Red Hat build of MicroShift 4.15

Running applications

Running applications in MicroShift

Last Updated: 2024-04-01

Red Hat build of MicroShift 4.15 Running applications

Running applications in MicroShift

Legal Notice

Abstract

This document provides details about how to run applications in MicroShift.

Table of Contents

CHAPTER 1. USING KUSTOMIZE MANIFESTS TO DEPLOY APPLICATIONS

You can use the **kustomize** configuration management tool with application manifests to deploy applications. Read through the following procedures for an example of how Kustomize works in MicroShift.

1.1. HOW KUSTOMIZE WORKS WITH MANIFESTS TO DEPLOY APPLICATIONS

The **kustomize** configuration management tool is integrated with MicroShift. You can use Kustomize and the OpenShift CLI (**oc**) together to apply customizations to your application manifests and deploy those applications to a MicroShift cluster.

- A **kustomization.yaml** file is a specification of resources plus customizations.
- Kustomize uses a **kustomization.yaml** file to load a resource, such as an application, then applies any changes you want to that application manifest and produces a copy of the manifest with the changes overlaid.
- Using a manifest copy with an overlay keeps the original configuration file for your application intact, while enabling you to deploy iterations and customizations of your applications efficiently.
- You can then deploy the application in your MicroShift cluster with an **oc** command.

1.1.1. How MicroShift uses manifests

At every start, MicroShift searches the following manifest directories for Kustomize manifest files:

- **/etc/microshift/manifests**
- **/etc/microshift/manifests.d/***
- `/usr/lib/microshift/
- **/usr/lib/microshift/manifests.d/***

MicroShift automatically runs the equivalent of the **kubectl apply -k** command to apply the manifests to the cluster if any of the following file types exists in the searched directories:

- **kustomization.yaml**
- **kustomization.yml**
- **Kustomization**

This automatic loading from multiple directories means you can manage MicroShift workloads with the flexibility of having different workloads run independently of each other.

Table 1.1. MicroShift manifest directories

Location	Intent

Location	Intent
/etc/microshift/manifests	Read-write location for configuration management systems or development.
/etc/microshift/manifests.d/*	Read-write location for configuration management systems or development.
/usr/lib/microshift/manifests	Read-only location for embedding configuration manifests on OSTree-based systems.
/usr/lib/microshift/manifestsd./*	Read-only location for embedding configuration manifests on OSTree-based systems.

1.2. OVERRIDE THE LIST OF MANIFEST PATHS

You can override the list of default manifest paths by using a new single path, or by using a new glob pattern for multiple files. Use the following procedure to customize your manifest paths.

Procedure

1. Override the list of default paths by inserting your own values and running one of the following commands:

   a. Set **manifests.kustomizePaths** to **<"/opt/alternate/path">** in the configuration file for a single path.

   b. Set **kustomizePaths** to **,"/opt/alternative/path.d/*".** in the configuration file for a glob pattern.

```
manifests:
  kustomizePaths:
    - <location> 1
```

1 Set each location entry to an exact path by using **"/opt/alternate/path"** or a glob pattern by using **"/opt/alternative/path.d/*"**.

2. To disable loading manifests, set the configuration option to an empty list.

```
manifests:
  kustomizePaths: []
```

NOTE

The configuration file overrides the defaults entirely. If the **kustomizePaths** value is set, only the values in the configuration file are used. Setting the value to an empty list disables manifest loading.

1.3. USING MANIFESTS EXAMPLE

This example demonstrates automatic deployment of a BusyBox container using **kustomize** manifests in the **/etc/microshift/manifests** directory.

Procedure

1. Create the BusyBox manifest files by running the following commands:

   a. Define the directory location:

   ```
   $ MANIFEST_DIR=/etc/microshift/manifests
   ```

   b. Make the directory:

   ```
   $ sudo mkdir -p ${MANIFEST_DIR}
   ```

   c. Place the YAML file in the directory:

   ```
   sudo tee ${MANIFEST_DIR}/busybox.yaml &>/dev/null <<EOF
   apiVersion: v1
   kind: Namespace
   metadata:
     name: busybox
   ---
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: busybox
     namespace: busybox-deployment
   spec:
     selector:
       matchLabels:
         app: busybox
     template:
       metadata:
         labels:
           app: busybox
       spec:
         containers:
         - name: busybox
           image: BUSYBOX_IMAGE
           command: [ "/bin/sh", "-c", "while true ; do date; sleep 3600; done;" ]
   EOF
   ```

2. Next, create the **kustomize** manifest files by running the following commands:

   a. Place the YAML file in the directory:

   ```
   sudo tee ${MANIFEST_DIR}/kustomization.yaml &>/dev/null <<EOF
   apiVersion: kustomize.config.k8s.io/v1beta1
   kind: Kustomization
   namespace: busybox
   resources:
     - busybox.yaml
   images:
   ```

```
  - name: BUSYBOX_IMAGE
    newName: busybox:1.35
EOF
```

3. Restart MicroShift to apply the manifests by running the following command:

```
$ sudo systemctl restart microshift
```

4. Apply the manifests and start the **busybox** pod by running the following command:

```
$ oc get pods -n busybox
```

CHAPTER 2. OPTIONS FOR EMBEDDING MICROSHIFT APPLICATIONS IN A RHEL FOR EDGE IMAGE

You can embed microservices-based workloads and applications in a Red Hat Enterprise Linux for Edge (RHEL for Edge) image to run in a MicroShift cluster. Embedded applications can be installed directly on edge devices to run in air-gapped, disconnected, or offline environments.

2.1. ADDING APPLICATION RPMS TO AN RPM-OSTREE IMAGE

If you have an application that includes APIs, container images, and configuration files for deployment such as manifests, you can build application RPMs. You can then add the RPMs to your RHEL for Edge system image.

The following is an outline of the procedures to embed applications or workloads in a fully self-contained operating system image:

- Build your own RPM that includes your application manifest.
- Add the RPM to the blueprint you used to install Red Hat build of MicroShift.
- Add the workload container images to the same blueprint.
- Create a bootable ISO.

For a step-by-step tutorial about preparing and embedding applications in a RHEL for Edge image, use the following tutorial:

- Embedding applications tutorial

2.2. ADDING APPLICATION MANIFESTS TO AN IMAGE FOR OFFLINE USE

If you have a simple application that includes a few files for deployment such as manifests, you can add those manifests directly to a RHEL for Edge system image.

See the "Create a custom file blueprint customization" section of the following RHEL for Edge documentation for an example:

- Create a custom file blueprint customization

2.3. EMBEDDING APPLICATIONS FOR OFFLINE USE

If you have an application that includes more than a few files, you can embed the application for offline use. See the following procedure:

- Embedding applications for offline use

2.4. ADDITIONAL RESOURCES

- Embedding Red Hat build of MicroShift in an RPM-OSTree image
- Composing, installing, and managing RHEL for Edge images

- Preparing for image building
- Meet Red Hat Device Edge
- Composing a RHEL for Edge image using image builder command-line
- Image Builder system requirements

CHAPTER 3. EMBEDDING APPLICATIONS FOR OFFLINE USE

You can embed microservices-based workloads and applications in a Red Hat Enterprise Linux for Edge (RHEL for Edge) image. Embedding means you can run a Red Hat build of MicroShift cluster in air-gapped, disconnected, or offline environments.

3.1. EMBEDDING WORKLOAD CONTAINER IMAGES FOR OFFLINE USE

To embed container images in devices at the edge that do not have any network connection, you must create a new container, mount the ISO, and then copy the contents into the file system.

Prerequisites

- You have root access to the host.
- Application RPMs have been added to a blueprint.

Procedure

1. Render the manifests, extract all of the container image references, and translate the application image to blueprint container sources by running the following command:

```
$ oc kustomize ~/manifests | grep "image:" | grep -oE '[^ ]+$' | while read line; do echo -e "
[[containers]]\nsource = \"${line}\"\n"; done >><my_blueprint>.toml
```

2. Push the updated blueprint to Image Builder by running the following command:

```
$ sudo composer-cli blueprints push <my_blueprint>.toml
```

3. If your workload containers are located in a private repository, you must provide Image Builder with the necessary pull secrets:

   a. Set the **auth_file_path** in the **[containers]** section of the **osbuilder worker** configuration in the **/etc/osbuild-worker/osbuild-worker.toml** file to point to the pull secret.

   b. If needed, create a directory and file for the pull secret, for example:

   Example directory and file

```
[containers]
auth_file_path = "/<path>/pull-secret.json" 1
```

   1 Use the custom location previously set for copying and retrieving images.

4. Build the container image by running the following command:

```
$ sudo composer-cli compose start-ostree <my_blueprint> edge-commit
```

5. Proceed with your preferred **rpm-ostree** image flow, such as waiting for the build to complete, exporting the image and integrating it into your **rpm-ostree** repository or creating a bootable ISO.

3.2. ADDITIONAL RESOURCES

- Options for embedding Red Hat build of MicroShift applications in a RHEL for Edge image
- Creating the RHEL for Edge image
- Add the blueprint to Image Builder and build the ISO
- Download the ISO and prepare it for use
- Upgrading RHEL for Edge systems

CHAPTER 4. EMBEDDING RED HAT BUILD OF MICROSHIFT APPLICATIONS TUTORIAL

The following tutorial gives a detailed example of how to embed applications in a RHEL for Edge image for use in a MicroShift cluster in various environments.

4.1. EMBED APPLICATION RPMS TUTORIAL

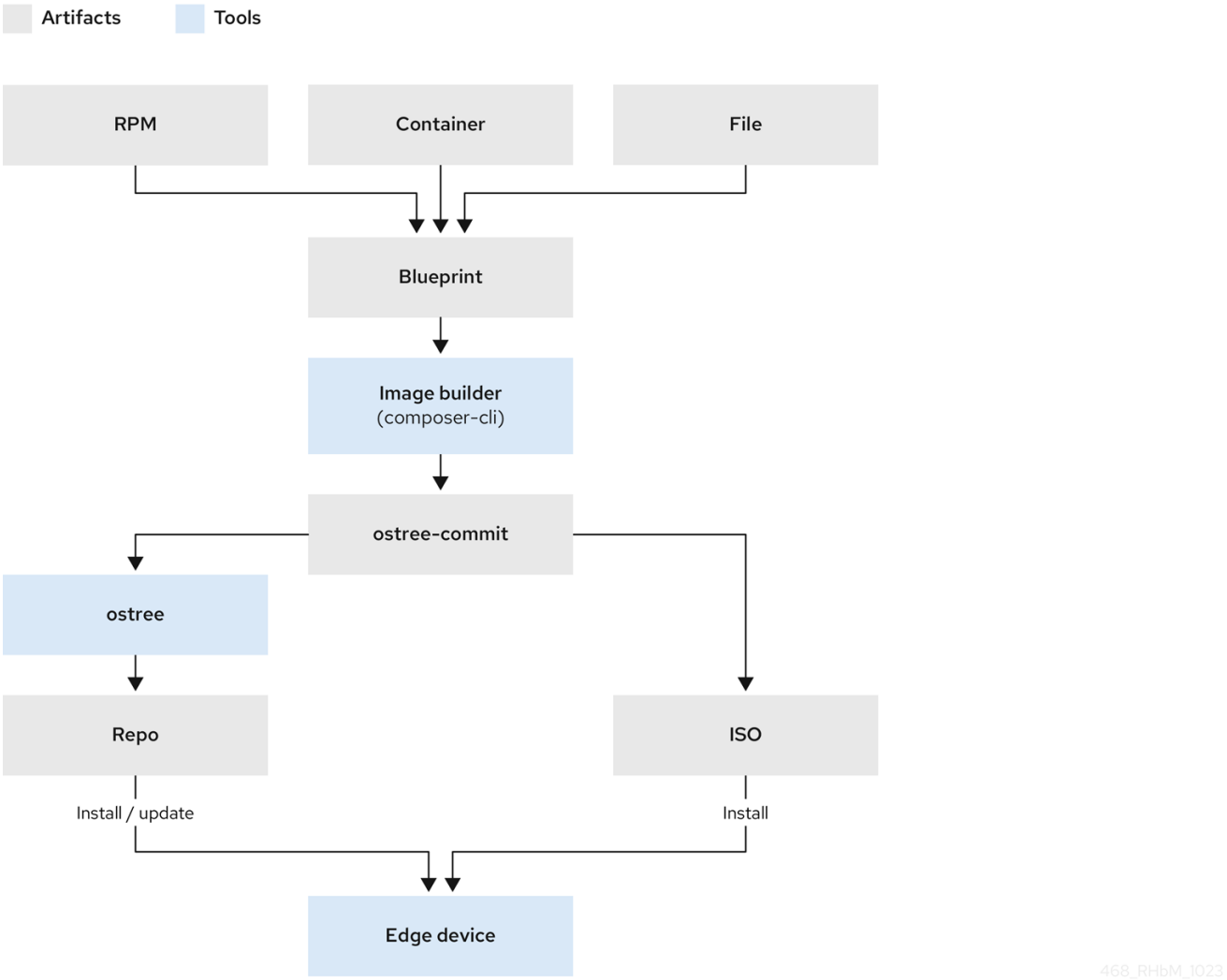
The following tutorial reviews the MicroShift installation steps and adds a description of the workflow for embedding applications. If you are already familiar with **rpm-ostree** systems such as Red Hat Enterprise Linux for Edge (RHEL for Edge) and MicroShift, you can go straight to the procedures.

4.1.1. Installation workflow review

Embedding applications requires a similar workflow to embedding MicroShift into a RHEL for Edge image.

- The following image shows how system artifacts such as RPMs, containers, and files are added to a blueprint and used by the image composer to create an ostree commit.
- The ostree commit then can follow either the ISO path or the repository path to edge devices.
- The ISO path can be used for disconnected environments, while the repository path is often used in places were the network is usually connected.

Embedding MicroShift workflow



Reviewing these steps can help you understand the steps needed to embed an application:

1. To embed MicroShift on RHEL for Edge, you added the MicroShift repositories to Image Builder.

2. You created a blueprint that declared all the RPMs, container images, files and customizations you needed, including the addition of MicroShift.

3. You added the blueprint to Image Builder and ran a build with the Image Builder CLI tool (**composer-cli**). This step created **rpm-ostree** commits, which were used to create the container image. This image contained RHEL for Edge.

4. You added the installer blueprint to Image Builder to create an **rpm-ostree** image (ISO) to boot from. This build contained both RHEL for Edge and MicroShift.

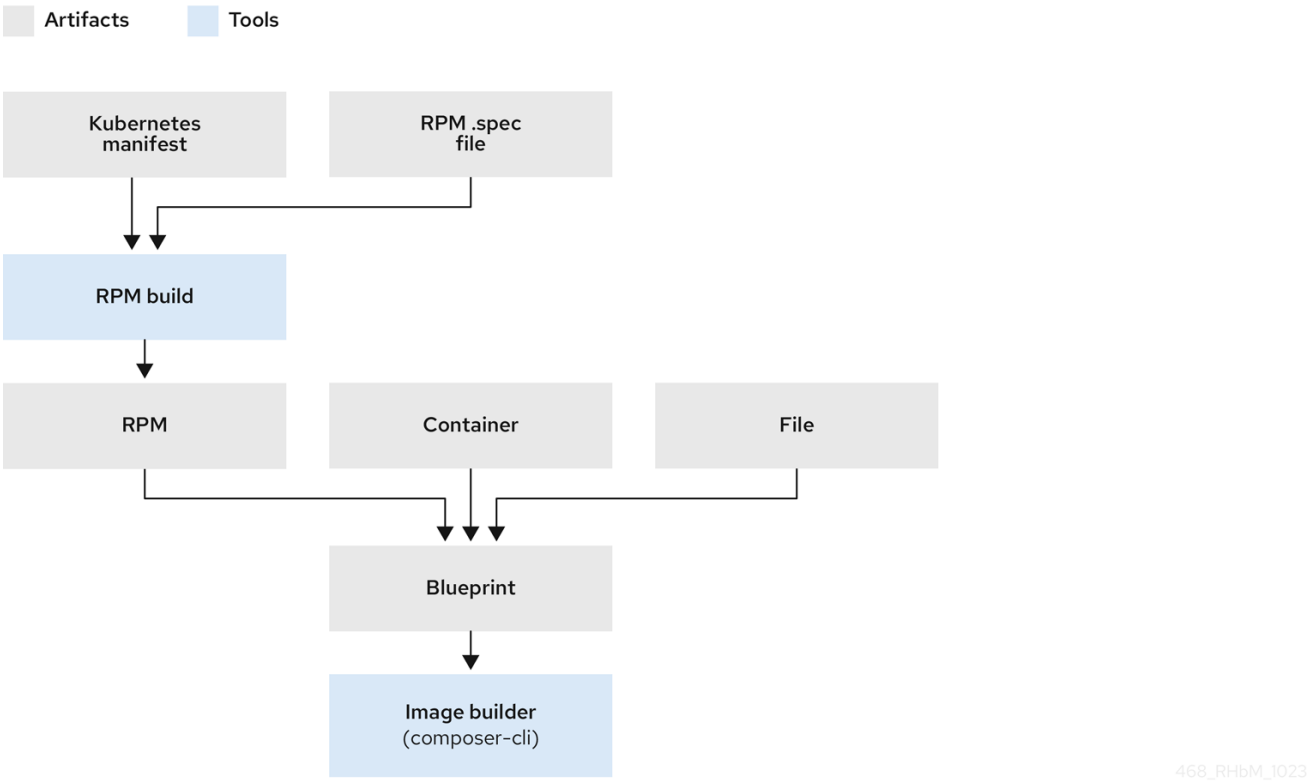
5. You downloaded the ISO with MicroShift embedded, prepared it for use, provisioned it, then installed it onto your edge devices.

4.1.2. Embed application RPMs workflow

After you have set up a build host that meets the Image Builder requirements, you can add your application in the form of a directory of manifests to the image. After those steps, the simplest way to embed your application or workload into a new ISO is to create your own RPMs that include the manifests. Your application RPMs contain all of the configuration files describing your deployment.

The following "Embedding applications workflow" image shows how Kubernetes application manifests and RPM spec files are combined in a single application RPM build. This build becomes the RPM artifact included in the workflow for embedding MicroShift in an ostree commit.

Embedding applications workflow



The following procedures use the **rpmbuild** tool to create a specification file and local repository. The specification file defines how the package is built, moving your application manifests to the correct location inside the RPM package for MicroShift to pick them up. That RPM package is then embedded in the ISO.

4.1.3. Preparing to make application RPMs

To build your own RPMs, choose a tool of your choice, such as the **rpmbuild** tool, and initialize the RPM build tree in your home directory. The following is an example procedure. As long as your RPMs are accessible to Image Builder, you can use the method you prefer to build the application RPMs.

Prerequisites

- You have set up a Red Hat Enterprise Linux for Edge (RHEL for Edge) 9.2 build host that meets the Image Builder system requirements.
- You have root access to the host.

Procedure

1. Install the **rpmbuild** tool and create the yum repository for it by running the following command:

```
$ sudo dnf install rpmdevtools rpmlint yum-utils createrepo
```

2. Create the file tree you need to build RPM packages by running the following command:

```
$ rpmdev-setuptree
```

Verification

- List the directories to confirm creation by running the following command:

```
$ ls ~/rpmbuild/
```

Example output

```
BUILD RPMS SOURCES SPECS SRPMS
```

4.1.4. Building the RPM package for the application manifests

To build your own RPMs, you must create a spec file that adds the application manifests to the RPM package. The following is an example procedure. As long as the application RPMs and other elements needed for image building are accessible to Image Builder, you can use the method that you prefer.

Prerequisites

- You have set up a Red Hat Enterprise Linux for Edge (RHEL for Edge) 9.2 build host that meets the Image Builder system requirements.
- You have root access to the host.
- The file tree required to build RPM packages was created.

Procedure

1. In the **~/rpmbuild/SPECS** directory, create a file such as **<application_workload_manifests.spec>** using the following template:

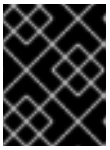
Example spec file

```
Name: <application_workload_manifests>
Version: 0.0.1
Release: 1%{?dist}
Summary: Adds workload manifests to microshift
BuildArch: noarch
License: GPL
Source0: %{name}-%{version}.tar.gz
#Requires: microshift
%description
Adds workload manifests to microshift
%prep
%autosetup
%install (1)
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/%{_prefix}/lib/microshift/manifests
cp -pr ~/manifests $RPM_BUILD_ROOT/%{_prefix}/lib/microshift/
%clean
```

```
rm -rf $RPM_BUILD_ROOT

%files
%{_prefix}/lib/microshift/manifests/**
%changelog
* <DDD MM DD YYYY username@domain - V major.minor.patch>
- <your_change_log_comment>
```

1 The **%install** section creates the target directory inside the RPM package, **/usr/lib/microshift/manifests/** and copies the manifests from the source home directory, **~/manifests**.



IMPORTANT

All of the required YAML files must be in the source home directory **~/manifests**, including a **kustomize.yaml** file if you are using kustomize.

2. Build your RPM package in the **~/rpmbuild/RPMS** directory by running the following command:

```
$ rpmbuild -bb ~/rpmbuild/SPECS/<application_workload_manifests.spec>
```

4.1.5. Adding application RPMs to a blueprint

To add application RPMs to a blueprint, you must create a local repository that Image Builder can use to create the ISO. With this procedure, the required container images for your workload can be pulled over the network.

Prerequisites

- You have root access to the host.
- Workload or application RPMs exist in the **~/rpmbuild/RPMS** directory.

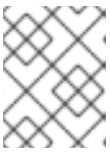
Procedure

1. Create a local RPM repository by running the following command:

```
$ createrepo ~/rpmbuild/RPMS/
```

2. Give Image Builder access to the RPM repository by running the following command:

```
$ sudo chmod a+rx ~
```



NOTE

You must ensure that Image Builder has all of the necessary permissions to access all of the files needed for image building, or the build cannot proceed.

3. Create the blueprint file, **repo-local-rpmbuild.toml** using the following template:

```
id = "local-rpm-build"
```

```
name = "RPMs build locally"
type = "yum-baseurl"
url = "file://<path>/rpmbuild/RPMS" 1
check_gpg = false
check_ssl = false
system = false
```

1 Specify part of the path to create a location that you choose. Use this path in the later commands to set up the repository and copy the RPMs.

4. Add the repository as a source for Image Builder by running the following command:

```
$ sudo composer-cli sources add repo-local-rpmbuild.toml
```

5. Add the RPM to your blueprint, by adding the following lines:

```
…
[[packages]]
name = "<application_workload_manifests>" 1
version = "*"
…
```

1 Add the name of your workload here.

6. Push the updated blueprint to Image Builder by running the following command:

```
$ sudo composer-cli blueprints push repo-local-rpmbuild.toml
```

7. At this point, you can either run Image Builder to create the ISO, or embed the container images for offline use.

   a. To create the ISO, start Image Builder by running the following command:

```
$ sudo composer-cli compose start-ostree repo-local-rpmbuild edge-commit
```

In this scenario, the container images are pulled over the network by the edge device during startup.

Additional resources

- Composing a RHEL for Edge image using the Image Builder CLI
- Network-based deployments workflow

4.2. ADDITIONAL RESOURCES

- Embedding applications for offline use
- Embedding Red Hat build of MicroShift in an RPM-OSTree image
- Composing, installing, and managing RHEL for Edge images
- Preparing for image building

- Meet Red Hat Device Edge with Red Hat build of MicroShift
- How to create a Linux RPM package
- Composing a RHEL for Edge image using image builder command-line
- Image Builder system requirements

CHAPTER 5. GREENBOOT WORKLOAD HEALTH CHECK SCRIPTS

Greenboot health check scripts are helpful on edge devices where direct serviceability is either limited or non-existent. You can create health check scripts to assess the health of your workloads and applications. These additional health check scripts are useful components of software problem checks and automatic system rollbacks.

A MicroShift health check script is included in the **microshift-greenboot** RPM. You can also create your own health check scripts based on the workloads you are running. For example, you can write one that verifies that a service has started.

5.1. HOW WORKLOAD HEALTH CHECK SCRIPTS WORK

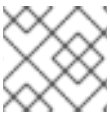
The workload or application health check script described in this tutorial uses the MicroShift health check functions that are available in the **/usr/share/microshift/functions/greenboot.sh** file. This enables you to reuse procedures already implemented for the MicroShift core services.

The script starts by running checks that the basic functions of the workload are operating as expected. To run the script successfully:

- Execute the script from a root user account.
- Enable the MicroShift service.

The health check performs the following actions:

- Gets a wait timeout of the current boot cycle for the **wait_for** function.
- Calls the **namespace_images_downloaded** function to wait until pod images are available.
- Calls the **namespace_pods_ready** function to wait until pods are ready.
- Calls the **namespace_pods_not_restarting** function to verify pods are not restarting.



NOTE

Restarting pods can indicate a crash loop.

5.2. INCLUDED GREENBOOT HEALTH CHECKS

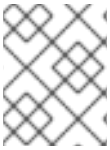
Health check scripts are available in **/usr/lib/greenboot/check**, a read-only directory in RPM-OSTree systems. The following health checks are included with the **greenboot-default-health-checks** framework.

- Check if repository URLs are still DNS solvable:
  This script is under **/usr/lib/greenboot/check/required.d/01_repository_dns_check.sh** and ensures that DNS queries to repository URLs are still available.
- Check if update platforms are still reachable:
  This script is under **/usr/lib/greenboot/check/wanted.d/01_update_platform_check.sh** and tries to connect and get a 2XX or 3XX HTTP code from the update platforms defined in **/etc/ostree/remotes.d**.

- Check if the current boot has been triggered by the hardware watchdog:
  This script is under **/usr/lib/greenboot/check/required.d/02_watchdog.sh** and checks whether the current boot has been watchdog-triggered or not.
  - If the watchdog-triggered reboot occurs within the grace period, the current boot is marked as red. Greenboot does not trigger a rollback to the previous deployment.
  - If the watchdog-triggered reboot occurs after the grace period, the current boot is not marked as red. Greenboot does not trigger a rollback to the previous deployment.
  - A 24-hour grace period is enabled by default. This grace period can be either disabled by modifying **GREENBOOT_WATCHDOG_CHECK_ENABLED** in **/etc/greenboot/greenboot.conf to false**, or configured by changing the **GREENBOOT_WATCHDOG_GRACE_PERIOD=number_of_hours** variable value in **/etc/greenboot/greenboot.conf**.

5.3. HOW TO CREATE A HEALTH CHECK SCRIPT FOR YOUR APPLICATION

You can create workload or application health check scripts in the text editor of your choice using the example in this documentation. Save the scripts in the **/etc/greenboot/check/required.d** directory. When a script in the **/etc/greenboot/check/required.d** directory exits with an error, Greenboot triggers a reboot in an attempt to heal the system.



NOTE

Any script in the **/etc/greenboot/check/required.d** directory triggers a reboot if it exits with an error.

If your health check logic requires any post-check steps, you can also create additional scripts and save them in the relevant greenboot directories. For example:

- You can also place shell scripts you want to run after a boot has been declared successful in **/etc/greenboot/green.d**.
- You can place shell scripts you want to run after a boot has been declared failed in **/etc/greenboot/red.d**. For example, if you have steps to heal the system before restarting, you can create scripts for your use case and place them in the **/etc/greenboot/red.d** directory.

5.3.1. About the workload health check script example

The following example uses the MicroShift health check script as a template. You can use this example with the provided libraries as a guide for creating basic health check scripts for your applications.

5.3.1.1. Basic prerequisites for creating a health check script

- The workload must be installed.
- You must have root access.

5.3.1.2. Example and functional requirements

You can start with the following example health check script. Modify it for your use case. In your workload health check script, you must complete the following minimum steps:

- Set the environment variables.
- Define the user workload namespaces.
- List the expected pod count.

IMPORTANT

Choose a name prefix for your application that ensures it runs after the **40_microshift_running_check.sh** script, which implements the Red Hat build of MicroShift health check procedure for its core services.

Example workload health check script

```
# #!/bin/bash
set -e

SCRIPT_NAME=$(basename $0)
PODS_NS_LIST=(<user_workload_namespace1> <user_workload_namespace2>)
PODS_CT_LIST=(<user_workload_namespace1_pod_count>
<user_workload_namespace2_pod_count>)
# Update these two lines with at least one namespace and the pod counts that are specific to your
workloads. Use the kubernetes <namespace> where your workload is deployed.

# Set Greenboot to read and execute the workload health check functions library.
source /usr/share/microshift/functions/greenboot.sh

# Set the exit handler to log the exit status.
trap 'script_exit' EXIT

# Set the script exit handler to log a `FAILURE` or `FINISHED` message depending on the exit status
of the last command.
# args: None
# return: None
function script_exit() {
    [ "$?" -ne 0 ] && status=FAILURE || status=FINISHED
    echo $status
}

# Set the system to automatically stop the script if the user running it is not 'root'.
if [ $(id -u) -ne 0 ] ; then
    echo "The '${SCRIPT_NAME}' script must be run with the 'root' user privileges"
    exit 1
fi

echo "STARTED"

# Set the script to stop without reporting an error if the MicroShift service is not running.
if [ $(systemctl is-enabled microshift.service 2>/dev/null) != "enabled" ] ; then
    echo "MicroShift service is not enabled. Exiting..."
    exit 0
fi

# Set the wait timeout for the current check based on the boot counter.
WAIT_TIMEOUT_SECS=$(get_wait_timeout)
```

```
# Set the script to wait for the pod images to be downloaded.
for i in ${!PODS_NS_LIST[@]}; do
  CHECK_PODS_NS=${PODS_NS_LIST[$i]}

  echo "Waiting ${WAIT_TIMEOUT_SECS}s for pod image(s) from the ${CHECK_PODS_NS}
namespace to be downloaded"
  wait_for ${WAIT_TIMEOUT_SECS} namespace_images_downloaded
done

# Set the script to wait for pods to enter ready state.
for i in ${!PODS_NS_LIST[@]}; do
  CHECK_PODS_NS=${PODS_NS_LIST[$i]}
  CHECK_PODS_CT=${PODS_CT_LIST[$i]}

  echo "Waiting ${WAIT_TIMEOUT_SECS}s for ${CHECK_PODS_CT} pod(s) from the
${CHECK_PODS_NS} namespace to be in 'Ready' state"
  wait_for ${WAIT_TIMEOUT_SECS} namespace_pods_ready
done

# Verify that pods are not restarting by running, which could indicate a crash loop.
for i in ${!PODS_NS_LIST[@]}; do
  CHECK_PODS_NS=${PODS_NS_LIST[$i]}

  echo "Checking pod restart count in the ${CHECK_PODS_NS} namespace"
  namespace_pods_not_restarting ${CHECK_PODS_NS}
done
```

5.4. TESTING A WORKLOAD HEALTH CHECK SCRIPT

Prerequisites

- You have root access.
- You have installed a workload.
- You have created a health check script for the workload.
- The Red Hat build of MicroShift service is enabled.

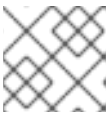
Procedure

1. To test that Greenboot is running a health check script file, reboot the host by running the following command:

```
$ sudo reboot
```

2. Examine the output of Greenboot health checks by running the following command:

```
$ sudo journalctl -o cat -u greenboot-healthcheck.service
```



NOTE

MicroShift core service health checks run before the workload health checks.

Example output

```
GRUB boot variables:
boot_success=0
boot_indeterminate=0
Greenboot variables:
GREENBOOT_WATCHDOG_CHECK_ENABLED=true
...
...
FINISHED
Script '40_microshift_running_check.sh' SUCCESS
Running Wanted Health Check Scripts...
Finished greenboot Health Checks Runner.
```

5.5. ADDITIONAL RESOURCES

- The Greenboot health check
- Auto applying manifests

CHAPTER 6. POD SECURITY AUTHENTICATION AND AUTHORIZATION

6.1. UNDERSTANDING AND MANAGING POD SECURITY ADMISSION

Pod security admission is an implementation of the Kubernetes pod security standards. Use pod security admission to restrict the behavior of pods.

6.2. SECURITY CONTEXT CONSTRAINT SYNCHRONIZATION WITH POD SECURITY STANDARDS

MicroShift includes Kubernetes pod security admission.

In addition to the global pod security admission control configuration, a controller exists that applies pod security admission control **warn** and **audit** labels to namespaces according to the security context constraint (SCC) permissions of the service accounts that are in a given namespace.

IMPORTANT

Namespaces that are defined as part of the cluster payload have pod security admission synchronization disabled permanently. You can enable pod security admission synchronization on other namespaces as necessary. If an Operator is installed in a user-created **openshift-*** namespace, synchronization is turned on by default after a cluster service version (CSV) is created in the namespace.

The controller examines **ServiceAccount** object permissions to use security context constraints in each namespace. Security context constraints (SCCs) are mapped to pod security profiles based on their field values; the controller uses these translated profiles. Pod security admission **warn** and **audit** labels are set to the most privileged pod security profile found in the namespace to prevent warnings and audit logging as pods are created.

Namespace labeling is based on consideration of namespace-local service account privileges.

Applying pods directly might use the SCC privileges of the user who runs the pod. However, user privileges are not considered during automatic labeling.

6.2.1. Viewing security context constraints in a namespace

You can view the security context constraints (SCC) permissions in a given namespace.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- To view the security context constraints in your namespace, run the following command:

```
oc get --show-labels namespace <namespace>
```

6.3. CONTROLLING POD SECURITY ADMISSION SYNCHRONIZATION

You can enable automatic pod security admission synchronization for most namespaces.

System defaults are not enforced when the **security.openshift.io/scc.podSecurityLabelSync** field is empty or set to **false**. You must set the label to **true** for synchronization to occur.

IMPORTANT

Namespaces that are defined as part of the cluster payload have pod security admission synchronization disabled permanently. These namespaces include:

- **default**
- **kube-node-lease**
- **kube-system**
- **kube-public**
- **openshift**
- All system-created namespaces that are prefixed with **openshift-**, except for **openshift-operators** By default, all namespaces that have an **openshift-** prefix are not synchronized. You can enable synchronization for any user-created **openshift-*** namespaces. You cannot enable synchronization for any system-created **openshift-*** namespaces, except for **openshift-operators**.

If an Operator is installed in a user-created **openshift-*** namespace, synchronization is turned on by default after a cluster service version (CSV) is created in the namespace. The synchronized label inherits the permissions of the service accounts in the namespace.

Procedure

- To enable pod security admission label synchronization in a namespace, set the value of the **security.openshift.io/scc.podSecurityLabelSync** label to **true**.
  Run the following command:

```
$ oc label namespace <namespace> security.openshift.io/scc.podSecurityLabelSync=true
```

NOTE

You can use the --overwrite flag to reverse the effects of the pod security label synchronization in a namespace.

CHAPTER 7. USING OPERATORS WITH MICROSHIFT

You can use Operators with MicroShift to create applications that monitor the running services in your cluster. Operators can manage applications and their resources, such as deploying a database or message bus. As customized software running inside your cluster, Operators can be used to implement and automate common operations.

Operators offer a more localized configuration experience and integrate with Kubernetes APIs and CLI tools such as **kubectl** and **oc**. Operators are designed specifically for your applications. Operators enable you to configure components instead of modifying a global configuration file.

MicroShift applications are generally expected to be deployed in static environments. However, Operators are available if helpful in your use case. To determine the compatibility of an Operator with MicroShift, check the Operator documentation.

7.1. HOW TO USE OPERATORS WITH MICROSHIFT CLUSTERS

There are two ways to use Operators for your MicroShift clusters:

7.1.1. Manifests for Operators

- Operators can be installed and managed directly by using manifests. You can use the **kustomize** configuration management tool with MicroShift to deploy an application. Use the same steps to install Operators with manifests. See Using Kustomize manifests to deploy applications and Using manifests example for details.

7.1.2. Operator Lifecycle Manager for Operators

- You can also install add-on Operators to a MicroShift cluster using Operator Lifecycle Manager (OLM). OLM can be used to manage both custom Operators and Operators that are widely available. Building catalogs is required to use OLM with MicroShift. For details, see Using Operator Lifecycle Manager with MicroShift.

CHAPTER 8. USING OPERATOR LIFECYCLE MANAGER WITH MICROSHIFT

The Operator Lifecycle Manager (OLM) package manager is used in MicroShift for installing and running optional add-on Operators.

- Cluster Operators as applied in OpenShift Container Platform are not used in MicroShift.
- You must create your own catalogs for the add-on Operators you want to use with your applications. Catalogs are not provided by default.
  - Each catalog must have an accessible **CatalogSource** added to a cluster, so that the OLM catalog Operator can use the catalog for content.
- You must use the CLI to conduct OLM activities with MicroShift. The console and OperatorHub GUIs are not available.
  - Use the Operator Package Manager **opm** CLI with network-connected clusters, or for building catalogs for custom Operators that use an internal registry.
  - To mirror your catalogs and Operators for disconnected or offline clusters, install the oc-mirror OpenShift CLI plugin.

IMPORTANT

Before using an Operator, verify with the provider that the Operator is supported on Red Hat build of MicroShift.

8.1. DETERMINING YOUR OLM INSTALLATION TYPE

You can install the OLM package manager for use with MicroShift 4.15 or newer versions. There are different ways to install OLM for MicroShift clusters, depending on your use case.

- You can install the **microshift-olm** RPM at the same time you install the MicroShift RPM on Red Hat Enterprise Linux (RHEL).
- You can install the **microshift-olm** on an existing MicroShift 4.15. Restart the MicroShift service after installing OLM for the changes to apply. See Installing the Operator Lifecycle Manager (OLM) from an RPM package.
- You can embed OLM in a Red Hat Enterprise Linux for Edge (RHEL for Edge) image. See Adding the Operator Lifecycle Manager (OLM) service to a blueprint .

8.2. NAMESPACE USE IN MICROSHIFT

The **microshift-olm** RPM creates the three default namespaces: one for running OLM, and two for catalog and Operator installation. You can create additional namespaces as needed for your use case.

8.2.1. Default namespaces

The following table lists the default namespaces and a brief description of how each namespace works.

Table 8.1. Default namespaces created by OLM for MicroShift

Default Namespace	Details
openshift-operator-lifecycle-manager	The OLM package manager runs in this namespace.
openshift-marketplace	The global namespace. Empty by default. To make the catalog source to be available globally to users in all namespaces, set the **openshift-marketplace** namespace in the catalog-source YAML.
openshift-operators	The default namespace where Operators run in MicroShift. Operators that reference catalogs in the **openshift-operators** namespace must have the **AllNamespaces** watch scope.

8.2.2. Custom namespaces

If you want to use a catalog and Operator together in a single namespace, then you must create a custom namespace. After you create the namespace, you must create the catalog in that namespace. All Operators running in the custom namespace must have the same single-namespace watch scope.

8.3. ABOUT BUILDING OPERATOR CATALOGS

To use Operator Lifecycle Manager (OLM) with MicroShift, you must build custom Operator catalogs that you can then manage with OLM. The standard catalogs that are included with OpenShift Container Platform are not included with MicroShift.

8.3.1. File-based Operator catalogs

You can create catalogs for your custom Operators or filter catalogs of widely available Operators. You can combine both methods to create the catalogs needed for your specific use case. To run MicroShift with your own Operators and OLM, make a catalog by using the file-based catalog structure.

- For details, see Managing custom catalogs and Example catalog.
- See also **opm** CLI reference.

IMPORTANT

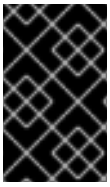
- When adding a catalog source to a cluster , set the **securityContextConfig** value to **restricted** in the **catalogSource.yaml** file. Ensure that your catalog can run with **restricted** permissions.

Additional resources

- **opm** CLI reference
- About Operator catalogs
- For instructions about creating file-based catalogs by using the **opm** CLI, see Managing custom catalogs

8.4. HOW TO DEPLOY OPERATORS

After you create and deploy your custom catalog, you must create a Subscription custom resource (CR) that can access the catalog and install the Operators you choose. Where Operators run depends on the namespace in which you create the Subscription CR.



IMPORTANT

Operators in OLM have a watch scope. For example, some Operators only support watching their own namespace, while others support watching every namespace in the cluster. All Operators installed in a given namespace must have the same watch scope.

8.4.1. Connectivity and Operator deployment

Operators can be deployed anywhere a catalog is running.

- For clusters that are connected to the internet, mirroring images is not required. Images can be pulled over the network.
- For restricted networks in which MicroShift has access to an internal network only, images must be mirrored to an internal registry.
- For use cases in which MicroShift clusters are completely offline, all images must be embedded into an **osbuild** blueprint.

Additional resources

- Operator group membership

8.4.2. Adding OLM-based Operators to a networked cluster using the global namespace

To deploy different operators to different namespaces, use this procedure. For MicroShift clusters that have network connectivity, Operator Lifecycle Manager (OLM) can access sources hosted on remote registries. The following procedure lists the basic steps of using configuration files to install an Operator that uses the global namespace.

NOTE

To use an Operator installed in a different namespace, or in more than one namespace, make sure that the catalog source and the Subscription CR that references the Operator are running in the **openshift-marketplace** namespace.

Prerequisites

- The OpenShift CLI (**oc**) is installed.
- Operator Lifecycle Manager (OLM) is installed.
- You have created a custom catalog in the global namespace.

Procedure

1. Confirm that OLM is running by using the following command:

```
$ oc -n openshift-operator-lifecycle-manager get pod -l app=olm-operator
```

Example output

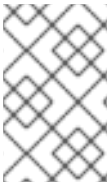
```
NAME                       READY   STATUS    RESTARTS   AGE
olm-operator-85b5c6786-n6kbc   1/1     Running   0          2m24s
```

2. Confirm that the OLM catalog Operator is running by using the following command:

```
$ oc -n openshift-operator-lifecycle-manager get pod -l app=catalog-operator
```

Example output

```
NAME                            READY   STATUS    RESTARTS   AGE
catalog-operator-5fc7f857b6-tj8cf   1/1     Running   0          2m33s
```



NOTE

The following steps assume you are using the global namespace, **openshift-marketplace**. The catalog must run in the same namespace as the Operator. The Operator must support the **AllNamespaces** mode.

1. Create the **CatalogSource** object by using the following example YAML:

Example catalog source YAML

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: operatorhubio-catalog
  namespace: openshift-marketplace 1
spec:
  sourceType: grpc
  image: quay.io/operatorhubio/catalog:latest
  displayName: Community Operators 2
  publisher: OperatorHub.io
  grpcPodConfig:
    securityContextConfig: restricted 3
  updateStrategy:
    registryPoll:
      interval: 60m
```

1. The global namespace. Setting the **metadata.namespace** to **openshift-marketplace** enables the catalog to run in all namespaces. Subscriptions in any namespace can reference catalogs created in the **openshift-marketplace** namespace.

2. Community Operators are not installed by default with OLM for {miroshift-short}. Listed here for example only.

3. The value of **securityContextConfig** must be set to **restricted** for MicroShift.

2. Apply the **CatalogSource** configuration by running the following command:

```
$ oc apply -f _<my-catalog-source.yaml>_ 1
```

1 Replace *<my-catalog-source.yaml>* with your catalog source configuration file name. In this example, **catalogsource.yaml** is used.

Example output

```
catalogsource.operators.coreos.com/operatorhubio-catalog created
```

3. To verify that the catalog source is applied, check for the **READY** state by using the following command:

```
$ oc describe catalogsources.operators.coreos.com -n openshift-marketplace operatorhubio-catalog
```

Example output

```
Name:         operatorhubio-catalog
Namespace:    openshift-marketplace
Labels:       <none>
Annotations:  <none>
API Version:  operators.coreos.com/v1alpha1
Kind:         CatalogSource
Metadata:
  Creation Timestamp:  2024-01-31T09:55:31Z
  Generation:          1
  Resource Version:    1212
  UID:                 4edc1a96-83cd-4de9-ac8c-c269ca895f3e
Spec:
  Display Name:  Community Operators
  Grpc Pod Config:
    Security Context Config:  restricted
  Image:                      quay.io/operatorhubio/catalog:latest
  Publisher:                  OperatorHub.io
  Source Type:                grpc
  Update Strategy:
    Registry Poll:
      Interval:  60m
Status:
  Connection State:
    Address:              operatorhubio-catalog.openshift-marketplace.svc:50051
    Last Connect:         2024-01-31T09:55:57Z
    Last Observed State:  READY 1
  Registry Service:
    Created At:         2024-01-31T09:55:31Z
    Port:               50051
    Protocol:           grpc
    Service Name:       operatorhubio-catalog
    Service Namespace:  openshift-marketplace
Events:                 <none>
```

1 The status is reported as **READY**.

4. Confirm that the catalog source is running by using the following command:

```
$ oc get pods -n openshift-marketplace -l olm.catalogSource=operatorhubio-catalog
```

Example output

```
NAME                          READY   STATUS    RESTARTS   AGE
operatorhubio-catalog-x24nh   1/1     Running   0          59s
```

5. Create a Subscription CR configuration file by using the following example YAML:

Example Subscription custom resource YAML

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: my-cert-manager
  namespace: openshift-operators
spec:
  channel: stable
  name: cert-manager
  source: operatorhubio-catalog
  sourceNamespace: openshift-marketplace 1
```

1 The global namespace. Setting the **sourceNamespace** value to **openshift-marketplace** enables Operators to run in multiple namespaces if the catalog also runs in the **openshift-marketplace** namespace.

6. Apply the Subscription CR configuration by running the following command:

```
$ oc apply -f _<my-subscription-cr.yaml>_ 1
```

1 Replace *<my-subscription-cr.yaml>* with your Subscription CR filename. In this example, **sub.yaml** is used.

Example output

```
subscription.operators.coreos.com/my-cert-manager created
```

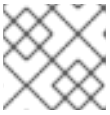
7. You can create a configuration file for the specific Operand you want to use and apply it now.

Verification

1. Verify that your Operator is running by using the following command:

```
$ oc get pods -n openshift-operators 1
```

1 The namespace from the Subscription CR is used.



NOTE

Allow a minute or two for the Operator start.

Example output

```
NAME                                  READY   STATUS    RESTARTS   AGE
cert-manager-7df8994ddb-4vrkr         1/1     Running   0          19s
cert-manager-cainjector-5746db8fd7-69442   1/1     Running   0          18s
cert-manager-webhook-f858bf58b-748nt      1/1     Running   0          18s
```

8.4.3. Adding OLM-based Operators to a networked cluster in a specific namespace

Use this procedure if you want to specify a namespace for an Operator, for example, **olm-microshift**. In this example, the catalog is scoped and available in the global **openshift-marketplace** namespace. The Operator uses content from the global namespace, but runs only in the **olm-microshift** namespace. For MicroShift clusters that have network connectivity, Operator Lifecycle Manager (OLM) can access sources hosted on remote registries.

IMPORTANT

All of the Operators installed in a specific namespace must have the same watch scope. In this case, the watch scope is **OwnNamespace**.

Prerequisites

- The OpenShift CLI (**oc**) is installed.
- Operator Lifecycle Manager (OLM) is installed.
- You have created a custom catalog that is running in the global namespace.

Procedure

1. Confirm that OLM is running by using the following command:

```
$ oc -n openshift-operator-lifecycle-manager get pod -l app=olm-operator
```

Example output

```
NAME                        READY   STATUS    RESTARTS   AGE
olm-operator-85b5c6786-n6kbc   1/1     Running   0          16m
```

2. Confirm that the OLM catalog Operator is running by using the following command:

```
$ oc -n openshift-operator-lifecycle-manager get pod -l app=catalog-operator
```

Example output

```
NAME                             READY   STATUS    RESTARTS   AGE
catalog-operator-5fc7f857b6-tj8cf   1/1     Running   0          16m
```

3. Create a namespace by using the following example YAML:

Example namespace YAML

```
apiVersion: v1
kind: Namespace
metadata:
  name: olm-microshift
```

4. Apply the namespace configuration using the following command:

```
$ oc apply -f _<ns.yaml>_ 1
```

1 Replace *<ns.yaml>* with the name of your namespace configuration file. In this example, **olm-microshift** is used.

Example output

```
namespace/olm-microshift created
```

5. Create the Operator group YAML by using the following example YAML:

Example Operator group YAML

```
kind: OperatorGroup
apiVersion: operators.coreos.com/v1
metadata:
  name: og
  namespace: olm-microshift
spec: 1
  targetNamespaces:
  - olm-microshift
```

1 For Operators using the global namespace, omit the **spec.targetNamespaces** field and values.

6. Apply the Operator group configuration by running the following command:

```
$ oc apply -f _<og.yaml>_ 1
```

1 Replace *<og.yaml>* with the name of your operator group configuration file.

Example output

```
operatorgroup.operators.coreos.com/og created
```

7. Create the **CatalogSource** object by using the following example YAML:

Example catalog source YAML

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: operatorhubio-catalog
  namespace: openshift-marketplace 1
spec:
  sourceType: grpc
  image: quay.io/operatorhubio/catalog:latest
  displayName: Community Operators 2
  publisher: OperatorHub.io
  grpcPodConfig:
    securityContextConfig: restricted 3
  updateStrategy:
    registryPoll:
      interval: 60m
```

1. The global namespace. Setting the **metadata.namespace** to **openshift-marketplace** enables the catalog to run in all namespaces. Subscriptions CRs in any namespace can reference catalogs created in the **openshift-marketplace** namespace.
2. Community Operators are not installed by default with OLM for MicroShift. Listed here for example only.
3. The value of **securityContextConfig** must be set to **restricted** for MicroShift.

8. Apply the **CatalogSource** configuration by running the following command:

```
$ oc apply -f _<my-catalog-source.yaml>_ 1
```

1. Replace *<my-catalog-source.yaml>* with your catalog source configuration file name.

9. To verify that the catalog source is applied, check for the **READY** state by using the following command:

```
$ oc describe catalogsources.operators.coreos.com -n openshift-marketplace operatorhubio-catalog
```

Example output

```
Name:         operatorhubio-catalog
Namespace:    openshift-marketplace
Labels:       <none>
Annotations:  <none>
API Version:  operators.coreos.com/v1alpha1
Kind:         CatalogSource
Metadata:
  Creation Timestamp:  2024-01-31T10:09:46Z
  Generation:          1
  Resource Version:    2811
  UID:                 60ce4a36-86d3-4921-b9fc-84d67c28df48
Spec:
  Display Name:  Community Operators
```

```
  Grpc Pod Config:
    Security Context Config:  restricted
  Image:                      quay.io/operatorhubio/catalog:latest
  Publisher:                  OperatorHub.io
  Source Type:                grpc
  Update Strategy:
    Registry Poll:
      Interval:  60m
Status:
  Connection State:
    Address:              operatorhubio-catalog.openshift-marketplace.svc:50051
    Last Connect:         2024-01-31T10:10:04Z
    Last Observed State:  READY (1)
  Registry Service:
    Created At:         2024-01-31T10:09:46Z
    Port:               50051
    Protocol:           grpc
    Service Name:       operatorhubio-catalog
    Service Namespace:  openshift-marketplace
Events:                 <none>
```

1. The status is reported as **READY**.

10. Confirm that the catalog source is running by using the following command:

```
$ oc get pods -n openshift-marketplace -l olm.catalogSource=operatorhubio-catalog
```

Example output

```
NAME                          READY   STATUS    RESTARTS   AGE
operatorhubio-catalog-j7sc8   1/1     Running   0          43s
```

11. Create a Subscription CR configuration file by using the following example YAML:

Example Subscription custom resource YAML

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: my-gitlab-operator-kubernetes
  namespace: olm-microshift (1)
spec:
  channel: stable
  name: gitlab-operator-kubernetes
  source: operatorhubio-catalog
  sourceNamespace: openshift-marketplace (2)
```

1. The specific namespace. Operators reference the global namespace for content, but run in the **olm-microshift** namespace.

2. The global namespace. Subscriptions CRs in any namespace can reference catalogs created in the **openshift-marketplace** namespace.

12. Apply the Subscription CR configuration by running the following command:

```
$ oc apply -f _<my-subscription-cr.yaml>_
```

Example output

```
subscription.operators.coreos.com/my-gitlab-operator-kubernetes
```

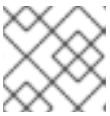
13. You can create a configuration file for the specific Operand you want to use and apply it now.

Verification

1. Verify that your Operator is running by using the following command:

```
$ oc get pods -n olm-microshift 1
```

1 The namespace from the Subscription CR is used.



NOTE

Allow a minute or two for the Operator start.

Example output

```
NAME                                  READY   STATUS    RESTARTS   AGE
gitlab-controller-manager-69bb6df7d6-g7ntx   2/2     Running   0          3m24s
```

Additional resources

- Updating installed Operators
- Deleting Operators from a cluster using the CLI

8.4.4. About adding OLM-based Operators to an offline cluster

For MicroShift clusters that are installed on disconnected or offline clusters, Operator Lifecycle Manager (OLM) by default cannot access sources hosted on remote registries because those remote sources require full internet connectivity.

The following steps are required to use OLM-based Operators in offline situations:

- Include OLM in your container image list for your mirror registry.
- Configure the system to use the mirror by updating your CRI-O configuration directly. **ImageContentSourcePolicy** is not supported in MicroShift.
- Add a **CatalogSource** object to the cluster so that the OLM catalog Operator can use the local catalog on the mirror registry.
- Ensure that MicroShift is installed to run in an offline capacity.
- Ensure that the network settings are configured to run in a disconnected mode.

After enabling OLM in an offline cluster, you can continue to use your unrestricted workstation to keep your local catalog sources updated as newer versions of Operators are released.

Additional resources

- Using Operator Lifecycle Manager on restricted networks for more information.
- Mirroring images for a disconnected installation using the oc-mirror plugin
- Configuring hosts for mirror registry access
- Configuring network settings for fully disconnected hosts
- Getting the mirror registry container image list
- Embedding in a RHEL for Edge image for offline use

CHAPTER 9. AUTOMATING APPLICATION MANAGEMENT WITH THE GITOPS CONTROLLER

GitOps with Argo CD for MicroShift is a lightweight, optional add-on controller derived from the Red Hat OpenShift GitOps Operator. GitOps for MicroShift uses the command-line interface (CLI) of Argo CD to interact with the GitOps controller that acts as the declarative GitOps engine. You can consistently configure and deploy Kubernetes-based infrastructure and applications across clusters and development lifecycles.

9.1. WHAT YOU CAN DO WITH THE GITOPS AGENT

By using the GitOps with Argo CD agent with MicroShift, you can utilize the following principles:

- Implement application lifecycle management.
    - Create and manage your clusters and application configuration files using the core principles of developing and maintaining software in a Git repository.
    - You can update the single repository and GitOps automates the deployment of new applications or updates to existing ones.
    - For example, if you have 1,000 edge devices, each using MicroShift and a local GitOps agent, you can easily add or update an application on all 1,000 devices with just one change in your central Git repository.
- The Git repository contains a declarative description of the infrastructure you need in your specified environment and contains an automated process to make your environment match the described state.
- You can also use the Git repository as an audit trail of changes so that you can create processes based on Git flows such as review and approval for merging pull requests that implement configuration changes.

9.2. LIMITATIONS OF USING THE GITOPS AGENT WITH MICROSHIFT

GitOps with Argo CD for MicroShift has the following differences from the Red Hat OpenShift GitOps Operator:

- The **gitops-operator** component is not used with MicroShift.
- To maintain the small resource use of MicroShift, the Argo CD web console is not available. You can use the Argo CD CLI or use a pull-based approach.
- Because MicroShift is single-node, there is no multi-cluster support. Each instance of MicroShift is paired with a local GitOps agent.
- Use sos reports for debugging because **oc adm must-gather** is not available in MicroShift.

9.3. ADDITIONAL RESOURCES

- Red Hat OpenShift GitOps
- Using sos reports