



OpenShift Container Platform 4.9

Migrating from version 3 to 4

Migrating to OpenShift Container Platform 4

OpenShift Container Platform 4.9 Migrating from version 3 to 4

Migrating to OpenShift Container Platform 4

Legal Notice

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

This document provides instructions for migrating your OpenShift Container Platform cluster from version 3 to version 4.

Table of Contents

CHAPTER 1. MIGRATION FROM OPENSIFT CONTAINER PLATFORM 3 TO 4 OVERVIEW	7
1.1. DIFFERENCES BETWEEN OPENSIFT CONTAINER PLATFORM 3 AND 4	7
1.2. PLANNING NETWORK CONSIDERATIONS	7
1.3. INSTALLING MTC	7
1.4. UPGRADING MTC	7
1.5. REVIEWING PREMIGRATION CHECKLISTS	8
1.6. MIGRATING APPLICATIONS	8
1.7. ADVANCED MIGRATION OPTIONS	8
1.8. TROUBLESHOOTING MIGRATIONS	8
1.9. ROLLING BACK A MIGRATION	8
1.10. UNINSTALLING MTC AND DELETING RESOURCES	8
CHAPTER 2. ABOUT MIGRATING FROM OPENSIFT CONTAINER PLATFORM 3 TO 4	9
CHAPTER 3. DIFFERENCES BETWEEN OPENSIFT CONTAINER PLATFORM 3 AND 4	10
3.1. ARCHITECTURE	10
Immutable infrastructure	10
Operators	10
3.2. INSTALLATION AND UPGRADE	10
Installation process	11
Infrastructure options	11
Upgrading your cluster	11
3.3. MIGRATION CONSIDERATIONS	11
3.3.1. Storage considerations	11
Local volume persistent storage	11
FlexVolume persistent storage	11
Container Storage Interface (CSI) persistent storage	12
Red Hat OpenShift Container Storage	12
Unsupported persistent storage options	12
3.3.2. Networking considerations	12
Network isolation mode	12
3.3.3. Logging considerations	12
Deploying OpenShift Logging	13
Aggregated logging data	13
Unsupported logging configurations	13
3.3.4. Security considerations	13
Unauthenticated access to discovery endpoints	13
Identity providers	13
OAuth token storage format	13
3.3.5. Monitoring considerations	13
Alert for monitoring infrastructure availability	13
CHAPTER 4. NETWORK CONSIDERATIONS	15
4.1. DNS CONSIDERATIONS	15
4.1.1. Isolating the DNS domain of the target cluster from the clients	15
4.1.2. Setting up the target cluster to accept the source DNS domain	15
4.2. NETWORK TRAFFIC REDIRECTION STRATEGIES	16
CHAPTER 5. ABOUT THE MIGRATION TOOLKIT FOR CONTAINERS	18
5.1. TERMINOLOGY	18
5.2. MTC WORKFLOW	19
5.3. ABOUT DATA COPY METHODS	22

5.3.1. File system copy method	22
5.3.2. Snapshot copy method	22
5.4. DIRECT VOLUME MIGRATION AND DIRECT IMAGE MIGRATION	23
CHAPTER 6. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS	24
6.1. COMPATIBILITY GUIDELINES	24
6.2. INSTALLING THE LEGACY MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSIFT CONTAINER PLATFORM 3	25
6.3. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSIFT CONTAINER PLATFORM 4.9	26
6.4. PROXY CONFIGURATION	27
6.4.1. Direct volume migration	27
6.4.1.1. TCP proxy setup for DVM	27
6.4.1.2. Why use a TCP proxy instead of an HTTP/HTTPS proxy?	28
6.4.1.3. Known issue	28
6.4.2. Tuning network policies for migrations	28
6.4.2.1. NetworkPolicy configuration	28
6.4.2.1.1. Egress traffic from Rsync pods	28
6.4.2.1.2. Ingress traffic to Rsync pods	29
6.4.2.2. EgressNetworkPolicy configuration	29
6.4.2.3. Choosing alternate endpoints for data transfer	30
6.4.2.4. Configuring supplemental groups for Rsync pods	30
6.4.3. Configuring proxies	30
6.5. CONFIGURING A REPLICATION REPOSITORY	31
6.5.1. Prerequisites	32
6.5.2. Retrieving Multicloud Object Gateway credentials	32
6.5.3. Configuring Amazon Web Services	32
6.5.4. Configuring Google Cloud Platform	34
6.5.5. Configuring Microsoft Azure	36
6.5.6. Additional resources	38
6.6. UNINSTALLING MTC AND DELETING RESOURCES	38
CHAPTER 7. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS IN A RESTRICTED NETWORK ENVIRONMENT	40
7.1. COMPATIBILITY GUIDELINES	40
7.2. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSIFT CONTAINER PLATFORM 4.9	41
7.3. INSTALLING THE LEGACY MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSIFT CONTAINER PLATFORM 3	42
7.4. PROXY CONFIGURATION	44
7.4.1. Direct volume migration	44
7.4.1.1. TCP proxy setup for DVM	44
7.4.1.2. Why use a TCP proxy instead of an HTTP/HTTPS proxy?	44
7.4.1.3. Known issue	45
7.4.2. Tuning network policies for migrations	45
7.4.2.1. NetworkPolicy configuration	45
7.4.2.1.1. Egress traffic from Rsync pods	45
7.4.2.1.2. Ingress traffic to Rsync pods	45
7.4.2.2. EgressNetworkPolicy configuration	46
7.4.2.3. Choosing alternate endpoints for data transfer	46
7.4.2.4. Configuring supplemental groups for Rsync pods	47
7.4.3. Configuring proxies	47
7.5. CONFIGURING A REPLICATION REPOSITORY	48
7.5.1. Prerequisites	48

7.5.2. Retrieving Multicloud Object Gateway credentials	48
7.5.3. Additional resources	48
7.6. UNINSTALLING MTC AND DELETING RESOURCES	49
CHAPTER 8. UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS	51
8.1. UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS ON OPENSIFT CONTAINER PLATFORM 4.9	51
8.2. UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS ON OPENSIFT CONTAINER PLATFORM 3	51
8.3. UPGRADING MTC 1.3 TO 1.7	53
CHAPTER 9. PREMIGRATION CHECKLISTS	54
9.1. RESOURCES	54
9.2. SOURCE CLUSTER	54
9.3. TARGET CLUSTER	55
9.4. PERFORMANCE	56
CHAPTER 10. MIGRATING YOUR APPLICATIONS	57
10.1. MIGRATION PREREQUISITES	57
Additional resources for migration prerequisites	58
10.2. MIGRATING YOUR APPLICATIONS BY USING THE MTC WEB CONSOLE	58
10.2.1. Launching the MTC web console	59
10.2.2. Adding a cluster to the MTC web console	59
10.2.3. Adding a replication repository to the MTC web console	61
10.2.4. Creating a migration plan in the MTC web console	62
Additional resources	64
10.2.5. Running a migration plan in the MTC web console	64
CHAPTER 11. ADVANCED MIGRATION OPTIONS	66
11.1. TERMINOLOGY	66
11.2. MIGRATING APPLICATIONS BY USING THE COMMAND LINE	67
11.2.1. Migration prerequisites	67
11.2.2. Creating a registry route for direct image migration	68
11.2.3. Proxy configuration	68
11.2.3.1. Direct volume migration	69
11.2.3.1.1. TCP proxy setup for DVM	69
11.2.3.1.2. Why use a TCP proxy instead of an HTTP/HTTPS proxy?	69
11.2.3.1.3. Known issue	69
11.2.3.2. Tuning network policies for migrations	70
11.2.3.2.1. NetworkPolicy configuration	70
11.2.3.2.1.1. Egress traffic from Rsync pods	70
11.2.3.2.1.2. Ingress traffic to Rsync pods	70
11.2.3.2.2. EgressNetworkPolicy configuration	71
11.2.3.2.3. Choosing alternate endpoints for data transfer	71
11.2.3.2.4. Configuring supplemental groups for Rsync pods	71
11.2.3.3. Configuring proxies	72
11.2.4. Migrating an application by using the MTC API	73
11.2.5. State migration	78
Additional resources	80
11.3. MIGRATION HOOKS	80
11.3.1. Writing an Ansible playbook for a migration hook	81
11.3.1.1. Ansible modules	81
11.3.1.2. Environment variables	82
11.4. MIGRATION PLAN OPTIONS	82

11.4.1. Excluding resources	82
11.4.2. Mapping namespaces	83
11.4.3. Excluding persistent volume claims	84
11.4.4. Mapping persistent volume claims	84
11.4.5. Editing persistent volume attributes	85
Additional resources	87
11.4.6. Performing a state migration of Kubernetes objects by using the MTC API	87
11.5. MIGRATION CONTROLLER OPTIONS	89
11.5.1. Increasing limits for large migrations	89
11.5.2. Enabling persistent volume resizing for direct volume migration	90
11.5.3. Enabling cached Kubernetes clients	91
CHAPTER 12. TROUBLESHOOTING	93
12.1. MTC WORKFLOW	93
About MTC custom resources	95
12.2. MTC CUSTOM RESOURCE MANIFESTS	96
12.2.1. DirectImageMigration	96
12.2.2. DirectImageStreamMigration	97
12.2.3. DirectVolumeMigration	97
12.2.4. DirectVolumeMigrationProgress	98
12.2.5. MigAnalytic	98
12.2.6. MigCluster	99
12.2.7. MigHook	100
12.2.8. MigMigration	100
12.2.9. MigPlan	101
12.2.10. MigStorage	103
12.3. LOGS AND DEBUGGING TOOLS	104
12.3.1. Viewing migration plan resources	104
12.3.2. Viewing a migration plan log	104
12.3.3. Using the migration log reader	105
12.3.4. Accessing performance metrics	105
12.3.4.1. Provided metrics	106
12.3.4.1.1. cam_app_workload_migrations	106
12.3.4.1.2. mtc_client_request_count	106
12.3.4.1.3. mtc_client_request_elapsed	106
12.3.4.1.4. Useful queries	107
12.3.5. Using the must-gather tool	107
Viewing metrics data with the Prometheus console	108
12.3.6. Debugging Velero resources with the Velero CLI tool	109
Syntax	109
Help option	109
Describe command	109
Logs command	109
12.3.7. Debugging a partial migration failure	110
12.3.8. Using MTC custom resources for troubleshooting	111
12.4. COMMON ISSUES AND CONCERNS	114
12.4.1. Updating deprecated internal images	114
12.4.2. Direct volume migration does not complete	116
12.4.3. Error messages and resolutions	117
12.4.3.1. CA certificate error displayed when accessing the MTC console for the first time	117
12.4.3.2. OAuth timeout error in the MTC console	117
12.4.3.3. Certificate signed by unknown authority error	117
12.4.3.4. Backup storage location errors in the Velero pod log	118

12.4.3.5. Pod volume backup timeout error in the Velero pod log	118
12.4.3.6. Restic verification errors in the MigMigration custom resource	119
12.4.3.7. Restic permission error when migrating from NFS storage with root_squash enabled	120
12.4.4. Known issues	121
12.5. ROLLING BACK A MIGRATION	121
12.5.1. Rolling back a migration by using the MTC web console	121
12.5.2. Rolling back a migration from the command line interface	122
12.5.3. Rolling back a migration manually	123
Additional resources	124

CHAPTER 1. MIGRATION FROM OPENSIFT CONTAINER PLATFORM 3 TO 4 OVERVIEW

OpenShift Container Platform 4 clusters are different from OpenShift Container Platform 3 clusters. OpenShift Container Platform 4 clusters contain new technologies and functionality that result in a cluster that is self-managing, flexible, and automated. To learn more about migrating from OpenShift Container Platform 3 to 4 see [About migrating from OpenShift Container Platform 3 to 4](#) .

1.1. DIFFERENCES BETWEEN OPENSIFT CONTAINER PLATFORM 3 AND 4

Before migrating from OpenShift Container Platform 3 to 4, you can check [differences between OpenShift Container Platform 3 and 4](#). Review the following information:

- [Architecture](#)
- [Installation and update](#)
- [Storage, network, logging, security, and monitoring considerations](#)

1.2. PLANNING NETWORK CONSIDERATIONS

Before migrating from OpenShift Container Platform 3 to 4, review the [differences between OpenShift Container Platform 3 and 4](#) for information about the following areas:

- [DNS considerations](#)
 - [Isolating the DNS domain of the target cluster from the clients](#) .
 - [Setting up the target cluster to accept the source DNS domain](#) .

You can migrate stateful application workloads from OpenShift Container Platform 3 to 4 at the granularity of a namespace. To learn more about MTC see [Understanding MTC](#).



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](#).

1.3. INSTALLING MTC

Review the following tasks to install the MTC:

1. [Install the Migration Toolkit for Containers Operator on target cluster by using Operator Lifecycle Manager \(OLM\)](#).
2. [Install the legacy Migration Toolkit for Containers Operator on the source cluster manually](#) .
3. [Configure object storage to use as a replication repository](#) .

1.4. UPGRADING MTC

You [upgrade the Migration Toolkit for Containers \(MTC\)](#) on OpenShift Container Platform 4.9 by using OLM. You upgrade MTC on OpenShift Container Platform 3 by reinstalling the legacy Migration Toolkit for Containers Operator.

1.5. REVIEWING PREMIGRATION CHECKLISTS

Before you migrate your application workloads with the Migration Toolkit for Containers (MTC), review the [premigration checklists](#).

1.6. MIGRATING APPLICATIONS

You can migrate your applications by using the MTC [web console](#) or [the command line](#).

1.7. ADVANCED MIGRATION OPTIONS

You can automate your migrations and modify MTC custom resources to improve the performance of large-scale migrations by using the following options:

- [Running a state migration](#)
- [Creating migration hooks](#)
- [Editing, excluding, and mapping migrated resources](#)
- [Configuring the migration controller for large migrations](#)

1.8. TROUBLESHOOTING MIGRATIONS

You can perform the following troubleshooting tasks:

- [Viewing migration plan resources by using the MTC web console](#)
- [Viewing the migration plan aggregated log file](#)
- [Using the migration log reader](#)
- [Accessing performance metrics](#)
- [Using the **must-gather** tool](#)
- [Using the Velero CLI to debug **Backup** and **Restore** CRs](#)
- [Using MTC custom resources for troubleshooting](#)
- [Checking common issues and concerns](#)

1.9. ROLLING BACK A MIGRATION

You can [roll back a migration](#) by using the MTC web console, by using the CLI, or manually.

1.10. UNINSTALLING MTC AND DELETING RESOURCES

You can [uninstall the MTC and delete its resources](#) to clean up the cluster.

CHAPTER 2. ABOUT MIGRATING FROM OPENSIFT CONTAINER PLATFORM 3 TO 4

OpenShift Container Platform 4 contains new technologies and functionality that result in a cluster that is self-managing, flexible, and automated. OpenShift Container Platform 4 clusters are deployed and managed very differently from OpenShift Container Platform 3.

The most effective way to migrate from OpenShift Container Platform 3 to 4 is by using a CI/CD pipeline to automate deployments in an [application lifecycle management](#) framework.

If you do not have a CI/CD pipeline or if you are migrating stateful applications, you can use the Migration Toolkit for Containers (MTC) to migrate your application workloads.

To successfully transition to OpenShift Container Platform 4, review the following information:

Differences between OpenShift Container Platform 3 and 4

- Architecture
- Installation and upgrade
- Storage, network, logging, security, and monitoring considerations

About the Migration Toolkit for Containers

- Workflow
- File system and snapshot copy methods for persistent volumes (PVs)
- Direct volume migration
- Direct image migration

Advanced migration options

- Automating your migration with migration hooks
- Using the MTC API
- Excluding resources from a migration plan
- Configuring the **MigrationController** custom resource for large-scale migrations
- Enabling automatic PV resizing for direct volume migration
- Enabling cached Kubernetes clients for improved performance

For new features and enhancements, technical changes, and known issues, see the [MTC release notes](#).

CHAPTER 3. DIFFERENCES BETWEEN OPENSIFT CONTAINER PLATFORM 3 AND 4

OpenShift Container Platform 4.9 introduces architectural changes and enhancements/ The procedures that you used to manage your OpenShift Container Platform 3 cluster might not apply to OpenShift Container Platform 4.

For information on configuring your OpenShift Container Platform 4 cluster, review the appropriate sections of the OpenShift Container Platform documentation. For information on new features and other notable technical changes, review the [OpenShift Container Platform 4.9 release notes](#).

It is not possible to upgrade your existing OpenShift Container Platform 3 cluster to OpenShift Container Platform 4. You must start with a new OpenShift Container Platform 4 installation. Tools are available to assist in migrating your control plane settings and application workloads.

3.1. ARCHITECTURE

With OpenShift Container Platform 3, administrators individually deployed Red Hat Enterprise Linux (RHEL) hosts, and then installed OpenShift Container Platform on top of these hosts to form a cluster. Administrators were responsible for properly configuring these hosts and performing updates.

OpenShift Container Platform 4 represents a significant change in the way that OpenShift Container Platform clusters are deployed and managed. OpenShift Container Platform 4 includes new technologies and functionality, such as Operators, machine sets, and Red Hat Enterprise Linux CoreOS (RHCOS), which are core to the operation of the cluster. This technology shift enables clusters to self-manage some functions previously performed by administrators. This also ensures platform stability and consistency, and simplifies installation and scaling.

For more information, see [OpenShift Container Platform architecture](#).

Immutable infrastructure

OpenShift Container Platform 4 uses Red Hat Enterprise Linux CoreOS (RHCOS), which is designed to run containerized applications, and provides efficient installation, Operator-based management, and simplified upgrades. RHCOS is an immutable container host, rather than a customizable operating system like RHEL. RHCOS enables OpenShift Container Platform 4 to manage and automate the deployment of the underlying container host. RHCOS is a part of OpenShift Container Platform, which means that everything runs inside a container and is deployed using OpenShift Container Platform.

In OpenShift Container Platform 4, control plane nodes must run RHCOS, ensuring that full-stack automation is maintained for the control plane. This makes rolling out updates and upgrades a much easier process than in OpenShift Container Platform 3.

For more information, see [Red Hat Enterprise Linux CoreOS \(RHCOS\)](#).

Operators

Operators are a method of packaging, deploying, and managing a Kubernetes application. Operators ease the operational complexity of running another piece of software. They watch over your environment and use the current state to make decisions in real time. Advanced Operators are designed to upgrade and react to failures automatically.

For more information, see [Understanding Operators](#).

3.2. INSTALLATION AND UPGRADE

Installation process

To install OpenShift Container Platform 3.11, you prepared your Red Hat Enterprise Linux (RHEL) hosts, set all of the configuration values your cluster needed, and then ran an Ansible playbook to install and set up your cluster.

In OpenShift Container Platform 4.9, you use the OpenShift installation program to create a minimum set of resources required for a cluster. After the cluster is running, you use Operators to further configure your cluster and to install new services. After first boot, Red Hat Enterprise Linux CoreOS (RHCOS) systems are managed by the Machine Config Operator (MCO) that runs in the OpenShift Container Platform cluster.

For more information, see [Installation process](#).

If you want to add Red Hat Enterprise Linux (RHEL) worker machines to your OpenShift Container Platform 4.9 cluster, you use an Ansible playbook to join the RHEL worker machines after the cluster is running. For more information, see [Adding RHEL compute machines to an OpenShift Container Platform cluster](#).

Infrastructure options

In OpenShift Container Platform 3.11, you installed your cluster on infrastructure that you prepared and maintained. In addition to providing your own infrastructure, OpenShift Container Platform 4 offers an option to deploy a cluster on infrastructure that the OpenShift Container Platform installation program provisions and the cluster maintains.

For more information, see [OpenShift Container Platform installation overview](#).

Upgrading your cluster

In OpenShift Container Platform 3.11, you upgraded your cluster by running Ansible playbooks. In OpenShift Container Platform 4.9, the cluster manages its own updates, including updates to Red Hat Enterprise Linux CoreOS (RHCOS) on cluster nodes. You can easily upgrade your cluster by using the web console or by using the **oc adm upgrade** command from the OpenShift CLI and the Operators will automatically upgrade themselves. If your OpenShift Container Platform 4.9 cluster has RHEL worker machines, then you will still need to run an Ansible playbook to upgrade those worker machines.

For more information, see [Updating clusters](#).

3.3. MIGRATION CONSIDERATIONS

Review the changes and other considerations that might affect your transition from OpenShift Container Platform 3.11 to OpenShift Container Platform 4.

3.3.1. Storage considerations

Review the following storage changes to consider when transitioning from OpenShift Container Platform 3.11 to OpenShift Container Platform 4.9.

Local volume persistent storage

Local storage is only supported by using the Local Storage Operator in OpenShift Container Platform 4.9. It is not supported to use the local provisioner method from OpenShift Container Platform 3.11.

For more information, see [Persistent storage using local volumes](#).

FlexVolume persistent storage

The FlexVolume plugin location changed from OpenShift Container Platform 3.11. The new location in OpenShift Container Platform 4.9 is **/etc/kubernetes/kubelet-plugins/volume/exec**. Attachable FlexVolume plugins are no longer supported.

For more information, see [Persistent storage using FlexVolume](#).

Container Storage Interface (CSI) persistent storage

Persistent storage using the Container Storage Interface (CSI) was [Technology Preview](#) in OpenShift Container Platform 3.11. OpenShift Container Platform 4.9 ships with [several CSI drivers](#). You can also install your own driver.

For more information, see [Persistent storage using the Container Storage Interface \(CSI\)](#).

Red Hat OpenShift Container Storage

Red Hat OpenShift Container Storage 3, which is available for use with OpenShift Container Platform 3.11, uses Red Hat Gluster Storage as the backing storage.

Red Hat OpenShift Container Storage 4, which is available for use with OpenShift Container Platform 4, uses Red Hat Ceph Storage as the backing storage.

For more information, see [Persistent storage using Red Hat OpenShift Container Storage](#) and the [interoperability matrix](#) article.

Unsupported persistent storage options

Support for the following persistent storage options from OpenShift Container Platform 3.11 has changed in OpenShift Container Platform 4.9:

- GlusterFS is no longer supported.
- CephFS as a standalone product is no longer supported.
- Ceph RBD as a standalone product is no longer supported.

If you used one of these in OpenShift Container Platform 3.11, you must choose a different persistent storage option for full support in OpenShift Container Platform 4.9.

For more information, see [Understanding persistent storage](#).

3.3.2. Networking considerations

Review the following networking changes to consider when transitioning from OpenShift Container Platform 3.11 to OpenShift Container Platform 4.9.

Network isolation mode

The default network isolation mode for OpenShift Container Platform 3.11 was **ovs-subnet**, though users frequently switched to use **ovn-multitenant**. The default network isolation mode for OpenShift Container Platform 4.9 is controlled by a network policy.

If your OpenShift Container Platform 3.11 cluster used the **ovs-subnet** or **ovs-multitenant** mode, it is recommended to switch to a network policy for your OpenShift Container Platform 4.9 cluster. Network policies are supported upstream, are more flexible, and they provide the functionality that **ovs-multitenant** does. If you want to maintain the **ovs-multitenant** behavior while using a network policy in OpenShift Container Platform 4.9, follow the steps to [configure multitenant isolation using network policy](#).

For more information, see [About network policy](#).

3.3.3. Logging considerations

Review the following logging changes to consider when transitioning from OpenShift Container Platform 3.11 to OpenShift Container Platform 4.9.

Deploying OpenShift Logging

OpenShift Container Platform 4 provides a simple deployment mechanism for OpenShift Logging, by using a Cluster Logging custom resource.

For more information, see [Installing OpenShift Logging](#).

Aggregated logging data

You cannot transition your aggregate logging data from OpenShift Container Platform 3.11 into your new OpenShift Container Platform 4 cluster.

For more information, see [About OpenShift Logging](#).

Unsupported logging configurations

Some logging configurations that were available in OpenShift Container Platform 3.11 are no longer supported in OpenShift Container Platform 4.9.

For more information on the explicitly unsupported logging cases, see [Maintenance and support](#).

3.3.4. Security considerations

Review the following security changes to consider when transitioning from OpenShift Container Platform 3.11 to OpenShift Container Platform 4.9.

Unauthenticated access to discovery endpoints

In OpenShift Container Platform 3.11, an unauthenticated user could access the discovery endpoints (for example, `/api/*` and `/apis/*`). For security reasons, unauthenticated access to the discovery endpoints is no longer allowed in OpenShift Container Platform 4.9. If you do need to allow unauthenticated access, you can configure the RBAC settings as necessary; however, be sure to consider the security implications as this can expose internal cluster components to the external network.

Identity providers

Configuration for identity providers has changed for OpenShift Container Platform 4, including the following notable changes:

- The request header identity provider in OpenShift Container Platform 4.9 requires mutual TLS, where in OpenShift Container Platform 3.11 it did not.
- The configuration of the OpenID Connect identity provider was simplified in OpenShift Container Platform 4.9. It now obtains data, which previously had to be specified in OpenShift Container Platform 3.11, from the provider's `/.well-known/openid-configuration` endpoint.

For more information, see [Understanding identity provider configuration](#).

OAuth token storage format

Newly created OAuth HTTP bearer tokens no longer match the names of their OAuth access token objects. The object names are now a hash of the bearer token and are no longer sensitive. This reduces the risk of leaking sensitive information.

3.3.5. Monitoring considerations

Review the following monitoring changes to consider when transitioning from OpenShift Container Platform 3.11 to OpenShift Container Platform 4.9.

Alert for monitoring infrastructure availability

The default alert that triggers to ensure the availability of the monitoring structure was called **DeadMansSwitch** in OpenShift Container Platform 3.11. This was renamed to **Watchdog** in OpenShift Container Platform 4. If you had PagerDuty integration set up with this alert in OpenShift Container Platform 3.11, you must set up the PagerDuty integration for the **Watchdog** alert in OpenShift Container Platform 4.

For more information, see [Applying custom Alertmanager configuration](#).

CHAPTER 4. NETWORK CONSIDERATIONS

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

4.1. 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.

4.1.1. 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.

4.1.2. 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.

4.2. 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.

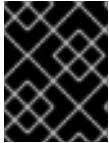
- User-based redirection of traffic for individual applications

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 5. ABOUT THE MIGRATION TOOLKIT FOR CONTAINERS

The Migration Toolkit for Containers (MTC) enables you to migrate stateful application workloads from OpenShift Container Platform 3 to 4.9 at the granularity of a namespace.



IMPORTANT

Before you begin your migration, be sure to review the [differences between OpenShift Container Platform 3 and 4](#).

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 an [OpenShift Container Platform 3 source cluster or on a remote cluster](#).

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.

The service catalog is deprecated in OpenShift Container Platform 4. You can migrate workload resources provisioned with the service catalog from OpenShift Container Platform 3 to 4 but you cannot perform service catalog actions such as **provision**, **deprovision**, or **update** on these workloads after migration. The MTC console displays a message if the service catalog resources cannot be migrated.

5.1. TERMINOLOGY

Table 5.1. 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.

Term	Definition
Remote cluster	<p>A remote cluster is usually the source cluster but this is not required.</p> <p>A remote cluster requires a Secret custom resource that contains the migration-controller service account token.</p> <p>A remote cluster requires an exposed secure registry route for direct image migration.</p>
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	<p>Data is copied to the destination cluster without stopping the application.</p> <p>Running a stage migration multiple times reduces the duration of the cutover migration.</p>
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.

5.2. MTC WORKFLOW

You can migrate Kubernetes resources, persistent volume data, and internal container images to OpenShift Container Platform 4.9 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.

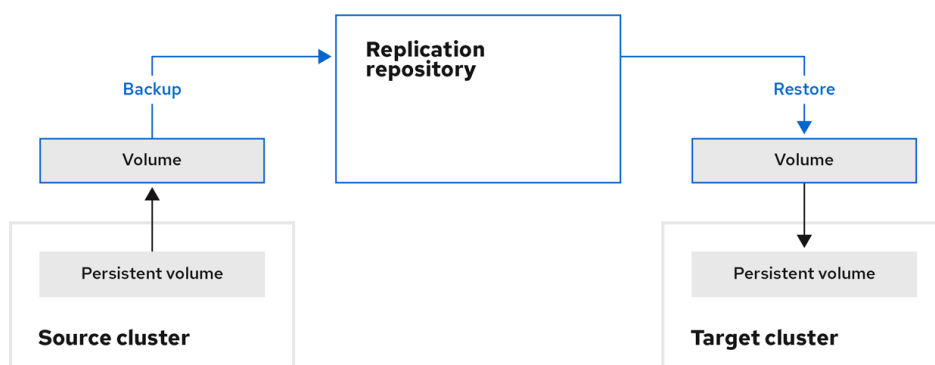
Migrating an application with the MTC web console involves the following steps:

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.



OpenShift_45_1019

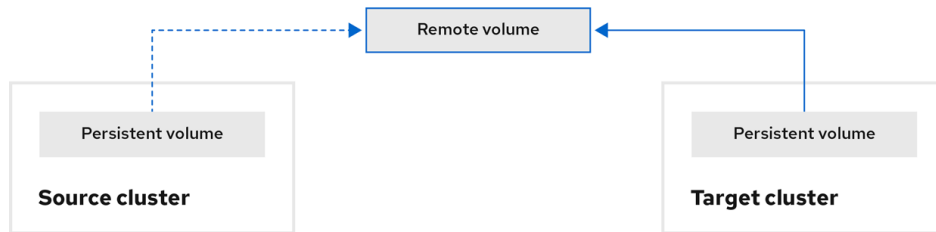
- **Move:** MTC unmounts a remote volume, for example, NFS, from the source cluster, creates a PV resource on the target cluster pointing to the remote volume, and then mounts the remote volume on the target cluster. Applications running on the target cluster use the

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



NOTE

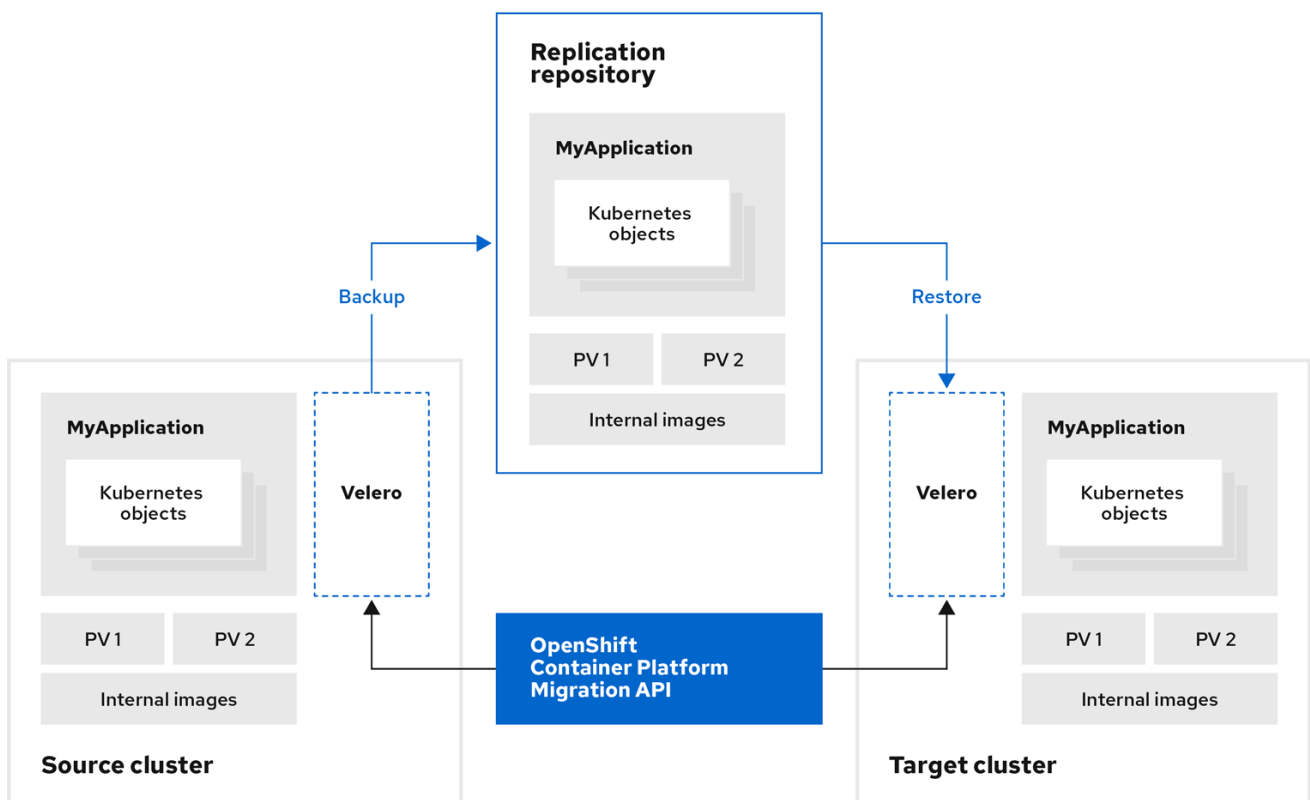
Although the replication repository does not appear in this diagram, it is required for migration.



OpenShift_45_1019

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.



OpenShift_45_1019

5.3. 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.

5.3.1. 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 5.2. File system copy method summary

Benefits	Limitations
<ul style="list-style-type: none"> • 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. 	<ul style="list-style-type: none"> • 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.

5.3.2. 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 Provider, and Microsoft Azure.

Table 5.3. Snapshot copy method summary

Benefits	Limitations
----------	-------------

Benefits	Limitations
<ul style="list-style-type: none"> ● Faster than the file system copy method. 	<ul style="list-style-type: none"> ● 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.

5.4. 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 6. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS

You can install the Migration Toolkit for Containers (MTC) on OpenShift Container Platform 3 and 4.

After you install the Migration Toolkit for Containers Operator on OpenShift Container Platform 4.9 by using the Operator Lifecycle Manager, you manually install 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 [source cluster](#) or 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](#).

6.1. 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.

Table 6.1. MTC compatibility: Migrating from a legacy platform

OpenShift Container Platform 4.5 or earlier	OpenShift Container Platform 4.6 or later
---	---

	OpenShift Container Platform 4.5 or earlier	OpenShift Container Platform 4.6 or later
Stable MTC version	MTC 1.7.z Legacy 1.7 operator: Install manually with the operator.yml file.	MTC 1.7.z Install with OLM, release channel release-v1.7
	 <p>IMPORTANT</p> <p>This cluster cannot be the control cluster.</p>	



NOTE

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 1.7, 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.

6.2. INSTALLING THE LEGACY MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSIFT CONTAINER PLATFORM 3

You can install the legacy Migration Toolkit for Containers Operator manually on OpenShift Container Platform 3.

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 create an [image stream secret](#) and copy it to each node in the cluster.

Procedure

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

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

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

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

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

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

- Log in to your source cluster.
- Verify that the cluster can authenticate with **registry.redhat.io**:

```
$ oc run test --image registry.redhat.io/ubi8 --command sleep infinity
```

- 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
```

- 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.

- Create the **MigrationController** object:

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

- Verify that the MTC pods are running:

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

6.3. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSIFT CONTAINER PLATFORM 4.9

You install the Migration Toolkit for Containers Operator on OpenShift Container Platform 4.9 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.

6.4. 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 4.9, the Migration Toolkit for Containers (MTC) inherits the cluster-wide proxy settings. You can change the proxy parameters if you want to override the cluster-wide proxy settings.

6.4.1. 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.

6.4.1.1. 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.

6.4.1.2. 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.

6.4.1.3. 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.

6.4.2. 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.

6.4.2.1. NetworkPolicy configuration

6.4.2.1.1. 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

```

6.4.2.1.2. 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

```

6.4.2.2. 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

```

6.4.2.3. 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]
```

6.4.2.4. 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 6.2. 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"
```

6.4.3. Configuring proxies

Prerequisites

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

Procedure

- 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](#).

6.5. 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. You can select a method that is suited for your environment and is supported by your storage provider.

The following storage providers are supported:

- [Multicloud Object Gateway](#)
- [Amazon Web Services S3](#)
- [Google Cloud Platform](#)
- [Microsoft Azure Blob](#)

- Generic S3 object storage, for example, Minio or Ceph S3

6.5.1. 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.

6.5.2. Retrieving Multicloud Object Gateway credentials

You must retrieve the Multicloud Object Gateway (MCG) credentials and S3 endpoint in order to configure MCG as a replication repository for the Migration Toolkit for Containers (MTC). You must retrieve the Multicloud Object Gateway (MCG) credentials in order to create a **Secret** custom resource (CR) for the OpenShift API for Data Protection (OADP).

MCG is a component of OpenShift Container Storage.

Prerequisites

- You must deploy OpenShift Container Storage by using the appropriate [OpenShift Container Storage deployment guide](#).

Procedure

1. Obtain the S3 endpoint, **AWS_ACCESS_KEY_ID**, and **AWS_SECRET_ACCESS_KEY** by running the [describe command](#) on the **NooBaa** custom resource. You use these credentials to add MCG as a replication repository.

6.5.3. Configuring Amazon Web Services

You configure Amazon Web Services (AWS) S3 object storage as a replication repository for the Migration Toolkit for Containers (MTC).

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 1
```

- 1** **us-east-1** does not support a **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 1
```

- 1** 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

```

Example output

```

{
  "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.

6.5.4. Configuring Google Cloud Platform

You configure a Google Cloud Platform (GCP) 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 GCP 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 GCP:

```
$ 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
)

```

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 GCP as a replication repository.

6.5.5. 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 ❶
```

- ❶ 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` \
  AZURE_CLIENT_SECRET=`az ad sp create-for-rbac --name "velero" \
  --role "Contributor" --query 'password' -o tsv` \
  AZURE_CLIENT_ID=`az ad sp list --display-name "velero" \
  --query '[0].appId' -o tsv`
```

9. 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
```

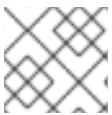
You use the **credentials-velero** file to add Azure as a replication repository.

6.5.6. Additional resources

- [MTC workflow](#)
- [About data copy methods](#)
- [Adding a replication repository to the MTC web console](#)

6.6. 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 7. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS IN A RESTRICTED NETWORK ENVIRONMENT

You can install the Migration Toolkit for Containers (MTC) on OpenShift Container Platform 3 and 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 Operator on the source cluster.
2. Install the Migration Toolkit for Containers Operator on the OpenShift Container Platform 4.9 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 [source cluster](#) or on a [remote cluster](#).
3. Install the *legacy* Migration Toolkit for Containers Operator on the OpenShift Container Platform 3 source cluster from the command line interface.
4. Configure object storage to use as a replication repository.

To uninstall MTC, see [Uninstalling MTC and deleting resources](#).

7.1. 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.

Table 7.1. MTC compatibility: Migrating from a legacy platform

	OpenShift Container Platform 4.5 or earlier	OpenShift Container Platform 4.6 or later
Stable MTC version	MTC 1.7.z Legacy 1.7 operator: Install manually with the operator.yml file.	MTC 1.7.z Install with OLM, release channel release-v1.7
	 <p>IMPORTANT</p> <p>This cluster cannot be the control cluster.</p>	



NOTE

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 1.7, 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.

7.2. INSTALLING THE MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSIFT CONTAINER PLATFORM 4.9

You install the Migration Toolkit for Containers Operator on OpenShift Container Platform 4.9 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.

7.3. INSTALLING THE LEGACY MIGRATION TOOLKIT FOR CONTAINERS OPERATOR ON OPENSIFT CONTAINER PLATFORM 3

You can install the legacy Migration Toolkit for Containers Operator manually on OpenShift Container Platform 3.

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 create an **image stream secret** and copy it to each node in the cluster.
- 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 4.9.

Procedure

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

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

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

```
$ sudo podman cp $(sudo 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:

```
$ sudo podman cp $(sudo 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@sha256:468a6126f73b1ee12085ca53a312d1f96ef5a2ca03442bcb63724af5e2614e8
a=<registry.apps.example.com>/rhmtc/openshift-migration-legacy-rhel8-operator
```

- 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> ❶
  ...
  - name: operator
    image: <registry.apps.example.com>/rhmtc/openshift-migration-legacy-rhel8-
operator@sha256:
<468a6126f73b1ee12085ca53a312d1f96ef5a2ca03442bcb63724af5e2614e8a> ❷
  ...
  env:
    - name: REGISTRY
      value: <registry.apps.example.com> ❸
```

❶ ❷ Specify your mirror registry and the **sha256** value of the Operator image.

❸ Specify your mirror registry.

- Log in to your source cluster.
- 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 ❶
Error from server (AlreadyExists): error when creating "./operator.yml":
rolebindings.rbac.authorization.k8s.io "system:image-pullers" already exists
```

❶ 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
```

7.4. 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 4.9, the Migration Toolkit for Containers (MTC) inherits the cluster-wide proxy settings. You can change the proxy parameters if you want to override the cluster-wide proxy settings.

7.4.1. 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.

7.4.1.1. 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.

7.4.1.2. 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.

7.4.1.3. 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.

7.4.2. 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.

7.4.2.1. NetworkPolicy configuration

7.4.2.1.1. 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
```

7.4.2.1.2. 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

```

7.4.2.2. 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

```

7.4.2.3. 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]

```

7.4.2.4. 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 7.2. 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"
```

7.4.3. Configuring proxies

Prerequisites

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

Procedure

- Get the **MigrationController** CR manifest:

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

- 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> ❶
  noProxy: example.com ❷
```

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

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](#).

7.5. 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.

7.5.1. 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.

7.5.2. Retrieving Multicloud Object Gateway credentials

You must retrieve the Multicloud Object Gateway (MCG) credentials in order to create a **Secret** custom resource (CR) for the OpenShift API for Data Protection (OADP).

MCG is a component of OpenShift Container Storage.

Prerequisites

- You must deploy OpenShift Container Storage by using the appropriate [OpenShift Container Storage deployment guide](#).

Procedure

1. Obtain the S3 endpoint, `AWS_ACCESS_KEY_ID`, and `AWS_SECRET_ACCESS_KEY` by running the [describe command](#) on the **NooBaa** custom resource.

7.5.3. Additional resources

- [Disconnected environment](#) in the Red Hat OpenShift Container Storage documentation.
- [MTC workflow](#)

- [About data copy methods](#)
- [Adding a replication repository to the MTC web console](#)

7.6. 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 8. UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS

You can upgrade the Migration Toolkit for Containers (MTC) on OpenShift Container Platform 4.9 by using Operator Lifecycle Manager.

You can upgrade MTC on OpenShift Container Platform 3 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).

8.1. UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS ON OPENSIFT CONTAINER PLATFORM 4.9

You can upgrade the Migration Toolkit for Containers (MTC) on OpenShift Container Platform 4.9 by using the Operator Lifecycle Manager.

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.

8.2. UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS ON OPENSIFT CONTAINER PLATFORM 3

You can upgrade Migration Toolkit for Containers (MTC) on OpenShift Container Platform 3 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:

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

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

```
$ sudo podman cp $(sudo 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:

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

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

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

9. If you have previously added the OpenShift Container Platform 3 cluster to the MTC web console, you must update the service account token in the web console because the upgrade process deletes and restores the **openshift-migration** namespace:

- a. Obtain the service account token by entering the following command:

```
$ oc sa get-token migration-controller -n openshift-migration
```


b. In the MTC web console, click **Clusters**.

c. Click the Options menu  next to the cluster and select **Edit**.

d. Enter the new service account token in the **Service account token** field.

e. Click **Update cluster** and then click **Close**.

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

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

8.3. UPGRADING MTC 1.3 TO 1.7

If you are upgrading Migration Toolkit for Containers (MTC) version 1.3.x to 1.7, 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**.

Prerequisites

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

Procedure

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

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

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

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

- Replace the **MigPlan** CR manifest to apply the changes:

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

- Get the updated **MigPlan** CR manifest to verify the changes:

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

CHAPTER 9. PREMIGRATION CHECKLISTS

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

9.1. RESOURCES

- If your application uses an internal service network or an external route for communicating with services, the relevant route exists.
- If your application uses cluster-level resources, you have re-created them on the target cluster.
- You have [excluded](#) persistent volumes (PVs), image streams, and other resources that you do not want to migrate.
- PV data has been backed up in case an application displays unexpected behavior after migration and corrupts the data.

9.2. SOURCE CLUSTER

- The cluster meets the [minimum hardware requirements](#).
- You have installed the correct legacy Migration Toolkit for Containers Operator version:
 - **operator-3.7.yml** on OpenShift Container Platform version 3.7.
 - **operator.yml** on OpenShift Container Platform versions 3.9 to 4.5.
- All nodes have an active OpenShift Container Platform subscription.
- You have performed all the [run-once tasks](#).
- You have performed all the [environment health checks](#).
- You have checked for 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'
```

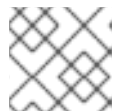
Even if the pods are in a **Running** state, a high restart count might indicate underlying problems.

- You have removed old builds, deployments, and images from each namespace to be migrated by [pruning](#).
- The internal registry uses a [supported storage type](#).
- Direct image migration only: The internal registry is [exposed](#) to external traffic.
- 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.
- 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 [identity provider](#) is working.

9.3. TARGET CLUSTER

- You have installed Migration Toolkit for Containers Operator version 1.5.1.
- All [MTC prerequisites](#) are met.
- The cluster meets the minimum hardware requirements for the specific platform and installation method, for example, on [bare metal](#).
- The cluster has [storage classes](#) defined for the storage types used by the source cluster, for example, block volume, file system, or object storage.



NOTE

NFS does not require a defined storage class.

- 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.
If an application uses an internal image in the **openshift** namespace that is not supported by OpenShift Container Platform 4.9, you can manually update the [OpenShift Container Platform 3 image stream tag](#) with **podman**.
- The target cluster and the replication repository have sufficient storage space.
- The [identity provider](#) is working.
- DNS records for your application exist on the target cluster.

- Set the value of the **annotation.openshift.io/host.generated** parameter to **true** for each OpenShift Container Platform route to update its host name for the target cluster. Otherwise, the migrated routes retain the source cluster host name.
- Certificates that your application uses exist on the target cluster.
- You have configured appropriate firewall rules on the target cluster.
- You have correctly configured load balancing on the target cluster.
- If you migrate objects to an existing namespace on the target cluster that has the same name as the namespace being migrated from the source, the target namespace contains no objects of the same name and type as the objects being migrated.

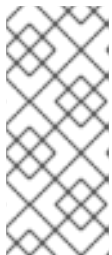


NOTE

Do not create namespaces for your application on the target cluster before migration because this might cause quotas to change.

9.4. PERFORMANCE

- 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 [memory and CPU usage](#) of the nodes are healthy.
- The [etcd disk performance](#) of the clusters has been checked with **fiio**.

CHAPTER 10. MIGRATING YOUR APPLICATIONS

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

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.

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.

During migration, 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.

10.1. 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 internal 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.

Internal images

- If your application uses internal images from the **openshift** namespace, you must ensure that the required versions of the images are present on the target cluster. You can manually update an image stream tag in order to use a deprecated OpenShift Container Platform 3 image on an OpenShift Container Platform 4.9 cluster.

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 3 cluster:
 - **8443** (API server)
 - **443** (routes)
 - **53** (DNS)
- 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.

Additional resources for migration prerequisites

- [Manually exposing a secure registry for OpenShift Container Platform 3](#)
- [Updating deprecated internal images](#)

10.2. 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.

10.2.1. 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**.

10.2.2. Adding a cluster to the MTC web console

You can add a cluster to the Migration Toolkit for Containers (MTC) web console.

Prerequisites

- 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

- **Require SSL verification:** Optional: Select this option to verify SSL connections 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.
6. Click **Add cluster**.
The cluster appears in the **Clusters** list.

10.2.3. 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 (GCP)
- 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.
 - **GCP:**
 - **Replication repository name** Specify the replication repository name in the MTC web console.
 - **GCP bucket name:** Specify the name of the GCP bucket.
 - **GCP 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.

10.2.4. 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.
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

- [MTC file system copy method](#)
- [MTC snapshot copy method](#)

10.2.5. 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 11. 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.

11.1. TERMINOLOGY

Table 11.1. 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.

11.2. 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.

11.2.1. 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 internal 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.

Internal images

- If your application uses internal images from the **openshift** namespace, you must ensure that the required versions of the images are present on the target cluster.
You can manually update an image stream tag in order to use a deprecated OpenShift Container Platform 3 image on an OpenShift Container Platform 4.9 cluster.

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 3 cluster:
 - **8443** (API server)

- **443** (routes)
- **53** (DNS)
- 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.

11.2.2. Creating a registry route for direct image migration

For direct image migration, you must create a route to the exposed internal registry on all remote clusters.

Prerequisites

- The internal registry must be exposed to external traffic on all remote clusters. The OpenShift Container Platform 4 registry is exposed by default.

The OpenShift Container Platform 3 registry must be [exposed manually](#).

Procedure

- To create a route to an OpenShift Container Platform 3 registry, run the following command:

```
$ oc create route passthrough --service=docker-registry -n default
```

- 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
```

11.2.3. 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 4.9, the Migration Toolkit for Containers (MTC) inherits the cluster-wide proxy settings. You can change the proxy parameters if you want to override the cluster-wide proxy settings.

11.2.3.1. 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.

11.2.3.1.1. 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.

11.2.3.1.2. 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.

11.2.3.1.3. 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.

11.2.3.2. 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.

11.2.3.2.1. NetworkPolicy configuration

11.2.3.2.1.1. 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
```

11.2.3.2.1.2. 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
```

11.2.3.2.2. 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
```

11.2.3.2.3. 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]
```

11.2.3.2.4. 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 11.2. 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"
```

11.2.3.3. Configuring proxies

Prerequisites

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

Procedure

- Get the **MigrationController** CR manifest:

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

- 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
```

- Stunnel proxy URL for direct volume migration.
- 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
```

11.2.4. 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> ❶
namespace: openshift-migration
spec:
  exposedRegistryPath: <exposed_registry_route> ❷
  insecure: false ❸
  isHostCluster: false
  serviceAccountSecretRef:
    name: <remote_cluster_secret> ❹
    namespace: openshift-config
  url: <remote_cluster_url> ❺
EOF

```

- ❶ Specify the **Cluster** CR of the remote cluster.
- ❷ Optional: For direct image migration, specify the exposed registry route.
- ❸ SSL verification is enabled if **false**. CA certificates are not required or checked if **true**.
- ❹ Specify the **Secret** object of the remote cluster.
- ❺ Specify the URL of the remote cluster.

4. Verify that all clusters are in a **Ready** state:

```
$ oc describe cluster <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> ❶
  aws-secret-access-key: <secret_key_base64> ❷
EOF

```

- ❶ Specify the key ID in base64 format.
- ❷ 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 ❶
```

- ❶ 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 in the **Secrets** CR of the object storage are correct.
- 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

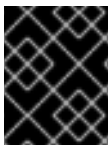
11.2.5. State migration

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.

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.

11.3. 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-rhel7:1.7**. 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.

11.3.1. 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.

11.3.1.1. 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_info 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

```

11.3.1.2. 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') }}"

```

11.4. MIGRATION PLAN OPTIONS

You can exclude, edit, and map components in the **MigPlan** custom resource (CR).

11.4.1. 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

```

11.4.2. 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
```

11.4.3. 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
```

11.4.4. 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.

11.4.5. 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/v1 alpha1
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.

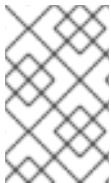
Additional resources

- 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](#).

11.4.6. 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
```

11.5. 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.

11.5.1. 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.

11.5.2. 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}}' \
--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}}' \
--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

```

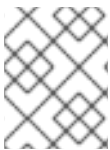


NOTE

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](#))

11.5.3. 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 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 4.9 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.

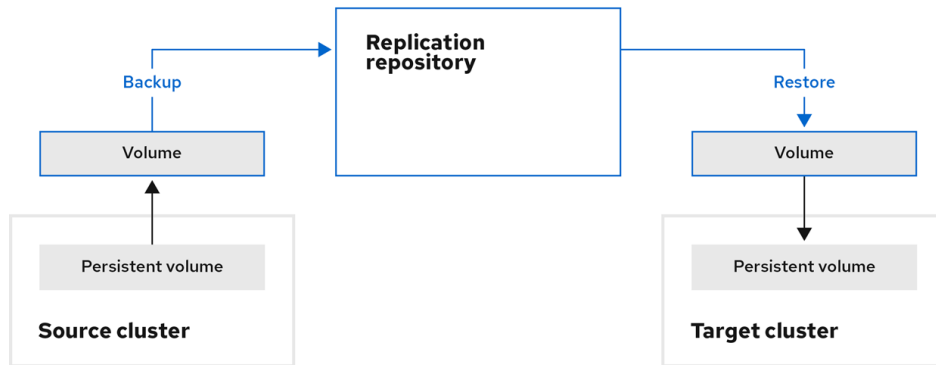
- Custom resources (CRs) and custom resource definitions (CRDs): MTC automatically migrates CRs and CRDs at the namespace level.

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.

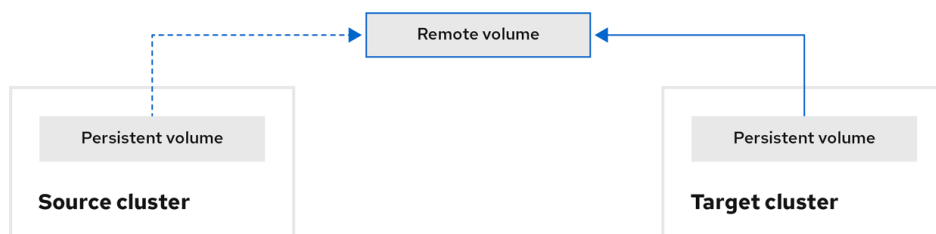


OpenShift_45_1019

- **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**

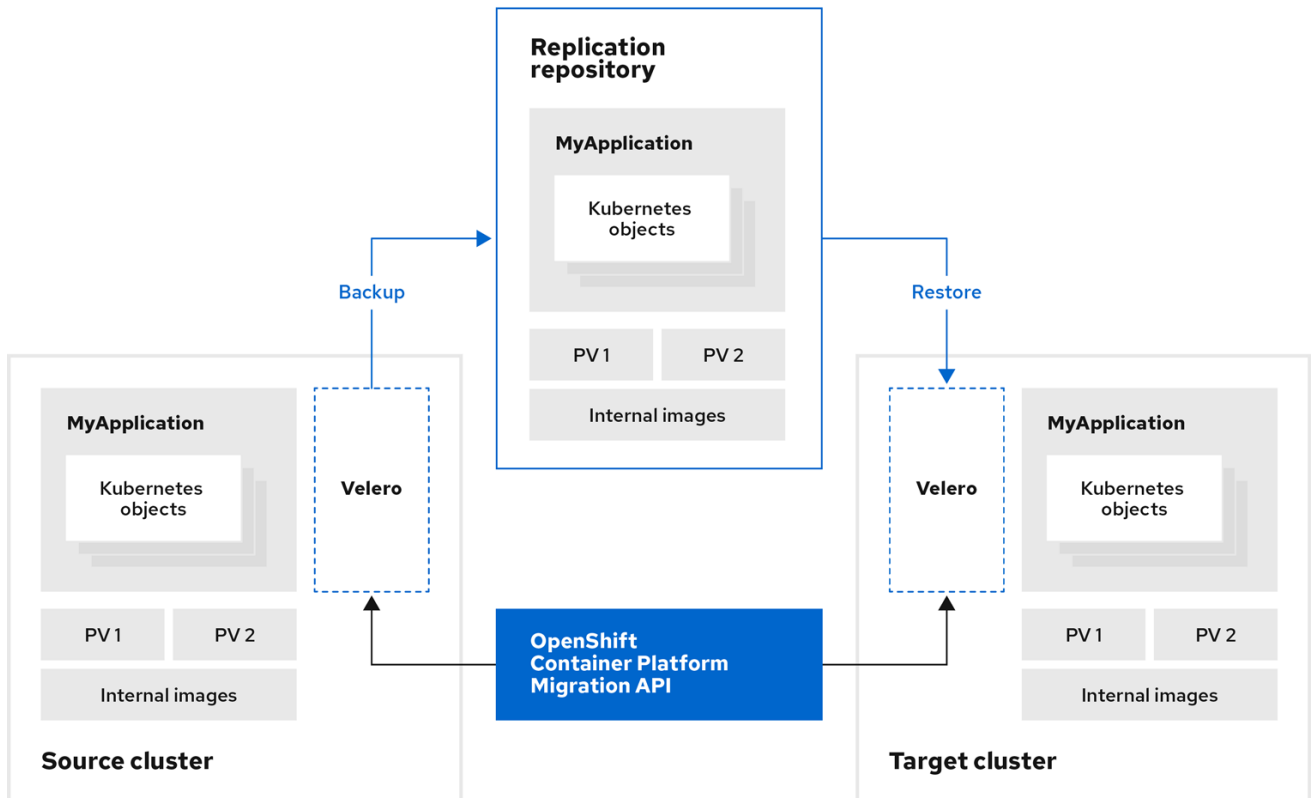
Although the replication repository does not appear in this diagram, it is required for migration.



OpenShift_45_1019

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

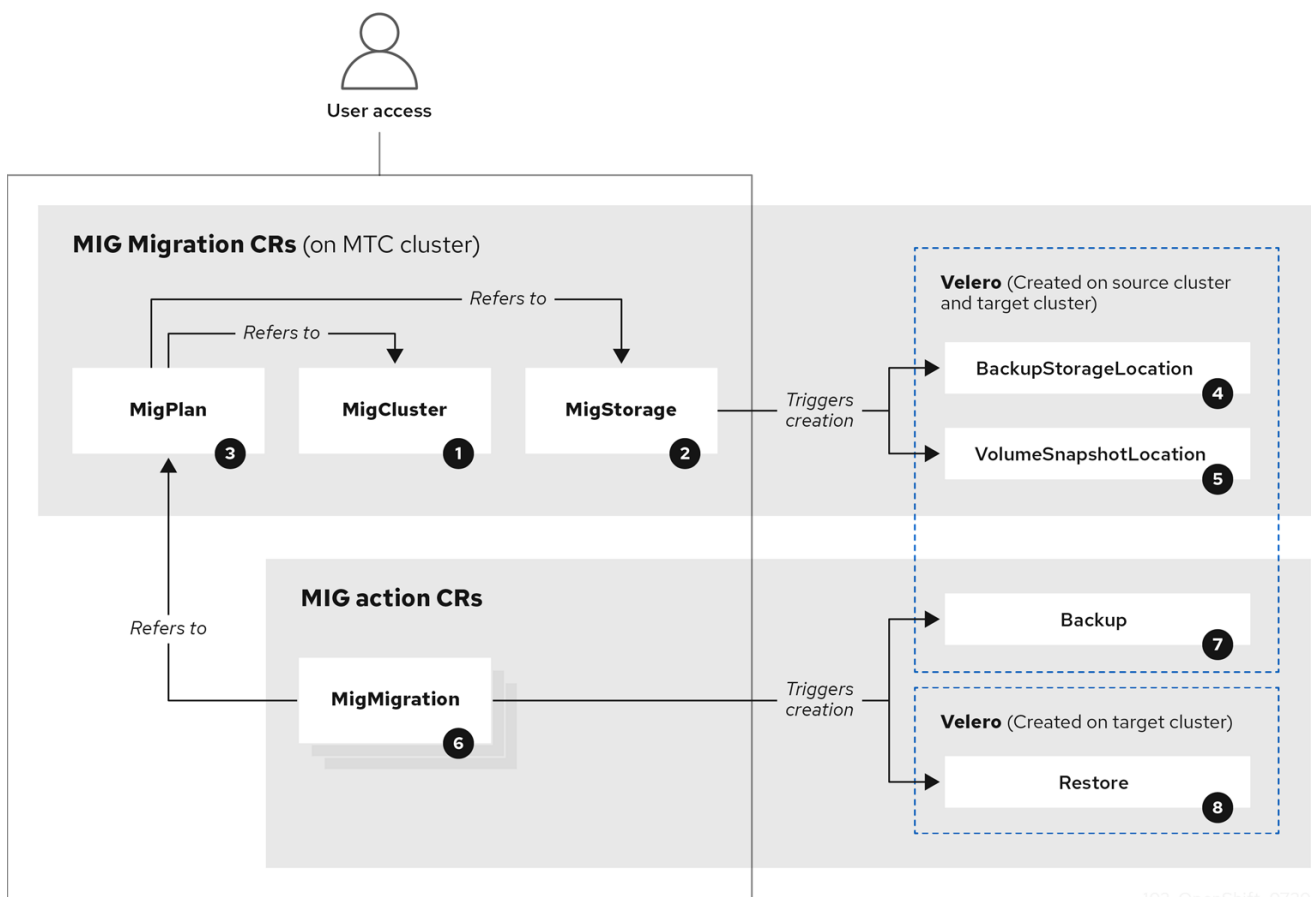
- **Stage** copies data to the target cluster without stopping the application. A stage migration can be run multiple times so that most of the data is copied to the target before migration. Running one or more stage migrations reduces the duration of the cutover migration.
- **Cutover** stops the application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



OpenShift_45_1019

About MTC custom resources

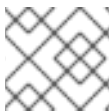
The Migration Toolkit for Containers (MTC) creates the following custom resources (CRs):



102_OpenShift_0720

- 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. MTC 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.

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
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
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 hour:

```
$ oc adm must-gather --image=registry.redhat.io/rhmtc/openshift-migration-must-gather-rhel8:v1.7
```

The data is saved as **must-gather/must-gather.tar.gz**. You can upload this file to a support case on the [Red Hat Customer Portal](#).

- To collect data for the past 24 hours:

```
$ oc adm must-gather --image=registry.redhat.io/rhmtc/openshift-migration-must-gather-rhel8:v1.7 \
-- /usr/bin/gather_metrics_dump
```

This operation can take a long time. The data is saved as **must-gather/metrics/prom_data.tar.gz**.

Viewing metrics data with the Prometheus console

You can view the metrics data with the Prometheus console.

Procedure

1. Decompress the **prom_data.tar.gz** file:

```
$ tar -xvzf must-gather/metrics/prom_data.tar.gz
```

2. Create a local Prometheus instance:

```
$ make prometheus-run
```

The command outputs the Prometheus URL.

Output

```
Started Prometheus on http://localhost:9090
```


3. Launch a web browser and navigate to the URL to view the data by using the Prometheus web console.
4. After you have viewed the data, delete the Prometheus instance and data:

```
$ make prometheus-cleanup
```

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.

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
```

Help option

Use the **velero --help** option to list all Velero CLI commands:

```
$ oc -n openshift-migration exec deployment/velero -c velero -- ./velero \
  --help
```

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
```

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. Updating deprecated internal images

If your application uses images from the **openshift** namespace, the required versions of the images must be present on the target cluster.

If an OpenShift Container Platform 3 image is deprecated in OpenShift Container Platform 4.9, you can manually update the image stream tag by using **podman**.

Prerequisites

- You must have **podman** installed.
- You must be logged in as a user with **cluster-admin** privileges.
- If you are using insecure registries, add your registry host values to the **[registries.insecure]** section of **/etc/container/registries.conf** to ensure that **podman** does not encounter a TLS verification error.
- The internal registries must be exposed on the source and target clusters.

Procedure

1. Ensure that the internal registries are exposed on the OpenShift Container Platform 3 and 4 clusters.

The internal registry is exposed by default on OpenShift Container Platform 4.

2. If you are using insecure registries, add your registry host values to the **[registries.insecure]** section of **/etc/container/registries.conf** to ensure that **podman** does not encounter a TLS verification error.

3. Log in to the OpenShift Container Platform 3 registry:

```
$ podman login -u $(oc whoami) -p $(oc whoami -t) --tls-verify=false <registry_url>:<port>
```

4. Log in to the OpenShift Container Platform 4 registry:

```
$ podman login -u $(oc whoami) -p $(oc whoami -t) --tls-verify=false <registry_url>:<port>
```

5. Pull the OpenShift Container Platform 3 image:

```
$ podman pull <registry_url>:<port>/openshift/<image>
```

6. Tag the OpenShift Container Platform 3 image for the OpenShift Container Platform 4 registry:

```
$ podman tag <registry_url>:<port>/openshift/<image> \ ❶  
  <registry_url>:<port>/openshift/<image> ❷
```

❶ Specify the registry URL and port for the OpenShift Container Platform 3 cluster.

❷ Specify the registry URL and port for the OpenShift Container Platform 4 cluster.

7. Push the image to the OpenShift Container Platform 4 registry:

```
$ podman push <registry_url>:<port>/openshift/<image> ❶
```

❶ Specify the OpenShift Container Platform 4 cluster.

- Verify that the image has a valid image stream:

```
$ oc get imagestream -n openshift | grep <image>
```

Example output

```
NAME      IMAGE REPOSITORY                                TAGS  UPDATED
my_image  image-registry.openshift-image-registry.svc:5000/openshift/my_image  latest 32
seconds ago
```

12.4.2. 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

- 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
```

- On the source cluster, obtain the details of a migrated namespace:

```
$ oc get namespace <namespace> -o yaml 1
```

- 1** Specify the migrated namespace.

- On the target cluster, edit the migrated namespace:

```
$ oc edit namespace <namespace>
```

- 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.

12.4.3. 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.

12.4.3.1. 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.

12.4.3.2. 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.

12.4.3.3. Certificate signed by unknown authority error

If you use a self-signed certificate to secure a cluster or a replication repository for the Migration Toolkit for Containers (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.

12.4.3.4. 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.

12.4.3.5. 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**.

12.4.3.6. 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>
```

12.4.3.7. 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.

12.4.4. Known issues

This release has the following known issues:

- During migration, the Migration Toolkit for Containers (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. ([BZ#1748440](#))
- 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.
- If a migration fails, the migration plan does not retain custom 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](#))
- If a large migration fails because Restic times out, you can increase the **restic_timeout** parameter value (default: **1h**) in the **MigrationController** custom resource (CR) manifest.
- If you select the data verification option for PVs that are migrated with the file system copy method, performance is significantly slower.
- If you are migrating data from NFS storage and **root_squash** is enabled, **Restic** maps to **nfsnobody**. The migration fails and a permission error is displayed in the **Restic** pod log. ([BZ#1873641](#))

You can resolve this issue by adding supplemental groups for **Restic** to the **MigrationController** CR manifest:

```
spec:
  ...
  restic_supplemental_groups:
  - 5555
  - 6666
```

- If you perform direct volume migration with nodes that are in different availability zones, the migration might fail because the migrated pods cannot access the PVC. ([BZ#1947487](#))

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](#)