OpenShift Container Platform 4.8 Migration Toolkit for Containers

Migrating to OpenShift Container Platform 4
Abstract

This document provides instructions for migrating your OpenShift Container Platform cluster from version 3 to version 4, and also for migrating from an earlier OpenShift Container Platform 4 release to the latest version.
# Table of Contents

**CHAPTER 1. MIGRATING FROM OPENSHIFT CONTAINER PLATFORM 3**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1. ABOUT MIGRATING OPENSHIFT CONTAINER PLATFORM 3 TO 4</td>
<td>8</td>
</tr>
<tr>
<td>1.2. PLANNING YOUR MIGRATION</td>
<td>8</td>
</tr>
<tr>
<td>1.2.1. Comparing OpenShift Container Platform 3 and OpenShift Container Platform 4</td>
<td>8</td>
</tr>
<tr>
<td>1.2.1.1. Architecture differences</td>
<td>9</td>
</tr>
<tr>
<td>1.2.1.2. Installation and update differences</td>
<td>9</td>
</tr>
<tr>
<td>1.2.1.2. Installation process</td>
<td>9</td>
</tr>
<tr>
<td>1.2.1.3. Infrastructure options</td>
<td>9</td>
</tr>
<tr>
<td>1.2.2. Migration considerations</td>
<td>10</td>
</tr>
<tr>
<td>1.2.2.1. Storage considerations</td>
<td>10</td>
</tr>
<tr>
<td>1.2.2.2. Networking considerations</td>
<td>11</td>
</tr>
<tr>
<td>1.2.2.3. Logging considerations</td>
<td>11</td>
</tr>
<tr>
<td>1.2.2.4. Security considerations</td>
<td>12</td>
</tr>
<tr>
<td>1.2.2.5. Monitoring considerations</td>
<td>12</td>
</tr>
<tr>
<td>1.3. ABOUT THE MIGRATION TOOLKIT FOR CONTAINERS</td>
<td>12</td>
</tr>
<tr>
<td>1.3.1. Migration Toolkit for Containers workflow</td>
<td>13</td>
</tr>
<tr>
<td>1.3.2. Migration Toolkit for Containers custom resources</td>
<td>15</td>
</tr>
<tr>
<td>1.3.3. About data copy methods</td>
<td>17</td>
</tr>
<tr>
<td>1.3.3.1. File system copy method</td>
<td>17</td>
</tr>
<tr>
<td>1.3.3.2. Snapshot copy method</td>
<td>17</td>
</tr>
<tr>
<td>1.3.3.3. Direct volume migration and direct image migration</td>
<td>18</td>
</tr>
<tr>
<td>1.3.4. About migration hooks</td>
<td>18</td>
</tr>
<tr>
<td>1.4. INSTALLING AND UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS</td>
<td>19</td>
</tr>
<tr>
<td>1.4.1. Installing the Migration Toolkit for Containers in a connected environment</td>
<td>19</td>
</tr>
<tr>
<td>1.4.1.1. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.7 target cluster</td>
<td>19</td>
</tr>
<tr>
<td>1.4.1.2. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 3 source cluster</td>
<td>20</td>
</tr>
<tr>
<td>1.4.2. Installing the Migration Toolkit for Containers in a restricted environment</td>
<td>21</td>
</tr>
<tr>
<td>1.4.2.1. Disabling the default OperatorHub sources</td>
<td>22</td>
</tr>
<tr>
<td>1.4.2.2. Pruning an index image</td>
<td>22</td>
</tr>
<tr>
<td>1.4.2.3. Mirroring an Operator catalog</td>
<td>24</td>
</tr>
<tr>
<td>1.4.2.4. Creating a catalog from an index image</td>
<td>28</td>
</tr>
<tr>
<td>1.4.2.5. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.7 target cluster</td>
<td>28</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1.4.2.6. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 3 source cluster in a restricted environment</td>
<td>30</td>
</tr>
<tr>
<td>1.4.3. Upgrading the Migration Toolkit for Containers</td>
<td></td>
</tr>
<tr>
<td>1.4.3.1. Upgrading the Migration Toolkit for Containers on an OpenShift Container Platform 4 cluster</td>
<td>33</td>
</tr>
<tr>
<td>1.4.3.2. Upgrading the Migration Toolkit for Containers on an OpenShift Container Platform 3 cluster</td>
<td>33</td>
</tr>
<tr>
<td>1.4.3.3. Upgrading MTC 1.3 to 1.4</td>
<td>35</td>
</tr>
<tr>
<td>1.5. CONFIGURING OBJECT STORAGE FOR A REPLICATION REPOSITORY</td>
<td></td>
</tr>
<tr>
<td>1.5.1. Configuring a Multi-Cloud Object Gateway storage bucket as a replication repository</td>
<td>36</td>
</tr>
<tr>
<td>1.5.1.1. Installing the OpenShift Container Storage Operator</td>
<td>36</td>
</tr>
<tr>
<td>1.5.1.2. Creating the Multi-Cloud Object Gateway storage bucket</td>
<td>37</td>
</tr>
<tr>
<td>1.5.2. Configuring an AWS S3 storage bucket as a replication repository</td>
<td>39</td>
</tr>
<tr>
<td>1.5.3. Configuring a Google Cloud Provider storage bucket as a replication repository</td>
<td>42</td>
</tr>
<tr>
<td>1.5.4. Configuring a Microsoft Azure Blob storage container as a replication repository</td>
<td>43</td>
</tr>
<tr>
<td>1.6. MIGRATING YOUR APPLICATIONS</td>
<td></td>
</tr>
<tr>
<td>1.6.1. Prerequisites</td>
<td></td>
</tr>
<tr>
<td>1.6.1.1. Updating deprecated internal images with podman</td>
<td>46</td>
</tr>
<tr>
<td>1.6.1.2. Creating a CA certificate bundle file</td>
<td>47</td>
</tr>
<tr>
<td>1.6.1.3. Configuring a proxy for direct volume migration</td>
<td>47</td>
</tr>
<tr>
<td>1.6.1.4. Writing an Ansible playbook for a migration hook</td>
<td>48</td>
</tr>
<tr>
<td>1.6.1.4.1. Ansible modules</td>
<td>48</td>
</tr>
<tr>
<td>1.6.1.4.2. Environment variables</td>
<td>49</td>
</tr>
<tr>
<td>1.6.1.5. Additional resources</td>
<td>50</td>
</tr>
<tr>
<td>1.6.2. Migrating your applications by using the MTC web console</td>
<td></td>
</tr>
<tr>
<td>1.6.2.1. Launching the MTC web console</td>
<td>50</td>
</tr>
<tr>
<td>1.6.2.2. Adding a cluster to the MTC web console</td>
<td>50</td>
</tr>
<tr>
<td>1.6.2.3. Adding a replication repository to the MTC web console</td>
<td>52</td>
</tr>
<tr>
<td>1.6.2.4. Creating a migration plan in the MTC web console</td>
<td>53</td>
</tr>
<tr>
<td>1.6.2.5. Running a migration plan in the MTC web console</td>
<td>55</td>
</tr>
<tr>
<td>1.6.3. Migrating your applications from the command line</td>
<td></td>
</tr>
<tr>
<td>MTC terminology</td>
<td>56</td>
</tr>
<tr>
<td>1.6.3.1. Migrating your applications with the Migration Toolkit for Containers API</td>
<td>57</td>
</tr>
<tr>
<td>1.6.3.2. MTC custom resource manifests</td>
<td></td>
</tr>
<tr>
<td>1.6.3.2.1. DirectImageMigration</td>
<td>63</td>
</tr>
<tr>
<td>1.6.3.2.2. DirectImageStreamMigration</td>
<td>64</td>
</tr>
<tr>
<td>1.6.3.2.3. DirectVolumeMigration</td>
<td>64</td>
</tr>
<tr>
<td>1.6.3.2.4. DirectVolumeMigrationProgress</td>
<td>65</td>
</tr>
<tr>
<td>1.6.3.2.5. MigAnalytic</td>
<td>65</td>
</tr>
<tr>
<td>1.6.3.2.6. MigCluster</td>
<td>66</td>
</tr>
<tr>
<td>1.6.3.2.7. MigHook</td>
<td>67</td>
</tr>
<tr>
<td>1.6.3.2.8. MigMigration</td>
<td>68</td>
</tr>
<tr>
<td>1.6.3.2.9. MigPlan</td>
<td>68</td>
</tr>
<tr>
<td>1.6.3.2.10. MigStorage</td>
<td>70</td>
</tr>
<tr>
<td>1.6.4. Additional resources</td>
<td></td>
</tr>
<tr>
<td>1.6.5. Configuring a migration plan</td>
<td>71</td>
</tr>
<tr>
<td>1.6.5.1. Increasing limits for large migrations</td>
<td>71</td>
</tr>
<tr>
<td>1.6.5.2. Excluding resources from a migration plan</td>
<td>72</td>
</tr>
<tr>
<td>1.7. TROUBLESHOOTING</td>
<td></td>
</tr>
<tr>
<td>1.7.1. Viewing MTC custom resources</td>
<td>74</td>
</tr>
<tr>
<td>1.7.2. Using the migration log reader</td>
<td>77</td>
</tr>
<tr>
<td>1.7.3. Downloading migration logs</td>
<td>77</td>
</tr>
<tr>
<td>1.7.4. Updating deprecated APIs</td>
<td>78</td>
</tr>
<tr>
<td>1.7.5. Error messages and resolutions</td>
<td>80</td>
</tr>
</tbody>
</table>
1.7.5.1. CA certificate error in the MTC console
1.7.5.2. OAuth timeout error in the MTC console
1.7.5.3. Backup storage location errors in the Velero pod log
1.7.5.4. Pod volume backup timeout error in the Velero pod log
1.7.5.5. Restic verification errors in the MigMigration custom resource
1.7.5.6. Restic permission error when migrating from NFS storage with root_squash enabled
1.7.6. Direct volume migration does not complete
1.7.7. Using the Velero CLI to debug Backup and Restore CRs
1.7.7.1. Velero command syntax
1.7.7.2. Help command
1.7.7.3. Describe command
1.7.7.4. Logs command
1.7.7.5. Debugging a partial migration failure
1.7.8. Using must-gather to collect data
1.7.9. Rolling back a migration
1.7.9.1. Rolling back a migration in the MTC web console
1.7.9.1.1. Rolling back a migration from the CLI
1.7.10. Known issues
1.7.11. Additional resources

CHAPTER 2. MIGRATING FROM OPENSHIFT CONTAINER PLATFORM 4.1

2.1. ABOUT THE MIGRATION TOOLKIT FOR CONTAINERS
2.1.1. Migration Toolkit for Containers workflow
2.1.2. Migration Toolkit for Containers custom resources
2.1.3. About data copy methods
2.1.3.1. File system copy method
2.1.3.2. Snapshot copy method
2.1.3.3. Direct volume migration and direct image migration
2.1.4. About migration hooks
2.2. INSTALLING AND UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS
2.2.1. Installing the Migration Toolkit for Containers in a connected environment
2.2.1.1. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.7 target cluster
2.2.1.2. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.1 source cluster
2.2.2. Installing the Migration Toolkit for Containers in a restricted environment
2.2.2.1. Disabling the default OperatorHub sources
2.2.2.2. Pruning an index image
2.2.2.3. Mirroring an Operator catalog
2.2.2.4. Creating a catalog from an index image
2.2.2.5. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.7 target cluster in a restricted environment
2.2.2.6. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.1 source cluster in a restricted environment
2.2.3. Upgrading the Migration Toolkit for Containers
2.2.3.1. Upgrading the Migration Toolkit for Containers on an OpenShift Container Platform 4 cluster
2.2.3.2. Upgrading MTC 1.3 to 1.4
2.3. CONFIGURING OBJECT STORAGE FOR A REPLICATION REPOSITORY
2.3.1. Configuring a Multi-Cloud Object Gateway storage bucket as a replication repository
2.3.1.1. Installing the OpenShift Container Storage Operator
2.3.1.2. Creating the Multi-Cloud Object Gateway storage bucket
2.3.2. Configuring an AWS S3 storage bucket as a replication repository
2.3.3. Configuring a Google Cloud Provider storage bucket as a replication repository
2.3.4. Configuring a Microsoft Azure Blob storage container as a replication repository

2.4. MIGRATING YOUR APPLICATIONS

2.4.1. Prerequisites

2.4.1.1. Creating a CA certificate bundle file

2.4.1.2. Configuring a proxy for direct volume migration

2.4.1.3. Writing an Ansible playbook for a migration hook
  2.4.1.3.1. Ansible modules
  2.4.1.3.2. Environment variables

2.4.1.4. Additional resources

2.4.2. Migrating your applications using the MTC web console

2.4.2.1. Launching the MTC web console

2.4.2.2. Adding a cluster to the MTC web console

2.4.2.3. Adding a replication repository to the MTC web console

2.4.2.4. Creating a migration plan in the MTC web console

2.4.2.5. Running a migration plan in the MTC web console

2.4.3. Migrating your applications from the command line

    MTC terminology

    2.4.3.1. Migrating your applications with the Migration Toolkit for Containers API

    2.4.3.2. MTC custom resource manifests
      2.4.3.2.1. DirectImageMigration
      2.4.3.2.2. DirectImageStreamMigration
      2.4.3.2.3. DirectVolumeMigration
      2.4.3.2.4. DirectVolumeMigrationProgress
      2.4.3.2.5. MigAnalytic
      2.4.3.2.6. MigCluster
      2.4.3.2.7. MigHook
      2.4.3.2.8. MigMigration
      2.4.3.2.9. MigPlan
      2.4.3.2.10. MigStorage

    2.4.4. Additional resources

2.4.5. Configuring a migration plan

    2.4.5.1. Increasing limits for large migrations

    2.4.5.2. Excluding resources from a migration plan

2.5. TROUBLESHOOTING

    2.5.1. Viewing MTC custom resources

    2.5.2. Using the migration log reader

    2.5.3. Downloading migration logs

    2.5.4. Updating deprecated APIs

    2.5.5. Error messages and resolutions
      2.5.5.1. CA certificate error in the MTC console
      2.5.5.2. OAuth timeout error in the MTC console
      2.5.5.3. Backup storage location errors in the Velero pod log
      2.5.5.4. Pod volume backup timeout error in the Velero pod log
      2.5.5.5. Restic verification errors in the MigMigration custom resource
      2.5.5.6. Restic permission error when migrating from NFS storage with root_squash enabled

    2.5.6. Direct volume migration does not complete

    2.5.7. Using the Velero CLI to debug Backup and Restore CRs
      2.5.7.1. Velero command syntax
      2.5.7.2. Help command
      2.5.7.3. Describe command
      2.5.7.4. Logs command
      2.5.7.5. Debugging a partial migration failure

    2.5.8. Using must-gather to collect data
3.4.2.5. Running a migration plan in the MTC web console

3.4.3. Migrating your applications from the command line

MTC terminology

3.4.3.1. Migrating your applications with the Migration Toolkit for Containers API

3.4.3.2. MTC custom resource manifests

3.4.3.2.1. DirectImageMigration

3.4.3.2.2. DirectImageStreamMigration

3.4.3.2.3. DirectVolumeMigration

3.4.3.2.4. DirectVolumeMigrationProgress

3.4.3.2.5. MigAnalytic

3.4.3.2.6. MigCluster

3.4.3.2.7. MigHook

3.4.3.2.8. MigMigration

3.4.3.2.9. MigPlan

3.4.3.2.10. MigStorage

3.4.3.3. Additional resources

3.4.4. Configuring a migration plan

3.4.4.1. Increasing limits for large migrations

3.4.4.2. Excluding resources from a migration plan

3.5. TROUBLESHOOTING

3.5.1. Viewing MTC custom resources

3.5.2. Using the migration log reader

3.5.3. Downloading migration logs

3.5.4. Updating deprecated APIs

3.5.5. Error messages and resolutions

3.5.5.1. CA certificate error in the MTC console

3.5.5.2. OAuth timeout error in the MTC console

3.5.5.3. Backup storage location errors in the Velero pod log

3.5.5.4. Pod volume backup timeout error in the Velero pod log

3.5.5.5. Restic verification errors in the MigMigration custom resource

3.5.5.6. Restic permission error when migrating from NFS storage with root_squash enabled

3.5.6. Direct volume migration does not complete

3.5.7. Using the Velero CLI to debug Backup and Restore CRs

3.5.7.1. Velero command syntax

3.5.7.2. Help command

3.5.7.3. Describe command

3.5.7.4. Logs command

3.5.7.5. Debugging a partial migration failure

3.5.8. Using must-gather to collect data

3.5.9. Rolling back a migration

3.5.9.1. Rolling back a migration in the MTC web console

3.5.9.1.1. Rolling back a migration from the CLI

3.5.10. Known issues

3.5.11. Additional resources
CHAPTER 1. MIGRATING FROM OPENSHIFT CONTAINER PLATFORM 3

1.1. ABOUT MIGRATING OPENSHIFT CONTAINER PLATFORM 3 TO 4

OpenShift Container Platform 4 includes new technologies and functionality that results in a cluster that is self-managing, flexible, and automated. The way that OpenShift Container Platform 4 clusters are deployed and managed drastically differs from OpenShift Container Platform 3.

To successfully transition from OpenShift Container Platform 3 to OpenShift Container Platform 4, it is important that you review the following information:

Planning your transition
Learn about the differences between OpenShift Container Platform versions 3 and 4. Prior to transitioning, be sure that you have reviewed and prepared for storage, networking, logging, security, and monitoring considerations.

Performing your migration
Learn about and use Migration Toolkit for Containers (MTC) to migrate your application workloads.

1.2. PLANNING YOUR MIGRATION

Before performing your migration to OpenShift Container Platform 4.7, it is important to take the time to properly plan for the transition. OpenShift Container Platform 4 introduces architectural changes and enhancements, so the procedures that you used to manage your OpenShift Container Platform 3 cluster might not apply for OpenShift Container Platform 4.

NOTE
This planning document assumes that you are transitioning from OpenShift Container Platform 3.11 to OpenShift Container Platform 4.7.

This document provides high-level information on the most important differences between OpenShift Container Platform 3 and OpenShift Container Platform 4 and the most noteworthy migration considerations. For detailed information on configuring your OpenShift Container Platform 4 cluster, review the appropriate sections of the OpenShift Container Platform documentation. For detailed information on new features and other notable technical changes, review the OpenShift Container Platform 4.7 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.

1.2.1. Comparing OpenShift Container Platform 3 and OpenShift Container Platform 4

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.

1.2.1.1. Architecture differences

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.

1.2.1.2. Installation and update differences

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 4.7, you use the OpenShift installation program to create a minimum set of resources required for a cluster. Once 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 4.7 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 4.7, 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 4.7 cluster has RHEL worker machines, then you will still need to run an Ansible playbook to upgrade those worker machines.

For more information, see Upgrading clusters.

1.2.2. Migration considerations

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

1.2.2.1. Storage considerations

Review the following storage changes to consider when transitioning from OpenShift Container Platform 3.11 to OpenShift Container Platform 4.7.

Local volume persistent storage
Local storage is only supported by using the Local Storage Operator in OpenShift Container Platform 4.7. 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 plug-in location changed from OpenShift Container Platform 3.11. The new location in OpenShift Container Platform 4.7 is `/etc/kubernetes/kubelet-plugins/volume/exec`. Attachable FlexVolume plug-ins 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 4.7 fully supports CSI version 1.1.0 and 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 4.7:
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 4.7.

For more information, see Understanding persistent storage.

1.2.2.2. Networking considerations

Review the following networking changes to consider when transitioning from OpenShift Container Platform 3.11 to OpenShift Container Platform 4.7.

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 4.7 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 4.7 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 4.7, follow the steps to configure multitenant isolation using network policy.

For more information, see About network policy.

Encrypting traffic between hosts
In OpenShift Container Platform 3.11, you could use IPsec to encrypt traffic between hosts. OpenShift Container Platform 4.7 does not support IPsec. It is recommended to use Red Hat OpenShift Service Mesh to enable mutual TLS between services.

For more information, see Understanding Red Hat OpenShift Service Mesh.

1.2.2.3. Logging considerations

Review the following logging changes to consider when transitioning from OpenShift Container Platform 3.11 to OpenShift Container Platform 4.7.

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 4.7.
For more information on the explicitly unsupported logging cases, see Maintenance and support.

1.2.2.4. Security considerations

Review the following security changes to consider when transitioning from OpenShift Container Platform 3.11 to OpenShift Container Platform 4.7.

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 4.7. 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 4.7 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 4.7. It now obtains data, which previously had to 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.

1.2.2.5. Monitoring considerations

Review the following monitoring changes to consider when transitioning from OpenShift Container Platform 3.11 to OpenShift Container Platform 4.7.

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.

1.3. ABOUT THE MIGRATION TOOLKIT FOR CONTAINERS

You can migrate application workloads from OpenShift Container Platform 3.7, 3.9, 3.10, and 3.11 to OpenShift Container Platform 4.7 with the Migration Toolkit for Containers (MTC). MTC enables you to control the migration and to minimize application downtime.
MTC is installed on the target cluster by default.

You can configure the Migration Toolkit for Containers Operator to install the MTC on an OpenShift Container Platform 3 cluster or on a remote cluster.

The MTC web console and API, based on Kubernetes custom resources, enable you to migrate stateful application workloads at the granularity of a namespace.

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.

You can use migration hooks to run Ansible playbooks at certain points during the migration. The hooks are added when you create a migration plan.

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 web console displays a message if the service catalog resources cannot be migrated.

Before you begin your migration, be sure to review the information on planning your migration.

1.3.1. Migration Toolkit for Containers workflow

You use the Migration Toolkit for Containers (MTC) to migrate Kubernetes resources, persistent volume data, and internal container images from an OpenShift Container Platform source cluster to an OpenShift Container Platform 4.7 target cluster by using the MTC web console or the Kubernetes API.

The (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 volume claims that are linked to the persistent volumes of the namespace.

- Custom resources (CRs) and custom resource definitions (CRDs): The MTC automatically migrates any CRs that exist at the namespace level as well as the CRDs that are linked to those CRs.
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. In a restricted environment, you can use an internally hosted S3 storage repository. 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.
     
     ![Diagram](https://via.placeholder.com/150)
     
     **NOTE**
     
     If you are using direct image migration or direct volume migration, the images or volumes are copied directly from the source cluster to the target cluster.

   - **Move**: MTC unmounts a remote volume, for example, NFS, from the source cluster, creates a PV resource on the target cluster pointing to the remote volume, and then mounts the remote volume on the target cluster. Applications running on the target cluster use the same remote volume that the source cluster was using. The remote volume must be accessible to the source and target clusters.
      
     **NOTE**
     
     Although the replication repository does not appear in this diagram, it is required for migration.
6. Run the migration plan, with one of the following options:

- **Stage** (optional) copies data to the target cluster without stopping the application. Staging can be run multiple times so that most of the data is copied to the target before migration. This minimizes the duration of the migration and application downtime.

- **Migrate** stops the application on the source cluster and recreates its resources on the target cluster. Optionally, you can migrate the workload without stopping the application.

1.3.2. Migration Toolkit for Containers custom resources

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

1. **MigCluster** (configuration, MTC cluster): Cluster definition
2. **MigStorage** (configuration, MTC cluster): Storage definition
3. **MigPlan** (configuration, MTC cluster): Migration plan

**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

1.3.3. 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.

1.3.3.1. File system copy method

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

Table 1.1. File system copy method summary

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

1.3.3.2. 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.

AWS, Google Cloud Provider, and Microsoft Azure support the snapshot copy method.

Table 1.2. Snapshot copy method summary

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Benefits</td>
<td>Limitations</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>- Faster than the file system copy method.</td>
<td>- Cloud provider must support snapshots.</td>
</tr>
<tr>
<td></td>
<td>- Clusters must be on the same cloud provider.</td>
</tr>
<tr>
<td></td>
<td>- Clusters must be in the same location or region.</td>
</tr>
<tr>
<td></td>
<td>- Clusters must have the same storage class.</td>
</tr>
<tr>
<td></td>
<td>- Storage class must be compatible with snapshots.</td>
</tr>
<tr>
<td></td>
<td>- Does not support direct volume migration.</td>
</tr>
</tbody>
</table>

**1.3.3.3. Direct volume migration and direct image migration**

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

Direct migration has significant performance benefits because it skips 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.

Direct migration uses Rsync to transfer the files.

**NOTE**

Direct image migration and direct volume migration have additional prerequisites.

**1.3.4. About migration hooks**

You can use migration hooks to run custom code at certain points during a migration with the Migration Toolkit for Containers (MTC). 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 using an Ansible playbook or 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 (CR). 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-rhel7:1.4. This image is based on the Ansible Runner image and includes python-openshift for Ansible Kubernetes resources and an updated oc binary.

Optional: You can use a custom Ansible runtime image containing additional Ansible modules or tools instead of the default image.

Custom hook container
You can create a custom hook container that includes Ansible playbooks or custom code.

1.4. INSTALLING AND UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS

You can install the Migration Toolkit for Containers (MTC) on an OpenShift Container Platform 4.7 target cluster and an OpenShift Container Platform 3 source cluster.

The Migration Controller pod runs on the target cluster by default. You can configure the Migration Controller pod to run on the source cluster or on a remote cluster.

1.4.1. Installing the Migration Toolkit for Containers in a connected environment

You can install the Migration Toolkit for Containers (MTC) in a connected environment.

IMPORTANT

You must install the same MTC version on all clusters.

1.4.1.1. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.7 target cluster

You can install the Migration Toolkit for Containers (MTC) on an OpenShift Container Platform 4.7 target cluster.

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.
NOTE

Do not change the subscription approval option to Automatic. The Migration Toolkit for Containers version must be the same on the source and the target clusters.

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.

1.4.1.2. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 3 source cluster

You can install the Migration Toolkit for Containers (MTC) manually on an OpenShift Container Platform 3 source cluster.

IMPORTANT

You must install the same MTC version on the OpenShift Container Platform 3 and 4 clusters.

To ensure that you have the latest version on the OpenShift Container Platform 3 cluster, download the operator.yml and controller-3.yml files when you are ready to create and run the migration plan.

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.
- The source cluster must be OpenShift Container Platform 3.7, 3.9, 3.10, or 3.11.
- The source cluster must be configured to pull images from registry.redhat.io.
  To pull images, you must create an image stream secret and copy it to each node in your cluster.

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:
3. Download the **controller-3.yml** file:

   ```
   $ sudo podman cp $(sudo podman create \n   registry.redhat.io/rhmtc/openshift-migration-rhel7-operator:v1.4):/controller-3.yml ./
   ```

4. Log in to your OpenShift Container Platform 3 cluster.

5. Verify that the cluster can authenticate with **registry.redhat.io**:

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

6. Create the Migration Toolkit for Containers Operator object:

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

   **Example output**

   ```
   namespace/openshift-migration created
   rolebinding.rbac.authorization.k8s.io/system:deployers created
   serviceaccount/migration-operator created
   customresourcedefinition.apiextensions.k8s.io/migrationcontrollers.migration.openshift.io created
   role.rbac.authorization.k8s.io/migration-operator created
   rolebinding.rbac.authorization.k8s.io/migration-operator created
   clusterrolebinding.rbac.authorization.k8s.io/migration-operator created
   deployment.apps/migration-operator created
   ```

   Error from server (AlreadyExists): error when creating "/operator.yml": rolebindings.rbac.authorization.k8s.io "system:image-builders" already exists

   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 3 that are provided in later releases.

7. Create the **MigrationController** object:

   ```
   $ oc create -f controller-3.yml
   ```

8. Verify that the **Velero** and **Restic** pods are running:

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

### 1.4.2. Installing the Migration Toolkit for Containers in a restricted environment

You can install the Migration Toolkit for Containers (MTC) in a restricted environment.
IMPORTANT

You must install the same MTC version on all clusters.

You can build a custom Operator catalog image for OpenShift Container Platform 4, push it to a local mirror image registry, and configure Operator Lifecycle Manager (OLM) to install the Migration Toolkit for Containers Operator from the local registry.

1.4.2.1. Disabling the default OperatorHub sources

Operator catalogs that source content provided by Red Hat and community projects are configured for OperatorHub by default during an OpenShift Container Platform installation.

Procedure

- Disable the sources for the default catalogs by adding `disableAllDefaultSources: true` to the `OperatorHub` object:

  ```
  $ oc patch OperatorHub cluster --type json \\
  -p '[{"op": "add", "path": "/spec/disableAllDefaultSources", "value": true}]'
  ```

TIP

Alternatively, you can use the web console to manage catalog sources. From the Administration → Cluster Settings → Global Configuration → OperatorHub page, click the Sources tab, where you can create, delete, disable, and enable individual sources.

1.4.2.2. Pruning an index image

An index image, based on the Operator Bundle Format, is a containerized snapshot of an Operator catalog. You can prune an index of all but a specified list of packages, which creates a copy of the source index containing only the Operators that you want.

When configuring Operator Lifecycle Manager (OLM) to use mirrored content on restricted network OpenShift Container Platform clusters, use this pruning method if you want to only mirror a subset of Operators from the default catalogs.

For the steps in this procedure, the target registry is an existing mirror registry that is accessible by your workstation with unrestricted network access. This example also shows pruning the index image for the default `redhat-operators` catalog, but the process is the same for any index image.

Prerequisites

- Workstation with unrestricted network access
- `podman` version 1.9.3+
- `grpcurl`
- `opm` version 1.12.3+
- Access to a registry that supports Docker v2-2

Procedure
1. Authenticate with `registry.redhat.io`:

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

2. Authenticate with your target registry:

   ```
   $ podman login <target_registry>
   ```

3. Determine the list of packages you want to include in your pruned index.
   a. Run the source index image that you want to prune in a container. For example:

   ```
   $ podman run -p50051:50051 \
   -it registry.redhat.io/redhat/redhat-operator-index:v4.7
   ```

   **Example output**

   Trying to pull registry.redhat.io/redhat/redhat-operator-index:v4.7...
   Getting image source signatures
   Copying blob ae8a0c23f5b1 done
   ...
   INFO[0000] serving registry
database=/database/index.db port=50051

   b. In a separate terminal session, use the `grpcurl` command to get a list of the packages provided by the index:

   ```
   $ grpcurl -plaintext localhost:50051 api.Registry/ListPackages > packages.out
   ```

c. Inspect the `packages.out` file and identify which package names from this list you want to keep in your pruned index. For example:

   **Example snippets of packages list**

   ```
   ...
   {
     "name": "advanced-cluster-management"
   }
   ...
   {
     "name": "jaeger-product"
   }
   ...
   {
     "name": "quay-operator"
   }
   ...
   ```

d. In the terminal session where you executed the `podman run` command, press **Ctrl** and **C** to stop the container process.

4. Run the following command to prune the source index of all but the specified packages:
Index to prune.

Comma-separated list of packages to keep.

Required only for IBM Power Systems and IBM Z images: Operator Registry base image with the tag that matches the target OpenShift Container Platform cluster major and minor version.

Custom tag for new index image being built.

5. Run the following command to push the new index image to your target registry:

```bash
$ podman push <target_registry>:<port>/<namespace>/redhat-operator-index:v4.7
```

where `<namespace>` is any existing namespace on the registry. For example, you might create an `olm-mirror` namespace to push all mirrored content to.

1.4.2.3. Mirroring an Operator catalog

You can mirror the Operator content of a Red Hat-provided catalog, or a custom catalog, into a container image registry using the `oc adm catalog mirror` command. The target registry must support Docker v2-2. For a cluster on a restricted network, this registry can be one that the cluster has network access to, such as a mirror registry created during a restricted network cluster installation.

The `oc adm catalog mirror` command also automatically mirrors the index image that is specified during the mirroring process, whether it be a Red Hat-provided index image or your own custom-built index image, to the target registry. You can then use the mirrored index image to create a catalog source that allows Operator Lifecycle Manager (OLM) to load the mirrored catalog onto your OpenShift Container Platform cluster.

Prerequisites

- Workstation with unrestricted network access.
- `podman` version 1.9.3 or later.
- Access to mirror registry that supports Docker v2-2.
- Decide which namespace on your mirror registry you will use to store the mirrored Operator content. For example, you might create an `olm-mirror` namespace.
- If your mirror registry does not have Internet access, connect removable media to your workstation with unrestricted network access.
- If you are working with private registries, set the `REG_CREDS` environment variable to the file path of your registry credentials for use in later steps. For example, for the `podman` CLI:

```bash
$ REG_CREDS=${XDG_RUNTIME_DIR}/containers/auth.json
```
Procedure

1. If you want to mirror a Red Hat-provided catalog, run the following command on your workstation with unrestricted network access to authenticate with `registry.redhat.io`:

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

2. The `oc adm catalog mirror` command extracts the contents of an index image to generate the manifests required for mirroring. The default behavior of the command generates manifests, then automatically mirrors all of the image content from the index image, as well as the index image itself, to your mirror registry. Alternatively, if your mirror registry is on a completely disconnected, or `airgapped`, host, you can first mirror the content to removable media, move the media to the disconnected environment, then mirror the content from the media to the registry.

   - **Option A:** If your mirror registry is on the same network as your workstation with unrestricted network access, take the following actions on your workstation:

     a. If your mirror registry requires authentication, run the following command to log in to the registry:

        ```
        $ podman login <mirror_registry>
        ```

     b. Run the following command to mirror the content:

        ```
        $ oc adm catalog mirror \
        <index_image> \
        <mirror_registry>:<port>/<namespace> \
        [-a ${REG_CREDS}] \
        [--insecure] \
        [--index-filter-by-os='<platform>/<arch>'] \
        [--manifests-only]
        ```

        1. Specify the index image for the catalog you want to mirror. For example, this might be a pruned index image that you created previously, or one of the source index images for the default catalogs, such as `registry.redhat.io/redhat/redhat-operator-index:v4.7`.

        2. Specify the target registry and namespace to mirror the Operator content to, where `<namespace>` is any existing namespace on the registry. For example, you might create an `olm-mirror` namespace to push all mirrored content to.

        3. Optional: If required, specify the location of your registry credentials file.

        4. Optional: If you do not want to configure trust for the target registry, add the `--insecure` flag.

        5. Optional: Specify which platform and architecture of the index image to select when multiple variants are available. Images are passed as `'<platform>/<arch> [/<variant>]'`. This does not apply to images referenced by the index. Valid values are `linux/amd64`, `linux/ppc64le`, and `linux/s390x`.

        6. Optional: Generate only the manifests required for mirroring, and do not actually mirror the image content to a registry. This option can be useful for reviewing what will be mirrored, and it allows you to make any changes to the mapping list if you require only a subset of packages. You can then use the `mapping.txt` file with the
**oc image mirror** command to mirror the modified list of images in a later step. This flag is intended for only advanced selective mirroring of content from the catalog; the **opm index prune** command, if you used it previously to prune the index image, is suitable for most catalog management use cases.

**Example output**

src image has index label for database path: /database/index.db
using database path mapping: /database/index.db:/tmp/153048078
wrote database to /tmp/153048078

wrote mirroring manifests to manifests-redhat-operator-index-1614211642

1 Directory for the temporary index.db database generated by the command.

2 Record the manifests directory name that is generated. This directory name is used in a later step.

- **Option B: If your mirror registry is on a disconnected host** take the following actions.

  a. Run the following command on your workstation with unrestricted network access to mirror the content to local files:

```
$ oc adm catalog mirror
  <index_image> file:///local/index
  [-a ${REG_CREDS}] [--insecure]
```

1 Specify the index image for the catalog you want to mirror. For example, this might be a pruned index image that you created previously, or one of the source index images for the default catalogs, such as `registry.redhat.io/redhat/redhat-operator-index:v4.7`.

2 Mirrors content to local files in your current directory.

**Example output**

... info: Mirroring completed in 5.93s (5.915MB/s) wrote mirroring manifests to manifests-my-index-1614985528

To upload local images to a registry, run:

```
oc adm catalog mirror file:///local/index/myrepo/my-index:v1
REGISTRY/REPOSITORY```

1 Record the manifests directory name that is generated. This directory name is used in a later step.

2 Record the expanded file:// path that based on your provided index image. This path is used in a later step.
b. Copy the `v2/` directory that is generated in your current directory to removable media.

c. Physically remove the media and attach it to a host in the disconnected environment that has access to the mirror registry.

d. If your mirror registry requires authentication, run the following command on your host in the disconnected environment to log in to the registry:

   ```bash
   $ podman login <mirror_registry>
   ```

   **NOTE**
   
   The manifests directory name is used in a later step.

3. After mirroring the content to your registry, inspect the manifests directory that is generated in your current directory.

   ```bash
   $ oc adm catalog mirror \
   file://local/index/<repo>/<index_image>:<tag> \1
   <mirror_registry>:<port>/<namespace> \2
   [-a ${REG_CREDS}] \
   [-insecure]
   ```

   **1** Specify the `file://` path from the previous command output.

   **2** Specify the target registry and namespace to mirror the Operator content to, where `<namespace>` is any existing namespace on the registry. For example, you might create an `olm-mirror` namespace to push all mirrored content to.

   ```bash
   manifests-<index_image_name>-<random_number>
   manifests-index/<namespace>/<index_image_name>-<random_number>
   ```

   If you mirrored content to a registry on the same network in the previous step, the directory name takes the following form:

   ```bash
   manifests-<index_image_name>-<random_number>
   ```

   If you mirrored content to a registry on a disconnected host in the previous step, the directory name takes the following form:

   ```bash
   manifests-index/<namespace>/<index_image_name>-<random_number>
   ```

   The manifests directory contains the following files, some of which might require further modification:

   - The `catalogSource.yaml` file is a basic definition for a `CatalogSource` object that is pre-populated with your index image tag and other relevant metadata. This file can be used as is or modified to add the catalog source to your cluster.
IMPORTANT

If you mirrored the content to local files, you must modify your catalogSource.yaml file to remove any backslash (/) characters from the metadata.name field. Otherwise, when you attempt to create the object, it fails with an "invalid resource name" error.

- The imageContentSourcePolicy.yaml file defines an ImageContentSourcePolicy object that can configure nodes to translate between the image references stored in Operator manifests and the mirrored registry.

NOTE

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

- The mapping.txt file contains all of the source images and where to map them in the target registry. This file is compatible with the oc image mirror command and can be used to further customize the mirroring configuration.

IMPORTANT

If you used the --manifests-only flag during the mirroring process and want to further trim the subset of packages to be mirrored, see the steps in the “Mirroring a Package Manifest Format catalog image” procedure about modifying your mapping.txt file and using the file with the oc image mirror command. After following those further actions, you can continue this procedure.

4. On a host with access to the disconnected cluster, create the ImageContentSourcePolicy object by running the following command to specify the imageContentSourcePolicy.yaml file in your manifests directory:

```bash
$ oc create -f <path/to/manifests/dir>/imageContentSourcePolicy.yaml
```

where <path/to/manifests/dir> is the path to the manifests directory for your mirrored content.

You can now create a CatalogSource object to reference your mirrored index image and Operator content.

1.4.2.4. Creating a catalog from an index image

You can create an Operator catalog from an index image and apply it to an OpenShift Container Platform cluster for use with Operator Lifecycle Manager (OLM).

Prerequisites

- An index image built and pushed to a registry.

Procedure

1. Create a CatalogSource object that references your index image.
a. Modify the following to your specifications and save it as a `catalogSource.yaml` file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: my-operator-catalog
  namespace: openshift-marketplace
spec:
  sourceType: grpc
  displayName: My Operator Catalog
  publisher: <publisher_name>
  updateStrategy:
    registryPoll:
      interval: 30m
```

<> Specify your index image. <> Specify your name or an organization name publishing the catalog. <> Catalog sources can automatically check for new versions to keep up to date.

b. Use the file to create the `CatalogSource` object:

```
$ oc apply -f catalogSource.yaml
```

2. Verify the following resources are created successfully.

a. Check the pods:

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

**Example output**

```
NAME                                    READY   STATUS    RESTARTS  AGE
my-operator-catalog-6njx6               1/1     Running   0         28s
marketplace-operator-d9f549946-96sgr    1/1     Running   0         26h
```

b. Check the catalog source:

```
$ oc get catalogsource -n openshift-marketplace
```

**Example output**

```
NAME                  DISPLAY               TYPE PUBLISHER  AGE
my-operator-catalog   My Operator Catalog  grpc            5s
```

c. Check the package manifest:

```
$ oc get packagemanifest -n openshift-marketplace
```

**Example output**

```
NAME                          CATALOG               AGE
jaeger-product                My Operator Catalog   93s
```
You can now install the Operators from the OperatorHub page on your OpenShift Container Platform web console.

1.4.2.5. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.7 target cluster in a restricted environment

You can install the Migration Toolkit for Containers (MTC) on an OpenShift Container Platform 4.7 target cluster.

Prerequisites

- You must be logged in as a user with cluster-admin privileges on all clusters.
- You must create a custom Operator catalog and push it to a mirror registry.
- You must configure Operator Lifecycle Manager to install the Migration Toolkit for Containers Operator from the mirror 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.

   NOTE

   Do not change the subscription approval option to Automatic. The Migration Toolkit for Containers version must be the same on the source and the target clusters.

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.

1.4.2.6. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 3 source cluster in a restricted environment

You can create a manifest file based on the Migration Toolkit for Containers (MTC) Operator image and edit the manifest to point to your local image registry. Then, you can use the local image to create the Migration Toolkit for Containers Operator on an OpenShift Container Platform 3 source cluster.
IMPORTANT

You must install the same MTC version on the OpenShift Container Platform 3 and 4 clusters.

To ensure that you have the latest version on the OpenShift Container Platform 3 cluster, download the operator.yml and controller-3.yml files when you are ready to create and run the migration plan.

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.
- The source cluster must be OpenShift Container Platform 3.7, 3.9, 3.10, or 3.11.
- You must have a Linux workstation with unrestricted network access.
- You must have access to a mirror registry that supports Docker v2-2

Procedure

1. On the workstation with unrestricted network access, 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:

   ```
   $ sudo podman cp $(sudo podman create \n registry.redhat.io/rhmtc/openshift-migration-rhel7-operator:v1.4):/operator.yml ./
   ```

3. Download the controller-3.yml file:

   ```
   $ sudo podman cp $(sudo podman create \n registry.redhat.io/rhmtc/openshift-migration-rhel7-operator:v1.4):/controller-3.yml ./
   ```

4. Obtain the Operator image value from the mapping.txt file that was created when you ran the oc adm catalog mirror on the OpenShift Container Platform 4 cluster:

   ```
   $ grep openshift-migration-rhel7-operator ./mapping.txt | grep rhmtc
   ```

   The output shows the mapping between the registry.redhat.io image and your mirror registry image.

   **Example output**

   ```
   registry.redhat.io/rhmtc/openshift-migration-rhel7-operator@sha256:468a6126f73b1ee12085ca53a312d1f96ef5a2ca03442bcb63724af5e2614e8a=<<registry.apps.example.com>/rhmtc/openshift-migration-rhel7-operator
   ```
5. Update the **image** and **REGISTRY** values in the Operator configuration file:

```yaml
containers:
  - name: ansible
    image: <registry.apps.example.com>/rhmtc/openshift-migration-rhel7-operator@sha256:<468a6126f73b1ee12085ca53a312d1f96ef5a2ca03442bcb63724af5e2614e8a>
  ...  
  - name: operator
    image: <registry.apps.example.com>/rhmtc/openshift-migration-rhel7-operator@sha256:<468a6126f73b1ee12085ca53a312d1f96ef5a2ca03442bcb63724af5e2614e8a>
  ...  
  env:
    - name: REGISTRY
      value: <registry.apps.example.com>
```

1. Specify your mirror registry and the **sha256** value of the Operator image in the **mapping.txt** file.

2. Specify your mirror registry and the **sha256** value of the Operator image in the **mapping.txt** file.

3. Specify your mirror registry.


7. Create the Migration Toolkit for Containers Operator object:

```bash
$ 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ole.rbac.authorization.k8s.io/migration-operator created
rolebinding.rbac.authorization.k8s.io/migration-operator created
clusterrolebinding.rbac.authorization.k8s.io/migration-operator created
deployment.apps/migration-operator created
Error from server (AlreadyExists): error when creating "./operator.yml":
rolebindings.rbac.authorization.k8s.io "system:image-builders" already exists 1
Error from server (AlreadyExists): error when creating "./operator.yml":
rolebindings.rbac.authorization.k8s.io "system:image-pullers" already exists
```

1. You can ignore **Error from server (AlreadyExists)** messages. They are caused by the Migration Toolkit for Containers Operator creating resources for earlier versions of OpenShift Container Platform 3 that are provided in later releases.

8. Create the **MigrationController** object:

```bash
$ oc create -f controller-3.yml
```
9. Verify that the Velero and Restic pods are running:

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

1.4.3. Upgrading the Migration Toolkit for Containers

You can upgrade the Migration Toolkit for Containers (MTC) by using the OpenShift Container Platform web console.

**IMPORTANT**

You must ensure that the same MTC version is installed on all clusters.

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

1.4.3.1. Upgrading the Migration Toolkit for Containers on an OpenShift Container Platform 4 cluster

You can upgrade the Migration Toolkit for Containers (MTC) on an OpenShift Container Platform 4 cluster by using the OpenShift Container Platform web console.

**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.

1.4.3.2. Upgrading the Migration Toolkit for Containers on an OpenShift Container Platform 3 cluster

You can upgrade Migration Toolkit for Containers (MTC) on an OpenShift Container Platform 3 cluster with podman.

**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:
   
   ```bash
   $ sudo podman login registry.redhat.io
   ```

2. Download the latest `operator.yml` file:
   
   ```bash
   $ sudo podman cp $(sudo podman create \
   registry.redhat.io/rhmtc/openshift-migration-rhel7-operator:v1.4):/operator.yml ./
   ```
   
   You can specify a z-stream release, if necessary.

3. Replace the Migration Toolkit for Containers Operator:
   
   ```bash
   $ oc replace --force -f operator.yml
   ```

4. Apply the changes:
   
   - For MTC 1.1.2 and earlier versions, delete the `Restic` pods:
     
     ```bash
     $ oc delete pod <restic_pod>
     ```
   
   - For MTC 1.2 and later versions:
     
     a. Scale the `migration-operator` deployment to 0 to stop the deployment:
     
     ```bash
     $ oc scale -n openshift-migration --replicas=0 deployment/migration-operator
     ```

     b. Scale the `migration-operator` deployment to 1 to start the deployment and apply the changes:
     
     ```bash
     $ oc scale -n openshift-migration --replicas=1 deployment/migration-operator
     ```

5. Verify that the `migration-operator` was upgraded:
   
   ```bash
   $ oc -o yaml -n openshift-migration get deployment/migration-operator | grep image: | awk -F "":" " '{ print $NF }'
   ```

6. Download the latest `controller-3.yml` file:
   
   ```bash
   $ sudo podman cp $(sudo podman create \
   registry.redhat.io/rhmtc/openshift-migration-rhel7-operator:v1.4):/controller-3.yml ./
   ```

7. Create the `migration-controller` object:
   
   ```bash
   $ oc create -f controller-3.yml
   ```
8. If your OpenShift Container Platform version is 3.10 or earlier, set the security context constraint of the migration-controller service account to anyuid to enable direct image migration and direct volume migration:

   $ oc adm policy add-scc-to-user anyuid -z migration-controller -n openshift-migration

9. Verify that the MTC pods are running:

   $ oc get pods -n openshift-migration

10. 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:

      $ 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.

1.4.3.3. Upgrading MTC 1.3 to 1.4

If you are upgrading Migration Toolkit for Containers (MTC) version 1.3.x to 1.4, 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 have MTC 1.3 installed.
- You must be logged in as a user with cluster-admin privileges.

Procedure

1. Log in to the cluster on which the MigrationController pod is running.

2. Get the MigPlan CR manifest:

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

3. Update the following parameter values and save the file as migplan.yaml:
4. Replace the *MigPlan* CR manifest to apply the changes:

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

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

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

### 1.5. CONFIGURING OBJECT STORAGE FOR 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](https://docs.openshift.com/container-platform/4.8/migration-toolkit/migration-toolkit-mtc-overview.html) 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:

- Multi-Cloud Object Gateway (MCG)
- Amazon Web Services (AWS) S3
- Google Cloud Provider (GCP)
- Microsoft Azure
- Generic S3 object storage, for example, Minio or Ceph S3

In a restricted environment, you can create an internally hosted replication repository.

#### 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.

### 1.5.1. Configuring a Multi-Cloud Object Gateway storage bucket as a replication repository

You can install the OpenShift Container Storage Operator and configure a Multi-Cloud Object Gateway (MCG) storage bucket as a replication repository for the Migration Toolkit for Containers (MTC).

#### 1.5.1.1. Installing the OpenShift Container Storage Operator

You can install the OpenShift Container Storage Operator from OperatorHub.
Procedure

1. In the OpenShift Container Platform web console, click Operators → OperatorHub.

2. Use Filter by keyword (in this case, OCS) to find the OpenShift Container Storage Operator.

3. Select the OpenShift Container Storage Operator and click Install.

4. Select an Update Channel, Installation Mode, and Approval Strategy.

5. Click Install.

On the Installed Operators page, the OpenShift Container Storage Operator appears in the openshift-storage project with the status Succeeded.

1.5.1.2. Creating the Multi-Cloud Object Gateway storage bucket

You can create the Multi-Cloud Object Gateway (MCG) storage bucket’s custom resources (CRs).

Procedure

1. Log in to the OpenShift Container Platform cluster:

   $ oc login -u <username>

2. Create the NooBaa CR configuration file, noobaa.yml, with the following content:

   ```yaml
   apiVersion: noobaa.io/v1alpha1
   kind: NooBaa
   metadata:
     name: noobaa
     namespace: openshift-storage
   spec:
     dbResources:
       requests:
         cpu: 0.5
         memory: 1Gi
     coreResources:
       requests:
         cpu: 0.5
         memory: 1Gi
   ```

1 2 For a very small cluster, you can change the `cpu` value to 0.1.

3. Create the NooBaa object:

   $ oc create -f noobaa.yml

4. Create the BackingStore CR configuration file, bs.yml, with the following content:

   ```yaml
   apiVersion: noobaa.io/v1alpha1
   kind: BackingStore
   metadata:
   finalizers:
   - noobaa.io/finalizer
   ```
Specify the number of volumes in the persistent volume pool.

Specify the size of the volumes.

Specify the storage class.

5. Create the **BackingStore** object:

   $ oc create -f bs.yml

6. Create the **BucketClass** CR configuration file, **bc.yml**, with the following content:

   ```yaml
   apiVersion: noobaa.io/v1alpha1
   kind: BucketClass
   metadata:
     labels:
       app: noobaa
       name: mcg-pv-pool-bc
       namespace: openshift-storage
   spec:
     placementPolicy:
       tiers:
         - backingStores:
           - mcg-pv-pool-bs
       placement: Spread
   ``

   $ oc create -f bc.yml

7. Create the **BucketClass** object:

   $ oc create -f bc.yml

8. Create the **ObjectBucketClaim** CR configuration file, **obc.yml**, with the following content:

   ```yaml
   apiVersion: objectbucket.io/v1alpha1
   kind: ObjectBucketClaim
   metadata:
     name: migstorage
     namespace: openshift-storage
   spec:
     bucketName: migstorage
   ```
1. Record the bucket name for adding the replication repository to the MTC web console.

9. Create the **ObjectBucketClaim** object:

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

10. Watch the resource creation process to verify that the **ObjectBucketClaim** status is **Bound**:

    ```
    $ watch -n 30 'oc get -n openshift-storage objectbucketclaim migstorage -o yaml'
    ```

    This process can take five to ten minutes.

11. Obtain and record the following values, which are required when you add the replication repository to the MTC web console:

    - S3 endpoint:
      ```
      $ oc get route -n openshift-storage s3
      ```
    - S3 provider access key:
      ```
      $ oc get secret -n openshift-storage migstorage -o go-template='{{.data.AWS_ACCESS_KEY_ID }}' | base64 --decode
      ```
    - S3 provider secret access key:
      ```
      $ oc get secret -n openshift-storage migstorage -o go-template='{{.data.AWS_SECRET_ACCESS_KEY }}' | base64 --decode
      ```

### 1.5.2. Configuring an AWS S3 storage bucket as a replication repository

You can configure an AWS S3 storage bucket as a replication repository for the Migration Toolkit for Containers (MTC).

**Prerequisites**

- The AWS S3 storage bucket must be accessible to the source and target clusters.
- You must have the **AWS CLI** installed.
- 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. Create an AWS S3 bucket:

   ```bash
   $ aws s3api create-bucket
     --bucket <bucket_name> 1
     --region <bucket_region> 2
   ``

   1 Specify your S3 bucket name.
   2 Specify your S3 bucket region, for example, `us-east-1`.

2. Create the IAM user `velero`:

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

3. Create an EC2 EBS snapshot policy:

   ```bash
   $ cat > velero-ec2-snapshot-policy.json <<EOF
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": ["ec2:DescribeVolumes",
                     "ec2:DescribeSnapshots",
                     "ec2:CreateTags",
                     "ec2:CreateVolume",
                     "ec2:CreateSnapshot",
                     "ec2:DeleteSnapshot"
                   ],
         "Resource": "*"
       }
     ]
   }
   EOF
   ``

4. Create an AWS S3 access policy for one or for all S3 buckets:

   ```bash
   $ cat > velero-s3-policy.json <<EOF
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": ["s3:GetObject",
                     "s3:DeleteObject",
                     "s3:PutObject",
                     "s3:AbortMultipartUpload",
                     "s3:ListMultipartUploadParts"
                   ],
         "Resource": [
   ```
To grant access to a single S3 bucket, specify the bucket name. To grant access to all AWS S3 buckets, specify * instead of a bucket name as in the following example:

Example output

```
"Resource": [
  "arn:aws:s3:::*"
]
```

5. Attach the EC2 EBS policy to velero:

```
$ aws iam put-user-policy \
  --user-name velero \
  --policy-name velero-ebs \
  --policy-document file:///velero-ec2-snapshot-policy.json
```

6. Attach the AWS S3 policy to velero:

```
$ aws iam put-user-policy \
  --user-name velero \
  --policy-name velero-s3 \
  --policy-document file:///velero-s3-policy.json
```

7. Create an access key for velero:

```
$ aws iam create-access-key --user-name velero \
  
```
Record the `AWS_SECRET_ACCESS_KEY` and the `AWS_ACCESS_KEY_ID` for adding the AWS repository to the MTC web console.

1.5.3. Configuring a Google Cloud Provider storage bucket as a replication repository

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

**Prerequisites**
- The GCP storage bucket must be accessible to the source and target clusters.
- You must have `gsutil` installed.
- 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 `gsutil`:
   
   ```bash
   $ gsutil init
   ```

   **Example output**
   
   Welcome! This command will take you through the configuration of gcloud.
   
   Your current configuration has been set to: [default]
   
   To continue, you must login. Would you like to login (Y/n)?

2. Set the `BUCKET` variable:
   
   ```bash
   $ BUCKET=<bucket_name>
   ```

   Specify your bucket name.

3. Create a storage bucket:
   
   ```bash
   $ gsutil mb gs://$BUCKET/
   ```

4. Set the `PROJECT_ID` variable to your active project:
   
   ```bash
   $ PROJECT_ID=`gcloud config get-value project`
   ```

5. Create a `velero` IAM service account:
$ gcloud iam service-accounts create velero \
   --display-name "Velero Storage"

6. Create the `SERVICE_ACCOUNT_EMAIL` variable:

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

7. Create the `ROLE_PERMISSIONS` variable:

   $ 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
   )

8. Create the `velero.server` custom role:

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

9. 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

10. Update the IAM service account:

    $ gsutil iam ch serviceAccount:$SERVICE_ACCOUNT_EMAIL:objectAdmin gs://$BUCKET

11. 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

### 1.5.4. Configuring a Microsoft Azure Blob storage container as a replication repository

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

**Prerequisites**
You must have an Azure storage account.

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. Set the `AZURE_RESOURCE_GROUP` variable:

   ```bash
   $ AZURE_RESOURCE_GROUP=Velero_Backups
   ```

2. Create an Azure resource group:

   ```bash
   $ az group create -n $AZURE_RESOURCE_GROUP --location <CentralUS>
   ```
   
   Specify your location.

3. Set the `AZURE_STORAGE_ACCOUNT_ID` variable:

   ```bash
   $ AZURE_STORAGE_ACCOUNT_ID=velerobackups
   ```

4. Create an Azure storage account:

   ```bash
   $ 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
   ```

5. Set the `BLOB_CONTAINER` variable:

   ```bash
   $ BLOB_CONTAINER=velero
   ```

6. Create an Azure Blob storage container:

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

7. Create a service principal and credentials for velero:
Save the service principal credentials in the credentials-velero file:

```
$ 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` 

$ 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
```

8. 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
```

### 1.6. MIGRATING YOUR APPLICATIONS

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

#### 1.6.1. Prerequisites

The Migration Toolkit for Containers (MTC) has the following prerequisites:

- You must be logged in as a user with `cluster-admin` privileges on all clusters.
- The MTC version must be the same on all clusters.
- 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 4.7 cluster.
- **Clusters:**
  - The source cluster must be upgraded to the latest MTC z-stream release.
  - The cluster on which the `migration-controller` pod is running must have unrestricted network access to the other clusters.
  - The clusters must have unrestricted network access to each other.
  - The clusters must have unrestricted network access to the replication repository.
  - The clusters must be able to communicate using OpenShift routes on port 443.
  - The clusters must have no critical conditions.
  - The clusters must be in a ready state.
- **Volume migration:**
  - The persistent volumes (PVs) must be valid.
The PVs must be bound to persistent volume claims.

- If you copy the PVs by using the move method, the clusters must have unrestricted network access to the remote volume.

- If you copy the PVs by using the snapshot copy method, the following prerequisites apply:
  - The cloud provider must support snapshots.
  - The volumes must have the same cloud provider.
  - The volumes must be located in the same geographic region.
  - The volumes must have the same storage class.

- If you perform a direct volume migration in a proxy environment, you must configure an Stunnel TCP proxy.

- If you perform a direct image migration, you must expose the internal registry of the source cluster to external traffic.

1.6.1.1. Updating deprecated internal images with podman

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

If the OpenShift Container Platform 3 image is deprecated in OpenShift Container Platform 4.7, 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.

Procedure

1. Expose the internal registries on the source and target clusters.

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 source cluster registry:

   $ podman login -u $(oc whoami) -p $(oc whoami -t) --tls-verify=false <source_cluster>

4. Log in to the target cluster registry:

   $ podman login -u $(oc whoami) -p $(oc whoami -t) --tls-verify=false <target_cluster>

5. Pull the deprecated image:

   $ podman pull <source_cluster>/openshift/<image>
6. Tag the image for the target cluster registry:

   $ podman tag <source_cluster>/openshift/<image> <target_cluster>/openshift/<image>

7. Push the image to the target cluster 4 registry:

   $ podman push <target_cluster>/openshift/<image>

8. Verify that the image has a valid image stream on the target cluster:

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

   **Example output**

   

<table>
<thead>
<tr>
<th>&lt;image&gt;</th>
<th>&lt;target_cluster&gt;/openshift/&lt;image&gt;</th>
<th>&lt;versions&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>more...</td>
<td>6 seconds ago</td>
<td></td>
</tr>
</tbody>
</table>

1.6.1.2. Creating a CA certificate bundle file

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> \  
   | sed -ne '/-BEGIN CERTIFICATE-/,-END CERTIFICATE-/p' > <ca_bundle.cert>

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.

1.6.1.3. Configuring a proxy for direct volume migration

If you are performing direct volume migration from a source cluster behind a proxy, you must configure an Stunnel proxy in the **MigrationController** custom resource (CR). Stunnel creates a transparent tunnel between the source and target clusters for the TCP connection without changing the certificates.

**NOTE**

Direct volume migration supports only one proxy. The source cluster cannot access the route of the target cluster if the target cluster is also behind a proxy.

**Prerequisites**

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

1. Log in to the cluster on which the `MigrationController` pod runs.

2. Get the `MigrationController` CR manifest:

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

3. Add the `stunnel_tcp_proxy` parameter:

   ```
   apiVersion: migration.openshift.io/v1alpha1
   kind: MigrationController
   metadata:
     name: migration-controller
     namespace: openshift-migration
   ...
   spec:
     stunnel_tcp_proxy: <stunnel_proxy>  # 1
   ```

   Specify the Stunnel proxy: `http://<user_name>:@<ip_address>:<port>`.

4. Save the manifest as `migration-controller.yaml`.

5. Apply the updated manifest:

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

1.6.1.4. 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.

1.6.1.4.1. Ansible modules

You can use the Ansible `shell` module to run `oc` commands.

**Example shell module**

```
- hosts: localhost
  gather_facts: false
  tasks:
    - name: get pod name
      shell: oc get po --all-namespaces
```

You can use `kubernetes.core` modules, such as `k8s_info`, to interact with Kubernetes resources.

**Example k8s_facts module**
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**

```yaml
- 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 }}"
```

**1.6.1.4.2. 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` plug-in.

**Example environment variables**

```yaml
- 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
    - set_fact:
      namespaces: "{{ (lookup('env', 'migration_namespaces')).split(',') }}"
    - debug:
      msg: "{{ item }}"
      with_items: "{{ namespaces }}"
    - debug:
      msg: "{{ lookup('env', 'migplan_name') }}"
```
1.6.1.5. Additional resources

- About migration hooks
- MigHook custom resource
- MigPlan custom resource

1.6.2. 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.

1.6.2.1. 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.

1.6.2.2. 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.

**Procedure**

1. Log in to the cluster.
2. Obtain the **migration-controller** service account token:

   ```bash
   $ oc sa get-token migration-controller -n openshift-migration
   eyJhbGciOiJSUzI1NiIsImtlmpZCI6Ij9.eJpc3MiOiJrdWJlcm5ldGVzL2c6NjcxMDI2MDc6MCJ9.
   ...
   ...
   eyJzdWIiOiJkaXN0aWNzIiwiaWQiOjEwMjQzLjEwIiwibGFuZ3VhZC1zaWQ6IjEwMjQzLjEwIn0.
   ...
   ...
   ```

   **Example output**

3. In the MTC web console, click **Clusters**.
4. Click **Add cluster**.
5. Fill in the following fields:

   - **Cluster name**: The cluster name can contain lower-case letters (a-z) and numbers (0-9). It must not contain spaces or international characters.
   - **URL**: Specify the API server URL, for example, `https://<www.example.com>:8443`.
   - **Service account token**: Paste the **migration-controller** service account token.
   - **Exposed route host to image registry**: If you are using direct image migration, specify the exposed route to the image registry of the source cluster, for example, `www.example.apps.cluster.com`. You can specify a port. The default port is **5000**.
   - **Azure cluster**: You must select this option if you use Azure snapshots to copy your data.
   - **Azure resource group**: This field is displayed if **Azure cluster** is selected. Specify the Azure resource group.

   - **Require SSL verification**: Optional. Select this option to verify SSL connections to the...
**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.

### 1.6.2.3. Adding a replication repository to the MTC web console

You can add an object storage bucket as a replication repository to the Migration Toolkit for Containers (MTC) web console.

**Prerequisites**

- You must configure an object storage bucket for migrating the data.

**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 AWS S3, MCG, and generic S3 providers:
     - **Replication repository name**: Specify the replication repository name in the MTC web console.
     - **S3 bucket name**: Specify the name of the S3 bucket you created.
     - **S3 bucket region**: Specify the S3 bucket region. **Required** for AWS S3. **Optional** for other S3 providers.
     - **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.
     - **S3 provider secret access key**: Specify the `<AWS_ACCESS_KEY_ID>` for AWS or the S3 provider secret access key for MCG.
     - **Require SSL verification**: Clear this check box if you are using a generic S3 provider.
     - If you use a custom CA bundle, click **Browse** and browse to the Base64-encoded CA bundle 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.

1.6.2.4. 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, the MigCluster custom resource manifest of the source cluster must specify the exposed route of the internal image registry.

Procedure

1. In the MTC web console, click Migration plans.

2. Click Add migration plan.

3. Enter the Plan name and click Next.
   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.

5. Select a Target cluster.

6. Select a Replication repository.

7. Select the projects to be migrated and click Next.

8. Select a Source cluster, a Target cluster, and a Repository, and click Next.
9. On the **Namespaces** page, select the projects to be migrated and click **Next**.

10. On the **Persistent volumes** page, click 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.

11. Click **Next**.

12. On the **Copy options** page, 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.

13. 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.

14. Select a **Target storage class**.
   If you selected **Filesystem copy**, you can change the target storage class.

15. Click **Next**.

16. 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.

17. Click **Next**.

18. Optional: On the **Hooks** page, 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**.

19. Click **Finish**.

The migration plan is displayed in the **Migration plans** list.

### 1.6.2.5. Running a migration plan in the MTC web console

You can stage or 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 source cluster.

2. Delete old images:

   $ oc adm prune images

3. Log in to the MTC web console and click **Migration plans**.
4. Click the Options menu next to a migration plan and select Stage to copy data from the source cluster to the target cluster without stopping the application. You can run Stage multiple times to reduce the actual migration time.

5. When you are ready to migrate the application workload, the Options menu beside a migration plan and select Migrate.

6. Optional: In the Migrate window, you can select Do not stop applications on the source cluster during migration.

7. Click Migrate.

8. 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.

1.6.3. Migrating your applications from the command line

You can migrate your applications on the command line by using the MTC custom resources (CRs).

You can migrate applications from a local cluster to a remote cluster, from a remote cluster to a local cluster, and between remote clusters.

MTC terminology
The following terms are relevant for configuring clusters:

- **host** cluster:
  - The migration-controller pod runs on the host cluster.
  - A host cluster does not require an exposed secure registry route for direct image migration.

- Local cluster: The local cluster is often the same as the host cluster but this is not a requirement.

- Remote cluster:
  - A remote cluster must have an exposed secure registry route for direct image migration.
  - A remote cluster must have a Secret CR containing the migration-controller service account token.

The following terms are relevant for performing a migration:
• Source cluster: Cluster from which the applications are migrated.
• Destination cluster: Cluster to which the applications are migrated.

1.6.3.1. Migrating your applications with the Migration Toolkit for Containers API

You can migrate your applications on the command line with the Migration Toolkit for Containers (MTC) API.

You can migrate applications from a local cluster to a remote cluster, from a remote cluster to a local cluster, and between remote clusters.

This procedure describes how to perform indirect migration and direct 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 migration: Images or volumes are copied directly from the source cluster to the destination cluster. Direct image migration and direct volume migration have significant performance benefits.

You create the following custom resources (CRs) to perform a migration:

• MigCluster CR: Defines a host, local, or remote cluster
  The migration-controller pod runs on the host cluster.

• Secret CR: Contains credentials for a remote cluster or storage

• MigStorage CR: Defines a replication repository
  Different storage providers require different parameters in the MigStorage CR manifest.

• MigPlan CR: Defines a migration plan

• MigMigration CR: Performs a migration defined in an associated MigPlan
  You can create multiple MigMigration CRs for a single MigPlan CR for the following purposes:

  • To perform stage migrations, which copy most of the data without stopping the application, before running a migration. Stage migrations improve the performance of the migration.

  • To cancel a migration in progress

  • To roll back a completed migration

Prerequisites

• You must have cluster-admin privileges for all clusters.

• You must install the OpenShift Container Platform CLI (oc).

• You must install the Migration Toolkit for Containers Operator on all clusters.

• The version of the installed Migration Toolkit for Containers Operator must be the same on all clusters.

• You must configure an object storage as a replication repository.
- If you are using direct image migration, you must expose a secure registry route on all remote clusters.
- If you are using direct volume migration, the source cluster must not have an HTTP proxy configured.

Procedure

1. Create a `MigCluster` CR manifest for the host cluster called `host-cluster.yaml`:

   ```yaml
   apiVersion: migration.openshift.io/v1alpha1
   kind: MigCluster
   metadata:
     name: host
     namespace: openshift-migration
   spec:
     isHostCluster: true
   ```

2. Create a `MigCluster` CR for the host cluster:

   ```bash
   $ oc create -f host-cluster.yaml -n openshift-migration
   ```

3. Create a `Secret` CR manifest for each remote cluster called `cluster-secret.yaml`:

   ```yaml
   apiVersion: v1
   kind: Secret
   metadata:
     name: <cluster_secret>
     namespace: openshift-config
   type: Opaque
   data:
     saToken: <sa_token>  
   ```

   1 Specify the base64-encoded migration-controller service account (SA) token of the remote cluster.

   You can obtain the SA token by running the following command:

   ```bash
   $ oc sa get-token migration-controller -n openshift-migration | base64 -w 0
   ```

4. Create a `Secret` CR for each remote cluster:

   ```bash
   $ oc create -f cluster-secret.yaml
   ```

5. Create a `MigCluster` CR manifest for each remote cluster called `remote-cluster.yaml`:

   ```yaml
   apiVersion: migration.openshift.io/v1alpha1
   kind: MigCluster
   metadata:
     name: <remote_cluster>
     namespace: openshift-migration
   spec:
     exposedRegistryPath: <exposed_registry_route>  
   ```
Optional: Specify the exposed registry route, for example, docker-registry-default.apps.example.com if you are using direct image migration.

SSL verification is enabled if false. CA certificates are not required or checked if true.

Specify the Secret CR of the remote cluster.

Specify the URL of the remote cluster.

6. Create a MigCluster CR for each remote cluster:

   $ oc create -f remote-cluster.yaml -n openshift-migration

7. Verify that all clusters are in a Ready state:

   $ oc describe cluster <cluster_name>

8. Create a Secret CR manifest for the replication repository called storage-secret.yaml:

   ```yaml
   apiVersion: v1
   kind: Secret
   metadata:
     namespace: openshift-config
   name: <migstorage_creds>
   type: Opaque
   data:
     aws-access-key-id: <key_id_base64>  
     aws-secret-access-key: <secret_key_base64>  
   
   1 Specify the key ID in base64 format.
   2 Specify the secret key in base64 format.
   ```

AWS credentials are base64-encoded by default. If you are using another storage provider, you must encode your credentials by running the following command with each key:

   $ echo -n "<key>" | base64 -w 0

   1 Specify the key ID or the secret key. Both keys must be base64-encoded.

9. Create the Secret CR for the replication repository:

   $ oc create -f storage-secret.yaml
10. Create a **MigStorage** CR manifest for the replication repository called *migstorage.yaml*:

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigStorage
metadata:
  name: <storage_name>
  namespace: openshift-migration
spec:
  backupStorageConfig:
    awsBucketName: <bucket_name>  
    credsSecretRef:
      name: <storage_secret_ref>  
      namespace: openshift-config
  backupStorageProvider: <storage_provider_name>  
  volumeSnapshotConfig:
    credsSecretRef:
      name: <storage_secret_ref>  
      namespace: openshift-config
  volumeSnapshotProvider: <storage_provider_name>  
```

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.

11. Create the **MigStorage** CR:

```
$ oc create -f migstorage.yaml -n openshift-migration
```

12. Verify that the **MigStorage** CR is in a **Ready** state:

```
$ oc describe migstorage <migstorage_name>
```

13. Create a **MigPlan** CR manifest called *migplan.yaml*:

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigPlan
metadata:
  name: <migration_plan>
  namespace: openshift-migration
spec:
  destMigClusterRef:
    name: host
    namespace: openshift-migration
  indirectImageMigration: true
```

1. Optional: If you are copying data by using snapshots, specify the storage provider.
indirectVolumeMigration: true
migStorageRef:
  name: <migstorage_ref>
namespace: openshift-migration
namespaces:
  - <application_namespace>
srcMigClusterRef:
  name: <remote_cluster_ref>
namespace: openshift-migration

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 namespaces to be migrated.
5. Specify the name of the source cluster MigCluster instance.

14. Create the MigPlan CR:

   $ oc create -f migplan.yaml -n openshift-migration

15. View the MigPlan instance to verify that it is in a Ready state:

   $ oc describe migplan <migplan_name> -n openshift-migration

16. Create a MigMigration CR manifest called migmigration.yaml:

   apiVersion: migration.openshift.io/v1alpha1
   kind: MigMigration
   metadata:
     name: <migmigration_name>
     namespace: openshift-migration
   spec:
     migPlanRef:
       name: <migplan_name>
       namespace: openshift-migration
     quiescePods: true
     stage: false
     rollback: false

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.

17. Create the MigMigration CR to start the migration defined in the MigPlan CR:
$ oc create -f migmigration.yaml -n openshift-migration

18. Verify the progress of the migration by watching the **MigMigration** CR:

$ oc watch migmigration <migmigration_name> -n openshift-migration

The output resembles the following:

**Example output**

<table>
<thead>
<tr>
<th>Name:</th>
<th>c8b034c0-6567-11eb-9a4f-0bc004db0fbc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace:</td>
<td>openshift-migration</td>
</tr>
<tr>
<td>Labels:</td>
<td>migration.openshift.io/migplan-name=django</td>
</tr>
<tr>
<td>Annotations:</td>
<td>openshift.io/touch: e99f9083-6567-11eb-8420-0a580a81020c</td>
</tr>
<tr>
<td>API Version:</td>
<td>migration.openshift.io/v1alpha1</td>
</tr>
<tr>
<td>Kind:</td>
<td>MigMigration</td>
</tr>
</tbody>
</table>

... 

**Spec:**

- Mig Plan Ref:
  - Name: my_application
  - 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
1.6.3.2. MTC custom resource manifests

Migration Toolkit for Containers (MTC) uses the following custom resource (CR) manifests to create CRs for migrating applications.

1.6.3.2.1. DirectImageMigration

The **DirectImageMigration** CR copies images directly from the source cluster to the destination cluster.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: DirectImageMigration
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: <directimagemigration_name>
spec:
  srcMigClusterRef:
    name: <source_cluster_ref> 1
    namespace: openshift-migration
destMigClusterRef:
    name: <destination_cluster_ref> 2
    namespace: openshift-migration
  namespaces:
  - <namespace> 3
```
1. Specify the **MigCluster** CR name of the source cluster.
2. Specify the **MigCluster** CR name of the destination cluster.
3. Specify one or more namespaces containing images to be migrated.

### 1.6.3.2.2. DirectImageStreamMigration

The **DirectImageStreamMigration** CR copies image stream references directly from the source cluster to the destination cluster.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: DirectImageStreamMigration
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: directimagestreammigration_name
spec:
  srcMigClusterRef:
    name: <source_cluster_ref> 1
    namespace: openshift-migration
destMigClusterRef:
    name: <destination_cluster_ref> 2
    namespace: openshift-migration
imageStreamRef:
  name: <image_stream_name> 3
  namespace: <source_image_stream_namespace> 4
destNamespace: <destination_image_stream_namespace> 5
```

1. Specify the **MigCluster** CR name of the source cluster.
2. Specify the **MigCluster** CR name of the destination cluster.
3. Specify the image stream name.
4. Specify the image stream namespace on the source cluster.
5. Specify the image stream namespace on the destination cluster.

### 1.6.3.2.3. DirectVolumeMigration

The **DirectVolumeMigration** CR copies persistent volumes (PVs) directly from the source cluster to the destination cluster.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: DirectVolumeMigration
metadata:
  name: <directvolumemigration_name>
  namespace: openshift-migration
spec:
  createDestinationNamespaces: false 1
deleteProgressReportingCRs: false 2
destMigClusterRef:
```

1. Specify the **MigCluster** CR name of the source cluster.
2. Specify the **MigCluster** CR name of the destination cluster.
Namespaces are created for the PVs on the destination cluster if true.

The `DirectVolumeMigrationProgress` CRs are deleted after migration if true. The default value is false so that `DirectVolumeMigrationProgress` CRs are retained for troubleshooting.

Update the cluster name if the destination cluster is not the host cluster.

Specify one or more PVCs to be migrated with direct volume migration.

Specify the namespace of each PVC.

Specify the `MigCluster` CR name of the source cluster.

### 1.6.3.2.4. DirectVolumeMigrationProgress

The `DirectVolumeMigrationProgress` CR shows the progress of the `DirectVolumeMigration` CR.

```yaml
apiVersion: migration.openshift.io/v1alpha1
group: migration.openshift.io
name: directvolumemigrationprogress
kind: DirectVolumeMigrationProgress
metadata:
  name: directvolumemigrationprogress
spec:
  clusterRef:
    name: source_cluster
    namespace: openshift-migration
  podRef:
    name: rsync_pod
    namespace: openshift-migration
```

### 1.6.3.2.5. MigAnalytic

The `MigAnalytic` CR collects the number of images, Kubernetes resources, and the PV capacity from an associated `MigPlan` CR.

```yaml
apiVersion: migration.openshift.io/v1alpha1
group: migration.openshift.io
name: miganalytic
kind: MigAnalytic
metadata:
  name: miganalytic
  namespace: openshift-migration
  labels:
    controller-tools.k8s.io: "1.0"
```

name: host
namespace: openshift-migration
persistentVolumeClaims:
- name: <pvc_name>
  namespace: <pvc_namespace>
srcMigClusterRef:
  name: <source_cluster_ref>
  namespace: openshift-migration

The `DirectVolumeMigrationProgress` CRs are deleted after migration if true. The default value is false so that `DirectVolumeMigrationProgress` CRs are retained for troubleshooting.
migplan: <migplan_name>  
spec:
  analyzeImageCount: true  
  analyzeK8SResources: true  
  analyzePVCapacity: true  
  listImages: false  
  listImagesLimit: 50  
migPlanRef:
  name: migplan_name  
  namespace: openshift-migration

Specify the **MigPlan** CR name associated with the **MigAnalytic** CR.

Specify the **MigPlan** CR name associated with the **MigAnalytic** CR.

Optional: The number of images is returned if **true**.

Optional: Returns the number, kind, and API version of the Kubernetes resources if **true**.

Optional: Returns the PV capacity if **true**.

Returns a list of image names if **true**. Default is **false** so that the output is not excessively long.

Optional: Specify the maximum number of image names to return if **listImages** is **true**.

Specify the **MigPlan** CR name associated with the **MigAnalytic** CR.

1.6.3.2.6. 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  
  namespace: openshift-migration
spec:
  isHostCluster: true
  azureResourceGroup: <azure_resource_group>  
  caBundle: <ca_bundle_base64>  
  insecure: false  
  refresh: false

# The 'restartRestic' parameter is relevant for a source cluster.
# restartRestic: true
# The following parameters are relevant for a remote cluster.
# isHostCluster: false
# exposedRegistryPath:  
# url: <destination_cluster_url>
# serviceAccountSecretRef:
#   name: <source_secret_ref>
#   namespace: openshift-config
Optional: Update the cluster name if the migration-controller pod is not running on this cluster.

The migration-controller pod runs on this cluster if true.

Optional: If the storage provider is Microsoft Azure, specify the resource group.

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.

SSL verification is enabled if false.

The cluster is validated if true.

The restic pods are restarted on the source cluster after the stage pods are created if true.

Optional: If you are using direct image migration, specify the exposed registry path of a remote cluster.

Specify the URL of the remote cluster.

Specify the name of the Secret CR for the remote cluster.

1.6.3.2.7. MigHook

The MigHook CR defines an Ansible playbook or a custom image that runs tasks at a specified stage of the migration.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigHook
metadata:
  generateName: <hook_name_prefix> 1
  name: <hook_name> 2
  namespace: openshift-migration
spec:
  activeDeadlineSeconds: 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 value 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.
Specify the entire base64-encoded Ansible playbook. Required if custom is false.

Specify source or destination as the cluster on which the hook will run.

1.6.3.2.8. MigMigration

The MigMigration CR runs an associated MigPlan CR.

You can create multiple MigMigration CRs associated with the same MigPlan CR for the following scenarios:

- You can run multiple stage or incremental migrations to copy data without stopping the pods on the source cluster. Running stage migrations improves the performance of the actual migration.
- You can cancel a migration in progress.
- You can roll back a migration.

A migration in progress is canceled if true.

A completed migration is rolled back if true.

Data is copied incrementally and the pods on the source cluster are not stopped if true.

The pods on the source cluster are scaled to 0 after the Backup stage of a migration if true.

The labels and annotations applied during the migration are retained if true.

The status of the migrated pods on the destination cluster are checked and the names of pods that are not in a Running state are returned if true.

migPlanRef.name: Specify the name of the associated MigPlan CR.

1.6.3.2.9. MigPlan
The **MigPlan** CR defines the parameters of a migration plan. It contains a group of virtual machines that are being migrated with the same parameters.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigPlan
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
name: migplan_name
namespace: openshift-migration
spec:
closed: false
srcMigClusterRef:
  name: <source_migcluster_ref>
  namespace: openshift-migration
destMigClusterRef:
  name: <destination_migcluster_ref>
  namespace: openshift-migration
hooks:
  - executionNamespace: <namespace>
    phase: <migration_phase>
    reference:
      name: <mighook_name>
      namespace: <hook_namespace>
      serviceAccount: <service_account>
indirectImageMigration: true
indirectVolumeMigration: false
migStorageRef:
  name: <migstorage_name>
  namespace: openshift-migration
namespaces:
  - <namespace>
refresh: false
```

1. The migration has completed if **true**. You cannot create another **MigMigration** CR for this **MigPlan** CR.

2. Specify the name of the source cluster **MigCluster** CR.

3. Specify the name of the destination cluster **MigCluster** CR.

4. Optional: You can specify up to four migration hooks.

5. Optional: Specify the namespace in which the hook will run.

6. Optional: Specify the migration phase during which a hook runs. One hook can be assigned to one phase. The expected values are **PreBackup**, **PostBackup**, **PreRestore**, and **PostRestore**.

7. Optional: Specify the name of the **MigHook** CR.

8. Optional: Specify the namespace of **MigHook** CR.

9. Optional: Specify a service account with **cluster-admin** privileges.
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.

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.

Specify the name of MigStorage CR.

Specify one or more namespaces.

The MigPlan CR is validated if true.

1.6.3.2.10. MigStorage

The MigStorage CR describes the object storage for the replication repository. You can configure Amazon Web Services, Microsoft Azure, Google Cloud Storage, and generic S3-compatible cloud storage, for example, Minio or NooBaa.

Different providers require different parameters.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigStorage
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
name: migstorage_name
namespace: openshift-migration
spec:
  backupStorageProvider: <storage_provider> 1
  volumeSnapshotProvider: 2
  backupStorageConfig:
    awsBucketName: 3
    awsRegion: 4
    credsSecretRef:
      namespace: openshift-config
      name: <storage_secret> 5
    awsKmsKeyId: 6
    awsPublicUrl: 7
    awsSignatureVersion: 8
  volumeSnapshotConfig:
    awsRegion: 9
    credsSecretRef:
      namespace: openshift-config
      name: 10
  refresh: false 11
```

1. Specify the storage provider.
2. Optional: If you are using the snapshot copy method, specify the storage provider.
3. If you are using AWS, specify the bucket name.
4. If you are using AWS, specify the bucket region, for example, us-east-1.
Specify the name of the **Secret** CR that you created for the **MigStorage** CR.

Optional: If you are using the AWS Key Management Service, specify the unique identifier of the key.

Optional: If you granted public access to the AWS bucket, specify the bucket URL.

Optional: Specify the AWS signature version for authenticating requests to the bucket, for example, 4.

Optional: If you are using the snapshot copy method, specify the geographical region of the clusters.

Optional: If you are using the snapshot copy method, specify the name of the **Secret** CR that you created for the **MigStorage** CR.

The cluster is validated if **true**.

### 1.6.4. Additional resources

- Exposing a secure registry manually on an OpenShift Container Platform 4 cluster
- MTC file system copy method
- MTC snapshot copy method
- Viewing migration custom resources

#### 1.6.5. Configuring a migration plan

You can increase the number of objects to be migrated or exclude resources from the migration.

#### 1.6.5.1. 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
   ...
   mig_controller_limits_cpu: "1"  
mig_controller_limits_memory: "10Gi"
   ```
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.

1.6.5.2. Excluding resources from a migration plan

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 a parameter to exclude specific resources or by adding a resource to the excluded_resources parameter if it does not have its own exclusion parameter:

   apiVersion: migration.openshift.io/v1alpha1
   kind: MigrationController
   metadata:
     name: migration-controller
     namespace: openshift-migration
   spec:
     disable_image_migration: true

Add `disable_image_migration: true` to exclude image streams from the migration. Do not edit the `excluded_resources` parameter. `imagestreams` is added to `excluded_resources` when the MigrationController pod restarts.

Add `disable_pv_migration: true` to exclude PVs from the migration plan. Do not edit the `excluded_resources` parameter. `persistentvolumes` and `persistentvolumeclaims` are added to `excluded_resources` when the MigrationController pod restarts. Disabling PV migration also disables PV discovery when you create the migration plan.

You can add OpenShift Container Platform resources to the `excluded_resources` list. Do not delete the default excluded resources. These resources are problematic to migrate and must be excluded.

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:
     imagestreams,templateinstances,clusterserviceversions,packagemanifests,subscriptions,servicebrokers,servicebindings,serviceclasses,serviceinstances,serviceplans,imagestreams,persistentvolumes,persistentvolumeclaims
   ```

### 1.7. TROUBLESHOOTING

You can view the Migration Toolkit for Containers (MTC) custom resources and download logs to troubleshoot a failed migration.

If the application was stopped during the failed migration, you must roll it back manually to prevent data corruption.
NOTE

Manual rollback is not required if the application was not stopped during migration because the original application is still running on the source cluster.

1.7.1. Viewing MTC custom resources

You can view 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-c9f8-11e9-85e6-5d593ce65610   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: 3b4b543-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
```
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-
1.7.2. 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

   The `-c plain` option displays the log without colors.

1.7.3. Downloading migration logs
You can download the Velero, Restic, and MigrationController pod logs in the Migration Toolkit for Containers (MTC) web console to troubleshoot a failed migration.

**Procedure**

1. In the MTC console, click **Migration plans** to view the list of migration plans.

2. Click the **Options** menu of a specific migration plan and select **Logs**.

3. Click **Download Logs** to download the logs of the MigrationController, Velero, and Restic pods for all clusters. You can download a single log by selecting the cluster, log source, and pod source, and then clicking **Download Selected**.

   You can access a pod log from the CLI by using the `oc logs` command:

```
$ oc logs <pod-name> -f -n openshift-migration
```

   **Specify the pod name.**

### 1.7.4. Updating deprecated APIs

If your source cluster uses deprecated APIs, the following warning message is displayed when you create a migration plan in the Migration Toolkit for Containers (MTC) web console:

```
Some namespaces contain GVKs incompatible with destination cluster
```

You can click **See details** to view the namespace and the incompatible APIs. This warning message does not block the migration.

During migration with the Migration Toolkit for Containers (MTC), the deprecated APIs are saved in the Velero Backup #1 for Kubernetes objects. You can download the Velero Backup, extract the deprecated API `yaml` files, and update them with the `oc convert` command. Then you can create the updated APIs on the target cluster.

**Procedure**

1. Run the migration plan.

2. View the **MigPlan** custom resource (CR):

```
$ oc describe migplan <migplan_name> -n openshift-migration
```

   **Specify the name of the MigPlan CR.**

   The output is similar to the following:

```
metadata:
  ...  
  uid: 79509e05-61d6-11e9-bc55-02ce4781844a
```
status:
  ... conditions:
  - category: Warn
    lastTransitionTime: 2020-04-30T17:16:23Z
    message: 'Some namespaces contain GVKs incompatible with destination cluster.
    See: `incompatibleNamespaces` for details'
    status: "True"
    type: GVKsIncompatible
    incompatibleNamespaces:
      - gvk:
        group: batch
        kind: cronjobs
        version: v2alpha1
      - gvk:
        group: batch
        kind: scheduledjobs
        version: v2alpha1

1. Record the **MigPlan** CR UID.

2. Record the deprecated APIs listed in the **gvks** section.

3. Get the **MigMigration** name associated with the **MigPlan** UID:

   $ oc get migmigration -o json | jq -r '.items[] | select(.metadata.ownerReferences[].uid=="<migplan_uid>") | .metadata.name'

1. Specify the **MigPlan** CR UID.

4. Get the **MigMigration** UID associated with the **MigMigration** name:

   $ oc get migmigration <migmigration_name> -o jsonpath='{.metadata.uid}'

1. Specify the **MigMigration** name.

5. Get the **Velero** Backup name associated with the **MigMigration** UID:

   $ oc get backup.velero.io --selector migration-initial-backup="<migmigration_uid>" -o jsonpath={.items[*].metadata.name}

1. Specify the **MigMigration** UID.

6. Download the contents of the **Velero** Backup to your local machine by running the command for your storage provider:

   - **AWS S3:**
     
     $ aws s3 cp s3://<bucket_name>/velero/backups/<backup_name> <backup_local_dir> --recursive

1. Specify the bucket, backup name, and your local backup directory name.
1. Specify the bucket, backup name, and your local backup directory name.

2. Specify the backup name and your local backup directory name.

7. Extract the Velero Backup archive file:

   $ tar -xfv <backup_local_dir>/<backup_name>.tar.gz -C <backup_local_dir>

8. Run `oc convert` in offline mode on each deprecated API:

   $ oc convert -f <backup_local_dir>/resources/<gvk>.json

9. Create the converted API on the target cluster:

   $ oc create -f <gvk>.json

### 1.7.5. 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.

#### 1.7.5.1. CA certificate error in the MTC console

If the MTC console displays a **CA certificate error** message the first time you try to access it, the likely cause is that a cluster uses self-signed CA certificates.

Navigate to the `oauth-authorization-server` URL in the error message and accept the certificate. To resolve this issue permanently, install the certificate authority so that it is trusted.

If the browser displays an **Unauthorized** message after you have accepted the CA certificate, navigate to the MTC console and then refresh the web page.

#### 1.7.5.2. OAuth timeout error in the MTC console

If the MTC console displays a **connection has timed out** message after you have accepted a self-signed certificate, the cause is likely to be one of the following:

- Interrupted network access to the OAuth server
- Interrupted network access to the OpenShift Container Platform console
- Proxy configuration blocking access to the OAuth server. See MTC console inaccessible because of OAuth timeout error for details.

To determine the cause:

- Inspect the MTC console web page with a browser web inspector.
- Check the Migration UI pod log for errors.

1.7.5.3. 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:

BSL error messages

Error checking repository for stale locks

Error getting backup storage location: backupstoragelocation.velero.io "my-bsl" not found

You can ignore these error messages. A missing BSL cannot cause a migration to fail.

1.7.5.4. Pod volume backup timeout error in the Velero pod log

If a migration fails because Restic times out, the Velero pod log displays the following error:

Pod volume backup timeout error

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

   Valid units are h (hours), m (minutes), and s (seconds), for example, 3h30m15s.

5. Click Save.
1.7.5.5. 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 MigMigration CR displays the following error:

MigMigration CR status

```
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
    status: "True"
    type: ResticVerifyErrors
```

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 troubleshoot 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.

**Restore CR with pod volume restore error**

```
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:
The output identifies the Restic pod that logged the errors.

**PodVolumeRestore CR with Restic pod error**

```
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>
```

### 1.7.5.6. 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 Restic pod log displays the following error:

**Restic permission error**

```
```

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:

   ```yaml
   spec:
     restic_supplemental_groups: <group_id>  # Specify the supplemental group ID.
   ```

4. Wait for the Restic pods to restart so that the changes are applied.

### 1.7.6. Direct volume migration does not complete

If direct volume migration does not complete, the target cluster might not have the same node-selector.
If direct volume migration does not complete, the target cluster might not have the same node-selector annotations as the source cluster.

Migration Toolkit for Containers (MTC) migrates namespaces with all annotations to preserve security context constraints and scheduling requirements. During direct volume migration, MTC creates Rsync transfer pods on the target cluster in the namespaces that were migrated from the source cluster. If a target cluster namespace does not have the same annotations as the source cluster namespace, the Rsync transfer pods cannot be scheduled. The Rsync pods remain in a Pending state.

You can identify and fix this issue by performing the following procedure.

**Procedure**

1. Check the status of the MigMigration CR:
   
   ```bash
   $ oc describe migmigration <pod_name> -n openshift-migration
   ```

   The output includes the following status message:

   **Example output**

   ```
   ... Some or all transfer pods are not running for more than 10 mins on destination cluster ...
   ```

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

   ```bash
   $ oc get namespace <namespace> -o yaml
   $ oc edit namespace <namespace>
   ```

   1. Specify the migrated namespace.

3. On the target cluster, edit the migrated namespace:

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

4. Add missing openshift.io/node-selector annotations to the migrated namespace as in the following example:

   ```yaml
   apiVersion: v1
   kind: Namespace
   metadata:
     annotations:
       openshift.io/node-selector: "region=east"
   ...
   ```

5. Run the migration plan again.

### 1.7.7. Using the Velero CLI to debug Backup and Restore CRs

You can debug the Backup and Restore custom resources (CRs) and partial migration failures with the Velero command line interface (CLI). The Velero CLI runs in the velero pod.
1.7.7.1. Velero command syntax

Velero CLI commands use the following syntax:

```
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -- ./velero <resource> <command> <resource_id>
```

You can specify `velero-<pod>` -n `openshift-migration` in place of $(oc get pods -n openshift-migration -o name | grep velero).

1.7.7.2. Help command

The Velero `help` command lists all the Velero CLI commands:

```
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -- ./velero --help
```

1.7.7.3. Describe command

The Velero `describe` command provides a summary of warnings and errors associated with a Velero resource:

```
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -- ./velero <resource> describe <resource_id>
```

Example

```
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -- ./velero backup describe 0e44ae00-5dc3-11eb-9ca8-df7e5254778b-2d8ql
```

1.7.7.4. Logs command

The Velero `logs` command provides the logs associated with a Velero resource:

```
velero <resource> logs <resource_id>
```

Example

```
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -- ./velero restore logs ccc7c2d0-6017-11eb-afab-85d0007f5a19-x4lbf
```

1.7.7.5. 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**

   ```yaml
   status:
   conditions:
   - category: Warn
     durable: true
     lastTransitionTime: "2021-01-26T20:48:40Z"
     message: 'Final Restore openshift-migration/ccc7c2d0-6017-11eb-afab-85d0007f5a19-x4lb: 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 exec $(oc get pods -n openshift-migration -o name | grep velero) -n openshift-migration -
   - ./velero restore describe <restore>

   **Example output**

   Phase: PartiallyFailed (run `velero restore logs ccc7c2d0-6017-11eb-afab-85d0007f5a19-x4lb` 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 exec $(oc get pods -n openshift-migration -o name | grep velero) -n openshift-migration -
   - ./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
The Restore CR log error message, the server could not find the requested resource, indicates the cause of the partially failed migration.

1.7.8. Using must-gather to collect data

You must run the must-gather tool if you open a customer support case on the Red Hat Customer Portal for the Migration Toolkit for Containers (MTC).

The openshift-migration-must-gather-rhel8 image for MTC collects migration-specific logs and data that are not collected by the default must-gather image.

Procedure

1. Navigate to the directory where you want to store the must-gather data.

2. Run the must-gather command:

   $ oc adm must-gather --image=registry.redhat.io/rhmtc/openshift-migration-must-gather-rhel8:v1.4

3. Remove authentication keys and other sensitive information.

4. Create an archive file containing the contents of the must-gather data directory:

   $ tar cvaf must-gather.tar.gz must-gather.local.<uid>/

5. Upload the compressed file as an attachment to your customer support case.

1.7.9. Rolling back a migration

You can roll back a migration by using the MTC web console or the CLI.

1.7.9.1. Rolling back a migration in the MTC web console

You can roll back a migration by using the Migration Toolkit for Containers (MTC) web console.

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.
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.

1.7.9.1.1. Rolling back a migration from the CLI

You can roll back a migration by creating a MigMigration custom resource (CR) from the CLI.

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:

   ```bash
   $ cat << EOF | oc apply -f -
   apiVersion: migration.openshift.io/v1alpha1
   kind: MigMigration
   metadata:
     labels:
       controller-tools.k8s.io: "1.0"
   name: migration-rollback
   namespace: openshift-migration
   spec:
     ...  
     rollback: true
     ...
     migPlanRef:
       name: <migplan_name>  
       namespace: openshift-migration
   EOF
   
   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.

1.7.10. 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:

  ```yaml
  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)

1.7.11. Additional resources

- MTC workflow
- MTC custom resources
CHAPTER 2. MIGRATING FROM OPENSIGHT CONTAINER PLATFORM 4.1

2.1. ABOUT THE MIGRATION TOOLKIT FOR CONTAINERS

You can migrate application workloads from OpenShift Container Platform 4.1 to 4.7 with the Migration Toolkit for Containers (MTC). MTC enables you to control the migration and to minimize application downtime.

NOTE

You can migrate between OpenShift Container Platform clusters of the same version, for example, from 4.1 to 4.1, as long as the source and target clusters are configured correctly.

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

The MTC web console and API, based on Kubernetes custom resources, enable you to migrate stateful and stateless application workloads at the granularity of a namespace.

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.

You can use migration hooks to run Ansible playbooks at certain points during the migration. The hooks are added when you create a migration plan.

2.1.1. Migration Toolkit for Containers workflow

You use the Migration Toolkit for Containers (MTC) to migrate Kubernetes resources, persistent volume data, and internal container images from an OpenShift Container Platform source cluster to an OpenShift Container Platform 4.7 target cluster by using the MTC web console or the Kubernetes API.

The (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 volume claims that are linked to the persistent volumes of the namespace.
- Custom resources (CRs) and custom resource definitions (CRDs): The MTC automatically migrates any CRs that exist at the namespace level as well as the CRDs that are linked to those CRs.

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. In a restricted environment, you can use an internally hosted S3 storage repository. 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.

     ![Diagram](image.png)

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

     - **Move**: MTC unmounts a remote volume, for example, NFS, from the source cluster, creates a PV resource on the target cluster pointing to the remote volume, and then mounts the remote volume on the target cluster. Applications running on the target cluster use the same remote volume that the source cluster was using. The remote volume must be accessible to the source and target clusters.

     **NOTE**
     Although the replication repository does not appear in this diagram, it is required for migration.
6. Run the migration plan, with one of the following options:

- **Stage** (optional) copies data to the target cluster without stopping the application. Staging can be run multiple times so that most of the data is copied to the target before migration. This minimizes the duration of the migration and application downtime.

- **Migrate** stops the application on the source cluster and recreates its resources on the target cluster. Optionally, you can migrate the workload without stopping the application.

2.1.2. Migration Toolkit for Containers custom resources

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

1. **MigCluster** (configuration, MTC cluster): Cluster definition
2. **MigStorage** (configuration, MTC cluster): Storage definition
3. **MigPlan** (configuration, MTC cluster): Migration plan

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

**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

### 2.1.3. 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.

#### 2.1.3.1. File system copy method

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

**Table 2.1. File system copy method summary**

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clusters can have different storage classes.</td>
<td>Slower than the snapshot copy method.</td>
</tr>
<tr>
<td>Supported for all S3 storage providers.</td>
<td>Optional data verification significantly reduces performance.</td>
</tr>
<tr>
<td>Optional data verification with checksum.</td>
<td></td>
</tr>
<tr>
<td>Supports direct volume migration, which significantly increases performance.</td>
<td></td>
</tr>
</tbody>
</table>

#### 2.1.3.2. 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.

AWS, Google Cloud Provider, and Microsoft Azure support the snapshot copy method.

**Table 2.2. Snapshot copy method summary**

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Benefits</td>
<td>Limitations</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>• Faster than the file system copy method.</td>
<td>• Cloud provider must support snapshots.</td>
</tr>
<tr>
<td></td>
<td>• Clusters must be on the same cloud provider.</td>
</tr>
<tr>
<td></td>
<td>• Clusters must be in the same location or region.</td>
</tr>
<tr>
<td></td>
<td>• Clusters must have the same storage class.</td>
</tr>
<tr>
<td></td>
<td>• Storage class must be compatible with snapshots.</td>
</tr>
<tr>
<td></td>
<td>• Does not support direct volume migration.</td>
</tr>
</tbody>
</table>

2.1.3.3. Direct volume migration and direct image migration

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

Direct migration has significant performance benefits because it skips 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.

Direct migration uses Rsync to transfer the files.

NOTE

Direct image migration and direct volume migration have additional prerequisites.

2.1.4. About migration hooks

You can use migration hooks to run custom code at certain points during a migration with the Migration Toolkit for Containers (MTC). 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 using an Ansible playbook or 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 (CR). 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-rhel7:1.4. This image is based on the Ansible Runner image and includes python-openshift for Ansible Kubernetes resources and an updated oc binary.

Optional: You can use a custom Ansible runtime image containing additional Ansible modules or tools instead of the default image.

Custom hook container

You can create a custom hook container that includes Ansible playbooks or custom code.

### 2.2. INSTALLING AND UPGRADING THE MIGRATION TOOLKIT FOR CONTAINERS

You can install the Migration Toolkit for Containers on an OpenShift Container Platform 4.7 target cluster and on a 4.1 source cluster.

MTC is installed on the target cluster by default. You can install the MTC on an OpenShift Container Platform 3 cluster or on a remote cluster.

#### 2.2.1. Installing the Migration Toolkit for Containers in a connected environment

You can install the Migration Toolkit for Containers (MTC) in a connected environment.

**IMPORTANT**

You must install the same MTC version on all clusters.

#### 2.2.1.1. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.7 target cluster

You can install the Migration Toolkit for Containers (MTC) on an OpenShift Container Platform 4.7 target cluster.

**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.
NOTE

Do not change the subscription approval option to Automatic. The Migration Toolkit for Containers version must be the same on the source and the target clusters.

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.

2.2.1.2. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.1 source cluster

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

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 Catalog → 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.

   NOTE

   Do not change the subscription approval option to Automatic. The Migration Toolkit for Containers version must be the same on the source and the target clusters.

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. Update the following parameters in the migration-controller custom resource manifest:

   ```yaml
   spec:
   ...
   migration_controller: false
   ```
Add the `deprecated_cors_configuration` parameter and its value.

8. Click **Create**.

9. Click **Workloads → Pods** to verify that the MTC pods are running.

### 2.2.2. Installing the Migration Toolkit for Containers in a restricted environment

You can install the Migration Toolkit for Containers (MTC) in a restricted environment.

**IMPORTANT**

You must install the same MTC version on all clusters.

You can build a custom Operator catalog image for OpenShift Container Platform 4, push it to a local mirror image registry, and configure Operator Lifecycle Manager (OLM) to install the Migration Toolkit for Containers Operator from the local registry.

#### 2.2.2.1. Disabling the default OperatorHub sources

Operator catalogs that source content provided by Red Hat and community projects are configured for OperatorHub by default during an OpenShift Container Platform installation.

**Procedure**

- Disable the sources for the default catalogs by adding `disableAllDefaultSources: true` to the `OperatorHub` object:

  ```bash
  $ oc patch OperatorHub cluster --type json \
  -p '[$("op": "add", "path": "/spec/disableAllDefaultSources", "value": true)]'
  ```

**TIP**

Alternatively, you can use the web console to manage catalog sources. From the **Administration → Cluster Settings → Global Configuration → OperatorHub** page, click the **Sources** tab, where you can create, delete, disable, and enable individual sources.

#### 2.2.2.2. Pruning an index image

An index image, based on the Operator Bundle Format, is a containerized snapshot of an Operator catalog. You can prune an index of all but a specified list of packages, which creates a copy of the source index containing only the Operators that you want.

When configuring Operator Lifecycle Manager (OLM) to use mirrored content on restricted network OpenShift Container Platform clusters, use this pruning method if you want to only mirror a subset of Operators from the default catalogs.
For the steps in this procedure, the target registry is an existing mirror registry that is accessible by your workstation with unrestricted network access. This example also shows pruning the index image for the default redhat-operators catalog, but the process is the same for any index image.

Prerequisites

- Workstation with unrestricted network access
- podman version 1.9.3+
- grpcurl
- opm version 1.12.3+
- Access to a registry that supports Docker v2-2

Procedure

1. Authenticate with registry.redhat.io:
   
   ```
   $ podman login registry.redhat.io
   ```

2. Authenticate with your target registry:
   
   ```
   $ podman login <target_registry>
   ```

3. Determine the list of packages you want to include in your pruned index.
   a. Run the source index image that you want to prune in a container. For example:

   ```
   ```

   Example output

   ```
   Trying to pull registry.redhat.io/redhat/redhat-operator-index:v4.7...
   Getting image source signatures
   Copying blob ae8a0c23f5b1 done
   INFO[0000] serving registry      database=/database/index.db port=50051
   ```

   b. In a separate terminal session, use the grpcurl command to get a list of the packages provided by the index:

   ```
   $ grpcurl -plaintext localhost:50051 api.Registry/ListPackages > packages.out
   ```

   c. Inspect the packages.out file and identify which package names from this list you want to keep in your pruned index. For example:

   **Example snippets of packages list**

   ```
   ...{
       "name": "advanced-cluster-management"
   }
   ```
d. In the terminal session where you executed the `podman run` command, press `Ctrl` and `C` to stop the container process.

4. Run the following command to prune the source index of all but the specified packages:

   ```bash
   $ opm index prune \
   -f registry.redhat.io/redhat/redhat-operator-index:v4.7 \1
   -p advanced-cluster-management,jaeger-product,quay-operator \2
   [-i registry.redhat.io/openshift4/ose-operator-registry:v4.7] \3
   -t <target_registry>:<port>/<namespace>/redhat-operator-index:v4.7 \4
   
   1 Index to prune.
   2 Comma-separated list of packages to keep.
   3 Required only for IBM Power Systems and IBM Z images: Operator Registry base image with the tag that matches the target OpenShift Container Platform cluster major and minor version.
   4 Custom tag for new index image being built.
   
5. Run the following command to push the new index image to your target registry:

   ```bash
   $ podman push <target_registry>:<port>/<namespace>/redhat-operator-index:v4.7
   
   where `<namespace>` is any existing namespace on the registry. For example, you might create an `olm-mirror` namespace to push all mirrored content to.

2.2.2.3. Mirroring an Operator catalog

You can mirror the Operator content of a Red Hat-provided catalog, or a custom catalog, into a container image registry using the `oc adm catalog mirror` command. The target registry must support Docker v2-2. For a cluster on a restricted network, this registry can be one that the cluster has network access to, such as a mirror registry created during a restricted network cluster installation.

The `oc adm catalog mirror` command also automatically mirrors the index image that is specified during the mirroring process, whether it be a Red Hat-provided index image or your own custom-built index image, to the target registry. You can then use the mirrored index image to create a catalog source that allows Operator Lifecycle Manager (OLM) to load the mirrored catalog onto your OpenShift Container Platform cluster.

Prerequisites
- Workstation with unrestricted network access.

- **podman** version 1.9.3 or later.

- Access to mirror registry that supports Docker v2-2.

- Decide which namespace on your mirror registry you will use to store the mirrored Operator content. For example, you might create an `olm-mirror` namespace.

- If your mirror registry does not have Internet access, connect removable media to your workstation with unrestricted network access.

- If you are working with private registries, set the `REG_CREDS` environment variable to the file path of your registry credentials for use in later steps. For example, for the **podman** CLI:

  \[
  \$ \text{REG\_CREDS} = \$(\text{XDG\_RUNTIME\_DIR})/\text{containers/auth.json}
  \]

**Procedure**

1. If you want to mirror a Red Hat-provided catalog, run the following command on your workstation with unrestricted network access to authenticate with `registry.redhat.io`:

  \[
  \$ \text{podman login registry.redhat.io}
  \]

2. The `oc adm catalog mirror` command extracts the contents of an index image to generate the manifests required for mirroring. The default behavior of the command generates manifests, then automatically mirrors all of the image content from the index image, as well as the index image itself, to your mirror registry. Alternatively, if your mirror registry is on a completely disconnected, or `airgapped`, host, you can first mirror the content to removable media, move the media to the disconnected environment, then mirror the content from the media to the registry.

   - **Option A:** If your mirror registry is on the same network as your workstation with unrestricted network access, take the following actions on your workstation:

     a. If your mirror registry requires authentication, run the following command to log in to the registry:

     \[
     \$ \text{podman login <mirror\_registry>}
     \]

     b. Run the following command to mirror the content:

     \[
     \$ \text{oc adm catalog mirror} \\
     \text{<index\_image>} \text{<mirror\_registry>:<port>/<namespace>} \text{-a \$(REG\_CREDS)} \text{--insecure} \text{--manifests-only}
     \]

     Specify the index image for the catalog you want to mirror. For example, this might be a pruned index image that you created previously, or one of the source index images for the default catalogs, such as `registry.redhat.io/redhat/redhat-operator-index:v4.7`.
Specify the target registry and namespace to mirror the Operator content to, where `<namespace>` is any existing namespace on the registry. For example, you might create a `olm-mirror` namespace to push all mirrored content to.

Optional: If required, specify the location of your registry credentials file.

Optional: If you do not want to configure trust for the target registry, add the `--insecure` flag.

Optional: Specify which platform and architecture of the index image to select when multiple variants are available. Images are passed as `<platform>/<arch>[/<variant>]`. This does not apply to images referenced by the index. Valid values are `linux/amd64`, `linux/ppc64le`, and `linux/s390x`.

Optional: Generate only the manifests required for mirroring, and do not actually mirror the image content to a registry. This option can be useful for reviewing what will be mirrored, and it allows you to make any changes to the mapping list if you require only a subset of packages. You can then use the `mapping.txt` file with the `oc image mirror` command to mirror the modified list of images in a later step. This flag is intended for only advanced selective mirroring of content from the catalog; the `opm index prune` command, if you used it previously to prune the index image, is suitable for most catalog management use cases.

Example output

```
src image has index label for database path: /database/index.db
using database path mapping: /database/index.db:/tmp/153048078
wrote database to /tmp/153048078

... 1
wrote mirroring manifests to manifests-redhat-operator-index-1614211642 2
```

1. Directory for the temporary `index.db` database generated by the command.

2. Record the manifests directory name that is generated. This directory name is used in a later step.

- Option B: If your mirror registry is on a disconnected host, take the following actions.

  a. Run the following command on your workstation with unrestricted network access to mirror the content to local files:

```
$ oc adm catalog mirror \
  <index_image> 1 \n  file:///local/index 2 \
  [-a ${REG_CREDS}] \n  [--insecure]
```

1. Specify the index image for the catalog you want to mirror. For example, this might be a pruned index image that you created previously, or one of the source index images for the default catalogs, such as `registry.redhat.io/redhat/redhat-operator-index:v4.7`.

2. Mirrors content to local files in your current directory.
Example output

```
...  
info: Mirroring completed in 5.93s (5.915MB/s)  
wrote mirroring manifests to manifests-my-index-1614985528  
```

To upload local images to a registry, run:

```
oc adm catalog mirror file://local/index/myrepo/my-index:v1  
REGISTRY/REPOSITORY
```

1. Record the manifests directory name that is generated. This directory name is used in a later step.

2. Record the expanded file:// path that based on your provided index image. This path is used in a later step.

b. Copy the v2/ directory that is generated in your current directory to removable media.

c. Physically remove the media and attach it to a host in the disconnected environment that has access to the mirror registry.

d. If your mirror registry requires authentication, run the following command on your host in the disconnected environment to log in to the registry:

```
$ podman login <mirror_registry>
```

e. Run the following command from the parent directory containing the v2/ directory to upload the images from local files to the mirror registry:

```
$ oc adm catalog mirror \
file://local/index/<repo>/<index_image>:<tag> \
<mirror_registry>:<port>/<namespace> \n[-a ${REG_CREDS}] \n[--insecure]
```

1. Specify the file:// path from the previous command output.

2. Specify the target registry and namespace to mirror the Operator content to, where <namespace> is any existing namespace on the registry. For example, you might create an olm-mirror namespace to push all mirrored content to.

3. After mirroring the content to your registry, inspect the manifests directory that is generated in your current directory.

   **NOTE**

   The manifests directory name is used in a later step.

If you mirrored content to a registry on the same network in the previous step, the directory name takes the following form:
manifests-<index_image_name>-<random_number>

If you mirrored content to a registry on a disconnected host in the previous step, the directory name takes the following form:

manifests-index/<namespace>/<index_image_name>-<random_number>

The manifests directory contains the following files, some of which might require further modification:

- The `catalogSource.yaml` file is a basic definition for a `CatalogSource` object that is pre-populated with your index image tag and other relevant metadata. This file can be used as is or modified to add the catalog source to your cluster.

  **IMPORTANT**

  If you mirrored the content to local files, you must modify your `catalogSource.yaml` file to remove any backslash (`/`) characters from the `metadata.name` field. Otherwise, when you attempt to create the object, it fails with an “invalid resource name” error.

- The `imageContentSourcePolicy.yaml` file defines an `ImageContentSourcePolicy` object that can configure nodes to translate between the image references stored in Operator manifests and the mirrored registry.

  **NOTE**

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

- The `mapping.txt` file contains all of the source images and where to map them in the target registry. This file is compatible with the `oc image mirror` command and can be used to further customize the mirroring configuration.

  **IMPORTANT**

  If you used the `--manifests-only` flag during the mirroring process and want to further trim the subset of packages to be mirrored, see the steps in the “Mirroring a Package Manifest Format catalog image” procedure about modifying your `mapping.txt` file and using the file with the `oc image mirror` command. After following those further actions, you can continue this procedure.

4. On a host with access to the disconnected cluster, create the `ImageContentSourcePolicy` object by running the following command to specify the `imageContentSourcePolicy.yaml` file in your manifests directory:

```
$ oc create -f <path/to/manifests/dir>/imageContentSourcePolicy.yaml
```

where `<path/to/manifests/dir>` is the path to the manifests directory for your mirrored content.
You can now create a `CatalogSource` object to reference your mirrored index image and Operator content.

### 2.2.2.4. Creating a catalog from an index image

You can create an Operator catalog from an index image and apply it to an OpenShift Container Platform cluster for use with Operator Lifecycle Manager (OLM).

**Prerequisites**

- An index image built and pushed to a registry.

**Procedure**

1. Create a `CatalogSource` object that references your index image.
   
   a. Modify the following to your specifications and save it as a `catalogSource.yaml` file:

   ```yaml
   apiVersion: operators.coreos.com/v1alpha1
   kind: CatalogSource
   metadata:
     name: my-operator-catalog
     namespace: openshift-marketplace
   spec:
     sourceType: grpc
     displayName: My Operator Catalog
     publisher: <publisher_name>
     updateStrategy:
       registryPoll: <>
       interval: 30m
   ```

   <-> Specify your index image. <-> Specify your name or an organization name publishing the catalog. <-> Catalog sources can automatically check for new versions to keep up to date.

   b. Use the file to create the `CatalogSource` object:

   ```bash
   $ oc apply -f catalogSource.yaml
   ```

2. Verify the following resources are created successfully.

   a. Check the pods:

   ```bash
   $ oc get pods -n openshift-marketplace
   ```

   **Example output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>my-operator-catalog-6njx6</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>28s</td>
</tr>
<tr>
<td>marketplace-operator-d9f549946-96sgr</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>26h</td>
</tr>
</tbody>
</table>

   b. Check the catalog source:

   ```bash
   $ oc get catalogsource -n openshift-marketplace
   ```
Example output

<table>
<thead>
<tr>
<th>NAME</th>
<th>DISPLAY</th>
<th>TYPE</th>
<th>PUBLISHER</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>my-operator-catalog</td>
<td>My Operator Catalog</td>
<td>grpc</td>
<td></td>
<td>5s</td>
</tr>
</tbody>
</table>

c. Check the package manifest:

```
$ oc get packagemanifest -n openshift-marketplace
```

Example output

<table>
<thead>
<tr>
<th>NAME</th>
<th>CATALOG</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>jaeger-product</td>
<td>My Operator Catalog</td>
<td>93s</td>
</tr>
</tbody>
</table>

You can now install the Operators from the OperatorHub page on your OpenShift Container Platform web console.

2.2.2.5. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.7 target cluster in a restricted environment

You can install the Migration Toolkit for Containers (MTC) on an OpenShift Container Platform 4.7 target cluster.

**Prerequisites**

- You must be logged in as a user with `cluster-admin` privileges on all clusters.
- You must create a custom Operator catalog and push it to a mirror registry.
- You must configure Operator Lifecycle Manager to install the Migration Toolkit for Containers Operator from the mirror 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.

   **NOTE**

   Do not change the subscription approval option to Automatic. The Migration Toolkit for Containers version must be the same on the source and the target clusters.

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.

2.2.2.6. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.1 source cluster in a restricted environment

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

Prerequisites

- You must be logged in as a user with cluster-admin privileges on all clusters.
- You must create a custom Operator catalog and push it to a mirror registry.
- You must configure Operator Lifecycle Manager to install the Migration Toolkit for Containers Operator from the mirror registry.

Procedure

1. Use the Filter by keyword field to find the Migration Toolkit for Containers Operator.

2. Select the Migration Toolkit for Containers Operator and click Install.

   NOTE

   Do not change the subscription approval option to Automatic. The Migration Toolkit for Containers version must be the same on the source and the target clusters.

3. Click Install.

   On the Installed Operators page, the Migration Toolkit for Containers Operator appears in the openshift-migration project with the status Succeeded.

4. Click Migration Toolkit for Containers Operator.

5. Under Provided APIs, locate the Migration Controller tile, and click Create Instance.

6. Click Create.

7. Click Workloads → Pods to verify that the MTC pods are running.

2.2.3. Upgrading the Migration Toolkit for Containers

You can upgrade the Migration Toolkit for Containers (MTC) by using the OpenShift Container Platform web console.

IMPORTANT

You must ensure that the same MTC version is installed on all clusters.

If you are upgrading MTC version 1.3, you must perform an additional procedure to update the MigPlan custom resource (CR).
2.2.3.1. Upgrading the Migration Toolkit for Containers on an OpenShift Container Platform 4 cluster

You can upgrade the Migration Toolkit for Containers (MTC) on an OpenShift Container Platform 4 cluster by using the OpenShift Container Platform web console.

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 Migration Toolkit for Containers Operator.

8. Under Provided APIs, locate the Migration Controller tile, and click Create Instance.

9. If you are upgrading MTC on a source cluster, update the following parameters in the MigrationController custom resource (CR) manifest:

   ```yaml
   spec:
   ...
   migration_controller: false
   migration_ui: false
   ...
   deprecated_cors_configuration: true
   ```

   You do not need to update the MigrationController CR manifest on the target cluster.

10. Click Create.

11. Click Workloads → Pods to verify that the MTC pods are running.

2.2.3.2. Upgrading MTC 1.3 to 1.4

If you are upgrading Migration Toolkit for Containers (MTC) version 1.3.x to 1.4, 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 have MTC 1.3 installed.
- You must be logged in as a user with `cluster-admin` privileges.

**Procedure**

1. Log in to the cluster on which the **MigrationController** pod is running.

2. Get the **MigPlan** CR manifest:

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

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

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

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

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

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

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

### 2.3. CONFIGURING OBJECT STORAGE FOR 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:

- **Multi-Cloud Object Gateway (MCG)**
- **Amazon Web Services (AWS) S3**
- **Google Cloud Provider (GCP)**
- **Microsoft Azure**
- **Generic S3 object storage**, for example, Minio or Ceph S3
In a restricted environment, you can create an internally hosted replication repository.

**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.

### 2.3.1. Configuring a Multi-Cloud Object Gateway storage bucket as a replication repository

You can install the OpenShift Container Storage Operator and configure a Multi-Cloud Object Gateway (MCG) storage bucket as a replication repository for the Migration Toolkit for Containers (MTC).

#### 2.3.1.1. Installing the OpenShift Container Storage Operator

You can install the OpenShift Container Storage Operator from OperatorHub.

**Procedure**

1. In the OpenShift Container Platform web console, click **Operators** → **OperatorHub**.
2. Use **Filter by keyword** (in this case, **OCS**) to find the **OpenShift Container Storage Operator**.
3. Select the **OpenShift Container Storage Operator** and click **Install**.
4. Select an **Update Channel**, **Installation Mode**, and **Approval Strategy**.
5. Click **Install**.
   - On the **Installed Operators** page, the **OpenShift Container Storage Operator** appears in the **openshift-storage** project with the status **Succeeded**.

#### 2.3.1.2. Creating the Multi-Cloud Object Gateway storage bucket

You can create the Multi-Cloud Object Gateway (MCG) storage bucket’s custom resources (CRs).

**Procedure**

1. Log in to the OpenShift Container Platform cluster:
   
   ```bash
   $ oc login -u <username>
   ```

2. Create the **NooBaa** CR configuration file, **noobaa.yml**, with the following content:

   ```yaml
   apiVersion: noobaa.io/v1alpha1
   kind: NooBaa
   metadata:
     name: noobaa
     namespace: openshift-storage
   spec:
     dbResources:
       requests:
         cpu: 0.5
   ```
For a very small cluster, you can change the cpu value to 0.1.

3. Create the NooBaa object:

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

4. Create the BackingStore CR configuration file, bs.yml, with the following content:

```
apiVersion: noobaa.io/v1alpha1
kind: BackingStore
metadata:
  finalizers:
    - noobaa.io/finalizer
  labels:
    app: noobaa
  name: mcg-pv-pool-bs
  namespace: openshift-storage
spec:
  pvPool:
    numVolumes: 3
    resources:
      requests:
        storage: 50Gi
        storageClass: gp2
      type: pv-pool
```

1. Specify the number of volumes in the persistent volume pool.
2. Specify the size of the volumes.
3. Specify the storage class.

5. Create the BackingStore object:

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

6. Create the BucketClass CR configuration file, bc.yml, with the following content:

```
apiVersion: noobaa.io/v1alpha1
kind: BucketClass
metadata:
  labels:
    app: noobaa
  name: mcg-pv-pool-bc
  namespace: openshift-storage
spec:
  placementPolicy:
```

1 2 3
7. Create the **BucketClass** object:

   $ oc create -f bc.yml

8. Create the **ObjectBucketClaim** CR configuration file, `obc.yml`, with the following content:

   ```yaml
   apiVersion: objectbucket.io/v1alpha1
   kind: ObjectBucketClaim
   metadata:
     name: migstorage
     namespace: openshift-storage
   spec:
     bucketName: migstorage
     storageClassName: openshift-storage.noobaa.io
     additionalConfig:
       bucketclass: mcg-pv-pool-bc
   ```

   ![Record the bucket name for adding the replication repository to the MTC web console.](image)

9. Create the **ObjectBucketClaim** object:

   $ oc create -f obc.yml

10. Watch the resource creation process to verify that the **ObjectBucketClaim** status is **Bound**:

    ```bash
    $ watch -n 30 'oc get -n openshift-storage objectbucketclaim migstorage -o yaml'
    ```

    This process can take five to ten minutes.

11. Obtain and record the following values, which are required when you add the replication repository to the MTC web console:

    - S3 endpoint:
      
      ```bash
      $ oc get route -n openshift-storage s3
      ```

    - S3 provider access key:
      
      ```bash
      $ oc get secret -n openshift-storage migstorage -o go-template="{{ .data.AWS_ACCESS_KEY_ID }}" | base64 --decode
      ```

    - S3 provider secret access key:
      
      ```bash
      $ oc get secret -n openshift-storage migstorage -o go-template="{{ .data.AWS_SECRET_ACCESS_KEY }}" | base64 --decode
      ```

2.3.2. Configuring an AWS S3 storage bucket as a replication repository
You can configure an AWS S3 storage bucket as a replication repository for the Migration Toolkit for Containers (MTC).

**Prerequisites**

- The AWS S3 storage bucket must be accessible to the source and target clusters.
- You must have the AWS CLI installed.
- 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. Create an AWS S3 bucket:
   
   ```
   $ aws s3api create-bucket
   --bucket <bucket_name> 
   --region <bucket_region>
   ```

   1. Specify your S3 bucket name.
   2. Specify your S3 bucket region, for example, `us-east-1`.

2. Create the IAM user `velero`:
   
   ```
   $ aws iam create-user --user-name velero
   ```

3. Create an EC2 EBS snapshot policy:
   
   ```
   $ cat > velero-ec2-snapshot-policy.json <<EOF
   {
   "Version": "2012-10-17",
   "Statement": [
   {
   "Effect": "Allow",
   "Action": [
   "ec2:DescribeVolumes",
   "ec2:DescribeSnapshots",
   "ec2:CreateTags",
   "ec2:CreateVolume",
   "ec2:CreateVolume",
   "ec2:CreateSnapshot",
   "ec2:DeleteSnapshot"
   ],
   "Resource": "*"
   }
   }
   ```
4. Create an AWS S3 access policy for one or for all S3 buckets:

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

To grant access to a single S3 bucket, specify the bucket name. To grant access to all AWS S3 buckets, specify * instead of a bucket name as in the following example:

```
"Resource": [
   "arn:aws:s3:::*"
]
```

5. Attach the EC2 EBS policy to velero:

```
$ aws iam put-user-policy \
   --user-name velero \
   --policy-name velero-ebs \
   --policy-document file://velero-ec2-snapshot-policy.json
```
6. Attach the AWS S3 policy to `velero`:

```bash
$ aws iam put-user-policy \
--user-name velero \
--policy-name velero-s3 \
--policy-document file://velero-s3-policy.json
```

7. Create an access key for `velero`:

```bash
$ aws iam create-access-key --user-name velero
{
    "AccessKey": {
        "UserName": "velero",
        "Status": "Active",
        "CreateDate": "2017-07-31T22:24:41.576Z",
        "SecretAccessKey": "<AWS_SECRET_ACCESS_KEY>"  \[1\]
        "AccessKeyId": "<AWS_ACCESS_KEY_ID>"  \[2\]
    }
}
```

Record the `AWS_SECRET_ACCESS_KEY` and the `AWS_ACCESS_KEY_ID` for adding the AWS repository to the MTC web console.

### 2.3.3. Configuring a Google Cloud Provider storage bucket as a replication repository

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

**Prerequisites**

- The GCP storage bucket must be accessible to the source and target clusters.
- You must have `gsutil` installed.
- 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 `gsutil`:

   ```bash
   $ gsutil init
   ```

   **Example output**

   Welcome! This command will take you through the configuration of gcloud.
Your current configuration has been set to: [default]
To continue, you must login. Would you like to login (Y/n)?

2. Set the `BUCKET` variable:

   ```shell
   $ BUCKET=<bucket_name>
   ```

   1 Specify your bucket name.

3. Create a storage bucket:

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

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

   ```shell
   $ PROJECT_ID=`gcloud config get-value project`
   ```

5. Create a `velero` IAM service account:

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

6. Create the `SERVICE_ACCOUNT_EMAIL` variable:

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

7. Create the `ROLE_PERMISSIONS` variable:

   ```shell
   $ 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
   )
   ```

8. Create the `velero.server` custom role:

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

9. Add IAM policy binding to the project:
2.3.4. Configuring a Microsoft Azure Blob storage container as a replication repository

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

Prerequisites

- You must have an Azure storage account.
- 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. Set the **AZURE_RESOURCE_GROUP** variable:

   ```bash
   $ AZURE_RESOURCE_GROUP=Velero_Backups
   ```

2. Create an Azure resource group:

   ```bash
   $ az group create -n $AZURE_RESOURCE_GROUP --location <CentralUS>
   ```

3. Set the **AZURE_STORAGE_ACCOUNT_ID** variable:

   ```bash
   $ AZURE_STORAGE_ACCOUNT_ID=velerobackups
   ```

4. Create an Azure storage account:

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

10. Update the IAM service account:

   ```bash
   $ gsutil iam ch serviceAccount:$SERVICE_ACCOUNT_EMAIL:objectAdmin gs://$[BUCKET]
   ```

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

   ```bash
   $ gcloud iam service-accounts keys create credentials-velero \
   --iam-account $SERVICE_ACCOUNT_EMAIL
   ```
5. Set the `BLOB_CONTAINER` variable:

```
$ BLOB_CONTAINER=velero
```

6. Create an Azure Blob storage container:

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

7. 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`
```

8. 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
```

### 2.4. MIGRATING YOUR APPLICATIONS

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

#### 2.4.1. Prerequisites

The Migration Toolkit for Containers (MTC) has the following prerequisites:

- You must be logged in as a user with `cluster-admin` privileges on all clusters.
- The MTC version must be the same on all clusters.
- Clusters:
The source cluster must be upgraded to the latest MTC z-stream release.

The cluster on which the **migration-controller** pod is running must have unrestricted network access to the other clusters.

The clusters must have unrestricted network access to each other.

The clusters must have unrestricted network access to the replication repository.

The clusters must be able to communicate using OpenShift routes on port 443.

The clusters must have no critical conditions.

The clusters must be in a ready state.

- **Volume migration:**
  - The persistent volumes (PVs) must be valid.
  - The PVs must be bound to persistent volume claims.
  - If you copy the PVs by using the `move` method, the clusters must have unrestricted network access to the remote volume.
  - If you copy the PVs by using the `snapshot` copy method, the following prerequisites apply:
    - The cloud provider must support snapshots.
    - The volumes must have the same cloud provider.
    - The volumes must be located in the same geographic region.
    - The volumes must have the same storage class.

- If you perform a direct volume migration in a proxy environment, you must configure an Stunnel TCP proxy.

- If you perform a direct image migration, you must expose the internal registry of the source cluster to external traffic.

### 2.4.1.1. Creating a CA certificate bundle file

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-/m' > <ca_bundle.cert> 2
```

1. Specify the host FQDN and port of the endpoint, for example, `api.my-cluster.example.com:6443`.  
2. The certificate is copied to the `<ca_bundle.cert>` file.
Specify the name of the CA bundle file.

2.4.1.2. Configuring a proxy for direct volume migration

If you are performing direct volume migration from a source cluster behind a proxy, you must configure an Stunnel proxy in the MigrationController custom resource (CR). Stunnel creates a transparent tunnel between the source and target clusters for the TCP connection without changing the certificates.

**NOTE**

Direct volume migration supports only one proxy. The source cluster cannot access the route of the target cluster if the target cluster is also behind a proxy.

Prerequisites

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

Procedure

1. Log in to the cluster on which the MigrationController pod runs.

2. Get the MigrationController CR manifest:

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

3. Add the stunnel_tcp_proxy parameter:

   ```
   apiVersion: migration.openshift.io/v1alpha1
   kind: MigrationController
   metadata:
     name: migration-controller
     namespace: openshift-migration
   ...
   spec:
     stunnel_tcp_proxy: <stunnel_proxy>  
   ```

   ① Specify the Stunnel proxy: http://<user_name>:<password>@<ip_address>[:port].

4. Save the manifest as migration-controller.yaml.

5. Apply the updated manifest:

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

2.4.1.3. 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.

2.4.1.3.1. Ansible modules

You can use the Ansible **shell** module to run `oc` commands.

**Example shell module**

```yaml
- hosts: localhost
  gather_facts: false
  tasks:
    - name: get pod name
      shell: oc get po --all-namespaces
```

You can use **kubernetes.core** modules, such as **k8s_info**, to interact with Kubernetes resources.

**Example k8s_facts module**

```yaml
- 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**

```yaml
- 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
```
2.4.1.3.2. 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` plug-in.

Example environment variables

```yaml
- hosts: localhost
gather_facts: false
tasks:
  - set_fact:
      namespaces: "{{ (lookup('env', 'migration_namespaces')).split(',') }}"
  - debug:
      msg: "{{ item }}"
    with_items: "{{ namespaces }}"
  - debug:
      msg: "{{ lookup('env', 'migplan_name') }}"
```

2.4.1.4. Additional resources

- About migration hooks
- MigHook custom resource
- MigPlan custom resource

2.4.2. Migrating your applications 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.

2.4.2.1. 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:

   ```bash
   $ 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`. 

---

OpenShift Container Platform 4.8 Migration Toolkit for Containers

122
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.

2.4.2.2. 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.

Procedure

1. Log in to the cluster.

2. Obtain the migration-controller service account token:

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

Example output

```
eyJhbGciOiJSUzI1NiIsImtlImtpZCI6IjJ9.eyJpc3MiOiJwdWJlcm5ldGVzL3Nlc3NzeyJpYy5kJyc3Mi5kaWQiOiJiZGlueSIsInNoYW5nZSI6IjBhY2NlbmNlIiwiZGlhbl9pZCI6IjE0Il0sImltYWdlIjoiYXN0cmV0YXMifSwiaWQiOiJ0b3RvY2F0aW9uLWNsb3NlcnN0aW9uLWJ1YiIsImNsaWNrIjoiYXNzZW4ifQ.
```

3. In the MTC web console, click Clusters.
4. Click **Add cluster**.

5. Fill in the following fields:

   - **Cluster name**: The cluster name can contain lower-case letters (a-z) and numbers (0-9). It must not contain spaces or international characters.
   - **URL**: Specify the API server URL, for example, `https://www.example.com`:8443.
   - **Service account token**: Paste the migration-controller service account token.
   - **Exposed route host to image registry**: If you are using direct image migration, specify the exposed route to the image registry of the source cluster, for example, `www.example.apps.cluster.com`. You can specify a port. The default port is 5000.
   - **Azure cluster**: You must select this option if you use Azure snapshots to copy your data.
   - **Azure resource group**: This field is displayed if Azure cluster is selected. Specify the Azure resource group.
   - **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.

2.4.2.3. Adding a replication repository to the MTC web console

You can add an object storage bucket as a replication repository to the Migration Toolkit for Containers (MTC) web console.

**Prerequisites**

- You must configure an object storage bucket for migrating the data.

**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 AWS S3, MCG, and generic S3 providers:
     - **Replication repository name**: Specify the replication repository name in the MTC web console.
     - **S3 bucket name**: Specify the name of the S3 bucket you created.
     - **S3 bucket region**: Specify the S3 bucket region. **Required** for AWS S3. **Optional** for other S3 providers.
**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.

**S3 provider secret access key**: Specify the `<AWS_ACCESS_KEY_ID>` for AWS or the S3 provider secret access key for MCG.

**Require SSL verification**: Clear this check box if you are using a generic S3 provider.

- If you use a custom CA bundle, click **Browse** and browse to the Base64-encoded CA bundle 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.

### 2.4.2.4. 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, the **MigCluster** custom resource manifest of the source cluster must specify the exposed route of the internal image registry.
Procedure

1. In the MTC web console, click **Migration plans**.

2. Click **Add migration plan**.

3. Enter the **Plan name** and click **Next**.
   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**.

5. Select a **Target cluster**.

6. Select a **Replication repository**.

7. Select the projects to be migrated and click **Next**.

8. Select a **Source cluster**, a **Target cluster**, and a **Repository**, and click **Next**.

9. On the **Namespaces** page, select the projects to be migrated and click **Next**.

10. On the **Persistent volumes** page, click 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.

11. Click **Next**.

12. On the **Copy options** page, 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 filesystem copy method is required for direct volume migration.

13. 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.

14. Select a **Target storage class**.
    If you selected **Filesystem copy**, you can change the target storage class.

15. Click **Next**.

16. 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
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.

17. Click **Next**.

18. Optional: On the **Hooks** page, 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**.

19. Click **Finish**.

   The migration plan is displayed in the **Migration plans** list.

### 2.4.2.5. Running a migration plan in the MTC web console

You can stage or 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 source cluster.

2. Delete old images:

   ```
   $ oc adm prune images
   ```

3. Log in to the MTC web console and click **Migration plans**.

4. Click the **Options** menu next to a migration plan and select **Stage** to copy data from the source cluster to the target cluster without stopping the application. You can run **Stage** multiple times to reduce the actual migration time.

5. When you are ready to migrate the application workload, the **Options** menu beside a migration plan and select **Migrate**.

6. Optional: In the **Migrate** window, you can select **Do not stop applications on the source cluster during migration**.

7. Click **Migrate**.

8. 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.

**2.4.3. Migrating your applications from the command line**

You can migrate your applications on the command line by using the MTC custom resources (CRs).

You can migrate applications from a local cluster to a remote cluster, from a remote cluster to a local cluster, and between remote clusters.

**MTC terminology**
The following terms are relevant for configuring clusters:

- **host** cluster:
  - The *migration-controller* pod runs on the **host** cluster.
  - A **host** cluster does not require an exposed secure registry route for direct image migration.

- Local cluster: The local cluster is often the same as the **host** cluster but this is not a requirement.

- Remote cluster:
  - A remote cluster must have an exposed secure registry route for direct image migration.
  - A remote cluster must have a **Secret CR** containing the *migration-controller* service account token.

The following terms are relevant for performing a migration:

- **Source cluster**: Cluster from which the applications are migrated.
- **Destination cluster**: Cluster to which the applications are migrated.

### 2.4.3.1. Migrating your applications with the Migration Toolkit for Containers API

You can migrate your applications on the command line with the Migration Toolkit for Containers (MTC) API.

You can migrate applications from a local cluster to a remote cluster, from a remote cluster to a local cluster, and between remote clusters.

This procedure describes how to perform indirect migration and direct 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 migration: Images or volumes are copied directly from the source cluster to the destination cluster. Direct image migration and direct volume migration have significant performance benefits.

You create the following custom resources (CRs) to perform a migration:

- **MigCluster CR**: Defines a **host**, local, or remote cluster
  The *migration-controller* pod runs on the **host** cluster.

- **Secret CR**: Contains credentials for a remote cluster or storage

- **MigStorage CR**: Defines a replication repository
  Different storage providers require different parameters in the **MigStorage CR** manifest.

- **MigPlan CR**: Defines a migration plan

- **MigMigration CR**: Performs a migration defined in an associated **MigPlan** CR
  You can create multiple **MigMigration CRs** for a single **MigPlan CR** for the following purposes:
To perform stage migrations, which copy most of the data without stopping the application, before running a migration. Stage migrations improve the performance of the migration.

- To cancel a migration in progress
- To roll back a completed migration

Prerequisites

- You must have **cluster-admin** privileges for all clusters.
- You must install the OpenShift Container Platform CLI (**oc**).
- You must install the Migration Toolkit for Containers Operator on all clusters.
- The **version** of the installed Migration Toolkit for Containers Operator must be the same on all clusters.
- You must configure an object storage as a replication repository.
- If you are using direct image migration, you must expose a secure registry route on all remote clusters.
- If you are using direct volume migration, the source cluster must not have an HTTP proxy configured.

Procedure

1. Create a **MigCluster** CR manifest for the **host** cluster called **host-cluster.yaml**:

   ```yaml
   apiVersion: migration.openshift.io/v1alpha1
   kind: MigCluster
   metadata:
     name: host
     namespace: openshift-migration
   spec:
     isHostCluster: true
   ```

2. Create a **MigCluster** CR for the **host** cluster:

   ```bash
   $ oc create -f host-cluster.yaml -n openshift-migration
   ```

3. Create a **Secret** CR manifest for each remote cluster called **cluster-secret.yaml**:

   ```yaml
   apiVersion: v1
   kind: Secret
   metadata:
     name: <cluster_secret>
     namespace: openshift-config
   type: Opaque
   data:
     saToken: <sa_token>
   ```

   1 Specify the base64-encoded **migration-controller** service account (SA) token of the remote cluster.
You can obtain the SA token by running the following command:

```bash
$ oc sa get-token migration-controller -n openshift-migration | base64 -w 0
```

4. Create a **Secret** CR for each remote cluster:

```bash
$ oc create -f cluster-secret.yaml
```

5. Create a **MigCluster** CR manifest for each remote cluster called `remote-cluster.yaml`:

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigCluster
metadata:
  name: <remote_cluster>
  namespace: openshift-migration
spec:
  exposedRegistryPath: <exposed_registry_route> 1
  insecure: false 2
  isHostCluster: false
  serviceAccountSecretRef:
    name: <remote_cluster_secret> 3
    namespace: openshift-config
  url: <remote_cluster_url> 4
```

1. Optional: Specify the exposed registry route, for example, `docker-registry-default.apps.example.com` if you are using direct image migration.

2. SSL verification is enabled if `false`. CA certificates are not required or checked if `true`.

3. Specify the **Secret** CR of the remote cluster.

4. Specify the URL of the remote cluster.

6. Create a **MigCluster** CR for each remote cluster:

```bash
$ oc create -f remote-cluster.yaml -n openshift-migration
```

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

```bash
$ oc describe cluster <cluster_name>
```

8. Create a **Secret** CR manifest for the replication repository called `storage-secret.yaml`:

```yaml
apiVersion: v1
kind: Secret
metadata:
  namespace: openshift-config
  name: <migstorage_creds>
  type: Opaque
data:
  aws-access-key-id: <key_id_base64> 1
  aws-secret-access-key: <secret_key_base64> 2
```
Specify the key ID in base64 format.

Specify the secret key in base64 format.

AWS credentials are base64-encoded by default. If you are using another storage provider, you must encode your credentials by running the following command with each key:

```
$ echo -n "<key>" | base64 -w 0
```

Specify the key ID or the secret key. Both keys must be base64-encoded.

9. Create the **Secret** CR for the replication repository:

```
$ oc create -f storage-secret.yaml
```

10. Create a **MigStorage** CR manifest for the replication repository called `migstorage.yaml`:

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigStorage
metadata:
  name: <storage_name>
  namespace: openshift-migration
spec:
  backupStorageConfig:
    awsBucketName: <bucket_name> 1
    credsSecretRef:
      name: <storage_secret_ref> 2
      namespace: openshift-config
  backupStorageProvider: <storage_provider_name> 3
  volumeSnapshotConfig:
    credsSecretRef:
      name: <storage_secret_ref> 4
      namespace: openshift-config
  volumeSnapshotProvider: <storage_provider_name> 5
```

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.

11. Create the **MigStorage** CR:

```
$ oc create -f migstorage.yaml -n openshift-migration
```
12. Verify that the **MigStorage** CR is in a **Ready** state:

   ```bash
   $ oc describe migstorage <migstorage_name>
   ```

13. Create a **MigPlan** CR manifest called **migplan.yaml**:

   ```yaml
   apiVersion: migration.openshift.io/v1alpha1
   kind: MigPlan
   metadata:
     name: <migration_plan>
     namespace: openshift-migration
   spec:
     destMigClusterRef:
       name: host
       namespace: openshift-migration
     indirectImageMigration: true
     indirectVolumeMigration: true
     migStorageRef:
       name: <migstorage_ref>
       namespace: openshift-migration
     namespaces:
     - <application_namespace>
     srcMigClusterRef:
       name: <remote_cluster_ref>
       namespace: openshift-migration
   ```

   - Direct image migration is enabled if **false**.
   - Direct volume migration is enabled if **false**.
   - Specify the name of the **MigStorage** CR instance.
   - Specify one or more namespaces to be migrated.
   - Specify the name of the source cluster **MigCluster** instance.

14. Create the **MigPlan** CR:

   ```bash
   $ oc create -f migplan.yaml -n openshift-migration
   ```

15. View the **MigPlan** instance to verify that it is in a **Ready** state:

   ```bash
   $ oc describe migplan <migplan_name> -n openshift-migration
   ```

16. Create a **MigMigration** CR manifest called **migmigration.yaml**:

   ```yaml
   apiVersion: migration.openshift.io/v1alpha1
   kind: MigMigration
   metadata:
     name: <migmigration_name>
     namespace: openshift-migration
   spec:
     migPlanRef:
   ```
Specify the MigPlan CR name.

The pods on the source cluster are stopped before migration if true.

A stage migration, which copies most of the data without stopping the application, is performed if true.

A completed migration is rolled back if true.

17. Create the MigMigration CR to start the migration defined in the MigPlan CR:

```
$ oc create -f migmigration.yaml -n openshift-migration
```

18. Verify the progress of the migration by watching the MigMigration CR:

```
$ oc watch migmigration <migmigration_name> -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: my_application
  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
```
2.4.3.2. MTC custom resource manifests

Migration Toolkit for Containers (MTC) uses the following custom resource (CR) manifests to create CRs for migrating applications.

2.4.3.2.1. 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: <directimagemigration_name>
spec:
  destMigrationClusterRef:
    name: <destination_cluster_ref>
    namespace: openshift-migration
  namespaces:
    - <namespace>

1. Specify the MigCluster CR name of the source cluster.
2. Specify the MigCluster CR name of the destination cluster.
3. Specify one or more namespaces containing images to be migrated.

2.4.3.2.2. 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: directimagestreammigration_name
spec:
  destMigrationClusterRef:
    name: <destination_cluster_ref>
    namespace: openshift-migration
  imageStreamRef:
    name: <image_stream_name>
    namespace: <source_image_stream_namespace>
  destNamespace: <destination_image_stream_namespace>

1. Specify the MigCluster CR name of the source cluster.
2. Specify the MigCluster CR name of the destination cluster.
3. Specify the image stream name.
4. Specify the image stream namespace on the source cluster.
Specify the image stream namespace on the destination cluster.

2.4.3.2.3. DirectVolumeMigration

The **DirectVolumeMigration** CR copies persistent volumes (PVs) directly from the source cluster to the destination cluster.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: DirectVolumeMigration
metadata:
  name: <directvolumemigration_name>
  namespace: openshift-migration
spec:
  createDestinationNamespaces: false
  deleteProgressReportingCRs: false
  destMigClusterRef:
    name: host
    namespace: openshift-migration
  persistentVolumeClaims:
    - name: <pvc_name>
      namespace: <pvc_namespace>
  srcMigClusterRef:
    name: <source_cluster_ref>
    namespace: openshift-migration
```

1 Namespaces are created for the PVs on the destination cluster if **true**.

2 The **DirectVolumeMigrationProgress** CRs are deleted after migration if **true**. The default value 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 with direct volume migration.

5 Specify the namespace of each PVC.

6 Specify the **MigCluster** CR name of the source cluster.

2.4.3.2.4. DirectVolumeMigrationProgress

The **DirectVolumeMigrationProgress** CR shows the progress of the **DirectVolumeMigration** CR.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: DirectVolumeMigrationProgress
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: directvolumemigrationprogress_name
spec:
  clusterRef:
    name: source_cluster
    namespace: openshift-migration
```
2.4.3.2.5. MigAnalytic

The **MigAnalytic** CR collects the number of images, Kubernetes resources, and the PV capacity from an associated **MigPlan** CR.

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigAnalytic
metadata:
  annotations:
    migplan: <migplan_name>
  name: miganalytic_name
  namespace: openshift-migration
  labels:
    migplan: <migplan_name>
spec:
  analyzeImageCount: true
  analyzeK8SResources: true
  analyzePVCapacity: true
  listImages: false
  listImagesLimit: 50
  migPlanRef:
    name: migplan_name
    namespace: openshift-migration
```

1. Specify the **MigPlan** CR name associated with the **MigAnalytic** CR.
2. Specify the **MigPlan** CR name associated with the **MigAnalytic** CR.
3. Optional: The number of images is returned if `true`.
4. Optional: Returns the number, kind, and API version of the Kubernetes resources if `true`.
5. Optional: Returns the PV capacity if `true`.
6. Returns a list of image names if `true`. Default is `false` so that the output is not excessively long.
7. Optional: Specify the maximum number of image names to return if `listImages` is `true`.
8. Specify the **MigPlan** CR name associated with the **MigAnalytic** CR.

2.4.3.2.6. 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"
```
Optional: Update the cluster name if the migration-controller pod is not running on this cluster.

The migration-controller pod runs on this cluster if true.

Optional: If the storage provider is Microsoft Azure, specify the resource group.

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.

SSL verification is enabled if false.

The cluster is validated if true.

The restic pods are restarted on the source cluster after the stage pods are created if true.

Optional: If you are using direct image migration, specify the exposed registry path of a remote cluster.

Specify the URL of the remote cluster.

Specify the name of the Secret CR for the remote cluster.

2.4.3.2.7. MigHook

The MigHook CR defines an Ansible playbook or a custom image that runs tasks at a specified stage of the migration.
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.

Specify the migration hook name, unless you specify the value of the generateName parameter.

Optional: Specify the maximum number of seconds that a hook can run. The default value is 1800.

The hook is a custom image if true. The custom image can include Ansible or it can be written in a different programming language.

Specify the custom image, for example, quay.io/konveyor/hook-runner:latest. Required if custom is true.

Specify the entire base64-encoded Ansible playbook. Required if custom is false.

Specify source or destination as the cluster on which the hook will run.

2.4.3.2.8. MigMigration

The MigMigration CR runs an associated MigPlan CR.

You can create multiple MigMigration CRs associated with the same MigPlan CR for the following scenarios:

- You can run multiple stage or incremental migrations to copy data without stopping the pods on the source cluster. Running stage migrations improves the performance of the actual migration.

- You can cancel a migration in progress.

- You can roll back a migration.

apiVersion: migration.openshift.io/v1alpha1
kind: MigMigration
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: migmigration_name
  namespace: openshift-migration
spec:
canceled: false
rollback: false
stage: false
quiescePods: true
keepAnnotations: true
verify: false
migPlanRef:
  name: <migplan_ref>
  namespace: openshift-migration
1. A migration in progress is canceled if `true`.

2. A completed migration is rolled back if `true`.

3. Data is copied incrementally and the pods on the source cluster are not stopped if `true`.

4. The pods on the source cluster are scaled to 0 after the Backup stage of a migration if `true`.

5. The labels and annotations applied during the migration are retained if `true`.

6. The status of the migrated pods on the destination cluster are checked and the names of pods that are not in a Running state are returned if `true`.

7. `migPlanRef.name`: Specify the name of the associated `MigPlan` CR.

2.4.3.2.9. MigPlan

The `MigPlan` CR defines the parameters of a migration plan. It contains a group of virtual machines that are being migrated with the same parameters.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigPlan
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
name: migplan_name
namespace: openshift-migration
spec:
  closed: false
  srcMigClusterRef:
    name: <source_migcluster_ref>
    namespace: openshift-migration
  destMigClusterRef:
    name: <destination_migcluster_ref>
    namespace: openshift-migration
  hooks:
    - executionNamespace: <namespace>
      phase: <migration_phase>
      reference:
        name: <mighook_name>
        namespace: <hook_namespace>
        serviceAccount: <service_account>
  indirectImageMigration: true
  indirectVolumeMigration: false
  migStorageRef:
    name: <migstorage_name>
    namespace: openshift-migration
  namespaces:
    - <namespace>
  refresh: false
```

1. The migration has completed if `true`. You cannot create another `MigMigration` CR for this `MigPlan` CR.
Specify the name of the source cluster **MigCluster** CR.

Specify the name of the destination cluster **MigCluster** CR.

Optional: You can specify up to four migration hooks.

Optional: Specify the namespace in which the hook will run.

Optional: Specify the migration phase during which a hook runs. One hook can be assigned to one phase. The expected values are **PreBackup**, **PostBackup**, **PreRestore**, and **PostRestore**.

Optional: Specify the name of the **MigHook** CR.

Optional: Specify the namespace of **MigHook** CR.

Optional: Specify a service account with **cluster-admin** privileges.

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.

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.

Specify the name of **MigStorage** CR.

Specify one or more namespaces.

The **MigPlan** CR is validated if **true**.

### 2.4.3.2.10. MigStorage

The **MigStorage** CR describes the object storage for the replication repository. You can configure Amazon Web Services, Microsoft Azure, Google Cloud Storage, and generic S3-compatible cloud storage, for example, Minio or NooBaa.

Different providers require different parameters.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigStorage
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: migstorage_name
  namespace: openshift-migration
spec:
  backupStorageProvider: <storage_provider> 1
  volumeSnapshotProvider: 2
  backupStorageConfig:
    awsBucketName: 3
    awsRegion: 4
    credsSecretRef:
      namespace: openshift-config
      name: <storage_secret> 5
    awsKmsKeyId: 6
    awsPublicUrl: 7
```

OpenShift Container Platform 4.8 Migration Toolkit for Containers
awsSignatureVersion: 8
volumeSnapshotConfig:
  awsRegion: 9
credsSecretRef:
    namespace: openshift-config
    name: 10
refresh: false 11

1. Specify the storage provider.
2. Optional: If you are using the snapshot copy method, specify the storage provider.
3. If you are using AWS, specify the bucket name.
4. If you are using AWS, specify the bucket region, for example, us-east-1.
5. Specify the name of the Secret CR that you created for the MigStorage CR.
6. Optional: If you are using the AWS Key Management Service, specify the unique identifier of the key.
7. Optional: If you granted public access to the AWS bucket, specify the bucket URL.
8. Optional: Specify the AWS signature version for authenticating requests to the bucket, for example, 4.
9. Optional: If you are using the snapshot copy method, specify the geographical region of the clusters.
10. Optional: If you are using the snapshot copy method, specify the name of the Secret CR that you created for the MigStorage CR.
11. The cluster is validated if true.

2.4.4. Additional resources
- Exposing a secure registry manually on an OpenShift Container Platform 4 cluster
- MTC file system copy method
- MTC snapshot copy method
- Viewing migration custom resources

2.4.5. Configuring a migration plan
You can increase the number of objects to be migrated or exclude resources from the migration.

2.4.5.1. 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.

2.4.5.2. Excluding resources from a migration plan

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 a parameter to exclude specific resources or by adding a resource to the **excluded_resources** parameter if it does not have its own exclusion 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
     ...
   excluded_resources: 3
     - imagetags
     - templateinstances
     - clusterserviceversions
     - packagemanifests
     - subscriptions
     - servicebrokers
     - servicebindings
     - serviceclasses
     - serviceinstances
     - serviceplans
     - operatorgroups
     - events
   ```

   **1** Add `disable_image_migration: true` to exclude image streams from the migration. Do not edit the `excluded_resources` parameter. `imagetags` is added to `excluded_resources` when the **MigrationController** pod restarts.

   **2** Add `disable_pv_migration: true` to exclude PVs from the migration plan. Do not edit the `excluded_resources` parameter. `persistentvolumes` and `persistentvolumeclaims` are added to `excluded_resources` 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 to the `excluded_resources` list. Do not delete the default excluded resources. These resources are problematic to migrate and must be excluded.

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:
2.5. TROUBLESHOOTING
You can view the Migration Toolkit for Containers (MTC) custom resources and download logs to troubleshoot a failed migration.

If the application was stopped during the failed migration, you must roll back the migration to prevent data corruption.

**NOTE**
Manual rollback is not required if the application was not stopped during migration because the original application is still running on the source cluster.

2.5.1. Viewing MTC custom resources
You can view 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**

<table>
<thead>
<tr>
<th>NAME</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>88435fe0-c9f8-11e9-85e6-5d593ce65e10</td>
<td>6m42s</td>
</tr>
</tbody>
</table>

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**

```yaml
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
geneneration: 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-0205fe66cb6
    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-0205fe66cb6
  migration-stage-backup: 8886de4c-c9f8-11e9-95ad-0205fe66cb6
  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-0205fe66cb6
spec:
excludedNamespaces: []
excludedResources: []
hooks:
  resources: []
includeClusterResources: null
includedNamespaces:
  - sock-shop
includeResources:
  - persistentvolumes
  - persistentvolumeclaims
  - namespaces
  - imagestreams
  - imagestreamtags
  - secrets
  - configmaps
  - pods
labelSelector:
  matchLabels:
    migration-included-stage-backup: 8886de4c-c9f8-11e9-95ad-0205fe66cb6
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

2.5.2. 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 The -c plain option displays the log without colors.

2.5.3. Downloading migration logs

You can download the Velero, Restic, and MigrationController pod logs in the Migration Toolkit for Containers (MTC) web console to troubleshoot a failed migration.

Procedure

1. In the MTC console, click Migration plans to view the list of migration plans.

2. Click the Options menu of a specific migration plan and select Logs.

3. Click Download Logs to download the logs of the MigrationController, Velero, and Restic pods for all clusters.
   You can download a single log by selecting the cluster, log source, and pod source, and then clicking Download Selected.

   You can access a pod log from the CLI by using the oc logs command:

   \$ oc logs <pod-name> -f -n openshift-migration

   1 Specify the pod name.

2.5.4. Updating deprecated APIs

If your source cluster uses deprecated APIs, the following warning message is displayed when you create a migration plan in the Migration Toolkit for Containers (MTC) web console:

Some namespaces contain GVKs incompatible with destination cluster

You can click See details to view the namespace and the incompatible APIs. This warning message does not block the migration.

During migration with the Migration Toolkit for Containers (MTC), the deprecated APIs are saved in the Velero Backup #1 for Kubernetes objects. You can download the Velero Backup, extract the deprecated API yaml files, and update them with the oc convert command. Then you can create the updated APIs on the target cluster.

Procedure
1. Run the migration plan.

2. View the **MigPlan** custom resource (CR):

   ```
   $ oc describe migplan <migplan_name> -n openshift-migration
   
   Specify the name of the **MigPlan** CR.
   
   The output is similar to the following:
   
   metadata:
   ...
   uid: 79509e05-61d6-11e9-bc55-02ce4781844a
   status:
   ...
   conditions:
   - category: Warn
     lastTransitionTime: 2020-04-30T17:16:23Z
     message: 'Some namespaces contain GVKs incompatible with destination cluster. See: `incompatibleNamespaces` for details'
     status: "True"
     type: GVKsIncompatible
   incompatibleNamespaces:
   - gvks:
     - group: batch
       kind: cronjobs
       version: v2alpha1
     - group: batch
       kind: scheduledjobs
       version: v2alpha1
   
   Record the **MigPlan** CR UID.
   
   Record the deprecated APIs listed in the **gvks** section.
   
3. Get the **MigMigration** name associated with the **MigPlan** UID:

   ```
   $ oc get migmigration -o json | jq -r '.items | select(.metadata.ownerReferences[].uid=="<migplan_uid>") | .metadata.name'
   
   Specify the **MigPlan** CR UID.
   
4. Get the **MigMigration** UID associated with the **MigMigration** name:

   ```
   $ oc get migmigration <migmigration_name> -o jsonpath='{.metadata.uid}'
   
   Specify the **MigMigration** name.
   
5. Get the **Velero** Backup name associated with the **MigMigration** UID:
Specify the *MigMigration* UID.

6. Download the contents of the *Velero* Backup to your local machine by running the command for your storage provider:

   - AWS S3:
     ```
     $ aws s3 cp s3://<bucket_name>/velero/backups/<backup_name> <backup_local_dir> --recursive
     ```
     Specify the bucket, backup name, and your local backup directory name.

   - GCP:
     ```
     $ gsutil cp gs://<bucket_name>/velero/backups/<backup_name> <backup_local_dir> --recursive
     ```
     Specify the bucket, backup name, and your local backup directory name.

   - Azure:
     ```
     $ azcopy copy 'https://velerobackups.blob.core.windows.net/velero/backups/<backup_name>' '<backup_local_dir>' --recursive
     ```
     Specify the backup name and your local backup directory name.

7. Extract the *Velero* Backup archive file:
   ```
   $ tar -xfv <backup_local_dir>/<backup_name>.tar.gz -C <backup_local_dir>
   ```

8. Run *oc convert* in offline mode on each deprecated API:
   ```
   $ oc convert -f <backup_local_dir>/resources/<gvk>.json
   ```

9. Create the converted API on the target cluster:
   ```
   $ oc create -f <gvk>.json
   ```

### 2.5.5. 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.

#### 2.5.5.1. CA certificate error in the MTC console
If the MTC console displays a **CA certificate error** message the first time you try to access it, the likely cause is that a cluster uses self-signed CA certificates.

Navigate to the `oauth-authorization-server` URL in the error message and accept the certificate. To resolve this issue permanently, install the certificate authority so that it is trusted.

If the browser displays an **Unauthorized** message after you have accepted the CA certificate, navigate to the MTC console and then refresh the web page.

### 2.5.5.2. OAuth timeout error in the MTC console

If the MTC console displays a **connection has timed out** message after you have accepted a self-signed certificate, the cause is likely to be one of the following:

- Interrupted network access to the OAuth server
-Interrupted network access to the OpenShift Container Platform console
- Proxy configuration blocking access to the OAuth server. See [MTC console inaccessible because of OAuth timeout error](#) for details.

To determine the cause:

- Inspect the MTC console web page with a browser web inspector.
- Check the **Migration UI** pod log for errors.

### 2.5.5.3. 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:

**BSL error messages**

- Error checking repository for stale locks
- Error getting backup storage location: backupstoragelocation.velero.io "my-bsl" not found

You can ignore these error messages. A missing BSL cannot cause a migration to fail.

### 2.5.5.4. Pod volume backup timeout error in the Velero pod log

If a migration fails because **Restic** times out, the **Velero** pod log displays the following error:

**Pod volume backup timeout error**

```
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:

   ```yaml
   spec:
     restic_timeout: 1h
   ```

   1 Valid units are h (hours), m (minutes), and s (seconds), for example, 3h30m15s.

5. Click Save.

2.5.5.5. 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 MigMigration CR displays the following error:

**MigMigration CR status**

```
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
  status: "True"
type: ResticVerifyErrors
```

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 troubleshoot 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.

### Restore CR with pod volume restore error

```yaml
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.

### PodVolumeRestore CR with Restic pod error

```yaml
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>
```

### 2.5.5.6. 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 **Restic** pod log displays the following error:

**Restic permission error**

```
```

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:

   ```yaml
   spec:
   restic_supplemental_groups: <group_id>
   ```

   Specify the supplemental group ID.

4. Wait for the Restic pods to restart so that the changes are applied.

2.5.6. Direct volume migration does not complete

If direct volume migration does not complete, the target cluster might not have the same `node-selector` annotations as the source cluster.

Migration Toolkit for Containers (MTC) migrates namespaces with all annotations to preserve security context constraints and scheduling requirements. During direct volume migration, MTC creates Rsync transfer pods on the target cluster in the namespaces that were migrated from the source cluster. If a target cluster namespace does not have the same annotations as the source cluster namespace, the Rsync transfer pods cannot be scheduled. The Rsync pods remain in a Pending state.

You can identify and fix this issue by performing the following procedure.

Procedure

1. Check the status of the MigMigration CR:

   ```bash
   $ oc describe migmigration <pod_name> -n openshift-migration
   ```

   The output includes the following status message:

   **Example output**

   ```
   ... Some or all transfer pods are not running for more than 10 mins on destination cluster ...
   ```

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

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

   Specify the migrated namespace.

3. On the target cluster, edit the migrated namespace:

   ```bash
   $ oc edit namespace <namespace>
   ```
4. Add missing `openshift.io/node-selector` annotations to the migrated namespace as in the following example:

```yaml
apiVersion: v1
class: Namespace
metadata:
  annotations:
    openshift.io/node-selector: "region=west"
... 
```

5. Run the migration plan again.

### 2.5.7. Using the Velero CLI to debug Backup and Restore CRs

You can debug the Backup and Restore custom resources (CRs) and partial migration failures with the Velero command line interface (CLI). The Velero CLI runs in the `velero` pod.

#### 2.5.7.1. Velero command syntax

Velero CLI commands use the following syntax:

```bash
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -- ./velero <command> <resource_id>
```

You can specify `velero-<pod> -n openshift-migration` in place of `$oc get pods -n openshift-migration -o name | grep velero`.

#### 2.5.7.2. Help command

The Velero `help` command lists all the Velero CLI commands:

```bash
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -- ./velero --help
```

#### 2.5.7.3. Describe command

The Velero `describe` command provides a summary of warnings and errors associated with a Velero resource:

```bash
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -- ./velero describe <resource_id>
```

**Example**

```bash
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -- ./velero backup describe 0e44ae00-5dc3-11eb-9ca8-df7e5254778b-2d8ql
```

#### 2.5.7.4. Logs command

The Velero `logs` command provides the logs associated with a Velero resource:

```bash
velero <resource> logs <resource_id>
```
2.5.7.5. 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**
   ```yaml
   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 exec $(oc get pods -n openshift-migration -o name | grep velero) -n openshift-migration -
   - ./velero restore describe <restore>
   ```

   **Example output**
   ```yaml
   Phase: PartiallyFailed (run `velero restore logs ccc7c2d0-6017-11eb-afab-85d0007f5a19-x4lbf` for more information)
   Errors:
   ```
3. Check the **Restore CR logs** by using the Velero **logs** command:

```
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -n openshift-migration - 
- ./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.

### 2.5.8. Using must-gather to collect data

You must run the **must-gather** tool if you open a customer support case on the Red Hat Customer Portal for the Migration Toolkit for Containers (MTC).

The **openshift-migration-must-gather-rhel8** image for MTC collects migration-specific logs and data that are not collected by the default **must-gather** image.

**Procedure**

1. Navigate to the directory where you want to store the **must-gather** data.
2. Run the **must-gather** command:

```
$ oc adm must-gather --image=registry.redhat.io/rhmtc/openshift-migration-must-gather-rhel8:v1.4
```
3. Remove authentication keys and other sensitive information.
4. Create an archive file containing the contents of the **must-gather** data directory:

```
$ tar cvaf must-gather.tar.gz must-gather.local.<uid>/
```
5. Upload the compressed file as an attachment to your customer support case.

### 2.5.9. Rolling back a migration

You can roll back a migration by using the MTC web console or the CLI.

#### 2.5.9.1. Rolling back a migration in the MTC web console
You can roll back a migration by using the Migration Toolkit for Containers (MTC) web console.

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**.

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.

2.5.9.1.1. Rolling back a migration from the CLI

You can roll back a migration by creating a **MigMigration** custom resource (CR) from the CLI.

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: migration-rollback
   namespace: openshift-migration
   spec:
   EOF
   ```
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.

2.5.10. 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: ...
```
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)

2.5.11. Additional resources

- MTC workflow
- MTC custom resources
3.1. ABOUT THE MIGRATION TOOLKIT FOR CONTAINERS

You can migrate application workloads from OpenShift Container Platform 4.2 to 4.7 with the Migration Toolkit for Containers (MTC). MTC enables you to control the migration and to minimize application downtime.

**NOTE**

You can migrate between OpenShift Container Platform clusters of the same version, for example, from 4.2 to 4.2 or from 4.3 to 4.3, as long as the source and target clusters are configured correctly.

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

The MTC web console and API, based on Kubernetes custom resources, enable you to migrate stateful and stateless application workloads at the granularity of a namespace.

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.

You can use migration hooks to run Ansible playbooks at certain points during the migration. The hooks are added when you create a migration plan.

### 3.1.1. Migration Toolkit for Containers workflow

You use the Migration Toolkit for Containers (MTC) to migrate Kubernetes resources, persistent volume data, and internal container images from an OpenShift Container Platform source cluster to an OpenShift Container Platform 4.7 target cluster by using the MTC web console or the Kubernetes API.

The (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 volume claims that are linked to the persistent volumes of the namespace.

- Custom resources (CRs) and custom resource definitions (CRDs): The MTC automatically migrates any CRs that exist at the namespace level as well as the CRDs that are linked to those CRs.

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. In a restricted environment, you can use an internally hosted S3 storage repository. 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.

     ![Diagram of data migration process](image)

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

   - **Move**: MTC unmounts a remote volume, for example, NFS, from the source cluster, creates a PV resource on the target cluster pointing to the remote volume, and then mounts the remote volume on the target cluster. Applications running on the target cluster use the same remote volume that the source cluster was using. The remote volume must be accessible to the source and target clusters.

     **NOTE**
     Although the replication repository does not appear in this diagram, it is required for migration.
6. Run the migration plan, with one of the following options:

- **Stage** (optional) copies data to the target cluster without stopping the application. Staging can be run multiple times so that most of the data is copied to the target before migration. This minimizes the duration of the migration and application downtime.

- **Migrate** stops the application on the source cluster and recreates its resources on the target cluster. Optionally, you can migrate the workload without stopping the application.

3.1.2. Migration Toolkit for Containers custom resources

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

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

### 3.1.3. 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.

#### 3.1.3.1. File system copy method

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

**Table 3.1. File system copy method summary**

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

#### 3.1.3.2. 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.

AWS, Google Cloud Provider, and Microsoft Azure support the snapshot copy method.

**Table 3.2. Snapshot copy method summary**

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.1.3.3. Direct volume migration and direct image migration

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

Direct migration has significant performance benefits because it skips 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.

Direct migration uses Rsync to transfer the files.

**NOTE**

Direct image migration and direct volume migration have additional prerequisites.

3.1.4. About migration hooks

You can use migration hooks to run custom code at certain points during a migration with the Migration Toolkit for Containers (MTC). 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 using an Ansible playbook or 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 (CR). 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-rhel7:1.4. This image is based on the Ansible Runner image and includes python-openshift for Ansible Kubernetes resources and an updated oc binary.

Optional: You can use a custom Ansible runtime image containing additional Ansible modules or tools instead of the default image.

**Custom hook container**

You can create a custom hook container that includes Ansible playbooks or custom code.

### 3.2. INSTALLING AND UPGRADEING THE MIGRATION TOOLKIT FOR CONTAINERS

You can install the Migration Toolkit for Containers Operator on your OpenShift Container Platform 4.7 target cluster and 4.2 source cluster.

MTC is installed on the target cluster by default. You can install MTC on an OpenShift Container Platform 3 cluster or on a remote cluster.

**IMPORTANT**

You must install the same MTC version on all clusters.

#### 3.2.1. Installing the Migration Toolkit for Containers in a connected environment

You can install the Migration Toolkit for Containers (MTC) in a connected environment.

#### 3.2.1.1. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.7 target cluster

You can install the Migration Toolkit for Containers (MTC) on an OpenShift Container Platform 4.7 target cluster.

**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.
NOTE

Do not change the subscription approval option to Automatic. The Migration Toolkit for Containers version must be the same on the source and the target clusters.

4. Click Install.
   On the Installed Operators page, the Migration Toolkit for Containers Operator appears in the openshift-migration project with the status Succeeded.

5. Click Migration Toolkit for Containers Operator.

6. Under Provided APIs, locate the Migration Controller tile, and click Create Instance.

7. Click Create.

8. Click Workloads → Pods to verify that the MTC pods are running.

3.2.1.2. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.2 source cluster

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

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.

   NOTE

   Do not change the subscription approval option to Automatic. The Migration Toolkit for Containers version must be the same on the source and the target clusters.

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. Update the following parameters in in the migration-controller custom resource manifest:

   ```yaml
   spec:
   ...
   migration_controller: false
   ```
migration_ui: false

8. Click Create.

9. Click Workloads → Pods to verify that the MTC pods are running.

3.2.2. Installing the Migration Toolkit for Containers in a restricted environment

You can install the Migration Toolkit for Containers (MTC) in a restricted environment.

**IMPORTANT**

You must install the same MTC version on all clusters.

You can build a custom Operator catalog image for OpenShift Container Platform 4, push it to a local mirror image registry, and configure Operator Lifecycle Manager (OLM) to install the Migration Toolkit for Containers Operator from the local registry.

3.2.2.1. Disabling the default OperatorHub sources

Operator catalogs that source content provided by Red Hat and community projects are configured for OperatorHub by default during an OpenShift Container Platform installation.

**Procedure**

- Disable the sources for the default catalogs by adding `disableAllDefaultSources: true` to the `OperatorHub` object:

  ```
  $ oc patch OperatorHub cluster --type json \ 
  -p '{"op": "add", "path": "/spec/disableAllDefaultSources", "value": true}]
  ```

**TIP**

Alternatively, you can use the web console to manage catalog sources. From the Administration → Cluster Settings → Global Configuration → OperatorHub page, click the Sources tab, where you can create, delete, disable, and enable individual sources.

3.2.2.2. Pruning an index image

An index image, based on the Operator Bundle Format, is a containerized snapshot of an Operator catalog. You can prune an index of all but a specified list of packages, which creates a copy of the source index containing only the Operators that you want.

When configuring Operator Lifecycle Manager (OLM) to use mirrored content on restricted network OpenShift Container Platform clusters, use this pruning method if you want to only mirror a subset of Operators from the default catalogs.

For the steps in this procedure, the target registry is an existing mirror registry that is accessible by your workstation with unrestricted network access. This example also shows pruning the index image for the default `redhat-operators` catalog, but the process is the same for any index image.

**Prerequisites**
- Workstation with unrestricted network access
- **podman** version 1.9.3+
- **grpcurl**
- **opm** version 1.12.3+
- Access to a registry that supports **Docker v2-2**

**Procedure**

1. Authenticate with **registry.redhat.io**:
   
   ```
   $ podman login registry.redhat.io
   ```

2. Authenticate with your target registry:
   
   ```
   $ podman login <target_registry>
   ```

3. Determine the list of packages you want to include in your pruned index.
   
   a. Run the source index image that you want to prune in a container. For example:
      
      ```
      ```

   b. In a separate terminal session, use the **grpcurl** command to get a list of the packages provided by the index:
      
      ```
      $ grpcurl -plaintext localhost:50051 api.Registry/ListPackages > packages.out
      ```

   c. Inspect the **packages.out** file and identify which package names from this list you want to keep in your pruned index. For example:

   **Example snippets of packages list**

   ```
   ...
   {
   "name": "advanced-cluster-management"
   }
   ...
   {
   "name": "jaeger-product"
   }
   ```
In the terminal session where you executed the **podman run** command, press **Ctrl** and **C** to stop the container process.

4. Run the following command to prune the source index of all but the specified packages:

```
$ opm index prune \
  -f registry.redhat.io/redhat/redhat-operator-index:v4.7 1 \
  -p advanced-cluster-management,jaeger-product,quay-operator \n  [-i registry.redhat.io/openshift4/ose-operator-registry:v4.7] 3 \
  -t <target_registry>:/<namespace>/redhat-operator-index:v4.7 4
```

1. Index to prune.
2. Comma-separated list of packages to keep.
3. Required only for IBM Power Systems and IBM Z images: Operator Registry base image with the tag that matches the target OpenShift Container Platform cluster major and minor version.
4. Custom tag for new index image being built.

5. Run the following command to push the new index image to your target registry:

```
$ podman push <target_registry>:<port>/<namespace>/redhat-operator-index:v4.7
```

where `<namespace>` is any existing namespace on the registry. For example, you might create an **olm-mirror** namespace to push all mirrored content to.

### 3.2.2.3. Mirroring an Operator catalog

You can mirror the Operator content of a Red Hat-provided catalog, or a custom catalog, into a container image registry using the **oc adm catalog mirror** command. The target registry must support **Docker v2-2**. For a cluster on a restricted network, this registry can be one that the cluster has network access to, such as a mirror registry created during a restricted network cluster installation.

The **oc adm catalog mirror** command also automatically mirrors the index image that is specified during the mirroring process, whether it be a Red Hat-provided index image or your own custom-built index image, to the target registry. You can then use the mirrored index image to create a catalog source that allows Operator Lifecycle Manager (OLM) to load the mirrored catalog onto your OpenShift Container Platform cluster.

**Prerequisites**

- Workstation with unrestricted network access.
- **podman** version 1.9.3 or later.
- Access to mirror registry that supports **Docker v2-2**.
- Decide which namespace on your mirror registry you will use to store the mirrored Operator content. For example, you might create an `olm-mirror` namespace.

- If your mirror registry does not have Internet access, connect removable media to your workstation with unrestricted network access.

- If you are working with private registries, set the `REG_CREDS` environment variable to the file path of your registry credentials for use in later steps. For example, for the `podman` CLI:

  ```bash
  $ REG_CREDS=${XDG_RUNTIME_DIR}/containers/auth.json
  ```

Procedure

1. If you want to mirror a Red Hat-provided catalog, run the following command on your workstation with unrestricted network access to authenticate with `registry.redhat.io`:

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

2. The `oc adm catalog mirror` command extracts the contents of an index image to generate the manifests required for mirroring. The default behavior of the command generates manifests, then automatically mirrors all of the image content from the index image, as well as the index image itself, to your mirror registry. Alternatively, if your mirror registry is on a completely disconnected, or `airgapped`, host, you can first mirror the content to removable media, move the media to the disconnected environment, then mirror the content from the media to the registry.

   - **Option A:** If your mirror registry is on the same network as your workstation with unrestricted network access, take the following actions on your workstation:

     a. If your mirror registry requires authentication, run the following command to log in to the registry:

        ```bash
        $ podman login <mirror_registry>
        ```

     b. Run the following command to mirror the content:

        ```bash
        $ oc adm catalog mirror \
        <index_image> \ 1
        <mirror_registry>:<port>/<namespace> \ 2
        [-a '${REG_CREDS}'] \ 3
        [-insecure] \ 4
        [-index-filter-by-os='<platform>/<arch>'] \ 5
        [-manifests-only] \ 6
        ```

        1 Specify the index image for the catalog you want to mirror. For example, this might be a pruned index image that you created previously, or one of the source index images for the default catalogs, such as `registry.redhat.io/redhat/redhat-operator-index:v4.7`.

        2 Specify the target registry and namespace to mirror the Operator content to, where `<namespace>` is any existing namespace on the registry. For example, you might create an `olm-mirror` namespace to push all mirrored content to.

        3 Optional: If required, specify the location of your registry credentials file.
Optional: If you do not want to configure trust for the target registry, add the `--insecure` flag.

Optional: Specify which platform and architecture of the index image to select when multiple variants are available. Images are passed as `'<platform>/<arch>[/<variant>]'`. This does not apply to images referenced by the index. Valid values are `linux/amd64`, `linux/ppc64le`, and `linux/s390x`.

Optional: Generate only the manifests required for mirroring, and do not actually mirror the image content to a registry. This option can be useful for reviewing what will be mirrored, and it allows you to make any changes to the mapping list if you require only a subset of packages. You can then use the `mapping.txt` file with the `oc image mirror` command to mirror the modified list of images in a later step. This flag is intended for only advanced selective mirroring of content from the catalog; the `opm index prune` command, if you used it previously to prune the index image, is suitable for most catalog management use cases.

Example output

```
src image has index label for database path: /database/index.db
using database path mapping: /database/index.db:/tmp/153048078
wrote database to /tmp/153048078
...  
wrote mirroring manifests to manifests-redhat-operator-index-1614211642
```

1. Directory for the temporary `index.db` database generated by the command.
2. Record the manifests directory name that is generated. This directory name is used in a later step.

- **Option B: If your mirror registry is on a disconnected host**, take the following actions.
  a. Run the following command on your workstation with unrestricted network access to mirror the content to local files:

```
$ oc adm catalog mirror
  <index_image>  
  file:///local/index  
  [-a ${REG_CREDS}]  
  [--insecure]
```

1. Specify the index image for the catalog you want to mirror. For example, this might be a pruned index image that you created previously, or one of the source index images for the default catalogs, such as `registry.redhat.io/redhat/redhat-operator-index:v4.7`.
2. Mirrors content to local files in your current directory.

**Example output**

```
...  
info: Mirroring completed in 5.93s (5.915MB/s)
```
Record the manifests directory name that is generated. This directory name is used in a later step.

Record the expanded file:// path that based on your provided index image. This path is used in a later step.

b. Copy the v2/ directory that is generated in your current directory to removable media.

c. Physically remove the media and attach it to a host in the disconnected environment that has access to the mirror registry.

d. If your mirror registry requires authentication, run the following command on your host in the disconnected environment to log in to the registry:

```
$ podman login <mirror_registry>
```

e. Run the following command from the parent directory containing the v2/ directory to upload the images from local files to the mirror registry:

```
$ oc adm catalog mirror \
   file://local/index/<repo>/<index_image>:<tag> \
   <mirror_registry>:<port>/<namespace> \
   [-a ${REG_CREDS}] \
   [--insecure]
```

1. Specify the file:// path from the previous command output.

2. Specify the target registry and namespace to mirror the Operator content to, where <namespace> is any existing namespace on the registry. For example, you might create an olm-mirror namespace to push all mirrored content to.

3. After mirroring the content to your registry, inspect the manifests directory that is generated in your current directory.

**NOTE**

The manifests directory name is used in a later step.

If you mirrored content to a registry on the same network in the previous step, the directory name takes the following form:

```
manifests-<index_image_name>-<random_number>
```

If you mirrored content to a registry on a disconnected host in the previous step, the directory name takes the following form:
The manifests directory contains the following files, some of which might require further modification:

- The `catalogSource.yaml` file is a basic definition for a `CatalogSource` object that is pre-populated with your index image tag and other relevant metadata. This file can be used as is or modified to add the catalog source to your cluster.

  **IMPORTANT**

  If you mirrored the content to local files, you must modify your `catalogSource.yaml` file to remove any backslash (`/`) characters from the `metadata.name` field. Otherwise, when you attempt to create the object, it fails with an “invalid resource name” error.

- The `imageContentSourcePolicy.yaml` file defines an `ImageContentSourcePolicy` object that can configure nodes to translate between the image references stored in Operator manifests and the mirrored registry.

  **NOTE**

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

- The `mapping.txt` file contains all of the source images and where to map them in the target registry. This file is compatible with the `oc image mirror` command and can be used to further customize the mirroring configuration.

  **IMPORTANT**

  If you used the `--manifests-only` flag during the mirroring process and want to further trim the subset of packages to be mirrored, see the steps in the “Mirroring a Package Manifest Format catalog image” procedure about modifying your `mapping.txt` file and using the file with the `oc image mirror` command. After following those further actions, you can continue this procedure.

4. On a host with access to the disconnected cluster, create the `ImageContentSourcePolicy` object by running the following command to specify the `imageContentSourcePolicy.yaml` file in your manifests directory:

   ```bash
   $ oc create -f <path/to/manifests/dir>/imageContentSourcePolicy.yaml
   ```

   where `<path/to/manifests/dir>` is the path to the manifests directory for your mirrored content.

You can now create a `CatalogSource` object to reference your mirrored index image and Operator content.

### 3.2.2.4. Creating a catalog from an index image

You can now create a `CatalogSource` object to reference your mirrored index image and Operator content.
You can create an Operator catalog from an index image and apply it to an OpenShift Container Platform cluster for use with Operator Lifecycle Manager (OLM).

**Prerequisites**

- An index image built and pushed to a registry.

**Procedure**

1. Create a **CatalogSource** object that references your index image.
   a. Modify the following to your specifications and save it as a `catalogSource.yaml` file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: my-operator-catalog
  namespace: openshift-marketplace
spec:
  sourceType: grpc
  displayName: My Operator Catalog
  publisher: <publisher_name>
  updateStrategy:
    registryPoll:
      interval: 30m
```
   <.> Specify your index image. <.> Specify your name or an organization name publishing the catalog. <.> Catalog sources can automatically check for new versions to keep up to date.
   b. Use the file to create the **CatalogSource** object:

```
$ oc apply -f catalogSource.yaml
```

2. Verify the following resources are created successfully.
   a. Check the pods:

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

**Example output**

```
NAME                                    READY   STATUS    RESTARTS  AGE
my-operator-catalog-6njx6               1/1     Running   0         28s
marketplace-operator-d9f549946-96sgr    1/1     Running   0         26h
```

b. Check the catalog source:

```
$ oc get catalogsource -n openshift-marketplace
```

**Example output**

```
NAME                  DISPLAY               TYPE PUBLISHER  AGE
my-operator-catalog   My Operator Catalog   grpc            5s
```
c. Check the package manifest:

```bash
$ oc get packagemanifest -n openshift-marketplace
```

**Example output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>CATALOG</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>jaeger-product</td>
<td>My Operator Catalog</td>
<td>93s</td>
</tr>
</tbody>
</table>

You can now install the Operators from the OperatorHub page on your OpenShift Container Platform web console.

### 3.2.2.5. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.7 target cluster in a restricted environment

You can install the Migration Toolkit for Containers (MTC) on an OpenShift Container Platform 4.7 target cluster.

**Prerequisites**

- You must be logged in as a user with `cluster-admin` privileges on all clusters.
- You must create a custom Operator catalog and push it to a mirror registry.
- You must configure Operator Lifecycle Manager to install the Migration Toolkit for Containers Operator from the mirror 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**.
   
   **NOTE**
   
   Do not change the subscription approval option to **Automatic**. The Migration Toolkit for Containers version must be the same on the source and the target clusters.

4. Click **Install**.  
   On the **Installed Operators** page, the **Migration Toolkit for Containers Operator** appears in the `openshift-migration` project with the status **Succeeded**.

5. Click **Migration Toolkit for Containers Operator**.
6. Under **Provided APIs**, locate the **Migration Controller** tile, and click **Create Instance**.
7. Click **Create**.
8. Click **Workloads** → **Pods** to verify that the MTC pods are running.
3.2.2.6. Installing the Migration Toolkit for Containers on an OpenShift Container Platform 4.2 source cluster in a restricted environment

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

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges on all clusters.
- You must create a custom Operator catalog and push it to a mirror registry.
- You must configure Operator Lifecycle Manager to install the Migration Toolkit for Containers Operator from the mirror 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.

   **NOTE**

   Do not change the subscription approval option to Automatic. The Migration Toolkit for Containers version must be the same on the source and the target clusters.

4. Click Install.
   On the Installed Operators page, the Migration Toolkit for Containers Operator appears in the openshift-migration project with the status Succeeded.
5. Click Migration Toolkit for Containers Operator.
6. Under Provided APIs, locate the Migration Controller tile, and click Create Instance.
7. Click Create.
8. Click Workloads → Pods to verify that the MTC pods are running.

3.2.3. Upgrading the Migration Toolkit for Containers

You can upgrade the Migration Toolkit for Containers (MTC) by using the OpenShift Container Platform web console.

**IMPORTANT**

You must ensure that the same MTC version is installed on all clusters.

If you are upgrading MTC version 1.3, you must perform an additional procedure to update the MigPlan custom resource (CR).
3.2.3.1. Upgrading the Migration Toolkit for Containers on an OpenShift Container Platform 4 cluster

You can upgrade the Migration Toolkit for Containers (MTC) on an OpenShift Container Platform 4 cluster by using the OpenShift Container Platform web console.

**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 **Migration Toolkit for Containers Operator**.

8. Under **Provided APIs**, locate the **Migration Controller** tile, and click **Create Instance**.

9. If you are upgrading MTC on a **source** cluster, update the following parameters in the **MigrationController** custom resource (CR) manifest:

   ```yaml
   spec:
     ...
     migration_controller: false
     migration_ui: false
   ```

   You do not need to update the **MigrationController** CR manifest on the target cluster.

10. Click **Create**.

11. Click **Workloads → Pods** to verify that the MTC pods are running.

3.2.3.2. Upgrading MTC 1.3 to 1.4

If you are upgrading Migration Toolkit for Containers (MTC) version 1.3.x to 1.4, 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 have MTC 1.3 installed.
- You must be logged in as a user with `cluster-admin` privileges.

Procedure

1. Log in to the cluster on which the MigrationController pod is running.
2. Get the MigPlan CR manifest:
   ```bash
   $ oc get migplan <migplan> -o yaml -n openshift-migration
   
   $ oc replace -f migplan.yaml -n openshift-migration
   
   $ oc get migplan <migplan> -o yaml -n openshift-migration
   ```
3. Update the following parameter values and save the file as `migplan.yaml`:
   ```yaml
   ... 
   spec:
   indirectImageMigration: true
   indirectVolumeMigration: true
   ```
4. Replace the MigPlan CR manifest to apply the changes:
   ```bash
   $ oc replace -f migplan.yaml -n openshift-migration
   ```
5. Get the updated MigPlan CR manifest to verify the changes:
   ```bash
   $ oc get migplan <migplan> -o yaml -n openshift-migration
   ```

3.3. CONFIGURING OBJECT STORAGE FOR 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:

- Multi-Cloud Object Gateway (MCG)
- Amazon Web Services (AWS) S3
- Google Cloud Provider (GCP)
- Microsoft Azure
- Generic S3 object storage, for example, Minio or Ceph S3

In a restricted environment, you can create an internally hosted replication repository.
Prerequisites

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

3.3.1. Configuring a Multi-Cloud Object Gateway storage bucket as a replication repository

You can install the OpenShift Container Storage Operator and configure a Multi-Cloud Object Gateway (MCG) storage bucket as a replication repository for the Migration Toolkit for Containers (MTC).

3.3.1.1. Installing the OpenShift Container Storage Operator

You can install the OpenShift Container Storage Operator from OperatorHub.

**Procedure**

1. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
2. Use **Filter by keyword** (in this case, OCS) to find the **OpenShift Container Storage Operator**.
3. Select the **OpenShift Container Storage Operator** and click **Install**.
4. Select an **Update Channel**, **Installation Mode**, and **Approval Strategy**.
5. Click **Install**. On the **Installed Operators** page, the **OpenShift Container Storage Operator** appears in the openshift-storage project with the status **Succeeded**.

3.3.1.2. Creating the Multi-Cloud Object Gateway storage bucket

You can create the Multi-Cloud Object Gateway (MCG) storage bucket’s custom resources (CRs).

**Procedure**

1. Log in to the OpenShift Container Platform cluster:

   ```bash
   $ oc login -u <username>
   ```

2. Create the **Noobaa** CR configuration file, **noobaa.yml**, with the following content:

   ```yaml
   apiVersion: noobaa.io/v1alpha1
   kind: NooBaaS
   metadata:
     name: noobaa
     namespace: openshift-storage
   spec:
     dbResources:
     requests:
       cpu: 0.5
       memory: 1Gi
     coreResources:
   ```
requests:
  cpu: 0.5 2
  memory: 1Gi

For a very small cluster, you can change the cpu value to 0.1.

3. Create the NooBaa object:

   $ oc create -f noobaa.yml

4. Create the BackingStore CR configuration file, bs.yml, with the following content:

   apiVersion: noobaa.io/v1alpha1
   kind: BackingStore
   metadata:
     finalizers:
     - noobaa.io/finalizer
     labels:
       app: noobaa
       name: mcg-pv-pool-bs
       namespace: openshift-storage
   spec:
     pvPool:
       numVolumes: 3 1
       resources:
         requests:
           storage: 50Gi 2
       storageClass: gp2 3
       type: pv-pool

1 Specify the number of volumes in the persistent volume pool.
2 Specify the size of the volumes.
3 Specify the storage class.

5. Create the BackingStore object:

   $ oc create -f bs.yml

6. Create the BucketClass CR configuration file, bc.yml, with the following content:

   apiVersion: noobaa.io/v1alpha1
   kind: BucketClass
   metadata:
     labels:
       app: noobaa
       name: mcg-pv-pool-bc
       namespace: openshift-storage
   spec:
     placementPolicy:
       tiers:
7. Create the **BucketClass** object:

```bash
$ oc create -f bc.yml
```

8. Create the **ObjectBucketClaim** CR configuration file, **obc.yml**, with the following content:

```yaml
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: migstorage
  namespace: openshift-storage
spec:
  bucketName: migstorage
  storageClassName: openshift-storage.noobaa.io
  additionalConfig:
    bucketclass: mcg-pv-pool-bc

$ oc create -f obc.yml
```

1. Record the bucket name for adding the replication repository to the MTC web console.

9. Create the **ObjectBucketClaim** object:

```bash
$ oc create -f obc.yml
```

10. Watch the resource creation process to verify that the **ObjectBucketClaim** status is **Bound**:

```bash
$ watch -n 30 'oc get -n openshift-storage objectbucketclaim migstorage -o yaml'
```

This process can take five to ten minutes.

11. Obtain and record the following values, which are required when you add the replication repository to the MTC web console:

- **S3 endpoint**:
  ```bash
  $ oc get route -n openshift-storage s3
  ```

- **S3 provider access key**:
  ```bash
  $ oc get secret -n openshift-storage migstorage -o go-template='{.data.AWS_ACCESS_KEY_ID }' | base64 --decode
  ```

- **S3 provider secret access key**:
  ```bash
  $ oc get secret -n openshift-storage migstorage -o go-template='{.data.AWS_SECRET_ACCESS_KEY }' | base64 --decode
  ```

### 3.3.2. Configuring an AWS S3 storage bucket as a replication repository
You can configure an AWS S3 storage bucket as a replication repository for the Migration Toolkit for Containers (MTC).

**Prerequisites**

- The AWS S3 storage bucket must be accessible to the source and target clusters.
- You must have the AWS CLI installed.
- 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. Create an AWS S3 bucket:

   ```bash
   $ aws s3api create-bucket \
     --bucket <bucket_name> \
     --region <bucket_region>
   ```

   1. Specify your S3 bucket name.
   2. Specify your S3 bucket region, for example, **us-east-1**.

2. Create the IAM user **velero**:

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

3. Create an EC2 EBS snapshot policy:

   ```bash
   $ cat > velero-ec2-snapshot-policy.json <<EOF
   {
   "Version": "2012-10-17",
   "Statement": [
   {
   "Effect": "Allow",
   "Action": [
   "ec2:DescribeVolumes",
   "ec2:DescribeSnapshots",
   "ec2:CreateTags",
   "ec2:CreateVolume",
   "ec2:CreateSnapshot",
   "ec2:DeleteSnapshot"
   ],
   "Resource": "*"
   }
   }
   ```
Create an AWS S3 access policy for one or for all S3 buckets:

$ cat > velero-s3-policy.json <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:DeleteObject",
                "s3:PutObject",
                "s3:AbortMultipartUpload",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:s3:::<bucket_name>/*"  
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:ListBucketMultipartUploads"
            ],
            "Resource": [  
                "arn:aws:s3:::<bucket_name>"
            ]
        }
    ]
}
EOF

To grant access to a single S3 bucket, specify the bucket name. To grant access to all AWS S3 buckets, specify * instead of a bucket name as in the following example:

Example output

"Resource": [  
    "arn:aws:s3:::*"
]

Attach the EC2 EBS policy to velero:

$ aws iam put-user-policy \  
   --user-name velero \  
   --policy-name velero-ebs \  
   --policy-document file://velero-ec2-snapshot-policy.json
6. Attach the AWS S3 policy to velero:

```
$ aws iam put-user-policy \
   --user-name velero \
   --policy-name velero-s3 \
   --policy-document file://velero-s3-policy.json
```

7. Create an access key for velero:

```
$ aws iam create-access-key --user-name velero
{
  "AccessKey": {
    "UserName": "velero",
    "Status": "Active",
    "CreateDate": "2017-07-31T22:24:41.576Z",
    "SecretAccessKey": "<AWS_SECRET_ACCESS_KEY>", ①
    "AccessKeyId": "<AWS_ACCESS_KEY_ID>" ②
  }
}
```

①② Record the AWS_SECRET_ACCESS_KEY and the AWS_ACCESS_KEY_ID for adding the AWS repository to the MTC web console.

### 3.3.3. Configuring a Google Cloud Provider storage bucket as a replication repository

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

**Prerequisites**

- The GCP storage bucket must be accessible to the source and target clusters.
- You must have gsutil installed.
- 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 gsutil:

```
$ gsutil init
```

**Example output**

Welcome! This command will take you through the configuration of gcloud.
Your current configuration has been set to: [default]
To continue, you must login. Would you like to login (Y/n)?

2. Set the `BUCKET` variable:

   ```sh
   $ BUCKET=<bucket_name>
   ```

   Specify your bucket name.

3. Create a storage bucket:

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

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

   ```sh
   $ PROJECT_ID=`gcloud config get-value project`
   ```

5. Create a `velero` IAM service account:

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

6. Create the `SERVICE_ACCOUNT_EMAIL` variable:

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

7. Create the `ROLE_PERMISSIONS` variable:

   ```sh
   $ 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
   )
   ```

8. Create the `velero.server` custom role:

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

9. Add IAM policy binding to the project:
Update the IAM service account:

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

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

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

3.3.4. Configuring a Microsoft Azure Blob storage container as a replication repository

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

**Prerequisites**

- You must have an Azure storage account.
- 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. Set the `AZURE_RESOURCE_GROUP` variable:

   ```bash
   $ AZURE_RESOURCE_GROUP=Velero_Backups
   ```

2. Create an Azure resource group:

   ```bash
   $ az group create -n $AZURE_RESOURCE_GROUP --location <CentralUS>  # Specify your location.
   ```

3. Set the `AZURE_STORAGE_ACCOUNT_ID` variable:

   ```bash
   $ AZURE_STORAGE_ACCOUNT_ID=velerobackups
   ```

4. Create an Azure storage account:
5. Set the `BLOB_CONTAINER` variable:

```bash
$ BLOB_CONTAINER=velero
```

6. Create an Azure Blob storage container:

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

7. Create a service principal and credentials for `velero`:

```bash
$ 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 --display-name "velero" --role "Contributor" --query 'password' -o tsv` \
AZURE_CLIENT_ID=`az ad sp list --display-name "velero" --query '[0].appId' -o tsv`
```

8. Save the service principal credentials in the `credentials-velero` file:

```bash
$ 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
```

### 3.4. MIGRATING YOUR APPLICATIONS

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

#### 3.4.1. Prerequisites

The Migration Toolkit for Containers (MTC) has the following prerequisites:

- You must be logged in as a user with `cluster-admin` privileges on all clusters.
- The MTC version must be the same on all clusters.
- Clusters:
- The source cluster must be upgraded to the latest MTC z-stream release.
- The cluster on which the migration-controller pod is running must have unrestricted network access to the other clusters.
- The clusters must have unrestricted network access to each other.
- The clusters must have unrestricted network access to the replication repository.
- The clusters must be able to communicate using OpenShift routes on port 443.
- The clusters must have no critical conditions.
- The clusters must be in a ready state.

- Volume migration:
  - The persistent volumes (PVs) must be valid.
  - The PVs must be bound to persistent volume claims.
  - If you copy the PVs by using the move method, the clusters must have unrestricted network access to the remote volume.
  - If you copy the PVs by using the snapshot copy method, the following prerequisites apply:
    - The cloud provider must support snapshots.
    - The volumes must have the same cloud provider.
    - The volumes must be located in the same geographic region.
    - The volumes must have the same storage class.

- If you perform a direct volume migration in a proxy environment, you must configure an Stunnel TCP proxy.

- If you perform a direct image migration, you must expose the internal registry of the source cluster to external traffic.

### 3.4.1.1. Creating a CA certificate bundle file

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> | sed -ne '/-BEGIN CERTIFICATE-/,-END CERTIFICATE-/p' > <ca_bundle.cert>
```

1. Specify the host FQDN and port of the endpoint, for example, `api.my-cluster.example.com:6443`.  
2. `ca_bundle.cert` is the path to the file where the CA certificate bundle is saved.
Specify the name of the CA bundle file.

### 3.4.1.2. Configuring a proxy for direct volume migration

If you are performing direct volume migration from a source cluster behind a proxy, you must configure an Stunnel proxy in the `MigrationController` custom resource (CR). Stunnel creates a transparent tunnel between the source and target clusters for the TCP connection without changing the certificates.

**NOTE**

Direct volume migration supports only one proxy. The source cluster cannot access the route of the target cluster if the target cluster is also behind a proxy.

**Prerequisites**

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

**Procedure**

1. Log in to the cluster on which the `MigrationController` pod runs.
2. Get the `MigrationController` CR manifest:
   ```bash
   $ oc get migrationcontroller <migration_controller> -n openshift-migration
   ```
3. Add the `stunnel_tcp_proxy` parameter:
   ```yaml
   apiVersion: migration.openshift.io/v1alpha1
   kind: MigrationController
   metadata:
     name: migration-controller
     namespace: openshift-migration
   spec:
     stunnel_tcp_proxy: <stunnel_proxy> # 1
   ```
   1 Specify the Stunnel proxy: `http://<user_name>:<password>@<ip_address>:<port>`.
4. Save the manifest as `migration-controller.yaml`.
5. Apply the updated manifest:
   ```bash
   $ oc replace -f migration-controller.yaml -n openshift-migration
   ```

### 3.4.1.3. 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.

### 3.4.1.3.1. Ansible modules

You can use the Ansible **shell** module to run **oc** commands.

**Example shell module**

```yaml
- 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**

```yaml
- 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**

```yaml
- 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
```
3.4.1.3.2. 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 plug-in.

Example environment variables

```yaml
- hosts: localhost
gather_facts: false
tasks:
- set_fact:
  namespaces: "{{ (lookup('env', 'migration_namespaces')).split(',') }}"

- debug:
  msg: "{{ item }}"
  with_items: "{{ namespaces }}"

- debug:
  msg: "{{ lookup('env', 'migplan_name') }}"
```

3.4.1.4. Additional resources

- About migration hooks
- MigHook custom resource
- MigPlan custom resource

3.4.2. 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.

3.4.2.1. 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**.

### 3.4.2.2. 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.

**Procedure**

1. Log in to the cluster.

2. Obtain the **migration-controller** service account token:

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

   **Example output**

   eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJwaWNpbmFyaWNldCBvcGVyIiwiY3Npc3NfZG9tYWluIjoiYm8iLCJ6eXJvdWQiOiJhY2NvdW50cyJ9.

3. In the MTC web console, click **Clusters**.
4. Click **Add cluster**.

5. Fill in the following fields:

- **Cluster name**: The cluster name can contain lower-case letters (a-z) and numbers (0-9). It must not contain spaces or international characters.
- **URL**: Specify the API server URL, for example, `https://<www.example.com>:8443`.
- **Service account token**: Paste the `migration-controller` service account token.
- **Exposed route host to image registry**: If you are using direct image migration, specify the exposed route to the image registry of the source cluster, for example, `www.example.apps.cluster.com`. You can specify a port. The default port is **5000**.
- **Azure cluster**: You must select this option if you use Azure snapshots to copy your data.
- **Azure resource group**: This field is displayed if **Azure cluster** is selected. Specify the Azure resource group.
- **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.

**3.4.2.3. Adding a replication repository to the MTC web console**

You can add an object storage bucket as a replication repository to the Migration Toolkit for Containers (MTC) web console.

**Prerequisites**

- You must configure an object storage bucket for migrating the data.

**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 AWS S3, MCG, and generic S3 providers:
     - **Replication repository name**: Specify the replication repository name in the MTC web console.
     - **S3 bucket name**: Specify the name of the S3 bucket you created.
     - **S3 bucket region**: Specify the S3 bucket region. **Required** for AWS S3. **Optional** for other S3 providers.
S3 endpoint: Specify the URL of the S3 service, not the bucket, for example, 

S3 provider access key: Specify the <AWS_SECRET_ACCESS_KEY> for AWS or the S3 provider access key for MCG.

S3 provider secret access key: Specify the <AWS_ACCESS_KEY_ID> for AWS or the S3 provider secret access key for MCG.

Require SSL verification: Clear this check box if you are using a generic S3 provider.

If you use a custom CA bundle, click Browse and browse to the Base64-encoded CA bundle 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.

4. Click Add repository and wait for connection validation.

5. Click Close.

The new repository appears in the Replication repositories list.

3.4.2.4. 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, the MigCluster custom resource manifest of the source cluster must specify the exposed route of the internal image registry.
Procedure

1. In the MTC web console, click Migration plans.

2. Click Add migration plan.

3. Enter the Plan name and click Next.
   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.

5. Select a Target cluster.

6. Select a Replication repository.

7. Select the projects to be migrated and click Next.

8. Select a Source cluster, a Target cluster, and a Repository, and click Next.

9. On the Namespaces page, select the projects to be migrated and click Next.

10. On the Persistent volumes page, click 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.

11. Click Next.

12. On the Copy options page, 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.

13. 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.

14. Select a Target storage class.
    If you selected Filesystem copy, you can change the target storage class.

15. Click Next.

16. 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
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.

17. Click **Next**.

18. Optional: On the **Hooks** page, 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**.

19. Click **Finish**.
The migration plan is displayed in the **Migration plans** list.

### 3.4.2.5. Running a migration plan in the MTC web console

You can stage or 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:
Procedure

1. Log in to the source cluster.
2. Delete old images:
   $$\texttt{oc adm prune images}$$
3. Log in to the MTC web console and click Migration plans.
4. Click the Options menu next to a migration plan and select Stage to copy data from the source cluster to the target cluster without stopping the application. You can run Stage multiple times to reduce the actual migration time.
5. When you are ready to migrate the application workload, the Options menu beside a migration plan and select Migrate.
6. Optional: In the Migrate window, you can select Do not stop applications on the source cluster during migration.
7. Click Migrate.
8. 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.

3.4.3. Migrating your applications from the command line

You can migrate your applications on the command line by using the MTC custom resources (CRs).

You can migrate applications from a local cluster to a remote cluster, from a remote cluster to a local cluster, and between remote clusters.

MTC terminology
The following terms are relevant for configuring clusters:

- **host** cluster:
  - The *migration-controller* pod runs on the **host** cluster.
  - A **host** cluster does not require an exposed secure registry route for direct image migration.

- **Local cluster**: The local cluster is often the same as the **host** cluster but this is not a requirement.

- **Remote cluster**:
  - A remote cluster must have an exposed secure registry route for direct image migration.
  - A remote cluster must have a **Secret CR** containing the *migration-controller* service account token.

The following terms are relevant for performing a migration:

- **Source cluster**: Cluster from which the applications are migrated.

- **Destination cluster**: Cluster to which the applications are migrated.

### 3.4.3.1. Migrating your applications with the Migration Toolkit for Containers API

You can migrate your applications on the command line with the Migration Toolkit for Containers (MTC) API.

You can migrate applications from a local cluster to a remote cluster, from a remote cluster to a local cluster, and between remote clusters.

This procedure describes how to perform indirect migration and direct 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 migration**: Images or volumes are copied directly from the source cluster to the destination cluster. Direct image migration and direct volume migration have significant performance benefits.

You create the following custom resources (CRs) to perform a migration:

- **MigCluster CR**: Defines a **host**, local, or remote cluster
  The *migration-controller* pod runs on the **host** cluster.

- **Secret CR**: Contains credentials for a remote cluster or storage

- **MigStorage CR**: Defines a replication repository
  Different storage providers require different parameters in the **MigStorage** CR manifest.

- **MigPlan CR**: Defines a migration plan

- **MigMigration CR**: Performs a migration defined in an associated **MigPlan**
  You can create multiple **MigMigration CRs** for a single **MigPlan** CR for the following purposes:
To perform stage migrations, which copy most of the data without stopping the application, before running a migration. Stage migrations improve the performance of the migration.

- To cancel a migration in progress
- To roll back a completed migration

**Prerequisites**

- You must have `cluster-admin` privileges for all clusters.
- You must install the OpenShift Container Platform CLI (`oc`).
- You must install the Migration Toolkit for Containers Operator on all clusters.
- The version of the installed Migration Toolkit for Containers Operator must be the same on all clusters.
- You must configure an object storage as a replication repository.
- If you are using direct image migration, you must expose a secure registry route on all remote clusters.
- If you are using direct volume migration, the source cluster must not have an HTTP proxy configured.

**Procedure**

1. Create a `MigCluster` CR manifest for the host cluster called `host-cluster.yaml`:

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigCluster
metadata:
  name: host
  namespace: openshift-migration
spec:
  isHostCluster: true
```

2. Create a `MigCluster` CR for the host cluster:

   ```bash
   $ oc create -f host-cluster.yaml -n openshift-migration
   ```

3. Create a `Secret` CR manifest for each remote cluster called `cluster-secret.yaml`:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: <cluster_secret>
  namespace: openshift-config
type: Opaque
data:
  saToken: <sa_token>  
```

1. Specify the base64-encoded `migration-controller` service account (SA) token of the remote cluster.
You can obtain the SA token by running the following command:

```
$ oc sa get-token migration-controller -n openshift-migration | base64 -w 0
```

4. Create a `Secret` CR for each remote cluster:

```
$ oc create -f cluster-secret.yaml
```

5. Create a `MigCluster` CR manifest for each remote called `remote-cluster.yaml`:

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigCluster
metadata:
  name: <remote_cluster>
  namespace: openshift-migration
spec:
  exposedRegistryPath: <exposed_registry_route> ①
  insecure: false ②
  isHostCluster: false
  serviceAccountSecretRef:
    name: <remote_cluster_secret> ③
    namespace: openshift-config
  url: <remote_cluster_url> ④
```

① Optional: Specify the exposed registry route, for example, `docker-registry-default.apps.example.com` if you are using direct image migration.

② SSL verification is enabled if `false`. CA certificates are not required or checked if `true`.

③ Specify the `Secret` CR of the remote cluster.

④ Specify the URL of the remote cluster.

6. Create a `MigCluster` CR for each remote cluster:

```
$ oc create -f remote-cluster.yaml -n openshift-migration
```

7. Verify that all clusters are in a `Ready` state:

```
$ oc describe cluster <cluster_name>
```

8. Create a `Secret` CR manifest for the replication repository called `storage-secret.yaml`:

```yaml
apiVersion: v1
kind: Secret
metadata:
  namespace: openshift-config
  name: <migstorage_creds>
  type: Opaque
data:
  aws-access-key-id: <key_id_base64> ①
  aws-secret-access-key: <secret_key_base64> ②
```

① Specify the access key ID.

② Specify the secret access key.
CHAPTER 3. MIGRATING FROM OPENSOURCE CONTAINER PLATFORM 4.2 AND LATER

1. Specify the key ID in base64 format.

2. Specify the secret key in base64 format.

AWS credentials are base64-encoded by default. If you are using another storage provider, you must encode your credentials by running the following command with each key:

```
$ echo -n "<key>" | base64 -w 0
```

1. Specify the key ID or the secret key. Both keys must be base64-encoded.

9. Create the Secret CR for the replication repository:

```
$ oc create -f storage-secret.yaml
```

10. Create a MigStorage CR manifest for the replication repository called `migstorage.yaml`:

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigStorage
metadata:
  name: <storage_name>
  namespace: openshift-migration
spec:
  backupStorageConfig:
    awsBucketName: <bucket_name>
    credsSecretRef:
      name: <storage_secret_ref>
      namespace: openshift-config
  backupStorageProvider: <storage_provider_name>
  volumeSnapshotConfig:
    credsSecretRef:
      name: <storage_secret_ref>
      namespace: openshift-config
  volumeSnapshotProvider: <storage_provider_name>
```

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.

11. Create the MigStorage CR:

```
$ oc create -f migstorage.yaml -n openshift-migration
```
12. Verify that the \texttt{MigStorage} CR is in a \textbf{Ready} state:
   ```
   $ oc describe migstorage <migstorage_name>
   ```

13. Create a \textbf{MigPlan} CR manifest called \texttt{migplan.yaml}:

   ```
   apiVersion: migration.openshift.io/v1alpha1
   kind: MigPlan
   metadata:
     name: <migration_plan>
     namespace: openshift-migration
   spec:
     destMigClusterRef:
       name: host
       namespace: openshift-migration
     indirectImageMigration: true
     indirectVolumeMigration: true
     migStorageRef:
       name: <migstorage_ref>
       namespace: openshift-migration
     namespaces:
       - <application_namespace>
     srcMigClusterRef:
       name: <remote_cluster_ref>
       namespace: openshift-migration
   ```

   \begin{enumerate}
   \item Direct image migration is enabled if \textbf{false}.
   \item Direct volume migration is enabled if \textbf{false}.
   \item Specify the name of the \texttt{MigStorage} CR instance.
   \item Specify one or more namespaces to be migrated.
   \item Specify the name of the source cluster \texttt{MigCluster} instance.
   \end{enumerate}

14. Create the \textbf{MigPlan} CR:
   ```
   $ oc create -f migplan.yaml -n openshift-migration
   ```

15. View the \textbf{MigPlan} instance to verify that it is in a \textbf{Ready} state:
   ```
   $ oc describe migplan <migplan_name> -n openshift-migration
   ```

16. Create a \textbf{MigMigration} CR manifest called \texttt{migmigration.yaml}:

   ```
   apiVersion: migration.openshift.io/v1alpha1
   kind: MigMigration
   metadata:
     name: <migmigration_name>
     namespace: openshift-migration
   spec:
     migPlanRef:
   ```
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**.

17. Create the **MigMigration** CR to start the migration defined in the **MigPlan** CR:

   ```bash
   $ oc create -f migmigration.yaml -n openshift-migration
   ``

18. Verify the progress of the migration by watching the **MigMigration** CR:

   ```bash
   $ oc watch migmigration <migmigration_name> -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=djongo
Annotations: openshift.io/touch: e99f9083-6567-11eb-8420-0a580a81020c
API Version: migration.openshift.io/v1alpha1
Kind:       MigMigration

Spec:
Mig Plan Ref:
    Name: my_application
    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
```
3.4.3.2. MTC custom resource manifests

Migration Toolkit for Containers (MTC) uses the following custom resource (CR) manifests to create CRs for migrating applications.

3.4.3.2.1. DirectImageMigration

The DirectImageMigration CR copies images directly from the source cluster to the destination cluster.
CHAPTER 3. MIGRATING FROM OPENSOURCE CONTAINER PLATFORM 4.2 AND LATER

apiVersion: migration.openshift.io/v1alpha1
kind: DirectImageMigration
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
name: <directimagemigration_name>
spec:
  srcMigClusterRef:
    name: <source_cluster_ref>
    namespace: openshift-migration
destMigClusterRef:
    name: <destination_cluster_ref>
    namespace: openshift-migration
namespaces:
  - <namespace>

Specify the MigCluster CR name of the source cluster.

Specify the MigCluster CR name of the destination cluster.

Specify one or more namespaces containing images to be migrated.

3.4.3.2.2. 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: directimagestreammigration_name
spec:
  srcMigClusterRef:
    name: <source_cluster_ref>
    namespace: openshift-migration
destMigClusterRef:
    name: <destination_cluster_ref>
    namespace: openshift-migration
imageStreamRef:
  name: <image_stream_name>
  namespace: <source_image_stream_namespace>
destNamespace: <destination_image_stream_namespace>

Specify the MigCluster CR name of the source cluster.

Specify the MigCluster CR name of the destination cluster.

Specify the image stream name.

Specify the image stream namespace on the source cluster.
Specify the image stream namespace on the destination cluster.

### 3.4.3.2.3. DirectVolumeMigration

The **DirectVolumeMigration** CR copies persistent volumes (PVs) directly from the source cluster to the destination cluster.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: DirectVolumeMigration
metadata:
  name: <directvolumemigration_name>
  namespace: openshift-migration
spec:
  createDestinationNamespaces: false
  deleteProgressReportingCRs: false
  destMigClusterRef:
    name: host
    namespace: openshift-migration
  persistentVolumeClaims:
    - name: <pvc_name>
      namespace: <pvc_namespace>
  srcMigClusterRef:
    name: <source_cluster_ref>
    namespace: openshift-migration
```

1. Namespaces are created for the PVs on the destination cluster if **true**.
2. The **DirectVolumeMigrationProgress** CRs are deleted after migration if **true**. The default value 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 with direct volume migration.
5. Specify the namespace of each PVC.
6. Specify the **MigCluster** CR name of the source cluster.

### 3.4.3.2.4. DirectVolumeMigrationProgress

The **DirectVolumeMigrationProgress** CR shows the progress of the **DirectVolumeMigration** CR.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: DirectVolumeMigrationProgress
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: directvolumemigrationprogress_name
spec:
  clusterRef:
    name: source_cluster
    namespace: openshift-migration
```
3.4.3.2.5. MigAnalytic

The **MigAnalytic** CR collects the number of images, Kubernetes resources, and the PV capacity from an associated **MigPlan** CR.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigAnalytic
metadata:
  annotations:
    migplan: <migplan_name>  
  name: miganalytic_name
  namespace: openshift-migration
  labels:
    migplan: <migplan_name>
spec:
  analyzeImageCount: true
  analyzeK8SResources: true
  analyzePVCapacity: true
  listImages: false
  listImagesLimit: 50
  migPlanRef:
    name: migplan_name
    namespace: openshift-migration
```

1. Specify the **MigPlan** CR name associated with the **MigAnalytic** CR.
2. Specify the **MigPlan** CR name associated with the **MigAnalytic** CR.
3. Optional: The number of images is returned if **true**.
4. Optional: Returns the number, kind, and API version of the Kubernetes resources if **true**.
5. Optional: Returns the PV capacity if **true**.
6. Returns a list of image names if **true**. Default is **false** so that the output is not excessively long.
7. Optional: Specify the maximum number of image names to return if **listImages** is **true**.
8. Specify the **MigPlan** CR name associated with the **MigAnalytic** CR.

3.4.3.2.6. MigCluster

The **MigCluster** CR defines a host, local, or remote cluster.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigCluster
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
```
Optional: Update the cluster name if the migration-controller pod is not running on this cluster.

The migration-controller pod runs on this cluster if true.

Optional: If the storage provider is Microsoft Azure, specify the resource group.

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.

SSL verification is enabled if false.

The cluster is validated if true.

The restic pods are restarted on the source cluster after the stage pods are created if true.

Optional: If you are using direct image migration, specify the exposed registry path of a remote cluster.

Specify the URL of the remote cluster.

Specify the name of the Secret CR for the remote cluster.

3.4.3.2.7. MigHook

The MigHook CR defines an Ansible playbook or a custom image that runs tasks at a specified stage of the migration.
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.

Specify the migration hook name, unless you specify the value of the `generateName` parameter.

Optional: Specify the maximum number of seconds that a hook can run. The default value is 1800.

The hook is a custom image if `true`. The custom image can include Ansible or it can be written in a different programming language.

Specify the custom image, for example, `quay.io/konveyor/hook-runner:latest`. Required if `custom` is `true`.

Specify the entire base64-encoded Ansible playbook. Required if `custom` is `false`.

Specify `source` or `destination` as the cluster on which the hook will run.

3.4.3.2.8. MigMigration

The MigMigration CR runs an associated MigPlan CR.

You can create multiple MigMigration CRs associated with the same MigPlan CR for the following scenarios:

- You can run multiple stage or incremental migrations to copy data without stopping the pods on the source cluster. Running stage migrations improves the performance of the actual migration.
- You can cancel a migration in progress.
- You can roll back a migration.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigMigration
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name:igmigration_name
  namespace: openshift-migration
spec:
canceled: false
rollback: false
stage: false
quiescePods: true
keepAnnotations: true
verify: false
migPlanRef:
  name: <migplan_ref>
  namespace: openshift-migration
```
A migration in progress is canceled if true.

A completed migration is rolled back if true.

Data is copied incrementally and the pods on the source cluster are not stopped if true.

The pods on the source cluster are scaled to 0 after the Backup stage of a migration if true.

The labels and annotations applied during the migration are retained if true.

The status of the migrated pods on the destination cluster are checked and the names of pods that are not in a Running state are returned if true.

3.4.3.2.9. MigPlan

The MigPlan CR defines the parameters of a migration plan. It contains a group of virtual machines that are being migrated with the same parameters.

```yaml
apiVersion: migration.openshift.io/v1alpha1
kind: MigPlan
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: migplan_name
  namespace: openshift-migration
spec:
  closed: false
  srcMigClusterRef:
    name: <source_migcluster_ref>
    namespace: openshift-migration
  destMigClusterRef:
    name: <destination_migcluster_ref>
    namespace: openshift-migration
  hooks:
  - executionNamespace: <namespace>
    phase: <migration_phase>
    reference:
      name: <mighook_name>
      namespace: <hook_namespace>
      serviceAccount: <service_account>
  indirectImageMigration: true
  indirectVolumeMigration: false
  migStorageRef:
    name: <migstorage_name>
    namespace: openshift-migration
  namespaces:
  - <namespace>
  refresh: false
```

The migration has completed if true. You cannot create another MigMigration CR for this MigPlan CR.
Specify the name of the source cluster **MigCluster** CR.

Specify the name of the destination cluster **MigCluster** CR.

Optional: You can specify up to four migration hooks.

Optional: Specify the namespace in which the hook will run.

Optional: Specify the migration phase during which a hook runs. One hook can be assigned to one phase. The expected values are **PreBackup**, **PostBackup**, **PreRestore**, and **PostRestore**.

Optional: Specify the name of the **MigHook** CR.

Optional: Specify the namespace of **MigHook** CR.

Optional: Specify a service account with `cluster-admin` privileges.

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.

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.

Specify the name of **MigStorage** CR.

Specify one or more namespaces.

The **MigPlan** CR is validated if `true`.

### 3.4.3.2.10. MigStorage

The **MigStorage** CR describes the object storage for the replication repository. You can configure Amazon Web Services, Microsoft Azure, Google Cloud Storage, and generic S3-compatible cloud storage, for example, Minio or NooBaa.

Different providers require different parameters.

```
apiVersion: migration.openshift.io/v1alpha1
kind: MigStorage
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: migstorage_name
  namespace: openshift-migration
spec:
  backupStorageProvider: <storage_provider>  
  volumeSnapshotProvider: 
  backupStorageConfig:
    awsBucketName:  
    awsRegion:  
    credsSecretRef:
      namespace: openshift-config
      name: <storage_secret>
    awsKmsKeyId:  
    awsPublicUrl:  
```
Specify the storage provider.

Optional: If you are using the snapshot copy method, specify the storage provider.

If you are using AWS, specify the bucket name.

If you are using AWS, specify the bucket region, for example, `us-east-1`.

Specify the name of the **Secret** CR that you created for the **MigStorage** CR.

Optional: If you are using the AWS Key Management Service, specify the unique identifier of the key.

Optional: If you granted public access to the AWS bucket, specify the bucket URL.

Optional: Specify the AWS signature version for authenticating requests to the bucket, for example, `4`.

Optional: If you are using the snapshot copy method, specify the geographical region of the clusters.

Optional: If you are using the snapshot copy method, specify the name of the **Secret** CR that you created for the **MigStorage** CR.

The cluster is validated if `true`.

### 3.4.3.3. Additional resources

- About migration hooks
- MigHook custom resource
- MigPlan custom resource

### 3.4.4. Configuring a migration plan

You can increase the number of objects to be migrated or exclude resources from the migration.

### 3.4.4.1. 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.

3.4.4.2. Excluding resources from a migration plan

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:
   
   ```bash
   $ oc edit migrationcontroller <migration_controller> -n openshift-migration
   ```

2. Update the spec section by adding a parameter to exclude specific resources or by adding a resource to the excluded_resources parameter if it does not have its own exclusion parameter:

   ```yaml
   apiVersion: migration.openshift.io/v1alpha1
   kind: MigrationController
   metadata:
     name: migration-controller
     namespace: openshift-migration
   spec:
     disable_image_migration: true
     disable_pv_migration: true
     ... 
     excluded_resources:
       - imagetags
       - templateinstances
       - clusterserviceversions
       - packagemanifests
       - subscriptions
       - servicebrokers
       - servicebindings
       - serviceclasses
       - serviceinstances
       - serviceplans
       - operatorgroups
       - events
   ```

   **1** Add disable_image_migration: true to exclude image streams from the migration. Do not edit the excluded_resources parameter. imagestreams is added to excluded_resources when the MigrationController pod restarts.

   **2** Add disable_pv_migration: true to exclude PVs from the migration plan. Do not edit the excluded_resources parameter. persistentvolumes and persistentvolumeclaims are added to excluded_resources 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 to the excluded_resources list. Do not delete the default excluded resources. These resources are problematic to migrate and must be excluded.

3. Wait two minutes for the MigrationController pod to restart so that the changes are applied.

4. Verify that the resource is excluded:
   
   ```bash
   $ oc get deployment -n openshift-migration migration-controller -o yaml | grep EXCLUDED_RESOURCES -A1
   ```

   The output contains the excluded resources:
3.5. TROUBLESHOOTING

You can view the Migration Toolkit for Containers (MTC) custom resources and download logs to troubleshoot a failed migration.

If the application was stopped during the failed migration, you must roll it back manually to prevent data corruption.

**NOTE**

Manual rollback is not required if the application was not stopped during migration because the original application is still running on the source cluster.

3.5.1. Viewing MTC custom resources

You can view 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:

  ```yaml
  labels:
  migrationcontroller: ebe13bee-c803-47d0-a9e9-83f380328b93
  ```

- **VolumeSnapshotLocation**
  
  The `VolumeSnapshotLocation` CR contains a `migrationcontroller` label to identify the MTC instance that created the CR:

  ```yaml
  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

```yaml
- name: EXCLUDED_RESOURCES
  value:
  imagetags,templateinstances,clusterserviceversions,packagemanifests,subscriptions,servicebrokers,servicebindings,serviceclasses,serviceinstances,serviceplans,imagestreams,persistentvolumes,persistentvolumeclaims
```
PVs.

- **Restore**

**Procedure**

1. List the **MigMigration** CRs in the **openshift-migration** namespace:

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

**Example output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>88435fe0-c9f8-11e9-85e6-5d593ce65e10</td>
<td>6m42s</td>
</tr>
</tbody>
</table>

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
  migration: 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"
Velero restore CR #2 example output that describes the Kubernetes resources

```yaml
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-
geneneration: 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
```

3.5.2. 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

   -c plain option displays the log without colors.

3.5.3. Downloading migration logs

You can download the **Velero**, **Restic**, and **MigrationController** pod logs in the Migration Toolkit for Containers (MTC) web console to troubleshoot a failed migration.

Procedure

1. In the MTC console, click **Migration plans** to view the list of migration plans.

2. Click the **Options** menu of a specific migration plan and select **Logs**.

3. Click **Download Logs** to download the logs of the **MigrationController**, **Velero**, and **Restic** pods for all clusters.

   You can download a single log by selecting the cluster, log source, and pod source, and then clicking **Download Selected**.

   You can access a pod log from the CLI by using the **oc logs** command:

   $ oc logs <pod-name> -f -n openshift-migration

   Specify the pod name.

3.5.4. Updating deprecated APIs

If your source cluster uses deprecated APIs, the following warning message is displayed when you create a migration plan in the Migration Toolkit for Containers (MTC) web console:

*Some namespaces contain GVKs incompatible with destination cluster*

You can click **See details** to view the namespace and the incompatible APIs. This warning message does not block the migration.

During migration with the Migration Toolkit for Containers (MTC), the deprecated APIs are saved in the **Velero Backup #1** for Kubernetes objects. You can download the **Velero Backup**, extract the deprecated API **yam** files, and update them with the **oc convert** command. Then you can create the updated APIs on the target cluster.

Procedure
1. Run the migration plan.

2. View the MigPlan custom resource (CR):

   $ oc describe migplan <migplan_name> -n openshift-migration

   Specify the name of the MigPlan CR.

   The output is similar to the following:

   metadata:
   ...
   uid: 79509e05-61d6-11e9-bc55-02ce4781844a
   status:
   ...
   conditions:
   - category: Warn
   lastTransitionTime: 2020-04-30T17:16:23Z
   message: 'Some namespaces contain GVKs incompatible with destination cluster. See: 'incompatibleNamespaces' for details'
   status: "True"
   type: GVKsIncompatible
   incompatibleNamespaces:
   - gvks:
     - group: batch
       kind: cronjobs
       version: v2alpha1
     - group: batch
       kind: scheduledjobs
       version: v2alpha1

   Record the MigPlan CR UID.

   Record the deprecated APIs listed in the gvks section.

3. Get the MigMigration name associated with the MigPlan UID:

   $ oc get migmigration -o json | jq -r '.items[] | select(.metadata.ownerReferences[].uid=="<migplan_uid>") | .metadata.name'

   Specify the MigPlan CR UID.

4. Get the MigMigration UID associated with the MigMigration name:

   $ oc get migmigration <migmigration_name> -o jsonpath='{.metadata.uid}'

   Specify the MigMigration name.

5. Get the Velero Backup name associated with the MigMigration UID:
$ oc get backup.velero.io --selector migration-initial-backup="<migmigration_uid>" -o jsonpath={.items[*].metadata.name}

Specify the MigMigration UID.

6. Download the contents of the Velero Backup to your local machine by running the command for your storage provider:

   - AWS S3:
     $ aws s3 cp s3://<bucket_name>/velero/backups/<backup_name> <backup_local_dir> --recursive
     Specify the bucket, backup name, and your local backup directory name.

   - GCP:
     $ gsutil cp gs://<bucket_name>/velero/backups/<backup_name> <backup_local_dir> --recursive
     Specify the bucket, backup name, and your local backup directory name.

   - Azure:
     $ azcopy copy
     'https://velerobackups.blob.core.windows.net/velero/backups/<backup_name>'
     '<backup_local_dir>' --recursive
     Specify the backup name and your local backup directory name.

7. Extract the Velero Backup archive file:

   $ tar -xfv <backup_local_dir>/<backup_name>.tar.gz -C <backup_local_dir>

8. Run oc convert in offline mode on each deprecated API:

   $ oc convert -f <backup_local_dir>/resources/<gvk>.json

9. Create the converted API on the target cluster:

   $ oc create -f <gvk>.json

3.5.5. 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.

3.5.5.1. CA certificate error in the MTC console
If the MTC console displays a **CA certificate error** message the first time you try to access it, the likely cause is that a cluster uses self-signed CA certificates.

Navigate to the **oauth-authorization-server** URL in the error message and accept the certificate. To resolve this issue permanently, install the certificate authority so that it is trusted.

If the browser displays an **Unauthorized** message after you have accepted the CA certificate, navigate to the MTC console and then refresh the web page.

### 3.5.5.2. OAuth timeout error in the MTC console

If the MTC console displays a **connection has timed out** message after you have accepted a self-signed certificate, the cause is likely to be one of the following:

- Interrupted network access to the OAuth server
- Interrupted network access to the OpenShift Container Platform console
- Proxy configuration blocking access to the OAuth server. See [MTC console inaccessible because of OAuth timeout error](#) for details.

To determine the cause:

- Inspect the MTC console web page with a browser web inspector.
- Check the **Migration UI** pod log for errors.

### 3.5.5.3. 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:

**BSL error messages**

```
Error checking repository for stale locks
Error getting backup storage location: backupstoragelocation.velero.io "my-bsl" not found
```

You can ignore these error messages. A missing BSL cannot cause a migration to fail.

### 3.5.5.4. Pod volume backup timeout error in the Velero pod log

If a migration fails because **Restic** times out, the **Velero** pod log displays the following error:

**Pod volume backup timeout error**

```
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:

   ```yaml
   spec:
     restic_timeout: 1h
   ```

   1 Valid units are h (hours), m (minutes), and s (seconds), for example, 3h30m15s.

5. Click Save.

3.5.5.5. 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 MigMigration CR displays the following error:

MigMigration CR status

```
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
  status: "True"
type: ResticVerifyErrors
```

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 troubleshoot 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.

**Restore CR with pod volume restore error**

```yaml
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:

```bash
$ oc describe <migration-example-rvwcm-98t49>
```

The output identifies the **Restic** pod that logged the errors.

**PodVolumeRestore CR with Restic pod error**

```yaml
completionTimestamp: 2020-05-01T20:49:12Z
errors: 1
resticErrors: 1

resticPod: <restic-nr2v5>
```

4. View the **Restic** pod log to locate the errors:

```bash
$ oc logs -f <restic-nr2v5>
```

3.5.5.6. 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 **Restic** pod log displays the following error:

**Restic permission error**

```bash
```

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:

   ```yaml
   spec:
   restic_supplemental_groups: <group_id>  
   ```

   Specify the supplemental group ID.

4. Wait for the Restic pods to restart so that the changes are applied.

3.5.6. Direct volume migration does not complete

If direct volume migration does not complete, the target cluster might not have the same `node-selector` annotations as the source cluster.

Migration Toolkit for Containers (MTC) migrates namespaces with all annotations to preserve security context constraints and scheduling requirements. During direct volume migration, MTC creates Rsync transfer pods on the target cluster in the namespaces that were migrated from the source cluster. If a target cluster namespace does not have the same annotations as the source cluster namespace, the Rsync transfer pods cannot be scheduled. The Rsync pods remain in a `Pending` state.

You can identify and fix this issue by performing the following procedure.

Procedure

1. Check the status of the `MigMigration` CR:

   ```bash
   $ oc describe migmigration <pod_name> -n openshift-migration
   ```

   The output includes the following status message:

   **Example output**

   ```
   ... Some or all transfer pods are not running for more than 10 mins on destination cluster ...
   ```

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

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

   Specify the migrated namespace.

3. On the target cluster, edit the migrated namespace:

   ```bash
   $ oc edit namespace <namespace>
   ```
4. Add missing `openshift.io/node-selector` annotations to the migrated namespace as in the following example:

```yaml
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    openshift.io/node-selector: "region=east"
...```

5. Run the migration plan again.

### 3.5.7. Using the Velero CLI to debug Backup and Restore CRs

You can debug the **Backup** and **Restore** custom resources (CRs) and partial migration failures with the Velero command line interface (CLI). The Velero CLI runs in the **velero** pod.

#### 3.5.7.1. Velero command syntax

Velero CLI commands use the following syntax:

```
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -- ./velero <resource> <command> <resource_id>
```

You can specify `velero-<pod> -n openshift-migration` in place of `$(oc get pods -n openshift-migration -o name | grep velero)`.

#### 3.5.7.2. Help command

The Velero **help** command lists all the Velero CLI commands:

```
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -- ./velero --help
```

#### 3.5.7.3. Describe command

The Velero **describe** command provides a summary of warnings and errors associated with a Velero resource:

```
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -- ./velero <resource> describe <resource_id>
```

**Example**

```
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -- ./velero backup describe 0e44ae00-5dc3-11eb-9ca8-df7e5254778b-2d8ql
```

#### 3.5.7.4. Logs command

The Velero **logs** command provides the logs associated with a Velero resource:

```
velero <resource> logs <resource_id>
```
### 3.5.7.5. 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:

   ```bash
   $ 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:

   ```bash
   $ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -n openshift-migration -
   - ./velero restore describe <restore>
   
   Example output
   
   Phase: PartiallyFailed (run `velero restore logs ccc7c2d0-6017-11eb-afab-85d0007f5a19-x4lbf` for more information)
   
   Errors:
3. Check the **Restore** CR logs by using the Velero **logs** command:

```bash
$ oc exec $(oc get pods -n openshift-migration -o name | grep velero) -n openshift-migration -
- ./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.

### 3.5.8. Using **must-gather** to collect data

You must run the **must-gather** tool if you open a customer support case on the [Red Hat Customer Portal](https://www.redhat.com) for the Migration Toolkit for Containers (MTC).

The **openshift-migration-must-gather-rhel8** image for MTC collects migration-specific logs and data that are not collected by the default **must-gather** image.

**Procedure**

1. Navigate to the directory where you want to store the **must-gather** data.

2. Run the **must-gather** command:

   ```bash
   $ oc adm must-gather --image=registry.redhat.io/rhmtc/openshift-migration-must-gather-rhel8:v1.4
   ```

3. Remove authentication keys and other sensitive information.

4. Create an archive file containing the contents of the **must-gather** data directory:

   ```bash
   $ tar cvaf must-gather.tar.gz must-gather.local.<uid>/
   ```

5. Upload the compressed file as an attachment to your customer support case.

### 3.5.9. Rolling back a migration

You can roll back a migration by using the MTC web console or the CLI.

#### 3.5.9.1. Rolling back a migration in the MTC web console
You can roll back a migration by using the Migration Toolkit for Containers (MTC) web console.

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**.

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.

3.5.9.1.1. Rolling back a migration from the CLI

You can roll back a migration by creating a **MigMigration** custom resource (CR) from the CLI.

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:

   ```bash
   $ cat << EOF | oc apply -f -
   apiVersion: migration.openshift.io/v1alpha1
   kind: MigMigration
   metadata:
     labels:
       controller-tools.k8s.io: "1.0"
   name: migration-rollback
   namespace: openshift-migration
   spec:
   EOF
   ```
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.

3.5.10. 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:

```yaml
spec:
  ...
  restic_supplemental_groups:
    ...
```
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)

3.5.11. Additional resources

- MTC workflow
- MTC custom resources