OpenShift Container Platform 4.14

Migration Toolkit for Containers

Migrating to OpenShift Container Platform 4

Last Updated: 2026-01-20

OpenShift Container Platform 4.14 Migration Toolkit for Containers

Migrating to OpenShift Container Platform 4

Legal Notice

Copyright © Red Hat.

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, 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 Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

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

All other trademarks are the property of their respective owners.

Abstract

This document provides instructions for migrating your stateful application workloads between OpenShift Container Platform 4 clusters.

Table of Contents

CHAPTER 1. ABOUT THE MIGRATION TOOLKIT FOR CONTAINERS 9
1.1. MTC 1.8 SUPPORT 9
1.1.1. Support for Migration Toolkit for Containers (MTC) 9
1.2. TERMINOLOGY 10
1.3. MTC WORKFLOW 11
1.4. ABOUT DATA COPY METHODS 14
1.4.1. File system copy method 14
1.4.2. Snapshot copy method 14
1.5. DIRECT VOLUME MIGRATION AND DIRECT IMAGE MIGRATION 15

CHAPTER 2. MTC RELEASE NOTES 16
2.1. MIGRATION TOOLKIT FOR CONTAINERS 1.8 RELEASE NOTES 16
2.1.1. Migration Toolkit for Containers 1.8.11 release notes 16
2.1.2. Migration Toolkit for Containers 1.8.10 release notes 16
2.1.2.1. Resolved issues 16
2.1.3. Migration Toolkit for Containers 1.8.9 release notes 16
2.1.3.1. Resolved issues 16
2.1.4. Migration Toolkit for Containers 1.8.8 release notes 17
2.1.4.1. Resolved issues 17
2.1.5. Migration Toolkit for Containers 1.8.7 release notes 17
2.1.5.1. Resolved issues 17
2.1.6. Migration Toolkit for Containers 1.8.6 release notes 18
2.1.6.1. Technical changes 18
2.1.6.2. Resolved issues 18
2.1.6.3. Known issues 19
2.1.7. Migration Toolkit for Containers 1.8.5 release notes 19
2.1.7.1. Technical changes 19
2.1.7.2. Resolved issues 19
2.1.7.3. Known issues 19
2.1.8. Migration Toolkit for Containers 1.8.4 release notes 20
2.1.8.1. Technical changes 20
2.1.8.2. Resolved issues 20
2.1.8.3. Known issues 21
2.1.9. Migration Toolkit for Containers 1.8.3 release notes 21
2.1.9.1. Technical changes 21
2.1.9.2. Resolved issues 22
2.1.9.3. Known issues 22
2.1.10. Migration Toolkit for Containers 1.8.2 release notes 23
2.1.10.1. Resolved issues 23
2.1.10.2. Known issues 23
2.1.11. Migration Toolkit for Containers 1.8.1 release notes 23
2.1.11.1. Resolved issues 23
2.1.11.2. Known issues 24
2.1.12. Migration Toolkit for Containers 1.8.0 release notes 24
2.1.12.1. Resolved issues 24
2.1.12.2. Known issues 24
2.1.12.3. Technical changes 25
2.2. MIGRATION TOOLKIT FOR CONTAINERS 1.7 RELEASE NOTES 26
2.2.1. Migration Toolkit for Containers 1.7.19 release notes 26
2.2.1.1. Resolved issues 26
2.2.1.2. Known issues 27

2.2.2. Migration Toolkit for Containers 1.7.18 release notes 27
2.2.2.1. Technical changes 27
2.2.3. Migration Toolkit for Containers 1.7.17 release notes 27
2.2.4. Migration Toolkit for Containers 1.7.16 release notes 27
2.2.4.1. Resolved issues 27
2.2.4.2. Known issues 29
2.2.5. Migration Toolkit for Containers 1.7.15 release notes 29
2.2.5.1. Resolved issues 29
2.2.5.2. Known issues 29
2.2.6. Migration Toolkit for Containers 1.7.14 release notes 30
2.2.6.1. Resolved issues 30
2.2.6.2. Known issues 31
2.2.7. Migration Toolkit for Containers 1.7.13 release notes 31
2.2.7.1. Resolved issues 31
2.2.7.2. Known issues 31
2.2.8. Migration Toolkit for Containers 1.7.12 release notes 31
2.2.8.1. Resolved issues 31
2.2.8.2. Known issues 31
2.2.9. Migration Toolkit for Containers 1.7.11 release notes 32
2.2.9.1. Resolved issues 32
2.2.9.2. Known issues 32
2.2.10. Migration Toolkit for Containers 1.7.10 release notes 32
2.2.10.1. Resolved issues 32
2.2.10.2. Known issues 32
2.2.11. Migration Toolkit for Containers 1.7.9 release notes 32
2.2.11.1. Resolved issues 32
2.2.11.2. Known issues 32
2.2.12. Migration Toolkit for Containers 1.7.8 release notes 33
2.2.12.1. Resolved issues 33
2.2.12.2. Known issues 33
2.2.13. Migration Toolkit for Containers 1.7.7 release notes 33
2.2.13.1. Resolved issues 33
2.2.13.2. Known issues 33
2.2.14. Migration Toolkit for Containers 1.7.6 release notes 33
2.2.14.1. New features 33
2.2.14.2. Resolved issues 34
2.2.14.3. Known issues 34
2.2.15. Migration Toolkit for Containers 1.7.5 release notes 34
2.2.15.1. Resolved issues 34
2.2.15.2. Known issues 34
2.2.16. Migration Toolkit for Containers 1.7.4 release notes 34
2.2.16.1. Resolved issues 34
2.2.16.2. Known issues 34
2.2.17. Migration Toolkit for Containers 1.7.3 release notes 35
2.2.17.1. Resolved issues 35
2.2.17.2. Known issues 35
2.2.18. Migration Toolkit for Containers 1.7.2 release notes 35
2.2.18.1. Resolved issues 35
2.2.18.2. Known issues 36
2.2.19. Migration Toolkit for Containers 1.7.1 release notes 36
2.2.19.1. Resolved issues 37
2.2.19.2. Known issues 37
2.2.20. Migration Toolkit for Containers 1.7.0 release notes 37

2.2.20.1. New features and enhancements 37
2.2.20.2. Known issues 38
2.3. MIGRATION TOOLKIT FOR CONTAINERS 1.6 RELEASE NOTES 38
2.3.1. Migration Toolkit for Containers 1.6 release notes 38
2.3.1.1. New features and enhancements 38
2.3.1.2. Deprecated features 39
2.3.1.3. Known issues 39
2.4. MIGRATION TOOLKIT FOR CONTAINERS 1.5 RELEASE NOTES 39
2.4.1. Migration Toolkit for Containers 1.5 release notes 39
2.4.1.1. New features and enhancements 39
2.4.1.2. Deprecated features 40
2.4.1.3. Known issues 40
2.4.1.4. Technical changes 41

CHAPTER 3. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS 42
3.1. COMPATIBILITY GUIDELINES 42
3.2. INSTALLING THE LEGACY MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSHIFT CONTAINER PLATFORM 4.2 TO 4.5 43
3.3. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSHIFT CONTAINER PLATFORM 4.14 44
3.4. PROXY CONFIGURATION 45
3.4.1. Direct volume migration 45
3.4.1.1. TCP proxy setup for DVM 45
3.4.1.2. Why use a TCP proxy instead of an HTTP/HTTPS proxy? 46
3.4.1.3. Known issue 46
3.4.2. Tuning network policies for migrations 46
3.4.2.1. NetworkPolicy configuration 46
3.4.2.1.1. Egress traffic from Rsync pods 46
3.4.2.1.2. Ingress traffic to Rsync pods 47
3.4.2.2. EgressNetworkPolicy configuration 47
3.4.2.3. Choosing alternate endpoints for data transfer 47
3.4.2.4. Configuring supplemental groups for Rsync pods 48
3.4.3. Configuring proxies 48
3.4.4. Running Rsync as either root or non-root 49
3.4.4.1. Manually overriding default non-root operation for data transfer 49
3.4.4.2. Configuring the MigrationController CR as root or non-root for all migrations 50
3.4.4.3. Configuring the MigMigration CR as root or non-root per migration 50
3.5. CONFIGURING A REPLICATION REPOSITORY 51
3.5.1. Prerequisites 51
3.5.2. Retrieving Multicloud Object Gateway credentials 51
3.5.3. Configuring Amazon Web Services 52
3.5.4. Configuring Google Cloud 54
3.5.5. Configuring Microsoft Azure 56
3.5.6. Additional resources 58
3.6. UNINSTALLING MTC AND DELETING RESOURCES 58

CHAPTER 4. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS IN A RESTRICTED NETWORK ENVIRONMENT 60
4.1. COMPATIBILITY GUIDELINES 60
4.2. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSHIFT CONTAINER PLATFORM 4.14 61
4.3. INSTALLING THE LEGACY MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSHIFT CONTAINER PLATFORM 4.2 TO 4.5 62
4.4. PROXY CONFIGURATION 64

4.4.1. Direct volume migration 64
4.4.1.1. TCP proxy setup for DVM 64
4.4.1.2. Why use a TCP proxy instead of an HTTP/HTTPS proxy? 65
4.4.1.3. Known issue 65
4.4.2. Tuning network policies for migrations 65
4.4.2.1. NetworkPolicy configuration 65
4.4.2.1.1. Egress traffic from Rsync pods 65
4.4.2.1.2. Ingress traffic to Rsync pods 66
4.4.2.2. EgressNetworkPolicy configuration 66
4.4.2.3. Choosing alternate endpoints for data transfer 67
4.4.2.4. Configuring supplemental groups for Rsync pods 67
4.4.3. Configuring proxies 67
4.5. RUNNING RSYNC AS EITHER ROOT OR NON-ROOT 68
Manually overriding default non-root operation for data transfer 69
4.5.1. Configuring the MigrationController CR as root or non-root for all migrations 69
4.5.2. Configuring the MigMigration CR as root or non-root per migration 69
4.6. CONFIGURING A REPLICATION REPOSITORY 70
4.6.1. Prerequisites 70
4.6.2. Retrieving Multicloud Object Gateway credentials 70
4.6.3. Additional resources 70
4.7. UNINSTALLING MTC AND DELETING RESOURCES 71

CHAPTER 5. UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS 73
5.1. UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS ON OPENSHIFT CONTAINER PLATFORM 4.14 73
5.2. UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS TO 1.8.0 74
5.2.1. Upgrading OADP 1.0 to 1.2 for Migration Toolkit for Containers 1.8.0 75
5.3. UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS ON OPENSHIFT CONTAINER PLATFORM VERSIONS 4.2 TO 4.5 76
5.4. UPGRADING MTC 1.3 TO 1.8 77

CHAPTER 6. PREMIGRATION CHECKLISTS 79
6.1. CLUSTER HEALTH CHECKLIST 79
6.2. SOURCE CLUSTER CHECKLIST 79
6.3. TARGET CLUSTER CHECKLIST 80

CHAPTER 7. NETWORK CONSIDERATIONS 81
7.1. DNS CONSIDERATIONS 81
7.1.1. Isolating the DNS domain of the target cluster from the clients 81
7.1.2. Setting up the target cluster to accept the source DNS domain 81
7.2. NETWORK TRAFFIC REDIRECTION STRATEGIES 82

CHAPTER 8. DIRECT MIGRATION REQUIREMENTS 84
8.1. PREREQUISITES 84
8.2. RSYNC CONFIGURATION FOR DIRECT VOLUME MIGRATION 84
8.2.1. Resource limit configurations for Rsync pods 86
8.2.1.1. Supplemental group configuration for Rsync pods 86
8.2.1.2. Rsync retry configuration 87
8.2.1.3. Running Rsync as either root or non-root 87
8.2.1.3.1. Manually overriding default non-root operation for data transfer 87
8.2.1.3.2. About pod security admission 88
8.2.1.3.2.1. Pod security admission modes 88
8.2.1.3.2.2. Pod security admission profiles 88
8.2.1.3.2.3. Privileged namespaces 89

8.2.1.3.3. Configuring the MigrationController CR as root or non-root for all migrations 89
8.2.1.3.4. Configuring the MigMigration CR as root or non-root per migration 90
8.2.2. MigCluster Configuration 90
8.3. DIRECT MIGRATION KNOWN ISSUES 91
8.3.1. Applying the Skip SELinux relabel workaround with spc_t automatically on workloads running on OpenShift Container Platform 91
8.3.1.1. Diagnosing the need for the Skip SELinux relabel workaround 91
8.3.1.2. Resolving using the Skip SELinux relabel workaround 92

CHAPTER 9. MIGRATING YOUR APPLICATIONS 93
9.1. MIGRATION PREREQUISITES 93
9.2. MIGRATING YOUR APPLICATIONS BY USING THE MTC WEB CONSOLE 94
9.2.1. Launching the MTC web console 94
9.2.2. Adding a cluster to the MTC web console 95
9.2.3. Adding a replication repository to the MTC web console 97
9.2.4. Creating a migration plan in the MTC web console 98
Additional resources for persistent volume copy methods 100
9.2.5. Running a migration plan in the MTC web console 101

CHAPTER 10. ADVANCED MIGRATION OPTIONS 103
10.1. TERMINOLOGY 103
10.2. MIGRATING APPLICATIONS BY USING THE COMMAND LINE 104
10.2.1. Migration prerequisites 104
10.2.2. Creating a registry route for direct image migration 105
10.2.3. Proxy configuration 105
10.2.3.1. Direct volume migration 105
10.2.3.1.1. TCP proxy setup for DVM 105
10.2.3.1.2. Why use a TCP proxy instead of an HTTP/HTTPS proxy? 106
10.2.3.1.3. Known issue 106
10.2.3.2. Tuning network policies for migrations 106
10.2.3.2.1. NetworkPolicy configuration 106
10.2.3.2.1.1. Egress traffic from Rsync pods 107
10.2.3.2.1.2. Ingress traffic to Rsync pods 107
10.2.3.2.2. EgressNetworkPolicy configuration 107
10.2.3.2.3. Choosing alternate endpoints for data transfer 108
10.2.3.2.4. Configuring supplemental groups for Rsync pods 108
10.2.3.3. Configuring proxies 108
10.2.4. Migrating an application by using the MTC API 109
10.2.5. State migration 115
Additional resources 116
10.3. MIGRATION HOOKS 116
10.3.1. Writing an Ansible playbook for a migration hook 116
10.3.1.1. Ansible modules 117
10.3.1.2. Environment variables 118
10.4. MIGRATION PLAN OPTIONS 118
10.4.1. Excluding resources 118
10.4.2. Mapping namespaces 119
10.4.3. Excluding persistent volume claims 120
10.4.4. Mapping persistent volume claims 120
10.4.5. Editing persistent volume attributes 121
10.4.6. Converting storage classes in the MTC web console 123
Additional resources 124
10.4.7. Performing a state migration of Kubernetes objects by using the MTC API 125

10.5. MIGRATION CONTROLLER OPTIONS 126
10.5.1. Increasing limits for large migrations 126
10.5.2. Enabling persistent volume resizing for direct volume migration 127
10.5.3. Enabling cached Kubernetes clients 129

CHAPTER 11. MIGRATING VIRTUAL MACHINE STORAGE 130
11.1. ABOUT THE VIRTUAL MACHINE STORAGE MIGRATION PROCESS 130
11.2. SUPPORTED PERSISTENT VOLUME ACTIONS 130
11.3. PREREQUISITES 131
11.4. DEPLOYING A VIRTUAL MACHINE 132
11.5. CREATING A MIGRATION PLAN IN THE MTC WEB CONSOLE 134
11.5.1. Creating the migration plan using YAML manifests 136
11.6. KNOWN ISSUES 137
11.6.1. Online migration limitations 137

CHAPTER 12. TROUBLESHOOTING 139
12.1. MTC WORKFLOW 139
About MTC custom resources 141
12.2. MIGRATION TOOLKIT FOR CONTAINERS CUSTOM RESOURCE MANIFESTS 142
12.2.1. DirectImageMigration 142
12.2.2. DirectImageStreamMigration 143
12.2.3. DirectVolumeMigration 143
12.2.4. DirectVolumeMigrationProgress 144
12.2.5. MigAnalytic 144
12.2.6. MigCluster 145
12.2.7. MigHook 146
12.2.8. MigMigration 147
12.2.9. MigPlan 147
12.2.10. MigStorage 149
12.3. LOGS AND DEBUGGING TOOLS 150
12.3.1. Viewing migration plan resources 150
12.3.2. Viewing a migration plan log 151
12.3.3. Using the migration log reader 151
12.3.4. Accessing performance metrics 152
12.3.4.1. Provided metrics 152
12.3.4.1.1. cam_app_workload_migrations 152
12.3.4.1.2. mtc_client_request_count 152
12.3.4.1.3. mtc_client_request_elapsed 153
12.3.4.1.4. Useful queries 153
12.3.5. Using the must-gather tool 154
12.3.6. Debugging Velero resources with the Velero CLI tool 154
12.3.6.1. Syntax 155
12.3.6.2. Help option 155
12.3.6.3. Describe command 155
12.3.6.4. Logs command 156
12.3.7. Debugging a partial migration failure 156
12.3.8. Using MTC custom resources for troubleshooting 157
12.4. COMMON ISSUES AND CONCERNS 161
12.4.1. Direct volume migration does not complete 161
12.4.2. Error messages and resolutions 162
12.4.2.1. CA certificate error displayed when accessing the MTC console for the first time 162
12.4.2.2. OAuth timeout error in the MTC console 162
12.4.2.3. Certificate signed by unknown authority error 163

12.4.2.4. Backup storage location errors in the Velero pod log 163
12.4.2.5. Pod volume backup timeout error in the Velero pod log 163
12.4.2.6. Restic verification errors in the MigMigration custom resource 164
12.4.2.7. Restic permission error when migrating from NFS storage with root_squash enabled 165
12.4.3. Applying the Skip SELinux relabel workaround with spc_t automatically on workloads running on OpenShift Container Platform 166
12.4.3.1. Diagnosing the need for the Skip SELinux relabel workaround 166
12.4.3.2. Resolving using the Skip SELinux relabel workaround 166
12.5. ROLLING BACK A MIGRATION 167
12.5.1. Rolling back a migration by using the MTC web console 167
12.5.2. Rolling back a migration from the command-line interface 168
12.5.3. Rolling back a migration manually 169
Additional resources 170

CHAPTER 1. ABOUT THE MIGRATION TOOLKIT FOR CONTAINERS

The Migration Toolkit for Containers (MTC) enables you to migrate stateful application workloads between OpenShift Container Platform 4 clusters at the granularity of a namespace.

NOTE

If you are migrating from OpenShift Container Platform 3, see About migrating from OpenShift Container Platform 3 to 4 and Installing the legacy Migration Toolkit for Containers Operator on OpenShift Container Platform 3.

You can migrate applications within the same cluster or between clusters by using state migration.

MTC provides a web console and an API, based on Kubernetes custom resources, to help you control the migration and minimize application downtime.

The MTC console is installed on the target cluster by default. You can configure the Migration Toolkit for Containers Operator to install the console on a remote cluster.

See Advanced migration options for information about the following topics:

- Automating your migration with migration hooks and the MTC API.
- Configuring your migration plan to exclude resources, support large-scale migrations, and enable automatic PV resizing for direct volume migration.

1.1. MTC 1.8 SUPPORT

- MTC 1.8.3 and earlier using OADP 1.3.z is supported on all {OCP-short} versions 4.15 and earlier.
- MTC 1.8.4 and later using OADP 1.3.z is currently supported on all {OCP-short} versions 4.15 and earlier.
- MTC 1.8.4 and later using OADP 1.4.z is currently supported on all supported {OCP-short} versions 4.13 and later.

1.1.1. Support for Migration Toolkit for Containers (MTC)

Table 1.1. Supported versions of MTC

MTC version	OpenShift Container Platform version	General availability	Full support ends	Maintenance ends	Extended Update Support (EUS)

1.8	• 4.14 • 4.15 • 4.16 • 4.17	02 Oct 2023	Release of 1.9	Release of 1.10	N/A
1.7	• 3 • 4.14 • 4.15 • 4.16	01 Mar 2022	01 Jul 2024	01 Jul 2025	N/A

For more details about EUS, see Extended Update Support.

1.2. TERMINOLOGY

Table 1.2. MTC terminology

Term	Definition
Source cluster	Cluster from which the applications are migrated.
Destination cluster[1]	Cluster to which the applications are migrated.
Replication repository	Object storage used for copying images, volumes, and Kubernetes objects during indirect migration or for Kubernetes objects during direct volume migration or direct image migration. The replication repository must be accessible to all clusters.
Host cluster	Cluster on which the **migration-controller** pod and the web console are running. The host cluster is usually the destination cluster but this is not required. The host cluster does not require an exposed registry route for direct image migration.

Term	Definition
Remote cluster	A remote cluster is usually the source cluster but this is not required. A remote cluster requires a **Secret** custom resource that contains the **migration-controller** service account token. A remote cluster requires an exposed secure registry route for direct image migration.
Indirect migration	Images, volumes, and Kubernetes objects are copied from the source cluster to the replication repository and then from the replication repository to the destination cluster.
Direct volume migration	Persistent volumes are copied directly from the source cluster to the destination cluster.
Direct image migration	Images are copied directly from the source cluster to the destination cluster.
Stage migration	Data is copied to the destination cluster without stopping the application. Running a stage migration multiple times reduces the duration of the cutover migration.
Cutover migration	The application is stopped on the source cluster and its resources are migrated to the destination cluster.
State migration	Application state is migrated by copying specific persistent volume claims to the destination cluster.
Rollback migration	Rollback migration rolls back a completed migration.

[1] Called the *target* cluster in the MTC web console.

1.3. MTC WORKFLOW

You can migrate Kubernetes resources, persistent volume data, and internal container images to OpenShift Container Platform 4.14 by using the Migration Toolkit for Containers (MTC) web console or the Kubernetes API.

MTC migrates the following resources:

- A namespace specified in a migration plan.
- Namespace-scoped resources: When the MTC migrates a namespace, it migrates all the objects and resources associated with that namespace, such as services or pods. Additionally, if a resource that exists in the namespace but not at the cluster level depends on a resource that exists at the cluster level, the MTC migrates both resources.
  For example, a security context constraint (SCC) is a resource that exists at the cluster level and a service account (SA) is a resource that exists at the namespace level. If an SA exists in a namespace that the MTC migrates, the MTC automatically locates any SCCs that are linked to

the SA and also migrates those SCCs. Similarly, the MTC migrates persistent volumes that are linked to the persistent volume claims of the namespace.

NOTE

Cluster-scoped resources might have to be migrated manually, depending on the resource.

- Custom resources (CRs) and custom resource definitions (CRDs): MTC automatically migrates CRs and CRDs at the namespace level.

Migrating an application with the MTC web console involves the following steps:

1. Install the Migration Toolkit for Containers Operator on all clusters.
   You can install the Migration Toolkit for Containers Operator in a restricted environment with limited or no internet access. The source and target clusters must have network access to each other and to a mirror registry.

2. Configure the replication repository, an intermediate object storage that MTC uses to migrate data.
   The source and target clusters must have network access to the replication repository during migration. If you are using a proxy server, you must configure it to allow network traffic between the replication repository and the clusters.

3. Add the source cluster to the MTC web console.

4. Add the replication repository to the MTC web console.

5. Create a migration plan, with one of the following data migration options:

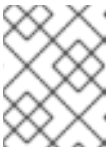
   - **Copy**: MTC copies the data from the source cluster to the replication repository, and from the replication repository to the target cluster.

NOTE

If you are using direct image migration or direct volume migration, the images or volumes are copied directly from the source cluster to the target cluster.

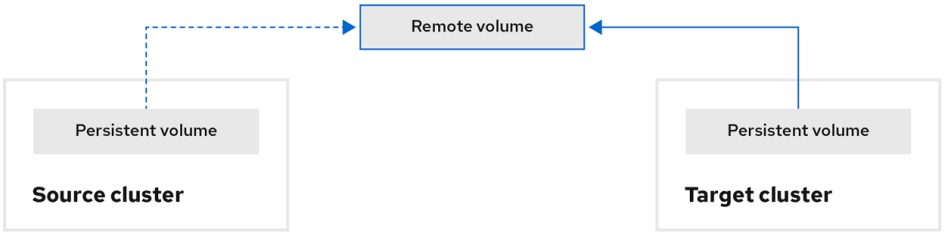
   - **Move**: MTC unmounts a remote volume, for example, NFS, from the source cluster, creates a PV resource on the target cluster pointing to the remote volume, and then mounts the remote volume on the target cluster. Applications running on the target cluster use the

same remote volume that the source cluster was using. The remote volume must be accessible to the source and target clusters.



NOTE

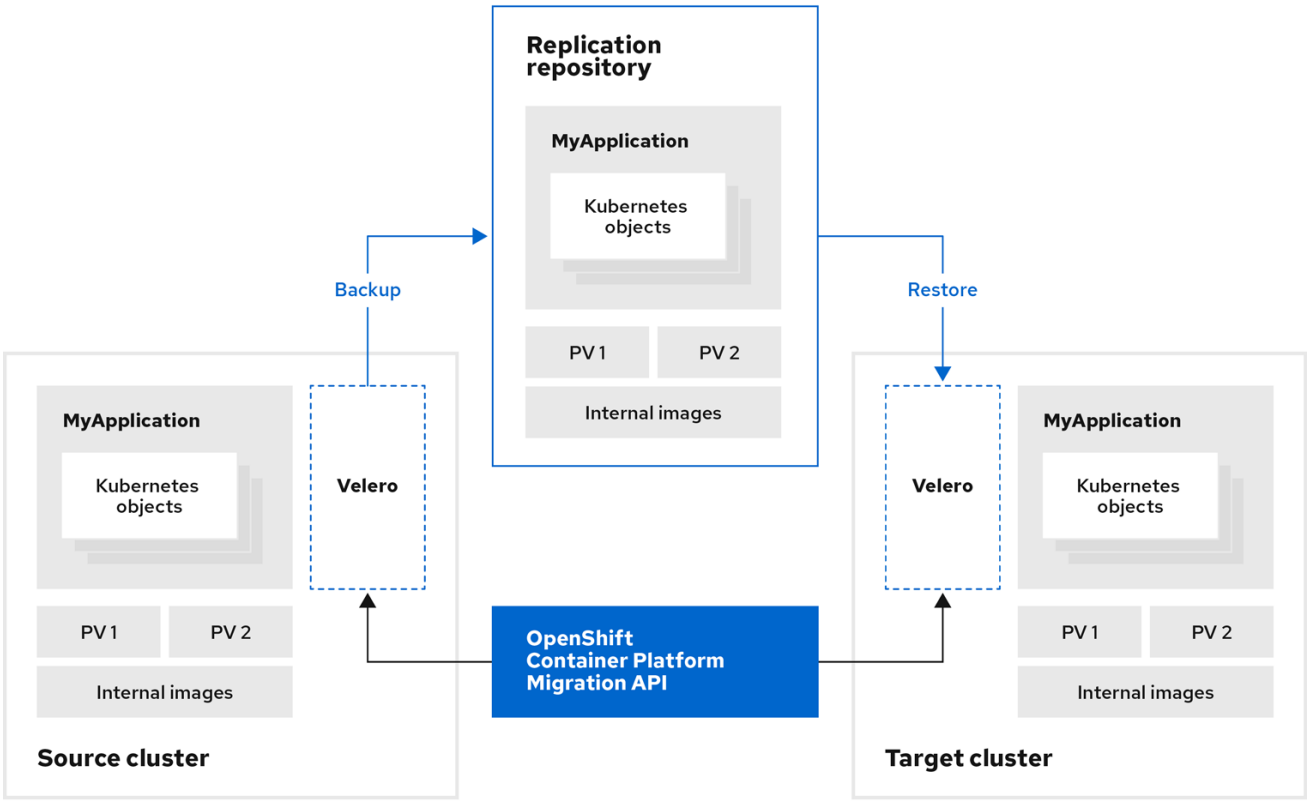
Although the replication repository does not appear in this diagram, it is required for migration.



6. Run the migration plan, with one of the following options:

   - **Stage** copies data to the target cluster without stopping the application.
     A stage migration can be run multiple times so that most of the data is copied to the target before migration. Running one or more stage migrations reduces the duration of the cutover migration.

   - **Cutover** stops the application on the source cluster and moves the resources to the target cluster.
     Optional: You can clear the **Halt transactions on the source cluster during migration** checkbox.



1.4. ABOUT DATA COPY METHODS

The Migration Toolkit for Containers (MTC) supports the file system and snapshot data copy methods for migrating data from the source cluster to the target cluster. You can select a method that is suited for your environment and is supported by your storage provider.

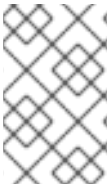
1.4.1. File system copy method

MTC copies data files from the source cluster to the replication repository, and from there to the target cluster.

The file system copy method uses Restic for indirect migration or Rsync for direct volume migration.

Table 1.3. File system copy method summary

Benefits	Limitations
• Clusters can have different storage classes. • Supported for all S3 storage providers. • Optional data verification with checksum. • Supports direct volume migration, which significantly increases performance.	• Slower than the snapshot copy method. • Optional data verification significantly reduces performance.



NOTE

The Restic and Rsync PV migration assumes that the PVs supported are only **volumeMode=filesystem**. Using **volumeMode=Block** for file system migration is *not* supported.

1.4.2. Snapshot copy method

MTC copies a snapshot of the source cluster data to the replication repository of a cloud provider. The data is restored on the target cluster.

The snapshot copy method can be used with Amazon Web Services, Google Cloud, and Microsoft Azure.

Table 1.4. Snapshot copy method summary

Benefits	Limitations

Benefits	Limitations
• Faster than the file system copy method.	• Cloud provider must support snapshots. • Clusters must be on the same cloud provider. • Clusters must be in the same location or region. • Clusters must have the same storage class. • Storage class must be compatible with snapshots. • Does not support direct volume migration.

1.5. DIRECT VOLUME MIGRATION AND DIRECT IMAGE MIGRATION

You can use direct image migration (DIM) and direct volume migration (DVM) to migrate images and data directly from the source cluster to the target cluster.

If you run DVM with nodes that are in different availability zones, the migration might fail because the migrated pods cannot access the persistent volume claim.

DIM and DVM have significant performance benefits because the intermediate steps of backing up files from the source cluster to the replication repository and restoring files from the replication repository to the target cluster are skipped. The data is transferred with Rsync.

DIM and DVM have additional prerequisites.

CHAPTER 2. MTC RELEASE NOTES

2.1. MIGRATION TOOLKIT FOR CONTAINERS 1.8 RELEASE NOTES

The release notes for Migration Toolkit for Containers (MTC) describe new features and enhancements, deprecated features, and known issues.

The MTC enables you to migrate application workloads between OpenShift Container Platform clusters at the granularity of a namespace.

MTC provides a web console and an API, based on Kubernetes custom resources, to help you control the migration and minimize application downtime.

For information on the support policy for MTC, see OpenShift Application and Cluster Migration Solutions, part of the *Red Hat OpenShift Container Platform Life Cycle Policy* .

2.1.1. Migration Toolkit for Containers 1.8.11 release notes

Migration Toolkit for Containers (MTC) 1.8.11 is a Container Grade Only (CGO) release, which is released to refresh the health grades of the containers. No code was changed in the product itself compared to that of MTC 1.8.10.

2.1.2. Migration Toolkit for Containers 1.8.10 release notes

2.1.2.1. Resolved issues

Migration Toolkit for Containers (MTC) 1.8.10 has the following major resolved issues:

migmigration failed to clean up stale virt-handler pods for stopped VMs

With this release, **migmigration** object no longer fails to clean up stale **virt-handler** pods for stopped virtual machines (VMs) in an Azure Red Hat OpenShift (ARO) cluster with MTC Operator and Red Hat OpenShift Data Foundation (ODF) storage cluster-CEPH RBD virtualization. This resolution addresses an issue where migration failure occurred due to an attempt to clean up non-existent **virt-handler** pods during live migration when the VM was stopped and no VMI existed. As a result, stopped VMs no longer prevent successful migration. (MIG-1762)

2.1.3. Migration Toolkit for Containers 1.8.9 release notes

2.1.3.1. Resolved issues

Migration Toolkit for Containers (MTC) 1.8.9 has the following major resolved issues:

VM restarts migration after the MigPlan deletion

Before this update, deleting the **MigPlan** triggered unnecessary virtual machine (VM) migrations. This caused the VMs to resume, leading to migration failures. This release introduces a fix that prevents VMs from migrating when the **MigPlan** is actively removed. As a result, VMs no longer unintentionally migrate when MigPlans are deleted, ensuring a stable environment for users. (MIG-1749)

Virt-launcher pod remain in pending after migrating the VM from hostpath-csi-basic to hostpath-csi-pvc-block

Before this update, the **virt-launcher** pod stalled after migrating a VM due to pod anti-affinity conflicts

with bound volumes. This resulted in a failed VM migration, causing the virt-launcher pod to get stuck. This release resolves the pod anti-affinity issue, allowing for scheduling on different nodes. As a result, you can now migrate VMs from **hostpath-csi-basic** to **hostpath-csi-pvc-block** without pods getting stuck in a pending state, thereby improving the overall migration process efficiency. (MIG-1750)

Migration-controller panics due to LimitRange "NIL" found in source project

Before this update, the missing **Min** spec in a LimitRange within the **openshift-migration** namespace caused a panic in the migration-controller pod, disrupting the user experience and preventing successful volume migration. With this release, the migration-controller pod no longer crashes when encountering a LimitRange set to **Nil**. As a result, **rsync** pods now comply with the specified limits, improving the migration process stability and eliminating crashloopbacks in source namespaces.

2.1.4. Migration Toolkit for Containers 1.8.8 release notes

2.1.4.1. Resolved issues

Migration Toolkit for Containers (MTC) 1.8.8 has the following major resolved issues:

The detection of storage live migration status failed

There was an error in detecting the status of storage live migration. This update resolves the issue. (MIG-1746)

2.1.5. Migration Toolkit for Containers 1.8.7 release notes

2.1.5.1. Resolved issues

Migration Toolkit for Containers (MTC) 1.8.7 has the following major resolved issues:

MTC migration stuck in a Prepare phase on OpenShift Container Platform 4.19 due to incompatible OADP version and outdated DPA specification

When running migrations use MTC 1.8.7 on OpenShift Container Platform 4.19, the process halts in the **Prepare** phase and the migration plan enters in the **Suspended** phase.

The root cause is the deployment of an incompatible OADP version, earlier than 1.5.0, whose DataProtectionApplication (DPA) specification format is incompatible with OpenShift Container Platform. (MIG-1735)

Velero backup fails with a file already closed error when using the new AWS plugin in MTC

During stage or full migrations, the backup process intermittently fails for MTC with OADP on OpenShift Container Platform clusters configured with the new Amazon Web Services (AWS) plugin. You can see the following error in Velero logs:

```
error="read |0: file already closed"
```

As a workaround, use the legacy AWS plugin by performing following actions:

1. Set **velero_use_legacy_aws: true** in the **MigrationController** custom resource (CR).
2. Restart the MTC Operator to apply changes.
3. Validate the AWS credentials for the cloud-credentials secret.

(MIG-1738)

2.1.6. Migration Toolkit for Containers 1.8.6 release notes

2.1.6.1. Technical changes

Multiple migration plans for a namespace is not supported

MTC version 1.8.6 and later do not support multiple migration plans for a single namespace.

VM storage migration

VM storage migration feature changes from Technology Preview (TP) status to being Generally Available (GA).

2.1.6.2. Resolved issues

MTC 1.8.6 has the following major resolved issues:

UI fails during a namespace search in the migration plan wizard

When searching for a namespace in the **Select Namespace** step of the migration plan wizard, the user interface (UI) fails and disappears after clicking **Search**. The browser console shows a JavaScript error indicating that an undefined value has been accessed. This issue has been resolved in MTC 1.8.6. (MIG-1704)

Unable to create a migration plan due to a reconciliation failure

In MTC, when creating a migration plan, the UI remains on **Persistent Volumes** and you cannot continue. This issue occurs due to a critical reconciliation failure and returns a 404 API error when you attempt to fetch the migration plan from the backend. These issues cause the migration plan to remain in a **Not Ready** state, and you are prevented from continuing. This issue has been resolved in MTC 1.8.6. (MIG-1705)

Migration process becomes fails to complete after the StageBackup phase

When migrating a Django and PostgreSQL application, the migration becomes fails to complete after the **StageBackup** phase. Even though all the pods in the source namespace are healthy before the migration begins, after the migration and when terminating the pods on the source cluster, the Django pod fails with a **CrashLoopBackOff** error. This issue has been resolved in MTC 1.8.6. (MIG-1707)

Migration shown as succeeded despite a failed phase due to a misleading UI status

After running a migration using MTC, the UI incorrectly indicates that the migration was successful, with the status shown as **Migration succeeded**. However, the Direct Volume Migration (DVM) phase failed. This misleading status appears on both the **Migration** and the **Migration Details** pages. This issue has been resolved in MTC 1.8.6. (MIG-1711)

Persistent Volumes page hangs indefinitely for namespaces without persistent volume claims

When a migration plan includes a namespace that does not have any persistent volume claims (PVCs), the **Persistent Volumes** selection page remains indefinitely with the following message shown: **Discovering persistent volumes attached to source projects...**. The page never completes loading, preventing you from proceeding with the migration. This issue has been resolved in MTC 1.8.6. (MIG-1713)

2.1.6.3. Known issues

MTC 1.8.6 has the following known issues:

Inconsistent reporting of migration failure status

There is a discrepancy in the reporting of namespace migration status following a rollback and subsequent re-migration attempt when the migration plan is deliberately faulted. Although the Distributed Volume Migration (DVM) phase correctly registers a failure, this failure is not consistently reflected in the user interface (UI) or the migration plan's YAML representation.

This issue is not only limited to unusual or unexpected cases. In certain circumstances, such as when network restrictions are applied that cause the DVM phase to fail, the UI still reports the migration status as successful. This behavior is similar to what was previously observed in MIG-1711 but occurs under different conditions. (MIG-1719)

2.1.7. Migration Toolkit for Containers 1.8.5 release notes

2.1.7.1. Technical changes

Migration Toolkit for Containers (MTC) 1.8.5 has the following technical changes:

Federal Information Processing Standard (FIPS)

FIPS is a set of computer security standards developed by the United States federal government in accordance with the Federal Information Security Management Act (FISMA).

Starting with version 1.8.5, MTC is designed for FIPS compliance.

2.1.7.2. Resolved issues

For more information, see the list of MTC 1.8.5 resolved issues in Jira.

2.1.7.3. Known issues

MTC 1.8.5 has the following known issues:

The associated SCC for service account cannot be migrated in OpenShift Container Platform 4.12

The associated Security Context Constraints (SCCs) for service accounts in OpenShift Container Platform 4.12 cannot be migrated. This issue is planned to be resolved in a future release of MTC. (MIG-1454)

MTC does not patch **statefulset.spec.volumeClaimTemplates[].spec.storageClassName** on storage class conversion

While running a Storage Class conversion for a StatefulSet application, MTC updates the persistent volume claims (PVC) references in **.spec.volumeClaimTemplates[].metadata.name** to use the migrated PVC names. MTC does not update **spec.volumeClaimTemplates[].spec.storageClassName**, which causes the application to scale up. Additionally, new replicas consume PVCs created under the old Storage Class instead of the migrated Storage Class. (MIG-1660)

Performing a StorageClass conversion triggers the scale-down of all applications in the namespace

When running a **StorageClass** conversion on more than one application, MTC scales down all the applications in the cutover phase, including those not involved in the migration. (MIG-1661)

MigPlan cannot be edited to have the same target namespace as the source cluster after it is changed

After changing the target namespace to something different from the source namespace while creating a **MigPlan** in the MTC UI, you cannot edit the **MigPlan** again to make the target namespace the same as the source namespace. (MIG-1600)

Migrated builder pod fails to push to the image registry

When migrating an application that includes **BuildConfig** from the source to the target cluster, the builder pod encounters an error, failing to push the image to the image registry. (BZ#2234781)

Conflict condition clears briefly after it is displayed

When creating a new state migration plan that results in a conflict error, the error is cleared shortly after it is displayed. (BZ#2144299)

PvCapacityAdjustmentRequired warning not displayed after setting pv_resizing_threshold

The **PvCapacityAdjustmentRequired** warning does not appear in the migration plan after the **pv_resizing_threshold** is adjusted. (BZ#2270160)

For a complete list of all known issues, see the list of MTC 1.8.5 known issues in Jira.

2.1.8. Migration Toolkit for Containers 1.8.4 release notes

2.1.8.1. Technical changes

Migration Toolkit for Containers (MTC) 1.8.4 has the following technical changes:

- MTC 1.8.4 extends its dependency resolution to include support for using OpenShift API for Data Protection (OADP) 1.4.

Support for KubeVirt Virtual Machines with DirectVolumeMigration

MTC 1.8.4 adds support for KubeVirt Virtual Machines (VMs) with Direct Volume Migration (DVM).

2.1.8.2. Resolved issues

MTC 1.8.4 has the following major resolved issues:

Ansible Operator is broken when OpenShift Virtualization is installed

There is a bug in the **python3-openshift** package that installing OpenShift Virtualization exposes, with an exception, **ValueError: too many values to unpack**, returned during the task. Earlier versions of MTC are impacted, while MTC 1.8.4 has implemented a workaround. Updating to MTC 1.8.4 means you are no longer affected by this issue. (OCPBUGS-38116)

UI stuck at Namespaces while creating a migration plan

When trying to create a migration plan from the MTC UI, the migration plan wizard becomes stuck at the **Namespaces** step. This issue has been resolved in MTC 1.8.4. (MIG-1597)

Migration fails with error of no matches for kind Virtual machine in version kubevirt/v1

During the migration of an application, all the necessary steps, including the backup, DVM, and restore, are successfully completed. However, the migration is marked as unsuccessful with the error message **no matches for kind Virtual machine in version kubevirt/v1**. (MIG-1594)

Direct Volume Migration fails when migrating to a namespace different from the source namespace

On performing a migration from source cluster to target cluster, with the target namespace different from the source namespace, the DVM fails. (MIG-1592)

Direct Image Migration does not respect label selector on migplan

When using Direct Image Migration (DIM), if a label selector is set on the migration plan, DIM does not respect it and attempts to migrate all imagestreams in the namespace. (MIG-1533)

2.1.8.3. Known issues

MTC 1.8.4 has the following known issues:

The associated SCC for service account cannot be migrated in OpenShift Container Platform 4.12

The associated Security Context Constraints (SCCs) for service accounts in OpenShift Container Platform 4.12 cannot be migrated. This issue is planned to be resolved in a future release of MTC. (MIG-1454).

Rsync pod fails to start causing the DVM phase to fail

The DVM phase fails due to the Rsync pod failing to start, because of a permission issue.

(BZ#2231403)

Migrated builder pod fails to push to image registry

When migrating an application including **BuildConfig** from source to target cluster, the builder pod results in error, failing to push the image to the image registry.

(BZ#2234781)

Conflict condition gets cleared briefly after it is created

When creating a new state migration plan that results in a conflict error, that error is cleared shorty after it is displayed.

(BZ#2144299)

PvCapacityAdjustmentRequired Warning Not Displayed After Setting pv_resizing_threshold

The **PvCapacityAdjustmentRequired** warning fails to appear in the migration plan after the **pv_resizing_threshold** is adjusted.

(BZ#2270160)

2.1.9. Migration Toolkit for Containers 1.8.3 release notes

2.1.9.1. Technical changes

Migration Toolkit for Containers (MTC) 1.8.3 has the following technical changes:

OADP 1.3 is now supported

MTC 1.8.3 adds support to OpenShift API for Data Protection (OADP) as a dependency of MTC 1.8.z.

2.1.9.2. Resolved issues

MTC 1.8.3 has the following major resolved issues:

CVE-2024-24786: Flaw in Golang **protobuf** module causes **unmarshal** function to enter infinite loop

In previous releases of MTC, a vulnerability was found in Golang's **protobuf** module, where the **unmarshal** function entered an infinite loop while processing certain invalid inputs. Consequently, an attacker provided carefully constructed invalid inputs, which caused the function to enter an infinite loop.

With this update, the **unmarshal** function works as expected.

For more information, see CVE-2024-24786.

CVE-2023-45857: Axios Cross-Site Request Forgery Vulnerability

In previous releases of MTC, a vulnerability was discovered in Axios 1.5.1 that inadvertently revealed a confidential **XSRF-TOKEN** stored in cookies by including it in the HTTP header **X-XSRF-TOKEN** for every request made to the host, allowing attackers to view sensitive information.

For more information, see CVE-2023-45857.

Restic backup does not work properly when the source workload is not quiesced

In previous releases of MTC, some files did not migrate when deploying an application with a route. The Restic backup did not function as expected when the quiesce option was unchecked for the source workload.

This issue has been resolved in MTC 1.8.3.

For more information, see BZ#2242064.

The **Migration Controller** fails to install due to an unsupported value error in Velero

The **MigrationController** failed to install due to an unsupported value error in Velero. Updating OADP 1.3.0 to OADP 1.3.1 resolves this problem. For more information, see BZ#2267018.

This issue has been resolved in MTC 1.8.3.

For a complete list of all resolved issues, see the list of MTC 1.8.3 resolved issues in Jira.

2.1.9.3. Known issues

Migration Toolkit for Containers (MTC) 1.8.3 has the following known issues:

Ansible Operator is broken when OpenShift Virtualization is installed

There is a bug in the **python3-openshift** package that installing OpenShift Virtualization exposes, with an exception, **ValueError: too many values to unpack**, returned during the task. MTC 1.8.4 has implemented a workaround. Updating to MTC 1.8.4 means you are no longer affected by this issue.

(OCPBUGS-38116)

The associated SCC for service account cannot be migrated in OpenShift Container Platform 4.12

The associated Security Context Constraints (SCCs) for service accounts in OpenShift Container Platform version 4.12 cannot be migrated. This issue is planned to be resolved in a future release of MTC. (MIG-1454).

For a complete list of all known issues, see the list of MTC 1.8.3 known issues in Jira.

2.1.10. Migration Toolkit for Containers 1.8.2 release notes

2.1.10.1. Resolved issues

This release has the following major resolved issues:

Backup phase fails after setting custom CA replication repository

In previous releases of Migration Toolkit for Containers (MTC), after editing the replication repository, adding a custom CA certificate, successfully connecting the repository, and triggering a migration, a failure occurred during the backup phase.

CVE-2023-26136: tough-cookie package before 4.1.3 are vulnerable to Prototype Pollution

In previous releases of (MTC), versions before 4.1.3 of the **tough-cookie** package used in MTC were vulnerable to prototype pollution. This vulnerability occurred because CookieJar did not handle cookies properly when the value of the **rejectPublicSuffixes** was set to **false**.

For more details, see (CVE-2023-26136)

CVE-2022-25883 openshift-migration-ui-container: nodejs-semver: Regular expression denial of service

In previous releases of (MTC), versions of the **semver** package before 7.5.2, used in MTC, were vulnerable to Regular Expression Denial of Service (ReDoS) from the function **newRange**, when untrusted user data was provided as a range.

For more details, see (CVE-2022-25883)

2.1.10.2. Known issues

MTC 1.8.2 has the following known issues:

Ansible Operator is broken when OpenShift Virtualization is installed

There is a bug in the **python3-openshift** package that installing OpenShift Virtualization exposes, with an exception, **ValueError: too many values to unpack**, returned during the task. MTC 1.8.4 has implemented a workaround. Updating to MTC 1.8.4 means you are no longer affected by this issue. (OCPBUGS-38116)

2.1.11. Migration Toolkit for Containers 1.8.1 release notes

2.1.11.1. Resolved issues

Migration Toolkit for Containers (MTC) 1.8.1 has the following major resolved issues:

CVE-2023-39325: golang: net/http, x/net/http2: rapid stream resets can cause excessive work

A flaw was found in handling multiplexed streams in the HTTP/2 protocol, which is used by MTC. A client could repeatedly make a request for a new multiplex stream and immediately send an **RST_STREAM** frame to cancel it. This creates additional workload for the server in terms of setting up and dismantling streams, while avoiding any server-side limitations on the maximum number of active streams per connection, resulting in a denial of service due to server resource consumption. (BZ#2245079)

It is advised to update to MTC 1.8.1 or later, which resolve this issue.

For more details, see (CVE-2023-39325) and (CVE-2023-44487)

2.1.11.2. Known issues

Migration Toolkit for Containers (MTC) 1.8.1 has the following known issues:

Ansible Operator is broken when OpenShift Virtualization is installed

There is a bug in the **python3-openshift** package that installing OpenShift Virtualization exposes. An exception, **ValueError: too many values to unpack**, is returned during the task. MTC 1.8.4 has implemented a workaround. Updating to MTC 1.8.4 means you are no longer affected by this issue. (OCPBUGS-38116)

2.1.12. Migration Toolkit for Containers 1.8.0 release notes

2.1.12.1. Resolved issues

Migration Toolkit for Containers (MTC) 1.8.0 has the following resolved issues:

Indirect migration is stuck on backup stage

In previous releases, an indirect migration became stuck at the backup stage, due to **InvalidImageName** error. ((BZ#2233097))

PodVolumeRestore remain In Progress keeping the migration stuck at Stage Restore

In previous releases, on performing an indirect migration, the migration became stuck at the **Stage Restore** step, waiting for the **podvolumerestore** to be completed. ((BZ#2233868))

Migrated application unable to pull image from internal registry on target cluster

In previous releases, on migrating an application to the target cluster, the migrated application failed to pull the image from the internal image registry resulting in an **application failure**. ((BZ#2233103))

Migration failing on Azure due to authorization issue

In previous releases, on an Azure cluster, when backing up to Azure storage, the migration failed at the **Backup** stage. ((BZ#2238974))

2.1.12.2. Known issues

MTC 1.8.0 has the following known issues:

Ansible Operator is broken when OpenShift Virtualization is installed

There is a bug in the **python3-openshift** package that installing OpenShift Virtualization exposes, with an exception **ValueError: too many values to unpack** returned during the task. MTC 1.8.4 has implemented a workaround. Updating to MTC 1.8.4 means you are no longer affected by this issue. (OCPBUGS-38116)

Old Restic pods are not getting removed on upgrading MTC 1.7.x → 1.8.x

In this release, on upgrading the MTC Operator from 1.7.x to 1.8.x, the old Restic pods are not being removed. Therefore after the upgrade, both Restic and node-agent pods are visible in the namespace. ((BZ#2236829))

Migrated builder pod fails to push to image registry

In this release, on migrating an application including a **BuildConfig** from a source to target cluster, builder pod results in **error**, failing to push the image to the image registry. ((BZ#2234781))

[UI] CA bundle file field is not properly cleared

In this release, after enabling **Require SSL verification** and adding content to the CA bundle file for an MCG NooBaa bucket in MigStorage, the connection fails as expected. However, when reverting these changes by removing the CA bundle content and clearing **Require SSL verification**, the connection still fails. The issue is only resolved by deleting and re-adding the repository. ((BZ#2240052))

Backup phase fails after setting custom CA replication repository

In (MTC), after editing the replication repository, adding a custom CA certificate, successfully connecting the repository, and triggering a migration, a failure occurs during the backup phase.

This issue is resolved in MTC 1.8.2.

CVE-2023-26136: tough-cookie package before 4.1.3 are vulnerable to Prototype Pollution

Versions before 4.1.3 of the **tough-cookie** package, used in MTC, are vulnerable to prototype pollution. This vulnerability occurs because CookieJar does not handle cookies properly when the value of the **rejectPublicSuffixes** is set to **false**.

This issue is resolved in MTC 1.8.2.

For more details, see (CVE-2023-26136)

CVE-2022-25883 openshift-migration-ui-container: nodejs-semver: Regular expression denial of service

In previous releases of (MTC), versions of the **semver** package before 7.5.2, used in MTC, are vulnerable to Regular Expression Denial of Service (ReDoS) from the function **newRange**, when untrusted user data is provided as a range.

This issue is resolved in MTC 1.8.2.

For more details, see (CVE-2022-25883)

2.1.12.3. Technical changes

This release has the following technical changes:

- Migration from OpenShift Container Platform 3 to OpenShift Container Platform 4 requires a legacy Migration Toolkit for Containers Operator and Migration Toolkit for Containers 1.7.x.

- Migration from MTC 1.7.x to MTC 1.8.x is not supported.
- You must use MTC 1.7.x to migrate anything with a source of OpenShift Container Platform 4.9 or earlier.
    - MTC 1.7.x must be used on both source and destination.
- Migration Toolkit for Containers (MTC) 1.8.x only supports migrations from OpenShift Container Platform 4.10 or later to OpenShift Container Platform 4.10 or later. For migrations only involving cluster versions 4.10 and later, either 1.7.x or 1.8.x might be used. However, but it must be the same MTC 1.Y.z on both source and destination.
    - Migration from source MTC 1.7.x to destination MTC 1.8.x is unsupported.
    - Migration from source MTC 1.8.x to destination MTC 1.7.x is unsupported.
    - Migration from source MTC 1.7.x to destination MTC 1.7.x is supported.
    - Migration from source MTC 1.8.x to destination MTC 1.8.x is supported.
- MTC 1.8.x by default installs OADP 1.2.x.
- Upgrading from MTC 1.7.x to MTC 1.8.0, requires manually changing the OADP channel to 1.2. If this is not done, the upgrade of the Operator fails.

2.2. MIGRATION TOOLKIT FOR CONTAINERS 1.7 RELEASE NOTES

The release notes for Migration Toolkit for Containers (MTC) describe new features and enhancements, deprecated features, and known issues.

The MTC enables you to migrate application workloads between OpenShift Container Platform clusters at the granularity of a namespace.

You can migrate from OpenShift Container Platform 3 to 4.14 and between OpenShift Container Platform 4 clusters.

MTC provides a web console and an API, based on Kubernetes custom resources, to help you control the migration and minimize application downtime.

For information on the support policy for MTC, see OpenShift Application and Cluster Migration Solutions, part of the *Red Hat OpenShift Container Platform Life Cycle Policy* .

2.2.1. Migration Toolkit for Containers 1.7.19 release notes

Migration Toolkit for Containers (MTC) 1.7.19 is a Container Grade Only (CGO) release, which is released to refresh the health grades of the containers. No code was changed in the product itself compared to that of MTC 1.7.18.

2.2.1.1. Resolved issues

This release has the following resolved issues:

python3-requests package vulnerability

A bug was introduced when building the MTC Operator for the 1.7.19 release. During QE testing, this bug meant that when attempting to install MTC 1.7.19 from the OperatorHub, the migration-operator pod

failed during startup. The logs indicated that the Ansible Runner was crashing with an unhandled **KeyError: 'http+unix'** issue. The MTC Operator was in an incomplete state, and the required controller pods were never deployed, which rendered the Operator non-functional. This issue was caused by the **python3-requests** package.

The workaround was to downgrade to the last known working version of the **python3-requests** package, used in MTC 1.7.19, to **python3-requests-2.20.0-3.el8_8.noarch**. However, this action does mean exposure to the following vulnerability, RHSA-2023:4520.

2.2.1.2. Known issues

The restore fails due to a missing backup storage location

Currently, if the backup storage location is not specified, application migration fails. (MIG-1649)

Workaround: Roll back and then restart the migration.

2.2.2. Migration Toolkit for Containers 1.7.18 release notes

Migration Toolkit for Containers (MTC) 1.7.18 is a Container Grade Only (CGO) release, which is released to refresh the health grades of the containers. No code was changed in the product itself compared to that of MTC 1.7.17.

2.2.2.1. Technical changes

Migration Toolkit for Containers (MTC) 1.7.18 has the following technical changes:

Federal Information Processing Standard (FIPS)

FIPS is a set of computer security standards developed by the United States federal government in accordance with the Federal Information Security Management Act (FISMA).

Starting with version 1.7.18, MTC is designed for FIPS compliance.

2.2.3. Migration Toolkit for Containers 1.7.17 release notes

Migration Toolkit for Containers (MTC) 1.7.17 is a Container Grade Only (CGO) release, which is released to refresh the health grades of the containers. No code was changed in the product itself compared to that of MTC 1.7.16.

2.2.4. Migration Toolkit for Containers 1.7.16 release notes

2.2.4.1. Resolved issues

This release has the following resolved issues:

CVE-2023-45290: Golang: net/http: Memory exhaustion in the Request.ParseMultipartForm method

A flaw was found in the **net/http** Golang standard library package, which impacts earlier versions of MTC. When parsing a **multipart** form, either explicitly with **Request.ParseMultipartForm** or implicitly with **Request.FormValue**, **Request.PostFormValue**, or **Request.FormFile** methods, limits on the total size of the parsed form are not applied to the memory consumed while reading a single form line. This permits a maliciously crafted input containing long lines to cause the allocation of arbitrarily large amounts of memory, potentially leading to memory exhaustion.

To resolve this issue, upgrade to MTC 1.7.16.

For more details, see CVE-2023-45290

CVE-2024-24783: Golang: crypto/x509: Verify panics on certificates with an unknown public key algorithm

A flaw was found in the **crypto/x509** Golang standard library package, which impacts earlier versions of MTC. Verifying a certificate chain that contains a certificate with an unknown public key algorithm causes **Certificate.Verify** to panic. This affects all **crypto/tls** clients and servers that set **Config.ClientAuth** to **VerifyClientCertIfGiven** or **RequireAndVerifyClientCert**. The default behavior is for TLS servers to not verify client certificates.

To resolve this issue, upgrade to MTC 1.7.16.

For more details, see CVE-2024-24783.

CVE-2024-24784: Golang: net/mail: Comments in display names are incorrectly handled

A flaw was found in the **net/mail** Golang standard library package, which impacts earlier versions of MTC. The **ParseAddressList** function incorrectly handles comments, text in parentheses, and display names. As this is a misalignment with conforming address parsers, it can result in different trust decisions being made by programs using different parsers.

To resolve this issue, upgrade to MTC 1.7.16.

For more details, see CVE-2024-24784.

CVE-2024-24785: Golang: html/template: Errors returned from MarshalJSON methods may break template escaping

A flaw was found in the **html/template** Golang standard library package, which impacts earlier versions of MTC. If errors returned from **MarshalJSON** methods contain user-controlled data, they could be used to break the contextual auto-escaping behavior of the **html/template** package, allowing subsequent actions to inject unexpected content into templates.

To resolve this issue, upgrade to MTC 1.7.16.

For more details, see CVE-2024-24785.

CVE-2024-29180: webpack-dev-middleware: Lack of URL validation may lead to file leak

A flaw was found in the **webpack-dev-middleware package**, which impacts earlier versions of MTC. This flaw fails to validate the supplied URL address sufficiently before returning local files, which could allow an attacker to craft URLs to return arbitrary local files from the developer's machine.

To resolve this issue, upgrade to MTC 1.7.16.

For more details, see CVE-2024-29180.

CVE-2024-30255: envoy: HTTP/2 CPU exhaustion due to CONTINUATION frame flood

A flaw was found in how the **envoy** proxy implements the HTTP/2 codec, which impacts earlier versions of MTC. There are insufficient limitations placed on the number of **CONTINUATION** frames that can be sent within a single stream, even after exceeding the header map limits of **envoy**. This flaw could allow an unauthenticated remote attacker to send packets to vulnerable servers. These packets could consume compute resources and cause a denial of service (DoS).

To resolve this issue, upgrade to MTC 1.7.16.

For more details, see CVE-2024-30255.

2.2.4.2. Known issues

This release has the following known issues:

Direct Volume Migration is failing as the Rsync pod on the source cluster goes into an Error state

On migrating any application with a Persistent Volume Claim (PVC), the **Stage** migration operation succeeds with warnings, but the Direct Volume Migration (DVM) fails with the **rsync** pod on the source namespace moving into an **error** state. (BZ#2256141)

The conflict condition is briefly cleared after it is created

When creating a new state migration plan that returns a conflict error message, the error message is cleared very shortly after it is displayed. (BZ#2144299)

Migration fails when there are multiple Volume Snapshot Locations of different provider types configured in a cluster

When there are multiple Volume Snapshot Locations (VSLs) in a cluster with different provider types, but you have not set any of them as the default VSL, Velero results in a validation error that causes migration operations to fail. (BZ#2180565)

2.2.5. Migration Toolkit for Containers 1.7.15 release notes

2.2.5.1. Resolved issues

This release has the following resolved issues:

CVE-2024-24786: A flaw was found in Golang's protobuf module, where the unmarshal function can enter an infinite loop

A flaw was found in the **protojson.Unmarshal** function that could cause the function to enter an infinite loop when unmarshaling certain forms of invalid JSON messages. This condition could occur when unmarshaling into a message that contained a **google.protobuf.Any** value or when the **UnmarshalOptions.DiscardUnknown** option was set in a JSON-formatted message.

To resolve this issue, upgrade to MTC 1.7.15.

For more details, see (CVE-2024-24786).

CVE-2024-28180: jose-go improper handling of highly compressed data

A vulnerability was found in Jose due to improper handling of highly compressed data. An attacker could send a JSON Web Encryption (JWE) encrypted message that contained compressed data that used large amounts of memory and CPU when decompressed by the **Decrypt** or **DecryptMulti** functions.

To resolve this issue, upgrade to MTC 1.7.15.

For more details, see (CVE-2024-28180).

2.2.5.2. Known issues

This release has the following known issues:

Direct Volume Migration is failing as the Rsync pod on the source cluster goes into an Error state

On migrating any application with Persistent Volume Claim (PVC), the **Stage** migration operation succeeds with warnings, and Direct Volume Migration (DVM) fails with the **rsync** pod on the source namespace going into an **error** state. (BZ#2256141)

The conflict condition is briefly cleared after it is created

When creating a new state migration plan that results in a conflict error message, the error message is cleared shortly after it is displayed. (BZ#2144299)

Migration fails when there are multiple Volume Snapshot Locations (VSLs) of different provider types configured in a cluster with no specified default VSL.

When there are multiple VSLs in a cluster with different provider types, and you set none of them as the default VSL, Velero results in a validation error that causes migration operations to fail. (BZ#2180565)

2.2.6. Migration Toolkit for Containers 1.7.14 release notes

2.2.6.1. Resolved issues

This release has the following resolved issues:

CVE-2023-39325 CVE-2023-44487: various flaws

A flaw was found in the handling of multiplexed streams in the HTTP/2 protocol, which is utilized by Migration Toolkit for Containers (MTC). A client could repeatedly make a request for a new multiplex stream then immediately send an **RST_STREAM** frame to cancel those requests. This activity created additional workloads for the server in terms of setting up and dismantling streams, but avoided any server-side limitations on the maximum number of active streams per connection. As a result, a denial of service occurred due to server resource consumption.

- (BZ#2243564)
- (BZ#2244013)
- (BZ#2244014)
- (BZ#2244015)
- (BZ#2244016)
- (BZ#2244017)

To resolve this issue, upgrade to MTC 1.7.14.

For more details, see (CVE-2023-44487) and (CVE-2023-39325).

CVE-2023-39318 CVE-2023-39319 CVE-2023-39321: various flaws

- (CVE-2023-39318): A flaw was discovered in Golang, utilized by MTC. The **html/template** package did not properly handle HTML-like **""** comment tokens, or the hashbang **"#!"** comment tokens, in **<script>** contexts. This flaw could cause the template parser to improperly interpret the contents of **<script>** contexts, causing actions to be improperly escaped.

    - (BZ#2238062)
    - (BZ#2238088)

- (CVE-2023-39319): A flaw was discovered in Golang, utilized by MTC. The **html/template** package did not apply the proper rules for handling occurrences of **"<script"**, **"<!--"**, and **"</script"** within JavaScript literals in <script> contexts. This could cause the template parser to improperly consider script contexts to be terminated early, causing actions to be improperly escaped.
    - (BZ#2238062)
    - (BZ#2238088)
- (CVE-2023-39321): A flaw was discovered in Golang, utilized by MTC. Processing an incomplete post-handshake message for a QUIC connection could cause a panic.
    - (BZ#2238062)
    - (BZ#2238088)
- (CVE-2023-3932): A flaw was discovered in Golang, utilized by MTC. Connections using the QUIC transport protocol did not set an upper bound on the amount of data buffered when reading post-handshake messages, allowing a malicious QUIC connection to cause unbounded memory growth.
    - (BZ#2238088)

To resolve these issues, upgrade to MTC 1.7.14.

For more details, see (CVE-2023-39318), (CVE-2023-39319), and (CVE-2023-39321).

2.2.6.2. Known issues

There are no major known issues in this release.

2.2.7. Migration Toolkit for Containers 1.7.13 release notes

2.2.7.1. Resolved issues

There are no major resolved issues in this release.

2.2.7.2. Known issues

There are no major known issues in this release.

2.2.8. Migration Toolkit for Containers 1.7.12 release notes

2.2.8.1. Resolved issues

There are no major resolved issues in this release.

2.2.8.2. Known issues

This release has the following known issues:

Error code 504 is displayed on the Migration details page

On the **Migration details** page, at first, the **migration details** are displayed without any issues. However, after sometime, the details disappear, and a **504** error is returned. (BZ#2231106)

Old restic pods are not removed when upgrading Migration Toolkit for Containers 1.7.x to Migration Toolkit for Containers 1.8

On upgrading the Migration Toolkit for Containers (MTC) operator from 1.7.x to 1.8.x, the old restic pods are not removed. After the upgrade, both restic and node-agent pods are visible in the namespace. (BZ#2236829)

2.2.9. Migration Toolkit for Containers 1.7.11 release notes

2.2.9.1. Resolved issues

There are no major resolved issues in this release.

2.2.9.2. Known issues

There are no known issues in this release.

2.2.10. Migration Toolkit for Containers 1.7.10 release notes

2.2.10.1. Resolved issues

This release has the following major resolved issue:

Adjust rsync options in DVM

In this release, you can prevent absolute symlinks from being manipulated by Rsync in the course of direct volume migration (DVM). Running DVM in privileged mode preserves absolute symlinks inside the persistent volume claims (PVCs). To switch to privileged mode, in the **MigrationController** CR, set the **migration_rsync_privileged** spec to **true**. (BZ#2204461)

2.2.10.2. Known issues

There are no known issues in this release.

2.2.11. Migration Toolkit for Containers 1.7.9 release notes

2.2.11.1. Resolved issues

There are no major resolved issues in this release.

2.2.11.2. Known issues

This release has the following known issue:

Adjust rsync options in DVM

In this release, users are unable to prevent absolute symlinks from being manipulated by rsync during direct volume migration (DVM). (BZ#2204461)

2.2.12. Migration Toolkit for Containers 1.7.8 release notes

2.2.12.1. Resolved issues

This release has the following major resolved issues:

Velero image cannot be overridden in the Migration Toolkit for Containers (MTC) operator

In previous releases, it was not possible to override the velero image using the **velero_image_fqin** parameter in the **MigrationController** Custom Resource (CR). (BZ#2143389)

Adding a MigCluster from the UI fails when the domain name has more than six characters

In previous releases, adding a MigCluster from the UI failed when the domain name had more than six characters. The UI code expected a domain name of between two and six characters. (BZ#2152149)

UI fails to render the Migrations' page: Cannot read properties of undefined (reading 'name')

In previous releases, the UI failed to render the Migrations' page, returning **Cannot read properties of undefined (reading 'name')**. (BZ#2163485)

Creating DPA resource fails on Red Hat OpenShift Container Platform 4.6 clusters

In previous releases, when deploying MTC on an OpenShift Container Platform 4.6 cluster, the DPA failed to be created according to the logs, which resulted in some pods missing. From the logs in the migration-controller in the OpenShift Container Platform 4.6 cluster, it indicated that an unexpected **null** value was passed, which caused the error. (BZ#2173742)

2.2.12.2. Known issues

There are no known issues in this release.

2.2.13. Migration Toolkit for Containers 1.7.7 release notes

2.2.13.1. Resolved issues

There are no major resolved issues in this release.

2.2.13.2. Known issues

There are no known issues in this release.

2.2.14. Migration Toolkit for Containers 1.7.6 release notes

2.2.14.1. New features

Implement proposed changes for DVM support with PSA in Red Hat OpenShift Container Platform 4.12

With the incoming enforcement of Pod Security Admission (PSA) in OpenShift Container Platform 4.12 the default pod would run with a **restricted** profile. This **restricted** profile would mean workloads to migrate would be in violation of this policy and no longer work as of now. The following enhancement outlines the changes that would be required to remain compatible with OCP 4.12. (MIG-1240)

2.2.14.2. Resolved issues

This release has the following major resolved issues:

Unable to create Storage Class Conversion plan due to missing cronjob error in Red Hat OpenShift Platform 4.12

In previous releases, on the persistent volumes page, an error is thrown that a CronJob is not available in version **batch/v1beta1**, and when clicking on cancel, the migplan is created with status **Not ready**. (BZ#2143628)

2.2.14.3. Known issues

This release has the following known issue:

Conflict conditions are cleared briefly after they are created

When creating a new state migration plan that will result in a conflict error, that error is cleared shorty after it is displayed. (BZ#2144299)

2.2.15. Migration Toolkit for Containers 1.7.5 release notes

2.2.15.1. Resolved issues

This release has the following major resolved issue:

Direct Volume Migration is failing as rsync pod on source cluster move into Error state

In previous release, migration succeeded with warnings but Direct Volume Migration failed with rsync pod on source namespace going into error state. (*BZ#2132978)

2.2.15.2. Known issues

This release has the following known issues:

Velero image cannot be overridden in the Migration Toolkit for Containers (MTC) operator

In previous releases, it was not possible to override the velero image using the **velero_image_fqin** parameter in the **MigrationController** Custom Resource (CR). (BZ#2143389)

When editing a MigHook in the UI, the page might fail to reload

The UI might fail to reload when editing a hook if there is a network connection issue. After the network connection is restored, the page will fail to reload until the cache is cleared. (BZ#2140208)

2.2.16. Migration Toolkit for Containers 1.7.4 release notes

2.2.16.1. Resolved issues

There are no major resolved issues in this release.

2.2.16.2. Known issues

Rollback missing out deletion of some resources from the target cluster

On performing the roll back of an application from the Migration Toolkit for Containers (MTC) UI, some resources are not being deleted from the target cluster and the roll back is showing a status as successfully completed. (BZ#2126880)

2.2.17. Migration Toolkit for Containers 1.7.3 release notes

2.2.17.1. Resolved issues

This release has the following major resolved issues:

Correct DNS validation for destination namespace

In previous releases, the MigPlan could not be validated if the destination namespace started with a non-alphabetic character. (BZ#2102231)

Deselecting all PVCs from UI still results in an attempted PVC transfer

In previous releases, while doing a full migration, unselecting the persistent volume claims (PVCs) would not skip selecting the PVCs and still try to migrate them. (BZ#2106073)

Incorrect DNS validation for destination namespace

In previous releases, MigPlan could not be validated because the destination namespace started with a non-alphabetic character. (BZ#2102231)

2.2.17.2. Known issues

There are no known issues in this release.

2.2.18. Migration Toolkit for Containers 1.7.2 release notes

2.2.18.1. Resolved issues

This release has the following major resolved issues:

MTC UI does not display logs correctly

In previous releases, the Migration Toolkit for Containers (MTC) UI did not display logs correctly. (BZ#2062266)

StorageClass conversion plan adding migstorage reference in migplan

In previous releases, StorageClass conversion plans had a **migstorage** reference even though it was not being used. (BZ#2078459)

Velero pod log missing from downloaded logs

In previous releases, when downloading a compressed (.zip) folder for all logs, the velero pod was missing. (BZ#2076599)

Velero pod log missing from UI drop down

In previous releases, after a migration was performed, the velero pod log was not included in the logs provided in the dropdown list. (BZ#2076593)

Rsync options logs not visible in log-reader pod

In previous releases, when trying to set any valid or invalid rsync options in the **migrationcontroller**, the log-reader was not showing any logs regarding the invalid options or about the rsync command being used. (BZ#2079252)

Default CPU requests on Velero/Restic are too demanding and fail in certain environments

In previous releases, the default CPU requests on Velero/Restic were too demanding and fail in certain environments. Default CPU requests for Velero and Restic Pods are set to 500m. These values were high. (BZ#2088022)

2.2.18.2. Known issues

This release has the following known issues:

Updating the replication repository to a different storage provider type is not respected by the UI

After updating the replication repository to a different type and clicking **Update Repository**, it shows connection successful, but the UI is not updated with the correct details. When clicking on the **Edit** button again, it still shows the old replication repository information.

Furthermore, when trying to update the replication repository again, it still shows the old replication details. When selecting the new repository, it also shows all the information you entered previously and the **Update repository** is not enabled, as if there are no changes to be submitted. (BZ#2102020)

Migrations fails because the backup is not found

Migration fails at the restore stage because of initial backup has not been found. (BZ#2104874)

Update Cluster button is not enabled when updating Azure resource group

When updating the remote cluster, selecting the **Azure resource group** checkbox, and adding a resource group does not enable the **Update cluster** option. (BZ#2098594)

Error pop-up in UI on deleting migstorage resource

When creating a **backupStorage** credential secret in OpenShift Container Platform, if the **migstorage** is removed from the UI, a 404 error is returned and the underlying secret is not removed. (BZ#2100828)

Miganalytic resource displaying resource count as 0 in UI

After creating a migplan from backend, the Miganalytic resource displays the resource count as **0** in UI. (BZ#2102139)

Registry validation fails when two trailing slashes are added to the Exposed route host to image registry

After adding two trailing slashes, meaning //, to the exposed registry route, the MigCluster resource is showing the status as **connected**. When creating a migplan from backend with DIM, the plans move to the **unready** status. (BZ#2104864)

Service Account Token not visible while editing source cluster

When editing the source cluster that has been added and is in **Connected** state, in the UI, the service account token is not visible in the field. To save the wizard, you have to fetch the token again and provide details inside the field. (BZ#2097668)

2.2.19. Migration Toolkit for Containers 1.7.1 release notes

2.2.19.1. Resolved issues

There are no major resolved issues in this release.

2.2.19.2. Known issues

This release has the following known issues:

Incorrect DNS validation for destination namespace

MigPlan cannot be validated because the destination namespace starts with a non-alphabetic character. (BZ#2102231)

Cloud propagation phase in migration controller is not functioning due to missing labels on Velero pods

The Cloud propagation phase in the migration controller is not functioning due to missing labels on Velero pods. The **EnsureCloudSecretPropagated** phase in the migration controller waits until replication repository secrets are propagated on both sides. As this label is missing on Velero pods, the phase is not functioning as expected. (BZ#2088026)

Default CPU requests on Velero/Restic are too demanding when making scheduling fail in certain environments

Default CPU requests on Velero/Restic are too demanding when making scheduling fail in certain environments. Default CPU requests for Velero and Restic Pods are set to 500m. These values are high. The resources can be configured in DPA using the **podConfig** field for Velero and Restic. Migration operator should set CPU requests to a lower value, such as 100m, so that Velero and Restic pods can be scheduled in resource constrained environments Migration Toolkit for Containers (MTC) often operates in. (BZ#2088022)

Warning is displayed on persistentVolumes page after editing storage class conversion plan

A warning is displayed on the **persistentVolumes** page after editing the storage class conversion plan. When editing the existing migration plan, a warning is displayed on the UI **At least one PVC must be selected for Storage Class Conversion**. (BZ#2079549)

Velero pod log missing from downloaded logs

When downloading a compressed (.zip) folder for all logs, the velero pod is missing. (BZ#2076599)

Velero pod log missing from UI drop down

After a migration is performed, the velero pod log is not included in the logs provided in the dropdown list. (BZ#2076593)

2.2.20. Migration Toolkit for Containers 1.7.0 release notes

2.2.20.1. New features and enhancements

This release has the following new features and enhancements:

- The Migration Toolkit for Containers (MTC) Operator now depends upon the OpenShift API for Data Protection (OADP) Operator. When you install the MTC Operator, the Operator Lifecycle Manager (OLM) automatically installs the OADP Operator in the same namespace.

- You can migrate from a source cluster that is behind a firewall to a cloud-based destination cluster by establishing a network tunnel between the two clusters by using the **crane tunnel-api** command.
- Converting storage classes in the MTC web console: You can convert the storage class of a persistent volume (PV) by migrating it within the same cluster.

2.2.20.2. Known issues

This release has the following known issues:

- **MigPlan** custom resource does not display a warning when an AWS gp2 PVC has no available space. (BZ#1963927)
- Direct and indirect data transfers do not work if the destination storage is a PV that is dynamically provisioned by the AWS Elastic File System (EFS). This is due to limitations of the AWS EFS Container Storage Interface (CSI) driver. (BZ#2085097)
- Block storage for IBM Cloud must be in the same availability zone. See the IBM FAQ for block storage for virtual private cloud.
- MTC 1.7.6 cannot migrate cron jobs from source clusters that support **v1beta1** cron jobs to clusters of OpenShift Container Platform 4.12 and later, which do not support **v1beta1** cron jobs. (BZ#2149119)

2.3. MIGRATION TOOLKIT FOR CONTAINERS 1.6 RELEASE NOTES

The release notes for Migration Toolkit for Containers (MTC) describe new features and enhancements, deprecated features, and known issues.

The MTC enables you to migrate application workloads between OpenShift Container Platform clusters at the granularity of a namespace.

You can migrate from OpenShift Container Platform 3 to 4.14 and between OpenShift Container Platform 4 clusters.

MTC provides a web console and an API, based on Kubernetes custom resources, to help you control the migration and minimize application downtime.

For information on the support policy for MTC, see OpenShift Application and Cluster Migration Solutions, part of the *Red Hat OpenShift Container Platform Life Cycle Policy* .

2.3.1. Migration Toolkit for Containers 1.6 release notes

2.3.1.1. New features and enhancements

This release has the following new features and enhancements:

- State migration: You can perform repeatable, state-only migrations by selecting specific persistent volume claims (PVCs).
- "New operator version available" notification: The Clusters page of the MTC web console displays a notification when a new Migration Toolkit for Containers Operator is available.

2.3.1.2. Deprecated features

The following features are deprecated:

- MTC version 1.4 is no longer supported.

2.3.1.3. Known issues

This release has the following known issues:

- On OpenShift Container Platform 3.10, the **MigrationController** pod takes too long to restart. The Bugzilla report contains a workaround. (BZ#1986796)
- **Stage** pods fail during direct volume migration from a classic OpenShift Container Platform source cluster on IBM Cloud. The IBM block storage plugin does not allow the same volume to be mounted on multiple pods of the same node. As a result, the PVCs cannot be mounted on the Rsync pods and on the application pods simultaneously. To resolve this issue, stop the application pods before migration. (BZ#1887526)
- **MigPlan** custom resource does not display a warning when an AWS gp2 PVC has no available space. (BZ#1963927)
- Block storage for IBM Cloud must be in the same availability zone. See the IBM FAQ for block storage for virtual private cloud.

2.4. MIGRATION TOOLKIT FOR CONTAINERS 1.5 RELEASE NOTES

The release notes for Migration Toolkit for Containers (MTC) describe new features and enhancements, deprecated features, and known issues.

The MTC enables you to migrate application workloads between OpenShift Container Platform clusters at the granularity of a namespace.

You can migrate from OpenShift Container Platform 3 to 4.14 and between OpenShift Container Platform 4 clusters.

MTC provides a web console and an API, based on Kubernetes custom resources, to help you control the migration and minimize application downtime.

For information on the support policy for MTC, see OpenShift Application and Cluster Migration Solutions, part of the *Red Hat OpenShift Container Platform Life Cycle Policy* .

2.4.1. Migration Toolkit for Containers 1.5 release notes

2.4.1.1. New features and enhancements

This release has the following new features and enhancements:

- The **Migration resource** tree on the **Migration details** page of the web console has been enhanced with additional resources, Kubernetes events, and live status information for monitoring and debugging migrations.
- The web console can support hundreds of migration plans.

- A source namespace can be mapped to a different target namespace in a migration plan. Previously, the source namespace was mapped to a target namespace with the same name.
- Hook phases with status information are displayed in the web console during a migration.
- The number of Rsync retry attempts is displayed in the web console during direct volume migration.
- Persistent volume (PV) resizing can be enabled for direct volume migration to ensure that the target cluster does not run out of disk space.
- The threshold that triggers PV resizing is configurable. Previously, PV resizing occurred when the disk usage exceeded 97%.
- Velero has been updated to version 1.6, which provides numerous fixes and enhancements.
- Cached Kubernetes clients can be enabled to provide improved performance.

2.4.1.2. Deprecated features

The following features are deprecated:

- MTC versions 1.2 and 1.3 are no longer supported.
- The procedure for updating deprecated APIs has been removed from the troubleshooting section of the documentation because the **oc convert** command is deprecated.

2.4.1.3. Known issues

This release has the following known issues:

- Microsoft Azure storage is unavailable if you create more than 400 migration plans. The **MigStorage** custom resource displays the following message: **The request is being throttled as the limit has been reached for operation type**. (BZ#1977226)
- If a migration fails, the migration plan does not retain custom persistent volume (PV) settings for quiesced pods. You must manually roll back the migration, delete the migration plan, and create a new migration plan with your PV settings. (BZ#1784899)
- PV resizing does not work as expected for AWS gp2 storage unless the **pv_resizing_threshold** is 42% or greater. (BZ#1973148)
- PV resizing does not work with OpenShift Container Platform 3.7 and 3.9 source clusters in the following scenarios:
  - The application was installed after MTC was installed.
  - An application pod was rescheduled on a different node after MTC was installed. OpenShift Container Platform 3.7 and 3.9 do not support the Mount Propagation feature that enables Velero to mount PVs automatically in the **Restic** pod. The **MigAnalytic** custom resource (CR) fails to collect PV data from the **Restic** pod and reports the resources as **0**. The **MigPlan** CR displays a status similar to the following:

    Example output

    ```
    status:
    ```

```
  conditions:
  - category: Warn
    lastTransitionTime: 2021-07-15T04:11:44Z
    message: Failed gathering extended PV usage information for PVs [nginx-logs nginx-
html], please see MigAnalytic openshift-migration/ocp-24706-basicvolmig-migplan-
1626319591-szwd6 for details
    reason: FailedRunningDf
    status: "True"
    type: ExtendedPVAnalysisFailed
```

To enable PV resizing, you can manually restart the Restic daemonset on the source cluster or restart the **Restic** pods on the same nodes as the application. If you do not restart Restic, you can run the direct volume migration without PV resizing. (BZ#1982729)

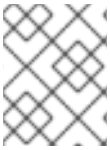
2.4.1.4. Technical changes

This release has the following technical changes:

- The legacy Migration Toolkit for Containers Operator version 1.5.1 is installed manually on OpenShift Container Platform versions 3.7 to 4.5.
- The Migration Toolkit for Containers Operator version 1.5.1 is installed on OpenShift Container Platform versions 4.6 and later by using the Operator Lifecycle Manager.

CHAPTER 3. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS

You can install the Migration Toolkit for Containers (MTC) on OpenShift Container Platform 4.



NOTE

To install MTC on OpenShift Container Platform 3, see Installing the legacy Migration Toolkit for Containers Operator on OpenShift Container Platform 3.

By default, the MTC web console and the **Migration Controller** pod run on the target cluster. You can configure the **Migration Controller** custom resource manifest to run the MTC web console and the **Migration Controller** pod on a remote cluster.

After you have installed MTC, you must configure an object storage to use as a replication repository.

To uninstall MTC, see Uninstalling MTC and deleting resources.

3.1. COMPATIBILITY GUIDELINES

You must install the Migration Toolkit for Containers (MTC) Operator that is compatible with your OpenShift Container Platform version.

Definitions

legacy platform
: OpenShift Container Platform 4.5 and earlier.

modern platform
: OpenShift Container Platform 4.6 and later.

legacy operator
: The MTC Operator designed for legacy platforms.

modern operator
: The MTC Operator designed for modern platforms.

control cluster
: The cluster that runs the MTC controller and GUI.

remote cluster
: A source or destination cluster for a migration that runs Velero. The Control Cluster communicates with Remote clusters via the Velero API to drive migrations.

You must use the compatible MTC version for migrating your OpenShift Container Platform clusters. For the migration to succeed both your source cluster and the destination cluster must use the same version of MTC.

MTC 1.7 supports migrations from OpenShift Container Platform 3.11 to 4.9.

MTC 1.8 only supports migrations from OpenShift Container Platform 4.10 and later.

Table 3.1. MTC compatibility: Migrating from a legacy or a modern platform

Details	OpenShift Container Platform 3.11	OpenShift Container Platform 4.0 to 4.5	OpenShift Container Platform 4.6 to 4.9	OpenShift Container Platform 4.10 or later
Stable MTC version	MTC v.1.7.*z*	MTC v.1.7.*z*	MTC v.1.7.*z*	MTC v.1.8.*z*
Installation		Legacy MTC v.1.7.*z* operator: Install manually with the **operator.yml** file. [**IMPORTANT**] This cluster cannot be the control cluster.	Install with OLM, release channel **release-v1.7**	Install with OLM, release channel **release-v1.8**

Edge cases exist in which network restrictions prevent modern clusters from connecting to other clusters involved in the migration. For example, when migrating from an OpenShift Container Platform 3.11 cluster on premises to a modern OpenShift Container Platform cluster in the cloud, where the modern cluster cannot connect to the OpenShift Container Platform 3.11 cluster.

With MTC v.1.7.*z*, if one of the remote clusters is unable to communicate with the control cluster because of network restrictions, use the **crane tunnel-api** command.

With the stable MTC release, although you should always designate the most modern cluster as the control cluster, in this specific case it is possible to designate the legacy cluster as the control cluster and push workloads to the remote cluster.

3.2. INSTALLING THE LEGACY MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSHIFT CONTAINER PLATFORM 4.2 TO 4.5

You can install the legacy Migration Toolkit for Containers Operator manually on OpenShift Container Platform versions 4.2 to 4.5.

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges on all clusters.
- You must have access to **registry.redhat.io**.
- You must have **podman** installed.

Procedure

1. Log in to **registry.redhat.io** with your Red Hat Customer Portal credentials:

```
$ podman login registry.redhat.io
```

2. Download the **operator.yml** file by entering the following command:

```
podman cp $(podman create registry.redhat.io/rhmtc/openshift-migration-legacy-rhel8-operator:v1.7):/operator.yml ./
```

3. Download the **controller.yml** file by entering the following command:

```
podman cp $(podman create registry.redhat.io/rhmtc/openshift-migration-legacy-rhel8-operator:v1.7):/controller.yml ./
```

4. Log in to your OpenShift Container Platform source cluster.
5. Verify that the cluster can authenticate with **registry.redhat.io**:

```
$ oc run test --image registry.redhat.io/ubi9 --command sleep infinity
```

6. Create the Migration Toolkit for Containers Operator object:

```
$ oc create -f operator.yml
```

Example output

```
namespace/openshift-migration created
rolebinding.rbac.authorization.k8s.io/system:deployers created
serviceaccount/migration-operator created
customresourcedefinition.apiextensions.k8s.io/migrationcontrollers.migration.openshift.io created
role.rbac.authorization.k8s.io/migration-operator created
rolebinding.rbac.authorization.k8s.io/migration-operator created
clusterrolebinding.rbac.authorization.k8s.io/migration-operator created
deployment.apps/migration-operator created
Error from server (AlreadyExists): error when creating "./operator.yml":
rolebindings.rbac.authorization.k8s.io "system:image-builders" already exists (1)
Error from server (AlreadyExists): error when creating "./operator.yml":
rolebindings.rbac.authorization.k8s.io "system:image-pullers" already exists
```

(1) You can ignore **Error from server (AlreadyExists)** messages. They are caused by the Migration Toolkit for Containers Operator creating resources for earlier versions of OpenShift Container Platform 4 that are provided in later releases.

7. Create the **MigrationController** object:

```
$ oc create -f controller.yml
```

8. Verify that the MTC pods are running:

```
$ oc get pods -n openshift-migration
```

3.3. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSHIFT CONTAINER PLATFORM 4.14

You install the Migration Toolkit for Containers Operator on OpenShift Container Platform 4.14 by using the Operator Lifecycle Manager.

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges on all clusters.

Procedure

1. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
2. Use the **Filter by keyword** field to find the **Migration Toolkit for Containers Operator**.
3. Select the **Migration Toolkit for Containers Operator** and click **Install**.
4. Click **Install**.
   On the **Installed Operators** page, the **Migration Toolkit for Containers Operator** appears in the **openshift-migration** project with the status **Succeeded**.
5. Click **Migration Toolkit for Containers Operator**.
6. Under **Provided APIs**, locate the **Migration Controller** tile, and click **Create Instance**.
7. Click **Create**.
8. Click **Workloads → Pods** to verify that the MTC pods are running.

3.4. PROXY CONFIGURATION

For OpenShift Container Platform 4.1 and earlier versions, you must configure proxies in the **MigrationController** custom resource (CR) manifest after you install the Migration Toolkit for Containers Operator because these versions do not support a cluster-wide **proxy** object.

For OpenShift Container Platform 4.2 to 4.14, the MTC inherits the cluster-wide proxy settings. You can change the proxy parameters if you want to override the cluster-wide proxy settings.

3.4.1. Direct volume migration

Direct Volume Migration (DVM) was introduced in MTC 1.4.2. DVM supports only one proxy. The source cluster cannot access the route of the target cluster if the target cluster is also behind a proxy.

If you want to perform a DVM from a source cluster behind a proxy, you must configure a TCP proxy that works at the transport layer and forwards the SSL connections transparently without decrypting and re-encrypting them with their own SSL certificates. A Stunnel proxy is an example of such a proxy.

3.4.1.1. TCP proxy setup for DVM

You can set up a direct connection between the source and the target cluster through a TCP proxy and configure the **stunnel_tcp_proxy** variable in the **MigrationController** CR to use the proxy:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigrationController
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  stunnel_tcp_proxy: http://username:password@ip:port
```

Direct volume migration (DVM) supports only basic authentication for the proxy. Moreover, DVM works only from behind proxies that can tunnel a TCP connection transparently. HTTP/HTTPS proxies in man-in-the-middle mode do not work. The existing cluster-wide proxies might not support this behavior. As a result, the proxy settings for DVM are intentionally kept different from the usual proxy configuration in MTC.

3.4.1.2. Why use a TCP proxy instead of an HTTP/HTTPS proxy?

You can enable DVM by running Rsync between the source and the target cluster over an OpenShift route. Traffic is encrypted using Stunnel, a TCP proxy. The Stunnel running on the source cluster initiates a TLS connection with the target Stunnel and transfers data over an encrypted channel.

Cluster-wide HTTP/HTTPS proxies in OpenShift are usually configured in man-in-the-middle mode where they negotiate their own TLS session with the outside servers. However, this does not work with Stunnel. Stunnel requires that its TLS session be untouched by the proxy, essentially making the proxy a transparent tunnel which simply forwards the TCP connection as-is. Therefore, you must use a TCP proxy.

3.4.1.3. Known issue

Migration fails with error Upgrade request required

The migration Controller uses the SPDY protocol to execute commands within remote pods. If the remote cluster is behind a proxy or a firewall that does not support the SPDY protocol, the migration controller fails to execute remote commands. The migration fails with the error message **Upgrade request required**. Workaround: Use a proxy that supports the SPDY protocol.

In addition to supporting the SPDY protocol, the proxy or firewall also must pass the **Upgrade** HTTP header to the API server. The client uses this header to open a websocket connection with the API server. If the **Upgrade** header is blocked by the proxy or firewall, the migration fails with the error message **Upgrade request required**. Workaround: Ensure that the proxy forwards the **Upgrade** header.

3.4.2. Tuning network policies for migrations

OpenShift supports restricting traffic to or from pods using *NetworkPolicy* or *EgressFirewalls* based on the network plugin used by the cluster. If any of the source namespaces involved in a migration use such mechanisms to restrict network traffic to pods, the restrictions might inadvertently stop traffic to Rsync pods during migration.

Rsync pods running on both the source and the target clusters must connect to each other over an OpenShift Route. Existing *NetworkPolicy* or *EgressNetworkPolicy* objects can be configured to automatically exempt Rsync pods from these traffic restrictions.

3.4.2.1. NetworkPolicy configuration

3.4.2.1.1. Egress traffic from Rsync pods

You can use the unique labels of Rsync pods to allow egress traffic to pass from them if the **NetworkPolicy** configuration in the source or destination namespaces blocks this type of traffic. The following policy allows **all** egress traffic from Rsync pods in the namespace:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
```

```
  name: allow-all-egress-from-rsync-pods
spec:
  podSelector:
    matchLabels:
      owner: directvolumemigration
      app: directvolumemigration-rsync-transfer
  egress:
  - {}
  policyTypes:
  - Egress
```

3.4.2.1.2. Ingress traffic to Rsync pods

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-egress-from-rsync-pods
spec:
  podSelector:
    matchLabels:
      owner: directvolumemigration
      app: directvolumemigration-rsync-transfer
  ingress:
  - {}
  policyTypes:
  - Ingress
```

3.4.2.2. EgressNetworkPolicy configuration

The **EgressNetworkPolicy** object or *Egress Firewalls* are OpenShift constructs designed to block egress traffic leaving the cluster.

Unlike the **NetworkPolicy** object, the Egress Firewall works at a project level because it applies to all pods in the namespace. Therefore, the unique labels of Rsync pods do not exempt only Rsync pods from the restrictions. However, you can add the CIDR ranges of the source or target cluster to the *Allow* rule of the policy so that a direct connection can be setup between two clusters.

Based on which cluster the Egress Firewall is present in, you can add the CIDR range of the other cluster to allow egress traffic between the two:

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: test-egress-policy
  namespace: <namespace>
spec:
  egress:
  - to:
      cidrSelector: <cidr_of_source_or_target_cluster>
    type: Deny
```

3.4.2.3. Choosing alternate endpoints for data transfer

By default, DVM uses an OpenShift Container Platform route as an endpoint to transfer PV data to destination clusters. You can choose another type of supported endpoint, if cluster topologies allow.

For each cluster, you can configure an endpoint by setting the **rsync_endpoint_type** variable on the appropriate **destination** cluster in your **MigrationController** CR:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigrationController
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  rsync_endpoint_type: [NodePort|ClusterIP|Route]
```

3.4.2.4. Configuring supplemental groups for Rsync pods

When your PVCs use a shared storage, you can configure the access to that storage by adding supplemental groups to Rsync pod definitions in order for the pods to allow access:

Table 3.2. Supplementary groups for Rsync pods

Variable	Type	Default	Description
src_supplemental_groups	string	Not set	Comma-separated list of supplemental groups for source Rsync pods
target_supplemental_groups	string	Not set	Comma-separated list of supplemental groups for target Rsync pods

Example usage

The **MigrationController** CR can be updated to set values for these supplemental groups:

```
spec:
  src_supplemental_groups: "1000,2000"
  target_supplemental_groups: "2000,3000"
```

3.4.3. Configuring proxies

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges on all clusters.

Procedure

1. Get the **MigrationController** CR manifest:

```
$ oc get migrationcontroller <migration_controller> -n openshift-migration
```

2. Update the proxy parameters:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigrationController
metadata:
  name: <migration_controller>
  namespace: openshift-migration
...
spec:
  stunnel_tcp_proxy: http://<username>:<password>@<ip>:<port> 1
  noProxy: example.com 2
```

1. Stunnel proxy URL for direct volume migration.
2. Comma-separated list of destination domain names, domains, IP addresses, or other network CIDRs to exclude proxying.

   Preface a domain with **.** to match subdomains only. For example, **.y.com** matches **x.y.com**, but not **y.com**. Use ***** to bypass proxy for all destinations. If you scale up workers that are not included in the network defined by the **networking.machineNetwork[].cidr** field from the installation configuration, you must add them to this list to prevent connection issues.

   This field is ignored if neither the **httpProxy** nor the **httpsProxy** field is set.

3. Save the manifest as **migration-controller.yaml**.

4. Apply the updated manifest:

```
$ oc replace -f migration-controller.yaml -n openshift-migration
```

For more information, see Configuring the cluster-wide proxy.

3.4.4. Running Rsync as either root or non-root

OpenShift Container Platform environments have the **PodSecurityAdmission** controller enabled by default. This controller requires cluster administrators to enforce Pod Security Standards by means of namespace labels. All workloads in the cluster are expected to run one of the following Pod Security Standard levels: **privileged**, **baseline** or **restricted**. Every cluster has its own default policy set.

To guarantee successful data transfer in all environments, Migration Toolkit for Containers (MTC) 1.7.5 introduced changes in Rsync pods, including running Rsync pods as non-root user by default. This ensures that data transfer is possible even for workloads that do not necessarily require higher privileges. This change was made because it is best to run workloads with the lowest level of privileges possible.

3.4.4.1. Manually overriding default non-root operation for data transfer

Although running Rsync pods as non-root user works in most cases, data transfer might fail when you run workloads as root user on the source side. MTC provides two ways to manually override default non-root operation for data transfer:

- Configure all migrations to run an Rsync pod as root on the destination cluster for all migrations.
- Run an Rsync pod as root on the destination cluster per migration.

In both cases, you must set the following labels on the source side of any namespaces that are running workloads with higher privileges before migration: **enforce**, **audit**, and **warn.**

To learn more about Pod Security Admission and setting values for labels, see Controlling pod security admission synchronization.

3.4.4.2. Configuring the MigrationController CR as root or non-root for all migrations

By default, Rsync runs as non-root.

On the destination cluster, you can configure the **MigrationController** CR to run Rsync as root.

Procedure

- Configure the **MigrationController** CR as follows:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigrationController
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  migration_rsync_privileged: true
```

This configuration will apply to all future migrations.

3.4.4.3. Configuring the MigMigration CR as root or non-root per migration

On the destination cluster, you can configure the **MigMigration** CR to run Rsync as root or non-root, with the following non-root options:

- As a specific user ID (UID)
- As a specific group ID (GID)

Procedure

1. To run Rsync as root, configure the **MigMigration** CR according to this example:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigMigration
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  runAsRoot: true
```

2. To run Rsync as a specific User ID (UID) or as a specific Group ID (GID), configure the **MigMigration** CR according to this example:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigMigration
```

```
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  runAsUser: 10010001
  runAsGroup: 3
```

3.5. CONFIGURING A REPLICATION REPOSITORY

You must configure an object storage to use as a replication repository. The Migration Toolkit for Containers (MTC) copies data from the source cluster to the replication repository, and then from the replication repository to the target cluster.

MTC supports the file system and snapshot data copy methods for migrating data from the source cluster to the target cluster. Select a method that is suited for your environment and is supported by your storage provider.

MTC supports the following storage providers:

- Multicloud Object Gateway
- Amazon Web Services S3
- Google Cloud
- Microsoft Azure Blob
- Generic S3 object storage, for example, Minio or Ceph S3

3.5.1. Prerequisites

- All clusters must have uninterrupted network access to the replication repository.
- If you use a proxy server with an internally hosted replication repository, you must ensure that the proxy allows access to the replication repository.

3.5.2. Retrieving Multicloud Object Gateway credentials

You must retrieve the Multicloud Object Gateway (MCG) credentials and S3 endpoint, which you need to configure MCG as a replication repository for the Migration Toolkit for Containers (MTC).

You must retrieve the Multicloud Object Gateway (MCG) credentials, which you need to create a **Secret** custom resource (CR) for MTC.

NOTE

Although the MCG Operator is deprecated, the MCG plugin is still available for OpenShift Data Foundation. To download the plugin, browse to Download Red Hat OpenShift Data Foundation and download the appropriate MCG plugin for your operating system.

Prerequisites

- You must deploy OpenShift Data Foundation by using the appropriate Red Hat OpenShift Data Foundation deployment guide.

Procedure

- Obtain the S3 endpoint, **AWS_ACCESS_KEY_ID**, and the **AWS_SECRET_ACCESS_KEY** value by running the **oc describe** command for the **NooBaa** CR.
  You use these credentials to add MCG as a replication repository.

3.5.3. Configuring Amazon Web Services

Configure Amazon Web Services (AWS) S3 storage and Identity and Access Management (IAM) credentials for backup storage with Migration Toolkit for Containers (MTC). This provides the necessary permissions and storage infrastructure for data protection operations.

Prerequisites

- You must have the AWS CLI installed.
- The AWS S3 storage bucket must be accessible to the source and target clusters.
- If you are using the snapshot copy method:
  - You must have access to EC2 Elastic Block Storage (EBS).
  - The source and target clusters must be in the same region.
  - The source and target clusters must have the same storage class.
  - The storage class must be compatible with snapshots.

Procedure

1. Set the **BUCKET** variable:

```
$ BUCKET=<your_bucket>
```

2. Set the **REGION** variable:

```
$ REGION=<your_region>
```

3. Create an AWS S3 bucket:

```
$ aws s3api create-bucket \
    --bucket $BUCKET \
    --region $REGION \
    --create-bucket-configuration LocationConstraint=$REGION
```

where:

LocationConstraint

Specifies the bucket configuration location constraint. **us-east-1** does not support **LocationConstraint**. If your region is **us-east-1**, omit **--create-bucket-configuration LocationConstraint=$REGION**.

4. Create an IAM user:

```
$ aws iam create-user --user-name velero
```

where:

velero

Specifies the user name. If you want to use Velero to back up multiple clusters with multiple S3 buckets, create a unique user name for each cluster.

5. Create a **velero-policy.json** file:

```
$ cat > velero-policy.json <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVolumes",
                "ec2:DescribeSnapshots",
                "ec2:CreateTags",
                "ec2:CreateVolume",
                "ec2:CreateSnapshot",
                "ec2:DeleteSnapshot"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:DeleteObject",
                "s3:PutObject",
                "s3:AbortMultipartUpload",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:s3:::${BUCKET}/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:ListBucketMultipartUploads"
            ],
            "Resource": [
                "arn:aws:s3:::${BUCKET}"
            ]
        }
    ]
}
EOF
```

6. Attach the policies to give the **velero** user the minimum necessary permissions:

```
$ aws iam put-user-policy \
  --user-name velero \
  --policy-name velero \
  --policy-document file://velero-policy.json
```

7. Create an access key for the **velero** user:

```
$ aws iam create-access-key --user-name velero
```

```
{
  "AccessKey": {
        "UserName": "velero",
        "Status": "Active",
        "CreateDate": "2017-07-31T22:24:41.576Z",
        "SecretAccessKey": <AWS_SECRET_ACCESS_KEY>,
        "AccessKeyId": <AWS_ACCESS_KEY_ID>
  }
}
```

Record the **AWS_SECRET_ACCESS_KEY** and the **AWS_ACCESS_KEY_ID**. You use the credentials to add AWS as a replication repository.

3.5.4. Configuring Google Cloud

You configure a Google Cloud storage bucket as a replication repository for the Migration Toolkit for Containers (MTC).

Prerequisites

- You must have the **gcloud** and **gsutil** CLI tools installed. See the Google cloud documentation for details.
- The Google Cloud storage bucket must be accessible to the source and target clusters.
- If you are using the snapshot copy method:
  - The source and target clusters must be in the same region.
  - The source and target clusters must have the same storage class.
  - The storage class must be compatible with snapshots.

Procedure

1. Log in to Google Cloud:

```
$ gcloud auth login
```

2. Set the **BUCKET** variable:

```
$ BUCKET=<bucket> 1
```

1 Specify your bucket name.

3. Create the storage bucket:

```
$ gsutil mb gs://$BUCKET/
```

4. Set the **PROJECT_ID** variable to your active project:

```
$ PROJECT_ID=$(gcloud config get-value project)
```

5. Create a service account:

```
$ gcloud iam service-accounts create velero \
    --display-name "Velero service account"
```

6. List your service accounts:

```
$ gcloud iam service-accounts list
```

7. Set the **SERVICE_ACCOUNT_EMAIL** variable to match its **email** value:

```
$ SERVICE_ACCOUNT_EMAIL=$(gcloud iam service-accounts list \
    --filter="displayName:Velero service account" \
    --format 'value(email)')
```

8. Attach the policies to give the **velero** user the minimum necessary permissions:

```
$ ROLE_PERMISSIONS=(
    compute.disks.get
    compute.disks.create
    compute.disks.createSnapshot
    compute.snapshots.get
    compute.snapshots.create
    compute.snapshots.useReadOnly
    compute.snapshots.delete
    compute.zones.get
    storage.objects.create
    storage.objects.delete
    storage.objects.get
    storage.objects.list
    iam.serviceAccounts.signBlob
)
```

9. Create the **velero.server** custom role:

```
$ gcloud iam roles create velero.server \
    --project $PROJECT_ID \
    --title "Velero Server" \
    --permissions "$(IFS=","; echo "${ROLE_PERMISSIONS[*]}")"
```

10. Add IAM policy binding to the project:

```
$ gcloud projects add-iam-policy-binding $PROJECT_ID \
    --member serviceAccount:$SERVICE_ACCOUNT_EMAIL \
    --role projects/$PROJECT_ID/roles/velero.server
```

11. Update the IAM service account:

```
$ gsutil iam ch serviceAccount:$SERVICE_ACCOUNT_EMAIL:objectAdmin gs://${BUCKET}
```

12. Save the IAM service account keys to the **credentials-velero** file in the current directory:

```
$ gcloud iam service-accounts keys create credentials-velero \
    --iam-account $SERVICE_ACCOUNT_EMAIL
```

You use the **credentials-velero** file to add Google Cloud as a replication repository.

3.5.5. Configuring Microsoft Azure

You configure a Microsoft Azure Blob storage container as a replication repository for the Migration Toolkit for Containers (MTC).

Prerequisites

- You must have the Azure CLI installed.
- The Azure Blob storage container must be accessible to the source and target clusters.
- If you are using the snapshot copy method:
  - The source and target clusters must be in the same region.
  - The source and target clusters must have the same storage class.
  - The storage class must be compatible with snapshots.

Procedure

1. Log in to Azure:

```
$ az login
```

2. Set the **AZURE_RESOURCE_GROUP** variable:

```
$ AZURE_RESOURCE_GROUP=Velero_Backups
```

3. Create an Azure resource group:

```
$ az group create -n $AZURE_RESOURCE_GROUP --location CentralUS (1)
```

(1) Specify your location.

4. Set the **AZURE_STORAGE_ACCOUNT_ID** variable:

```
$ AZURE_STORAGE_ACCOUNT_ID="velero$(uuidgen | cut -d '-' -f5 | tr '[A-Z]' '[a-z]')"
```

5. Create an Azure storage account:

```
$ az storage account create \
   --name $AZURE_STORAGE_ACCOUNT_ID \
   --resource-group $AZURE_RESOURCE_GROUP \
   --sku Standard_GRS \
   --encryption-services blob \
   --https-only true \
   --kind BlobStorage \
   --access-tier Hot
```

6. Set the **BLOB_CONTAINER** variable:

```
$ BLOB_CONTAINER=velero
```

7. Create an Azure Blob storage container:

```
$ az storage container create \
  -n $BLOB_CONTAINER \
  --public-access off \
  --account-name $AZURE_STORAGE_ACCOUNT_ID
```

8. Create a service principal and credentials for **velero**:

```
$ AZURE_SUBSCRIPTION_ID=`az account list --query '[?isDefault].id' -o tsv`
  AZURE_TENANT_ID=`az account list --query '[?isDefault].tenantId' -o tsv`
```

9. Create a service principal with the **Contributor** role, assigning a specific **--role** and **--scopes**:

```
$ AZURE_CLIENT_SECRET=`az ad sp create-for-rbac --name "velero" \
                                    --role "Contributor" \
                                    --query 'password' -o tsv \
                                    --scopes
/subscriptions/$AZURE_SUBSCRIPTION_ID/resourceGroups/$AZURE_RESOURCE_GROU
P`
```

The CLI generates a password for you. Ensure you capture the password.

10. After creating the service principal, obtain the client id.

```
$ AZURE_CLIENT_ID=`az ad app credential list --id <your_app_id>`
```

NOTE

For this to be successful, you must know your Azure application ID.

11. Save the service principal credentials in the **credentials-velero** file:

```
$ cat << EOF > ./credentials-velero
AZURE_SUBSCRIPTION_ID=${AZURE_SUBSCRIPTION_ID}
```

```
AZURE_TENANT_ID=${AZURE_TENANT_ID}
AZURE_CLIENT_ID=${AZURE_CLIENT_ID}
AZURE_CLIENT_SECRET=${AZURE_CLIENT_SECRET}
AZURE_RESOURCE_GROUP=${AZURE_RESOURCE_GROUP}
AZURE_CLOUD_NAME=AzurePublicCloud
EOF
```

You use the **credentials-velero** file to add Azure as a replication repository.

3.5.6. Additional resources

- MTC workflow
- About data copy methods
- Adding a replication repository to the MTC web console

3.6. UNINSTALLING MTC AND DELETING RESOURCES

You can uninstall the Migration Toolkit for Containers (MTC) and delete its resources to clean up the cluster.

NOTE

Deleting the **velero** CRDs removes Velero from the cluster.

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges.

Procedure

1. Delete the **MigrationController** custom resource (CR) on all clusters:

```
$ oc delete migrationcontroller <migration_controller>
```

2. Uninstall the Migration Toolkit for Containers Operator on OpenShift Container Platform 4 by using the Operator Lifecycle Manager.
3. Delete cluster-scoped resources on all clusters by running the following commands:
   - **migration** custom resource definitions (CRDs):

```
$ oc delete $(oc get crds -o name | grep 'migration.openshift.io')
```

   - **velero** CRDs:

```
$ oc delete $(oc get crds -o name | grep 'velero')
```

   - **migration** cluster roles:

```
$ oc delete $(oc get clusterroles -o name | grep 'migration.openshift.io')
```

- **migration-operator** cluster role:

```
$ oc delete clusterrole migration-operator
```

- **velero** cluster roles:

```
$ oc delete $(oc get clusterroles -o name | grep 'velero')
```

- **migration** cluster role bindings:

```
$ oc delete $(oc get clusterrolebindings -o name | grep 'migration.openshift.io')
```

- **migration-operator** cluster role bindings:

```
$ oc delete clusterrolebindings migration-operator
```

- **velero** cluster role bindings:

```
$ oc delete $(oc get clusterrolebindings -o name | grep 'velero')
```

CHAPTER 4. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS IN A RESTRICTED NETWORK ENVIRONMENT

You can install the Migration Toolkit for Containers (MTC) on OpenShift Container Platform 4 in a restricted network environment by performing the following procedures:

1. Create a mirrored Operator catalog.
   This process creates a **mapping.txt** file, which contains the mapping between the **registry.redhat.io** image and your mirror registry image. The **mapping.txt** file is required for installing the *legacy* Migration Toolkit for Containers Operator on an OpenShift Container Platform 4.2 to 4.5 source cluster.

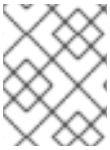
2. Install the Migration Toolkit for Containers Operator on the OpenShift Container Platform 4.14 target cluster by using Operator Lifecycle Manager.
   By default, the MTC web console and the **Migration Controller** pod run on the target cluster. You can configure the **Migration Controller** custom resource manifest to run the MTC web console and the **Migration Controller** pod on a remote cluster.

3. Install the Migration Toolkit for Containers Operator on the source cluster:

   - OpenShift Container Platform 4.6 or later: Install the Migration Toolkit for Containers Operator by using Operator Lifecycle Manager.

   - OpenShift Container Platform 4.2 to 4.5: Install the legacy Migration Toolkit for Containers Operator from the command-line interface.

4. Configure object storage to use as a replication repository.



NOTE

To install MTC on OpenShift Container Platform 3, see Installing the legacy Migration Toolkit for Containers Operator on OpenShift Container Platform 3.

To uninstall MTC, see Uninstalling MTC and deleting resources.

4.1. COMPATIBILITY GUIDELINES

You must install the Migration Toolkit for Containers (MTC) Operator that is compatible with your OpenShift Container Platform version.

Definitions

legacy platform
OpenShift Container Platform 4.5 and earlier.

modern platform
OpenShift Container Platform 4.6 and later.

legacy operator
The MTC Operator designed for legacy platforms.

modern operator
The MTC Operator designed for modern platforms.

control cluster

The cluster that runs the MTC controller and GUI.

remote cluster

A source or destination cluster for a migration that runs Velero. The Control Cluster communicates with Remote clusters via the Velero API to drive migrations.

You must use the compatible MTC version for migrating your OpenShift Container Platform clusters. For the migration to succeed both your source cluster and the destination cluster must use the same version of MTC.

MTC 1.7 supports migrations from OpenShift Container Platform 3.11 to 4.9.

MTC 1.8 only supports migrations from OpenShift Container Platform 4.10 and later.

Table 4.1. MTC compatibility: Migrating from a legacy or a modern platform

Details	OpenShift Container Platform 3.11	OpenShift Container Platform 4.0 to 4.5	OpenShift Container Platform 4.6 to 4.9	OpenShift Container Platform 4.10 or later
Stable MTC version	MTC v.1.7.*z*	MTC v.1.7.*z*	MTC v.1.7.*z*	MTC v.1.8.*z*
Installation		Legacy MTC v.1.7.*z* operator: Install manually with the **operator.yml** file. [**IMPORTANT**] This cluster cannot be the control cluster.	Install with OLM, release channel **release-v1.7**	Install with OLM, release channel **release-v1.8**

Edge cases exist in which network restrictions prevent modern clusters from connecting to other clusters involved in the migration. For example, when migrating from an OpenShift Container Platform 3.11 cluster on premises to a modern OpenShift Container Platform cluster in the cloud, where the modern cluster cannot connect to the OpenShift Container Platform 3.11 cluster.

With MTC v.1.7.*z*, if one of the remote clusters is unable to communicate with the control cluster because of network restrictions, use the **crane tunnel-api** command.

With the stable MTC release, although you should always designate the most modern cluster as the control cluster, in this specific case it is possible to designate the legacy cluster as the control cluster and push workloads to the remote cluster.

4.2. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSHIFT CONTAINER PLATFORM 4.14

You install the Migration Toolkit for Containers Operator on OpenShift Container Platform 4.14 by using the Operator Lifecycle Manager.

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges on all clusters.
- You must create an Operator catalog from a mirror image in a local registry.

Procedure

1. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
2. Use the **Filter by keyword** field to find the **Migration Toolkit for Containers Operator**.
3. Select the **Migration Toolkit for Containers Operator** and click **Install**.
4. Click **Install**.
   On the **Installed Operators** page, the **Migration Toolkit for Containers Operator** appears in the **openshift-migration** project with the status **Succeeded**.
5. Click **Migration Toolkit for Containers Operator**.
6. Under **Provided APIs**, locate the **Migration Controller** tile, and click **Create Instance**.
7. Click **Create**.
8. Click **Workloads → Pods** to verify that the MTC pods are running.

4.3. INSTALLING THE LEGACY MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSHIFT CONTAINER PLATFORM 4.2 TO 4.5

You can install the legacy Migration Toolkit for Containers Operator manually on OpenShift Container Platform versions 4.2 to 4.5.

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges on all clusters.
- You must have access to **registry.redhat.io**.
- You must have **podman** installed.
- You must have a Linux workstation with network access in order to download files from **registry.redhat.io**.
- You must create a mirror image of the Operator catalog.
- You must install the Migration Toolkit for Containers Operator from the mirrored Operator catalog on OpenShift Container Platform 4.14.

Procedure

1. Log in to **registry.redhat.io** with your Red Hat Customer Portal credentials:

   ```
   $ podman login registry.redhat.io
   ```

2. Download the **operator.yml** file by entering the following command:

```
podman cp $(podman create registry.redhat.io/rhmtc/openshift-migration-legacy-rhel8-operator:v1.7):/operator.yml ./
```

3. Download the **controller.yml** file by entering the following command:

```
podman cp $(podman create registry.redhat.io/rhmtc/openshift-migration-legacy-rhel8-operator:v1.7):/controller.yml ./
```

4. Obtain the Operator image mapping by running the following command:

```
$ grep openshift-migration-legacy-rhel8-operator ./mapping.txt | grep rhmtc
```

The **mapping.txt** file was created when you mirrored the Operator catalog. The output shows the mapping between the **registry.redhat.io** image and your mirror registry image.

Example output

```
registry.redhat.io/rhmtc/openshift-migration-legacy-rhel8-operator@sha256:468a6126f73b1ee12085ca53a312d1f96ef5a2ca03442bcb63724af5e2614e8a=<registry.apps.example.com>/rhmtc/openshift-migration-legacy-rhel8-operator
```

5. Update the **image** values for the **ansible** and **operator** containers and the **REGISTRY** value in the **operator.yml** file:

```
containers:
  - name: ansible
    image: <registry.apps.example.com>/rhmtc/openshift-migration-legacy-rhel8-operator@sha256:
<468a6126f73b1ee12085ca53a312d1f96ef5a2ca03442bcb63724af5e2614e8a> 1
...
  - name: operator
    image: <registry.apps.example.com>/rhmtc/openshift-migration-legacy-rhel8-operator@sha256:
<468a6126f73b1ee12085ca53a312d1f96ef5a2ca03442bcb63724af5e2614e8a> 2
...
    env:
    - name: REGISTRY
      value: <registry.apps.example.com> 3
```

1 2 Specify your mirror registry and the **sha256** value of the Operator image.

3 Specify your mirror registry.

6. Log in to your OpenShift Container Platform source cluster.

7. Create the Migration Toolkit for Containers Operator object:

```
$ oc create -f operator.yml
```

Example output

```
namespace/openshift-migration created
```

```
rolebinding.rbac.authorization.k8s.io/system:deployers created
serviceaccount/migration-operator created
customresourcedefinition.apiextensions.k8s.io/migrationcontrollers.migration.openshift.io
created
role.rbac.authorization.k8s.io/migration-operator created
rolebinding.rbac.authorization.k8s.io/migration-operator created
clusterrolebinding.rbac.authorization.k8s.io/migration-operator created
deployment.apps/migration-operator created
Error from server (AlreadyExists): error when creating "./operator.yml":
rolebindings.rbac.authorization.k8s.io "system:image-builders" already exists 1
Error from server (AlreadyExists): error when creating "./operator.yml":
rolebindings.rbac.authorization.k8s.io "system:image-pullers" already exists
```

1 You can ignore **Error from server (AlreadyExists)** messages. They are caused by the Migration Toolkit for Containers Operator creating resources for earlier versions of OpenShift Container Platform 4 that are provided in later releases.

8. Create the **MigrationController** object:

```
$ oc create -f controller.yml
```

9. Verify that the MTC pods are running:

```
$ oc get pods -n openshift-migration
```

4.4. PROXY CONFIGURATION

For OpenShift Container Platform 4.1 and earlier versions, you must configure proxies in the **MigrationController** custom resource (CR) manifest after you install the Migration Toolkit for Containers Operator because these versions do not support a cluster-wide **proxy** object.

For OpenShift Container Platform 4.2 to 4.14, the MTC inherits the cluster-wide proxy settings. You can change the proxy parameters if you want to override the cluster-wide proxy settings.

4.4.1. Direct volume migration

Direct Volume Migration (DVM) was introduced in MTC 1.4.2. DVM supports only one proxy. The source cluster cannot access the route of the target cluster if the target cluster is also behind a proxy.

If you want to perform a DVM from a source cluster behind a proxy, you must configure a TCP proxy that works at the transport layer and forwards the SSL connections transparently without decrypting and re-encrypting them with their own SSL certificates. A Stunnel proxy is an example of such a proxy.

4.4.1.1. TCP proxy setup for DVM

You can set up a direct connection between the source and the target cluster through a TCP proxy and configure the **stunnel_tcp_proxy** variable in the **MigrationController** CR to use the proxy:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigrationController
metadata:
  name: migration-controller
```

```
  namespace: openshift-migration
spec:
  [...]
  stunnel_tcp_proxy: http://username:password@ip:port
```

Direct volume migration (DVM) supports only basic authentication for the proxy. Moreover, DVM works only from behind proxies that can tunnel a TCP connection transparently. HTTP/HTTPS proxies in man-in-the-middle mode do not work. The existing cluster-wide proxies might not support this behavior. As a result, the proxy settings for DVM are intentionally kept different from the usual proxy configuration in MTC.

4.4.1.2. Why use a TCP proxy instead of an HTTP/HTTPS proxy?

You can enable DVM by running Rsync between the source and the target cluster over an OpenShift route. Traffic is encrypted using Stunnel, a TCP proxy. The Stunnel running on the source cluster initiates a TLS connection with the target Stunnel and transfers data over an encrypted channel.

Cluster-wide HTTP/HTTPS proxies in OpenShift are usually configured in man-in-the-middle mode where they negotiate their own TLS session with the outside servers. However, this does not work with Stunnel. Stunnel requires that its TLS session be untouched by the proxy, essentially making the proxy a transparent tunnel which simply forwards the TCP connection as-is. Therefore, you must use a TCP proxy.

4.4.1.3. Known issue

Migration fails with error Upgrade request required

The migration Controller uses the SPDY protocol to execute commands within remote pods. If the remote cluster is behind a proxy or a firewall that does not support the SPDY protocol, the migration controller fails to execute remote commands. The migration fails with the error message **Upgrade request required**. Workaround: Use a proxy that supports the SPDY protocol.

In addition to supporting the SPDY protocol, the proxy or firewall also must pass the **Upgrade** HTTP header to the API server. The client uses this header to open a websocket connection with the API server. If the **Upgrade** header is blocked by the proxy or firewall, the migration fails with the error message **Upgrade request required**. Workaround: Ensure that the proxy forwards the **Upgrade** header.

4.4.2. Tuning network policies for migrations

OpenShift supports restricting traffic to or from pods using *NetworkPolicy* or *EgressFirewalls* based on the network plugin used by the cluster. If any of the source namespaces involved in a migration use such mechanisms to restrict network traffic to pods, the restrictions might inadvertently stop traffic to Rsync pods during migration.

Rsync pods running on both the source and the target clusters must connect to each other over an OpenShift Route. Existing *NetworkPolicy* or *EgressNetworkPolicy* objects can be configured to automatically exempt Rsync pods from these traffic restrictions.

4.4.2.1. NetworkPolicy configuration

4.4.2.1.1. Egress traffic from Rsync pods

You can use the unique labels of Rsync pods to allow egress traffic to pass from them if the **NetworkPolicy** configuration in the source or destination namespaces blocks this type of traffic. The following policy allows **all** egress traffic from Rsync pods in the namespace:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-egress-from-rsync-pods
spec:
  podSelector:
    matchLabels:
      owner: directvolumemigration
      app: directvolumemigration-rsync-transfer
  egress:
  - {}
  policyTypes:
  - Egress
```

4.4.2.1.2. Ingress traffic to Rsync pods

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-egress-from-rsync-pods
spec:
  podSelector:
    matchLabels:
      owner: directvolumemigration
      app: directvolumemigration-rsync-transfer
  ingress:
  - {}
  policyTypes:
  - Ingress
```

4.4.2.2. EgressNetworkPolicy configuration

The **EgressNetworkPolicy** object or *Egress Firewalls* are OpenShift constructs designed to block egress traffic leaving the cluster.

Unlike the **NetworkPolicy** object, the Egress Firewall works at a project level because it applies to all pods in the namespace. Therefore, the unique labels of Rsync pods do not exempt only Rsync pods from the restrictions. However, you can add the CIDR ranges of the source or target cluster to the *Allow* rule of the policy so that a direct connection can be setup between two clusters.

Based on which cluster the Egress Firewall is present in, you can add the CIDR range of the other cluster to allow egress traffic between the two:

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: test-egress-policy
  namespace: <namespace>
spec:
  egress:
```

```
  - to:
      cidrSelector: <cidr_of_source_or_target_cluster>
    type: Deny
```

4.4.2.3. Choosing alternate endpoints for data transfer

By default, DVM uses an OpenShift Container Platform route as an endpoint to transfer PV data to destination clusters. You can choose another type of supported endpoint, if cluster topologies allow.

For each cluster, you can configure an endpoint by setting the **rsync_endpoint_type** variable on the appropriate **destination** cluster in your **MigrationController** CR:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigrationController
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  rsync_endpoint_type: [NodePort|ClusterIP|Route]
```

4.4.2.4. Configuring supplemental groups for Rsync pods

When your PVCs use a shared storage, you can configure the access to that storage by adding supplemental groups to Rsync pod definitions in order for the pods to allow access:

Table 4.2. Supplementary groups for Rsync pods

Variable	Type	Default	Description
src_supplemental_groups	string	Not set	Comma-separated list of supplemental groups for source Rsync pods
target_supplemental_groups	string	Not set	Comma-separated list of supplemental groups for target Rsync pods

Example usage

The **MigrationController** CR can be updated to set values for these supplemental groups:

```
spec:
  src_supplemental_groups: "1000,2000"
  target_supplemental_groups: "2000,3000"
```

4.4.3. Configuring proxies

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges on all clusters.

Procedure

1. Get the **MigrationController** CR manifest:

```
$ oc get migrationcontroller <migration_controller> -n openshift-migration
```

2. Update the proxy parameters:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigrationController
metadata:
  name: <migration_controller>
  namespace: openshift-migration
...
spec:
  stunnel_tcp_proxy: http://<username>:<password>@<ip>:<port> 1
  noProxy: example.com 2
```

1. Stunnel proxy URL for direct volume migration.
2. Comma-separated list of destination domain names, domains, IP addresses, or other network CIDRs to exclude proxying.

   Preface a domain with **.** to match subdomains only. For example, **.y.com** matches **x.y.com**, but not **y.com**. Use ***** to bypass proxy for all destinations. If you scale up workers that are not included in the network defined by the **networking.machineNetwork[].cidr** field from the installation configuration, you must add them to this list to prevent connection issues.

   This field is ignored if neither the **httpProxy** nor the **httpsProxy** field is set.

3. Save the manifest as **migration-controller.yaml**.

4. Apply the updated manifest:

```
$ oc replace -f migration-controller.yaml -n openshift-migration
```

For more information, see Configuring the cluster-wide proxy.

4.5. RUNNING RSYNC AS EITHER ROOT OR NON-ROOT

IMPORTANT

This section applies only when you are working with the OpenShift API, not the web console.

OpenShift environments have the **PodSecurityAdmission** controller enabled by default. This controller requires cluster administrators to enforce Pod Security Standards by means of namespace labels. All workloads in the cluster are expected to run one of the following Pod Security Standard levels: **Privileged**, **Baseline** or **Restricted**. Every cluster has its own default policy set.

To guarantee successful data transfer in all environments, MTC 1.7.5 introduced changes in Rsync pods, including running Rsync pods as non-root user by default. This ensures that data transfer is possible

even for workloads that do not necessarily require higher privileges. This change was made because it is best to run workloads with the lowest level of privileges possible.

Manually overriding default non-root operation for data transfer

Although running Rsync pods as non-root user works in most cases, data transfer might fail when you run workloads as root user on the source side. MTC provides two ways to manually override default non-root operation for data transfer:

- Configure all migrations to run an Rsync pod as root on the destination cluster for all migrations.
- Run an Rsync pod as root on the destination cluster per migration.

In both cases, you must set the following labels on the source side of any namespaces that are running workloads with higher privileges prior to migration: **enforce**, **audit**, and **warn.**

To learn more about Pod Security Admission and setting values for labels, see Controlling pod security admission synchronization.

4.5.1. Configuring the MigrationController CR as root or non-root for all migrations

By default, Rsync runs as non-root.

On the destination cluster, you can configure the **MigrationController** CR to run Rsync as root.

Procedure

- Configure the **MigrationController** CR as follows:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigrationController
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  migration_rsync_privileged: true
```

This configuration will apply to all future migrations.

4.5.2. Configuring the MigMigration CR as root or non-root per migration

On the destination cluster, you can configure the **MigMigration** CR to run Rsync as root or non-root, with the following non-root options:

- As a specific user ID (UID)
- As a specific group ID (GID)

Procedure

1. To run Rsync as root, configure the **MigMigration** CR according to this example:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigMigration
metadata:
```

```
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  runAsRoot: true
```

2. To run Rsync as a specific User ID (UID) or as a specific Group ID (GID), configure the **MigMigration** CR according to this example:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigMigration
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  runAsUser: 10010001
  runAsGroup: 3
```

4.6. CONFIGURING A REPLICATION REPOSITORY

The Multicloud Object Gateway is the only supported option for a restricted network environment.

MTC supports the file system and snapshot data copy methods for migrating data from the source cluster to the target cluster. You can select a method that is suited for your environment and is supported by your storage provider.

4.6.1. Prerequisites

- All clusters must have uninterrupted network access to the replication repository.
- If you use a proxy server with an internally hosted replication repository, you must ensure that the proxy allows access to the replication repository.

4.6.2. Retrieving Multicloud Object Gateway credentials

NOTE

Although the MCG Operator is deprecated, the MCG plugin is still available for OpenShift Data Foundation. To download the plugin, browse to Download Red Hat OpenShift Data Foundation and download the appropriate MCG plugin for your operating system.

Prerequisites

- You must deploy OpenShift Data Foundation by using the appropriate Red Hat OpenShift Data Foundation deployment guide.

4.6.3. Additional resources

Procedure

- Disconnected environment in the Red Hat OpenShift Data Foundation documentation.

- MTC workflow
- About data copy methods
- Adding a replication repository to the MTC web console

4.7. UNINSTALLING MTC AND DELETING RESOURCES

You can uninstall the Migration Toolkit for Containers (MTC) and delete its resources to clean up the cluster.

NOTE

Deleting the **velero** CRDs removes Velero from the cluster.

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges.

Procedure

1. Delete the **MigrationController** custom resource (CR) on all clusters:

```
$ oc delete migrationcontroller <migration_controller>
```

2. Uninstall the Migration Toolkit for Containers Operator on OpenShift Container Platform 4 by using the Operator Lifecycle Manager.
3. Delete cluster-scoped resources on all clusters by running the following commands:
   - **migration** custom resource definitions (CRDs):

```
$ oc delete $(oc get crds -o name | grep 'migration.openshift.io')
```

   - **velero** CRDs:

```
$ oc delete $(oc get crds -o name | grep 'velero')
```

   - **migration** cluster roles:

```
$ oc delete $(oc get clusterroles -o name | grep 'migration.openshift.io')
```

   - **migration-operator** cluster role:

```
$ oc delete clusterrole migration-operator
```

   - **velero** cluster roles:

```
$ oc delete $(oc get clusterroles -o name | grep 'velero')
```

   - **migration** cluster role bindings:

```
$ oc delete $(oc get clusterrolebindings -o name | grep 'migration.openshift.io')
```

- **migration-operator** cluster role bindings:

```
$ oc delete clusterrolebindings migration-operator
```

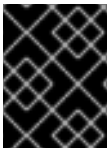
- **velero** cluster role bindings:

```
$ oc delete $(oc get clusterrolebindings -o name | grep 'velero')
```

CHAPTER 5. UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS

You can upgrade the Migration Toolkit for Containers (MTC) on OpenShift Container Platform 4.14 by using Operator Lifecycle Manager.

You can upgrade MTC on OpenShift Container Platform 4.5, and earlier versions, by reinstalling the legacy Migration Toolkit for Containers Operator.



IMPORTANT

If you are upgrading from MTC version 1.3, you must perform an additional procedure to update the **MigPlan** custom resource (CR).

5.1. UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS ON OPENSHIFT CONTAINER PLATFORM 4.14

You can upgrade the Migration Toolkit for Containers (MTC) on OpenShift Container Platform 4.14 by using the Operator Lifecycle Manager.

IMPORTANT

When upgrading the MTC by using the Operator Lifecycle Manager, you must use a supported migration path.

Migration paths

- Migrating from OpenShift Container Platform 3 to OpenShift Container Platform 4 requires a legacy MTC Operator and MTC 1.7.x.
- Migrating from MTC 1.7.x to MTC 1.8.x is not supported.
- You must use MTC 1.7.x to migrate anything with a source of OpenShift Container Platform 4.9 or earlier.
    - MTC 1.7.x must be used on both source and destination.
- MTC 1.8.x only supports migrations from OpenShift Container Platform 4.10 or later to OpenShift Container Platform 4.10 or later. For migrations only involving cluster versions 4.10 and later, either 1.7.x or 1.8.x may be used. However, it must be the same MTC version on both source & destination.
    - Migration from source MTC 1.7.x to destination MTC 1.8.x is unsupported.
    - Migration from source MTC 1.8.x to destination MTC 1.7.x is unsupported.
    - Migration from source MTC 1.7.x to destination MTC 1.7.x is supported.
    - Migration from source MTC 1.8.x to destination MTC 1.8.x is supported

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges.

Procedure

1. In the OpenShift Container Platform console, navigate to **Operators → Installed Operators**. Operators that have a pending upgrade display an **Upgrade available** status.
2. Click **Migration Toolkit for Containers Operator**.
3. Click the **Subscription** tab. Any upgrades requiring approval are displayed next to **Upgrade Status**. For example, it might display **1 requires approval**.
4. Click **1 requires approval**, then click **Preview Install Plan**.
5. Review the resources that are listed as available for upgrade and click **Approve**.
6. Navigate back to the **Operators → Installed Operators** page to monitor the progress of the upgrade. When complete, the status changes to **Succeeded** and **Up to date**.
7. Click **Workloads → Pods** to verify that the MTC pods are running.

5.2. UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS TO 1.8.0

To upgrade the Migration Toolkit for Containers to 1.8.0, complete the following steps.

Procedure

1. Determine subscription names and current channels to work with for upgrading by using one of the following methods:
   - Determine the subscription names and channels by running the following command:

     ```
     $ oc -n openshift-migration get sub
     ```

     Example output

     ```
     NAME                                                                PACKAGE             SOURCE
     CHANNEL
     mtc-operator                                                         mtc-operator        mtc-operator-
     catalog   release-v1.7
     redhat-oadp-operator-stable-1.0-mtc-operator-catalog-openshift-marketplace   redhat-
     oadp-operator   mtc-operator-catalog   stable-1.0
     ```

   - Or return the subscription names and channels in JSON by running the following command:

     ```
     $ oc -n openshift-migration get sub -o json | jq -r '.items[] | { name: .metadata.name,
     package: .spec.name, channel: .spec.channel }'
     ```

     Example output

     ```
     {
       "name": "mtc-operator",
       "package": "mtc-operator",
       "channel": "release-v1.7"
     }
     ```

```
{
  "name": "redhat-oadp-operator-stable-1.0-mtc-operator-catalog-openshift-marketplace",
  "package": "redhat-oadp-operator",
  "channel": "stable-1.0"
}
```

2. For each subscription, patch to move from the MTC 1.7 channel to the MTC 1.8 channel by running the following command:

```
$ oc -n openshift-migration patch subscription mtc-operator --type merge --patch '{"spec":
{"channel": "release-v1.8"}}'
```

Example output

```
subscription.operators.coreos.com/mtc-operator patched
```

5.2.1. Upgrading OADP 1.0 to 1.2 for Migration Toolkit for Containers 1.8.0

To upgrade OADP 1.0 to 1.2 for Migration Toolkit for Containers 1.8.0, complete the following steps.

Procedure

- For each subscription, patch the OADP operator from OADP 1.0 to OADP 1.2 by running the following command:

```
$ oc -n openshift-migration patch subscription redhat-oadp-operator-stable-1.0-mtc-operator-
catalog-openshift-marketplace --type merge --patch '{"spec": {"channel":"stable-1.2"}}'
```

NOTE

Sections indicating the user-specific returned **NAME** values that are used for the installation of MTC & OADP, respectively.

Example output

```
subscription.operators.coreos.com/redhat-oadp-operator-stable-1.0-mtc-operator-catalog-
openshift-marketplace patched
```

NOTE

The returned value will be similar to **redhat-oadp-operator-stable-1.0-mtc-operator-catalog-openshift-marketplace**, which is used in this example.

  - If the **installPlanApproval** parameter is set to **Automatic**, the Operator Lifecycle Manager (OLM) begins the upgrade process.
  - If the **installPlanApproval** parameter is set to **Manual**, you must approve each **installPlan** before the OLM begins the upgrades.

Verification

1. Verify that the OLM has completed the upgrades of OADP and MTC by running the following command:

```
$ oc -n openshift-migration get subscriptions.operators.coreos.com mtc-operator -o json | jq
'.status | (."state"=="AtLatestKnown")'
```

2. When a value of **true** is returned, verify the channel used for each subscription by running the following command:

```
$ oc -n openshift-migration get sub -o json | jq -r '.items[] | {name: .metadata.name, channel:
.spec.channel }'
```

Example output

```
{
  "name": "mtc-operator",
  "channel": "release-v1.8"
}
{
  "name": "redhat-oadp-operator-stable-1.0-mtc-operator-catalog-openshift-marketplace",
  "channel": "stable-1.2"
}
```

Confirm that the `mtc-operator.v1.8.0` and `oadp-operator.v1.2.x` packages are installed by running the following command:

```
$ oc -n openshift-migration get csv
```

Example output

```
NAME                  DISPLAY                                VERSION   REPLACES
PHASE
mtc-operator.v1.8.0      Migration Toolkit for Containers Operator   1.8.0     mtc-
operator.v1.7.13     Succeeded
oadp-operator.v1.2.2     OADP Operator                          1.2.2     oadp-operator.v1.0.13
Succeeded
```

5.3. UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS ON OPENSHIFT CONTAINER PLATFORM VERSIONS 4.2 TO 4.5

You can upgrade Migration Toolkit for Containers (MTC) on OpenShift Container Platform versions 4.2 to 4.5 by manually installing the legacy Migration Toolkit for Containers Operator.

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges.
- You must have access to **registry.redhat.io**.
- You must have **podman** installed.

Procedure

1. Log in to **registry.redhat.io** with your Red Hat Customer Portal credentials by entering the following command:

```
$ podman login registry.redhat.io
```

2. Download the **operator.yml** file by entering the following command:

```
$ podman cp $(podman create registry.redhat.io/rhmtc/openshift-migration-legacy-rhel8-operator:v1.7:/operator.yml ./
```

3. Replace the Migration Toolkit for Containers Operator by entering the following command:

```
$ oc replace --force -f operator.yml
```

4. Scale the **migration-operator** deployment to **0** to stop the deployment by entering the following command:

```
$ oc scale -n openshift-migration --replicas=0 deployment/migration-operator
```

5. Scale the **migration-operator** deployment to **1** to start the deployment and apply the changes by entering the following command:

```
$ oc scale -n openshift-migration --replicas=1 deployment/migration-operator
```

6. Verify that the **migration-operator** was upgraded by entering the following command:

```
$ oc -o yaml -n openshift-migration get deployment/migration-operator | grep image: | awk -F ":" '{ print $NF }'
```

7. Download the **controller.yml** file by entering the following command:

```
$ podman cp $(podman create registry.redhat.io/rhmtc/openshift-migration-legacy-rhel8-operator:v1.7):/operator.yml ./
```

8. Create the **migration-controller** object by entering the following command:

```
$ oc create -f controller.yml
```

9. Verify that the MTC pods are running by entering the following command:

```
$ oc get pods -n openshift-migration
```

5.4. UPGRADING MTC 1.3 TO 1.8

If you are upgrading Migration Toolkit for Containers (MTC) version 1.3.x to 1.8, you must update the **MigPlan** custom resource (CR) manifest on the cluster on which the **MigrationController** pod is running.

Because the **indirectImageMigration** and **indirectVolumeMigration** parameters do not exist in MTC

1.3, their default value in version 1.4 is **false**, which means that direct image migration and direct volume migration are enabled. Because the direct migration requirements are not fulfilled, the migration plan cannot reach a **Ready** state unless these parameter values are changed to **true**.

IMPORTANT

- Migrating from OpenShift Container Platform 3 to OpenShift Container Platform 4 requires a legacy MTC Operator and MTC 1.7.x.
- Upgrading MTC 1.7.x to 1.8.x requires manually updating the OADP channel from **stable-1.0** to **stable-1.2** in order to successfully complete the upgrade from 1.7.x to 1.8.x.

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges.

Procedure

1. Log in to the cluster on which the **MigrationController** pod is running.
2. Get the **MigPlan** CR manifest:

```
$ oc get migplan <migplan> -o yaml -n openshift-migration
```

3. Update the following parameter values and save the file as **migplan.yaml**:

```
...
spec:
  indirectImageMigration: true
  indirectVolumeMigration: true
```

4. Replace the **MigPlan** CR manifest to apply the changes:

```
$ oc replace -f migplan.yaml -n openshift-migration
```

5. Get the updated **MigPlan** CR manifest to verify the changes:

```
$ oc get migplan <migplan> -o yaml -n openshift-migration
```

CHAPTER 6. PREMIGRATION CHECKLISTS

Before you migrate your application workloads with the Migration Toolkit for Containers (MTC), review the following checklists.

6.1. CLUSTER HEALTH CHECKLIST

❑ The clusters meet the minimum hardware requirements for the specific platform and installation method, for example, on bare metal.

❑ All MTC prerequisites are met.

❑ All nodes have an active OpenShift Container Platform subscription.

❑ You have verified node health.

❑ The identity provider is working.

❑ The migration network has a minimum throughput of 10 Gbps.

❑ The clusters have sufficient resources for migration.

NOTE

Clusters require additional memory, CPUs, and storage in order to run a migration on top of normal workloads. Actual resource requirements depend on the number of Kubernetes resources being migrated in a single migration plan. You must test migrations in a non-production environment in order to estimate the resource requirements.

❑ The etcd disk performance of the clusters has been checked with **fio**.

6.2. SOURCE CLUSTER CHECKLIST

❑ You have checked for persistent volumes (PVs) with abnormal configurations stuck in a **Terminating** state by running the following command:

```
$ oc get pv
```

❑ You have checked for pods whose status is other than **Running** or **Completed** by running the following command:

```
$ oc get pods --all-namespaces | egrep -v 'Running | Completed'
```

❑ You have checked for pods with a high restart count by running the following command:

```
$ oc get pods --all-namespaces --field-selector=status.phase=Running \
  -o json | jq '.items[]|select(any( .status.containerStatuses[]; \
  .restartCount > 3))|.metadata.name'
```

Even if the pods are in a **Running** state, a high restart count might indicate underlying problems.

❑ The cluster certificates are valid for the duration of the migration process.

❑ You have checked for pending certificate-signing requests by running the following command:

```
$ oc get csr -A | grep pending -i
```

❑ The registry uses a recommended storage type.

❑ You can read and write images to the registry.

❑ The etcd cluster is healthy.

❑ The average API server response time on the source cluster is less than 50 ms.

6.3. TARGET CLUSTER CHECKLIST

❑ The cluster has the correct network configuration and permissions to access external services, for example, databases, source code repositories, container image registries, and CI/CD tools.

❑ External applications and services that use services provided by the cluster have the correct network configuration and permissions to access the cluster.

❑ Internal container image dependencies are met.

❑ The target cluster and the replication repository have sufficient storage space.

CHAPTER 7. NETWORK CONSIDERATIONS

Review the strategies for redirecting your application network traffic after migration.

7.1. DNS CONSIDERATIONS

The DNS domain of the target cluster is different from the domain of the source cluster. By default, applications get FQDNs of the target cluster after migration.

To preserve the source DNS domain of migrated applications, select one of the two options described below.

7.1.1. Isolating the DNS domain of the target cluster from the clients

You can allow the clients' requests sent to the DNS domain of the source cluster to reach the DNS domain of the target cluster without exposing the target cluster to the clients.

Procedure

1. Place an exterior network component, such as an application load balancer or a reverse proxy, between the clients and the target cluster.
2. Update the application FQDN on the source cluster in the DNS server to return the IP address of the exterior network component.
3. Configure the network component to send requests received for the application in the source domain to the load balancer in the target cluster domain.
4. Create a wildcard DNS record for the ***.apps.source.example.com** domain that points to the IP address of the load balancer of the source cluster.
5. Create a DNS record for each application that points to the IP address of the exterior network component in front of the target cluster. A specific DNS record has higher priority than a wildcard record, so no conflict arises when the application FQDN is resolved.

NOTE

- The exterior network component must terminate all secure TLS connections. If the connections pass through to the target cluster load balancer, the FQDN of the target application is exposed to the client and certificate errors occur.
- The applications must not return links referencing the target cluster domain to the clients. Otherwise, parts of the application might not load or work properly.

7.1.2. Setting up the target cluster to accept the source DNS domain

You can set up the target cluster to accept requests for a migrated application in the DNS domain of the source cluster.

Procedure

For both non-secure HTTP access and secure HTTPS access, perform the following steps:

1. Create a route in the target cluster's project that is configured to accept requests addressed to the application's FQDN in the source cluster:

```
$ oc expose svc <app1-svc> --hostname <app1.apps.source.example.com> \
 -n <app1-namespace>
```

With this new route in place, the server accepts any request for that FQDN and sends it to the corresponding application pods. In addition, when you migrate the application, another route is created in the target cluster domain. Requests reach the migrated application using either of these hostnames.

2. Create a DNS record with your DNS provider that points the application's FQDN in the source cluster to the IP address of the default load balancer of the target cluster. This will redirect traffic away from your source cluster to your target cluster.
The FQDN of the application resolves to the load balancer of the target cluster. The default Ingress Controller router accept requests for that FQDN because a route for that hostname is exposed.

For secure HTTPS access, perform the following additional step:

1. Replace the x509 certificate of the default Ingress Controller created during the installation process with a custom certificate.

2. Configure this certificate to include the wildcard DNS domains for both the source and target clusters in the **subjectAltName** field.
The new certificate is valid for securing connections made using either DNS domain.

Additional resources

- See Replacing the default ingress certificate for more information.

7.2. NETWORK TRAFFIC REDIRECTION STRATEGIES

After a successful migration, you must redirect network traffic of your stateless applications from the source cluster to the target cluster.

The strategies for redirecting network traffic are based on the following assumptions:

- The application pods are running on both the source and target clusters.
- Each application has a route that contains the source cluster hostname.
- The route with the source cluster hostname contains a CA certificate.
- For HTTPS, the target router CA certificate contains a Subject Alternative Name for the wildcard DNS record of the source cluster.

Consider the following strategies and select the one that meets your objectives.

- Redirecting all network traffic for all applications at the same time
Change the wildcard DNS record of the source cluster to point to the target cluster router's virtual IP address (VIP).

  This strategy is suitable for simple applications or small migrations.

- Redirecting network traffic for individual applications

Create a DNS record for each application with the source cluster hostname pointing to the target cluster router's VIP. This DNS record takes precedence over the source cluster wildcard DNS record.

- Redirecting network traffic gradually for individual applications

    1. Create a proxy that can direct traffic to both the source cluster router's VIP and the target cluster router's VIP, for each application.

    2. Create a DNS record for each application with the source cluster hostname pointing to the proxy.

    3. Configure the proxy entry for the application to route a percentage of the traffic to the target cluster router's VIP and the rest of the traffic to the source cluster router's VIP.

    4. Gradually increase the percentage of traffic that you route to the target cluster router's VIP until all the network traffic is redirected.

- User-based redirection of traffic for individual applications
  Using this strategy, you can filter TCP/IP headers of user requests to redirect network traffic for predefined groups of users. This allows you to test the redirection process on specific populations of users before redirecting the entire network traffic.

    1. Create a proxy that can direct traffic to both the source cluster router's VIP and the target cluster router's VIP, for each application.

    2. Create a DNS record for each application with the source cluster hostname pointing to the proxy.

    3. Configure the proxy entry for the application to route traffic matching a given header pattern, such as **test customers**, to the target cluster router's VIP and the rest of the traffic to the source cluster router's VIP.

    4. Redirect traffic to the target cluster router's VIP in stages until all the traffic is on the target cluster router's VIP.

CHAPTER 8. DIRECT MIGRATION REQUIREMENTS

Direct Migration is available with Migration Toolkit for Containers (MTC) 1.4.0 or later.

There are two parts of the Direct Migration:

- Direct Volume Migration
- Direct Image Migration

Direct Migration enables the migration of persistent volumes and internal images directly from the source cluster to the destination cluster without an intermediary replication repository (object storage).

8.1. PREREQUISITES

- Expose the internal registries for both clusters (source and destination) involved in the migration for external traffic.
- Ensure the remote source and destination clusters can communicate using OpenShift Container Platform routes on port 443.
- Configure the exposed registry route in the source and destination MTC clusters; do this by specifying the **spec.exposedRegistryPath** field or from the MTC UI.

NOTE

  - If the destination cluster is the same as the host cluster (where a migration controller exists), there is no need to configure the exposed registry route for that particular MTC cluster.
  - The **spec.exposedRegistryPath** is required only for Direct Image Migration and not Direct Volume Migration.

- Ensure the two spec flags in **MigPlan** custom resource (CR) **indirectImageMigration** and **indirectVolumeMigration** are set to **false** for Direct Migration to be performed. The default value for these flags is **false**.

The Direct Migration feature of MTC uses the Rsync utility.

8.2. RSYNC CONFIGURATION FOR DIRECT VOLUME MIGRATION

Direct Volume Migration (DVM) in MTC uses Rsync to synchronize files between the source and the target persistent volumes (PVs), using a direct connection between the two PVs.

Rsync is a command-line tool that allows you to transfer files and directories to local and remote destinations.

The **rsync** command used by DVM is optimized for clusters functioning as expected.

The **MigrationController** CR exposes the following variables to configure **rsync_options** in Direct Volume Migration:

Variable	Type	Default value	Description
rsync_opt_bwlimit	int	Not set	When set to a positive integer, **--bwlimit=<int>** option is added to Rsync command.
rsync_opt_archive	bool	**true**	Sets the **--archive** option in the Rsync command.
rsync_opt_partial	bool	**true**	Sets the **--partial** option in the Rsync command.
rsync_opt_delete	bool	**true**	Sets the **--delete** option in the Rsync command.
rsync_opt_hardlinks	bool	**true**	Sets the **--hard-links** option is the Rsync command.
rsync_opt_info	string	**COPY2** **DEL2** **REMOVE2** **SKIP2** **FLIST2** **PROGRESS2** **STATS2**	Enables detailed logging in Rsync Pod.
rsync_opt_extras	string	Empty	Reserved for any other arbitrary options.

- Setting the options set through the variables above are *global* for all migrations. The configuration will take effect for all future migrations as soon as the Operator successfully reconciles the **MigrationController** CR. Any ongoing migration can use the updated settings depending on which step it currently is in. Therefore, it is recommended that the settings be applied before running a migration. The users can always update the settings as needed.
- Use the **rsync_opt_extras** variable with caution. Any options passed using this variable are appended to the **rsync** command, with addition. Ensure you add white spaces when specifying more than one option. Any error in specifying options can lead to a failed migration. However, you can update **MigrationController** CR as many times as you require for future migrations.
- Customizing the **rsync_opt_info** flag can adversely affect the progress reporting capabilities in MTC. However, removing progress reporting can have a performance advantage. This option should only be used when the performance of Rsync operation is observed to be unacceptable.

NOTE

The default configuration used by DVM is tested in various environments. It is acceptable for most production use cases provided the clusters are healthy and performing well. These configuration variables should be used in case the default settings do not work and the Rsync operation fails.

8.2.1. Resource limit configurations for Rsync pods

The **MigrationController** CR exposes following variables to configure resource usage requirements and limits on Rsync:

Variable	Type	Default	Description
source_rsync_pod_cpu_limits	string	**1**	Source rsync pod's CPU limit
source_rsync_pod_memory_limits	string	**1Gi**	Source rsync pod's memory limit
source_rsync_pod_cpu_requests	string	**400m**	Source rsync pod's cpu requests
source_rsync_pod_memory_requests	string	**1Gi**	Source rsync pod's memory requests
target_rsync_pod_cpu_limits	string	**1**	Target rsync pod's cpu limit
target_rsync_pod_cpu_requests	string	**400m**	Target rsync pod's cpu requests
target_rsync_pod_memory_limits	string	**1Gi**	Target rsync pod's memory limit
target_rsync_pod_memory_requests	string	**1Gi**	Target rsync pod's memory requests

8.2.1.1. Supplemental group configuration for Rsync pods

If Persistent Volume Claims (PVC) are using a shared storage, the access to storage can be configured by adding supplemental groups to Rsync pod definitions in order for the pods to allow access:

Variable	Type	Default	Description
src_supplemental_groups	string	Not Set	Comma separated list of supplemental groups for source Rsync pods

Variable	Type	Default	Description
target_supplemental_groups	string	Not Set	Comma separated list of supplemental groups for target Rsync Pods

For example, the **MigrationController** CR can be updated to set the previous values:

```
spec:
  src_supplemental_groups: "1000,2000"
  target_supplemental_groups: "2000,3000"
```

8.2.1.2. Rsync retry configuration

With Migration Toolkit for Containers (MTC) 1.4.3 and later, a new ability of retrying a failed Rsync operation is introduced.

By default, the migration controller retries Rsync until all of the data is successfully transferred from the source to the target volume or a specified number of retries is met. The default retry limit is set to **20**.

For larger volumes, a limit of **20** retries may not be sufficient.

You can increase the retry limit by using the following variable in the **MigrationController** CR:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigrationController
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  rsync_backoff_limit: 40
```

In this example, the retry limit is increased to **40**.

8.2.1.3. Running Rsync as either root or non-root

OpenShift Container Platform environments have the **PodSecurityAdmission** controller enabled by default. This controller requires cluster administrators to enforce Pod Security Standards by means of namespace labels. All workloads in the cluster are expected to run one of the following Pod Security Standard levels: **privileged**, **baseline** or **restricted**. Every cluster has its own default policy set.

To guarantee successful data transfer in all environments, Migration Toolkit for Containers (MTC) 1.7.5 introduced changes in Rsync pods, including running Rsync pods as non-root user by default. This ensures that data transfer is possible even for workloads that do not necessarily require higher privileges. This change was made because it is best to run workloads with the lowest level of privileges possible.

8.2.1.3.1. Manually overriding default non-root operation for data transfer

Although running Rsync pods as non-root user works in most cases, data transfer might fail when you run workloads as root user on the source side. MTC provides two ways to manually override default non-root operation for data transfer:

- Configure all migrations to run an Rsync pod as root on the destination cluster for all migrations.
- Run an Rsync pod as root on the destination cluster per migration.

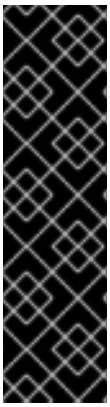
In both cases, you must set the following labels on the source side of any namespaces that are running workloads with higher privileges before migration: **enforce**, **audit**, and **warn.**

8.2.1.3.2. About pod security admission

OpenShift Container Platform includes Kubernetes pod security admission . Pods that do not comply with the pod security admission defined globally or at the namespace level are not admitted to the cluster and cannot run.

Globally, the **privileged** profile is enforced, and the **restricted** profile is used for warnings and audits.

You can also configure the pod security admission settings at the namespace level.



IMPORTANT

Do not run workloads in or share access to default projects. Default projects are reserved for running core cluster components.

The following default projects are considered highly privileged: **default**, **kube-public**, **kube-system**, **openshift**, **openshift-infra**, **openshift-node**, and other system-created projects that have the **openshift.io/run-level** label set to **0** or **1**. Functionality that relies on admission plugins, such as pod security admission, security context constraints, cluster resource quotas, and image reference resolution, does not work in highly privileged projects.

8.2.1.3.2.1. Pod security admission modes

You can configure the following pod security admission modes for a namespace:

Table 8.1. Pod security admission modes

Mode	Label	Description
enforce	**pod-security.kubernetes.io/enforce**	Rejects a pod from admission if it does not comply with the set profile
audit	**pod-security.kubernetes.io/audit**	Logs audit events if a pod does not comply with the set profile
warn	**pod-security.kubernetes.io/warn**	Displays warnings if a pod does not comply with the set profile

8.2.1.3.2.2. Pod security admission profiles

You can set each of the pod security admission modes to one of the following profiles:

Table 8.2. Pod security admission profiles

Profile	Description
privileged	Least restrictive policy; allows for known privilege escalation
baseline	Minimally restrictive policy; prevents known privilege escalations
restricted	Most restrictive policy; follows current pod hardening best practices

8.2.1.3.2.3. Privileged namespaces

The following system namespaces are always set to the **privileged** pod security admission profile:

- **default**
- **kube-public**
- **kube-system**

You cannot change the pod security profile for these privileged namespaces.

Example privileged namespace configuration

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/audit: privileged
    pod-security.kubernetes.io/warn: privileged
  name: "<mig_namespace>"
# ...
```

Additional resources

- Controlling pod security admission synchronization .

8.2.1.3.3. Configuring the MigrationController CR as root or non-root for all migrations

By default, Rsync runs as non-root.

On the destination cluster, you can configure the **MigrationController** CR to run Rsync as root.

Procedure

- Configure the **MigrationController** CR as follows:

```
apiVersion: migration.openshift.io/v1alpha1
```

```
kind: MigrationController
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  migration_rsync_privileged: true
```

This configuration will apply to all future migrations.

8.2.1.3.4. Configuring the MigMigration CR as root or non-root per migration

On the destination cluster, you can configure the **MigMigration** CR to run Rsync as root or non-root, with the following non-root options:

- As a specific user ID (UID)
- As a specific group ID (GID)

Procedure

1. To run Rsync as root, configure the **MigMigration** CR according to this example:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigMigration
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  runAsRoot: true
```

2. To run Rsync as a specific User ID (UID) or as a specific Group ID (GID), configure the **MigMigration** CR according to this example:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigMigration
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  runAsUser: 10010001
  runAsGroup: 3
```

8.2.2. MigCluster Configuration

For every **MigCluster** resource created in Migration Toolkit for Containers (MTC), a **ConfigMap** named **migration-cluster-config** is created in the Migration Operator's namespace on the cluster which MigCluster resource represents.

The **migration-cluster-config** allows you to configure MigCluster specific values. The Migration Operator manages the **migration-cluster-config**.

You can configure every value in the **ConfigMap** using the variables exposed in the **MigrationController** CR:

Variable	Type	Required	Description
migration_stage_image_fqin	string	No	Image to use for Stage Pods (applicable only to IndirectVolumeMigration)
migration_registry_image_fqin	string	No	Image to use for Migration Registry
rsync_endpoint_type	string	No	Type of endpoint for data transfer (**Route**, **ClusterIP**, **NodePort**)
rsync_transfer_image_fqin	string	No	Image to use for Rsync Pods (applicable only to DirectVolumeMigration)
migration_rsync_privileged	bool	No	Whether to run Rsync Pods as privileged or not
migration_rsync_super_privileged	bool	No	Whether to run Rsync Pods as super privileged containers (**spc_t** SELinux context) or not
cluster_subdomain	string	No	Cluster's subdomain
migration_registry_readiness_timeout	int	No	Readiness timeout (in seconds) for Migration Registry Deployment
migration_registry_liveness_timeout	int	No	Liveness timeout (in seconds) for Migration Registry Deployment
exposed_registry_validation_path	string	No	Subpath to validate exposed registry in a MigCluster (for example /v2)

8.3. DIRECT MIGRATION KNOWN ISSUES

8.3.1. Applying the Skip SELinux relabel workaround with spc_t automatically on workloads running on OpenShift Container Platform

When attempting to migrate a namespace with Migration Toolkit for Containers (MTC) and a substantial volume associated with it, the **rsync-server** may become frozen without any further information to troubleshoot the issue.

8.3.1.1. Diagnosing the need for the Skip SELinux relabel workaround

Search for an error of **Unable to attach or mount volumes for pod…timed out waiting for the condition** in the kubelet logs from the node where the **rsync-server** for the Direct Volume Migration (DVM) runs.

Example kubelet log

```
kubenswrapper[3879]: W0326 16:30:36.749224    3879 volume_linux.go:49] Setting volume ownership for /var/lib/kubelet/pods/8905d88e-6531-4d65-9c2a-eff11dc7eb29/volumes/kubernetes.io~csi/pvc-287d1988-3fd9-4517-a0c7-22539acd31e6/mount and fsGroup set. If the volume has a lot of files then setting volume ownership could be slow, see https://github.com/kubernetes/kubernetes/issues/69699

kubenswrapper[3879]: E0326 16:32:02.706363    3879 kubelet.go:1841] "Unable to attach or mount volumes for pod; skipping pod" err="unmounted volumes=[8db9d5b032dab17d4ea9495af12e085a], unattached volumes=[crane2-rsync-server-secret 8db9d5b032dab17d4ea9495af12e085a kube-api-access-dlbd2 crane2-stunnel-server-config crane2-stunnel-server-secret crane2-rsync-server-config]: timed out waiting for the condition" pod="caboodle-preprod/rsync-server"

kubenswrapper[3879]: E0326 16:32:02.706496    3879 pod_workers.go:965] "Error syncing pod, skipping" err="unmounted volumes=[8db9d5b032dab17d4ea9495af12e085a], unattached volumes=[crane2-rsync-server-secret 8db9d5b032dab17d4ea9495af12e085a kube-api-access-dlbd2 crane2-stunnel-server-config crane2-stunnel-server-secret crane2-rsync-server-config]: timed out waiting for the condition" pod="caboodle-preprod/rsync-server" podUID=8905d88e-6531-4d65-9c2a-eff11dc7eb29
```

8.3.1.2. Resolving using the Skip SELinux relabel workaround

To resolve this issue, set the **migration_rsync_super_privileged** parameter to **true** in both the source and destination **MigClusters** using the **MigrationController** custom resource (CR).

Example MigrationController CR

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigrationController
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  migration_rsync_super_privileged: true (1)
  azure_resource_group: ""
  cluster_name: host
  mig_namespace_limit: "10"
  mig_pod_limit: "100"
  mig_pv_limit: "100"
  migration_controller: true
  migration_log_reader: true
  migration_ui: true
  migration_velero: true
  olm_managed: true
  restic_timeout: 1h
  version: 1.8.3
```

(1) The value of the **migration_rsync_super_privileged** parameter indicates whether or not to run Rsync Pods as *super privileged* containers (**spc_t selinux context**). Valid settings are **true** or **false**.

CHAPTER 9. MIGRATING YOUR APPLICATIONS

You can migrate your applications by using the Migration Toolkit for Containers (MTC) web console or the command line.

Most cluster-scoped resources are not yet handled by MTC. If your applications require cluster-scoped resources, you might have to create them manually on the target cluster.

You can use stage migration and cutover migration to migrate an application between clusters:

- Stage migration copies data from the source cluster to the target cluster without stopping the application. You can run a stage migration multiple times to reduce the duration of the cutover migration.
- Cutover migration stops the transactions on the source cluster and moves the resources to the target cluster.

You can use state migration to migrate an application's state:

- State migration copies selected persistent volume claims (PVCs).
- You can use state migration to migrate a namespace within the same cluster.

During migration, the MTC preserves the following namespace annotations:

- **openshift.io/sa.scc.mcs**
- **openshift.io/sa.scc.supplemental-groups**
- **openshift.io/sa.scc.uid-range**
  These annotations preserve the UID range, ensuring that the containers retain their file system permissions on the target cluster. There is a risk that the migrated UIDs could duplicate UIDs within an existing or future namespace on the target cluster.

9.1. MIGRATION PREREQUISITES

- You must be logged in as a user with **cluster-admin** privileges on all clusters.

Direct image migration

- You must ensure that the secure OpenShift image registry of the source cluster is exposed.
- You must create a route to the exposed registry.

Direct volume migration

- If your clusters use proxies, you must configure an Stunnel TCP proxy.

Clusters

- The source cluster must be upgraded to the latest MTC z-stream release.
- The MTC version must be the same on all clusters.

Network

- The clusters have unrestricted network access to each other and to the replication repository.
- If you copy the persistent volumes with **move**, the clusters must have unrestricted network access to the remote volumes.
- You must enable the following ports on an OpenShift Container Platform 4 cluster:
  - **6443** (API server)
  - **443** (routes)
  - **53** (DNS)
- You must enable port **443** on the replication repository if you are using TLS.

Persistent volumes (PVs)

- The PVs must be valid.
- The PVs must be bound to persistent volume claims.
- If you use snapshots to copy the PVs, the following additional prerequisites apply:
  - The cloud provider must support snapshots.
  - The PVs must have the same cloud provider.
  - The PVs must be located in the same geographic region.
  - The PVs must have the same storage class.

9.2. MIGRATING YOUR APPLICATIONS BY USING THE MTC WEB CONSOLE

You can configure clusters and a replication repository by using the MTC web console. Then, you can create and run a migration plan.

9.2.1. Launching the MTC web console

You can launch the Migration Toolkit for Containers (MTC) web console in a browser.

Prerequisites

- The MTC web console must have network access to the OpenShift Container Platform web console.
- The MTC web console must have network access to the OAuth authorization server.

Procedure

1. Log in to the OpenShift Container Platform cluster on which you have installed MTC.
2. Obtain the MTC web console URL by entering the following command:

```
$ oc get -n openshift-migration route/migration -o go-template='https://{{ .spec.host }}'
```

The output resembles the following: **https://migration-openshift-migration.apps.cluster.openshift.com**.

3. Launch a browser and navigate to the MTC web console.

NOTE

If you try to access the MTC web console immediately after installing the Migration Toolkit for Containers Operator, the console might not load because the Operator is still configuring the cluster. Wait a few minutes and retry.

4. If you are using self-signed CA certificates, you will be prompted to accept the CA certificate of the source cluster API server. The web page guides you through the process of accepting the remaining certificates.

5. Log in with your OpenShift Container Platform **username** and **password**.

9.2.2. Adding a cluster to the MTC web console

You can add a cluster to the Migration Toolkit for Containers (MTC) web console.

Prerequisites

- Cross-origin resource sharing must be configured on the source cluster.
- If you are using Azure snapshots to copy data:
  - You must specify the Azure resource group name for the cluster.
  - The clusters must be in the same Azure resource group.
  - The clusters must be in the same geographic location.
- If you are using direct image migration, you must expose a route to the image registry of the source cluster.

Procedure

1. Log in to the cluster.
2. Obtain the **migration-controller** service account token:

```
$ oc create token migration-controller -n openshift-migration
```

Example output

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwi
a3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJtaWciLCJrdWJlcm5ldGVz
LmlvL3NlcnZpY2VhY2NvdW50L3NlY3JldC5uYW1lIjoibWlnLXRva2VuLWs4dDJyIiwia3ViZXJuZ
XRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6Im1pZyIsImt1YmV
ybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImE1YjFiYWM
wLWMxYmYtMTFlOS05Y2NiLTAyOWRmODYwYjMwOCIsInN1YiI6InN5c3RlbTpzZXJ2aWNlY
WNjb3VudDptaWc6bWlnIn0.xqeeAINK7UXpdRqAtOj70qhBJPeMwmgLomV9iFxr5RoqUgKchZ
RG2J2rkqmPm6vr7K-
```

```
cm7ibD1IBpdQJCcVDuoHYsFgV4mp9vgOfn9osSDp2TGikwNz4Az95e81xnjVUmzh-
NjDsEpw71DH92iHV_xt2sTwtzftS49LpPW2LjrV0evtNBP_t_RfskdArt5VSv25eORl7zScqfe1CiM
kcVbf2UqACQjo3LbkpfN26HAioO2oH0ECPiRzT0Xyh-KwFutJLS9Xgghyw-
LD9kPKcE_xbbJ9Y4Rqajh7WdPYuB0Jd9DPVrslmzK-F6cgHHYoZEv0SvLQi-
PO0rpDrcjOEQQ
```

3. Log in to the MTC web console.
4. In the MTC web console, click **Clusters**.
5. Click **Add cluster**.
6. Fill in the following fields:
   - **Cluster name**: The cluster name can contain lower-case letters (**a-z**) and numbers (**0-9**). It must not contain spaces or international characters.
   - **URL**: Specify the API server URL, for example, **https://<www.example.com>:8443**.
   - **Service account token**: Paste the **migration-controller** service account token.
   - **Exposed route host to image registry**: If you are using direct image migration, specify the exposed route to the image registry of the source cluster.
     To create the route, run the following command:
     - For OpenShift Container Platform 3:

       ```
       $ oc create route passthrough --service=docker-registry --port=5000 -n default
       ```

     - For OpenShift Container Platform 4:

       ```
       $ oc create route passthrough --service=image-registry --port=5000 -n openshift-
       image-registry
       ```

   - **Azure cluster**: You must select this option if you use Azure snapshots to copy your data.
     - **Azure resource group**: This field is displayed if **Azure cluster** is selected. Specify the Azure resource group.
       When an OpenShift Container Platform cluster is created on Microsoft Azure, an Azure Resource Group is created to contain all resources associated with the cluster. In the Azure CLI, you can display all resource groups by issuing the following command:

       ```
       $ az group list
       ```

       ResourceGroups associated with OpenShift Container Platform clusters are tagged, where **sample-rg-name** is the value you would extract and supply to the UI:

       ```
       {
         "id": "/subscriptions/...//resourceGroups/sample-rg-name",
         "location": "centralus",
         "name": "...",
         "properties": {
           "provisioningState": "Succeeded"
         },
         "tags": {
       ```

```
    "kubernetes.io_cluster.sample-ld57c": "owned",
    "openshift_creationDate": "2019-10-25T23:28:57.988208+00:00"
  },
  "type": "Microsoft.Resources/resourceGroups"
},
```

This information is also available from the Azure Portal in the **Resource groups** blade.

- **Require SSL verification**: Optional: Select this option to verify the Secure Socket Layer (SSL) connection to the cluster.
- **CA bundle file**: This field is displayed if **Require SSL verification** is selected. If you created a custom CA certificate bundle file for self-signed certificates, click **Browse**, select the CA bundle file, and upload it.

7. Click **Add cluster**.
   The cluster appears in the **Clusters** list.

9.2.3. Adding a replication repository to the MTC web console

You can add an object storage as a replication repository to the Migration Toolkit for Containers (MTC) web console.

MTC supports the following storage providers:

- Amazon Web Services (AWS) S3
- Multi-Cloud Object Gateway (MCG)
- Generic S3 object storage, for example, Minio or Ceph S3
- Google Cloud Provider
- Microsoft Azure Blob

Prerequisites

- You must configure the object storage as a replication repository.

Procedure

1. In the MTC web console, click **Replication repositories**.
2. Click **Add repository**.
3. Select a **Storage provider type** and fill in the following fields:
   - **AWS** for S3 providers, including AWS and MCG:
     - **Replication repository name**: Specify the replication repository name in the MTC web console.
     - **S3 bucket name**: Specify the name of the S3 bucket.
     - **S3 bucket region**: Specify the S3 bucket region. **Required** for AWS S3. **Optional** for some S3 providers. Check the product documentation of your S3 provider for expected values.

- **S3 endpoint**: Specify the URL of the S3 service, not the bucket, for example, **https://<s3-storage.apps.cluster.com>**. **Required** for a generic S3 provider. You must use the **https://** prefix.
- **S3 provider access key**: Specify the **<AWS_SECRET_ACCESS_KEY>** for AWS or the S3 provider access key for MCG and other S3 providers.
- **S3 provider secret access key**: Specify the **<AWS_ACCESS_KEY_ID>** for AWS or the S3 provider secret access key for MCG and other S3 providers.
- **Require SSL verification**: Clear this checkbox if you are using a generic S3 provider.
- If you created a custom CA certificate bundle for self-signed certificates, click **Browse** and browse to the Base64-encoded file.

- **Google Cloud**:
  - **Replication repository name**: Specify the replication repository name in the MTC web console.
  - **Google Cloud bucket name**: Specify the name of the Google Cloud bucket.
  - **Google Cloud credential JSON blob**: Specify the string in the **credentials-velero** file.
- **Azure**:
  - **Replication repository name**: Specify the replication repository name in the MTC web console.
  - **Azure resource group**: Specify the resource group of the Azure Blob storage.
  - **Azure storage account name**: Specify the Azure Blob storage account name.
  - **Azure credentials - INI file contents** Specify the string in the **credentials-velero** file.

4. Click **Add repository** and wait for connection validation.
5. Click **Close**.
   The new repository appears in the **Replication repositories** list.

9.2.4. Creating a migration plan in the MTC web console

You can create a migration plan in the Migration Toolkit for Containers (MTC) web console.

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges on all clusters.
- You must ensure that the same MTC version is installed on all clusters.
- You must add the clusters and the replication repository to the MTC web console.
- If you want to use the *move* data copy method to migrate a persistent volume (PV), the source and target clusters must have uninterrupted network access to the remote volume.

- If you want to use direct image migration, you must specify the exposed route to the image registry of the source cluster. This can be done by using the MTC web console or by updating the **MigCluster** custom resource manifest.

Procedure

1. In the MTC web console, click **Migration plans**.
2. Click **Add migration plan**.
3. Enter the **Plan name**.
   The migration plan name must not exceed 253 lower-case alphanumeric characters (**a-z, 0-9**) and must not contain spaces or underscores (_).
4. Select a **Source cluster**, a **Target cluster**, and a **Repository**.
5. Click **Next**.
6. Select the projects for migration.
7. Optional: Click the edit icon beside a project to change the target namespace.

WARNING

Migration Toolkit for Containers 1.8.6 and later versions do not support multiple migration plans for a single namespace.

8. Click **Next**.
9. Select a **Migration type** for each PV:
   - The **Copy** option copies the data from the PV of a source cluster to the replication repository and then restores the data on a newly created PV, with similar characteristics, in the target cluster.
   - The **Move** option unmounts a remote volume, for example, NFS, from the source cluster, creates a PV resource on the target cluster pointing to the remote volume, and then mounts the remote volume on the target cluster. Applications running on the target cluster use the same remote volume that the source cluster was using.
10. Click **Next**.
11. Select a **Copy method** for each PV:
    - **Snapshot copy** backs up and restores data using the cloud provider's snapshot functionality. It is significantly faster than **Filesystem copy**.
    - **Filesystem copy** backs up the files on the source cluster and restores them on the target cluster.
      The file system copy method is required for direct volume migration.

12. You can select **Verify copy** to verify data migrated with **Filesystem copy**. Data is verified by generating a checksum for each source file and checking the checksum after restoration. Data verification significantly reduces performance.

13. Select a **Target storage class**.
    If you selected **Filesystem copy**, you can change the target storage class.

14. Click **Next**.

15. On the **Migration options** page, the **Direct image migration** option is selected if you specified an exposed image registry route for the source cluster. The **Direct PV migration** option is selected if you are migrating data with **Filesystem copy**.
    The direct migration options copy images and files directly from the source cluster to the target cluster. This option is much faster than copying images and files from the source cluster to the replication repository and then from the replication repository to the target cluster.

16. Click **Next**.

17. Optional: Click **Add Hook** to add a hook to the migration plan.
    A hook runs custom code. You can add up to four hooks to a single migration plan. Each hook runs during a different migration step.

    a. Enter the name of the hook to display in the web console.

    b. If the hook is an Ansible playbook, select **Ansible playbook** and click **Browse** to upload the playbook or paste the contents of the playbook in the field.

    c. Optional: Specify an Ansible runtime image if you are not using the default hook image.

    d. If the hook is not an Ansible playbook, select **Custom container image** and specify the image name and path.
    A custom container image can include Ansible playbooks.

    e. Select **Source cluster** or **Target cluster**.

    f. Enter the **Service account name** and the **Service account namespace**

    g. Select the migration step for the hook:

    - **preBackup**: Before the application workload is backed up on the source cluster
    - **postBackup**: After the application workload is backed up on the source cluster
    - **preRestore**: Before the application workload is restored on the target cluster
    - **postRestore**: After the application workload is restored on the target cluster

    h. Click **Add**.

18. Click **Finish**.
    The migration plan is displayed in the **Migration plans** list.

Additional resources for persistent volume copy methods

- MTC file system copy method
- MTC snapshot copy method

9.2.5. Running a migration plan in the MTC web console

You can migrate applications and data with the migration plan you created in the Migration Toolkit for Containers (MTC) web console.

NOTE

During migration, MTC sets the reclaim policy of migrated persistent volumes (PVs) to **Retain** on the target cluster.

The **Backup** custom resource contains a **PVOriginalReclaimPolicy** annotation that indicates the original reclaim policy. You can manually restore the reclaim policy of the migrated PVs.

Prerequisites

The MTC web console must contain the following:

- Source cluster in a **Ready** state
- Target cluster in a **Ready** state
- Replication repository
- Valid migration plan

Procedure

1. Log in to the MTC web console and click **Migration plans**.
2. Click the Options menu next to a migration plan and select one of the following options under **Migration**:
   - **Stage** copies data from the source cluster to the target cluster without stopping the application.
   - **Cutover** stops the transactions on the source cluster and moves the resources to the target cluster.
     Optional: In the **Cutover migration** dialog, you can clear the **Halt transactions on the source cluster during migration** checkbox.
   - **State** copies selected persistent volume claims (PVCs).

     IMPORTANT

     Do not use state migration to migrate a namespace between clusters. Use stage or cutover migration instead.

     - Select one or more PVCs in the **State migration** dialog and click **Migrate**.
3. When the migration is complete, verify that the application migrated successfully in the OpenShift Container Platform web console:
   a. Click **Home → Projects**.

b. Click the migrated project to view its status.

c. In the **Routes** section, click **Location** to verify that the application is functioning, if applicable.

d. Click **Workloads → Pods** to verify that the pods are running in the migrated namespace.

e. Click **Storage → Persistent volumes** to verify that the migrated persistent volumes are correctly provisioned.

CHAPTER 10. ADVANCED MIGRATION OPTIONS

You can automate your migrations and modify the **MigPlan** and **MigrationController** custom resources in order to perform large-scale migrations and to improve performance.

10.1. TERMINOLOGY

Table 10.1. MTC terminology

Term	Definition
Source cluster	Cluster from which the applications are migrated.
Destination cluster[1]	Cluster to which the applications are migrated.
Replication repository	Object storage used for copying images, volumes, and Kubernetes objects during indirect migration or for Kubernetes objects during direct volume migration or direct image migration. The replication repository must be accessible to all clusters.
Host cluster	Cluster on which the **migration-controller** pod and the web console are running. The host cluster is usually the destination cluster but this is not required. The host cluster does not require an exposed registry route for direct image migration.
Remote cluster	A remote cluster is usually the source cluster but this is not required. A remote cluster requires a **Secret** custom resource that contains the **migration-controller** service account token. A remote cluster requires an exposed secure registry route for direct image migration.
Indirect migration	Images, volumes, and Kubernetes objects are copied from the source cluster to the replication repository and then from the replication repository to the destination cluster.
Direct volume migration	Persistent volumes are copied directly from the source cluster to the destination cluster.
Direct image migration	Images are copied directly from the source cluster to the destination cluster.
Stage migration	Data is copied to the destination cluster without stopping the application. Running a stage migration multiple times reduces the duration of the cutover migration.
Cutover migration	The application is stopped on the source cluster and its resources are migrated to the destination cluster.

Term	Definition
State migration	Application state is migrated by copying specific persistent volume claims to the destination cluster.
Rollback migration	Rollback migration rolls back a completed migration.

[1] Called the *target* cluster in the MTC web console.

10.2. MIGRATING APPLICATIONS BY USING THE COMMAND LINE

You can migrate applications with the MTC API by using the command-line interface (CLI) in order to automate the migration.

10.2.1. Migration prerequisites

- You must be logged in as a user with **cluster-admin** privileges on all clusters.

Direct image migration

- You must ensure that the secure OpenShift image registry of the source cluster is exposed.
- You must create a route to the exposed registry.

Direct volume migration

- If your clusters use proxies, you must configure an Stunnel TCP proxy.

Clusters

- The source cluster must be upgraded to the latest MTC z-stream release.
- The MTC version must be the same on all clusters.

Network

- The clusters have unrestricted network access to each other and to the replication repository.
- If you copy the persistent volumes with **move**, the clusters must have unrestricted network access to the remote volumes.
- You must enable the following ports on an OpenShift Container Platform 4 cluster:
  - **6443** (API server)
  - **443** (routes)
  - **53** (DNS)
- You must enable port **443** on the replication repository if you are using TLS.

Persistent volumes (PVs)

- The PVs must be valid.
- The PVs must be bound to persistent volume claims.
- If you use snapshots to copy the PVs, the following additional prerequisites apply:
  - The cloud provider must support snapshots.
  - The PVs must have the same cloud provider.
  - The PVs must be located in the same geographic region.
  - The PVs must have the same storage class.

10.2.2. Creating a registry route for direct image migration

For direct image migration, you must create a route to the exposed OpenShift image registry on all remote clusters.

Prerequisites

- The OpenShift image registry must be exposed to external traffic on all remote clusters. The OpenShift Container Platform 4 registry is exposed by default.

Procedure

- To create a route to an OpenShift Container Platform 4 registry, run the following command:

```
$ oc create route passthrough --service=image-registry -n openshift-image-registry
```

10.2.3. Proxy configuration

For OpenShift Container Platform 4.1 and earlier versions, you must configure proxies in the **MigrationController** custom resource (CR) manifest after you install the Migration Toolkit for Containers Operator because these versions do not support a cluster-wide **proxy** object.

For OpenShift Container Platform 4.2 to 4.14, the MTC inherits the cluster-wide proxy settings. You can change the proxy parameters if you want to override the cluster-wide proxy settings.

10.2.3.1. Direct volume migration

Direct Volume Migration (DVM) was introduced in MTC 1.4.2. DVM supports only one proxy. The source cluster cannot access the route of the target cluster if the target cluster is also behind a proxy.

If you want to perform a DVM from a source cluster behind a proxy, you must configure a TCP proxy that works at the transport layer and forwards the SSL connections transparently without decrypting and re-encrypting them with their own SSL certificates. A Stunnel proxy is an example of such a proxy.

10.2.3.1.1. TCP proxy setup for DVM

You can set up a direct connection between the source and the target cluster through a TCP proxy and configure the **stunnel_tcp_proxy** variable in the **MigrationController** CR to use the proxy:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigrationController
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  stunnel_tcp_proxy: http://username:password@ip:port
```

Direct volume migration (DVM) supports only basic authentication for the proxy. Moreover, DVM works only from behind proxies that can tunnel a TCP connection transparently. HTTP/HTTPS proxies in man-in-the-middle mode do not work. The existing cluster-wide proxies might not support this behavior. As a result, the proxy settings for DVM are intentionally kept different from the usual proxy configuration in MTC.

10.2.3.1.2. Why use a TCP proxy instead of an HTTP/HTTPS proxy?

You can enable DVM by running Rsync between the source and the target cluster over an OpenShift route. Traffic is encrypted using Stunnel, a TCP proxy. The Stunnel running on the source cluster initiates a TLS connection with the target Stunnel and transfers data over an encrypted channel.

Cluster-wide HTTP/HTTPS proxies in OpenShift are usually configured in man-in-the-middle mode where they negotiate their own TLS session with the outside servers. However, this does not work with Stunnel. Stunnel requires that its TLS session be untouched by the proxy, essentially making the proxy a transparent tunnel which simply forwards the TCP connection as-is. Therefore, you must use a TCP proxy.

10.2.3.1.3. Known issue

Migration fails with error Upgrade request required

The migration Controller uses the SPDY protocol to execute commands within remote pods. If the remote cluster is behind a proxy or a firewall that does not support the SPDY protocol, the migration controller fails to execute remote commands. The migration fails with the error message **Upgrade request required**. Workaround: Use a proxy that supports the SPDY protocol.

In addition to supporting the SPDY protocol, the proxy or firewall also must pass the **Upgrade** HTTP header to the API server. The client uses this header to open a websocket connection with the API server. If the **Upgrade** header is blocked by the proxy or firewall, the migration fails with the error message **Upgrade request required**. Workaround: Ensure that the proxy forwards the **Upgrade** header.

10.2.3.2. Tuning network policies for migrations

OpenShift supports restricting traffic to or from pods using *NetworkPolicy* or *EgressFirewalls* based on the network plugin used by the cluster. If any of the source namespaces involved in a migration use such mechanisms to restrict network traffic to pods, the restrictions might inadvertently stop traffic to Rsync pods during migration.

Rsync pods running on both the source and the target clusters must connect to each other over an OpenShift Route. Existing *NetworkPolicy* or *EgressNetworkPolicy* objects can be configured to automatically exempt Rsync pods from these traffic restrictions.

10.2.3.2.1. NetworkPolicy configuration

10.2.3.2.1.1. Egress traffic from Rsync pods

You can use the unique labels of Rsync pods to allow egress traffic to pass from them if the **NetworkPolicy** configuration in the source or destination namespaces blocks this type of traffic. The following policy allows **all** egress traffic from Rsync pods in the namespace:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-egress-from-rsync-pods
spec:
  podSelector:
    matchLabels:
      owner: directvolumemigration
      app: directvolumemigration-rsync-transfer
  egress:
  - {}
  policyTypes:
  - Egress
```

10.2.3.2.1.2. Ingress traffic to Rsync pods

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-egress-from-rsync-pods
spec:
  podSelector:
    matchLabels:
      owner: directvolumemigration
      app: directvolumemigration-rsync-transfer
  ingress:
  - {}
  policyTypes:
  - Ingress
```

10.2.3.2.2. EgressNetworkPolicy configuration

The **EgressNetworkPolicy** object or *Egress Firewalls* are OpenShift constructs designed to block egress traffic leaving the cluster.

Unlike the **NetworkPolicy** object, the Egress Firewall works at a project level because it applies to all pods in the namespace. Therefore, the unique labels of Rsync pods do not exempt only Rsync pods from the restrictions. However, you can add the CIDR ranges of the source or target cluster to the *Allow* rule of the policy so that a direct connection can be setup between two clusters.

Based on which cluster the Egress Firewall is present in, you can add the CIDR range of the other cluster to allow egress traffic between the two:

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: test-egress-policy
  namespace: <namespace>
```

```
spec:
  egress:
  - to:
      cidrSelector: <cidr_of_source_or_target_cluster>
    type: Deny
```

10.2.3.2.3. Choosing alternate endpoints for data transfer

By default, DVM uses an OpenShift Container Platform route as an endpoint to transfer PV data to destination clusters. You can choose another type of supported endpoint, if cluster topologies allow.

For each cluster, you can configure an endpoint by setting the **rsync_endpoint_type** variable on the appropriate **destination** cluster in your **MigrationController** CR:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigrationController
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  [...]
  rsync_endpoint_type: [NodePort|ClusterIP|Route]
```

10.2.3.2.4. Configuring supplemental groups for Rsync pods

When your PVCs use a shared storage, you can configure the access to that storage by adding supplemental groups to Rsync pod definitions in order for the pods to allow access:

Table 10.2. Supplementary groups for Rsync pods

Variable	Type	Default	Description
src_supplemental_groups	string	Not set	Comma-separated list of supplemental groups for source Rsync pods
target_supplemental_groups	string	Not set	Comma-separated list of supplemental groups for target Rsync pods

Example usage

The **MigrationController** CR can be updated to set values for these supplemental groups:

```
spec:
  src_supplemental_groups: "1000,2000"
  target_supplemental_groups: "2000,3000"
```

10.2.3.3. Configuring proxies

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges on all clusters.

Procedure

1. Get the **MigrationController** CR manifest:

```
$ oc get migrationcontroller <migration_controller> -n openshift-migration
```

2. Update the proxy parameters:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigrationController
metadata:
  name: <migration_controller>
  namespace: openshift-migration
...
spec:
  stunnel_tcp_proxy: http://<username>:<password>@<ip>:<port> 1
  noProxy: example.com 2
```

1. Stunnel proxy URL for direct volume migration.
2. Comma-separated list of destination domain names, domains, IP addresses, or other network CIDRs to exclude proxying.

   Preface a domain with **.** to match subdomains only. For example, **.y.com** matches **x.y.com**, but not **y.com**. Use ***** to bypass proxy for all destinations. If you scale up workers that are not included in the network defined by the **networking.machineNetwork[].cidr** field from the installation configuration, you must add them to this list to prevent connection issues.

   This field is ignored if neither the **httpProxy** nor the **httpsProxy** field is set.

3. Save the manifest as **migration-controller.yaml**.

4. Apply the updated manifest:

```
$ oc replace -f migration-controller.yaml -n openshift-migration
```

10.2.4. Migrating an application by using the MTC API

You can migrate an application from the command line by using the Migration Toolkit for Containers (MTC) API.

Procedure

1. Create a **MigCluster** CR manifest for the host cluster:

```
$ cat << EOF | oc apply -f -
apiVersion: migration.openshift.io/v1alpha1
kind: MigCluster
metadata:
  name: <host_cluster>
  namespace: openshift-migration
```

```
spec:
  isHostCluster: true
EOF
```

2. Create a **Secret** object manifest for each remote cluster:

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: <cluster_secret>
  namespace: openshift-config
type: Opaque
data:
  saToken: <sa_token> (1)
EOF
```

(1) Specify the base64-encoded **migration-controller** service account (SA) token of the remote cluster. You can obtain the token by running the following command:

```
$ oc sa get-token migration-controller -n openshift-migration | base64 -w 0
```

3. Create a **MigCluster** CR manifest for each remote cluster:

```
$ cat << EOF | oc apply -f -
apiVersion: migration.openshift.io/v1alpha1
kind: MigCluster
metadata:
  name: <remote_cluster> (1)
  namespace: openshift-migration
spec:
  exposedRegistryPath: <exposed_registry_route> (2)
  insecure: false (3)
  isHostCluster: false
  serviceAccountSecretRef:
    name: <remote_cluster_secret> (4)
    namespace: openshift-config
  url: <remote_cluster_url> (5)
EOF
```

(1) Specify the **Cluster** CR of the remote cluster.

(2) Optional: For direct image migration, specify the exposed registry route.

(3) SSL verification is enabled if **false**. CA certificates are not required or checked if **true**.

(4) Specify the **Secret** object of the remote cluster.

(5) Specify the URL of the remote cluster.

4. Verify that all clusters are in a **Ready** state:

```
$ oc describe MigCluster <cluster>
```

5. Create a **Secret** object manifest for the replication repository:

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Secret
metadata:
  namespace: openshift-config
  name: <migstorage_creds>
type: Opaque
data:
  aws-access-key-id: <key_id_base64> 1
  aws-secret-access-key: <secret_key_base64> 2
EOF
```

1. Specify the key ID in base64 format.
2. Specify the secret key in base64 format.

AWS credentials are base64-encoded by default. For other storage providers, you must encode your credentials by running the following command with each key:

```
$ echo -n "<key>" | base64 -w 0 1
```

1. Specify the key ID or the secret key. Both keys must be base64-encoded.

6. Create a **MigStorage** CR manifest for the replication repository:

```
$ cat << EOF | oc apply -f -
apiVersion: migration.openshift.io/v1alpha1
kind: MigStorage
metadata:
  name: <migstorage>
  namespace: openshift-migration
spec:
  backupStorageConfig:
    awsBucketName: <bucket> 1
    credsSecretRef:
      name: <storage_secret> 2
      namespace: openshift-config
  backupStorageProvider: <storage_provider> 3
  volumeSnapshotConfig:
    credsSecretRef:
      name: <storage_secret> 4
      namespace: openshift-config
  volumeSnapshotProvider: <storage_provider> 5
EOF
```

1. Specify the bucket name.
2. Specify the **Secrets** CR of the object storage. You must ensure that the credentials stored

3. Specify the storage provider.
4. Optional: If you are copying data by using snapshots, specify the **Secrets** CR of the object storage. You must ensure that the credentials stored in the **Secrets** CR of the object storage are correct.
5. Optional: If you are copying data by using snapshots, specify the storage provider.

7. Verify that the **MigStorage** CR is in a **Ready** state:

```
$ oc describe migstorage <migstorage>
```

8. Create a **MigPlan** CR manifest:

```
$ cat << EOF | oc apply -f -
apiVersion: migration.openshift.io/v1alpha1
kind: MigPlan
metadata:
  name: <migplan>
  namespace: openshift-migration
spec:
  destMigClusterRef:
    name: <host_cluster>
    namespace: openshift-migration
  indirectImageMigration: true 1
  indirectVolumeMigration: true 2
  migStorageRef:
    name: <migstorage> 3
    namespace: openshift-migration
  namespaces:
    - <source_namespace_1> 4
    - <source_namespace_2>
    - <source_namespace_3>:<destination_namespace> 5
  srcMigClusterRef:
    name: <remote_cluster> 6
    namespace: openshift-migration
EOF
```

1. Direct image migration is enabled if **false**.
2. Direct volume migration is enabled if **false**.
3. Specify the name of the **MigStorage** CR instance.
4. Specify one or more source namespaces. By default, the destination namespace has the same name.
5. Specify a destination namespace if it is different from the source namespace.
6. Specify the name of the source cluster **MigCluster** instance.

9. Verify that the **MigPlan** instance is in a **Ready** state:

```
$ oc describe migplan <migplan> -n openshift-migration
```

10. Create a **MigMigration** CR manifest to start the migration defined in the **MigPlan** instance:

```
$ cat << EOF | oc apply -f -
apiVersion: migration.openshift.io/v1alpha1
kind: MigMigration
metadata:
  name: <migmigration>
  namespace: openshift-migration
spec:
  migPlanRef:
    name: <migplan> 1
    namespace: openshift-migration
  quiescePods: true 2
  stage: false 3
  rollback: false 4
EOF
```

1. Specify the **MigPlan** CR name.
2. The pods on the source cluster are stopped before migration if **true**.
3. A stage migration, which copies most of the data without stopping the application, is performed if **true**.
4. A completed migration is rolled back if **true**.

11. Verify the migration by watching the **MigMigration** CR progress:

```
$ oc watch migmigration <migmigration> -n openshift-migration
```

The output resembles the following:

Example output

```
Name:         c8b034c0-6567-11eb-9a4f-0bc004db0fbc
Namespace:    openshift-migration
Labels:       migration.openshift.io/migplan-name=django
Annotations:  openshift.io/touch: e99f9083-6567-11eb-8420-0a580a81020c
API Version:  migration.openshift.io/v1alpha1
Kind:         MigMigration
...
Spec:
  Mig Plan Ref:
    Name:         migplan
    Namespace:    openshift-migration
  Stage:          false
Status:
  Conditions:
    Category:              Advisory
    Last Transition Time:  2021-02-02T15:04:09Z
    Message:               Step: 19/47
    Reason:                InitialBackupCreated
```

```
    Status:             True
    Type:               Running
    Category:             Required
    Last Transition Time:  2021-02-02T15:03:19Z
    Message:              The migration is ready.
    Status:             True
    Type:               Ready
    Category:             Required
    Durable:             true
    Last Transition Time:  2021-02-02T15:04:05Z
    Message:              The migration registries are healthy.
    Status:             True
    Type:               RegistriesHealthy
  Itinerary:            Final
  Observed Digest:
7fae9d21f15979c71ddc7dd075cb97061895caac5b936d92fae967019ab616d5
  Phase:                InitialBackupCreated
  Pipeline:
    Completed:  2021-02-02T15:04:07Z
    Message:    Completed
    Name:       Prepare
    Started:    2021-02-02T15:03:18Z
    Message:    Waiting for initial Velero backup to complete.
    Name:       Backup
    Phase:      InitialBackupCreated
    Progress:
      Backup openshift-migration/c8b034c0-6567-11eb-9a4f-0bc004db0fbc-wpc44: 0 out of
estimated total of 0 objects backed up (5s)
    Started:        2021-02-02T15:04:07Z
    Message:        Not started
    Name:           StageBackup
    Message:        Not started
    Name:           StageRestore
    Message:        Not started
    Name:           DirectImage
    Message:        Not started
    Name:           DirectVolume
    Message:        Not started
    Name:           Restore
    Message:        Not started
    Name:           Cleanup
  Start Timestamp:  2021-02-02T15:03:18Z
Events:
  Type    Reason   Age                From                    Message
  ----    ------   ----               ----                    -------
  Normal  Running  57s                migmigration_controller  Step: 2/47
  Normal  Running  57s                migmigration_controller  Step: 3/47
  Normal  Running  57s (x3 over 57s)   migmigration_controller  Step: 4/47
  Normal  Running  54s                migmigration_controller  Step: 5/47
  Normal  Running  54s                migmigration_controller  Step: 6/47
  Normal  Running  52s (x2 over 53s)   migmigration_controller  Step: 7/47
  Normal  Running  51s (x2 over 51s)   migmigration_controller  Step: 8/47
  Normal  Ready    50s (x12 over 57s)  migmigration_controller  The migration is ready.
  Normal  Running  50s                migmigration_controller  Step: 9/47
  Normal  Running  50s                migmigration_controller  Step: 10/47
```

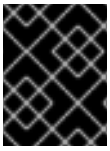
10.2.5. State migration

You can perform repeatable, state-only migrations by using Migration Toolkit for Containers (MTC) to migrate persistent volume claims (PVCs) that constitute an application's state. You migrate specified PVCs by excluding other PVCs from the migration plan. You can map the PVCs to ensure that the source and the target PVCs are synchronized. Persistent volume (PV) data is copied to the target cluster. The PV references are not moved, and the application pods continue to run on the source cluster.

State migration is specifically designed to be used in conjunction with external CD mechanisms, such as OpenShift Gitops. You can migrate application manifests using GitOps while migrating the state using MTC.

If you have a CI/CD pipeline, you can migrate stateless components by deploying them on the target cluster. Then you can migrate stateful components by using MTC.

You can perform a state migration between clusters or within the same cluster.



IMPORTANT

State migration migrates only the components that constitute an application's state. If you want to migrate an entire namespace, use stage or cutover migration.

Prerequisites

- The state of the application on the source cluster is persisted in **PersistentVolumes** provisioned through **PersistentVolumeClaims**.
- The manifests of the application are available in a central repository that is accessible from both the source and the target clusters.

Procedure

1. Migrate persistent volume data from the source to the target cluster.
   You can perform this step as many times as needed. The source application continues running.
2. Quiesce the source application.
   You can do this by setting the replicas of workload resources to **0**, either directly on the source cluster or by updating the manifests in GitHub and re-syncing the Argo CD application.
3. Clone application manifests to the target cluster.
   You can use Argo CD to clone the application manifests to the target cluster.
4. Migrate the remaining volume data from the source to the target cluster.
   Migrate any new data created by the application during the state migration process by performing a final data migration.
5. If the cloned application is in a quiesced state, unquiesce it.
6. Switch the DNS record to the target cluster to re-direct user traffic to the migrated application.

NOTE

MTC 1.6 cannot quiesce applications automatically when performing state migration. It can only migrate PV data. Therefore, you must use your CD mechanisms for quiescing or unquiescing applications.

MTC 1.7 introduces explicit Stage and Cutover flows. You can use staging to perform initial data transfers as many times as needed. Then you can perform a cutover, in which the source applications are quiesced automatically.

Additional resources

- See Excluding PVCs from migration to select PVCs for state migration.
- See Mapping PVCs to migrate source PV data to provisioned PVCs on the destination cluster.
- See Migrating Kubernetes objects to migrate the Kubernetes objects that constitute an application's state.

10.3. MIGRATION HOOKS

You can add up to four migration hooks to a single migration plan, with each hook running at a different phase of the migration. Migration hooks perform tasks such as customizing application quiescence, manually migrating unsupported data types, and updating applications after migration.

A migration hook runs on a source or a target cluster at one of the following migration steps:

- **PreBackup**: Before resources are backed up on the source cluster.
- **PostBackup**: After resources are backed up on the source cluster.
- **PreRestore**: Before resources are restored on the target cluster.
- **PostRestore**: After resources are restored on the target cluster.

You can create a hook by creating an Ansible playbook that runs with the default Ansible image or with a custom hook container.

Ansible playbook

The Ansible playbook is mounted on a hook container as a config map. The hook container runs as a job, using the cluster, service account, and namespace specified in the **MigPlan** custom resource. The job continues to run until it reaches the default limit of 6 retries or a successful completion. This continues even if the initial pod is evicted or killed.

The default Ansible runtime image is **registry.redhat.io/rhmtc/openshift-migration-hook-runner-rhel7:1.8**. This image is based on the Ansible Runner image and includes **python-openshift** for Ansible Kubernetes resources and an updated **oc** binary.

Custom hook container

You can use a custom hook container instead of the default Ansible image.

10.3.1. Writing an Ansible playbook for a migration hook

You can write an Ansible playbook to use as a migration hook. The hook is added to a migration plan by using the MTC web console or by specifying values for the **spec.hooks** parameters in the **MigPlan** custom resource (CR) manifest.

The Ansible playbook is mounted onto a hook container as a config map. The hook container runs as a job, using the cluster, service account, and namespace specified in the **MigPlan** CR. The hook container uses a specified service account token so that the tasks do not require authentication before they run in the cluster.

10.3.1.1. Ansible modules

You can use the Ansible **shell** module to run **oc** commands.

Example shell module

```
- hosts: localhost
  gather_facts: false
  tasks:
  - name: get pod name
    shell: oc get po --all-namespaces
```

You can use **kubernetes.core** modules, such as **k8s_info**, to interact with Kubernetes resources.

Example k8s_facts module

```
- hosts: localhost
  gather_facts: false
  tasks:
  - name: Get pod
    k8s_info:
      kind: pods
      api: v1
      namespace: openshift-migration
      name: "{{ lookup( 'env', 'HOSTNAME') }}"
    register: pods

  - name: Print pod name
    debug:
      msg: "{{ pods.resources[0].metadata.name }}"
```

You can use the **fail** module to produce a non-zero exit status in cases where a non-zero exit status would not normally be produced, ensuring that the success or failure of a hook is detected. Hooks run as jobs and the success or failure status of a hook is based on the exit status of the job container.

Example fail module

```
- hosts: localhost
  gather_facts: false
  tasks:
  - name: Set a boolean
    set_fact:
      do_fail: true

  - name: "fail"
```

```
    fail:
      msg: "Cause a failure"
    when: do_fail
```

10.3.1.2. Environment variables

The **MigPlan** CR name and migration namespaces are passed as environment variables to the hook container. These variables are accessed by using the **lookup** plugin.

Example environment variables

```
- hosts: localhost
  gather_facts: false
  tasks:
  - set_fact:
      namespaces: "{{ (lookup( 'env', 'MIGRATION_NAMESPACES')).split(',') }}"

  - debug:
      msg: "{{ item }}"
    with_items: "{{ namespaces }}"

  - debug:
      msg: "{{ lookup( 'env', 'MIGRATION_PLAN_NAME') }}"
```

10.4. MIGRATION PLAN OPTIONS

You can exclude, edit, and map components in the **MigPlan** custom resource (CR).

10.4.1. Excluding resources

You can exclude resources, for example, image streams, persistent volumes (PVs), or subscriptions, from a Migration Toolkit for Containers (MTC) migration plan to reduce the resource load for migration or to migrate images or PVs with a different tool.

By default, the MTC excludes service catalog resources and Operator Lifecycle Manager (OLM) resources from migration. These resources are parts of the service catalog API group and the OLM API group, neither of which is supported for migration at this time.

Procedure

1. Edit the **MigrationController** custom resource manifest:

   ```
   $ oc edit migrationcontroller <migration_controller> -n openshift-migration
   ```

2. Update the **spec** section by adding parameters to exclude specific resources. For those resources that do not have their own exclusion parameters, add the **additional_excluded_resources** parameter:

   ```
   apiVersion: migration.openshift.io/v1alpha1
   kind: MigrationController
   metadata:
     name: migration-controller
     namespace: openshift-migration
   ```

```
spec:
  disable_image_migration: true (1)
  disable_pv_migration: true (2)
  additional_excluded_resources: (3)
  - resource1
  - resource2
  ...
```

(1) Add **disable_image_migration: true** to exclude image streams from the migration. **imagestreams** is added to the **excluded_resources** list in **main.yml** when the **MigrationController** pod restarts.

(2) Add **disable_pv_migration: true** to exclude PVs from the migration plan. **persistentvolumes** and **persistentvolumeclaims** are added to the **excluded_resources** list in **main.yml** when the **MigrationController** pod restarts. Disabling PV migration also disables PV discovery when you create the migration plan.

(3) You can add OpenShift Container Platform resources that you want to exclude to the **additional_excluded_resources** list.

3. Wait two minutes for the **MigrationController** pod to restart so that the changes are applied.
4. Verify that the resource is excluded:

```
$ oc get deployment -n openshift-migration migration-controller -o yaml | grep
EXCLUDED_RESOURCES -A1
```

The output contains the excluded resources:

Example output

```
name: EXCLUDED_RESOURCES
value:
resource1,resource2,imagetags,templateinstances,clusterserviceversions,packagemanifests,sul
scriptions,servicebrokers,servicebindings,serviceclasses,serviceinstances,serviceplans,imagest
ams,persistentvolumes,persistentvolumeclaims
```

10.4.2. Mapping namespaces

If you map namespaces in the **MigPlan** custom resource (CR), you must ensure that the namespaces are not duplicated on the source or the destination clusters because the UID and GID ranges of the namespaces are copied during migration.

Two source namespaces mapped to the same destination namespace

```
spec:
  namespaces:
    - namespace_2
    - namespace_1:namespace_2
```

If you want the source namespace to be mapped to a namespace of the same name, you do not need to create a mapping. By default, a source namespace and a target namespace have the same name.

Incorrect namespace mapping

```
spec:
  namespaces:
    - namespace_1:namespace_1
```

Correct namespace reference

```
spec:
  namespaces:
    - namespace_1
```

10.4.3. Excluding persistent volume claims

You select persistent volume claims (PVCs) for state migration by excluding the PVCs that you do not want to migrate. You exclude PVCs by setting the **spec.persistentVolumes.pvc.selection.action** parameter of the **MigPlan** custom resource (CR) after the persistent volumes (PVs) have been discovered.

Prerequisites

- **MigPlan** CR is in a **Ready** state.

Procedure

- Add the **spec.persistentVolumes.pvc.selection.action** parameter to the **MigPlan** CR and set it to **skip**:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigPlan
metadata:
  name: <migplan>
  namespace: openshift-migration
spec:
...
  persistentVolumes:
  - capacity: 10Gi
    name: <pv_name>
    pvc:
...
    selection:
      action: skip
```

10.4.4. Mapping persistent volume claims

You can migrate persistent volume (PV) data from the source cluster to persistent volume claims (PVCs) that are already provisioned in the destination cluster in the **MigPlan** CR by mapping the PVCs. This mapping ensures that the destination PVCs of migrated applications are synchronized with the source PVCs.

You map PVCs by updating the **spec.persistentVolumes.pvc.name** parameter in the **MigPlan** custom resource (CR) after the PVs have been discovered.

Prerequisites

- **MigPlan** CR is in a **Ready** state.

Procedure

- Update the **spec.persistentVolumes.pvc.name** parameter in the **MigPlan** CR:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigPlan
metadata:
  name: <migplan>
  namespace: openshift-migration
spec:
...
  persistentVolumes:
  - capacity: 10Gi
    name: <pv_name>
    pvc:
      name: <source_pvc>:<destination_pvc> 1
```

1 Specify the PVC on the source cluster and the PVC on the destination cluster. If the destination PVC does not exist, it will be created. You can use this mapping to change the PVC name during migration.

10.4.5. Editing persistent volume attributes

After you create a **MigPlan** custom resource (CR), the **MigrationController** CR discovers the persistent volumes (PVs). The **spec.persistentVolumes** block and the **status.destStorageClasses** block are added to the **MigPlan** CR.

You can edit the values in the **spec.persistentVolumes.selection** block. If you change values outside the **spec.persistentVolumes.selection** block, the values are overwritten when the **MigPlan** CR is reconciled by the **MigrationController** CR.

NOTE

The default value for the **spec.persistentVolumes.selection.storageClass** parameter is determined by the following logic:

1. If the source cluster PV is Gluster or NFS, the default is either **cephfs**, for **accessMode: ReadWriteMany**, or **cephrbd**, for **accessMode: ReadWriteOnce**.
2. If the PV is neither Gluster nor NFS *or* if **cephfs** or **cephrbd** are not available, the default is a storage class for the same provisioner.
3. If a storage class for the same provisioner is not available, the default is the default storage class of the destination cluster.

You can change the **storageClass** value to the value of any **name** parameter in the **status.destStorageClasses** block of the **MigPlan** CR.

If the **storageClass** value is empty, the PV will have no storage class after migration. This option is appropriate if, for example, you want to move the PV to an NFS volume on the destination cluster.

Prerequisites

- **MigPlan** CR is in a **Ready** state.

Procedure

- Edit the **spec.persistentVolumes.selection** values in the **MigPlan** CR:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigPlan
metadata:
  name: <migplan>
  namespace: openshift-migration
spec:
  persistentVolumes:
  - capacity: 10Gi
    name: pvc-095a6559-b27f-11eb-b27f-021bddcaf6e4
    proposedCapacity: 10Gi
    pvc:
      accessModes:
      - ReadWriteMany
      hasReference: true
      name: mysql
      namespace: mysql-persistent
    selection:
      action: <copy> (1)
      copyMethod: <filesystem> (2)
      verify: true (3)
      storageClass: <gp2> (4)
      accessMode: <ReadWriteMany> (5)
    storageClass: cephfs
```

1. Allowed values are **move**, **copy**, and **skip**. If only one action is supported, the default value is the supported action. If multiple actions are supported, the default value is **copy**.
2. Allowed values are **snapshot** and **filesystem**. Default value is **filesystem**.
3. The **verify** parameter is displayed if you select the verification option for file system copy in the MTC web console. You can set it to **false**.
4. You can change the default value to the value of any **name** parameter in the **status.destStorageClasses** block of the **MigPlan** CR. If no value is specified, the PV will have no storage class after migration.
5. Allowed values are **ReadWriteOnce** and **ReadWriteMany**. If this value is not specified, the default is the access mode of the source cluster PVC. You can only edit the access mode in the **MigPlan** CR. You cannot edit it by using the MTC web console.

10.4.6. Converting storage classes in the MTC web console

You can convert the storage class of a persistent volume (PV) by migrating it within the same cluster. To do so, you must create and run a migration plan in the Migration Toolkit for Containers (MTC) web console.

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges on the cluster on which MTC is running.
- You must add the cluster to the MTC web console.

Procedure

1. In the left-side navigation pane of the OpenShift Container Platform web console, click **Projects**.
2. In the list of projects, click your project.
   The **Project details** page opens.
3. Click the **DeploymentConfig** name. Note the name of its running pod.
4. Open the YAML tab of the project. Find the PVs and note the names of their corresponding persistent volume claims (PVCs).
5. In the MTC web console, click **Migration plans**.
6. Click **Add migration plan**.
7. Enter the **Plan name**.
   The migration plan name must contain 3 to 63 lower-case alphanumeric characters (**a-z, 0-9**) and must not contain spaces or underscores (_).
8. From the **Migration type** menu, select **Storage class conversion**
9. From the **Source cluster** list, select the desired cluster for storage class conversion.

10. Click **Next**.
The **Namespaces** page opens.

11. Select the required project.

12. Click **Next**.
The **Persistent volumes** page opens. The page displays the PVs in the project, all selected by default.

13. For each PV, select the desired target storage class.

14. Click **Next**.
The wizard validates the new migration plan and shows that it is ready.

15. Click **Close**.
The new plan appears on the **Migration plans** page.

16. To start the conversion, click the options menu of the new plan.
Under **Migrations**, two options are displayed, **Stage** and **Cutover**.

NOTE

Cutover migration updates PVC references in the applications.

Stage migration does not update PVC references in the applications.

17. Select the desired option.
Depending on which option you selected, the **Stage migration** or **Cutover migration** notification appears.

18. Click **Migrate**.
Depending on which option you selected, the **Stage started** or **Cutover started** message appears.

19. To see the status of the current migration, click the number in the **Migrations** column.
The **Migrations** page opens.

20. To see more details on the current migration and monitor its progress, select the migration from the **Type** column.
The **Migration details** page opens. When the migration progresses to the DirectVolume step and the status of the step becomes **Running Rsync Pods to migrate Persistent Volume data**, you can click **View details** and see the detailed status of the copies.

21. In the breadcrumb bar, click **Stage** or **Cutover** and wait for all steps to complete.

22. Open the **PersistentVolumeClaims** tab of the OpenShift Container Platform web console.
You can see new PVCs with the names of the initial PVCs but ending in **new**, which are using the target storage class.

23. In the left-side navigation pane, click **Pods**. See that the pod of your project is running again.

Additional resources

- For details about the **move** and **copy** actions, see MTC workflow.
- For details about the **skip** action, see Excluding PVCs from migration .

- For details about the file system and snapshot copy methods, see About data copy methods.

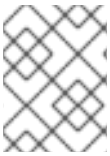
10.4.7. Performing a state migration of Kubernetes objects by using the MTC API

After you migrate all the PV data, you can use the Migration Toolkit for Containers (MTC) API to perform a one-time state migration of Kubernetes objects that constitute an application.

You do this by configuring **MigPlan** custom resource (CR) fields to provide a list of Kubernetes resources with an additional label selector to further filter those resources, and then performing a migration by creating a **MigMigration** CR. The **MigPlan** resource is closed after the migration.

NOTE

Selecting Kubernetes resources is an API-only feature. You must update the **MigPlan** CR and create a **MigMigration** CR for it by using the CLI. The MTC web console does not support migrating Kubernetes objects.



NOTE

After migration, the **closed** parameter of the **MigPlan** CR is set to **true**. You cannot create another **MigMigration** CR for this **MigPlan** CR.

You add Kubernetes objects to the **MigPlan** CR by using one of the following options:

- Adding the Kubernetes objects to the **includedResources** section. When the **includedResources** field is specified in the **MigPlan** CR, the plan takes a list of **group-kind** as input. Only resources present in the list are included in the migration.
- Adding the optional **labelSelector** parameter to filter the **includedResources** in the **MigPlan**. When this field is specified, only resources matching the label selector are included in the migration. For example, you can filter a list of **Secret** and **ConfigMap** resources by using the label **app: frontend** as a filter.

Procedure

1. Update the **MigPlan** CR to include Kubernetes resources and, optionally, to filter the included resources by adding the **labelSelector** parameter:
   a. To update the **MigPlan** CR to include Kubernetes resources:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigPlan
metadata:
  name: <migplan>
  namespace: openshift-migration
spec:
  includedResources:
  - kind: <kind> 1
    group: ""
  - kind: <kind>
    group: ""
```

1 Specify the Kubernetes object, for example, **Secret** or **ConfigMap**.

b. Optional: To filter the included resources by adding the **labelSelector** parameter:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigPlan
metadata:
  name: <migplan>
  namespace: openshift-migration
spec:
  includedResources:
  - kind: <kind> (1)
    group: ""
  - kind: <kind>
    group: ""
...
  labelSelector:
    matchLabels:
      <label> (2)
```

(1) Specify the Kubernetes object, for example, **Secret** or **ConfigMap**.

(2) Specify the label of the resources to migrate, for example, **app: frontend**.

2. Create a **MigMigration** CR to migrate the selected Kubernetes resources. Verify that the correct **MigPlan** is referenced in **migPlanRef**:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigMigration
metadata:
  generateName: <migplan>
  namespace: openshift-migration
spec:
  migPlanRef:
    name: <migplan>
    namespace: openshift-migration
  stage: false
```

10.5. MIGRATION CONTROLLER OPTIONS

You can edit migration plan limits, enable persistent volume resizing, or enable cached Kubernetes clients in the **MigrationController** custom resource (CR) for large migrations and improved performance.

10.5.1. Increasing limits for large migrations

You can increase the limits on migration objects and container resources for large migrations with the Migration Toolkit for Containers (MTC).

IMPORTANT

You must test these changes before you perform a migration in a production environment.

Procedure

1. Edit the **MigrationController** custom resource (CR) manifest:

```
$ oc edit migrationcontroller -n openshift-migration
```

2. Update the following parameters:

```
...
mig_controller_limits_cpu: "1" 1
mig_controller_limits_memory: "10Gi" 2
...
mig_controller_requests_cpu: "100m" 3
mig_controller_requests_memory: "350Mi" 4
...
mig_pv_limit: 100 5
mig_pod_limit: 100 6
mig_namespace_limit: 10 7
...
```

1. Specifies the number of CPUs available to the **MigrationController** CR.
2. Specifies the amount of memory available to the **MigrationController** CR.
3. Specifies the number of CPU units available for **MigrationController** CR requests. **100m** represents 0.1 CPU units (100 * 1e-3).
4. Specifies the amount of memory available for **MigrationController** CR requests.
5. Specifies the number of persistent volumes that can be migrated.
6. Specifies the number of pods that can be migrated.
7. Specifies the number of namespaces that can be migrated.

3. Create a migration plan that uses the updated parameters to verify the changes.
If your migration plan exceeds the **MigrationController** CR limits, the MTC console displays a warning message when you save the migration plan.

10.5.2. Enabling persistent volume resizing for direct volume migration

You can enable persistent volume (PV) resizing for direct volume migration to avoid running out of disk space on the destination cluster.

When the disk usage of a PV reaches a configured level, the **MigrationController** custom resource (CR) compares the requested storage capacity of a persistent volume claim (PVC) to its actual provisioned capacity. Then, it calculates the space required on the destination cluster.

A **pv_resizing_threshold** parameter determines when PV resizing is used. The default threshold is **3%**. This means that PV resizing occurs when the disk usage of a PV is more than **97%**. You can increase this threshold so that PV resizing occurs at a lower disk usage level.

PVC capacity is calculated according to the following criteria:

- If the requested storage capacity (**spec.resources.requests.storage**) of the PVC is not equal to its actual provisioned capacity (**status.capacity.storage**), the greater value is used.
- If a PV is provisioned through a PVC and then subsequently changed so that its PV and PVC capacities no longer match, the greater value is used.

Prerequisites

- The PVCs must be attached to one or more running pods so that the **MigrationController** CR can execute commands.

Procedure

1. Log in to the host cluster.
2. Enable PV resizing by patching the **MigrationController** CR:

```
$ oc patch migrationcontroller migration-controller -p '{"spec":
{"enable_dvm_pv_resizing":true}}' \ 1
  --type='merge' -n openshift-migration
```

1 Set the value to **false** to disable PV resizing.

3. Optional: Update the **pv_resizing_threshold** parameter to increase the threshold:

```
$ oc patch migrationcontroller migration-controller -p '{"spec":{"pv_resizing_threshold":41}}' \
1
  --type='merge' -n openshift-migration
```

1 The default value is **3**.

When the threshold is exceeded, the following status message is displayed in the **MigPlan** CR status:

```
status:
  conditions:
...
  - category: Warn
    durable: true
    lastTransitionTime: "2021-06-17T08:57:01Z"
    message: 'Capacity of the following volumes will be automatically adjusted to avoid disk
capacity issues in the target cluster:  [pvc-b800eb7b-cf3b-11eb-a3f7-0eae3e0555f3]'
    reason: Done
    status: "False"
    type: PvCapacityAdjustmentRequired
```

NOTE

For AWS gp2 storage, this message does not appear unless the **pv_resizing_threshold** is 42% or greater because of the way gp2 calculates volume usage and size. (BZ#1973148)

10.5.3. Enabling cached Kubernetes clients

You can enable cached Kubernetes clients in the **MigrationController** custom resource (CR) for improved performance during migration. The greatest performance benefit is displayed when migrating between clusters in different regions or with significant network latency.

NOTE

Delegated tasks, for example, Rsync backup for direct volume migration or Velero backup and restore, however, do not show improved performance with cached clients.

Cached clients require extra memory because the **MigrationController** CR caches all API resources that are required for interacting with **MigCluster** CRs. Requests that are normally sent to the API server are directed to the cache instead. The cache watches the API server for updates.

You can increase the memory limits and requests of the **MigrationController** CR if **OOMKilled** errors occur after you enable cached clients.

Procedure

1. Enable cached clients by running the following command:

```
$ oc -n openshift-migration patch migrationcontroller migration-controller --type=json --patch \
  '[{ "op": "replace", "path": "/spec/mig_controller_enable_cache", "value": true}]'
```

2. Optional: Increase the **MigrationController** CR memory limits by running the following command:

```
$ oc -n openshift-migration patch migrationcontroller migration-controller --type=json --patch \
  '[{ "op": "replace", "path": "/spec/mig_controller_limits_memory", "value": <10Gi>}]'
```

3. Optional: Increase the **MigrationController** CR memory requests by running the following command:

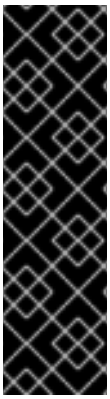
```
$ oc -n openshift-migration patch migrationcontroller migration-controller --type=json --patch \
  '[{ "op": "replace", "path": "/spec/mig_controller_requests_memory", "value": <350Mi>}]'
```

CHAPTER 11. MIGRATING VIRTUAL MACHINE STORAGE

You can migrate virtual machine (VM) storage *offline*, when you have the VM turned off, or *online*, when the VM is running.

For KubeVirt VMs, the primary use case for VM storage migration is if you want to migrate from one storage class to another. You might migrate VM storage for one of the following reasons:

- You are rebalancing between different storage providers.
- New storage is available that is better suited to the workload running inside the VM.



IMPORTANT

Virtual machine storage migration 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 Technology Preview Features Support Scope .

11.1. ABOUT THE VIRTUAL MACHINE STORAGE MIGRATION PROCESS

During the migration, a **MigMigration** resource is created indicating what type of migration is happening.

The types of migration are as follows:

- Stage: Stage migration copies data from the source cluster to the target cluster without stopping the application. You can run a stage migration multiple times to reduce the duration of the cutover migration.
- Rollback: Rolls back a completed migration.
- Cutover: Cutover migration stops the transactions on the source cluster and moves the resources to the target cluster.

The status of the **MigMigration** resource contains progress information about any live storage migrations.

Any offline migrations contain the **DirectVolumeMigrationProgress** status that shows the progress of the offline migration.

Each **MigMigration** creates a **DirectVolumeMigration** if the migration plan is a direct volume migration plan.

To perform a storage live migration, a direct volume migration is required.

11.2. SUPPORTED PERSISTENT VOLUME ACTIONS

Table 11.1. Supported persistent volume actions

Action	Supported	User-initiated steps

Copy	Yes	• Create a new persistent volume (PV) in the same namespace.
		• Copy data from the source PV to the target PV, and change the VM definition to point to the new PV.
		  ◦ If you have the **liveMigrate** flag set, the VM migrates live.
		  ◦ If you do have the **liveMigrate** flag set, the VM shuts down, the source PV contents are copied to the target PV, and the VM is started.

Move	No	This action is not supported.

11.3. PREREQUISITES

Before migrating virtual machine storage, you must install OpenShift Virtualization Operator.

To support storage live migration, you need to deploy OpenShift Virtualization version 4.17 or later. Earlier versions of OpenShift Virtualization do not support live storage migration.

You also need to configure **KubeVirt** to enable storage live migration according to the Configuring live migration.

In OpenShift Virtualization 4.17.0, not all the required feature gates are enabled. However, to use the storage live migration feature, you must enable the feature gate.

Enable the feature gate by running the following command:

```
$ oc annotate --overwrite -n openshift-cnv hco kubevirt-hyperconverged
kubevirt.kubevirt.io/jsonpatch='[ {"op": "add", "path":
"/spec/configuration/developerConfiguration/featureGates/-", "value": "VolumesUpdateStrategy"},
{"op": "add", "path": "/spec/configuration/developerConfiguration/featureGates/-", "value":
"VolumeMigration"} ]'
```

WARNING

Red Hat does not support clusters with the annotation enabling this feature gate.

Do not add this annotation in a production cluster, if you add that annotation you receive a cluster wide alert indicating that your cluster is no longer supported.

For more information about the deployments and custom resource definitions (CRDs) that the migration controller uses to manipulate the VMs, see Migration controller options.

NOTE

If the **mig-controller** pod starts before you install OpenShift Virtualization, the migration controller does not automatically see that you have the OpenShift Virtualization Custom Resource Definition (CRD) installed.

Restart the **mig-controller** pod in the **openshift-migration** namespace after installing OpenShift Virtualization.

The following table explains that to use storage live migrations, you need to have OpenShift Virtualization installed. Moreover, you must use MTC CRDs and at least two storage classes.

Table 11.2. Storage live migration requirements

Resource	Purpose
MigCluster	Represents the cluster to use when migrating the storage.
StorageClass	The storage class, ensure there are at least two storage classes.
VirtualMachine	A virtual machine definition, installed by KubeVirt.
VirtualMachineInstance	A running virtual machine, installed by KubeVirt.
DataVolume	A definition on how to populate a persistent volume (PV) with a VM disk, installed by Containerized Data Importer (CDI).

11.4. DEPLOYING A VIRTUAL MACHINE

After installing and activating OpenShift Virtualization and Containerized Data Importer (CDI), create a namespace and deploy a virtual machine (VM).

Procedure

- Deploy the YAML, which creates both a VM definition and a data volume containing the Fedora operating system.
  In the following example, the namespace **mig-vm** is used and the following YAML is used to create a Fedora VM, create and a datavolume containing the Fedora operating system:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: rhel9-lime-damselfly-72
  namespace: mig-vm 1
  labels:
    app: rhel9-lime-damselfly-72
    kubevirt.io/dynamic-credentials-support: 'true'
    vm.kubevirt.io/template: rhel9-server-small
    vm.kubevirt.io/template.namespace: openshift
```

```
    vm.kubevirt.io/template.revision: '1'
    vm.kubevirt.io/template.version: v0.31.1
spec:
  dataVolumeTemplates:
    - apiVersion: cdi.kubevirt.io/v1beta1
      kind: DataVolume
      metadata:
        name: rhel9-lime-damselfly-72
      spec:
        sourceRef:
          kind: DataSource
          name: rhel9
          namespace: openshift-virtualization-os-images
        storage:
          resources:
            requests:
              storage: 30Gi
  running: true 2
  template:
    metadata:
      annotations:
        vm.kubevirt.io/flavor: small
        vm.kubevirt.io/os: rhel9
        vm.kubevirt.io/workload: server
      creationTimestamp: null
      labels:
        kubevirt.io/domain: rhel9-lime-damselfly-72 3
        kubevirt.io/size: small
        network.kubevirt.io/headlessService: headless
    spec:
      architecture: amd64
      domain:
        cpu:
          cores: 1
          sockets: 1
          threads: 1
        devices:
          disks:
            - disk:
                bus: virtio
              name: rootdisk
            - disk:
                bus: virtio
              name: cloudinitdisk
          interfaces:
            - masquerade: {}
              model: virtio
              name: default
          rng: {}
        features:
          acpi: {}
          smm:
            enabled: true
        firmware:
          bootloader:
            efi: {}
```

```
    machine:
      type: pc-q35-rhel9.4.0
    memory:
      guest: 2Gi
    resources: {}
  networks:
    - name: default
      pod: {}
  terminationGracePeriodSeconds: 180
  volumes:
    - dataVolume:
        name: rhel9-lime-damselfly-72
      name: rootdisk
    - cloudInitNoCloud:
        userData: |-
          #cloud-config
          user: cloud-user
          password: password
          chpasswd: { expire: False }
      name: cloudinitdisk
```

1. In this example, the namespace **mig-vm** is used.
2. Use **running: true** to indicate that the VM should be started after creation.
3. The data volume creates a persistent volume claim (PVC) called **rhel9-lime-damselfly-72**, which is the same name as the data volume.

The persistent volume (PV) is populated with the operating system, and the VM is started.

11.5. CREATING A MIGRATION PLAN IN THE MTC WEB CONSOLE

You can create a migration plan in the Migration Toolkit for Containers (MTC) web console.

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges on all clusters.
- You must ensure that the same MTC version is installed on all clusters.
- You must add the clusters and the replication repository to the MTC web console.
- If you want to use the *move* data copy method to migrate a persistent volume (PV), the source and target clusters must have uninterrupted network access to the remote volume.
- If you want to use direct image migration, you must specify the exposed route to the image registry of the source cluster. This can be done by using the MTC web console or by updating the **MigCluster** custom resource manifest.

Procedure

1. In the MTC web console, click **Migration plans**.
2. Click **Add migration plan**.

3. Enter the **Plan name**.
   The migration plan name must not exceed 253 lower-case alphanumeric characters (**a-z, 0-9**) and must not contain spaces or underscores (_).

4. Select a **Source cluster**, a **Target cluster**, and a **Repository**.

5. Click **Next**.

6. Select the projects for migration.

7. Optional: Click the edit icon beside a project to change the target namespace.

WARNING

Migration Toolkit for Containers 1.8.6 and later versions do not support multiple migration plans for a single namespace.

8. Click **Next**.

9. Select a **Migration type** for each PV:

   - The **Copy** option copies the data from the PV of a source cluster to the replication repository and then restores the data on a newly created PV, with similar characteristics, in the target cluster.

   - The **Move** option unmounts a remote volume, for example, NFS, from the source cluster, creates a PV resource on the target cluster pointing to the remote volume, and then mounts the remote volume on the target cluster. Applications running on the target cluster use the same remote volume that the source cluster was using.

10. Click **Next**.

11. Select a **Copy method** for each PV:

    - **Snapshot copy** backs up and restores data using the cloud provider's snapshot functionality. It is significantly faster than **Filesystem copy**.

    - **Filesystem copy** backs up the files on the source cluster and restores them on the target cluster.
      The file system copy method is required for direct volume migration.

12. You can select **Verify copy** to verify data migrated with **Filesystem copy**. Data is verified by generating a checksum for each source file and checking the checksum after restoration. Data verification significantly reduces performance.

13. Select a **Target storage class**.
    If you selected **Filesystem copy**, you can change the target storage class.

14. Click **Next**.

15. On the **Migration options** page, the **Direct image migration** option is selected if you specified an exposed image registry route for the source cluster. The **Direct PV migration** option is selected if you are migrating data with **Filesystem copy**.
    The direct migration options copy images and files directly from the source cluster to the target cluster. This option is much faster than copying images and files from the source cluster to the replication repository and then from the replication repository to the target cluster.

16. Click **Next**.

17. Optional: Click **Add Hook** to add a hook to the migration plan.
    A hook runs custom code. You can add up to four hooks to a single migration plan. Each hook runs during a different migration step.

    a. Enter the name of the hook to display in the web console.

    b. If the hook is an Ansible playbook, select **Ansible playbook** and click **Browse** to upload the playbook or paste the contents of the playbook in the field.

    c. Optional: Specify an Ansible runtime image if you are not using the default hook image.

    d. If the hook is not an Ansible playbook, select **Custom container image** and specify the image name and path.
    A custom container image can include Ansible playbooks.

    e. Select **Source cluster** or **Target cluster**.

    f. Enter the **Service account name** and the **Service account namespace**

    g. Select the migration step for the hook:

    - **preBackup**: Before the application workload is backed up on the source cluster
    - **postBackup**: After the application workload is backed up on the source cluster
    - **preRestore**: Before the application workload is restored on the target cluster
    - **postRestore**: After the application workload is restored on the target cluster

    h. Click **Add**.

18. Click **Finish**.
    The migration plan is displayed in the **Migration plans** list.

11.5.1. Creating the migration plan using YAML manifests

You can create a migration plan using YAML. However, it is recommended to create a migration plan in the Migration Toolkit for Containers (MTC) web console.

Procedure

1. To migrate the **mig-vm** namespace, ensure that the **namespaces** field of the migration plan includes **mig-vm**.

2. Modify the contents of the migration plan by adding **mig-vm** to the namespaces.

   Example migration plan YAML

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigPlan
metadata:
  name: live-migrate-plan
  namespace: openshift-migration
spec:
  namespaces:
  - mig-vm 1
...
```

1 Add **mig-vm** to the namespaces.

- To attempt a live storage migration, the **liveMigrate** field in the migration plan specification must be set to true, and **KubeVirt** must be configured, and be enabled to perform live storage migration.

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigPlan
metadata:
  name: live-migrate-plan
  namespace: openshift-migration
spec:
  liveMigrate: true 1
  namespaces:
...
```

1 Live migration only happens during the cutover of a migration plan.

Staging the migration plan skips any running virtual machines and does not sync the data. Any stopped virtual machine disks are synced.

11.6. KNOWN ISSUES

The following known issues apply when migrating virtual machine (VM) storage:

- You can only migrate VM storage in the same namespace.

11.6.1. Online migration limitations

The following limitations apply to the online migration:

- The VM must be running.
- The volume housing the disk must not have any of the following limitations:
  - Shared disks cannot be migrated live.
  - The **virtio-fs** filesystem volume cannot be migrated live.
  - LUN-to-Disk and Disk-to-LUN migrations are not supported in **libvirt**.
  - LUNs must have persistent reservations.
  - Target PV size must match the source PV size.

- The VM must be migrated to a different node.

CHAPTER 12. TROUBLESHOOTING

This section describes resources for troubleshooting the Migration Toolkit for Containers (MTC).

For known issues, see the MTC release notes.

12.1. MTC WORKFLOW

You can migrate Kubernetes resources, persistent volume data, and internal container images to OpenShift Container Platform 4.14 by using the Migration Toolkit for Containers (MTC) web console or the Kubernetes API.

MTC migrates the following resources:

- A namespace specified in a migration plan.
- Namespace-scoped resources: When the MTC migrates a namespace, it migrates all the objects and resources associated with that namespace, such as services or pods. Additionally, if a resource that exists in the namespace but not at the cluster level depends on a resource that exists at the cluster level, the MTC migrates both resources.
  For example, a security context constraint (SCC) is a resource that exists at the cluster level and a service account (SA) is a resource that exists at the namespace level. If an SA exists in a namespace that the MTC migrates, the MTC automatically locates any SCCs that are linked to the SA and also migrates those SCCs. Similarly, the MTC migrates persistent volumes that are linked to the persistent volume claims of the namespace.

  NOTE

  Cluster-scoped resources might have to be migrated manually, depending on the resource.

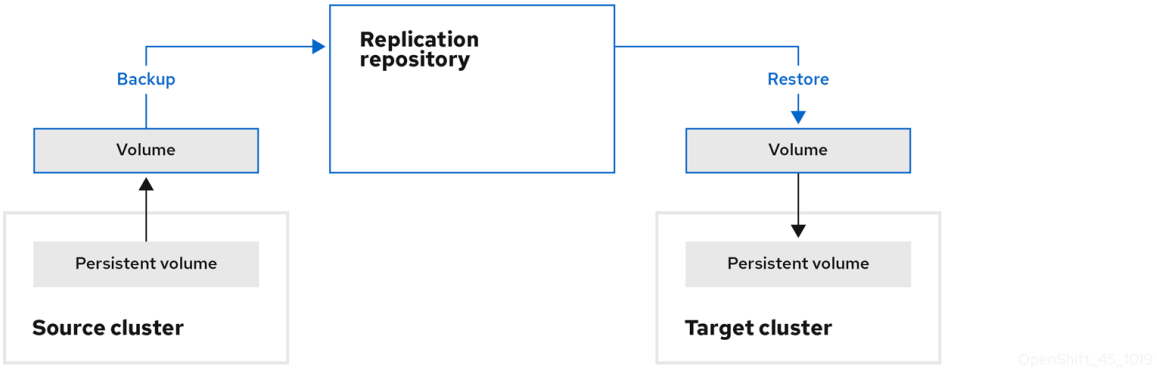
- Custom resources (CRs) and custom resource definitions (CRDs): MTC automatically migrates CRs and CRDs at the namespace level.

Migrating an application with the MTC web console involves the following steps:

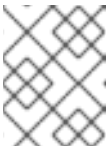
1. Install the Migration Toolkit for Containers Operator on all clusters.
   You can install the Migration Toolkit for Containers Operator in a restricted environment with limited or no internet access. The source and target clusters must have network access to each other and to a mirror registry.
2. Configure the replication repository, an intermediate object storage that MTC uses to migrate data.
   The source and target clusters must have network access to the replication repository during migration. If you are using a proxy server, you must configure it to allow network traffic between the replication repository and the clusters.
3. Add the source cluster to the MTC web console.
4. Add the replication repository to the MTC web console.
5. Create a migration plan, with one of the following data migration options:
   - **Copy**: MTC copies the data from the source cluster to the replication repository, and from the replication repository to the target cluster.

NOTE

If you are using direct image migration or direct volume migration, the images or volumes are copied directly from the source cluster to the target cluster.



- **Move**: MTC unmounts a remote volume, for example, NFS, from the source cluster, creates a PV resource on the target cluster pointing to the remote volume, and then mounts the remote volume on the target cluster. Applications running on the target cluster use the same remote volume that the source cluster was using. The remote volume must be accessible to the source and target clusters.



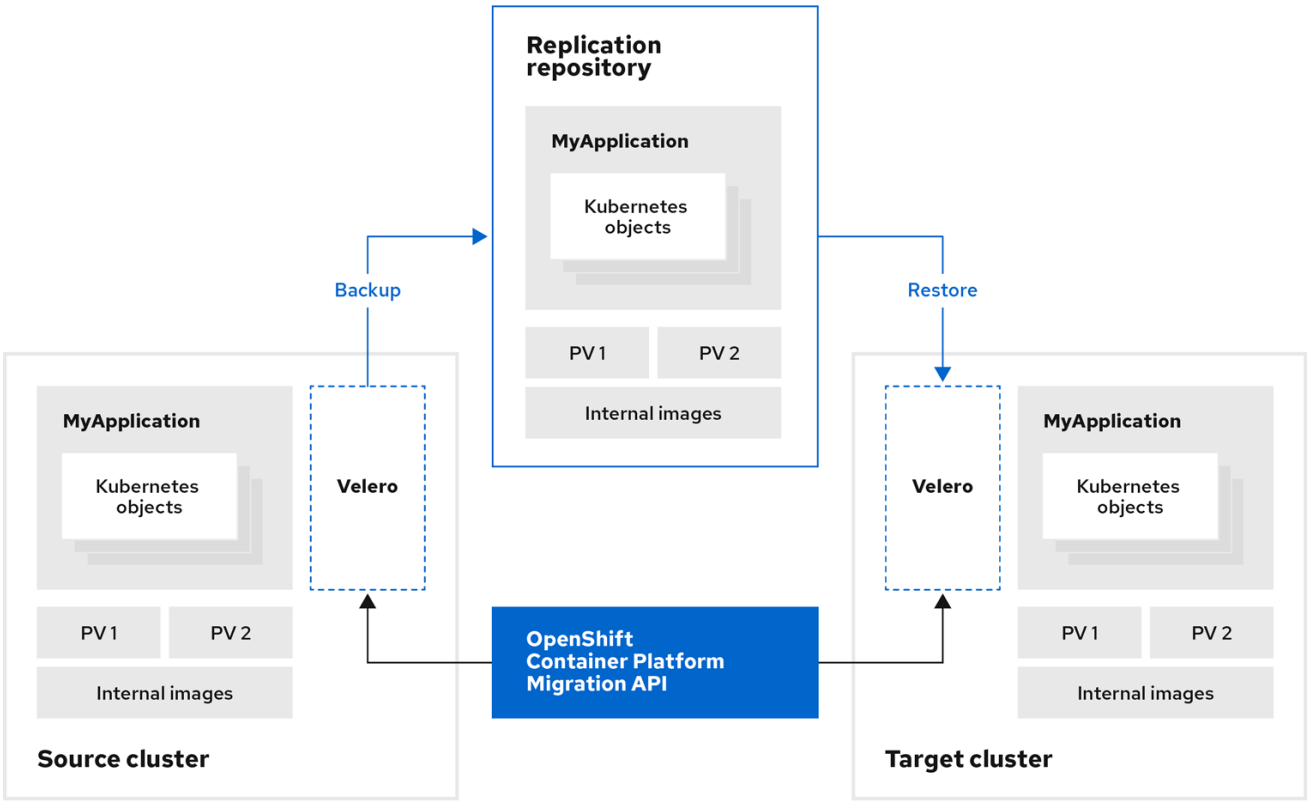
NOTE

Although the replication repository does not appear in this diagram, it is required for migration.

6. Run the migration plan, with one of the following options:

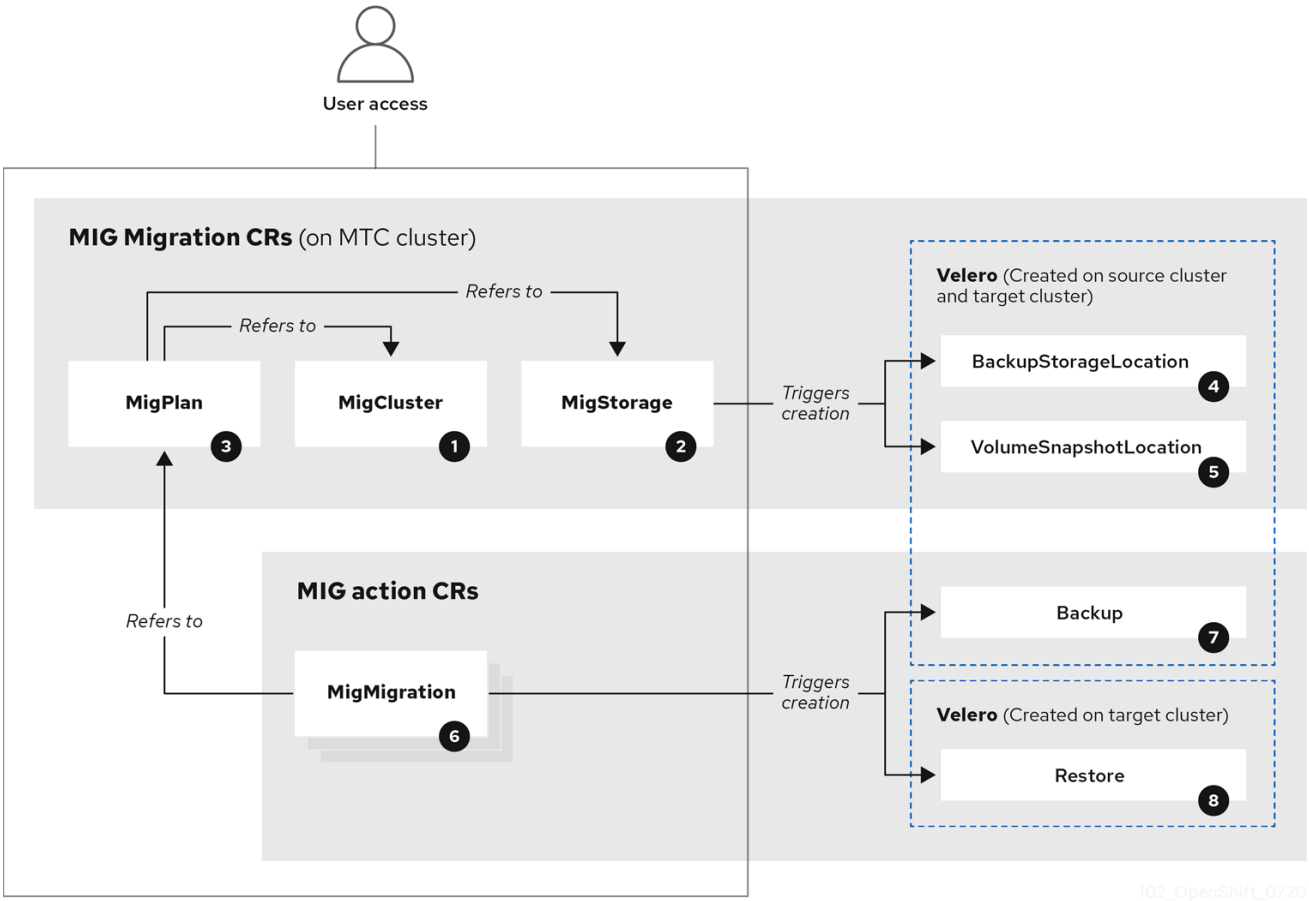
- **Stage** copies data to the target cluster without stopping the application.
  A stage migration can be run multiple times so that most of the data is copied to the target before migration. Running one or more stage migrations reduces the duration of the cutover migration.

- **Cutover** stops the application on the source cluster and moves the resources to the target cluster.
  Optional: You can clear the **Halt transactions on the source cluster during migration** checkbox.



About MTC custom resources

The Migration Toolkit for Containers (MTC) creates the following custom resources (CRs):



1 MigCluster (configuration, MTC cluster): Cluster definition

2 MigStorage (configuration, MTC cluster): Storage definition

3 MigPlan (configuration, MTC cluster): Migration plan

The **MigPlan** CR describes the source and target clusters, replication repository, and namespaces being migrated. It is associated with 0, 1, or many **MigMigration** CRs.

NOTE

Deleting a **MigPlan** CR deletes the associated **MigMigration** CRs.

4 BackupStorageLocation (configuration, MTC cluster): Location of **Velero** backup objects

5 VolumeSnapshotLocation (configuration, MTC cluster): Location of **Velero** volume snapshots

6 MigMigration (action, MTC cluster): Migration, created every time you stage or migrate data. Each **MigMigration** CR is associated with a **MigPlan** CR.

7 Backup (action, source cluster): When you run a migration plan, the **MigMigration** CR creates two **Velero** backup CRs on each source cluster:

- Backup CR #1 for Kubernetes objects
- Backup CR #2 for PV data

8 Restore (action, target cluster): When you run a migration plan, the **MigMigration** CR creates two **Velero** restore CRs on the target cluster:

- Restore CR #1 (using Backup CR #2) for PV data
- Restore CR #2 (using Backup CR #1) for Kubernetes objects

12.2. MIGRATION TOOLKIT FOR CONTAINERS CUSTOM RESOURCE MANIFESTS

Migration Toolkit for Containers (MTC) uses the following custom resource (CR) manifests for migrating applications.

12.2.1. DirectImageMigration

The **DirectImageMigration** CR copies images directly from the source cluster to the destination cluster.

```
apiVersion: migration.openshift.io/v1alpha1
kind: DirectImageMigration
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
```

```
  name: <direct_image_migration>
spec:
  srcMigClusterRef:
    name: <source_cluster>
    namespace: openshift-migration
  destMigClusterRef:
    name: <destination_cluster>
    namespace: openshift-migration
  namespaces: (1)
    - <source_namespace_1>
    - <source_namespace_2>:<destination_namespace_3> (2)
```

(1) One or more namespaces containing images to be migrated. By default, the destination namespace has the same name as the source namespace.

(2) Source namespace mapped to a destination namespace with a different name.

12.2.2. DirectImageStreamMigration

The **DirectImageStreamMigration** CR copies image stream references directly from the source cluster to the destination cluster.

```
apiVersion: migration.openshift.io/v1alpha1
kind: DirectImageStreamMigration
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: <direct_image_stream_migration>
spec:
  srcMigClusterRef:
    name: <source_cluster>
    namespace: openshift-migration
  destMigClusterRef:
    name: <destination_cluster>
    namespace: openshift-migration
  imageStreamRef:
    name: <image_stream>
    namespace: <source_image_stream_namespace>
  destNamespace: <destination_image_stream_namespace>
```

12.2.3. DirectVolumeMigration

The **DirectVolumeMigration** CR copies persistent volumes (PVs) directly from the source cluster to the destination cluster.

```
apiVersion: migration.openshift.io/v1alpha1
kind: DirectVolumeMigration
metadata:
  name: <direct_volume_migration>
  namespace: openshift-migration
spec:
  createDestinationNamespaces: false (1)
  deleteProgressReportingCRs: false (2)
```

```
  destMigClusterRef:
    name: <host_cluster> 3
    namespace: openshift-migration
  persistentVolumeClaims:
  - name: <pvc> 4
    namespace: <pvc_namespace>
  srcMigClusterRef:
    name: <source_cluster>
    namespace: openshift-migration
```

1. Set to **true** to create namespaces for the PVs on the destination cluster.
2. Set to **true** to delete **DirectVolumeMigrationProgress** CRs after migration. The default is **false** so that **DirectVolumeMigrationProgress** CRs are retained for troubleshooting.
3. Update the cluster name if the destination cluster is not the host cluster.
4. Specify one or more PVCs to be migrated.

12.2.4. DirectVolumeMigrationProgress

The **DirectVolumeMigrationProgress** CR shows the progress of the **DirectVolumeMigration** CR.

```
apiVersion: migration.openshift.io/v1alpha1
kind: DirectVolumeMigrationProgress
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: <direct_volume_migration_progress>
spec:
  clusterRef:
    name: <source_cluster>
    namespace: openshift-migration
  podRef:
    name: <rsync_pod>
    namespace: openshift-migration
```

12.2.5. MigAnalytic

The **MigAnalytic** CR collects the number of images, Kubernetes resources, and the persistent volume (PV) capacity from an associated **MigPlan** CR.

You can configure the data that it collects.

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigAnalytic
metadata:
  annotations:
    migplan: <migplan>
  name: <miganalytic>
  namespace: openshift-migration
  labels:
    migplan: <migplan>
```

```
spec:
  analyzeImageCount: true (1)
  analyzeK8SResources: true (2)
  analyzePVCapacity: true (3)
  listImages: false (4)
  listImagesLimit: 50 (5)
  migPlanRef:
    name: <migplan>
    namespace: openshift-migration
```

1. Optional: Returns the number of images.
2. Optional: Returns the number, kind, and API version of the Kubernetes resources.
3. Optional: Returns the PV capacity.
4. Returns a list of image names. The default is **false** so that the output is not excessively long.
5. Optional: Specify the maximum number of image names to return if **listImages** is **true**.

12.2.6. MigCluster

The **MigCluster** CR defines a host, local, or remote cluster.

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigCluster
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: <host_cluster> (1)
  namespace: openshift-migration
spec:
  isHostCluster: true (2)
# The 'azureResourceGroup' parameter is relevant only for Microsoft Azure.
  azureResourceGroup: <azure_resource_group> (3)
  caBundle: <ca_bundle_base64> (4)
  insecure: false (5)
  refresh: false (6)
# The 'restartRestic' parameter is relevant for a source cluster.
  restartRestic: true (7)
# The following parameters are relevant for a remote cluster.
  exposedRegistryPath: <registry_route> (8)
  url: <destination_cluster_url> (9)
  serviceAccountSecretRef:
    name: <source_secret> (10)
    namespace: openshift-config
```

1. Update the cluster name if the **migration-controller** pod is not running on this cluster.
2. The **migration-controller** pod runs on this cluster if **true**.
3. Microsoft Azure only: Specify the resource group.

4. Optional: If you created a certificate bundle for self-signed CA certificates and if the **insecure** parameter value is **false**, specify the base64-encoded certificate bundle.
5. Set to **true** to disable SSL verification.
6. Set to **true** to validate the cluster.
7. Set to **true** to restart the **Restic** pods on the source cluster after the **Stage** pods are created.
8. Remote cluster and direct image migration only: Specify the exposed secure registry path.
9. Remote cluster only: Specify the URL.
10. Remote cluster only: Specify the name of the **Secret** object.

12.2.7. MigHook

The **MigHook** CR defines a migration hook that runs custom code at a specified stage of the migration. You can create up to four migration hooks. Each hook runs during a different phase of the migration.

You can configure the hook name, runtime duration, a custom image, and the cluster where the hook will run.

The migration phases and namespaces of the hooks are configured in the **MigPlan** CR.

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigHook
metadata:
  generateName: <hook_name_prefix> 1
  name: <mighook> 2
  namespace: openshift-migration
spec:
  activeDeadlineSeconds: 1800 3
  custom: false 4
  image: <hook_image> 5
  playbook: <ansible_playbook_base64> 6
  targetCluster: source 7
```

1. Optional: A unique hash is appended to the value for this parameter so that each migration hook has a unique name. You do not need to specify the value of the **name** parameter.
2. Specify the migration hook name, unless you specify the value of the **generateName** parameter.
3. Optional: Specify the maximum number of seconds that a hook can run. The default is **1800**.
4. The hook is a custom image if **true**. The custom image can include Ansible or it can be written in a different programming language.
5. Specify the custom image, for example, **quay.io/konveyor/hook-runner:latest**. Required if **custom** is **true**.
6. Base64-encoded Ansible playbook. Required if **custom** is **false**.

7 Specify the cluster on which the hook will run. Valid values are **source** or **destination**.

12.2.8. MigMigration

The **MigMigration** CR runs a **MigPlan** CR.

You can configure a **Migmigration** CR to run a stage or incremental migration, to cancel a migration in progress, or to roll back a completed migration.

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigMigration
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: <migmigration>
  namespace: openshift-migration
spec:
  canceled: false 1
  rollback: false 2
  stage: false 3
  quiescePods: true 4
  keepAnnotations: true 5
  verify: false 6
  migPlanRef:
    name: <migplan>
    namespace: openshift-migration
```

1 Set to **true** to cancel a migration in progress.

2 Set to **true** to roll back a completed migration.

3 Set to **true** to run a stage migration. Data is copied incrementally and the pods on the source cluster are not stopped.

4 Set to **true** to stop the application during migration. The pods on the source cluster are scaled to **0** after the **Backup** stage.

5 Set to **true** to retain the labels and annotations applied during the migration.

6 Set to **true** to check the status of the migrated pods on the destination cluster are checked and to return the names of pods that are not in a **Running** state.

12.2.9. MigPlan

The **MigPlan** CR defines the parameters of a migration plan.

You can configure destination namespaces, hook phases, and direct or indirect migration.

NOTE

By default, a destination namespace has the same name as the source namespace. If you configure a different destination namespace, you must ensure that the namespaces are not duplicated on the source or the destination clusters because the UID and GID ranges are copied during migration.

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigPlan
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: <migplan>
  namespace: openshift-migration
spec:
  closed: false (1)
  srcMigClusterRef:
    name: <source_cluster>
    namespace: openshift-migration
  destMigClusterRef:
    name: <destination_cluster>
    namespace: openshift-migration
  hooks: (2)
    - executionNamespace: <namespace> (3)
      phase: <migration_phase> (4)
      reference:
        name: <hook> (5)
        namespace: <hook_namespace> (6)
      serviceAccount: <service_account> (7)
  indirectImageMigration: true (8)
  indirectVolumeMigration: false (9)
  migStorageRef:
    name: <migstorage>
    namespace: openshift-migration
  namespaces:
    - <source_namespace_1> (10)
    - <source_namespace_2>
    - <source_namespace_3>:<destination_namespace_4> (11)
  refresh: false (12)
```

(1) The migration has completed if **true**. You cannot create another **MigMigration** CR for this **MigPlan** CR.

(2) Optional: You can specify up to four migration hooks. Each hook must run during a different migration phase.

(3) Optional: Specify the namespace in which the hook will run.

(4) Optional: Specify the migration phase during which a hook runs. One hook can be assigned to one phase. Valid values are **PreBackup**, **PostBackup**, **PreRestore**, and **PostRestore**.

(5) Optional: Specify the name of the **MigHook** CR.

(6) Optional: Specify the namespace of **MigHook** CR.

7 Optional: Specify a service account with **cluster-admin** privileges.

8 Direct image migration is disabled if **true**. Images are copied from the source cluster to the replication repository and from the replication repository to the destination cluster.

9 Direct volume migration is disabled if **true**. PVs are copied from the source cluster to the replication repository and from the replication repository to the destination cluster.

10 Specify one or more source namespaces. If you specify only the source namespace, the destination namespace is the same.

11 Specify the destination namespace if it is different from the source namespace.

12 The **MigPlan** CR is validated if **true**.

12.2.10. MigStorage

The **MigStorage** CR describes the object storage for the replication repository.

Amazon Web Services (AWS), Microsoft Azure, Google Cloud Storage, Multi-Cloud Object Gateway, and generic S3-compatible cloud storage are supported.

AWS and the snapshot copy method have additional parameters.

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigStorage
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: <migstorage>
  namespace: openshift-migration
spec:
  backupStorageProvider: <backup_storage_provider> 1
  volumeSnapshotProvider: <snapshot_storage_provider> 2
  backupStorageConfig:
    awsBucketName: <bucket> 3
    awsRegion: <region> 4
    credsSecretRef:
      namespace: openshift-config
      name: <storage_secret> 5
    awsKmsKeyId: <key_id> 6
    awsPublicUrl: <public_url> 7
    awsSignatureVersion: <signature_version> 8
  volumeSnapshotConfig:
    awsRegion: <region> 9
    credsSecretRef:
      namespace: openshift-config
      name: <storage_secret> 10
  refresh: false 11
```

1 Specify the storage provider.

2 Snapshot copy method only: Specify the storage provider.

3. AWS only: Specify the bucket name.
4. AWS only: Specify the bucket region, for example, **us-east-1**.
5. Specify the name of the **Secret** object that you created for the storage.
6. AWS only: If you are using the AWS Key Management Service, specify the unique identifier of the key.
7. AWS only: If you granted public access to the AWS bucket, specify the bucket URL.
8. AWS only: Specify the AWS signature version for authenticating requests to the bucket, for example, **4**.
9. Snapshot copy method only: Specify the geographical region of the clusters.
10. Snapshot copy method only: Specify the name of the **Secret** object that you created for the storage.
11. Set to **true** to validate the cluster.

12.3. LOGS AND DEBUGGING TOOLS

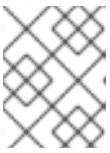
This section describes logs and debugging tools that you can use for troubleshooting.

12.3.1. Viewing migration plan resources

You can view migration plan resources to monitor a running migration or to troubleshoot a failed migration by using the MTC web console and the command-line interface (CLI).

Procedure

1. In the MTC web console, click **Migration Plans**.
2. Click the **Migrations** number next to a migration plan to view the **Migrations** page.
3. Click a migration to view the **Migration details**.
4. Expand **Migration resources** to view the migration resources and their status in a tree view.



NOTE

To troubleshoot a failed migration, start with a high-level resource that has failed and then work down the resource tree towards the lower-level resources.

5. Click the Options menu  next to a resource and select one of the following options:
   - **Copy oc describe command** copies the command to your clipboard.
     - Log in to the relevant cluster and then run the command.
       The conditions and events of the resource are displayed in YAML format.

- **Copy oc logs command** copies the command to your clipboard.
  - Log in to the relevant cluster and then run the command.
    If the resource supports log filtering, a filtered log is displayed.
- **View JSON** displays the resource data in JSON format in a web browser.
  The data is the same as the output for the **oc get <resource>** command.

12.3.2. Viewing a migration plan log

You can view an aggregated log for a migration plan. You use the MTC web console to copy a command to your clipboard and then run the command from the command-line interface (CLI).

The command displays the filtered logs of the following pods:

- **Migration Controller**
- **Velero**
- **Restic**
- **Rsync**
- **Stunnel**
- **Registry**

Procedure

1. In the MTC web console, click **Migration Plans**.
2. Click the **Migrations** number next to a migration plan.
3. Click **View logs**.
4. Click the Copy icon to copy the **oc logs** command to your clipboard.
5. Log in to the relevant cluster and enter the command on the CLI.
   The aggregated log for the migration plan is displayed.

12.3.3. Using the migration log reader

You can use the migration log reader to display a single filtered view of all the migration logs.

Procedure

1. Get the **mig-log-reader** pod:

```
$ oc -n openshift-migration get pods | grep log
```

2. Enter the following command to display a single migration log:

```
$ oc -n openshift-migration logs -f <mig-log-reader-pod> -c color 1
```

1 The **-c plain** option displays the log without colors.

12.3.4. Accessing performance metrics

The **MigrationController** custom resource (CR) records metrics and pulls them into on-cluster monitoring storage. You can query the metrics by using Prometheus Query Language (PromQL) to diagnose migration performance issues. All metrics are reset when the Migration Controller pod restarts.

You can access the performance metrics and run queries by using the OpenShift Container Platform web console.

Procedure

1. In the OpenShift Container Platform web console, click **Observe → Metrics**.
2. Enter a PromQL query, select a time window to display, and click **Run Queries**.
   If your web browser does not display all the results, use the Prometheus console.

12.3.4.1. Provided metrics

The **MigrationController** custom resource (CR) provides metrics for the **MigMigration** CR count and for its API requests.

12.3.4.1.1. cam_app_workload_migrations

This metric is a count of **MigMigration** CRs over time. It is useful for viewing alongside the **mtc_client_request_count** and **mtc_client_request_elapsed** metrics to collate API request information with migration status changes. This metric is included in Telemetry.

Table 12.1. cam_app_workload_migrations metric

Queryable label name	Sample label values	Label description
status	**running**, **idle**, **failed**, **completed**	Status of the **MigMigration** CR
type	stage, final	Type of the **MigMigration** CR

12.3.4.1.2. mtc_client_request_count

This metric is a cumulative count of Kubernetes API requests that **MigrationController** issued. It is not included in Telemetry.

Table 12.2. mtc_client_request_count metric

Queryable label name	Sample label values	Label description
cluster	**https://migcluster-url:443**	Cluster that the request was issued against

Queryable label name	Sample label values	Label description
component	**MigPlan**, **MigCluster**	Sub-controller API that issued request
function	**(*ReconcileMigPlan).Reconcile**	Function that the request was issued from
kind	**SecretList**, **Deployment**	Kubernetes kind the request was issued for

12.3.4.1.3. mtc_client_request_elapsed

This metric is a cumulative latency, in milliseconds, of Kubernetes API requests that **MigrationController** issued. It is not included in Telemetry.

Table 12.3. mtc_client_request_elapsed metric

Queryable label name	Sample label values	Label description
cluster	**https://cluster-url.com:443**	Cluster that the request was issued against
component	**migplan**, **migcluster**	Sub-controller API that issued request
function	**(*ReconcileMigPlan).Reconcile**	Function that the request was issued from
kind	**SecretList**, **Deployment**	Kubernetes resource that the request was issued for

12.3.4.1.4. Useful queries

The table lists some helpful queries that can be used for monitoring performance.

Table 12.4. Useful queries

Query	Description
mtc_client_request_count	Number of API requests issued, sorted by request type
sum(mtc_client_request_count)	Total number of API requests issued
mtc_client_request_elapsed	API request latency, sorted by request type
sum(mtc_client_request_elapsed)	Total latency of API requests

Query	Description
sum(mtc_client_request_elapsed) / sum(mtc_client_request_count)	Average latency of API requests
mtc_client_request_elapsed / mtc_client_request_count	Average latency of API requests, sorted by request type
cam_app_workload_migrations{status="running"} * 100	Count of running migrations, multiplied by 100 for easier viewing alongside request counts

12.3.5. Using the must-gather tool

You can collect logs, metrics, and information about MTC custom resources by using the **must-gather** tool.

The **must-gather** data must be attached to all customer cases.

You can collect data for a one-hour or a 24-hour period and view the data with the Prometheus console.

Prerequisites

- You must be logged in to the OpenShift Container Platform cluster as a user with the **cluster-admin** role.
- You must have the OpenShift CLI (**oc**) installed.

Procedure

1. Navigate to the directory where you want to store the **must-gather** data.
2. Run the **oc adm must-gather** command for one of the following data collection options:
   - To collect data for the past 24 hours, run the following command:

     ```
     $ oc adm must-gather --image=registry.redhat.io/rhmtc/openshift-migration-must-gather-rhel8:v1.8
     ```

     This command saves the data as the **must-gather/must-gather.tar.gz** file. You can upload this file to a support case on the Red Hat Customer Portal .
   - To collect data for the past 24 hours, run the following command:

     ```
     $ oc adm must-gather --image=registry.redhat.io/rhmtc/openshift-migration-must-gather-rhel8:v1.8 -- /usr/bin/gather_metrics_dump
     ```

     This operation can take a long time. This command saves the data as the **must-gather/metrics/prom_data.tar.gz** file.

12.3.6. Debugging Velero resources with the Velero CLI tool

You can debug **Backup** and **Restore** custom resources (CRs) and retrieve logs with the Velero CLI tool. The Velero CLI tool provides more detailed information than the OpenShift CLI tool.

The Velero CLI tool provides more detailed information than the OpenShift CLI tool.

12.3.6.1. Syntax

Use the **oc exec** command to run a Velero CLI command:

```
$ oc -n openshift-migration exec deployment/velero -c velero -- ./velero \
  <backup_restore_cr> <command> <cr_name>
```

Example

```
$ oc -n openshift-migration exec deployment/velero -c velero -- ./velero \
  backup describe 0e44ae00-5dc3-11eb-9ca8-df7e5254778b-2d8ql
```

12.3.6.2. Help option

Use the **velero --help** option to list all Velero CLI commands:

```
$ oc -n openshift-migration exec deployment/velero -c velero -- ./velero \
  --help
```

12.3.6.3. Describe command

Use the **velero describe** command to retrieve a summary of warnings and errors associated with a **Backup** or **Restore** CR:

```
$ oc -n openshift-migration exec deployment/velero -c velero -- ./velero \
  <backup_restore_cr> describe <cr_name>
```

Example

```
$ oc -n openshift-migration exec deployment/velero -c velero -- ./velero \
  backup describe 0e44ae00-5dc3-11eb-9ca8-df7e5254778b-2d8ql
```

The following types of restore errors and warnings are shown in the output of a **velero describe** request:

- **Velero**: A list of messages related to the operation of Velero itself, for example, messages related to connecting to the cloud, reading a backup file, and so on
- **Cluster**: A list of messages related to backing up or restoring cluster-scoped resources
- **Namespaces**: A list of list of messages related to backing up or restoring resources stored in namespaces

One or more errors in one of these categories results in a **Restore** operation receiving the status of **PartiallyFailed** and not **Completed**. Warnings do not lead to a change in the completion status.

Consider the following points for these restore errors:

IMPORTANT

- For resource-specific errors, that is, **Cluster** and **Namespaces** errors, the **restore describe --details** output includes a resource list that lists all resources that Velero succeeded in restoring. For any resource that has such an error, check to see if the resource is actually in the cluster.
- If there are **Velero** errors, but no resource-specific errors, in the output of a **describe** command, it is possible that the restore completed without any actual problems in restoring workloads, but carefully validate post-restore applications. For example, if the output contains **PodVolumeRestore** or node agent-related errors, check the status of **PodVolumeRestores** and **DataDownloads**. If none of these are failed or still running, then volume data might have been fully restored.

12.3.6.4. Logs command

Use the **velero logs** command to retrieve the logs of a **Backup** or **Restore** CR:

```
$ oc -n openshift-migration exec deployment/velero -c velero -- ./velero \
  <backup_restore_cr> logs <cr_name>
```

Example

```
$ oc -n openshift-migration exec deployment/velero -c velero -- ./velero \
  restore logs ccc7c2d0-6017-11eb-afab-85d0007f5a19-x4lbf
```

12.3.7. Debugging a partial migration failure

You can debug a partial migration failure warning message by using the Velero CLI to examine the **Restore** custom resource (CR) logs.

A partial failure occurs when Velero encounters an issue that does not cause a migration to fail. For example, if a custom resource definition (CRD) is missing or if there is a discrepancy between CRD versions on the source and target clusters, the migration completes but the CR is not created on the target cluster.

Velero logs the issue as a partial failure and then processes the rest of the objects in the **Backup** CR.

Procedure

1. Check the status of a **MigMigration** CR:

```
$ oc get migmigration <migmigration> -o yaml
```

Example output

```
status:
  conditions:
  - category: Warn
    durable: true
    lastTransitionTime: "2021-01-26T20:48:40Z"
    message: 'Final Restore openshift-migration/ccc7c2d0-6017-11eb-afab-85d0007f5a19-
x4lbf: partially failed on destination cluster'
```

```
    status: "True"
    type: VeleroFinalRestorePartiallyFailed
  - category: Advisory
    durable: true
    lastTransitionTime: "2021-01-26T20:48:42Z"
    message: The migration has completed with warnings, please look at `Warn` conditions.
    reason: Completed
    status: "True"
    type: SucceededWithWarnings
```

2. Check the status of the **Restore** CR by using the Velero **describe** command:

```
$ oc -n {namespace} exec deployment/velero -c velero -- ./velero \
  restore describe <restore>
```

Example output

```
Phase:  PartiallyFailed (run 'velero restore logs ccc7c2d0-6017-11eb-afab-85d0007f5a19-x4lbf' for more information)

Errors:
  Velero:     <none>
  Cluster:    <none>
  Namespaces:
    migration-example:  error restoring example.com/migration-example/migration-example: the server could not find the requested resource
```

3. Check the **Restore** CR logs by using the Velero **logs** command:

```
$ oc -n {namespace} exec deployment/velero -c velero -- ./velero \
  restore logs <restore>
```

Example output

```
time="2021-01-26T20:48:37Z" level=info msg="Attempting to restore migration-example: migration-example" logSource="pkg/restore/restore.go:1107" restore=openshift-migration/ccc7c2d0-6017-11eb-afab-85d0007f5a19-x4lbf
time="2021-01-26T20:48:37Z" level=info msg="error restoring migration-example: the server could not find the requested resource" logSource="pkg/restore/restore.go:1170" restore=openshift-migration/ccc7c2d0-6017-11eb-afab-85d0007f5a19-x4lbf
```

The **Restore** CR log error message, **the server could not find the requested resource**, indicates the cause of the partially failed migration.

12.3.8. Using MTC custom resources for troubleshooting

You can check the following Migration Toolkit for Containers (MTC) custom resources (CRs) to troubleshoot a failed migration:

- **MigCluster**
- **MigStorage**

- **MigPlan**
- **BackupStorageLocation**
  The **BackupStorageLocation** CR contains a **migrationcontroller** label to identify the MTC instance that created the CR:

```
labels:
  migrationcontroller: ebe13bee-c803-47d0-a9e9-83f380328b93
```

- **VolumeSnapshotLocation**
  The **VolumeSnapshotLocation** CR contains a **migrationcontroller** label to identify the MTC instance that created the CR:

```
labels:
  migrationcontroller: ebe13bee-c803-47d0-a9e9-83f380328b93
```

- **MigMigration**
- **Backup**
  MTC changes the reclaim policy of migrated persistent volumes (PVs) to **Retain** on the target cluster. The **Backup** CR contains an **openshift.io/orig-reclaim-policy** annotation that indicates the original reclaim policy. You can manually restore the reclaim policy of the migrated PVs.
- **Restore**

Procedure

1. List the **MigMigration** CRs in the **openshift-migration** namespace:

```
$ oc get migmigration -n openshift-migration
```

Example output

```
NAME                                   AGE
88435fe0-c9f8-11e9-85e6-5d593ce65e10   6m42s
```

2. Inspect the **MigMigration** CR:

```
$ oc describe migmigration 88435fe0-c9f8-11e9-85e6-5d593ce65e10 -n openshift-migration
```

The output is similar to the following examples.

MigMigration example output

```
name:         88435fe0-c9f8-11e9-85e6-5d593ce65e10
namespace:    openshift-migration
labels:       <none>
annotations:  touch: 3b48b543-b53e-4e44-9d34-33563f0f8147
apiVersion:  migration.openshift.io/v1alpha1
kind:         MigMigration
metadata:
```

```
  creationTimestamp:  2019-08-29T01:01:29Z
  generation:         20
  resourceVersion:    88179
  selfLink:           /apis/migration.openshift.io/v1alpha1/namespaces/openshift-
migration/migmigrations/88435fe0-c9f8-11e9-85e6-5d593ce65e10
  uid:                8886de4c-c9f8-11e9-95ad-0205fe66cbb6
spec:
  migPlanRef:
    name:       socks-shop-mig-plan
    namespace:  openshift-migration
  quiescePods:  true
  stage:        false
status:
  conditions:
    category:             Advisory
    durable:              True
    lastTransitionTime:  2019-08-29T01:03:40Z
    message:              The migration has completed successfully.
    reason:               Completed
    status:               True
    type:                 Succeeded
  phase:                  Completed
  startTimestamp:         2019-08-29T01:01:29Z
events:                   <none>
```

Velero backup CR #2 example output that describes the PV data

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  annotations:
    openshift.io/migrate-copy-phase: final
    openshift.io/migrate-quiesce-pods: "true"
    openshift.io/migration-registry: 172.30.105.179:5000
    openshift.io/migration-registry-dir: /socks-shop-mig-plan-registry-44dd3bd5-c9f8-11e9-95ad-
0205fe66cbb6
    openshift.io/orig-reclaim-policy: delete
  creationTimestamp: "2019-08-29T01:03:15Z"
  generateName: 88435fe0-c9f8-11e9-85e6-5d593ce65e10-
  generation: 1
  labels:
    app.kubernetes.io/part-of: migration
    migmigration: 8886de4c-c9f8-11e9-95ad-0205fe66cbb6
    migration-stage-backup: 8886de4c-c9f8-11e9-95ad-0205fe66cbb6
    velero.io/storage-location: myrepo-vpzq9
  name: 88435fe0-c9f8-11e9-85e6-5d593ce65e10-59gb7
  namespace: openshift-migration
  resourceVersion: "87313"
  selfLink: /apis/velero.io/v1/namespaces/openshift-migration/backups/88435fe0-c9f8-11e9-85e6-
5d593ce65e10-59gb7
  uid: c80dbbc0-c9f8-11e9-95ad-0205fe66cbb6
spec:
  excludedNamespaces: []
  excludedResources: []
  hooks:
    resources: []
```

```
  includeClusterResources: null
  includedNamespaces:
  - sock-shop
  includedResources:
  - persistentvolumes
  - persistentvolumeclaims
  - namespaces
  - imagestreams
  - imagestreamtags
  - secrets
  - configmaps
  - pods
  labelSelector:
    matchLabels:
      migration-included-stage-backup: 8886de4c-c9f8-11e9-95ad-0205fe66cbb6
  storageLocation: myrepo-vpzq9
  ttl: 720h0m0s
  volumeSnapshotLocations:
  - myrepo-wv6fx
status:
  completionTimestamp: "2019-08-29T01:02:36Z"
  errors: 0
  expiration: "2019-09-28T01:02:35Z"
  phase: Completed
  startTimestamp: "2019-08-29T01:02:35Z"
  validationErrors: null
  version: 1
  volumeSnapshotsAttempted: 0
  volumeSnapshotsCompleted: 0
  warnings: 0
```

Velero restore CR #2 example output that describes the Kubernetes resources

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  annotations:
    openshift.io/migrate-copy-phase: final
    openshift.io/migrate-quiesce-pods: "true"
    openshift.io/migration-registry: 172.30.90.187:5000
    openshift.io/migration-registry-dir: /socks-shop-mig-plan-registry-36f54ca7-c925-11e9-825a-
06fa9fb68c88
  creationTimestamp: "2019-08-28T00:09:49Z"
  generateName: e13a1b60-c927-11e9-9555-d129df7f3b96-
  generation: 3
  labels:
    app.kubernetes.io/part-of: migration
    migmigration: e18252c9-c927-11e9-825a-06fa9fb68c88
    migration-final-restore: e18252c9-c927-11e9-825a-06fa9fb68c88
  name: e13a1b60-c927-11e9-9555-d129df7f3b96-gb8nx
  namespace: openshift-migration
  resourceVersion: "82329"
  selfLink: /apis/velero.io/v1/namespaces/openshift-migration/restores/e13a1b60-c927-11e9-9555-
d129df7f3b96-gb8nx
  uid: 26983ec0-c928-11e9-825a-06fa9fb68c88
spec:
```

```
  backupName: e13a1b60-c927-11e9-9555-d129df7f3b96-sz24f
  excludedNamespaces: null
  excludedResources:
  - nodes
  - events
  - events.events.k8s.io
  - backups.velero.io
  - restores.velero.io
  - resticrepositories.velero.io
  includedNamespaces: null
  includedResources: null
  namespaceMapping: null
  restorePVs: true
status:
  errors: 0
  failureReason: ""
  phase: Completed
  validationErrors: null
  warnings: 15
```

12.4. COMMON ISSUES AND CONCERNS

This section describes common issues and concerns that can cause issues during migration.

12.4.1. Direct volume migration does not complete

If direct volume migration does not complete, the target cluster might not have the same **node-selector** annotations as the source cluster.

Migration Toolkit for Containers (MTC) migrates namespaces with all annotations to preserve security context constraints and scheduling requirements. During direct volume migration, MTC creates Rsync transfer pods on the target cluster in the namespaces that were migrated from the source cluster. If a target cluster namespace does not have the same annotations as the source cluster namespace, the Rsync transfer pods cannot be scheduled. The Rsync pods remain in a **Pending** state.

You can identify and fix this issue by performing the following procedure.

Procedure

1. Check the status of the **MigMigration** CR:

   ```
   $ oc describe migmigration <pod> -n openshift-migration
   ```

   The output includes the following status message:

   #### Example output

   ```
   Some or all transfer pods are not running for more than 10 mins on destination cluster
   ```

2. On the source cluster, obtain the details of a migrated namespace:

   ```
   $ oc get namespace <namespace> -o yaml 1
   ```

1 Specify the migrated namespace.

3. On the target cluster, edit the migrated namespace:

```
$ oc edit namespace <namespace>
```

4. Add the missing **openshift.io/node-selector** annotations to the migrated namespace as in the following example:

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    openshift.io/node-selector: "region=east"
...
```

5. Run the migration plan again.

12.4.2. Error messages and resolutions

This section describes common error messages you might encounter with the Migration Toolkit for Containers (MTC) and how to resolve their underlying causes.

12.4.2.1. CA certificate error displayed when accessing the MTC console for the first time

If a **CA certificate error** message is displayed the first time you try to access the MTC console, the likely cause is the use of self-signed CA certificates in one of the clusters.

To resolve this issue, navigate to the **oauth-authorization-server** URL displayed in the error message and accept the certificate. To resolve this issue permanently, add the certificate to the trust store of your web browser.

If an **Unauthorized** message is displayed after you have accepted the certificate, navigate to the MTC console and refresh the web page.

12.4.2.2. OAuth timeout error in the MTC console

If a **connection has timed out** message is displayed in the MTC console after you have accepted a self-signed certificate, the causes are likely to be the following:

- Interrupted network access to the OAuth server
- Interrupted network access to the OpenShift Container Platform console
- Proxy configuration that blocks access to the **oauth-authorization-server** URL. See MTC console inaccessible because of OAuth timeout error for details.

To determine the cause of the timeout:

- Inspect the MTC console web page with a browser web inspector.
- Check the **Migration UI** pod log for errors.

12.4.2.3. Certificate signed by unknown authority error

If you use a self-signed certificate to secure a cluster or a replication repository for the MTC, certificate verification might fail with the following error message: **Certificate signed by unknown authority**.

You can create a custom CA certificate bundle file and upload it in the MTC web console when you add a cluster or a replication repository.

Procedure

Download a CA certificate from a remote endpoint and save it as a CA bundle file:

```
$ echo -n | openssl s_client -connect <host_FQDN>:<port> \ 1
  | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > <ca_bundle.cert> 2
```

1. Specify the host FQDN and port of the endpoint, for example, **api.my-cluster.example.com:6443**.
2. Specify the name of the CA bundle file.

12.4.2.4. Backup storage location errors in the Velero pod log

If a **Velero Backup** custom resource contains a reference to a backup storage location (BSL) that does not exist, the **Velero** pod log might display the following error messages:

```
$ oc logs <Velero_Pod> -n openshift-migration
```

Example output

```
level=error msg="Error checking repository for stale locks" error="error getting backup storage
location: BackupStorageLocation.velero.io \"ts-dpa-1\" not found" error.file="/remote-
source/src/github.com/vmware-tanzu/velero/pkg/restic/repository_manager.go:259"
```

You can ignore these error messages. A missing BSL cannot cause a migration to fail.

12.4.2.5. Pod volume backup timeout error in the Velero pod log

If a migration fails because Restic times out, the following error is displayed in the **Velero** pod log.

```
level=error msg="Error backing up item" backup=velero/monitoring error="timed out waiting for all
PodVolumeBackups to complete"
error.file="/go/src/github.com/heptio/velero/pkg/restic/backupper.go:165"
error.function="github.com/heptio/velero/pkg/restic.(*backupper).BackupPodVolumes" group=v1
```

The default value of **restic_timeout** is one hour. You can increase this parameter for large migrations, keeping in mind that a higher value may delay the return of error messages.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators → Installed Operators**.
2. Click **Migration Toolkit for Containers Operator**.

3. In the **MigrationController** tab, click **migration-controller**.
4. In the **YAML** tab, update the following parameter value:

```
spec:
  restic_timeout: 1h (1)
```

(1) Valid units are **h** (hours), **m** (minutes), and **s** (seconds), for example, **3h30m15s**.

5. Click **Save**.

12.4.2.6. Restic verification errors in the MigMigration custom resource

If data verification fails when migrating a persistent volume with the file system data copy method, the following error is displayed in the **MigMigration** CR.

Example output

```
status:
  conditions:
  - category: Warn
    durable: true
    lastTransitionTime: 2020-04-16T20:35:16Z
    message: There were verify errors found in 1 Restic volume restores. See restore `<registry-example-migration-rvwcm>`
      for details (1)
    status: "True"
    type: ResticVerifyErrors (2)
```

(1) The error message identifies the **Restore** CR name.

(2) **ResticVerifyErrors** is a general error warning type that includes verification errors.

NOTE

A data verification error does not cause the migration process to fail.

You can check the **Restore** CR to identify the source of the data verification error.

Procedure

1. Log in to the target cluster.
2. View the **Restore** CR:

```
$ oc describe <registry-example-migration-rvwcm> -n openshift-migration
```

The output identifies the persistent volume with **PodVolumeRestore** errors.

Example output

```
status:
  phase: Completed
  podVolumeRestoreErrors:
  - kind: PodVolumeRestore
    name: <registry-example-migration-rvwcm-98t49>
    namespace: openshift-migration
  podVolumeRestoreResticErrors:
  - kind: PodVolumeRestore
    name: <registry-example-migration-rvwcm-98t49>
    namespace: openshift-migration
```

3. View the **PodVolumeRestore** CR:

```
$ oc describe <migration-example-rvwcm-98t49>
```

The output identifies the **Restic** pod that logged the errors.

Example output

```
  completionTimestamp: 2020-05-01T20:49:12Z
  errors: 1
  resticErrors: 1
  ...
  resticPod: <restic-nr2v5>
```

4. View the **Restic** pod log to locate the errors:

```
$ oc logs -f <restic-nr2v5>
```

12.4.2.7. Restic permission error when migrating from NFS storage with root_squash enabled

If you are migrating data from NFS storage and **root_squash** is enabled, **Restic** maps to **nfsnobody** and does not have permission to perform the migration. The following error is displayed in the **Restic** pod log.

Example output

```
backup=openshift-migration/<backup_id> controller=pod-volume-backup error="fork/exec
/usr/bin/restic: permission denied" error.file="/go/src/github.com/vmware-
tanzu/velero/pkg/controller/pod_volume_backup_controller.go:280"
error.function="github.com/vmware-tanzu/velero/pkg/controller.
(*podVolumeBackupController).processBackup"
logSource="pkg/controller/pod_volume_backup_controller.go:280" name=<backup_id>
namespace=openshift-migration
```

You can resolve this issue by creating a supplemental group for Restic and adding the group ID to the **MigrationController** CR manifest.

Procedure

1. Create a supplemental group for Restic on the NFS storage.

2. Set the **setgid** bit on the NFS directories so that group ownership is inherited.
3. Add the **restic_supplemental_groups** parameter to the **MigrationController** CR manifest on the source and target clusters:

```
spec:
  restic_supplemental_groups: <group_id> 1
```

1 Specify the supplemental group ID.

4. Wait for the **Restic** pods to restart so that the changes are applied.

12.4.3. Applying the Skip SELinux relabel workaround with spc_t automatically on workloads running on OpenShift Container Platform

When attempting to migrate a namespace with Migration Toolkit for Containers (MTC) and a substantial volume associated with it, the **rsync-server** may become frozen without any further information to troubleshoot the issue.

12.4.3.1. Diagnosing the need for the Skip SELinux relabel workaround

Search for an error of **Unable to attach or mount volumes for pod…timed out waiting for the condition** in the kubelet logs from the node where the **rsync-server** for the Direct Volume Migration (DVM) runs.

Example kubelet log

```
kubenswrapper[3879]: W0326 16:30:36.749224    3879 volume_linux.go:49] Setting volume ownership for /var/lib/kubelet/pods/8905d88e-6531-4d65-9c2a-eff11dc7eb29/volumes/kubernetes.io~csi/pvc-287d1988-3fd9-4517-a0c7-22539acd31e6/mount and fsGroup set. If the volume has a lot of files then setting volume ownership could be slow, see https://github.com/kubernetes/kubernetes/issues/69699

kubenswrapper[3879]: E0326 16:32:02.706363    3879 kubelet.go:1841] "Unable to attach or mount volumes for pod; skipping pod" err="unmounted volumes=[8db9d5b032dab17d4ea9495af12e085a], unattached volumes=[crane2-rsync-server-secret 8db9d5b032dab17d4ea9495af12e085a kube-api-access-dlbd2 crane2-stunnel-server-config crane2-stunnel-server-secret crane2-rsync-server-config]: timed out waiting for the condition" pod="caboodle-preprod/rsync-server"

kubenswrapper[3879]: E0326 16:32:02.706496    3879 pod_workers.go:965] "Error syncing pod, skipping" err="unmounted volumes=[8db9d5b032dab17d4ea9495af12e085a], unattached volumes=[crane2-rsync-server-secret 8db9d5b032dab17d4ea9495af12e085a kube-api-access-dlbd2 crane2-stunnel-server-config crane2-stunnel-server-secret crane2-rsync-server-config]: timed out waiting for the condition" pod="caboodle-preprod/rsync-server" podUID=8905d88e-6531-4d65-9c2a-eff11dc7eb29
```

12.4.3.2. Resolving using the Skip SELinux relabel workaround

To resolve this issue, set the **migration_rsync_super_privileged** parameter to **true** in both the source and destination **MigClusters** using the **MigrationController** custom resource (CR).

Example MigrationController CR

-

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigrationController
metadata:
  name: migration-controller
  namespace: openshift-migration
spec:
  migration_rsync_super_privileged: true (1)
  azure_resource_group: ""
  cluster_name: host
  mig_namespace_limit: "10"
  mig_pod_limit: "100"
  mig_pv_limit: "100"
  migration_controller: true
  migration_log_reader: true
  migration_ui: true
  migration_velero: true
  olm_managed: true
  restic_timeout: 1h
  version: 1.8.3
```

1. The value of the **migration_rsync_super_privileged** parameter indicates whether or not to run Rsync Pods as *super privileged* containers (**spc_t selinux context**). Valid settings are **true** or **false**.

12.5. ROLLING BACK A MIGRATION

You can roll back a migration by using the MTC web console or the CLI.

You can also roll back a migration manually.

12.5.1. Rolling back a migration by using the MTC web console

You can roll back a migration by using the Migration Toolkit for Containers (MTC) web console.

NOTE

The following resources remain in the migrated namespaces for debugging after a failed direct volume migration (DVM):

- Config maps (source and destination clusters)
- **Secret** objects (source and destination clusters)
- **Rsync** CRs (source cluster)

These resources do not affect rollback. You can delete them manually.

If you later run the same migration plan successfully, the resources from the failed migration are deleted automatically.

If your application was stopped during a failed migration, you must roll back the migration to prevent data corruption in the persistent volume.

Rollback is not required if the application was not stopped during migration because the original application is still running on the source cluster.

Procedure

1. In the MTC web console, click **Migration plans**.
2. Click the Options menu  beside a migration plan and select **Rollback** under **Migration**.
3. Click **Rollback** and wait for rollback to complete.
   In the migration plan details, **Rollback succeeded** is displayed.
4. Verify that rollback was successful in the OpenShift Container Platform web console of the source cluster:
   a. Click **Home → Projects**.
   b. Click the migrated project to view its status.
   c. In the **Routes** section, click **Location** to verify that the application is functioning, if applicable.
   d. Click **Workloads → Pods** to verify that the pods are running in the migrated namespace.
   e. Click **Storage → Persistent volumes** to verify that the migrated persistent volume is correctly provisioned.

12.5.2. Rolling back a migration from the command-line interface

You can roll back a migration by creating a **MigMigration** custom resource (CR) from the command-line interface.

NOTE

The following resources remain in the migrated namespaces for debugging after a failed direct volume migration (DVM):

- Config maps (source and destination clusters)
- **Secret** objects (source and destination clusters)
- **Rsync** CRs (source cluster)

These resources do not affect rollback. You can delete them manually.

If you later run the same migration plan successfully, the resources from the failed migration are deleted automatically.

If your application was stopped during a failed migration, you must roll back the migration to prevent data corruption in the persistent volume.

Rollback is not required if the application was not stopped during migration because the original application is still running on the source cluster.

Procedure

1. Create a **MigMigration** CR based on the following example:

```
$ cat << EOF | oc apply -f -
apiVersion: migration.openshift.io/v1alpha1
kind: MigMigration
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: <migmigration>
  namespace: openshift-migration
spec:
...
  rollback: true
...
  migPlanRef:
    name: <migplan> 1
    namespace: openshift-migration
EOF
```

1 Specify the name of the associated **MigPlan** CR.

2. In the MTC web console, verify that the migrated project resources have been removed from the target cluster.

3. Verify that the migrated project resources are present in the source cluster and that the application is running.

12.5.3. Rolling back a migration manually

You can roll back a failed migration manually by deleting the **stage** pods and unquiescing the application.

If you run the same migration plan successfully, the resources from the failed migration are deleted automatically.

NOTE

The following resources remain in the migrated namespaces after a failed direct volume migration (DVM):

- Config maps (source and destination clusters)
- **Secret** objects (source and destination clusters)
- **Rsync** CRs (source cluster)

These resources do not affect rollback. You can delete them manually.

Procedure

1. Delete the **stage** pods on all clusters:

```
$ oc delete $(oc get pods -l migration.openshift.io/is-stage-pod -n <namespace>) 1
```

1 Namespaces specified in the **MigPlan** CR.

2. Unquiesce the application on the source cluster by scaling the replicas to their premigration number:

```
$ oc scale deployment <deployment> --replicas=<premigration_replicas>
```

The **migration.openshift.io/preQuiesceReplicas** annotation in the **Deployment** CR displays the premigration number of replicas:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
    migration.openshift.io/preQuiesceReplicas: "1"
```

3. Verify that the application pods are running on the source cluster:

```
$ oc get pod -n <namespace>
```

Additional resources

- Deleting Operators from a cluster using the web console