpenShift Container Platform 4.7.

4.1.32.1. Support matrix

Some features in this release are currently in Technology Preview. These experimental features are not intended for production use.

Technology Preview Features Support Scope

In the table below, features are marked with the following statuses:

- **TP:** *Technology Preview*
- **GA:** *General Availability*

Note the following scope of support on the Red Hat Customer Portal for these features:

Table 4.3. Support matrix

Feature	Red Hat OpenShift GitOps 1.1
Argo CD	GA
Argo CD ApplicationSet	TP
Red Hat OpenShift GitOps Application Manager (kam)	TP

4.1.32.2. New features

In addition to the fixes and stability improvements, the following sections highlight what is new in Red Hat OpenShift GitOps 1.1:

- The **ApplicationSet** feature is now added (Technology Preview). The **ApplicationSet** feature enables both automation and greater flexibility when managing Argo CD applications across a large number of clusters and within monorepos. It also makes self-service usage possible on multitenant Kubernetes clusters.
- Argo CD is now integrated with cluster logging stack and with the OpenShift Container Platform Monitoring and Alerting features.
- Argo CD auth is now integrated with OpenShift Container Platform.
- Argo CD applications controller now supports horizontal scaling.
- Argo CD Redis servers now support high availability (HA).

4.1.32.3. Fixed issues

The following issues were resolved in the current release:

- Previously, Red Hat OpenShift GitOps did not work as expected in a proxy server setup with active global proxy settings. This issue is fixed and now Argo CD is configured by the Red Hat OpenShift GitOps Operator using fully qualified domain names (FQDN) for the pods to enable communication between components. [GITOPS-703](#)
- The Red Hat OpenShift GitOps backend relies on the **?ref=** query parameter in the Red Hat OpenShift GitOps URL to make API calls. Previously, this parameter was not read from the URL, causing the backend to always consider the default reference. This issue is fixed and the Red

Hat OpenShift GitOps backend now extracts the reference query parameter from the Red Hat OpenShift GitOps URL and only uses the default reference when there is no input reference provided. [GITOPS-817](#)

- Previously, the Red Hat OpenShift GitOps backend failed to find the valid GitLab repository. This was because the Red Hat OpenShift GitOps backend checked for **main** as the branch reference, instead of **master** in the GitLab repository. This issue is fixed now. [GITOPS-768](#)
- The **Environments** page in the **Developer** perspective of the OpenShift Container Platform web console now shows the list of applications and the number of environments. This page also displays an Argo CD link that directs you to the Argo CD **Applications** page that lists all the applications. The Argo CD **Applications** page has **LABELS** (for example, **app.kubernetes.io/name=appName**) that help you filter only the applications of your choice. [GITOPS-544](#)

4.1.32.4. Known issues

These are the known issues in Red Hat OpenShift GitOps 1.1:

- Red Hat OpenShift GitOps does not support Helm v2 and ksonnet.
- The Red Hat SSO (RH SSO) Operator is not supported in disconnected clusters. As a result, the Red Hat OpenShift GitOps Operator and RH SSO integration is not supported in disconnected clusters.
- When you delete an Argo CD application from the OpenShift Container Platform web console, the Argo CD application gets deleted in the user interface, but the deployments are still present in the cluster. As a workaround, delete the Argo CD application from the Argo CD console. [GITOPS-830](#)

4.1.32.5. Breaking Change

4.1.32.5.1. Upgrading from Red Hat OpenShift GitOps v1.0.1

When you upgrade from Red Hat OpenShift GitOps **v1.0.1** to **v1.1**, the Red Hat OpenShift GitOps Operator renames the default Argo CD instance created in the **openshift-gitops** namespace from **argocd-cluster** to **openshift-gitops**.

This is a breaking change and needs the following steps to be performed manually, before the upgrade:

1. Go to the OpenShift Container Platform web console and copy the content of the **argocd-cm.yml** config map file in the **openshift-gitops** namespace to a local file. The content may look like the following example:

Example argocd config map YAML

```
kind: ConfigMap
apiVersion: v1
metadata:
  selfLink: /api/v1/namespaces/openshift-gitops/configmaps/argocd-cm
  resourceVersion: '112532'
  name: argocd-cm
  uid: f5226fbc-883d-47db-8b53-b5e363f007af
  creationTimestamp: '2021-04-16T19:24:08Z'
  managedFields:
```

```

...
namespace: openshift-gitops
labels:
  app.kubernetes.io/managed-by: argocd-cluster
  app.kubernetes.io/name: argocd-cm
  app.kubernetes.io/part-of: argocd
data: "" ❶
admin.enabled: 'true'
statusbadge.enabled: 'false'
resource.exclusions: |
  - apiGroups:
    - tekton.dev
    clusters:
      - '*'
    kinds:
      - TaskRun
      - PipelineRun
ga.trackingid: ""
repositories: |
  - type: git
    url: https://github.com/user-name/argocd-example-apps
ga.anonymizeusers: 'false'
help.chatUrl: ""
url: >-
  https://argocd-cluster-server-openshift-gitops.apps.dev-svc-4.7-
  041614.devcluster.openshift.com "" ❷
help.chatText: ""
kustomize.buildOptions: ""
resource.inclusions: ""
repository.credentials: ""
users.anonymous.enabled: 'false'
configManagementPlugins: ""
application.instanceLabelKey: ""

```

- ❶ Restore only the **data** section of the content in the **argocd-cm.yml** config map file manually.
- ❷ Replace the URL value in the config map entry with the new instance name **openshift-gitops**.

2. Delete the default **argocd-cluster** instance.
3. Edit the new **argocd-cm.yml** config map file to restore the entire **data** section manually.
4. Replace the URL value in the config map entry with the new instance name **openshift-gitops**. For example, in the preceding example, replace the URL value with the following URL value:

```

url: >-
  https://openshift-gitops-server-openshift-gitops.apps.dev-svc-4.7-
  041614.devcluster.openshift.com

```

5. Login to the Argo CD cluster and verify that the previous configurations are present.

4.2. UNDERSTANDING OPENSIFT GITOPS

4.2.1. About GitOps

GitOps is a declarative way to implement continuous deployment for cloud native applications. You can use GitOps to create repeatable processes for managing OpenShift Container Platform clusters and applications across multi-cluster Kubernetes environments. GitOps handles and automates complex deployments at a fast pace, saving time during deployment and release cycles.

The GitOps workflow pushes an application through development, testing, staging, and production. GitOps either deploys a new application or updates an existing one, so you only need to update the repository; GitOps automates everything else.

GitOps is a set of practices that use Git pull requests to manage infrastructure and application configurations. In GitOps, the Git repository is the only source of truth for system and application configuration. This Git repository contains a declarative description of the infrastructure you need in your specified environment and contains an automated process to make your environment match the described state. Also, it contains the entire state of the system so that the trail of changes to the system state are visible and auditable. By using GitOps, you resolve the issues of infrastructure and application configuration sprawl.

GitOps defines infrastructure and application definitions as code. Then, it uses this code to manage multiple workspaces and clusters to simplify the creation of infrastructure and application configurations. By following the principles of the code, you can store the configuration of clusters and applications in Git repositories, and then follow the Git workflow to apply these repositories to your chosen clusters. You can apply the core principles of developing and maintaining software in a Git repository to the creation and management of your cluster and application configuration files.

4.2.2. About Red Hat OpenShift GitOps

Red Hat OpenShift GitOps ensures consistency in applications when you deploy them to different clusters in different environments, such as: development, staging, and production. Red Hat OpenShift GitOps organizes the deployment process around the configuration repositories and makes them the central element. It always has at least two repositories:

1. Application repository with the source code
2. Environment configuration repository that defines the desired state of the application

These repositories contain a declarative description of the infrastructure you need in your specified environment. They also contain an automated process to make your environment match the described state.

Red Hat OpenShift GitOps uses Argo CD to maintain cluster resources. Argo CD is an open-source declarative tool for the continuous integration and continuous deployment (CI/CD) of applications. Red Hat OpenShift GitOps implements Argo CD as a controller so that it continuously monitors application definitions and configurations defined in a Git repository. Then, Argo CD compares the specified state of these configurations with their live state on the cluster.

Argo CD reports any configurations that deviate from their specified state. These reports allow administrators to automatically or manually resync configurations to the defined state. Therefore, Argo CD enables you to deliver global custom resources, like the resources that are used to configure OpenShift Container Platform clusters.

4.2.2.1. Key features

Red Hat OpenShift GitOps helps you automate the following tasks:

- Ensure that the clusters have similar states for configuration, monitoring, and storage
- Apply or revert configuration changes to multiple OpenShift Container Platform clusters
- Associate templated configuration with different environments
- Promote applications across clusters, from staging to production

4.3. INSTALLING OPENSIFT GITOPS

Red Hat OpenShift GitOps uses Argo CD to manage specific cluster-scoped resources, including cluster Operators, optional Operator Lifecycle Manager (OLM) Operators, and user management.

Prerequisites

- You have access to the OpenShift Container Platform web console.
- You are logged in as a user with the **cluster-admin** role.
- You are logged in to the OpenShift Container Platform cluster as an administrator.
- Your cluster has the [Marketplace capability](#) enabled or the Red Hat Operator catalog source configured manually.



WARNING

If you have already installed the Community version of the Argo CD Operator, remove the Argo CD Community Operator before you install the Red Hat OpenShift GitOps Operator.

This guide explains how to install the Red Hat OpenShift GitOps Operator to an OpenShift Container Platform cluster and log in to the Argo CD instance.

4.3.1. Installing OpenShift GitOps Operator in web console

Procedure

1. Open the **Administrator** perspective of the web console and navigate to **Operators** → **OperatorHub** in the menu on the left.
2. Search for **OpenShift GitOps**, click the **Red Hat OpenShift GitOps** tile, and then click **Install**. Red Hat OpenShift GitOps will be installed in all namespaces of the cluster.

After the Red Hat OpenShift GitOps Operator is installed, it automatically sets up a ready-to-use Argo CD instance that is available in the **openshift-gitops** namespace, and an Argo CD icon is displayed in the console toolbar. You can create subsequent Argo CD instances for your applications under your projects.

4.3.2. Installing OpenShift GitOps Operator using CLI

You can install Red Hat OpenShift GitOps Operator from the OperatorHub using the CLI.

Procedure

1. Create a Subscription object YAML file to subscribe a namespace to the Red Hat OpenShift GitOps, for example, **sub.yaml**:

Example Subscription

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-gitops-operator
  namespace: openshift-operators
spec:
  channel: stable ❶
  installPlanApproval: Automatic
  name: openshift-gitops-operator ❷
  source: redhat-operators ❸
  sourceNamespace: openshift-marketplace ❹
```

- ❶ Specify the channel name from where you want to subscribe the Operator.
- ❷ Specify the name of the Operator to subscribe to.
- ❸ Specify the name of the CatalogSource that provides the Operator.
- ❹ The namespace of the CatalogSource. Use **openshift-marketplace** for the default OperatorHub CatalogSources.

2. Apply the **Subscription** to the cluster:

```
$ oc apply -f openshift-gitops-sub.yaml
```

3. After the installation is complete, ensure that all the pods in the **openshift-gitops** namespace are running:

```
$ oc get pods -n openshift-gitops
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
cluster-b5798d6f9-zr576	1/1	Running	0	65m
kam-69866d7c48-8nsjv	1/1	Running	0	65m
openshift-gitops-application-controller-0	1/1	Running	0	53m
openshift-gitops-applicationset-controller-6447b8dfdd-5ckgh	1/1	Running	0	65m
openshift-gitops-redis-74bd8d7d96-49bjf	1/1	Running	0	65m
openshift-gitops-repo-server-c999f75d5-l4rsg	1/1	Running	0	65m
openshift-gitops-server-5785f7668b-wj57t	1/1	Running	0	53m

4.3.3. Logging in to the Argo CD instance by using the Argo CD admin account

Red Hat OpenShift GitOps Operator automatically creates a ready-to-use Argo CD instance that is available in the **openshift-gitops** namespace.

Prerequisites

- You have installed the Red Hat OpenShift GitOps Operator in your cluster.

Procedure

1. In the **Administrator** perspective of the web console, navigate to **Operators → Installed Operators** to verify that the Red Hat OpenShift GitOps Operator is installed.



2. Navigate to the **OpenShift GitOps → Cluster Argo CD**. The login page of the Argo CD UI is displayed in a new window.

3. Obtain the password for the Argo CD instance:

- a. In the left panel of the console, use the perspective switcher to switch to the **Developer** perspective.
- b. Use the **Project** drop-down list and select the **openshift-gitops** project.
- c. Use the left navigation panel to navigate to the **Secrets** page.
- d. Select the **openshift-gitops-cluster** instance to display the password.
- e. Copy the password.



NOTE

To login with your OpenShift Container Platform credentials, select the **LOG IN VIA OPENSHIFT** option in the Argo CD user interface.

4. Use this password and **admin** as the username to log in to the Argo CD UI in the new window.



NOTE

You cannot create two Argo CD CRs in the same namespace.

4.4. UNINSTALLING OPENSHIFT GITOPS

Uninstalling the Red Hat OpenShift GitOps Operator is a two-step process:

1. Delete the Argo CD instances that were added under the default namespace of the Red Hat OpenShift GitOps Operator.
2. Uninstall the Red Hat OpenShift GitOps Operator.

Uninstalling only the Operator will not remove the Argo CD instances created.

4.4.1. Deleting the Argo CD instances

Delete the Argo CD instances added to the namespace of the GitOps Operator.

Procedure

1. In the **Terminal** type the following command:

```
$ oc delete gitopsservice cluster -n openshift-gitops
```



NOTE

You cannot delete an Argo CD cluster from the web console UI.

After the command runs successfully all the Argo CD instances will be deleted from the **openshift-gitops** namespace.

Delete any other Argo CD instances from other namespaces using the same command:

```
$ oc delete gitopsservice cluster -n <namespace>
```

4.4.2. Uninstalling the GitOps Operator

Procedure

1. From the **Operators → OperatorHub** page, use the **Filter by keyword** box to search for **Red Hat OpenShift GitOps Operator** tile.
2. Click the **Red Hat OpenShift GitOps Operator** tile. The Operator tile indicates it is installed.
3. In the **Red Hat OpenShift GitOps Operator** descriptor page, click **Uninstall**.

Additional resources

- You can learn more about uninstalling Operators on OpenShift Container Platform in the [Deleting Operators from a cluster](#) section.

4.5. SETTING UP A NEW ARGO CD INSTANCE

By default, the Red Hat OpenShift GitOps installs an instance of Argo CD in the **openshift-gitops** namespace with additional permissions for managing certain cluster-scoped resources. To manage cluster configurations or deploy applications, you can install and deploy a new Argo CD instance. By default, any new instance has permissions to manage resources only in the namespace where it is deployed.

4.5.1. Installing Argo CD

To manage cluster configurations or deploy applications, you can install and deploy a new Argo CD instance.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Click **Operators → Installed Operators**.

3. Create or select the project where you want to install the Argo CD instance from the **Project** drop-down menu.
4. Select **OpenShift GitOps Operator** from the installed operators and select the **Argo CD** tab.
5. Click **Create** to configure the parameters:
 - a. Enter the **Name** of the instance. By default, the **Name** is set to **argocd**.
 - b. Create an external OS Route to access Argo CD server. Click **Server → Route** and check **Enabled**.
6. To open the Argo CD web UI, click the route by navigating to **Networking → Routes → <instance name> → server** in the project where the Argo CD instance is installed.

4.5.2. Enabling replicas for Argo CD server and repo server

Argo CD-server and Argo CD-repo-server workloads are stateless. To better distribute your workloads among pods, you can increase the number of Argo CD-server and Argo CD-repo-server replicas. However, if a horizontal autoscaler is enabled on the Argo CD-server, it overrides the number of replicas you set.

Procedure

- Set the **replicas** parameters for the **repo** and **server** spec to the number of replicas you want to run:

Example Argo CD custom resource

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: repo
spec:
  repo:
    replicas: <number_of_replicas>
  server:
    replicas: <number_of_replicas>
    route:
      enabled: true
      path: /
      tls:
        insecureEdgeTerminationPolicy: Redirect
        termination: passthrough
        wildcardPolicy: None
```

4.5.3. Deploying resources to a different namespace

To allow Argo CD to manage resources in other namespaces apart from where it is installed, configure the target namespace with a **argocd.argoproj.io/managed-by** label.

Procedure

- Configure the namespace:

```
$ oc label namespace <namespace> \
  argocd.argoproj.io/managed-by=<namespace> 1
```

- 1 The namespace where Argo CD is installed.

4.6. CONFIGURING AN OPENSIFT CLUSTER BY DEPLOYING AN APPLICATION WITH CLUSTER CONFIGURATIONS

With Red Hat OpenShift GitOps, you can configure Argo CD to recursively sync the content of a Git directory with an application that contains custom configurations for your cluster.

Prerequisites

- Red Hat OpenShift GitOps is installed in your cluster.
- Logged into Argo CD instance.

4.6.1. Running the Argo CD instance at the cluster-level

The default Argo CD instance and the accompanying controllers, installed by the Red Hat OpenShift GitOps Operator, can now run on the infrastructure nodes of the cluster by setting a simple configuration toggle.

Procedure

1. Label the existing nodes:

```
$ oc label node <node-name> node-role.kubernetes.io/infra=""
```

2. Optional: If required, you can also apply taints and isolate the workloads on infrastructure nodes and prevent other workloads from scheduling on these nodes:

```
$ oc adm taint nodes -l node-role.kubernetes.io/infra \
  infra=reserved:NoSchedule infra=reserved:NoExecute
```

3. Add the **runOnInfra** toggle in the **GitOpsService** custom resource:

```
apiVersion: pipelines.openshift.io/v1alpha1
kind: GitopsService
metadata:
  name: cluster
spec:
  runOnInfra: true
```

4. Optional: If taints have been added to the nodes, then add **tolerations** to the **GitOpsService** custom resource, for example:

```
spec:
  runOnInfra: true
  tolerations:
```

```

- effect: NoSchedule
  key: infra
  value: reserved
- effect: NoExecute
  key: infra
  value: reserved

```

5. Verify that the workloads in the **openshift-gitops** namespace are now scheduled on the infrastructure nodes by viewing **Pods → Pod details** for any pod in the console UI.



NOTE

Any **nodeSelectors** and **tolerations** manually added to the default Argo CD custom resource are overwritten by the toggle and **tolerations** in the **GitOpsService** custom resource.

4.6.2. Creating an application by using the Argo CD dashboard

Argo CD provides a dashboard which allows you to create applications.

This sample workflow walks you through the process of configuring Argo CD to recursively sync the content of the **cluster** directory to the **cluster-configs** application. The directory defines the OpenShift Container Platform web console cluster configurations that add a link to the **Red Hat**



Developer Blog - Kubernetes under the menu in the web console, and defines a namespace **spring-petclinic** on the cluster.

Procedure

1. In the Argo CD dashboard, click **NEW APP** to add a new Argo CD application.
2. For this workflow, create a **cluster-configs** application with the following configurations:

Application Name

cluster-configs

Project

default

Sync Policy

Manual

Repository URL

<https://github.com/redhat-developer/openshift-gitops-getting-started>

Revision

HEAD

Path

cluster

Destination

<https://kubernetes.default.svc>

Namespace

spring-petclinic

Directory Recurse checked

3. Click **CREATE** to create your application.
4. Open the **Administrator** perspective of the web console and navigate to **Administration** → **Namespaces** in the menu on the left.
5. Search for and select the namespace, then enter **argocd.argoproj.io/managed-by=openshift-gitops** in the **Label** field so that the Argo CD instance in the **openshift-gitops** namespace can manage your namespace.

4.6.3. Creating an application by using the oc tool

You can create Argo CD applications in your terminal by using the **oc** tool.

Procedure

1. Download [the sample application](#):

```
$ git clone git@github.com:redhat-developer/openshift-gitops-getting-started.git
```

2. Create the application:

```
$ oc create -f openshift-gitops-getting-started/argo/app.yaml
```

3. Run the **oc get** command to review the created application:

```
$ oc get application -n openshift-gitops
```

4. Add a label to the namespace your application is deployed in so that the Argo CD instance in the **openshift-gitops** namespace can manage it:


```
$ oc label namespace spring-petclinic argocd.argoproj.io/managed-by=openshift-gitops
```

4.6.4. Synchronizing your application with your Git repository

Procedure

1. In the Argo CD dashboard, notice that the **cluster-configs** Argo CD application has the statuses **Missing** and **OutOfSync**. Because the application was configured with a manual sync policy, Argo CD does not sync it automatically.
2. Click **SYNC** on the **cluster-configs** tile, review the changes, and then click **SYNCHRONIZE**. Argo CD will detect any changes in the Git repository automatically. If the configurations are changed, Argo CD will change the status of the **cluster-configs** to **OutOfSync**. You can modify the synchronization policy for Argo CD to automatically apply changes from your Git repository to the cluster.
3. Notice that the **cluster-configs** Argo CD application now has the statuses **Healthy** and **Synced**. Click the **cluster-configs** tile to check the details of the synchronized resources and their status on the cluster.



4. Navigate to the OpenShift Container Platform web console and click  to verify that a link to the **Red Hat Developer Blog - Kubernetes** is now present there.
5. Navigate to the **Project** page and search for the **spring-petclinic** namespace to verify that it has been added to the cluster.
Your cluster configurations have been successfully synchronized to the cluster.

4.6.5. In-built permissions for cluster configuration

By default, the Argo CD instance has permissions to manage specific cluster-scoped resources such as cluster Operators, optional OLM Operators and user management.



NOTE

Argo CD does not have cluster-admin permissions.

Permissions for the Argo CD instance:

Resources	Descriptions
Resource Groups	Configure the user or administrator
operators.coreos.com	Optional Operators managed by OLM
user.openshift.io , rbac.authorization.k8s.io	Groups, Users and their permissions
config.openshift.io	Control plane Operators managed by CVO used to configure cluster-wide build configuration, registry configuration and scheduler policies
storage.k8s.io	Storage
console.openshift.io	Console customization

4.6.6. Adding permissions for cluster configuration

You can grant permissions for an Argo CD instance to manage cluster configuration. Create a cluster role with additional permissions and then create a new cluster role binding to associate the cluster role with a service account.

Procedure

1. Log in to the OpenShift Container Platform web console as an admin.
2. In the web console, select **User Management** → **Roles** → **Create Role**. Use the following **ClusterRole** YAML template to add rules to specify the additional permissions.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
```

```

metadata:
  name: secrets-cluster-role
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["*"]

```

3. Click **Create** to add the cluster role.
4. Now create the cluster role binding. In the web console, select **User Management → Role Bindings → Create Binding**.
5. Select **All Projects** from the **Project** drop-down.
6. Click **Create binding**.
7. Select **Binding type** as **Cluster-wide role binding (ClusterRoleBinding)**.
8. Enter a unique value for the **RoleBinding name**.
9. Select the newly created cluster role or an existing cluster role from the drop down list.
10. Select the **Subject** as **ServiceAccount** and the provide the **Subject namespace** and **name**.
 - a. **Subject namespace: openshift-gitops**
 - b. **Subject name: openshift-gitops-argocd-application-controller**
11. Click **Create**. The YAML file for the **ClusterRoleBinding** object is as follows:

```

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cluster-role-binding
subjects:
- kind: ServiceAccount
  name: openshift-gitops-argocd-application-controller
  namespace: openshift-gitops
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin

```

4.7. DEPLOYING A SPRING BOOT APPLICATION WITH ARGO CD

With Argo CD, you can deploy your applications to the OpenShift cluster either by using the Argo CD dashboard or by using the **oc** tool.

Prerequisites

- Red Hat OpenShift GitOps is installed in your cluster.
- Logged into Argo CD instance.

4.7.1. Creating an application by using the Argo CD dashboard

Argo CD provides a dashboard which allows you to create applications.

This sample workflow walks you through the process of configuring Argo CD to recursively sync the content of the **cluster** directory to the **cluster-configs** application. The directory defines the OpenShift Container Platform web console cluster configurations that add a link to the **Red Hat**



Developer Blog - Kubernetes under the menu in the web console, and defines a namespace **spring-petclinic** on the cluster.

Procedure

1. In the Argo CD dashboard, click **NEW APP** to add a new Argo CD application.
2. For this workflow, create a **cluster-configs** application with the following configurations:

Application Name

cluster-configs

Project

default

Sync Policy

Manual

Repository URL

<https://github.com/redhat-developer/openshift-gitops-getting-started>

Revision

HEAD

Path

cluster

Destination

<https://kubernetes.default.svc>

Namespace

spring-petclinic

Directory Recurse

checked

3. For this workflow, create a **spring-petclinic** application with the following configurations:

Application Name

spring-petclinic

Project

default

Sync Policy

Automatic

Repository URL

<https://github.com/redhat-developer/openshift-gitops-getting-started>

Revision

HEAD

Path

app

Destination

<https://kubernetes.default.svc>

Namespace

spring-petclinic

4. Click **CREATE** to create your application.
5. Open the **Administrator** perspective of the web console and navigate to **Administration** → **Namespaces** in the menu on the left.
6. Search for and select the namespace, then enter **argocd.argoproj.io/managed-by=openshift-gitops** in the **Label** field so that the Argo CD instance in the **openshift-gitops** namespace can manage your namespace.

4.7.2. Creating an application by using the oc tool

You can create Argo CD applications in your terminal by using the **oc** tool.

Procedure

1. Download [the sample application](#):

```
$ git clone git@github.com:redhat-developer/openshift-gitops-getting-started.git
```

2. Create the application:

```
$ oc create -f openshift-gitops-getting-started/argo/app.yaml
```

```
$ oc create -f openshift-gitops-getting-started/argo/app.yaml
```

3. Run the **oc get** command to review the created application:

```
$ oc get application -n openshift-gitops
```

4. Add a label to the namespace your application is deployed in so that the Argo CD instance in the **openshift-gitops** namespace can manage it:

```
$ oc label namespace spring-petclinic argocd.argoproj.io/managed-by=openshift-gitops
```

```
$ oc label namespace spring-petclinic argocd.argoproj.io/managed-by=openshift-gitops
```

4.7.3. Verifying Argo CD self-healing behavior

Argo CD constantly monitors the state of deployed applications, detects differences between the specified manifests in Git and live changes in the cluster, and then automatically corrects them. This behavior is referred to as self-healing.

You can test and observe the self-healing behavior in Argo CD.

Prerequisites

- The sample **app-spring-petclinic** application is deployed and configured.

Procedure

1. In the Argo CD dashboard, verify that your application has the **Synced** status.
2. Click the **app-spring-petclinic** tile in the Argo CD dashboard to view the application resources that are deployed to the cluster.
3. In the OpenShift Container Platform web console, navigate to the **Developer** perspective.
4. Modify the Spring PetClinic deployment and commit the changes to the **app/** directory of the Git repository. Argo CD will automatically deploy the changes to the cluster.
 - a. Fork the [OpenShift GitOps getting started repository](#).
 - b. In the **deployment.yaml** file, change the **failureThreshold** value to **5**.
 - c. In the deployment cluster, run the following command to verify the changed value of the **failureThreshold** field:

```
$ oc edit deployment spring-petclinic -n spring-petclinic
```

5. Test the self-healing behavior by modifying the deployment on the cluster and scaling it up to two pods while watching the application in the OpenShift Container Platform web console.
 - a. Run the following command to modify the deployment:

```
$ oc scale deployment spring-petclinic --replicas 2 -n spring-petclinic
```
 - b. In the OpenShift Container Platform web console, notice that the deployment scales up to two pods and immediately scales down again to one pod. Argo CD detected a difference from the Git repository and auto-healed the application on the OpenShift Container Platform cluster.
6. In the Argo CD dashboard, click the **app-spring-petclinic** tile → **APP DETAILS** → **EVENTS**. The **EVENTS** tab displays the following events: Argo CD detecting out of sync deployment resources on the cluster and then resyncing the Git repository to correct it.

4.8. ARGO CD OPERATOR

The **ArgoCD** custom resource is a Kubernetes Custom Resource (CRD) that describes the desired state for a given Argo CD cluster that allows you to configure the components which make up an Argo CD cluster.

4.8.1. Argo CD CLI tool

The Argo CD CLI tool is a tool used to configure Argo CD through the command line. Red Hat OpenShift GitOps does not support this binary. Use the OpenShift Console to configure the Argo CD.

4.8.2. Argo CD custom resource properties

The Argo CD Custom Resource consists of the following properties:

Name	Description	Default	Properties
ApplicationInstanceLabelKey	The metadata.label key name where Argo CD injects the app name as a tracking label.	app.kubernetes.io/instance	

ApplicationSet	ApplicationSet controller configuration options.	<Object>	<ul style="list-style-type: none"> ● <Image> - The container image for the ApplicationSet controller. This overrides the ARGOCD_APPLICATIONS_ET_IMAGE environment variable. ● <Version> - The tag to use with the ApplicationSet container image. ● <Resources> - The container compute resources. ● <LogLevel> - The log level used by the Argo CD Application Controller component. Valid options are debug, info, error, and warn. ● <LogFormat> - The log format used by the Argo CD Application Controller component. Valid options are text or json. ● <ParallelismLimit> - The kubectl parallelism limit to set for the controller (--kubectl-parallelism-limit flag).
----------------	--	----------	---

ConfigManagementPlugins	Add a configuration management plugin.	<empty>	
Controller	Argo CD Application Controller options.	<Object>	<ul style="list-style-type: none"> ● <Processors.Operation> - The number of operation processors. ● <Processors.Status> - The number of status processors. ● <Resources> - The container compute resources. ● <LogLevel> - The log level used by the Argo CD Application Controller component. Valid options are debug, info, error, and warn. ● <AppSync> - AppSync is used to control the sync frequency of Argo CD applications ● <Sharding.enabled> - Enable sharding on the Argo CD Application Controller component. This property is used to manage a large number of clusters to relieve memory pressure on the controller component. ● <Sharding.replicas> - The number of replicas that will be used to support

			sharding of the Argo CD Application Controller. <ul style="list-style-type: none"> • <code><Env></code> - Environment to set for the application controller workloads.
DisableAdmin	Disables the built-in admin user.	false	
GATrackingID	Use a Google Analytics tracking ID.	<code><empty></code>	
GAAnonymizeusers	Enable hashed usernames sent to google analytics.	false	
HA	High availability options.	<code><Object></code>	<ul style="list-style-type: none"> • <code><Enabled></code> - Toggle high availability support globally for Argo CD. • <code><RedisProxyImage></code> - The Redis HAProxy container image. This overrides the ARGOCD_REDIS_HA_PROXY_IMAGE environment variable. • <code><RedisProxyVersion></code> - The tag to use for the Redis HAProxy container image.
HelpChatURL	URL for getting chat help (this will typically be your Slack channel for support).	https://mycorp.slack.com/argo-cd	
HelpChatText	The text that appears in a text box for getting chat help.	Chat now!	

Image	The container image for all Argo CD components. This overrides the ARGOCD_IMAGE environment variable.	argoproj/argocd	
Ingress	Ingress configuration options.	<Object>	
InitialRepositories	Initial Git repositories to configure Argo CD to use upon creation of the cluster.	<empty>	
Notifications	Notifications controller configuration options.	<Object>	<ul style="list-style-type: none"> ● <Enabled> - The toggle to start the notifications-controller. ● <Image> - The container image for all Argo CD components. This overrides the ARGOCD_IMAGE environment variable. ● <Version> - The tag to use with the Notifications container image. ● <Resources> - The container compute resources. ● <LogLevel> - The log level used by the Argo CD Application Controller component. Valid options are debug, info, error, and warn.

RepositoryCredentials	Git repository credential templates to configure Argo CD to use upon creation of the cluster.	<empty>	
InitialSSHKnownHosts	Initial SSH Known Hosts for Argo CD to use upon creation of the cluster.	<default_Argo_CD_Known_Hosts>	
KustomizeBuildOptions	The build options and parameters to use with kustomize build .	<empty>	
OIDCConfig	The OIDC configuration as an alternative to Dex.	<empty>	
NodePlacement	Add the nodeSelector and the tolerations .	<empty>	
Prometheus	Prometheus configuration options.	<Object>	<ul style="list-style-type: none"> ● <Enabled> - Toggle Prometheus support globally for Argo CD. ● <Host> - The hostname to use for Ingress or Route resources. ● <Ingress> - Toggles Ingress for Prometheus. ● <Route> - Route configuration options. ● <Size> - The replica count for the Prometheus StatefulSet.

RBAC	RBAC configuration options.	<Object>	<ul style="list-style-type: none">• <DefaultPolicy> - The policy.default property in the argocd-rbac-cm config map. The name of the default role which Argo CD will fall back to, when authorizing API requests.• <Policy> - The policy.csv property in the argocd-rbac-cm config map. CSV data containing user-defined RBAC policies and role definitions.• <Scopes> - The scopes property in the argocd-rbac-cm config map. Controls which OIDC scopes to examine during RBAC enforcement (in addition to sub scope).
------	-----------------------------	----------	---

Redis	Redis configuration options.	<Object>	<ul style="list-style-type: none"> ● <AutoTLS> - Use the provider to create the Redis server's TLS certificate (one of: openshift). Currently only available for OpenShift Container Platform. ● <DisableTLSVerification> - Define whether the Redis server should be accessed using strict TLS validation. ● <Image> - The container image for Redis. This overrides the ARGOCD_REDIS_IMAGE environment variable. ● <Resources> - The container compute resources. ● <Version> - The tag to use with the Redis container image.
ResourceCustomizations	Customize resource behavior.	<empty>	
ResourceExclusions	Completely ignore entire classes of resource group.	<empty>	
ResourceInclusions	The configuration to configure which resource group/kinds are applied.	<empty>	
Server	Argo CD Server configuration options.	<Object>	<ul style="list-style-type: none"> ● <Autoscale> - Server

autoscale
configuration
options.

- *<ExtraCommandArgs>* - List of arguments added to the existing arguments set by the Operator.
- *<GRPC>* - GRPC configuration options.
- *<Host>* - The hostname used for Ingress or Route resources.
- *<Ingress>* - Ingress configuration for the Argo CD server component.
- *<Insecure>* - Toggles the insecure flag for Argo CD server.
- *<Resources>* - The container compute resources.
- *<Replicas>* - The number of replicas for the Argo CD server. Must be greater than or equal to **0**. If **Autoscale** is enabled, **Replicas** is ignored.
- *<Route>* - Route configuration options.
- *<Service.Type>* - The **ServiceType** used for the service resource.

			<ul style="list-style-type: none">● <i><LogLevel></i> - The log level to be used by the Argo CD Server component. Valid options are debug, info, error, and warn.● <i><LogFormat></i> - The log format used by the Argo CD Application Controller component. Valid options are text or json.● <i><Env></i> - Environment to set for the server workloads.
--	--	--	---

SSO	Single Sign-on options.	<Object>	<ul style="list-style-type: none"> ● <Image> - The container image for Keycloak. This overrides the ARGOCD_KEYCLOAK_IMAGE environment variable. ● <Keycloak> - Configuration options for Keycloak SSO provider. ● <Dex> - Configuration options for Dex SSO provider. ● <Provider> - The name of the provider used to configure Single Sign-on. For now the supported options are Dex and Keycloak. ● <Resources> - The container compute resources. ● <VerifyTLS> - Whether to enforce strict TLS checking when communicating with Keycloak service. ● <Version> - The tag to use with the Keycloak container image.
StatusBadgeEnabled	Enable application status badge.	true	

TLS	TLS configuration options.	<Object>	<ul style="list-style-type: none"> • <CA.ConfigMapName> - The name of the ConfigMap which contains the CA certificate. • <CA.SecretName> - The name of the secret which contains the CA Certificate and Key. • <InitialCerts> - Initial set of certificates in the argocd-tls-certs-cm config map for connecting Git repositories via HTTPS.
UserAnonymousEnabled	Enable anonymous user access.	true	
Version	The tag to use with the container image for all Argo CD components.	Latest Argo CD version	
Banner	Add a UI banner message.	<Object>	<ul style="list-style-type: none"> • <Banner.Content> - The banner message content (required if a banner is displayed). • <Banner.URL.SecretName> - The banner message link URL (optional).

4.8.3. Repo server properties

The following properties are available for configuring the Repo server component:

Name	Default	Description
Resources	<empty>	The container compute resources.

MountSAToken	false	Whether the ServiceAccount token should be mounted to the repo-server pod.
ServiceAccount	""	The name of the ServiceAccount to use with the repo-server pod.
VerifyTLS	false	Whether to enforce strict TLS checking on all components when communicating with repo server.
AutoTLS	""	Provider to use for setting up TLS the repo-server's gRPC TLS certificate (one of: openshift). Currently only available for OpenShift.
Image	argoproj/argocd	The container image for Argo CD Repo server. This overrides the ARGOCD_REPOSERVER_IMAGE environment variable.
Version	same as .spec.Version	The tag to use with the Argo CD Repo server.
LogLevel	info	The log level used by the Argo CD Repo server. Valid options are debug, info, error, and warn.
LogFormat	text	The log format to be used by the Argo CD Repo server. Valid options are text or json.
ExecTimeout	180	Execution timeout in seconds for rendering tools (e.g. Helm, Kustomize).
Env	<empty>	Environment to set for the repository server workloads.
Replicas	<empty>	The number of replicas for the Argo CD Repo server. Must be greater than or equal to 0 .

4.8.4. Enabling notifications with Argo CD instance

To enable or disable the [Argo CD notifications controller](#), set a parameter in the Argo CD custom resource. By default, notifications are disabled. To enable notifications, set the **enabled** parameter to **true** in the **.yaml** file:

Procedure

1. Set the **enabled** parameter to **true**:

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  notifications:
    enabled: true
```

4.9. MONITORING HEALTH INFORMATION FOR APPLICATION RESOURCES AND DEPLOYMENTS

The environment details page displays the health status of the application resources, such as routes, synchronization status, deployment configuration and deployment history.

4.9.1. Checking health information

The Red Hat OpenShift GitOps Operator will install the GitOps backend service in the **openshift-gitops** namespace.

Prerequisites

- The Red Hat OpenShift GitOps Operator is installed from **OperatorHub**.
- Argo CD applications are in sync.

Procedure

1. Click **Environments** under the **Developer** perspective. The **Environments** page shows the list of applications along with their **Environment status**.
2. Hover over the icons under the **Environment status** column to see the synchronization status of all the environments.
3. Click the application name from the list to view the details of a specific application.
4. If the **Resources** section displays icons, hover over the icons to get status details.
 - A broken heart indicates that resource issues have degraded the application's performance.
 - A yellow yield sign indicates that resource issues have delayed data about the application's health.
5. To view the deployment history of an application, click the **Deployment History** tab. The page includes details such as the **Last deployment**, **Description** (commit message), **Environment**, **Author**, and **Revision**.

4.10. CONFIGURING SSO FOR ARGO CD USING DEX

After the Red Hat OpenShift GitOps Operator is installed, Argo CD automatically creates a user with **admin** permissions. To manage multiple users, cluster administrators can use Argo CD to configure Single Sign-On (SSO).

4.10.1. Enabling the Dex OpenShift OAuth Connector

Dex uses the users and groups defined within OpenShift by checking the **OAuth** server provided by the platform. The following example shows the properties of Dex along with example configurations:

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: openshift-oauth
spec:
  dex:
    openShiftOAuth: true ❶
    groups: ❷
    - default
  rbac: ❸
    defaultPolicy: 'role:readonly'
    policy: |
      g, cluster-admins, role:admin
    scopes: '[groups]'
```

- ❶ The **openShiftOAuth** property triggers the Operator to automatically configure the built-in OpenShift **OAuth** server when the value is set to **true**.
- ❷ The **groups** property allows users of the specified group(s) to log in.
- ❸ The RBAC policy property assigns the admin role in the Argo CD cluster to users in the OpenShift **cluster-admins** group.

4.10.1.1. Mapping users to specific roles

Argo CD cannot map users to specific roles if they have a direct **ClusterRoleBinding** role. You can manually change the role as **role:admin** on SSO through OpenShift.

Procedure

1. Create a group named **cluster-admins**.

```
$ oc adm groups new cluster-admins
```

2. Add the user to the group.

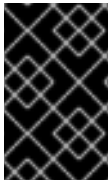
```
$ oc adm groups add-users cluster-admins USER
```

3. Apply the **cluster-admin ClusterRole** to the group:

```
$ oc adm policy add-cluster-role-to-group cluster-admin cluster-admins
```

4.10.2. Disabling Dex

Dex is installed by default for all the Argo CD instances created by the Operator. You can configure Red Hat OpenShift GitOps to use Dex as the SSO authentication provider by setting the **.spec.dex** parameter.



IMPORTANT

In Red Hat OpenShift GitOps v1.6.0, **DISABLE_DEX** is deprecated and is planned to be removed in Red Hat OpenShift GitOps v1.9.0. Consider using the **.spec.sso.dex** parameter instead. See "Enabling or disabling Dex using .spec.sso".

Procedure

- Set the environmental variable **DISABLE_DEX** to **true** in the YAML resource of the Operator:

```
...
spec:
  config:
    env:
      - name: DISABLE_DEX
        value: "true"
...
```

4.10.3. Enabling or disabling Dex using .spec.sso

You can configure Red Hat OpenShift GitOps to use Dex as its SSO authentication provider by setting the **.spec.sso** parameter.

Procedure

1. To enable Dex, set the **.spec.sso.provider: dex** parameter in the YAML resource of the Operator:

```
...
spec:
  sso:
    provider: dex
    dex:
      openShiftOAuth: true
...
```

2. To disable dex, either remove the **spec.sso** element from the Argo CD custom resource, or specify a different SSO provider.

4.11. CONFIGURING SSO FOR ARGO CD USING KEYCLOAK

After the Red Hat OpenShift GitOps Operator is installed, Argo CD automatically creates a user with **admin** permissions. To manage multiple users, cluster administrators can use Argo CD to configure Single Sign-On (SSO).

Prerequisites

- Red Hat SSO is installed on the cluster.
- Argo CD is installed on the cluster.

4.11.1. Configuring a new client in Keycloak

Dex is installed by default for all the Argo CD instances created by the Operator. However, you can delete the Dex configuration and add Keycloak instead to log in to Argo CD using your OpenShift credentials. Keycloak acts as an identity broker between Argo CD and OpenShift.

Procedure

To configure Keycloak, follow these steps:

1. Delete the Dex configuration by removing the following section from the Argo CD Custom Resource (CR), and save the CR:

```
dex:
  openShiftOAuth: true
  resources:
    limits:
      cpu:
      memory:
    requests:
      cpu:
      memory:
```

2. Configure Keycloak by editing the Argo CD CR, and updating the value for the **provider** parameter as **keycloak**. For example:

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
  server:
    route:
      enabled: true
```



NOTE

The Keycloak instance takes 2-3 minutes to install and run.

4.11.2. Logging in to Keycloak

Log in to the Keycloak console to manage identities or roles and define the permissions assigned to the various roles.

Prerequisites

- The default configuration of Dex is removed.
- Your Argo CD CR must be configured to use the Keycloak SSO provider.

Procedure

1. Get the Keycloak route URL for login:

```
$ oc -n argocd get route keycloak
```

NAME	HOST/PORT	PATH	SERVICES	PORT
TERMINATION	WILDCARD			
keycloak	keycloak-default.apps.ci-ln-*****.origin-ci-int-aws.dev.**.com		keycloak	<all>
reencrypt	None			

2. Get the Keycloak pod name that stores the user name and password as environment variables:

```
$ oc -n argocd get pods
```

NAME	READY	STATUS	RESTARTS	AGE
keycloak-1-2sjcl	1/1	Running	0	45m

- a. Get the Keycloak user name:

```
$ oc -n argocd exec keycloak-1-2sjcl -- "env" | grep SSO_ADMIN_USERNAME
```

```
SSO_ADMIN_USERNAME=Cqid54lh
```

- b. Get the Keycloak password:

```
$ oc -n argocd exec keycloak-1-2sjcl -- "env" | grep SSO_ADMIN_PASSWORD
```

```
SSO_ADMIN_PASSWORD=GVXxHifH
```

3. On the login page, click **LOG IN VIA KEYCLOAK**



NOTE

You only see the option **LOGIN VIA KEYCLOAK** after the Keycloak instance is ready.

4. Click **Login with OpenShift**.



NOTE

Login using **kubeadmin** is not supported.

5. Enter the OpenShift credentials to log in.
6. Optional: By default, any user logged in to Argo CD has read-only access. You can manage the user level access by updating the **argocd-rbac-cm** config map:


```
policy.csv:
<name>, <email>, role:admin
```

4.11.3. Uninstalling Keycloak

You can delete the Keycloak resources and their relevant configurations by removing the **SSO** field from the Argo CD Custom Resource (CR) file. After you remove the **SSO** field, the values in the file look similar to the following:

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  server:
    route:
      enabled: true
```



NOTE

A Keycloak application created by using this method is currently not persistent. Additional configurations created in the Argo CD Keycloak realm are deleted when the server restarts.

4.12. CONFIGURING ARGO CD RBAC

By default, if you are logged into Argo CD using RHSSO, you are a read-only user. You can change and manage the user level access.

4.12.1. Configuring user level access

To manage and modify the user level access, configure the RBAC section in Argo CD custom resource.

Procedure

- Edit the **argocd** Custom Resource:

```
$ oc edit argocd [argocd-instance-name] -n [namespace]
```

Output

```
metadata
...
...
rbac:
  policy: 'g, rbacsystem:cluster-admins, role:admin'
  scopes: '[groups]'
```

- Add the **policy** configuration to the **rbac** section and add the **name**, **email** and the **role** of the user:

```

metadata
...
...
rbac:
  policy: <name>, <email>, role:<admin>
  scopes: '[groups]'

```

**NOTE**

Currently, RHSSO cannot read the group information of Red Hat OpenShift GitOps users. Therefore, configure the RBAC at the user level.

4.12.2. Modifying RHSSO resource requests/limits

By default, the RHSSO container is created with resource requests and limitations. You can change and manage the resource requests.

Resource	Requests	Limits
CPU	500	1000m
Memory	512 Mi	1024 Mi

Procedure

Modify the default resource requirements patching the Argo CD CR:

```

$ oc -n openshift-gitops patch argocd openshift-gitops --type=json -p='[{"op": "add", "path":
"/spec/sso", "value": {"provider": "keycloak", "resources": {"requests": {"cpu": "512m", "memory":
"512Mi"}, "limits": {"cpu": "1024m", "memory": "1024Mi"}}}]'

```

**NOTE**

RHSSO created by the Red Hat OpenShift GitOps only persists the changes that are made by the operator. If the RHSSO restarts, any additional configuration created by the Admin in RHSSO is deleted.

4.13. CONFIGURING RESOURCE QUOTA OR REQUESTS

With the Argo CD Custom Resource, you can create, update, and delete resource requests and limits for Argo CD workloads.

4.13.1. Configuring workloads with resource requests and limits

You can create Argo CD custom resource workloads with resource requests and limits. This is required when you want to deploy the Argo CD instance in a namespace that is configured with resource quotas.

The following Argo CD instance deploys the Argo CD workloads such as **Application Controller**, **ApplicationSet Controller**, **Dex**, **Redis**, **Repo Server**, and **Server** with resource requests and limits. You can also create the other workloads with resource requirements in the same manner.

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example
spec:
  server:
    resources:
      limits:
        cpu: 500m
        memory: 256Mi
      requests:
        cpu: 125m
        memory: 128Mi
    route:
      enabled: true
  applicationSet:
    resources:
      limits:
        cpu: '2'
        memory: 1Gi
      requests:
        cpu: 250m
        memory: 512Mi
  repo:
    resources:
      limits:
        cpu: '1'
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 256Mi
  dex:
    resources:
      limits:
        cpu: 500m
        memory: 256Mi
      requests:
        cpu: 250m
        memory: 128Mi
  redis:
    resources:
      limits:
        cpu: 500m
        memory: 256Mi
      requests:
        cpu: 250m
        memory: 128Mi
  controller:
    resources:
      limits:
        cpu: '2'
        memory: 2Gi
      requests:
        cpu: 250m
        memory: 1Gi
```

4.13.2. Patching Argo CD instance to update the resource requirements

You can update the resource requirements for all or any of the workloads post installation.

Procedure

Update the **Application Controller** resource requests of an Argo CD instance in the Argo CD namespace.

```
oc -n argocd patch argocd example --type='json' -p='[{"op": "replace", "path":
"/spec/controller/resources/requests/cpu", "value": "1"}]'

oc -n argocd patch argocd example --type='json' -p='[{"op": "replace", "path":
"/spec/controller/resources/requests/memory", "value": "512Mi"}]'
```

4.13.3. Removing resource requests

You can also remove resource requirements for all or any of your workloads after installation.

Procedure

Remove the **Application Controller** resource requests of an Argo CD instance in the Argo CD namespace.

```
oc -n argocd patch argocd example --type='json' -p='[{"op": "remove", "path":
"/spec/controller/resources/requests/cpu"}]'

oc -n argocd argocd patch argocd example --type='json' -p='[{"op": "remove", "path":
"/spec/controller/resources/requests/memory"}]'
```

4.14. RUNNING GITOPS CONTROL PLANE WORKLOADS ON INFRASTRUCTURE NODES

You can use infrastructure nodes to prevent additional billing cost against subscription counts.

You can use the OpenShift Container Platform to run certain workloads on infrastructure nodes installed by the Red Hat OpenShift GitOps Operator. This comprises the workloads that are installed by the Red Hat OpenShift GitOps Operator by default in the **openshift-gitops** namespace, including the default Argo CD instance in that namespace.



NOTE

Any other Argo CD instances installed to user namespaces are not eligible to run on Infrastructure nodes.

4.14.1. Moving GitOps workloads to infrastructure nodes

You can move the default workloads installed by the Red Hat OpenShift GitOps to the infrastructure nodes. The workloads that can be moved are:

- **kam deployment**
- **cluster deployment** (backend service)

- **openshift-gitops-applicationset-controller deployment**
- **openshift-gitops-dex-server deployment**
- **openshift-gitops-redis deployment**
- **openshift-gitops-redis-ha-haproxy deployment**
- **openshift-gitops-repo-sever deployment**
- **openshift-gitops-server deployment**
- **openshift-gitops-application-controller statefulset**
- **openshift-gitops-redis-server statefulset**

Procedure

1. Label existing nodes as infrastructure by running the following command:

```
$ oc label node <node-name> node-role.kubernetes.io/infra=
```

2. Edit the **GitOpsService** Custom Resource (CR) to add the infrastructure node selector:

```
$ oc edit gitopsservice -n openshift-gitops
```

3. In the **GitOpsService** CR file, add **runOnInfra** field to the **spec** section and set it to **true**. This field moves the workloads in **openshift-gitops** namespace to the infrastructure nodes:

```
apiVersion: pipelines.openshift.io/v1alpha1
kind: GitopsService
metadata:
  name: cluster
spec:
  runOnInfra: true
```

4. Optional: Apply taints and isolate the workloads on infrastructure nodes and prevent other workloads from scheduling on these nodes.

```
$ oc adm taint nodes -l node-role.kubernetes.io/infra
infra=reserved:NoSchedule infra=reserved:NoExecute
```

5. Optional: If you apply taints to the nodes, you can add tolerations in the **GitOpsService** CR:

```
spec:
  runOnInfra: true
  tolerations:
    - effect: NoSchedule
      key: infra
      value: reserved
    - effect: NoExecute
      key: infra
      value: reserved
```

To verify that the workloads are scheduled on infrastructure nodes in the Red Hat OpenShift GitOps namespace, click any of the pod names and ensure that the **Node selector** and **Tolerations** have been added.



NOTE

Any manually added **Node selectors** and **Tolerations** in the default Argo CD CR will be overwritten by the toggle and the tolerations in the **GitOpsService** CR.

4.15. SIZING REQUIREMENTS FOR GITOPS OPERATOR

The sizing requirements page displays the sizing requirements for installing Red Hat OpenShift GitOps on OpenShift Container Platform. It also provides the sizing details for the default ArgoCD instance that is instantiated by the GitOps Operator.

4.15.1. Sizing requirements for GitOps

Red Hat OpenShift GitOps is a declarative way to implement continuous deployment for cloud-native applications. Through GitOps, you can define and configure the CPU and memory requirements of your application.

Every time you install the Red Hat OpenShift GitOps Operator, the resources on the namespace are installed within the defined limits. If the default installation does not set any limits or requests, the Operator fails within the namespace with quotas. Without enough resources, the cluster cannot schedule ArgoCD related pods. The following table details the resource requests and limits for the default workloads:

Workload	CPU requests	CPU limits	Memory requests	Memory limits
argocd-application-controller	1	2	1024M	2048M
applicationset-controller	1	2	512M	1024M
argocd-server	0.125	0.5	128M	256M
argocd-repo-server	0.5	1	256M	1024M
argocd-redis	0.25	0.5	128M	256M
argocd-dex	0.25	0.5	128M	256M
HAProxy	0.25	0.5	128M	256M

Optionally, you can also use the ArgoCD custom resource with the **oc** command to see the specifics and modify them:

```
oc edit argocd <name of argo cd> -n namespace
```

■

CHAPTER 5. JENKINS

5.1. CONFIGURING JENKINS IMAGES

OpenShift Container Platform provides a container image for running Jenkins. This image provides a Jenkins server instance, which can be used to set up a basic flow for continuous testing, integration, and delivery.

The image is based on the Red Hat Universal Base Images (UBI).

OpenShift Container Platform follows the [LTS](#) release of Jenkins. OpenShift Container Platform provides an image that contains Jenkins 2.x.

The OpenShift Container Platform Jenkins images are available on [Quay.io](#) or [registry.redhat.io](#).

For example:

```
$ podman pull registry.redhat.io/ocp-tools-4/jenkins-rhel8:<image_tag>
```

To use these images, you can either access them directly from these registries or push them into your OpenShift Container Platform container image registry. Additionally, you can create an image stream that points to the image, either in your container image registry or at the external location. Your OpenShift Container Platform resources can then reference the image stream.

But for convenience, OpenShift Container Platform provides image streams in the **openshift** namespace for the core Jenkins image as well as the example Agent images provided for OpenShift Container Platform integration with Jenkins.

5.1.1. Configuration and customization

You can manage Jenkins authentication in two ways:

- OpenShift Container Platform OAuth authentication provided by the OpenShift Container Platform Login plugin.
- Standard authentication provided by Jenkins.

5.1.1.1. OpenShift Container Platform OAuth authentication

OAuth authentication is activated by configuring options on the **Configure Global Security** panel in the Jenkins UI, or by setting the **OPENSIFT_ENABLE_OAUTH** environment variable on the Jenkins **Deployment configuration** to anything other than **false**. This activates the OpenShift Container Platform Login plugin, which retrieves the configuration information from pod data or by interacting with the OpenShift Container Platform API server.

Valid credentials are controlled by the OpenShift Container Platform identity provider.

Jenkins supports both browser and non-browser access.

Valid users are automatically added to the Jenkins authorization matrix at log in, where OpenShift Container Platform roles dictate the specific Jenkins permissions that users have. The roles used by default are the predefined **admin**, **edit**, and **view**. The login plugin executes self-SAR requests against those roles in the project or namespace that Jenkins is running in.

Users with the **admin** role have the traditional Jenkins administrative user permissions. Users with the **edit** or **view** role have progressively fewer permissions.

The default OpenShift Container Platform **admin**, **edit**, and **view** roles and the Jenkins permissions those roles are assigned in the Jenkins instance are configurable.

When running Jenkins in an OpenShift Container Platform pod, the login plugin looks for a config map named **openshift-jenkins-login-plugin-config** in the namespace that Jenkins is running in.

If this plugin finds and can read in that config map, you can define the role to Jenkins Permission mappings. Specifically:

- The login plugin treats the key and value pairs in the config map as Jenkins permission to OpenShift Container Platform role mappings.
- The key is the Jenkins permission group short ID and the Jenkins permission short ID, with those two separated by a hyphen character.
- If you want to add the **Overall Jenkins Administer** permission to an OpenShift Container Platform role, the key should be **Overall-Administer**.
- To get a sense of which permission groups and permissions IDs are available, go to the matrix authorization page in the Jenkins console and IDs for the groups and individual permissions in the table they provide.
- The value of the key and value pair is the list of OpenShift Container Platform roles the permission should apply to, with each role separated by a comma.
- If you want to add the **Overall Jenkins Administer** permission to both the default **admin** and **edit** roles, as well as a new Jenkins role you have created, the value for the key **Overall-Administer** would be **admin,edit,jenkins**.



NOTE

The **admin** user that is pre-populated in the OpenShift Container Platform Jenkins image with administrative privileges is not given those privileges when OpenShift Container Platform OAuth is used. To grant these permissions the OpenShift Container Platform cluster administrator must explicitly define that user in the OpenShift Container Platform identity provider and assigns the **admin** role to the user.

Jenkins users' permissions that are stored can be changed after the users are initially established. The OpenShift Container Platform Login plugin polls the OpenShift Container Platform API server for permissions and updates the permissions stored in Jenkins for each user with the permissions retrieved from OpenShift Container Platform. If the Jenkins UI is used to update permissions for a Jenkins user, the permission changes are overwritten the next time the plugin polls OpenShift Container Platform.

You can control how often the polling occurs with the **OPENSSHIFT_PERMISSIONS_POLL_INTERVAL** environment variable. The default polling interval is five minutes.

The easiest way to create a new Jenkins service using OAuth authentication is to use a template.

5.1.1.2. Jenkins authentication

Jenkins authentication is used by default if the image is run directly, without using a template.

The first time Jenkins starts, the configuration is created along with the administrator user and

password. The default user credentials are **admin** and **password**. Configure the default password by setting the **JENKINS_PASSWORD** environment variable when using, and only when using, standard Jenkins authentication.

Procedure

- Create a Jenkins application that uses standard Jenkins authentication:

```
$ oc new-app -e \
  JENKINS_PASSWORD=<password> \
  ocp-tools-4/jenkins-rhel8
```

5.1.2. Jenkins environment variables

The Jenkins server can be configured with the following environment variables:

Variable	Definition	Example values and settings
OPENSIFT_ENABLE_OAUTH	Determines whether the OpenShift Container Platform Login plugin manages authentication when logging in to Jenkins. To enable, set to true .	Default: false
JENKINS_PASSWORD	The password for the admin user when using standard Jenkins authentication. Not applicable when OPENSIFT_ENABLE_OAUTH is set to true .	Default: password
JAVA_MAX_HEAP_PARAMETER, CONTAINER_HEAP_PERCENT, JENKINS_MAX_HEAP_UPPER_BOUND_MB	<p>These values control the maximum heap size of the Jenkins JVM. If JAVA_MAX_HEAP_PARAMETER is set, its value takes precedence. Otherwise, the maximum heap size is dynamically calculated as CONTAINER_HEAP_PERCENT of the container memory limit, optionally capped at JENKINS_MAX_HEAP_UPPER_BOUND_MB MiB.</p> <p>By default, the maximum heap size of the Jenkins JVM is set to 50% of the container memory limit with no cap.</p>	<p>JAVA_MAX_HEAP_PARAMETER example setting: -Xmx512m</p> <p>CONTAINER_HEAP_PERCENT default: 0.5, or 50%</p> <p>JENKINS_MAX_HEAP_UPPER_BOUND_MB example setting: 512 MiB</p>

Variable	Definition	Example values and settings
JAVA_INITIAL_HEAP_PARAMETER, CONTAINER_INITIAL_PERCENT	<p>These values control the initial heap size of the Jenkins JVM. If JAVA_INITIAL_HEAP_PARAMETER is set, its value takes precedence. Otherwise, the initial heap size is dynamically calculated as CONTAINER_INITIAL_PERCENT of the dynamically calculated maximum heap size.</p> <p>By default, the JVM sets the initial heap size.</p>	<p>JAVA_INITIAL_HEAP_PARAMETER example setting: -Xms32m</p> <p>CONTAINER_INITIAL_PERCENT example setting: 0.1, or 10%</p>
CONTAINER_CORE_LIMIT	If set, specifies an integer number of cores used for sizing numbers of internal JVM threads.	Example setting: 2
JAVA_TOOL_OPTIONS	Specifies options to apply to all JVMs running in this container. It is not recommended to override this value.	<p>Default: -</p> <p>XX:+UnlockExperimentalVMOptions -</p> <p>XX:+UseCGroupMemoryLimitForHeap -</p> <p>Dsun.zip.disableMemoryMapping=true</p>
JAVA_GC_OPTS	Specifies Jenkins JVM garbage collection parameters. It is not recommended to override this value.	<p>Default: -</p> <p>XX:+UseParallelGC -</p> <p>XX:MinHeapFreeRatio=5 -</p> <p>XX:MaxHeapFreeRatio=10 -</p> <p>XX:GCTimeRatio=4 -</p> <p>XX:AdaptiveSizePolicyWeight=90</p>
JENKINS_JAVA_OVERRIDES	Specifies additional options for the Jenkins JVM. These options are appended to all other options, including the Java options above, and may be used to override any of them if necessary. Separate each additional option with a space; if any option contains space characters, escape them with a backslash.	<p>Example settings: -Dfoo -Dbar; -</p> <p>Dfoo=first\ value -</p> <p>Dbar=second\ value.</p>
JENKINS_OPTS	Specifies arguments to Jenkins.	

Variable	Definition	Example values and settings
INSTALL_PLUGINS	Specifies additional Jenkins plugins to install when the container is first run or when OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS is set to true . Plugins are specified as a comma-delimited list of name:version pairs.	Example setting: git:3.7.0,subversion:2.10.2
OPENSIFT_PERMISSIONS_POLL_INTERVAL	Specifies the interval in milliseconds that the OpenShift Container Platform Login plugin polls OpenShift Container Platform for the permissions that are associated with each user that is defined in Jenkins.	Default: 300000 - 5 minutes
OVERRIDE_PV_CONFIG_WITH_IMAGE_CONFIG	When running this image with an OpenShift Container Platform persistent volume (PV) for the Jenkins configuration directory, the transfer of configuration from the image to the PV is performed only the first time the image starts because the PV is assigned when the persistent volume claim (PVC) is created. If you create a custom image that extends this image and updates the configuration in the custom image after the initial startup, the configuration is not copied over unless you set this environment variable to true .	Default: false
OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS	When running this image with an OpenShift Container Platform PV for the Jenkins configuration directory, the transfer of plugins from the image to the PV is performed only the first time the image starts because the PV is assigned when the PVC is created. If you create a custom image that extends this image and updates plugins in the custom image after the initial startup, the plugins are not copied over unless you set this environment variable to true .	Default: false

Variable	Definition	Example values and settings
ENABLE_FATAL_ERROR_LOG_FILE	When running this image with an OpenShift Container Platform PVC for the Jenkins configuration directory, this environment variable allows the fatal error log file to persist when a fatal error occurs. The fatal error file is saved at /var/lib/jenkins/logs .	Default: false
AGENT_BASE_IMAGE	Setting this value overrides the image used for the jnl container in the sample Kubernetes plugin pod templates provided with this image. Otherwise, the image from the jenkins-agent-base-rhel8:latest image stream tag in the openshift namespace is used.	Default: image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-rhel8:latest
JAVA_BUILDER_IMAGE	Setting this value overrides the image used for the java-builder container in the java-builder sample Kubernetes plugin pod templates provided with this image. Otherwise, the image from the java:latest image stream tag in the openshift namespace is used.	Default: image-registry.openshift-image-registry.svc:5000/openshift/java:latest
JAVA_FIPS_OPTIONS	Setting this value controls how the JVM operates when running on a FIPS node. For more information, see Configure OpenJDK 11 in FIPS mode .	Default: - Dcom.redhat.fips=false

5.1.3. Providing Jenkins cross project access

If you are going to run Jenkins somewhere other than your same project, you must provide an access token to Jenkins to access your project.

Procedure

1. Identify the secret for the service account that has appropriate permissions to access the project Jenkins must access:

```
$ oc describe serviceaccount jenkins
```

Example output

```
Name:      default
Labels:    <none>
Secrets:   { jenkins-token-uyswp  }
           { jenkins-dockercfg-xcr3d  }
Tokens:    jenkins-token-izv1u
           jenkins-token-uyswp
```

In this case the secret is named **jenkins-token-uyswp**.

2. Retrieve the token from the secret:

```
$ oc describe secret <secret name from above>
```

Example output

```
Name:      jenkins-token-uyswp
Labels:    <none>
Annotations:  kubernetes.io/service-account.name=jenkins,kubernetes.io/service-
account.uid=32f5b661-2a8f-11e5-9528-3c970e3bf0b7
Type:  kubernetes.io/service-account-token
Data
====
ca.crt: 1066 bytes
token: eyJhbGc...<content cut>....wRA
```

The token parameter contains the token value Jenkins requires to access the project.

5.1.4. Jenkins cross volume mount points

The Jenkins image can be run with mounted volumes to enable persistent storage for the configuration:

- **/var/lib/jenkins** is the data directory where Jenkins stores configuration files, including job definitions.

5.1.5. Customizing the Jenkins image through source-to-image

To customize the official OpenShift Container Platform Jenkins image, you can use the image as a source-to-image (S2I) builder.

You can use S2I to copy your custom Jenkins jobs definitions, add additional plugins, or replace the provided **config.xml** file with your own, custom, configuration.

To include your modifications in the Jenkins image, you must have a Git repository with the following directory structure:

plugins

This directory contains those binary Jenkins plugins you want to copy into Jenkins.

plugins.txt

This file lists the plugins you want to install using the following syntax:

```
pluginId:pluginVersion
```

configuration/jobs

This directory contains the Jenkins job definitions.

configuration/config.xml

This file contains your custom Jenkins configuration.

The contents of the **configuration/** directory is copied to the **/var/lib/jenkins/** directory, so you can also include additional files, such as **credentials.xml**, there.

Sample build configuration customizes the Jenkins image in OpenShift Container Platform

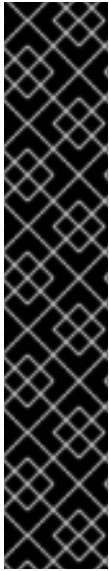
```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: custom-jenkins-build
spec:
  source: 1
    git:
      uri: https://github.com/custom/repository
      type: Git
  strategy: 2
    sourceStrategy:
      from:
        kind: ImageStreamTag
        name: jenkins:2
        namespace: openshift
      type: Source
  output: 3
    to:
      kind: ImageStreamTag
      name: custom-jenkins:latest
```

- 1 The **source** parameter defines the source Git repository with the layout described above.
- 2 The **strategy** parameter defines the original Jenkins image to use as a source image for the build.
- 3 The **output** parameter defines the resulting, customized Jenkins image that you can use in deployment configurations instead of the official Jenkins image.

5.1.6. Configuring the Jenkins Kubernetes plugin

The OpenShift Jenkins image includes the pre-installed [Kubernetes plugin for Jenkins](#) so that Jenkins agents can be dynamically provisioned on multiple container hosts using Kubernetes and OpenShift Container Platform.

To use the Kubernetes plugin, OpenShift Container Platform provides an OpenShift Agent Base image that is suitable for use as a Jenkins agent.



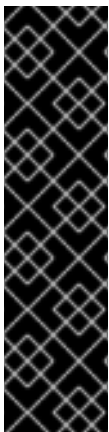
IMPORTANT

OpenShift Container Platform 4.11 moves the OpenShift Jenkins and OpenShift Agent Base images to the **ocp-tools-4** repository at **registry.redhat.io** so that Red Hat can produce and update the images outside the OpenShift Container Platform lifecycle. Previously, these images were in the OpenShift Container Platform install payload and the **openshift4** repository at **registry.redhat.io**.

The OpenShift Jenkins Maven and NodeJS Agent images were removed from the OpenShift Container Platform 4.11 payload. Red Hat no longer produces these images, and they are not available from the **ocp-tools-4** repository at **registry.redhat.io**. Red Hat maintains the 4.10 and earlier versions of these images for any significant bug fixes or security CVEs, following the [OpenShift Container Platform lifecycle policy](#).

For more information, see the "Important changes to OpenShift Jenkins images" link in the following "Additional resources" section.

The Maven and Node.js agent images are automatically configured as Kubernetes pod template images within the OpenShift Container Platform Jenkins image configuration for the Kubernetes plugin. That configuration includes labels for each image that you can apply to any of your Jenkins jobs under their **Restrict where this project can be run** setting. If the label is applied, jobs run under an OpenShift Container Platform pod running the respective agent image.



IMPORTANT

In OpenShift Container Platform 4.10 and later, the recommended pattern for running Jenkins agents using the Kubernetes plugin is to use pod templates with both **jnlp** and **sidecar** containers. The **jnlp** container uses the OpenShift Container Platform Jenkins Base agent image to facilitate launching a separate pod for your build. The **sidecar** container image has the tools needed to build in a particular language within the separate pod that was launched. Many container images from the Red Hat Container Catalog are referenced in the sample image streams in the **openshift** namespace. The OpenShift Container Platform Jenkins image has a pod template named **java-build** with sidecar containers that demonstrate this approach. This pod template uses the latest Java version provided by the **java** image stream in the **openshift** namespace.

The Jenkins image also provides auto-discovery and auto-configuration of additional agent images for the Kubernetes plugin.

With the OpenShift Container Platform sync plugin, on Jenkins startup, the Jenkins image searches within the project it is running, or the projects listed in the plugin's configuration, for the following items:

- Image streams with the **role** label set to **jenkins-agent**.
- Image stream tags with the **role** annotation set to **jenkins-agent**.
- Config maps with the **role** label set to **jenkins-agent**.

When the Jenkins image finds an image stream with the appropriate label, or an image stream tag with the appropriate annotation, it generates the corresponding Kubernetes plugin configuration. This way, you can assign your Jenkins jobs to run in a pod running the container image provided by the image stream.

The name and image references of the image stream, or image stream tag, are mapped to the name and image fields in the Kubernetes plugin pod template. You can control the label field of the Kubernetes plugin pod template by setting an annotation on the image stream, or image stream tag

object, with the key **agent-label**. Otherwise, the name is used as the label.



NOTE

Do not log in to the Jenkins console and change the pod template configuration. If you do so after the pod template is created, and the OpenShift Container Platform Sync plugin detects that the image associated with the image stream or image stream tag has changed, it replaces the pod template and overwrites those configuration changes. You cannot merge a new configuration with the existing configuration.

Consider the config map approach if you have more complex configuration needs.

When it finds a config map with the appropriate label, the Jenkins image assumes that any values in the key-value data payload of the config map contain Extensible Markup Language (XML) consistent with the configuration format for Jenkins and the Kubernetes plugin pod templates. One key advantage of config maps over image streams and image stream tags is that you can control all the Kubernetes plugin pod template parameters.

Sample config map for jenkins-agent

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: jenkins-agent
  labels:
    role: jenkins-agent
data:
  template1: |-
    <org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
    <inheritFrom></inheritFrom>
    <name>template1</name>
    <instanceCap>2147483647</instanceCap>
    <idleMinutes>0</idleMinutes>
    <label>template1</label>
    <serviceAccount>jenkins</serviceAccount>
    <nodeSelector></nodeSelector>
    <volumes/>
    <containers>
    <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
    <name>jnlp</name>
    <image>openshift/jenkins-agent-maven-35-centos7:v3.10</image>
    <privileged>>false</privileged>
    <alwaysPullImage>true</alwaysPullImage>
    <workingDir>/tmp</workingDir>
    <command></command>
    <args>${computer.jnlpmac} ${computer.name}</args>
    <ttyEnabled>>false</ttyEnabled>
    <resourceRequestCpu></resourceRequestCpu>
    <resourceRequestMemory></resourceRequestMemory>
    <resourceLimitCpu></resourceLimitCpu>
    <resourceLimitMemory></resourceLimitMemory>
    <envVars/>
    </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
    </containers>
    <envVars/>
```

```

    <annotations/>
    <imagePullSecrets/>
    <nodeProperties/>
  </org.csanchez.jenkins.plugins.kubernetes.PodTemplate>

```

The following example shows two containers that reference image streams in the **openshift** namespace. One container handles the JNLP contract for launching Pods as Jenkins Agents. The other container uses an image with tools for building code in a particular coding language:

```

kind: ConfigMap
apiVersion: v1
metadata:
  name: jenkins-agent
  labels:
    role: jenkins-agent
data:
  template2: |-
    <org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
      <inheritFrom></inheritFrom>
      <name>template2</name>
      <instanceCap>2147483647</instanceCap>
      <idleMinutes>0</idleMinutes>
      <label>template2</label>
      <serviceAccount>jenkins</serviceAccount>
      <nodeSelector></nodeSelector>
      <volumes/>
      <containers>
        <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
          <name>jnlp</name>
          <image>image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-
rhel8:latest</image>
          <privileged>>false</privileged>
          <alwaysPullImage>true</alwaysPullImage>
          <workingDir>/home/jenkins/agent</workingDir>
          <command></command>
          <args>$(JENKINS_SECRET) \$(JENKINS_NAME)</args>
          <ttyEnabled>>false</ttyEnabled>
          <resourceRequestCpu></resourceRequestCpu>
          <resourceRequestMemory></resourceRequestMemory>
          <resourceLimitCpu></resourceLimitCpu>
          <resourceLimitMemory></resourceLimitMemory>
          <envVars/>
        </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
        <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
          <name>java</name>
          <image>image-registry.openshift-image-registry.svc:5000/openshift/java:latest</image>
          <privileged>>false</privileged>
          <alwaysPullImage>true</alwaysPullImage>
          <workingDir>/home/jenkins/agent</workingDir>
          <command>cat</command>
          <args></args>
          <ttyEnabled>true</ttyEnabled>
          <resourceRequestCpu></resourceRequestCpu>
          <resourceRequestMemory></resourceRequestMemory>
          <resourceLimitCpu></resourceLimitCpu>
          <resourceLimitMemory></resourceLimitMemory>

```

```

    <envVars/>
  </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
</containers>
<envVars/>
<annotations/>
<imagePullSecrets/>
<nodeProperties/>
</org.csanchez.jenkins.plugins.kubernetes.PodTemplate>

```



NOTE

Do not log in to the Jenkins console and change the pod template configuration. If you do so after the pod template is created, and the OpenShift Container Platform Sync plugin detects that the image associated with the image stream or image stream tag has changed, it replaces the pod template and overwrites those configuration changes. You cannot merge a new configuration with the existing configuration.

Consider the config map approach if you have more complex configuration needs.

After it is installed, the OpenShift Container Platform Sync plugin monitors the API server of OpenShift Container Platform for updates to image streams, image stream tags, and config maps and adjusts the configuration of the Kubernetes plugin.

The following rules apply:

- Removing the label or annotation from the config map, image stream, or image stream tag deletes any existing **PodTemplate** from the configuration of the Kubernetes plugin.
- If those objects are removed, the corresponding configuration is removed from the Kubernetes plugin.
- If you create appropriately labeled or annotated **ConfigMap**, **ImageStream**, or **ImageStreamTag** objects, or add labels after their initial creation, this results in the creation of a **PodTemplate** in the Kubernetes-plugin configuration.
- In the case of the **PodTemplate** by config map form, changes to the config map data for the **PodTemplate** are applied to the **PodTemplate** settings in the Kubernetes plugin configuration. The changes also override any changes that were made to the **PodTemplate** through the Jenkins UI between changes to the config map.

To use a container image as a Jenkins agent, the image must run the agent as an entry point. For more details, see the official [Jenkins documentation](#).

Additional resources

- [Important changes to OpenShift Jenkins images](#)

5.1.7. Jenkins permissions

If in the config map the **<serviceAccount>** element of the pod template XML is the OpenShift Container Platform service account used for the resulting pod, the service account credentials are mounted into the pod. The permissions are associated with the service account and control which operations against the OpenShift Container Platform master are allowed from the pod.

Consider the following scenario with service accounts used for the pod, which is launched by the Kubernetes Plugin that runs in the OpenShift Container Platform Jenkins image.

If you use the example template for Jenkins that is provided by OpenShift Container Platform, the **jenkins** service account is defined with the **edit** role for the project Jenkins runs in, and the master Jenkins pod has that service account mounted.

The two default Maven and NodeJS pod templates that are injected into the Jenkins configuration are also set to use the same service account as the Jenkins master.

- Any pod templates that are automatically discovered by the OpenShift Container Platform sync plugin because their image streams or image stream tags have the required label or annotations are configured to use the Jenkins master service account as their service account.
- For the other ways you can provide a pod template definition into Jenkins and the Kubernetes plugin, you have to explicitly specify the service account to use. Those other ways include the Jenkins console, the **podTemplate** pipeline DSL that is provided by the Kubernetes plugin, or labeling a config map whose data is the XML configuration for a pod template.
- If you do not specify a value for the service account, the **default** service account is used.
- Ensure that whatever service account is used has the necessary permissions, roles, and so on defined within OpenShift Container Platform to manipulate whatever projects you choose to manipulate from the within the pod.

5.1.8. Creating a Jenkins service from a template

Templates provide parameter fields to define all the environment variables with predefined default values. OpenShift Container Platform provides templates to make creating a new Jenkins service easy. The Jenkins templates should be registered in the default **openshift** project by your cluster administrator during the initial cluster setup.

The two available templates both define deployment configuration and a service. The templates differ in their storage strategy, which affects whether the Jenkins content persists across a pod restart.



NOTE

A pod might be restarted when it is moved to another node or when an update of the deployment configuration triggers a redeployment.

- **jenkins-ephemeral** uses ephemeral storage. On pod restart, all data is lost. This template is only useful for development or testing.
- **jenkins-persistent** uses a Persistent Volume (PV) store. Data survives a pod restart.

To use a PV store, the cluster administrator must define a PV pool in the OpenShift Container Platform deployment.

After you select which template you want, you must instantiate the template to be able to use Jenkins.

Procedure

1. Create a new Jenkins application using one of the following methods:
 - A PV:

```
$ oc new-app jenkins-persistent
```

- Or an **emptyDir** type volume where configuration does not persist across pod restarts:

```
$ oc new-app jenkins-ephemeral
```

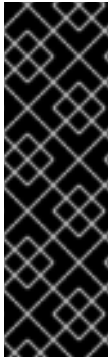
With both templates, you can run **oc describe** on them to see all the parameters available for overriding.

For example:

```
$ oc describe jenkins-ephemeral
```

5.1.9. Using the Jenkins Kubernetes plugin

In the following example, the **openshift-jee-sample BuildConfig** object causes a Jenkins Maven agent pod to be dynamically provisioned. The pod clones some Java source code, builds a WAR file, and causes a second **BuildConfig, openshift-jee-sample-docker** to run. The second **BuildConfig** layers the new WAR file into a container image.



IMPORTANT

OpenShift Container Platform 4.11 removed the OpenShift Jenkins Maven and NodeJS Agent images from its payload. Red Hat no longer produces these images, and they are not available from the **ocp-tools-4** repository at **registry.redhat.io**. Red Hat maintains the 4.10 and earlier versions of these images for any significant bug fixes or security CVEs, following the [OpenShift Container Platform lifecycle policy](#).

For more information, see the "Important changes to OpenShift Jenkins images" link in the following "Additional resources" section.

Sample BuildConfig that uses the Jenkins Kubernetes plugin

```
kind: List
apiVersion: v1
items:
- kind: ImageStream
  apiVersion: image.openshift.io/v1
  metadata:
    name: openshift-jee-sample
- kind: BuildConfig
  apiVersion: build.openshift.io/v1
  metadata:
    name: openshift-jee-sample-docker
  spec:
    strategy:
      type: Docker
    source:
      type: Docker
      dockerfile: |-
        FROM openshift/wildfly-101-centos7:latest
        COPY ROOT.war /wildfly/standalone/deployments/ROOT.war
        CMD $STI_SCRIPTS_PATH/run
    binary:
```

```

    asFile: ROOT.war
  output:
    to:
      kind: ImageStreamTag
      name: openshift-jee-sample:latest
- kind: BuildConfig
  apiVersion: build.openshift.io/v1
  metadata:
    name: openshift-jee-sample
  spec:
    strategy:
      type: JenkinsPipeline
      jenkinsPipelineStrategy:
        jenkinsfile: |-
          node("maven") {
            sh "git clone https://github.com/openshift/openshift-jee-sample.git ."
            sh "mvn -B -Popenshift package"
            sh "oc start-build -F openshift-jee-sample-docker --from-file=target/ROOT.war"
          }
    triggers:
      - type: ConfigChange

```

It is also possible to override the specification of the dynamically created Jenkins agent pod. The following is a modification to the preceding example, which overrides the container memory and specifies an environment variable.

Sample BuildConfig that uses the Jenkins Kubernetes plugin, specifying memory limit and environment variable

```

kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: openshift-jee-sample
spec:
  strategy:
    type: JenkinsPipeline
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        podTemplate(label: "mypod", ❶
          cloud: "openshift", ❷
          inheritFrom: "maven", ❸
          containers: [
            containerTemplate(name: "jnlp", ❹
              image: "openshift/jenkins-agent-maven-35-centos7:v3.10", ❺
              resourceRequestMemory: "512Mi", ❻
              resourceLimitMemory: "512Mi", ❼
              envVars: [
                envVar(key: "CONTAINER_HEAP_PERCENT", value: "0.25") ❽
              ]
            )
          ] {
            node("mypod") { ❾
              sh "git clone https://github.com/openshift/openshift-jee-sample.git ."
              sh "mvn -B -Popenshift package"
              sh "oc start-build -F openshift-jee-sample-docker --from-file=target/ROOT.war"
            }
          }

```

```

    }
  }
  triggers:
  - type: ConfigChange

```

- 1 A new pod template called **mypod** is defined dynamically. The new pod template name is referenced in the node stanza.
- 2 The **cloud** value must be set to **openshift**.
- 3 The new pod template can inherit its configuration from an existing pod template. In this case, inherited from the Maven pod template that is pre-defined by OpenShift Container Platform.
- 4 This example overrides values in the pre-existing container, and must be specified by name. All Jenkins agent images shipped with OpenShift Container Platform use the Container name **jnlp**.
- 5 Specify the container image name again. This is a known issue.
- 6 A memory request of **512 Mi** is specified.
- 7 A memory limit of **512 Mi** is specified.
- 8 An environment variable **CONTAINER_HEAP_PERCENT**, with value **0.25**, is specified.
- 9 The node stanza references the name of the defined pod template.

By default, the pod is deleted when the build completes. This behavior can be modified with the plugin or within a pipeline Jenkinsfile.

Upstream Jenkins has more recently introduced a YAML declarative format for defining a **podTemplate** pipeline DSL in-line with your pipelines. An example of this format, using the sample **java-builder** pod template that is defined in the OpenShift Container Platform Jenkins image:

```

def nodeLabel = 'java-buidler'

pipeline {
  agent {
    kubernetes {
      cloud 'openshift'
      label nodeLabel
      yml """
apiVersion: v1
kind: Pod
metadata:
  labels:
    worker: ${nodeLabel}
spec:
  containers:
  - name: jnlp
    image: image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-rhel8:latest
    args: ["$(JENKINS_SECRET)", "$(JENKINS_NAME)"]
  - name: java
    image: image-registry.openshift-image-registry.svc:5000/openshift/java:latest
    command:
    - cat
  tty: true

```

```

    }
  }

  options {
    timeout(time: 20, unit: 'MINUTES')
  }

  stages {
    stage('Build App') {
      steps {
        container("java") {
          sh "mvn --version"
        }
      }
    }
  }
}

```

Additional resources

- [Important changes to OpenShift Jenkins images](#)

5.1.10. Jenkins memory requirements

When deployed by the provided Jenkins Ephemeral or Jenkins Persistent templates, the default memory limit is **1 Gi**.

By default, all other process that run in the Jenkins container cannot use more than a total of **512 MiB** of memory. If they require more memory, the container halts. It is therefore highly recommended that pipelines run external commands in an agent container wherever possible.

And if **Project** quotas allow for it, see recommendations from the Jenkins documentation on what a Jenkins master should have from a memory perspective. Those recommendations proscribe to allocate even more memory for the Jenkins master.

It is recommended to specify memory request and limit values on agent containers created by the Jenkins Kubernetes plugin. Admin users can set default values on a per-agent image basis through the Jenkins configuration. The memory request and limit parameters can also be overridden on a per-container basis.

You can increase the amount of memory available to Jenkins by overriding the **MEMORY_LIMIT** parameter when instantiating the Jenkins Ephemeral or Jenkins Persistent template.

5.1.11. Additional resources

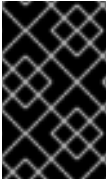
- See [Base image options](#) for more information about the [Red Hat Universal Base Images](#) (UBI).
- [Important changes to OpenShift Jenkins images](#)

5.2. JENKINS AGENT

OpenShift Container Platform provides a base image for use as a Jenkins agent.

The Base image for Jenkins agents does the following:

- Pulls in both the required tools, headless Java, the Jenkins JNLP client, and the useful ones, including **git**, **tar**, **zip**, and **nss**, among others.
- Establishes the JNLP agent as the entry point.
- Includes the **oc** client tool for invoking command line operations from within Jenkins jobs.
- Provides Dockerfiles for both Red Hat Enterprise Linux (RHEL) and **localdev** images.



IMPORTANT

Use a version of the agent image that is appropriate for your OpenShift Container Platform release version. Embedding an **oc** client version that is not compatible with the OpenShift Container Platform version can cause unexpected behavior.

The OpenShift Container Platform Jenkins image also defines the following sample **java-builder** pod template to illustrate how you can use the agent image with the Jenkins Kubernetes plugin.

The **java-builder** pod template employs two containers: * A **jnlp** container that uses the OpenShift Container Platform Base agent image and handles the JNLP contract for starting and stopping Jenkins agents. * A **java** container that uses the **java** OpenShift Container Platform Sample ImageStream, which contains the various Java binaries, including the Maven binary **mvn**, for building code.

5.2.1. Jenkins agent images

The OpenShift Container Platform Jenkins agent images are available on [Quay.io](https://quay.io) or registry.redhat.io.

Jenkins images are available through the Red Hat Registry:

```
$ docker pull registry.redhat.io/ocp-tools-4/jenkins-rhel8:<image_tag>
```

```
$ docker pull registry.redhat.io/ocp-tools-4/jenkins-agent-base-rhel8:<image_tag>
```

To use these images, you can either access them directly from [Quay.io](https://quay.io) or registry.redhat.io or push them into your OpenShift Container Platform container image registry.

5.2.2. Jenkins agent environment variables

Each Jenkins agent container can be configured with the following environment variables.

Variable	Definition	Example values and settings
----------	------------	-----------------------------

Variable	Definition	Example values and settings
JAVA_MAX_HEAP_PARAM, CONTAINER_HEAP_PERCENT, JENKINS_MAX_HEAP_UPPER_BOUND_MB	<p>These values control the maximum heap size of the Jenkins JVM. If JAVA_MAX_HEAP_PARAM is set, its value takes precedence. Otherwise, the maximum heap size is dynamically calculated as CONTAINER_HEAP_PERCENT of the container memory limit, optionally capped at JENKINS_MAX_HEAP_UPPER_BOUND_MB MiB.</p> <p>By default, the maximum heap size of the Jenkins JVM is set to 50% of the container memory limit with no cap.</p>	<p>JAVA_MAX_HEAP_PARAM example setting: -Xmx512m</p> <p>CONTAINER_HEAP_PERCENT default: 0.5, or 50%</p> <p>JENKINS_MAX_HEAP_UPPER_BOUND_MB example setting: 512 MiB</p>
JAVA_INITIAL_HEAP_PARAM, CONTAINER_INITIAL_PERCENT	<p>These values control the initial heap size of the Jenkins JVM. If JAVA_INITIAL_HEAP_PARAM is set, its value takes precedence. Otherwise, the initial heap size is dynamically calculated as CONTAINER_INITIAL_PERCENT of the dynamically calculated maximum heap size.</p> <p>By default, the JVM sets the initial heap size.</p>	<p>JAVA_INITIAL_HEAP_PARAM example setting: -Xms32m</p> <p>CONTAINER_INITIAL_PERCENT example setting: 0.1, or 10%</p>
CONTAINER_CORE_LIMIT	If set, specifies an integer number of cores used for sizing numbers of internal JVM threads.	Example setting: 2
JAVA_TOOL_OPTIONS	Specifies options to apply to all JVMs running in this container. It is not recommended to override this value.	<p>Default: -</p> <p>XX:+UnlockExperimentalVMOptions -</p> <p>XX:+UseCGroupMemoryLimitForHeap -</p> <p>Dsun.zip.disableMemoryMapping=true</p>
JAVA_GC_OPTS	Specifies Jenkins JVM garbage collection parameters. It is not recommended to override this value.	<p>Default: -XX:+UseParallelGC -</p> <p>XX:MinHeapFreeRatio=5 -</p> <p>XX:MaxHeapFreeRatio=10 -</p> <p>XX:GCTimeRatio=4 -</p> <p>XX:AdaptiveSizePolicyWeight=90</p>

Variable	Definition	Example values and settings
JENKINS_JAVA_OVERRIDES	Specifies additional options for the Jenkins JVM. These options are appended to all other options, including the Java options above, and can be used to override any of them, if necessary. Separate each additional option with a space and if any option contains space characters, escape them with a backslash.	Example settings: -Dfoo -Dbar; -Dfoo=first\ value -Dbar=second\ value
USE_JAVA_VERSION	Specifies the version of Java version to use to run the agent in its container. The container base image has two versions of java installed: java-11 and java-1.8.0 . If you extend the container base image, you can specify any alternative version of java using its associated suffix.	The default value is java-11 . Example setting: java-1.8.0

5.2.3. Jenkins agent memory requirements

A JVM is used in all Jenkins agents to host the Jenkins JNLP agent as well as to run any Java applications such as **javac**, Maven, or Gradle.

By default, the Jenkins JNLP agent JVM uses 50% of the container memory limit for its heap. This value can be modified by the **CONTAINER_HEAP_PERCENT** environment variable. It can also be capped at an upper limit or overridden entirely.

By default, any other processes run in the Jenkins agent container, such as shell scripts or **oc** commands run from pipelines, cannot use more than the remaining 50% memory limit without provoking an OOM kill.

By default, each further JVM process that runs in a Jenkins agent container uses up to 25% of the container memory limit for its heap. It might be necessary to tune this limit for many build workloads.

5.2.4. Jenkins agent Gradle builds

Hosting Gradle builds in the Jenkins agent on OpenShift Container Platform presents additional complications because in addition to the Jenkins JNLP agent and Gradle JVMs, Gradle spawns a third JVM to run tests if they are specified.

The following settings are suggested as a starting point for running Gradle builds in a memory constrained Jenkins agent on OpenShift Container Platform. You can modify these settings as required.

- Ensure the long-lived Gradle daemon is disabled by adding **org.gradle.daemon=false** to the **gradle.properties** file.
- Disable parallel build execution by ensuring **org.gradle.parallel=true** is not set in the **gradle.properties** file and that **--parallel** is not set as a command line argument.

- To prevent Java compilations running out-of-process, set **java { options.fork = false }** in the **build.gradle** file.
- Disable multiple additional test processes by ensuring **test { maxParallelForks = 1 }** is set in the **build.gradle** file.
- Override the Gradle JVM memory parameters by the **GRADLE_OPTS**, **JAVA_OPTS** or **JAVA_TOOL_OPTIONS** environment variables.
- Set the maximum heap size and JVM arguments for any Gradle test JVM by defining the **maxHeapSize** and **jvmArgs** settings in **build.gradle**, or through the **-Dorg.gradle.jvmargs** command line argument.

5.2.5. Jenkins agent pod retention

Jenkins agent pods, are deleted by default after the build completes or is stopped. This behavior can be changed by the Kubernetes plugin pod retention setting. Pod retention can be set for all Jenkins builds, with overrides for each pod template. The following behaviors are supported:

- **Always** keeps the build pod regardless of build result.
- **Default** uses the plugin value, which is the pod template only.
- **Never** always deletes the pod.
- **On Failure** keeps the pod if it fails during the build.

You can override pod retention in the pipeline Jenkinsfile:

```
podTemplate(label: "mypod",
  cloud: "openshift",
  inheritFrom: "maven",
  podRetention: onFailure(), 1
  containers: [
    ...
  ]) {
  node("mypod") {
    ...
  }
}
```

- 1 Allowed values for **podRetention** are **never()**, **onFailure()**, **always()**, and **default()**.



WARNING

Pods that are kept might continue to run and count against resource quotas.

5.3. MIGRATING FROM JENKINS TO OPENSIFT PIPELINES OR TEKTON

You can migrate your CI/CD workflows from Jenkins to [Red Hat OpenShift Pipelines](#), a cloud-native CI/CD experience based on the Tekton project.

5.3.1. Comparison of Jenkins and OpenShift Pipelines concepts

You can review and compare the following equivalent terms used in Jenkins and OpenShift Pipelines.

5.3.1.1. Jenkins terminology

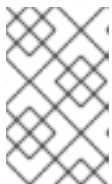
Jenkins offers declarative and scripted pipelines that are extensible using shared libraries and plugins. Some basic terms in Jenkins are as follows:

- **Pipeline:** Automates the entire process of building, testing, and deploying applications by using [Groovy](#) syntax.
- **Node:** A machine capable of either orchestrating or executing a scripted pipeline.
- **Stage:** A conceptually distinct subset of tasks performed in a pipeline. Plugins or user interfaces often use this block to display the status or progress of tasks.
- **Step:** A single task that specifies the exact action to be taken, either by using a command or a script.

5.3.1.2. OpenShift Pipelines terminology

OpenShift Pipelines uses [YAML](#) syntax for declarative pipelines and consists of tasks. Some basic terms in OpenShift Pipelines are as follows:

- **Pipeline:** A set of tasks in a series, in parallel, or both.
- **Task:** A sequence of steps as commands, binaries, or scripts.
- **PipelineRun:** Execution of a pipeline with one or more tasks.
- **TaskRun:** Execution of a task with one or more steps.



NOTE

You can initiate a PipelineRun or a TaskRun with a set of inputs such as parameters and workspaces, and the execution results in a set of outputs and artifacts.

- **Workspace:** In OpenShift Pipelines, workspaces are conceptual blocks that serve the following purposes:
 - Storage of inputs, outputs, and build artifacts.
 - Common space to share data among tasks.
 - Mount points for credentials held in secrets, configurations held in config maps, and common tools shared by an organization.

**NOTE**

In Jenkins, there is no direct equivalent of OpenShift Pipelines workspaces. You can think of the control node as a workspace, as it stores the cloned code repository, build history, and artifacts. When a job is assigned to a different node, the cloned code and the generated artifacts are stored in that node, but the control node maintains the build history.

5.3.1.3. Mapping of concepts

The building blocks of Jenkins and OpenShift Pipelines are not equivalent, and a specific comparison does not provide a technically accurate mapping. The following terms and concepts in Jenkins and OpenShift Pipelines correlate in general:

Table 5.1. Jenkins and OpenShift Pipelines - basic comparison

Jenkins	OpenShift Pipelines
Pipeline	Pipeline and PipelineRun
Stage	Task
Step	A step in a task

5.3.2. Migrating a sample pipeline from Jenkins to OpenShift Pipelines

You can use the following equivalent examples to help migrate your build, test, and deploy pipelines from Jenkins to OpenShift Pipelines.

5.3.2.1. Jenkins pipeline

Consider a Jenkins pipeline written in Groovy for building, testing, and deploying:

```

pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        sh 'make'
      }
    }
    stage('Test'){
      steps {
        sh 'make check'
        junit 'reports/**/*.xml'
      }
    }
    stage('Deploy') {
      steps {
        sh 'make publish'
      }
    }
  }
}

```

```

    }
  }
}

```

5.3.2.2. OpenShift Pipelines pipeline

To create a pipeline in OpenShift Pipelines that is equivalent to the preceding Jenkins pipeline, you create the following three tasks:

Example build task YAML definition file

```

apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myproject-build
spec:
  workspaces:
    - name: source
  steps:
    - image: my-ci-image
      command: ["make"]
      workingDir: $(workspaces.source.path)

```

Example test task YAML definition file

```

apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myproject-test
spec:
  workspaces:
    - name: source
  steps:
    - image: my-ci-image
      command: ["make check"]
      workingDir: $(workspaces.source.path)
    - image: junit-report-image
      script: |
        #!/usr/bin/env bash
        junit-report reports/**/*.xml
      workingDir: $(workspaces.source.path)

```

Example deploy task YAML definition file

```

apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myprojectd-deploy
spec:
  workspaces:
    - name: source
  steps:

```

```
- image: my-deploy-image
  command: ["make deploy"]
  workingDir: $(workspaces.source.path)
```

You can combine the three tasks sequentially to form a pipeline in OpenShift Pipelines:

Example: OpenShift Pipelines pipeline for building, testing, and deployment

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: myproject-pipeline
spec:
  workspaces:
    - name: shared-dir
  tasks:
    - name: build
      taskRef:
        name: myproject-build
      workspaces:
        - name: source
          workspace: shared-dir
    - name: test
      taskRef:
        name: myproject-test
      workspaces:
        - name: source
          workspace: shared-dir
    - name: deploy
      taskRef:
        name: myproject-deploy
      workspaces:
        - name: source
          workspace: shared-dir
```

5.3.3. Migrating from Jenkins plugins to Tekton Hub tasks

You can extend the capability of Jenkins by using [plugins](#). To achieve similar extensibility in OpenShift Pipelines, use any of the tasks available from [Tekton Hub](#).

For example, consider the [git-clone](#) task in Tekton Hub, which corresponds to the [git plugin](#) for Jenkins.

Example: git-clone task from Tekton Hub

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: demo-pipeline
spec:
  params:
    - name: repo_url
    - name: revision
  workspaces:
    - name: source
  tasks:
```



```

- name: fetch-from-git
  taskRef:
    name: git-clone
  params:
    - name: url
      value: $(params.repo_url)
    - name: revision
      value: $(params.revision)
  workspaces:
    - name: output
      workspace: source

```

5.3.4. Extending OpenShift Pipelines capabilities using custom tasks and scripts

In OpenShift Pipelines, if you do not find the right task in Tekton Hub, or need greater control over tasks, you can create custom tasks and scripts to extend the capabilities of OpenShift Pipelines.

Example: A custom task for running the `maven test` command

```

apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: maven-test
spec:
  workspaces:
    - name: source
  steps:
    - image: my-maven-image
      command: ["mvn test"]
      workingDir: $(workspaces.source.path)

```

Example: Run a custom shell script by providing its path

```

...
steps:
  image: ubuntu
  script: |
    #!/usr/bin/env bash
    /workspace/my-script.sh
...

```

Example: Run a custom Python script by writing it in the YAML file

```

...
steps:
  image: python
  script: |
    #!/usr/bin/env python3
    print("hello from python!")
...

```

5.3.5. Comparison of Jenkins and OpenShift Pipelines execution models

Jenkins and OpenShift Pipelines offer similar functions but are different in architecture and execution.

Table 5.2. Comparison of execution models in Jenkins and OpenShift Pipelines

Jenkins	OpenShift Pipelines
Jenkins has a controller node. Jenkins runs pipelines and steps centrally, or orchestrates jobs running in other nodes.	OpenShift Pipelines is serverless and distributed, and there is no central dependency for execution.
Containers are launched by the Jenkins controller node through the pipeline.	OpenShift Pipelines adopts a 'container-first' approach, where every step runs as a container in a pod (equivalent to nodes in Jenkins).
Extensibility is achieved by using plugins.	Extensibility is achieved by using tasks in Tekton Hub or by creating custom tasks and scripts.

5.3.6. Examples of common use cases

Both Jenkins and OpenShift Pipelines offer capabilities for common CI/CD use cases, such as:

- Compiling, building, and deploying images using Apache Maven
- Extending the core capabilities by using plugins
- Reusing shareable libraries and custom scripts

5.3.6.1. Running a Maven pipeline in Jenkins and OpenShift Pipelines

You can use Maven in both Jenkins and OpenShift Pipelines workflows for compiling, building, and deploying images. To map your existing Jenkins workflow to OpenShift Pipelines, consider the following examples:

Example: Compile and build an image and deploy it to OpenShift using Maven in Jenkins

```
#!/usr/bin/groovy
node('maven') {
    stage 'Checkout'
    checkout scm

    stage 'Build'
    sh 'cd helloworld && mvn clean'
    sh 'cd helloworld && mvn compile'

    stage 'Run Unit Tests'
    sh 'cd helloworld && mvn test'

    stage 'Package'
    sh 'cd helloworld && mvn package'

    stage 'Archive artifact'
    sh 'mkdir -p artifacts/deployments && cp helloworld/target/*.war artifacts/deployments'
    archive 'helloworld/target/*.war'
```

```

stage 'Create Image'
sh 'oc login https://kubernetes.default -u admin -p admin --insecure-skip-tls-verify=true'
sh 'oc new-project helloworldproject'
sh 'oc project helloworldproject'
sh 'oc process -f helloworld/jboss-eap70-binary-build.json | oc create -f -'
sh 'oc start-build eap-helloworld-app --from-dir=artifacts/'

stage 'Deploy'
sh 'oc new-app helloworld/jboss-eap70-deploy.json' }

```

Example: Compile and build an image and deploy it to OpenShift using Maven in OpenShift Pipelines.

```

apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: maven-pipeline
spec:
  workspaces:
    - name: shared-workspace
    - name: maven-settings
    - name: kubeconfig-dir
      optional: true
  params:
    - name: repo-url
    - name: revision
    - name: context-path
  tasks:
    - name: fetch-repo
      taskRef:
        name: git-clone
      workspaces:
        - name: output
          workspace: shared-workspace
      params:
        - name: url
          value: "${params.repo-url}"
        - name: subdirectory
          value: ""
        - name: deleteExisting
          value: "true"
        - name: revision
          value: ${params.revision}
    - name: mvn-build
      taskRef:
        name: maven
      runAfter:
        - fetch-repo
      workspaces:
        - name: source
          workspace: shared-workspace
        - name: maven-settings
          workspace: maven-settings
      params:

```

```

- name: CONTEXT_DIR
  value: "${params.context-path}"
- name: GOALS
  value: ["-DskipTests", "clean", "compile"]
- name: mvn-tests
  taskRef:
    name: maven
  runAfter:
    - mvn-build
  workspaces:
    - name: source
      workspace: shared-workspace
    - name: maven-settings
      workspace: maven-settings
  params:
    - name: CONTEXT_DIR
      value: "${params.context-path}"
    - name: GOALS
      value: ["test"]
- name: mvn-package
  taskRef:
    name: maven
  runAfter:
    - mvn-tests
  workspaces:
    - name: source
      workspace: shared-workspace
    - name: maven-settings
      workspace: maven-settings
  params:
    - name: CONTEXT_DIR
      value: "${params.context-path}"
    - name: GOALS
      value: ["package"]
- name: create-image-and-deploy
  taskRef:
    name: openshift-client
  runAfter:
    - mvn-package
  workspaces:
    - name: manifest-dir
      workspace: shared-workspace
    - name: kubeconfig-dir
      workspace: kubeconfig-dir
  params:
    - name: SCRIPT
      value: |
        cd "${params.context-path}"
        mkdir -p ./artifacts/deployments && cp ./target/*.war ./artifacts/deployments
        oc new-project helloworldproject
        oc project helloworldproject
        oc process -f jboss-eap70-binary-build.json | oc create -f -
        oc start-build eap-helloworld-app --from-dir=artifacts/
        oc new-app jboss-eap70-deploy.json

```

5.3.6.2. Extending the core capabilities of Jenkins and OpenShift Pipelines by using plugins

Jenkins has the advantage of a large ecosystem of numerous plugins developed over the years by its extensive user base. You can search and browse the plugins in the [Jenkins Plugin Index](#).

OpenShift Pipelines also has many tasks developed and contributed by the community and enterprise users. A publicly available catalog of reusable OpenShift Pipelines tasks are available in the [Tekton Hub](#).

In addition, OpenShift Pipelines incorporates many of the plugins of the Jenkins ecosystem within its core capabilities. For example, authorization is a critical function in both Jenkins and OpenShift Pipelines. While Jenkins ensures authorization using the [Role-based Authorization Strategy](#) plugin, OpenShift Pipelines uses OpenShift's built-in Role-based Access Control system.

5.3.6.3. Sharing reusable code in Jenkins and OpenShift Pipelines

Jenkins [shared libraries](#) provide reusable code for parts of Jenkins pipelines. The libraries are shared between [Jenkinsfiles](#) to create highly modular pipelines without code repetition.

Although there is no direct equivalent of Jenkins shared libraries in OpenShift Pipelines, you can achieve similar workflows by using tasks from the [Tekton Hub](#) in combination with custom tasks and scripts.

5.3.7. Additional resources

- [Understanding OpenShift Pipelines](#)
- [Role-based Access Control](#)

5.4. IMPORTANT CHANGES TO OPENSIFT JENKINS IMAGES

OpenShift Container Platform 4.11 moves the OpenShift Jenkins and OpenShift Agent Base images to the **ocp-tools-4** repository at **registry.redhat.io**. It also removes the OpenShift Jenkins Maven and NodeJS Agent images from its payload:

- OpenShift Container Platform 4.11 moves the OpenShift Jenkins and OpenShift Agent Base images to the **ocp-tools-4** repository at **registry.redhat.io** so that Red Hat can produce and update the images outside the OpenShift Container Platform lifecycle. Previously, these images were in the OpenShift Container Platform install payload and the **openshift4** repository at **registry.redhat.io**.
- OpenShift Container Platform 4.10 deprecated the OpenShift Jenkins Maven and NodeJS Agent images. OpenShift Container Platform 4.11 removes these images from its payload. Red Hat no longer produces these images, and they are not available from the **ocp-tools-4** repository at **registry.redhat.io**. Red Hat maintains the 4.10 and earlier versions of these images for any significant bug fixes or security CVEs, following the [OpenShift Container Platform lifecycle policy](#).

These changes support the OpenShift Container Platform 4.10 recommendation to use [multiple container Pod Templates with the Jenkins Kubernetes Plugin](#).

5.4.1. Relocation of OpenShift Jenkins images

OpenShift Container Platform 4.11 makes significant changes to the location and availability of specific OpenShift Jenkins images. Additionally, you can configure when and how to update these images.

What stays the same with the OpenShift Jenkins images?

- The Cluster Samples Operator manages the **ImageStream** and **Template** objects for operating the OpenShift Jenkins images.
- By default, the Jenkins **DeploymentConfig** object from the Jenkins pod template triggers a redeployment when the Jenkins image changes. By default, this image is referenced by the **jenkins:2** image stream tag of Jenkins image stream in the **openshift** namespace in the **ImageStream** YAML file in the Samples Operator payload.
- If you upgrade from OpenShift Container Platform 4.10 and earlier to 4.11, the deprecated **maven** and **nodejs** pod templates are still in the default image configuration.
- If you upgrade from OpenShift Container Platform 4.10 and earlier to 4.11, the **jenkins-agent-maven** and **jenkins-agent-nodejs** image streams still exist in your cluster. To maintain these image streams, see the following section, "What happens with the **jenkins-agent-maven** and **jenkins-agent-nodejs** image streams in the **openshift** namespace?"

What changes in the support matrix of the OpenShift Jenkins image?

Each new image in the **ocp-tools-4** repository in the **registry.redhat.io** registry supports multiple versions of OpenShift Container Platform. When Red Hat updates one of these new images, it is simultaneously available for all versions. This availability is ideal when Red Hat updates an image in response to a security advisory. Initially, this change applies to OpenShift Container Platform 4.11 and later. It is planned that this change will eventually apply to OpenShift Container Platform 4.9 and later.

Previously, each Jenkins image supported only one version of OpenShift Container Platform and Red Hat might update those images sequentially over time.

What additions are there with the OpenShift Jenkins and Jenkins Agent Base ImageStream and ImageStreamTag objects?

By moving from an in-payload image stream to an image stream that references non-payload images, OpenShift Container Platform can define additional image stream tags. Red Hat has created a series of new image stream tags to go along with the existing "**value**": "**jenkins:2**" and "**value**": "**image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-rhel8:latest**" image stream tags present in OpenShift Container Platform 4.10 and earlier. These new image stream tags address some requests to improve how the Jenkins-related image streams are maintained.

About the new image stream tags:

ocp-upgrade-redeploy

To update your Jenkins image when you upgrade OpenShift Container Platform, use this image stream tag in your Jenkins deployment configuration. This image stream tag corresponds to the existing **2** image stream tag of the **jenkins** image stream and the **latest** image stream tag of the **jenkins-agent-base-rhel8** image stream. It employs an image tag specific to only one SHA or image digest. When the **ocp-tools-4** image changes, such as for Jenkins security advisories, Red Hat Engineering updates the Cluster Samples Operator payload.

user-maintained-upgrade-redeploy

To manually redeploy Jenkins after you upgrade OpenShift Container Platform, use this image stream tag in your Jenkins deployment configuration. This image stream tag uses the least specific image version indicator available. When you redeploy Jenkins, run the following command: **\$ oc import-image jenkins:user-maintained-upgrade-redeploy -n openshift**. When you issue this command, the OpenShift Container Platform **ImageStream** controller accesses the **registry.redhat.io** image registry and stores any updated images in the OpenShift Container Platform internal image registry's slot for that Jenkins **ImageStreamTag** object. Otherwise, if you do not run this command, your Jenkins deployment configuration does not trigger a redeployment.

scheduled-upgrade-redeploy

To automatically redeploy the latest version of the Jenkins image when it is released, use this image stream tag in your Jenkins deployment configuration. This image stream tag uses the periodic importing of image stream tags feature of the OpenShift Container Platform image stream controller, which checks for changes in the backing image. If the image changes, for example, due to a recent Jenkins security advisory, OpenShift Container Platform triggers a redeployment of your Jenkins deployment configuration. See "Configuring periodic importing of image stream tags" in the following "Additional resources."

What happens with the **jenkins-agent-maven** and **jenkins-agent-nodejs** image streams in the openshift namespace?

The OpenShift Jenkins Maven and NodeJS Agent images for OpenShift Container Platform were deprecated in 4.10, and are removed from the OpenShift Container Platform install payload in 4.11. They do not have alternatives defined in the **ocp-tools-4** repository. However, you can work around this by using the sidecar pattern described in the "Jenkins agent" topic mentioned in the following "Additional resources" section.

However, the Cluster Samples Operator does not delete the **jenkins-agent-maven** and **jenkins-agent-nodejs** image streams created by prior releases, which point to the tags of the respective OpenShift Container Platform payload images on **registry.redhat.io**. Therefore, you can pull updates to these images by running the following commands:

```
$ oc import-image jenkins-agent-nodejs -n openshift
```

```
$ oc import-image jenkins-agent-maven -n openshift
```

5.4.2. Customizing the Jenkins image stream tag

To override the default upgrade behavior and control how the Jenkins image is upgraded, you set the image stream tag value that your Jenkins deployment configurations use.

The default upgrade behavior is the behavior that existed when the Jenkins image was part of the install payload. The image stream tag names, **2** and **ocp-upgrade-redeploy**, in the **jenkins-rhel.json** image stream file use SHA-specific image references. Therefore, when those tags are updated with a new SHA, the OpenShift Container Platform image change controller automatically redeploys the Jenkins deployment configuration from the associated templates, such as **jenkins-ephemeral.json** or **jenkins-persistent.json**.

For new deployments, to override that default value, you change the value of the **JENKINS_IMAGE_STREAM_TAG** in the **jenkins-ephemeral.json** Jenkins template. For example, replace the **2** in "**value**": "**jenkins:2**" with one of the following image stream tags:

- **ocp-upgrade-redeploy**, the default value, updates your Jenkins image when you upgrade OpenShift Container Platform.
- **user-maintained-upgrade-redeploy** requires you to manually redeploy Jenkins by running **\$ oc import-image jenkins:user-maintained-upgrade-redeploy -n openshift** after upgrading OpenShift Container Platform.
- **scheduled-upgrade-redeploy** periodically checks the given **<image>:<tag>** combination for changes and upgrades the image when it changes. The image change controller pulls the changed image and redeploys the Jenkins deployment configuration provisioned by the templates. For more information about this scheduled import policy, see the "Adding tags to image streams" in the following "Additional resources."

**NOTE**

To override the current upgrade value for existing deployments, change the values of the environment variables that correspond to those template parameters.

Prerequisites

- You are running OpenShift Jenkins on OpenShift Container Platform 4.11.
- You know the namespace where OpenShift Jenkins is deployed.

Procedure

- Set the image stream tag value, replacing **<namespace>** with namespace where OpenShift Jenkins is deployed and **<image_stream_tag>** with an image stream tag:

Example

```
$ oc patch dc jenkins -p '{"spec":{"triggers":[{"type":"ImageChange","imageChangeParams":{"automatic":true,"containerNames":["jenkins"],"from":{"kind":"ImageStreamTag","namespace":"<namespace>","name":"jenkins:<image_stream_tag>"}}}]}}'
```

TIP

Alternatively, to edit the Jenkins deployment configuration YAML, enter **\$ oc edit dc/jenkins -n <namespace>** and update the **value: 'jenkins:<image_stream_tag>'** line.

5.4.3. Additional resources

- [Adding tags to image streams](#)
- [Configuring periodic importing of image stream tags](#)
- [Jenkins agent](#)
- [Certified **jenkins** images](#)
- [Certified **jenkins-agent-base** images](#)
- [Certified **jenkins-agent-maven** images](#)
- [Certified **jenkins-agent-nodejs** images](#)