OpenShift Container Platform 4.16

Windows Container Support for OpenShift

Red Hat OpenShift for Windows Containers Guide

Last Updated: 2024-09-11

OpenShift Container Platform 4.16 Windows Container Support for OpenShift

Red Hat OpenShift for Windows Containers Guide

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

Red Hat OpenShift for Windows Containers provides built-in support for running Microsoft Windows Server containers on OpenShift Container Platform. This guide provides all the details.

Table of Contents

CHAPTER 1. RED HAT OPENSHIFT SUPPORT FOR WINDOWS CONTAINERS OVERVIEW 4

CHAPTER 2. RELEASE NOTES 5
2.1. RED HAT OPENSHIFT SUPPORT FOR WINDOWS CONTAINERS RELEASE NOTES 5
2.1.1. Windows Machine Config Operator numbering 5
2.1.2. Release notes for Red Hat Windows Machine Config Operator 10.16.1 5
2.1.2.1. Bug fixes 5
2.2. RELEASE NOTES FOR PAST RELEASES OF THE WINDOWS MACHINE CONFIG OPERATOR 5
2.2.1. Release notes for Red Hat Windows Machine Config Operator 10.16.0 5
2.2.1.1. New features and improvements 5
2.2.1.1.1. WMCO is now supported in disconnected networks 5
2.2.1.1.2. WMCO can pull images from mirrored registries 6
2.2.1.1.3. Filesystem metrics now display for Windows nodes 6
2.2.1.1.4. Pod network metrics now display for the pods on Windows nodes 6
2.2.1.1.5. Pod CPU and memory metrics now display for the pods on Windows nodes 6
2.2.1.1.6. Kubernetes upgrade 6
2.2.1.2. Bug fixes 6
2.2.2. Windows Machine Config Operator prerequisites 7
2.2.2.1. WMCO 10.16.0 supported platforms and Windows Server versions 7
2.2.2.2. Supported networking 8
2.2.3. Known limitations 9

CHAPTER 3. GETTING SUPPORT 11

CHAPTER 4. UNDERSTANDING WINDOWS CONTAINER WORKLOADS 12
4.1. WINDOWS WORKLOAD MANAGEMENT 12
4.2. WINDOWS NODE SERVICES 14

CHAPTER 5. ENABLING WINDOWS CONTAINER WORKLOADS 16
Prerequisites 16
5.1. INSTALLING THE WINDOWS MACHINE CONFIG OPERATOR 16
5.1.1. Installing the Windows Machine Config Operator using the web console 16
5.1.2. Installing the Windows Machine Config Operator using the CLI 17
5.2. CONFIGURING A SECRET FOR THE WINDOWS MACHINE CONFIG OPERATOR 19
5.3. USING WINDOWS CONTAINERS IN A PROXY-ENABLED CLUSTER 20
5.4. USING WINDOWS CONTAINERS WITH A MIRROR REGISTRY 20
5.4.1. Understanding image registry repository mirroring 21
5.4.2. Configuring image registry repository mirroring 22
5.5. ADDITIONAL RESOURCES 26

CHAPTER 6. CREATING WINDOWS MACHINE SETS 27
6.1. CREATING A WINDOWS MACHINE SET ON AWS 27
Prerequisites 27
6.1.1. Machine API overview 27
6.1.2. Sample YAML for a Windows MachineSet object on AWS 29
6.1.3. Creating a compute machine set 30
6.1.4. Additional resources 32
6.2. CREATING A WINDOWS MACHINE SET ON AZURE 32
Prerequisites 32
6.2.1. Machine API overview 33
6.2.2. Sample YAML for a Windows MachineSet object on Azure 34
6.2.3. Creating a compute machine set 35

6.2.4. Additional resources 38
6.3. CREATING A WINDOWS MACHINE SET ON GCP 38
Prerequisites 38
6.3.1. Machine API overview 38
6.3.2. Sample YAML for a Windows MachineSet object on GCP 39
6.3.3. Creating a compute machine set 41
6.3.4. Additional resources 43
6.4. CREATING A WINDOWS MACHINESET OBJECT ON NUTANIX 43
Prerequisites 43
6.4.1. Machine API overview 43
6.4.2. Sample YAML for a Windows MachineSet object on Nutanix 44
6.4.3. Creating a compute machine set 46
6.4.4. Additional resources 48
6.5. CREATING A WINDOWS MACHINE SET ON VSPHERE 48
Prerequisites 48
6.5.1. Machine API overview 49
6.5.2. Preparing your vSphere environment for Windows container workloads 50
6.5.2.1. Creating the vSphere Windows VM golden image 50
6.5.2.1.1. Additional resources 53
6.5.2.2. Enabling communication with the internal API server for the WMCO on vSphere 53
6.5.3. Sample YAML for a Windows MachineSet object on vSphere 54
6.5.4. Creating a compute machine set 56
6.5.5. Additional resources 58

CHAPTER 7. SCHEDULING WINDOWS CONTAINER WORKLOADS 59
Prerequisites 59
7.1. WINDOWS POD PLACEMENT 59
Additional resources 59
7.2. CREATING A RUNTIMECLASS OBJECT TO ENCAPSULATE SCHEDULING MECHANISMS 60
7.3. SAMPLE WINDOWS CONTAINER WORKLOAD DEPLOYMENT 61
7.4. SUPPORT FOR WINDOWS CSI DRIVERS 62
7.5. SCALING A COMPUTE MACHINE SET MANUALLY 63

CHAPTER 8. WINDOWS NODE UPGRADES 65
8.1. WINDOWS MACHINE CONFIG OPERATOR UPGRADES 65

CHAPTER 9. USING BRING-YOUR-OWN-HOST (BYOH) WINDOWS INSTANCES AS NODES 66
9.1. CONFIGURING A BYOH WINDOWS INSTANCE 66
9.2. REMOVING BYOH WINDOWS INSTANCES 67

CHAPTER 10. REMOVING WINDOWS NODES 68
10.1. DELETING A SPECIFIC MACHINE 68

CHAPTER 11. DISABLING WINDOWS CONTAINER WORKLOADS 69
11.1. UNINSTALLING THE WINDOWS MACHINE CONFIG OPERATOR 69
11.2. DELETING THE WINDOWS MACHINE CONFIG OPERATOR NAMESPACE 69
Additional resources 69

CHAPTER 1. RED HAT OPENSHIFT SUPPORT FOR WINDOWS CONTAINERS OVERVIEW

You can add Windows nodes either by creating a compute machine set or by specifying existing Bring-Your-Own-Host (BYOH) Window instances through a configuration map.

NOTE

Compute machine sets are not supported for bare metal or provider agnostic clusters.

For workloads including both Linux and Windows, OpenShift Container Platform allows you to deploy Windows workloads running on Windows Server containers while also providing traditional Linux workloads hosted on Red Hat Enterprise Linux CoreOS (RHCOS) or Red Hat Enterprise Linux (RHEL). For more information, see getting started with Windows container workloads.

You need the WMCO to run Windows workloads in your cluster. The WMCO orchestrates the process of deploying and managing Windows workloads on a cluster. For more information, see how to enable Windows container workloads.

You can create a Windows **MachineSet** object to create infrastructure Windows machine sets and related machines so that you can move supported Windows workloads to the new Windows machines. You can create a Windows **MachineSet** object on multiple platforms.

You can schedule Windows workloads to Windows compute nodes.

You can perform Windows Machine Config Operator upgrades to ensure that your Windows nodes have the latest updates.

You can remove a Windows node by deleting a specific machine.

You can use Bring-Your-Own-Host (BYOH) Windows instances to repurpose Windows Server VMs and bring them to OpenShift Container Platform. BYOH Windows instances benefit users who are looking to mitigate major disruptions in the event that a Windows server goes offline. You can use BYOH Windows instances as nodes on OpenShift Container Platform 4.8 and later versions.

You can disable Windows container workloads by performing the following:

- Uninstalling the Windows Machine Config Operator
- Deleting the Windows Machine Config Operator namespace

CHAPTER 2. RELEASE NOTES

2.1. RED HAT OPENSHIFT SUPPORT FOR WINDOWS CONTAINERS RELEASE NOTES

The release notes for Red Hat OpenShift for Windows Containers tracks the development of the Windows Machine Config Operator (WMCO), which provides all Windows container workload capabilities in OpenShift Container Platform.

2.1.1. Windows Machine Config Operator numbering

Y-stream releases of the WMCO are in step with OpenShift Container Platform, with only z-stream releases between OpenShift Container Platform releases. The WMCO numbering reflects the associated OpenShift Container Platform version in the y-stream position. For example, the current release of WMCO is associated with OpenShift Container Platform version 4.16. Thus, the numbering is WMCO 10.15.z.

2.1.2. Release notes for Red Hat Windows Machine Config Operator 10.16.1

This release of the WMCO provides new features and bug fixes for running Windows compute nodes in an OpenShift Container Platform cluster. The components of the WMCO 10.16.1 were released in RHSA-2024:5749.

2.1.2.1. Bug fixes

- Previously, if a Windows VM had its PowerShell **ExecutionPolicy** set to **Restricted**, the Windows Instance Config Daemon (WICD) could not run the commands on that VM that are necessary for creating Windows nodes. With this fix, the WICD now bypasses the execution policy on the VM when running PowerShell commands. As a result, the WICD can create Windows nodes on the VM as expected. (**OCPBUGS-37609**)

2.2. RELEASE NOTES FOR PAST RELEASES OF THE WINDOWS MACHINE CONFIG OPERATOR

The following release notes are for previous versions of the Windows Machine Config Operator (WMCO).

2.2.1. Release notes for Red Hat Windows Machine Config Operator 10.16.0

This release of the WMCO provides bug fixes for running Windows compute nodes in an OpenShift Container Platform cluster. The components of the WMCO 10.16.0 were released in RHBA-2024:5014.

2.2.1.1. New features and improvements

2.2.1.1.1. WMCO is now supported in disconnected networks

The WMCO is now supported in environments with disconnected networks, which is a cluster that is intentionally impeded from reaching the internet, also known as restricted or air-gapped clusters.

For more information, see Using Windows containers with a mirror registry .

2.2.1.1.2. WMCO can pull images from mirrored registries

The WMCO can now use both **ImageDigestMirrorSet** (IDMS) and **ImageTagMirrorSet** (ITMS) objects to pull images from mirrored registries.

For more information, see Understanding image registry repository mirroring

2.2.1.1.3. Filesystem metrics now display for Windows nodes

The **Filesystem** metrics are now available for Windows nodes in the **Utilization** tile of the **Node details** page in the OpenShift Container Platform web console. You can query the metrics by running Prometheus Query Language (PromQL) queries. The charts previously reported **No datapoints found**.

2.2.1.1.4. Pod network metrics now display for the pods on Windows nodes

The **Network in** and **Network out** charts are now available for Windows pods on the **Pod details** page in the OpenShift Container Platform web console. You can query the metrics by running PromQL queries. The charts previously reported **No datapoints found**.

2.2.1.1.5. Pod CPU and memory metrics now display for the pods on Windows nodes

The CPU and memory usage metrics are now available for Windows pods on the **Pods** and **Pod details** pages in the OpenShift Container Platform web console. You can query the metrics by running PromQL queries. The chart previously reported **No datapoints found**.

2.2.1.1.6. Kubernetes upgrade

The WMCO now uses Kubernetes 1.29.

2.2.1.2. Bug fixes

Because the WICD service account was missing a required secret, the WMCO was unable to properly configure Windows nodes in a Nutanix cluster. With this fix, the WMCO creates a long-lived token secret for the WICD service account. As a result, the WMCO is able to configure a Windows node on Nutanix. (**OCPBUGS-22680**)

Previously, the WMCO performed a sanitization step that incorrectly replaced commas with semicolons in a user's cluster-wide proxy configuration. This behavior caused Windows to ignore the values set in the **noProxy** environment variable. As a consequence, the WMCO incorrectly sent traffic through the proxy for the endpoints specified in the **no-proxy** parameter. With this fix, the sanitization step that replaced commas with semicolons was removed. As a result, web requests from a Windows node to a cluster-internal endpoint or an endpoint that exists in the **no-proxy** parameter do not go through the proxy. (**OCPBUGS-24264**)

Previously, because of bad logic in the networking configuration script, the WMCO was incorrectly reading carriage returns in the containderd CNI configuration file as changes, and identified the file as modified. This bahavior caused the CNI configuration to be unnecessarily reloaded, potentially resulting in container restarts and brief network outages. With this fix, the WMCO now reloads the CNI configuration only when the CNI configuration is actually modified. (**OCPBUGS-2887**)

Previously, because of routing issues present in Windows Server 2019, under certain conditions and after more than one hour of running time, workloads on Windows Server 2019 could have experienced packet loss when communicating with other containers in the cluster. This fix enables Direct Server Return (DSR) routing within kube-proxy. As a result, DSR now causes request and response traffic to use a different network path, circumventing the bug within Windows Server 2019. (**OCPBUGS-26761**)

Previously, the kubelet on Windows nodes was unable to authenticate with private Amazon Elastic Container Registries (ECR). Because of this error, the kubelet was not able to pull images from these registries. With this fix, the kubelet is able to pull images from these registries as expected. (OCPBUGS-26602)

Previously, on Azure clusters the WMCO would check if an external Cloud Controller Manager (CCM) was being used on the cluster. If a CCM was being used, the Operator would adjust configuration logic accordingly. Because the status condition that the WMCO used to check for the CCM was removed, the WMCO proceeded as if a CCM was not in use. This fix removes the check. As a result, the WMCO always configures the required logic on Azure clusters. (OCPBUGS-31626)

Previously, the WMCO logged error messages when a command that was run through an SSH connection to a Windows instance failed. This behavior was incorrect because some commands are expected to fail. For example, when the WMCO reboots a node, the Operator runs PowerShell commands on the instance until they fail, meaning the SSH connection rebooted as expected. With this fix, only actual errors are now logged. (OCPBUGS-20255)

Previously, after rotating the **kube-apiserver-to-kubelet-client-ca** certificate, the contents of the **kubetl-ca.crt** file on Windows nodes was not populated correctly. With this fix, after certificate rotation, the **kubetl-ca.crt** file contains the correct certificates. (OCPBUGS-22237)

Previously, because of a missing DNS suffix in the kubelet host name on instances that are part of a Windows AD domain controller, the cloud provider failed to find VMs by name. With this fix, the DNS suffix is now included in the host name resolution. As a result, the WMCO is able to configure and join Windows instances that are part of AD domain controller. (OCPBUGS-34758)

Previously, registry certificates provided to the cluster by a user were not loaded into the Windows trust store on each node. As a consequence, image pulls from a mirror registry failed, because a self-signed CA is required. With this fix, registry certificates are loaded into the Windows trust store on each node. As a result, images can be pulled from mirror registries with self-signed CAs. (OCPBUGS-36408)

Previously, if there were multiple service account token secrets in the WMCO namespace, scaling Windows nodes would fail. With this fix, the WMCO uses only the secret it creates, ignoring any other service account token secrets in the WMCO namespace. As a result, Windows nodes scale properly. (OCPBUGS-37481)

Previously, if reverse DNS lookup failed due to an error, such as the reverse DNS lookup services being unavailable, the WMCO would not fall back to using the VM hostname to determine if a certificate signing requests (CSR) should be approved. As a consequence, Bring-Your-Own-Host (BYOH) Windows nodes configured with an IP address would not become available. With this fix, BYOH nodes are properly added if reverse DNS is not available. (OCPBUGS-36643)

2.2.2. Windows Machine Config Operator prerequisites

The following information details the supported platform versions, Windows Server versions, and networking configurations for the Windows Machine Config Operator. See the vSphere documentation for any information that is relevant to only that platform.

2.2.2.1. WMCO 10.16.0 supported platforms and Windows Server versions

The following table lists the Windows Server versions that are supported by WMCO 10.16.0, based on the applicable platform. Windows Server versions not listed are not supported and attempting to use them will cause errors. To prevent these errors, use only an appropriate version for your platform.

Platform	Supported Windows Server version
Amazon Web Services (AWS)	• Windows Server 2022, OS Build 20348.681 or later • Windows Server 2019, version 1809
Microsoft Azure	• Windows Server 2022, OS Build 20348.681 or later • Windows Server 2019, version 1809
VMware vSphere	Windows Server 2022, OS Build 20348.681 or later
Google Cloud Platform (GCP)	Windows Server 2022, OS Build 20348.681 or later
Nutanix	Windows Server 2022, OS Build 20348.681 or later
Bare metal or provider agnostic	• Windows Server 2022, OS Build 20348.681 or later • Windows Server 2019, version 1809

2.2.2.2. Supported networking

Hybrid networking with OVN-Kubernetes is the only supported networking configuration. See the additional resources below for more information on this functionality. The following tables outline the type of networking configuration and Windows Server versions to use based on your platform. You must specify the network configuration when you install the cluster.

NOTE

- The WMCO does not support OVN-Kubernetes without hybrid networking or OpenShift SDN.
- Dual NIC is not supported on WMCO-managed Windows instances.

Table 2.1. Platform networking support

Platform	Supported networking
Amazon Web Services (AWS)	Hybrid networking with OVN-Kubernetes
Microsoft Azure	Hybrid networking with OVN-Kubernetes
VMware vSphere	Hybrid networking with OVN-Kubernetes with a custom VXLAN port
Google Cloud Platform (GCP)	Hybrid networking with OVN-Kubernetes

Platform	Supported networking
Nutanix	Hybrid networking with OVN-Kubernetes
Bare metal or provider agnostic	Hybrid networking with OVN-Kubernetes

Table 2.2. Hybrid OVN-Kubernetes Windows Server support

Hybrid networking with OVN-Kubernetes	Supported Windows Server version
Default VXLAN port	• Windows Server 2022, OS Build 20348.681 or later • Windows Server 2019, version 1809
Custom VXLAN port	Windows Server 2022, OS Build 20348.681 or later

2.2.3. Known limitations

Note the following limitations when working with Windows nodes managed by the WMCO (Windows nodes):

- The following OpenShift Container Platform features are not supported on Windows nodes:
  - Image builds
  - OpenShift Pipelines
  - OpenShift Service Mesh
  - OpenShift monitoring of user-defined projects
  - OpenShift Serverless
  - Horizontal Pod Autoscaling
  - Vertical Pod Autoscaling
- The following Red Hat features are not supported on Windows nodes:
  - Red Hat Insights cost management
  - Red Hat OpenShift Local
- Dual NIC is not supported on WMCO-managed Windows instances.
- Windows nodes do not support workloads created by using deployment configs. You can use a deployment or other method to deploy workloads.

- Red Hat OpenShift support for Windows Containers does not support adding Windows nodes to a cluster through a trunk port. The only supported networking configuration for adding Windows nodes is through an access port that carries traffic for the VLAN.
- Red Hat OpenShift support for Windows Containers does not support any Windows operating system language other than English (United States).
- Due to a limitation within the Windows operating system, **clusterNetwork** CIDR addresses of class E, such as **240.0.0.0**, are not compatible with Windows nodes.
- Kubernetes has identified the following node feature limitations :
  - Huge pages are not supported for Windows containers.
  - Privileged containers are not supported for Windows containers.
- Kubernetes has identified several API compatibility issues .

CHAPTER 3. GETTING SUPPORT

Windows Container Support for Red Hat OpenShift is provided and available as an optional, installable component. Windows Container Support for Red Hat OpenShift is not part of the OpenShift Container Platform subscription. It requires an additional Red Hat subscription and is supported according to the Scope of coverage and Service level agreements.

You must have this separate subscription to receive support for Windows Container Support for Red Hat OpenShift. Without this additional Red Hat subscription, deploying Windows container workloads in production clusters is not supported. You can request support through the Red Hat Customer Portal.

For more information, see the Red Hat OpenShift Container Platform Life Cycle Policy document for Red Hat OpenShift support for Windows Containers .

If you do not have this additional Red Hat subscription, you can use the Community Windows Machine Config Operator, a distribution that lacks official support.

CHAPTER 4. UNDERSTANDING WINDOWS CONTAINER WORKLOADS

Red Hat OpenShift support for Windows Containers provides built-in support for running Microsoft Windows Server containers on OpenShift Container Platform. For those that administer heterogeneous environments with a mix of Linux and Windows workloads, OpenShift Container Platform allows you to deploy Windows workloads running on Windows Server containers while also providing traditional Linux workloads hosted on Red Hat Enterprise Linux CoreOS (RHCOS) or Red Hat Enterprise Linux (RHEL).

NOTE

Multi-tenancy for clusters that have Windows nodes is not supported. Clusters are considered *multi-tenant* when multiple workloads operate on shared infrastructure and resources. If one or more workloads running on an infrastructure cannot be trusted, the multi-tenant environment is considered *hostile*.

Hostile multi-tenant clusters introduce security concerns in all Kubernetes environments. Additional security features like pod security policies, or more fine-grained role-based access control (RBAC) for nodes, make exploiting your environment more difficult. However, if you choose to run hostile multi-tenant workloads, a hypervisor is the only security option you should use. The security domain for Kubernetes encompasses the entire cluster, not an individual node. For these types of hostile multi-tenant workloads, you should use physically isolated clusters.

Windows Server Containers provide resource isolation using a shared kernel but are not intended to be used in hostile multitenancy scenarios. Scenarios that involve hostile multitenancy should use Hyper-V Isolated Containers to strongly isolate tenants.

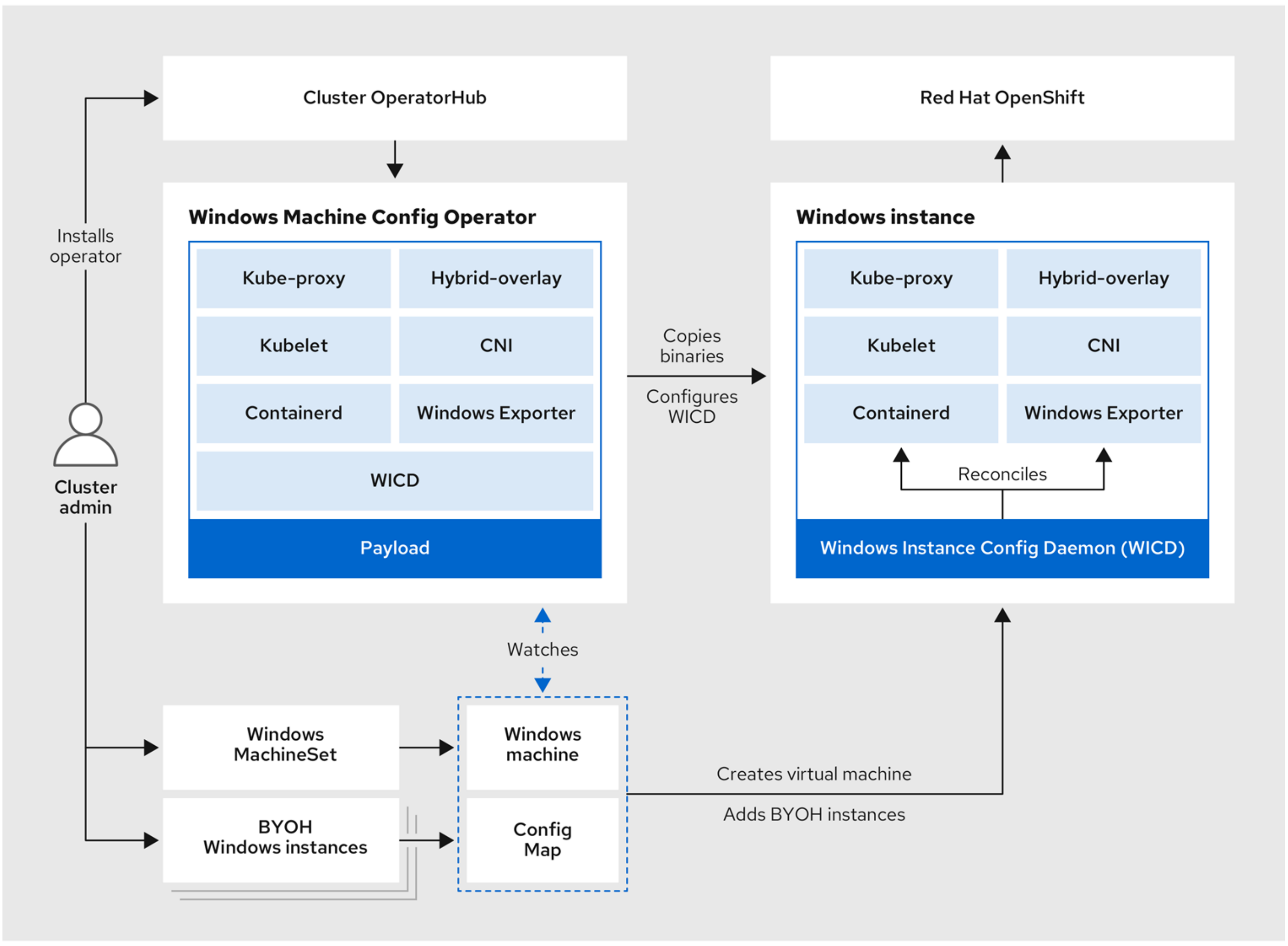
Additional resources

- See Configuring hybrid networking with OVN-Kubernetes

4.1. WINDOWS WORKLOAD MANAGEMENT

To run Windows workloads in your cluster, you must first install the Windows Machine Config Operator (WMCO). The WMCO is a Linux-based Operator that runs on Linux-based control plane and compute nodes. The WMCO orchestrates the process of deploying and managing Windows workloads on a cluster.

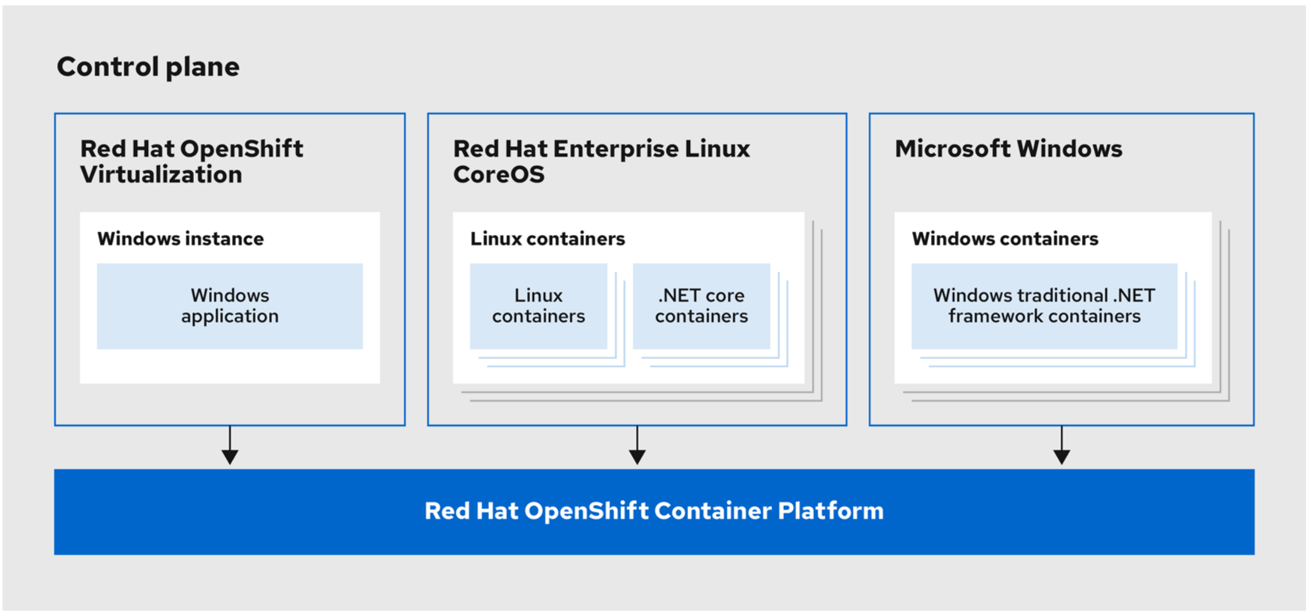
Figure 4.1. WMCO design



Before deploying Windows workloads, you must create a Windows compute node and have it join the cluster. The Windows node hosts the Windows workloads in a cluster, and can run alongside other Linux-based compute nodes. You can create a Windows compute node by creating a Windows compute machine set to host Windows Server compute machines. You must apply a Windows-specific label to the compute machine set that specifies a Windows OS image.

The WMCO watches for machines with the Windows label. After a Windows compute machine set is detected and its respective machines are provisioned, the WMCO configures the underlying Windows virtual machine (VM) so that it can join the cluster as a compute node.

Figure 4.2. Mixed Windows and Linux workloads


Control plane
Red Hat OpenShift Virtualization
Windows instance
Windows application
Red Hat Enterprise Linux CoreOS
Linux containers
Linux containers
.NET core containers
Microsoft Windows
Windows containers
Windows traditional .NET framework containers
Red Hat OpenShift Container Platform

The WMCO expects a predetermined secret in its namespace containing a private key that is used to interact with the Windows instance. WMCO checks for this secret during boot up time and creates a user data secret which you must reference in the Windows **MachineSet** object that you created. Then the WMCO populates the user data secret with a public key that corresponds to the private key. With this data in place, the cluster can connect to the Windows VM using an SSH connection.

After the cluster establishes a connection with the Windows VM, you can manage the Windows node using similar practices as you would a Linux-based node.

NOTE

The OpenShift Container Platform web console provides most of the same monitoring capabilities for Windows nodes that are available for Linux nodes. However, the ability to monitor workload graphs for pods running on Windows nodes is not available at this time.

Scheduling Windows workloads to a Windows node can be done with typical pod scheduling practices like taints, tolerations, and node selectors; alternatively, you can differentiate your Windows workloads from Linux workloads and other Windows-versioned workloads by using a **RuntimeClass** object.

4.2. WINDOWS NODE SERVICES

The following Windows-specific services are installed on each Windows node:

Service	Description
kubelet	Registers the Windows node and manages its status.
Container Network Interface (CNI) plugins	Exposes networking for Windows nodes.

Service	Description
Windows Instance Config Daemon (WICD)	Maintains the state of all services running on the Windows instance to ensure the instance functions as a worker node.
Windows Exporter	Exports Prometheus metrics from Windows nodes
Kubernetes Cloud Controller Manager (CCM)	Interacts with the underlying Azure cloud platform.
hybrid-overlay	Creates the OpenShift Container Platform Host Network Service (HNS).
kube-proxy	Maintains network rules on nodes allowing outside communication.
containerd container runtime	Manages the complete container lifecycle.
CSI Proxy	Enables CSI drivers to perform storage operations on the node, which allows containerized CSI drivers to run on Windows nodes.

CHAPTER 5. ENABLING WINDOWS CONTAINER WORKLOADS

Before adding Windows workloads to your cluster, you must install the Windows Machine Config Operator (WMCO), which is available in the OpenShift Container Platform OperatorHub. The WMCO orchestrates the process of deploying and managing Windows workloads on a cluster.

NOTE

Dual NIC is not supported on WMCO-managed Windows instances.

Prerequisites

- You have access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.
- You have installed the OpenShift CLI (**oc**).
- You have installed your cluster using installer-provisioned infrastructure, or using user-provisioned infrastructure with the **platform: none** field set in your **install-config.yaml** file.
- You have configured hybrid networking with OVN-Kubernetes for your cluster. For more information, see Configuring hybrid networking.
- You are running an OpenShift Container Platform cluster version 4.6.8 or later.

NOTE

Windows instances deployed by the WMCO are configured with the containerd container runtime. Because WMCO installs and manages the runtime, it is recommended that you do not manually install containerd on nodes.

Additional resources

- For the comprehensive prerequisites for the Windows Machine Config Operator, see Windows Machine Config Operator prerequisites.

5.1. INSTALLING THE WINDOWS MACHINE CONFIG OPERATOR

You can install the Windows Machine Config Operator using either the web console or OpenShift CLI (**oc**).

NOTE

Due to a limitation within the Windows operating system, **clusterNetwork** CIDR addresses of class E, such as **240.0.0.0**, are not compatible with Windows nodes.

5.1.1. Installing the Windows Machine Config Operator using the web console

You can use the OpenShift Container Platform web console to install the Windows Machine Config Operator (WMCO).

NOTE

Dual NIC is not supported on WMCO-managed Windows instances.

Procedure

1. From the **Administrator** perspective in the OpenShift Container Platform web console, navigate to the **Operators → OperatorHub** page.

2. Use the **Filter by keyword** box to search for **Windows Machine Config Operator** in the catalog. Click the **Windows Machine Config Operator** tile.

3. Review the information about the Operator and click **Install**.

4. On the **Install Operator** page:

   a. Select the **stable** channel as the **Update Channel**. The **stable** channel enables the latest stable release of the WMCO to be installed.

   b. The **Installation Mode** is preconfigured because the WMCO must be available in a single namespace only.

   c. Choose the **Installed Namespace** for the WMCO. The default Operator recommended namespace is **openshift-windows-machine-config-operator**.

   d. Click the **Enable Operator recommended cluster monitoring on the Namespace** checkbox to enable cluster monitoring for the WMCO.

   e. Select an **Approval Strategy**.

      - The **Automatic** strategy allows Operator Lifecycle Manager (OLM) to automatically update the Operator when a new version is available.

      - The **Manual** strategy requires a user with appropriate credentials to approve the Operator update.

1. Click **Install**. The WMCO is now listed on the **Installed Operators** page.

   NOTE

   The WMCO is installed automatically into the namespace you defined, like **openshift-windows-machine-config-operator**.

2. Verify that the **Status** shows **Succeeded** to confirm successful installation of the WMCO.

5.1.2. Installing the Windows Machine Config Operator using the CLI

You can use the OpenShift CLI (**oc**) to install the Windows Machine Config Operator (WMCO).

NOTE

Dual NIC is not supported on WMCO-managed Windows instances.

Procedure

1. Create a namespace for the WMCO.

   a. Create a **Namespace** object YAML file for the WMCO. For example, **wmco-namespace.yaml**:

   ```
   apiVersion: v1
   kind: Namespace
   metadata:
     name: openshift-windows-machine-config-operator (1)
     labels:
       openshift.io/cluster-monitoring: "true" (2)
   ```

   (1) It is recommended to deploy the WMCO in the **openshift-windows-machine-config-operator** namespace.

   (2) This label is required for enabling cluster monitoring for the WMCO.

   b. Create the namespace:

   ```
   $ oc create -f <file-name>.yaml
   ```

   For example:

   ```
   $ oc create -f wmco-namespace.yaml
   ```

2. Create the Operator group for the WMCO.

   a. Create an **OperatorGroup** object YAML file. For example, **wmco-og.yaml**:

   ```
   apiVersion: operators.coreos.com/v1
   kind: OperatorGroup
   metadata:
     name: windows-machine-config-operator
     namespace: openshift-windows-machine-config-operator
   spec:
     targetNamespaces:
     - openshift-windows-machine-config-operator
   ```

   b. Create the Operator group:

   ```
   $ oc create -f <file-name>.yaml
   ```

   For example:

   ```
   $ oc create -f wmco-og.yaml
   ```

3. Subscribe the namespace to the WMCO.

   a. Create a **Subscription** object YAML file. For example, **wmco-sub.yaml**:

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
   ```

```
  name: windows-machine-config-operator
  namespace: openshift-windows-machine-config-operator
spec:
  channel: "stable" 1
  installPlanApproval: "Automatic" 2
  name: "windows-machine-config-operator"
  source: "redhat-operators" 3
  sourceNamespace: "openshift-marketplace" 4
```

1 Specify **stable** as the channel.

2 Set an approval strategy. You can set **Automatic** or **Manual**.

3 Specify the **redhat-operators** catalog source, which contains the **windows-machine-config-operator** package manifests. If your OpenShift Container Platform is installed on a restricted network, also known as a disconnected cluster, specify the name of the **CatalogSource** object you created when you configured the Operator LifeCycle Manager (OLM).

4 Namespace of the catalog source. Use **openshift-marketplace** for the default OperatorHub catalog sources.

b. Create the subscription:

```
$ oc create -f <file-name>.yaml
```

For example:

```
$ oc create -f wmco-sub.yaml
```

The WMCO is now installed to the **openshift-windows-machine-config-operator**.

4. Verify the WMCO installation:

```
$ oc get csv -n openshift-windows-machine-config-operator
```

Example output

```
NAME                                  DISPLAY                           VERSION   REPLACES   PHASE
windows-machine-config-operator.2.0.0   Windows Machine Config Operator   2.0.0
Succeeded
```

5.2. CONFIGURING A SECRET FOR THE WINDOWS MACHINE CONFIG OPERATOR

To run the Windows Machine Config Operator (WMCO), you must create a secret in the WMCO namespace containing a private key. This is required to allow the WMCO to communicate with the Windows virtual machine (VM).

Prerequisites

- You installed the Windows Machine Config Operator (WMCO) using Operator Lifecycle Manager (OLM).
- You created a PEM-encoded file containing an RSA key.

Procedure

- Define the secret required to access the Windows VMs:

```
$ oc create secret generic cloud-private-key --from-file=private-
key.pem=${HOME}/.ssh/<key> \
    -n openshift-windows-machine-config-operator 1
```

1 You must create the private key in the WMCO namespace, like **openshift-windows-machine-config-operator**.

It is recommended to use a different private key than the one used when installing the cluster.

5.3. USING WINDOWS CONTAINERS IN A PROXY-ENABLED CLUSTER

The Windows Machine Config Operator (WMCO) can consume and use a cluster-wide egress proxy configuration when making external requests outside the cluster's internal network.

This allows you to add Windows nodes and run workloads in a proxy-enabled cluster, allowing your Windows nodes to pull images from registries that are secured behind your proxy server or to make requests to off-cluster services and services that use a custom public key infrastructure.

NOTE

The cluster-wide proxy affects system components only, not user workloads.

In proxy-enabled clusters, the WMCO is aware of the **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** values that are set for the cluster. The WMCO periodically checks whether the proxy environment variables have changed. If there is a discrepancy, the WMCO reconciles and updates the proxy environment variables on the Windows instances.

Windows workloads created on Windows nodes in proxy-enabled clusters do not inherit proxy settings from the node by default, the same as with Linux nodes. Also, by default PowerShell sessions do not inherit proxy settings on Windows nodes in proxy-enabled clusters.

Additional resources

- Configuring the cluster-wide proxy.

5.4. USING WINDOWS CONTAINERS WITH A MIRROR REGISTRY

The Windows Machine Config Operator (WMCO) can pull images from a registry mirror rather than from a public registry by using an **ImageDigestMirrorSet** (IDMS) or **ImageTagMirrorSet** (ITMS) object to configure your cluster to pull images from the mirror registry.

A mirror registry has the following benefits:

- Avoids public registry outages

- Speeds up node and pod creation
- Pulls images from behind your organization's firewall

A mirror registry can also be used with a OpenShift Container Platform cluster in a disconnected, or air-gapped, network. A *disconnected network* is a restricted network without direct internet connectivity. Because the cluster does not have access to the internet, any external container images cannot be referenced.

Using a mirror registry requires the following general steps:

- Create the mirror registry, using a tool such as Red Hat Quay.
- Create a container image registry credentials file.
- Copy the images from your online image repository to your mirror registry.

For information about these steps, see "About disconnected installation mirroring."

After creating the mirror registry and mirroring the images, you can use an **ImageDigestMirrorSet** (IDMS) or **ImageTagMirrorSet** (ITMS) object to configure your cluster to pull images from the mirror registry without needing to update each of your pod specs. The IDMS and ITMS objects redirect requests to pull images from a repository on a source image registry and have it resolved by the mirror repository instead.

If changes are made to the IDMS or ITMS object, the WMCO automatically updates the appropriate **hosts.toml** file on your Windows nodes with the new information. Note that the WMCO sequentially updates each Windows node when mirror settings are changed. As such, the time required for these updates increases with the number of Windows nodes in the cluster.

Also, because Windows nodes configured by the WMCO rely on containerd container runtime, the WMCO ensures that the containerd config files are up-to-date with the registry settings. For new nodes, these files are copied to the instances upon creation. For existing nodes, after activating the mirror registry, the registry controller uses SSH to access each node and copy the generated config files, replacing any existing files.

You can use a mirror registry with machine set or Bring-Your-Own-Host (BYOH) Windows nodes.

Additional references

- About disconnected installation mirroring

5.4.1. Understanding image registry repository mirroring

Setting up container registry repository mirroring enables you to perform the following tasks:

- Configure your OpenShift Container Platform cluster to redirect requests to pull images from a repository on a source image registry and have it resolved by a repository on a mirrored image registry.
- Identify multiple mirrored repositories for each target repository, to make sure that if one mirror is down, another can be used.

Repository mirroring in OpenShift Container Platform includes the following attributes:

- Image pulls are resilient to registry downtimes.

- Clusters in disconnected environments can pull images from critical locations, such as quay.io, and have registries behind a company firewall provide the requested images.
- A particular order of registries is tried when an image pull request is made, with the permanent registry typically being the last one tried.
- The mirror information you enter is added to the appropriate **hosts.toml** containerd configuration file(s) on every Windows node in the OpenShift Container Platform cluster.
- When a node makes a request for an image from the source repository, it tries each mirrored repository in turn until it finds the requested content. If all mirrors fail, the cluster tries the source repository. If successful, the image is pulled to the node.

Setting up repository mirroring can be done in the following ways:

- At OpenShift Container Platform installation:
  By pulling container images needed by OpenShift Container Platform and then bringing those images behind your company’s firewall, you can install OpenShift Container Platform into a data center that is in a disconnected environment.
- After OpenShift Container Platform installation:
  If you did not configure mirroring during OpenShift Container Platform installation, you can do so postinstallation by using any of the following custom resource (CR) objects:
  - **ImageDigestMirrorSet** (IDMS). This object allows you to pull images from a mirrored registry by using digest specifications. The IDMS CR enables you to set a fall back policy that allows or stops continued attempts to pull from the source registry if the image pull fails.
  - **ImageTagMirrorSet** (ITMS). This object allows you to pull images from a mirrored registry by using image tags. The ITMS CR enables you to set a fall back policy that allows or stops continued attempts to pull from the source registry if the image pull fails.

Each of these custom resource objects identify the following information:

- The source of the container image repository you want to mirror.
- A separate entry for each mirror repository you want to offer the content requested from the source repository.

The Windows Machine Config Operator (WMCO) watches for changes to the IDMS and ITMS resources and generates a set of **hosts.toml** containerd configuration files, one file for each source registry, with those changes. The WMCO then updates any existing Windows nodes to use the new registry configuration.

NOTE

The IDMS and ITMS objects must be created before you can add Windows nodes using a mirrored registry.

5.4.2. Configuring image registry repository mirroring

You can create postinstallation mirror configuration custom resources (CR) to redirect image pull requests from a source image registry to a mirrored image registry.

IMPORTANT

Windows images mirrored through **ImageDigestMirrorSet** and **ImageTagMirrorSet** objects have specific naming requirements. The final portion of the namespace and the image name of the mirror image must match the image being mirrored. For example, when mirroring the **mcr.microsoft.com/oss/kubernetes/pause:3.9** image, the mirror image must have the **<mirror_registry>/<optional_namespaces>/oss/kubernetes/pause:3.9** format. The **optional_namespaces** can be any number of leading repository namespaces.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Configure mirrored repositories, by either:

   - Setting up a mirrored repository with Red Hat Quay, as described in Red Hat Quay Repository Mirroring. Using Red Hat Quay allows you to copy images from one repository to another and also automatically sync those repositories repeatedly over time.

   - Using a tool such as **skopeo** to copy images manually from the source repository to the mirrored repository.
     For example, after installing the skopeo RPM package on a Red Hat Enterprise Linux (RHEL) 7 or RHEL 8 system, use the **skopeo** command as shown in this example:

     ```
     $ skopeo copy --all \
     docker://registry.access.redhat.com/ubi9/ubi-minimal:latest@sha256:5cf... \
     docker://example.io/example/ubi-minimal
     ```

     In this example, you have a container image registry that is named **example.io** with an image repository named **example** to which you want to copy the **ubi9/ubi-minimal** image from **registry.access.redhat.com**. After you create the mirrored registry, you can configure your OpenShift Container Platform cluster to redirect requests made of the source repository to the mirrored repository.

   IMPORTANT

   You must mirror the **mcr.microsoft.com/oss/kubernetes/pause:3.9** image. For example, you could use the following **skopeo** command to mirror the image:

   ```
   $ skopeo copy \
   docker://mcr.microsoft.com/oss/kubernetes/pause:3.9\
   docker://example.io/oss/kubernetes/pause:3.9
   ```

2. Log in to your OpenShift Container Platform cluster.

3. Create an **ImageDigestMirrorSet** or **ImageTagMirrorSet** CR, as needed, replacing the source and mirrors with your own registry and repository pairs and images:

   ```
   apiVersion: config.openshift.io/v1 (1)
   kind: ImageDigestMirrorSet (2)
   metadata:
   ```

```
  name: ubi9repo
spec:
  imageDigestMirrors: (3)
  - mirrors:
    - example.io/example/ubi-minimal (4)
    - example.com/example2/ubi-minimal (5)
    source: registry.access.redhat.com/ubi9/ubi-minimal (6)
    mirrorSourcePolicy: AllowContactingSource (7)
  - mirrors:
    - mirror.example.com
    source: registry.redhat.io
    mirrorSourcePolicy: NeverContactSource
  - mirrors:
    - docker.io
    source: docker-mirror.internal
    mirrorSourcePolicy: AllowContactingSource
```

1. Indicates the API to use with this CR. This must be **config.openshift.io/v1**.
2. Indicates the kind of object according to the pull type:
   - **ImageDigestMirrorSet**: Pulls a digest reference image.
   - **ImageTagMirrorSet**: Pulls a tag reference image.
3. Indicates the type of image pull method, either:
   - **imageDigestMirrors**: Use for an **ImageDigestMirrorSet** CR.
   - **imageTagMirrors**: Use for an **ImageTagMirrorSet** CR.
4. Indicates the name of the mirrored image registry and repository.
5. Optional: Indicates a secondary mirror repository for each target repository. If one mirror is down, the target repository can use another mirror.
6. Indicates the registry and repository source, which is the repository that is referred to in image pull specifications.
7. Optional: Indicates the fallback policy if the image pull fails:
   - **AllowContactingSource**: Allows continued attempts to pull the image from the source repository. This is the default.
   - **NeverContactSource**: Prevents continued attempts to pull the image from the source repository.

4. Create the new object:

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

5. To check that the mirrored configuration settings are applied, do the following on one of the nodes.

   a. List your nodes:

```
$ oc get node
```

Example output

```
NAME                        STATUS             ROLES    AGE  VERSION
ip-10-0-137-44.ec2.internal    Ready                  worker   7m   v1.29.4
ip-10-0-138-148.ec2.internal   Ready                  master   11m  v1.29.4
ip-10-0-139-122.ec2.internal   Ready                  master   11m  v1.29.4
ip-10-0-147-35.ec2.internal    Ready                  worker   7m   v1.29.4
ip-10-0-153-12.ec2.internal    Ready                  worker   7m   v1.29.4
ip-10-0-154-10.ec2.internal    Ready                  master   11m  v1.29.4
```

b. Start the debugging process to access the node:

```
$ oc debug node/ip-10-0-147-35.ec2.internal
```

Example output

```
Starting pod/ip-10-0-147-35ec2internal-debug ...
To use host binaries, run `chroot /host`
```

c. Change your root directory to **/host**:

```
sh-4.2# chroot /host
```

d. Check that the WMCO generated a **hosts.toml** file for each registry on each Windows instance. For the previous example IDMS object, there should be three files in the following file structure:

```
$ tree $config_path
```

Example output

```
C:/k/containerd/registries/
|── registry.access.redhat.com
|   └── hosts.toml
|── mirror.example.com
|   └── hosts.toml
└── docker.io
    └── hosts.toml:
```

The following output represents a **hosts.toml** containerd configuration file where the previous example IDMS object was applied.

Example host.toml files

```
$ cat "$config_path"/registry.access.redhat.com/host.toml
server = "https://registry.access.redhat.com" # default fallback server since
"AllowContactingSource" mirrorSourcePolicy is set

[host."https://example.io/example/ubi-minimal"]
 capabilities = ["pull"]
```

```
[host."https://example.com/example2/ubi-minimal"] # secondary mirror
 capabilities = ["pull"]


$ cat "$config_path"/registry.redhat.io/host.toml
# "server" omitted since "NeverContactSource" mirrorSourcePolicy is set

[host."https://mirror.example.com"]
 capabilities = ["pull"]


$ cat "$config_path"/docker.io/host.toml
server = "https://docker.io"

[host."https://docker-mirror.internal"]
 capabilities = ["pull", "resolve"] # resolve tags
```

e. Pull an image to the node from the source and check if it is resolved by the mirror.

```
sh-4.2# podman pull --log-level=debug registry.access.redhat.com/ubi9/ubi-
minimal@sha256:5cf...
```

Troubleshooting repository mirroring

If the repository mirroring procedure does not work as described, use the following information about how repository mirroring works to help troubleshoot the problem.

- The first working mirror is used to supply the pulled image.
- The main registry is only used if no other mirror works.
- From the system context, the **Insecure** flags are used as fallback.

5.5. ADDITIONAL RESOURCES

- Generating a key pair for cluster node SSH access
- Adding Operators to a cluster .

CHAPTER 6. CREATING WINDOWS MACHINE SETS

6.1. CREATING A WINDOWS MACHINE SET ON AWS

You can create a Windows **MachineSet** object to serve a specific purpose in your OpenShift Container Platform cluster on Amazon Web Services (AWS). For example, you might create infrastructure Windows machine sets and related machines so that you can move supporting Windows workloads to the new Windows machines.

Prerequisites

- You installed the Windows Machine Config Operator (WMCO) using Operator Lifecycle Manager (OLM).
- You are using a supported Windows Server as the operating system image.
  Use one of the following **aws** commands, as appropriate for your Windows Server release, to query valid AMI images:

  #### Example Windows Server 2022 command

  ```
  $ aws ec2 describe-images --region <aws_region_name> --filters
  "Name=name,Values=Windows_Server-2022*English*Core*Base*" "Name=is-
  public,Values=true" --query "reverse(sort_by(Images, &CreationDate))[*].{name: Name, id:
  ImageId}" --output table
  ```

  #### Example Windows Server 2019 command

  ```
  $ aws ec2 describe-images --region <aws_region_name> --filters
  "Name=name,Values=Windows_Server-2019*English*Core*Base*" "Name=is-
  public,Values=true" --query "reverse(sort_by(Images, &CreationDate))[*].{name: Name, id:
  ImageId}" --output table
  ```

  where:

  <aws_region_name>
  : Specifies the name of your AWS region.

6.1.1. Machine API overview

The Machine API is a combination of primary resources that are based on the upstream Cluster API project and custom OpenShift Container Platform resources.

For OpenShift Container Platform 4.16 clusters, the Machine API performs all node host provisioning management actions after the cluster installation finishes. Because of this system, OpenShift Container Platform 4.16 offers an elastic, dynamic provisioning method on top of public or private cloud infrastructure.

The two primary resources are:

Machines
: A fundamental unit that describes the host for a node. A machine has a **providerSpec** specification, which describes the types of compute nodes that are offered for different cloud platforms. For example, a machine type for a compute node might define a specific machine type and required

metadata.

Machine sets

MachineSet resources are groups of compute machines. Compute machine sets are to compute machines as replica sets are to pods. If you need more compute machines or must scale them down, you change the **replicas** field on the **MachineSet** resource to meet your compute need.

> **WARNING**
>
> Control plane machines cannot be managed by compute machine sets.
>
> Control plane machine sets provide management capabilities for supported control plane machines that are similar to what compute machine sets provide for compute machines.
>
> For more information, see “Managing control plane machines".

The following custom resources add more capabilities to your cluster:

Machine autoscaler

The **MachineAutoscaler** resource automatically scales compute machines in a cloud. You can set the minimum and maximum scaling boundaries for nodes in a specified compute machine set, and the machine autoscaler maintains that range of nodes.
The **MachineAutoscaler** object takes effect after a **ClusterAutoscaler** object exists. Both **ClusterAutoscaler** and **MachineAutoscaler** resources are made available by the **ClusterAutoscalerOperator** object.

Cluster autoscaler

This resource is based on the upstream cluster autoscaler project. In the OpenShift Container Platform implementation, it is integrated with the Machine API by extending the compute machine set API. You can use the cluster autoscaler to manage your cluster in the following ways:

- Set cluster-wide scaling limits for resources such as cores, nodes, memory, and GPU
- Set the priority so that the cluster prioritizes pods and new nodes are not brought online for less important pods
- Set the scaling policy so that you can scale up nodes but not scale them down

Machine health check

The **MachineHealthCheck** resource detects when a machine is unhealthy, deletes it, and, on supported platforms, makes a new machine.

In OpenShift Container Platform version 3.11, you could not roll out a multi-zone architecture easily because the cluster did not manage machine provisioning. Beginning with OpenShift Container Platform version 4.1, this process is easier. Each compute machine set is scoped to a single zone, so the installation program sends out compute machine sets across availability zones on your behalf. And then because your compute is dynamic, and in the face of a zone failure, you always have a zone for when you must rebalance your machines. In global Azure regions that do not have multiple availability zones, you can use availability sets to ensure high availability. The autoscaler provides best-effort balancing over the life of a cluster.

6.1.2. Sample YAML for a Windows MachineSet object on AWS

This sample YAML defines a Windows **MachineSet** object running on Amazon Web Services (AWS) that the Windows Machine Config Operator (WMCO) can react upon.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> (1)
  name: <infrastructure_id>-windows-worker-<zone> (2)
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id> (3)
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-windows-worker-<zone> (4)
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id> (5)
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-windows-worker-<zone> (6)
        machine.openshift.io/os-id: Windows (7)
    spec:
      metadata:
        labels:
          node-role.kubernetes.io/worker: "" (8)
      providerSpec:
        value:
          ami:
            id: <windows_container_ami> (9)
          apiVersion: awsproviderconfig.openshift.io/v1beta1
          blockDevices:
            - ebs:
                iops: 0
                volumeSize: 120
                volumeType: gp2
          credentialsSecret:
            name: aws-cloud-credentials
          deviceIndex: 0
          iamInstanceProfile:
            id: <infrastructure_id>-worker-profile (10)
          instanceType: m5a.large
          kind: AWSMachineProviderConfig
          placement:
            availabilityZone: <zone> (11)
            region: <region> (12)
          securityGroups:
            - filters:
                - name: tag:Name
                  values:
                    - <infrastructure_id>-worker-sg 13
          subnet:
            filters:
              - name: tag:Name
                values:
                  - <infrastructure_id>-private-<zone> 14
          tags:
            - name: kubernetes.io/cluster/<infrastructure_id> 15
              value: owned
          userDataSecret:
            name: windows-user-data 16
            namespace: openshift-machine-api
```

1 3 5 10 13 14 15 Specify the infrastructure ID that is based on the cluster ID that you set when you provisioned the cluster. You can obtain the infrastructure ID by running the following command:

```
$ oc get -o jsonpath='{.status.infrastructureName}{"\n"}' infrastructure cluster
```

2 4 6 Specify the infrastructure ID, worker label, and zone.

7 Configure the compute machine set as a Windows machine.

8 Configure the Windows node as a compute machine.

9 Specify the AMI ID of a supported Windows image with a container runtime installed.

11 Specify the AWS zone, like **us-east-1a**.

12 Specify the AWS region, like **us-east-1**.

16 Created by the WMCO when it is configuring the first Windows machine. After that, the **windows-user-data** is available for all subsequent compute machine sets to consume.

6.1.3. Creating a compute machine set

In addition to the compute machine sets created by the installation program, you can create your own to dynamically manage the machine compute resources for specific workloads of your choice.

Prerequisites

- Deploy an OpenShift Container Platform cluster.
- Install the OpenShift CLI (**oc**).
- Log in to **oc** as a user with **cluster-admin** permission.
- In disconnected environments, the image specified in the **MachineSet** custom resource (CR) must have the OpenSSH server v0.0.1.0 installed.

Procedure

1. Create a new YAML file that contains the compute machine set custom resource (CR) sample and is named **<file_name>.yaml**.
   Ensure that you set the **<clusterID>** and **<role>** parameter values.

2. Optional: If you are not sure which value to set for a specific field, you can check an existing compute machine set from your cluster.

   a. To list the compute machine sets in your cluster, run the following command:

   ```
   $ oc get machinesets -n openshift-machine-api
   ```

   #### Example output

   ```
   NAME                                DESIRED   CURRENT   READY   AVAILABLE   AGE
   agl030519-vplxk-worker-us-east-1a   1         1         1       1           55m
   agl030519-vplxk-worker-us-east-1b   1         1         1       1           55m
   agl030519-vplxk-worker-us-east-1c   1         1         1       1           55m
   agl030519-vplxk-worker-us-east-1d   0         0                             55m
   agl030519-vplxk-worker-us-east-1e   0         0                             55m
   agl030519-vplxk-worker-us-east-1f   0         0                             55m
   ```

   b. To view values of a specific compute machine set custom resource (CR), run the following command:

   ```
   $ oc get machineset <machineset_name> \
     -n openshift-machine-api -o yaml
   ```

   #### Example output

   ```
   apiVersion: machine.openshift.io/v1beta1
   kind: MachineSet
   metadata:
     labels:
       machine.openshift.io/cluster-api-cluster: <infrastructure_id> (1)
     name: <infrastructure_id>-<role> (2)
     namespace: openshift-machine-api
   spec:
     replicas: 1
     selector:
       matchLabels:
         machine.openshift.io/cluster-api-cluster: <infrastructure_id>
         machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
     template:
       metadata:
         labels:
           machine.openshift.io/cluster-api-cluster: <infrastructure_id>
           machine.openshift.io/cluster-api-machine-role: <role>
           machine.openshift.io/cluster-api-machine-type: <role>
           machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
       spec:
         providerSpec: (3)
           ...
   ```

1 The cluster infrastructure ID.

2 A default node label.

> **NOTE**
>
> For clusters that have user-provisioned infrastructure, a compute machine set can only create **worker** and **infra** type machines.

3 The values in the **<providerSpec>** section of the compute machine set CR are platform-specific. For more information about **<providerSpec>** parameters in the CR, see the sample compute machine set CR configuration for your provider.

3. Create a **MachineSet** CR by running the following command:

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

Verification

- View the list of compute machine sets by running the following command:

```
$ oc get machineset -n openshift-machine-api
```

Example output

```
NAME                                     DESIRED  CURRENT  READY  AVAILABLE  AGE
agl030519-vplxk-windows-worker-us-east-1a  1        1        1      1          11m
agl030519-vplxk-worker-us-east-1a          1        1        1      1          55m
agl030519-vplxk-worker-us-east-1b          1        1        1      1          55m
agl030519-vplxk-worker-us-east-1c          1        1        1      1          55m
agl030519-vplxk-worker-us-east-1d          0        0                          55m
agl030519-vplxk-worker-us-east-1e          0        0                          55m
agl030519-vplxk-worker-us-east-1f          0        0                          55m
```

When the new compute machine set is available, the **DESIRED** and **CURRENT** values match. If the compute machine set is not available, wait a few minutes and run the command again.

6.1.4. Additional resources

- Overview of machine management

6.2. CREATING A WINDOWS MACHINE SET ON AZURE

You can create a Windows **MachineSet** object to serve a specific purpose in your OpenShift Container Platform cluster on Microsoft Azure. For example, you might create infrastructure Windows machine sets and related machines so that you can move supporting Windows workloads to the new Windows machines.

Prerequisites

- You installed the Windows Machine Config Operator (WMCO) using Operator Lifecycle Manager (OLM).

- You are using a supported Windows Server as the operating system image.

6.2.1. Machine API overview

The Machine API is a combination of primary resources that are based on the upstream Cluster API project and custom OpenShift Container Platform resources.

For OpenShift Container Platform 4.16 clusters, the Machine API performs all node host provisioning management actions after the cluster installation finishes. Because of this system, OpenShift Container Platform 4.16 offers an elastic, dynamic provisioning method on top of public or private cloud infrastructure.

The two primary resources are:

Machines

A fundamental unit that describes the host for a node. A machine has a **providerSpec** specification, which describes the types of compute nodes that are offered for different cloud platforms. For example, a machine type for a compute node might define a specific machine type and required metadata.

Machine sets

MachineSet resources are groups of compute machines. Compute machine sets are to compute machines as replica sets are to pods. If you need more compute machines or must scale them down, you change the **replicas** field on the **MachineSet** resource to meet your compute need.

> **WARNING**
>
> Control plane machines cannot be managed by compute machine sets.
>
> Control plane machine sets provide management capabilities for supported control plane machines that are similar to what compute machine sets provide for compute machines.
>
> For more information, see “Managing control plane machines".

The following custom resources add more capabilities to your cluster:

Machine autoscaler

The **MachineAutoscaler** resource automatically scales compute machines in a cloud. You can set the minimum and maximum scaling boundaries for nodes in a specified compute machine set, and the machine autoscaler maintains that range of nodes.
The **MachineAutoscaler** object takes effect after a **ClusterAutoscaler** object exists. Both **ClusterAutoscaler** and **MachineAutoscaler** resources are made available by the **ClusterAutoscalerOperator** object.

Cluster autoscaler

This resource is based on the upstream cluster autoscaler project. In the OpenShift Container Platform implementation, it is integrated with the Machine API by extending the compute machine set API. You can use the cluster autoscaler to manage your cluster in the following ways:

- Set cluster-wide scaling limits for resources such as cores, nodes, memory, and GPU
- Set the priority so that the cluster prioritizes pods and new nodes are not brought online for less important pods
- Set the scaling policy so that you can scale up nodes but not scale them down

Machine health check

The **MachineHealthCheck** resource detects when a machine is unhealthy, deletes it, and, on supported platforms, makes a new machine.

In OpenShift Container Platform version 3.11, you could not roll out a multi-zone architecture easily because the cluster did not manage machine provisioning. Beginning with OpenShift Container Platform version 4.1, this process is easier. Each compute machine set is scoped to a single zone, so the installation program sends out compute machine sets across availability zones on your behalf. And then because your compute is dynamic, and in the face of a zone failure, you always have a zone for when you must rebalance your machines. In global Azure regions that do not have multiple availability zones, you can use availability sets to ensure high availability. The autoscaler provides best-effort balancing over the life of a cluster.

6.2.2. Sample YAML for a Windows MachineSet object on Azure

This sample YAML defines a Windows **MachineSet** object running on Microsoft Azure that the Windows Machine Config Operator (WMCO) can react upon.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <windows_machine_set_name> 2
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id> 3
      machine.openshift.io/cluster-api-machineset: <windows_machine_set_name> 4
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id> 5
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: <windows_machine_set_name> 6
        machine.openshift.io/os-id: Windows 7
    spec:
      metadata:
        labels:
          node-role.kubernetes.io/worker: "" 8
      providerSpec:
        value:
          apiVersion: azureproviderconfig.openshift.io/v1beta1
          credentialsSecret:
            name: azure-cloud-credentials
            namespace: openshift-machine-api
          image: 9
            offer: WindowsServer
            publisher: MicrosoftWindowsServer
            resourceID: ""
            sku: 2019-Datacenter-with-Containers
            version: latest
          kind: AzureMachineProviderSpec
          location: <location> 10
          managedIdentity: <infrastructure_id>-identity 11
          networkResourceGroup: <infrastructure_id>-rg 12
          osDisk:
            diskSizeGB: 128
            managedDisk:
              storageAccountType: Premium_LRS
            osType: Windows
          publicIP: false
          resourceGroup: <infrastructure_id>-rg 13
          subnet: <infrastructure_id>-worker-subnet
          userDataSecret:
            name: windows-user-data 14
            namespace: openshift-machine-api
          vmSize: Standard_D2s_v3
          vnet: <infrastructure_id>-vnet 15
          zone: "<zone>" 16
```

1 3 5 11 12 13 15 Specify the infrastructure ID that is based on the cluster ID that you set when you provisioned the cluster. You can obtain the infrastructure ID by running the following command:

```
$ oc get -o jsonpath='{.status.infrastructureName}{"\n"}' infrastructure cluster
```

2 4 6 Specify the Windows compute machine set name. Windows machine names on Azure cannot be more than 15 characters long. Therefore, the compute machine set name cannot be more than 9 characters long, due to the way machine names are generated from it.

7 Configure the compute machine set as a Windows machine.

8 Configure the Windows node as a compute machine.

9 Specify a **WindowsServer** image offering that defines the **2019-Datacenter-with-Containers** SKU.

10 Specify the Azure region, like **centralus**.

14 Created by the WMCO when it is configuring the first Windows machine. After that, the **windows-user-data** is available for all subsequent compute machine sets to consume.

16 Specify the zone within your region to place machines on. Be sure that your region supports the zone that you specify.

6.2.3. Creating a compute machine set

In addition to the compute machine sets created by the installation program, you can create your own to dynamically manage the machine compute resources for specific workloads of your choice.

Prerequisites

- Deploy an OpenShift Container Platform cluster.
- Install the OpenShift CLI (**oc**).
- Log in to **oc** as a user with **cluster-admin** permission.
- In disconnected environments, the image specified in the **MachineSet** custom resource (CR) must have the OpenSSH server v0.0.1.0 installed.

Procedure

1. Create a new YAML file that contains the compute machine set custom resource (CR) sample and is named **<file_name>.yaml**.
   Ensure that you set the **<clusterID>** and **<role>** parameter values.

2. Optional: If you are not sure which value to set for a specific field, you can check an existing compute machine set from your cluster.

   a. To list the compute machine sets in your cluster, run the following command:

   ```
   $ oc get machinesets -n openshift-machine-api
   ```

   Example output

   ```
   NAME                                DESIRED   CURRENT   READY   AVAILABLE   AGE
   agl030519-vplxk-worker-us-east-1a   1         1         1       1           55m
   agl030519-vplxk-worker-us-east-1b   1         1         1       1           55m
   agl030519-vplxk-worker-us-east-1c   1         1         1       1           55m
   agl030519-vplxk-worker-us-east-1d   0         0                             55m
   agl030519-vplxk-worker-us-east-1e   0         0                             55m
   agl030519-vplxk-worker-us-east-1f   0         0                             55m
   ```

   b. To view values of a specific compute machine set custom resource (CR), run the following command:

   ```
   $ oc get machineset <machineset_name> \
     -n openshift-machine-api -o yaml
   ```

   Example output

   ```
   apiVersion: machine.openshift.io/v1beta1
   kind: MachineSet
   metadata:
     labels:
       machine.openshift.io/cluster-api-cluster: <infrastructure_id> (1)
     name: <infrastructure_id>-<role> (2)
     namespace: openshift-machine-api
   spec:
     replicas: 1
   ```

```
selector:
  matchLabels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id>
    machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
template:
  metadata:
    labels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machine-role: <role>
      machine.openshift.io/cluster-api-machine-type: <role>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
  spec:
    providerSpec: 3
      ...
```

1 The cluster infrastructure ID.

2 A default node label.

NOTE

For clusters that have user-provisioned infrastructure, a compute machine set can only create **worker** and **infra** type machines.

3 The values in the **<providerSpec>** section of the compute machine set CR are platform-specific. For more information about **<providerSpec>** parameters in the CR, see the sample compute machine set CR configuration for your provider.

3. Create a **MachineSet** CR by running the following command:

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

Verification

- View the list of compute machine sets by running the following command:

```
$ oc get machineset -n openshift-machine-api
```

Example output

```
NAME                                    DESIRED   CURRENT   READY   AVAILABLE   AGE
agl030519-vplxk-windows-worker-us-east-1a  1         1         1       1           11m
agl030519-vplxk-worker-us-east-1a          1         1         1       1           55m
agl030519-vplxk-worker-us-east-1b          1         1         1       1           55m
agl030519-vplxk-worker-us-east-1c          1         1         1       1           55m
agl030519-vplxk-worker-us-east-1d          0         0                             55m
agl030519-vplxk-worker-us-east-1e          0         0                             55m
agl030519-vplxk-worker-us-east-1f          0         0                             55m
```

When the new compute machine set is available, the **DESIRED** and **CURRENT** values match. If the compute machine set is not available, wait a few minutes and run the command again.

6.2.4. Additional resources

- Overview of machine management

6.3. CREATING A WINDOWS MACHINE SET ON GCP

You can create a Windows **MachineSet** object to serve a specific purpose in your OpenShift Container Platform cluster on Google Cloud Platform (GCP). For example, you might create infrastructure Windows machine sets and related machines so that you can move supporting Windows workloads to the new Windows machines.

Prerequisites

- You installed the Windows Machine Config Operator (WMCO) using Operator Lifecycle Manager (OLM).
- You are using a supported Windows Server as the operating system image.

6.3.1. Machine API overview

The Machine API is a combination of primary resources that are based on the upstream Cluster API project and custom OpenShift Container Platform resources.

For OpenShift Container Platform 4.16 clusters, the Machine API performs all node host provisioning management actions after the cluster installation finishes. Because of this system, OpenShift Container Platform 4.16 offers an elastic, dynamic provisioning method on top of public or private cloud infrastructure.

The two primary resources are:

Machines

A fundamental unit that describes the host for a node. A machine has a **providerSpec** specification, which describes the types of compute nodes that are offered for different cloud platforms. For example, a machine type for a compute node might define a specific machine type and required metadata.

Machine sets

MachineSet resources are groups of compute machines. Compute machine sets are to compute machines as replica sets are to pods. If you need more compute machines or must scale them down, you change the **replicas** field on the **MachineSet** resource to meet your compute need.

> **WARNING**
>
> Control plane machines cannot be managed by compute machine sets.
>
> Control plane machine sets provide management capabilities for supported control plane machines that are similar to what compute machine sets provide for compute machines.
>
> For more information, see “Managing control plane machines".

The following custom resources add more capabilities to your cluster:

Machine autoscaler

The **MachineAutoscaler** resource automatically scales compute machines in a cloud. You can set the minimum and maximum scaling boundaries for nodes in a specified compute machine set, and the machine autoscaler maintains that range of nodes.
The **MachineAutoscaler** object takes effect after a **ClusterAutoscaler** object exists. Both **ClusterAutoscaler** and **MachineAutoscaler** resources are made available by the **ClusterAutoscalerOperator** object.

Cluster autoscaler

This resource is based on the upstream cluster autoscaler project. In the OpenShift Container Platform implementation, it is integrated with the Machine API by extending the compute machine set API. You can use the cluster autoscaler to manage your cluster in the following ways:

- Set cluster-wide scaling limits for resources such as cores, nodes, memory, and GPU
- Set the priority so that the cluster prioritizes pods and new nodes are not brought online for less important pods
- Set the scaling policy so that you can scale up nodes but not scale them down

Machine health check

The **MachineHealthCheck** resource detects when a machine is unhealthy, deletes it, and, on supported platforms, makes a new machine.

In OpenShift Container Platform version 3.11, you could not roll out a multi-zone architecture easily because the cluster did not manage machine provisioning. Beginning with OpenShift Container Platform version 4.1, this process is easier. Each compute machine set is scoped to a single zone, so the installation program sends out compute machine sets across availability zones on your behalf. And then because your compute is dynamic, and in the face of a zone failure, you always have a zone for when you must rebalance your machines. In global Azure regions that do not have multiple availability zones, you can use availability sets to ensure high availability. The autoscaler provides best-effort balancing over the life of a cluster.

6.3.2. Sample YAML for a Windows MachineSet object on GCP

This sample YAML file defines a Windows **MachineSet** object running on Google Cloud Platform (GCP) that the Windows Machine Config Operator (WMCO) can use.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-windows-worker-<zone_suffix> 2
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id> 3
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-windows-worker-<zone_suffix>
4
```

```
template:
  metadata:
    labels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id> 5
      machine.openshift.io/cluster-api-machine-role: worker
      machine.openshift.io/cluster-api-machine-type: worker
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-windows-worker-<zone_suffix>
6
      machine.openshift.io/os-id: Windows 7
  spec:
    metadata:
      labels:
        node-role.kubernetes.io/worker: "" 8
    providerSpec:
      value:
        apiVersion: machine.openshift.io/v1beta1
        canIPForward: false
        credentialsSecret:
          name: gcp-cloud-credentials
        deletionProtection: false
        disks:
        - autoDelete: true
          boot: true
          image: <windows_server_image> 9
          sizeGb: 128
          type: pd-ssd
        kind: GCPMachineProviderSpec
        machineType: n1-standard-4
        networkInterfaces:
        - network: <infrastructure_id>-network 10
          subnetwork: <infrastructure_id>-worker-subnet
        projectID: <project_id> 11
        region: <region> 12
        serviceAccounts:
        - email: <infrastructure_id>-w@<project_id>.iam.gserviceaccount.com
          scopes:
          - https://www.googleapis.com/auth/cloud-platform
        tags:
        - <infrastructure_id>-worker
        userDataSecret:
          name: windows-user-data 13
        zone: <zone> 14
```

1 3 5 10 Specify the infrastructure ID that is based on the cluster ID that you set when you provisioned the cluster. You can obtain the infrastructure ID by running the following command:

```
$ oc get -o jsonpath='{.status.infrastructureName}{"\n"}' infrastructure cluster
```

2 4 6 Specify the infrastructure ID, worker label, and zone suffix (such as **a**).

7 Configure the machine set as a Windows machine.

8 Configure the Windows node as a compute machine.

9 Specify the full path to an image of a supported version of Windows Server.

11 Specify the GCP project that this cluster was created in.

12 Specify the GCP region, such as **us-central1**.

13 Created by the WMCO when it configures the first Windows machine. After that, the **windows-user-data** is available for all subsequent machine sets to consume.

14 Specify the zone within the chosen region, such as **us-central1-a**.

6.3.3. Creating a compute machine set

In addition to the compute machine sets created by the installation program, you can create your own to dynamically manage the machine compute resources for specific workloads of your choice.

Prerequisites

- Deploy an OpenShift Container Platform cluster.
- Install the OpenShift CLI (**oc**).
- Log in to **oc** as a user with **cluster-admin** permission.

Procedure

1. Create a new YAML file that contains the compute machine set custom resource (CR) sample and is named **<file_name>.yaml**.
   Ensure that you set the **<clusterID>** and **<role>** parameter values.
2. Optional: If you are not sure which value to set for a specific field, you can check an existing compute machine set from your cluster.
   a. To list the compute machine sets in your cluster, run the following command:

```
$ oc get machinesets -n openshift-machine-api
```

Example output

```
NAME                                DESIRED   CURRENT   READY   AVAILABLE   AGE
agl030519-vplxk-worker-us-east-1a   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1b   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1c   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1d   0         0                             55m
agl030519-vplxk-worker-us-east-1e   0         0                             55m
agl030519-vplxk-worker-us-east-1f   0         0                             55m
```

   b. To view values of a specific compute machine set custom resource (CR), run the following command:

```
$ oc get machineset <machineset_name> \
  -n openshift-machine-api -o yaml
```

Example output

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> (1)
  name: <infrastructure_id>-<role> (2)
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: <role>
        machine.openshift.io/cluster-api-machine-type: <role>
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
    spec:
      providerSpec: (3)
        ...
```

1. The cluster infrastructure ID.
2. A default node label.

   NOTE

   For clusters that have user-provisioned infrastructure, a compute machine set can only create **worker** and **infra** type machines.
3. The values in the **<providerSpec>** section of the compute machine set CR are platform-specific. For more information about **<providerSpec>** parameters in the CR, see the sample compute machine set CR configuration for your provider.

3. Create a **MachineSet** CR by running the following command:

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

Verification

- View the list of compute machine sets by running the following command:

```
$ oc get machineset -n openshift-machine-api
```

Example output

```
NAME                          DESIRED   CURRENT   READY   AVAILABLE   AGE
```

```
agl030519-vplxk-infra-us-east-1a    1     1     1     1      11m
agl030519-vplxk-worker-us-east-1a   1     1     1     1      55m
agl030519-vplxk-worker-us-east-1b   1     1     1     1      55m
agl030519-vplxk-worker-us-east-1c   1     1     1     1      55m
agl030519-vplxk-worker-us-east-1d   0     0                  55m
agl030519-vplxk-worker-us-east-1e   0     0                  55m
agl030519-vplxk-worker-us-east-1f   0     0                  55m
```

When the new compute machine set is available, the **DESIRED** and **CURRENT** values match. If the compute machine set is not available, wait a few minutes and run the command again.

6.3.4. Additional resources

- Overview of machine management

6.4. CREATING A WINDOWS MACHINESET OBJECT ON NUTANIX

You can create a Windows **MachineSet** object to serve a specific purpose in your OpenShift Container Platform cluster on Nutanix. For example, you might create infrastructure Windows machine sets and related machines so that you can move supporting Windows workloads to the new Windows machines.

Prerequisites

- You installed the Windows Machine Config Operator (WMCO) using Operator Lifecycle Manager (OLM).
- You are using a supported Windows Server as the operating system image.
- You added a new DNS entry for the internal API server URL, **api-int.<cluster_name>.<base_domain>**, that points to the external API server URL, **api.<cluster_name>.<base_domain>**. This can be a CNAME or an additional A record.

6.4.1. Machine API overview

The Machine API is a combination of primary resources that are based on the upstream Cluster API project and custom OpenShift Container Platform resources.

For OpenShift Container Platform 4.16 clusters, the Machine API performs all node host provisioning management actions after the cluster installation finishes. Because of this system, OpenShift Container Platform 4.16 offers an elastic, dynamic provisioning method on top of public or private cloud infrastructure.

The two primary resources are:

Machines

A fundamental unit that describes the host for a node. A machine has a **providerSpec** specification, which describes the types of compute nodes that are offered for different cloud platforms. For example, a machine type for a compute node might define a specific machine type and required metadata.

Machine sets

MachineSet resources are groups of compute machines. Compute machine sets are to compute machines as replica sets are to pods. If you need more compute machines or must scale them down, you change the **replicas** field on the **MachineSet** resource to meet your compute need.

WARNING

Control plane machines cannot be managed by compute machine sets.

Control plane machine sets provide management capabilities for supported control plane machines that are similar to what compute machine sets provide for compute machines.

For more information, see “Managing control plane machines".

The following custom resources add more capabilities to your cluster:

Machine autoscaler

The **MachineAutoscaler** resource automatically scales compute machines in a cloud. You can set the minimum and maximum scaling boundaries for nodes in a specified compute machine set, and the machine autoscaler maintains that range of nodes.
The **MachineAutoscaler** object takes effect after a **ClusterAutoscaler** object exists. Both **ClusterAutoscaler** and **MachineAutoscaler** resources are made available by the **ClusterAutoscalerOperator** object.

Cluster autoscaler

This resource is based on the upstream cluster autoscaler project. In the OpenShift Container Platform implementation, it is integrated with the Machine API by extending the compute machine set API. You can use the cluster autoscaler to manage your cluster in the following ways:

- Set cluster-wide scaling limits for resources such as cores, nodes, memory, and GPU
- Set the priority so that the cluster prioritizes pods and new nodes are not brought online for less important pods
- Set the scaling policy so that you can scale up nodes but not scale them down

Machine health check

The **MachineHealthCheck** resource detects when a machine is unhealthy, deletes it, and, on supported platforms, makes a new machine.

In OpenShift Container Platform version 3.11, you could not roll out a multi-zone architecture easily because the cluster did not manage machine provisioning. Beginning with OpenShift Container Platform version 4.1, this process is easier. Each compute machine set is scoped to a single zone, so the installation program sends out compute machine sets across availability zones on your behalf. And then because your compute is dynamic, and in the face of a zone failure, you always have a zone for when you must rebalance your machines. In global Azure regions that do not have multiple availability zones, you can use availability sets to ensure high availability. The autoscaler provides best-effort balancing over the life of a cluster.

6.4.2. Sample YAML for a Windows MachineSet object on Nutanix

This sample YAML defines a Windows **MachineSet** object running on Nutanix that the Windows Machine Config Operator (WMCO) can react upon.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-windows-worker-<zone> 2
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id> 3
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-windows-worker-<zone> 4
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id> 5
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-windows-worker-<zone> 6
        machine.openshift.io/os-id: Windows 7
    spec:
      metadata:
        labels:
          node-role.kubernetes.io/worker: "" 8
      providerSpec:
        value:
          apiVersion: machine.openshift.io/v1
          bootType: "" 9
          categories: null
          cluster: 10
            type: uuid
            uuid: <cluster_uuid>
          credentialsSecret:
            name: nutanix-credentials 11
          image: 12
            name: <image_id>
            type: name
          kind: NutanixMachineProviderConfig 13
          memorySize: 16Gi 14
          project:
            type: ""
          subnets: 15
          - type: uuid
            uuid: <subnet_uuid>
          systemDiskSize: 120Gi 16
          userDataSecret:
            name: windows-user-data 17
          vcpuSockets: 4 18
          vcpusPerSocket: 1 19
```

1 3 5 Specify the infrastructure ID that is based on the cluster ID that you set when you provisioned the cluster. You can obtain the infrastructure ID by running the following command:

```
$ oc get -o jsonpath='{.status.infrastructureName}{"\n"}' infrastructure cluster
```

2 4 6 Specify the infrastructure ID, worker label, and zone.

7 Configure the compute machine set as a Windows machine.

8 Configure the Windows node as a compute machine.

9 Specifies the boot type that the compute machines use. For more information about boot types, see Understanding UEFI, Secure Boot, and TPM in the Virtualized Environment . Valid values are **Legacy**, **SecureBoot**, or **UEFI**. The default is **Legacy**.

> NOTE
>
> You must use the **Legacy** boot type in OpenShift Container Platform 4.16.

10 Specifies a Nutanix Prism Element cluster configuration. In this example, the cluster type is **uuid**, so there is a **uuid** stanza.

11 Specifies the secret name for the cluster. Do not change this value.

12 Specifies the image to use. Use an image from an existing default compute machine set for the cluster.

13 Specifies the cloud provider platform type. Do not change this value.

14 Specifies the amount of memory for the cluster in Gi.

15 Specifies a subnet configuration. In this example, the subnet type is **uuid**, so there is a **uuid** stanza.

16 Specifies the size of the system disk in Gi.

17 Specifies the name of the secret in the user data YAML file that is in the **openshift-machine-api** namespace. Use the value that installation program populates in the default compute machine set.

18 Specifies the number of vCPU sockets.

19 Specifies the number of vCPUs per socket.

6.4.3. Creating a compute machine set

In addition to the compute machine sets created by the installation program, you can create your own to dynamically manage the machine compute resources for specific workloads of your choice.

Prerequisites

- Deploy an OpenShift Container Platform cluster.
- Install the OpenShift CLI (**oc**).
- Log in to **oc** as a user with **cluster-admin** permission.

Procedure

1. Create a new YAML file that contains the compute machine set custom resource (CR) sample and is named **<file_name>.yaml**.
   Ensure that you set the **<clusterID>** and **<role>** parameter values.

2. Optional: If you are not sure which value to set for a specific field, you can check an existing compute machine set from your cluster.

   a. To list the compute machine sets in your cluster, run the following command:

   ```
   $ oc get machinesets -n openshift-machine-api
   ```

   Example output

   ```
   NAME                                DESIRED   CURRENT   READY   AVAILABLE   AGE
   agl030519-vplxk-worker-us-east-1a   1         1         1       1           55m
   agl030519-vplxk-worker-us-east-1b   1         1         1       1           55m
   agl030519-vplxk-worker-us-east-1c   1         1         1       1           55m
   agl030519-vplxk-worker-us-east-1d   0         0                             55m
   agl030519-vplxk-worker-us-east-1e   0         0                             55m
   agl030519-vplxk-worker-us-east-1f   0         0                             55m
   ```

   b. To view values of a specific compute machine set custom resource (CR), run the following command:

   ```
   $ oc get machineset <machineset_name> \
     -n openshift-machine-api -o yaml
   ```

   Example output

   ```
   apiVersion: machine.openshift.io/v1beta1
   kind: MachineSet
   metadata:
     labels:
       machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
     name: <infrastructure_id>-<role> 2
     namespace: openshift-machine-api
   spec:
     replicas: 1
     selector:
       matchLabels:
         machine.openshift.io/cluster-api-cluster: <infrastructure_id>
         machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
     template:
       metadata:
         labels:
           machine.openshift.io/cluster-api-cluster: <infrastructure_id>
           machine.openshift.io/cluster-api-machine-role: <role>
           machine.openshift.io/cluster-api-machine-type: <role>
           machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
       spec:
         providerSpec: 3
           ...
   ```

   1 The cluster infrastructure ID.

2 A default node label.

> **NOTE**
>
> For clusters that have user-provisioned infrastructure, a compute machine set can only create **worker** and **infra** type machines.

3 The values in the **<providerSpec>** section of the compute machine set CR are platform-specific. For more information about **<providerSpec>** parameters in the CR, see the sample compute machine set CR configuration for your provider.

3. Create a **MachineSet** CR by running the following command:

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

Verification

- View the list of compute machine sets by running the following command:

```
$ oc get machineset -n openshift-machine-api
```

Example output

```
NAME                          DESIRED   CURRENT   READY   AVAILABLE   AGE
agl030519-vplxk-infra-us-east-1a    1         1         1       1           11m
agl030519-vplxk-worker-us-east-1a   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1b   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1c   1         1         1       1           55m
agl030519-vplxk-worker-us-east-1d   0         0                             55m
agl030519-vplxk-worker-us-east-1e   0         0                             55m
agl030519-vplxk-worker-us-east-1f   0         0                             55m
```

When the new compute machine set is available, the **DESIRED** and **CURRENT** values match. If the compute machine set is not available, wait a few minutes and run the command again.

6.4.4. Additional resources

- Overview of machine management.

6.5. CREATING A WINDOWS MACHINE SET ON VSPHERE

You can create a Windows **MachineSet** object to serve a specific purpose in your OpenShift Container Platform cluster on VMware vSphere. For example, you might create infrastructure Windows machine sets and related machines so that you can move supporting Windows workloads to the new Windows machines.

Prerequisites

- You installed the Windows Machine Config Operator (WMCO) using Operator Lifecycle Manager (OLM).
- You are using a supported Windows Server as the operating system image.

6.5.1. Machine API overview

The Machine API is a combination of primary resources that are based on the upstream Cluster API project and custom OpenShift Container Platform resources.

For OpenShift Container Platform 4.16 clusters, the Machine API performs all node host provisioning management actions after the cluster installation finishes. Because of this system, OpenShift Container Platform 4.16 offers an elastic, dynamic provisioning method on top of public or private cloud infrastructure.

The two primary resources are:

Machines

A fundamental unit that describes the host for a node. A machine has a **providerSpec** specification, which describes the types of compute nodes that are offered for different cloud platforms. For example, a machine type for a compute node might define a specific machine type and required metadata.

Machine sets

MachineSet resources are groups of compute machines. Compute machine sets are to compute machines as replica sets are to pods. If you need more compute machines or must scale them down, you change the **replicas** field on the **MachineSet** resource to meet your compute need.

> **WARNING**
>
> Control plane machines cannot be managed by compute machine sets.
>
> Control plane machine sets provide management capabilities for supported control plane machines that are similar to what compute machine sets provide for compute machines.
>
> For more information, see “Managing control plane machines".

The following custom resources add more capabilities to your cluster:

Machine autoscaler

The **MachineAutoscaler** resource automatically scales compute machines in a cloud. You can set the minimum and maximum scaling boundaries for nodes in a specified compute machine set, and the machine autoscaler maintains that range of nodes.
The **MachineAutoscaler** object takes effect after a **ClusterAutoscaler** object exists. Both **ClusterAutoscaler** and **MachineAutoscaler** resources are made available by the **ClusterAutoscalerOperator** object.

Cluster autoscaler

This resource is based on the upstream cluster autoscaler project. In the OpenShift Container Platform implementation, it is integrated with the Machine API by extending the compute machine set API. You can use the cluster autoscaler to manage your cluster in the following ways:

- Set cluster-wide scaling limits for resources such as cores, nodes, memory, and GPU

- Set the priority so that the cluster prioritizes pods and new nodes are not brought online for less important pods
- Set the scaling policy so that you can scale up nodes but not scale them down

Machine health check

The **MachineHealthCheck** resource detects when a machine is unhealthy, deletes it, and, on supported platforms, makes a new machine.

In OpenShift Container Platform version 3.11, you could not roll out a multi-zone architecture easily because the cluster did not manage machine provisioning. Beginning with OpenShift Container Platform version 4.1, this process is easier. Each compute machine set is scoped to a single zone, so the installation program sends out compute machine sets across availability zones on your behalf. And then because your compute is dynamic, and in the face of a zone failure, you always have a zone for when you must rebalance your machines. In global Azure regions that do not have multiple availability zones, you can use availability sets to ensure high availability. The autoscaler provides best-effort balancing over the life of a cluster.

6.5.2. Preparing your vSphere environment for Windows container workloads

You must prepare your vSphere environment for Windows container workloads by creating the vSphere Windows VM golden image and enabling communication with the internal API server for the WMCO.

6.5.2.1. Creating the vSphere Windows VM golden image

Create a vSphere Windows virtual machine (VM) golden image.

Prerequisites

- You have created a private/public key pair, which is used to configure key-based authentication in the OpenSSH server. The private key must also be configured in the Windows Machine Config Operator (WMCO) namespace. This is required to allow the WMCO to communicate with the Windows VM. See the "Configuring a secret for the Windows Machine Config Operator" section for more details.

NOTE

You must use Microsoft PowerShell commands in several cases when creating your Windows VM. PowerShell commands in this guide are distinguished by the **PS C:\>** prefix.

Procedure

1. Select a compatible Windows Server version. Currently, the Windows Machine Config Operator (WMCO) stable version supports Windows Server 2022 Long-Term Servicing Channel with the OS-level container networking patch KB5012637.
2. Create a new VM in the vSphere client using the VM golden image with a compatible Windows Server version. For more information about compatible versions, see the "Windows Machine Config Operator prerequisites" section of the "Red Hat OpenShift support for Windows Containers release notes."

IMPORTANT

The virtual hardware version for your VM must meet the infrastructure requirements for OpenShift Container Platform. For more information, see the "VMware vSphere infrastructure requirements" section in the OpenShift Container Platform documentation. Also, you can refer to VMware's documentation on virtual machine hardware versions.

3. Install and configure VMware Tools version 11.0.6 or greater on the Windows VM. See the VMware Tools documentation for more information.

4. After installing VMware Tools on the Windows VM, verify the following:

   a. The **C:\ProgramData\VMware\VMware Tools\tools.conf** file exists with the following entry:

   ```
   exclude-nics=
   ```

   If the **tools.conf** file does not exist, create it with the **exclude-nics** option uncommented and set as an empty value.

   This entry ensures the cloned vNIC generated on the Windows VM by the hybrid-overlay is not ignored.

   b. The Windows VM has a valid IP address in vCenter:

   ```
   C:\> ipconfig
   ```

   c. The VMTools Windows service is running:

   ```
   PS C:\> Get-Service -Name VMTools | Select Status, StartType
   ```

5. Install and configure the OpenSSH Server on the Windows VM. See Microsoft's documentation on installing OpenSSH for more details.

6. Set up SSH access for an administrative user. See Microsoft's documentation on the Administrative user to do this.

IMPORTANT

The public key used in the instructions must correspond to the private key you create later in the WMCO namespace that holds your secret. See the "Configuring a secret for the Windows Machine Config Operator" section for more details.

7. You must create a new firewall rule in the Windows VM that allows incoming connections for container logs. Run the following PowerShell command to create the firewall rule on TCP port 10250:

   ```
   PS C:\> New-NetFirewallRule -DisplayName "ContainerLogsPort" -LocalPort 10250 -
   Enabled True -Direction Inbound -Protocol TCP -Action Allow -EdgeTraversalPolicy Allow
   ```

8. Clone the Windows VM so it is a reusable image. Follow the VMware documentation on how to clone an existing virtual machine for more details.

9. In the cloned Windows VM, run the Windows Sysprep tool:

```
C:\> C:\Windows\System32\Sysprep\sysprep.exe /generalize /oobe /shutdown /unattend:
<path_to_unattend.xml> 1
```

1 Specify the path to your **unattend.xml** file.

NOTE

There is a limit on how many times you can run the **sysprep** command on a Windows image. Consult Microsoft's documentation for more information.

An example **unattend.xml** is provided, which maintains all the changes needed for the WMCO. You must modify this example; it cannot be used directly.

Example 6.1. Example unattend.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<unattend xmlns="urn:schemas-microsoft-com:unattend">
  <settings pass="specialize">
    <component xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="Microsoft-Windows-
International-Core" processorArchitecture="amd64"
publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS">
      <InputLocale>0409:00000409</InputLocale>
      <SystemLocale>en-US</SystemLocale>
      <UILanguage>en-US</UILanguage>
      <UILanguageFallback>en-US</UILanguageFallback>
      <UserLocale>en-US</UserLocale>
    </component>
    <component xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="Microsoft-Windows-
Security-SPP-UX" processorArchitecture="amd64" publicKeyToken="31bf3856ad364e35"
language="neutral" versionScope="nonSxS">
      <SkipAutoActivation>true</SkipAutoActivation>
    </component>
    <component xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="Microsoft-Windows-
SQMApi" processorArchitecture="amd64" publicKeyToken="31bf3856ad364e35"
language="neutral" versionScope="nonSxS">
      <CEIPEnabled>0</CEIPEnabled>
    </component>
    <component xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="Microsoft-Windows-
Shell-Setup" processorArchitecture="amd64" publicKeyToken="31bf3856ad364e35"
language="neutral" versionScope="nonSxS">
      <ComputerName>winhost</ComputerName> 1
    </component>
  </settings>
  <settings pass="oobeSystem">
    <component xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="Microsoft-Windows-
Shell-Setup" processorArchitecture="amd64" publicKeyToken="31bf3856ad364e35"
```

```
language="neutral" versionScope="nonSxS">
      <AutoLogon>
        <Enabled>false</Enabled> 2
      </AutoLogon>
      <OOBE>
        <HideEULAPage>true</HideEULAPage>
        <HideLocalAccountScreen>true</HideLocalAccountScreen>
        <HideOEMRegistrationScreen>true</HideOEMRegistrationScreen>
        <HideOnlineAccountScreens>true</HideOnlineAccountScreens>
        <HideWirelessSetupInOOBE>true</HideWirelessSetupInOOBE>
        <NetworkLocation>Work</NetworkLocation>
        <ProtectYourPC>1</ProtectYourPC>
        <SkipMachineOOBE>true</SkipMachineOOBE>
        <SkipUserOOBE>true</SkipUserOOBE>
      </OOBE>
      <RegisteredOrganization>Organization</RegisteredOrganization>
      <RegisteredOwner>Owner</RegisteredOwner>
      <DisableAutoDaylightTimeSet>false</DisableAutoDaylightTimeSet>
      <TimeZone>Eastern Standard Time</TimeZone>
      <UserAccounts>
        <AdministratorPassword>
          <Value>MyPassword</Value> 3
          <PlainText>true</PlainText>
        </AdministratorPassword>
      </UserAccounts>
    </component>
  </settings>
</unattend>
```

1. Specify the **ComputerName**, which must follow the Kubernetes' names specification. These specifications also apply to Guest OS customization performed on the resulting template while creating new VMs.

2. Disable the automatic logon to avoid the security issue of leaving an open terminal with Administrator privileges at boot. This is the default value and must not be changed.

3. Replace the **MyPassword** placeholder with the password for the Administrator account. This prevents the built-in Administrator account from having a blank password by default. Follow Microsoft's best practices for choosing a password .

After the Sysprep tool has completed, the Windows VM will power off. You must not use or power on this VM anymore.

10. Convert the Windows VM to a template in vCenter .

6.5.2.1.1. Additional resources

- Configuring a secret for the Windows Machine Config Operator
- VMware vSphere infrastructure requirements

6.5.2.2. Enabling communication with the internal API server for the WMCO on vSphere

The Windows Machine Config Operator (WMCO) downloads the Ignition config files from the internal API server endpoint. You must enable communication with the internal API server so that your Windows virtual machine (VM) can download the Ignition config files, and the kubelet on the configured VM can only communicate with the internal API server.

Prerequisites

- You have installed a cluster on vSphere.

Procedure

- Add a new DNS entry for **api-int.<cluster_name>.<base_domain>** that points to the external API server URL **api.<cluster_name>.<base_domain>**. This can be a CNAME or an additional A record.

NOTE

The external API endpoint was already created as part of the initial cluster installation on vSphere.

6.5.3. Sample YAML for a Windows MachineSet object on vSphere

This sample YAML defines a Windows **MachineSet** object running on VMware vSphere that the Windows Machine Config Operator (WMCO) can react upon.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <windows_machine_set_name> 2
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id> 3
      machine.openshift.io/cluster-api-machineset: <windows_machine_set_name> 4
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id> 5
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: <windows_machine_set_name> 6
        machine.openshift.io/os-id: Windows 7
    spec:
      metadata:
        labels:
          node-role.kubernetes.io/worker: "" 8
      providerSpec:
        value:
          apiVersion: vsphereprovider.openshift.io/v1beta1
          credentialsSecret:
```

```
          name: vsphere-cloud-credentials
        diskGiB: 128 (9)
        kind: VSphereMachineProviderSpec
        memoryMiB: 16384
        network:
          devices:
          - networkName: "<vm_network_name>" (10)
        numCPUs: 4
        numCoresPerSocket: 1
        snapshot: ""
        template: <windows_vm_template_name> (11)
        userDataSecret:
          name: windows-user-data (12)
        workspace:
           datacenter: <vcenter_data_center_name> (13)
           datastore: <vcenter_datastore_name> (14)
           folder: <vcenter_vm_folder_path> (15)
           resourcePool: <vsphere_resource_pool> (16)
           server: <vcenter_server_ip> (17)
```

1 3 5 Specify the infrastructure ID that is based on the cluster ID that you set when you provisioned the cluster. You can obtain the infrastructure ID by running the following command:

```
$ oc get -o jsonpath='{.status.infrastructureName}{"\n"}' infrastructure cluster
```

2 4 6 Specify the Windows compute machine set name. The compute machine set name cannot be more than 9 characters long, due to the way machine names are generated in vSphere.

7 Configure the compute machine set as a Windows machine.

8 Configure the Windows node as a compute machine.

9 Specify the size of the vSphere Virtual Machine Disk (VMDK).

NOTE

This parameter does not set the size of the Windows partition. You can resize the Windows partition by using the **unattend.xml** file or by creating the vSphere Windows virtual machine (VM) golden image with the required disk size.

10 Specify the vSphere VM network to deploy the compute machine set to. This VM network must be where other Linux compute machines reside in the cluster.

11 Specify the full path of the Windows vSphere VM template to use, such as **golden-images/windows-server-template**. The name must be unique.

IMPORTANT

Do not specify the original VM template. The VM template must remain off and must be cloned for new Windows machines. Starting the VM template configures the VM template as a VM on the platform, which prevents it from being used as a template that compute machine sets can apply configurations to.

12 The **windows-user-data** is created by the WMCO when the first Windows machine is configured. After that, the **windows-user-data** is available for all subsequent compute machine sets to

13 Specify the vCenter data center to deploy the compute machine set on.

14 Specify the vCenter datastore to deploy the compute machine set on.

15 Specify the path to the vSphere VM folder in vCenter, such as **/dc1/vm/user-inst-5ddjd**.

16 Optional: Specify the vSphere resource pool for your Windows VMs.

17 Specify the vCenter server IP or fully qualified domain name.

6.5.4. Creating a compute machine set

In addition to the compute machine sets created by the installation program, you can create your own to dynamically manage the machine compute resources for specific workloads of your choice.

Prerequisites

- Deploy an OpenShift Container Platform cluster.
- Install the OpenShift CLI (**oc**).
- Log in to **oc** as a user with **cluster-admin** permission.
- In disconnected environments, the image specified in the **MachineSet** custom resource (CR) must have the OpenSSH server v0.0.1.0 installed.

Procedure

1. Create a new YAML file that contains the compute machine set custom resource (CR) sample and is named **<file_name>.yaml**.
   Ensure that you set the **<clusterID>** and **<role>** parameter values.
2. Optional: If you are not sure which value to set for a specific field, you can check an existing compute machine set from your cluster.
   a. To list the compute machine sets in your cluster, run the following command:

   ```
   $ oc get machinesets -n openshift-machine-api
   ```

   #### Example output

   ```
   NAME                                DESIRED   CURRENT   READY   AVAILABLE   AGE
   agl030519-vplxk-worker-us-east-1a   1         1         1       1           55m
   agl030519-vplxk-worker-us-east-1b   1         1         1       1           55m
   agl030519-vplxk-worker-us-east-1c   1         1         1       1           55m
   agl030519-vplxk-worker-us-east-1d   0         0                             55m
   agl030519-vplxk-worker-us-east-1e   0         0                             55m
   agl030519-vplxk-worker-us-east-1f   0         0                             55m
   ```

   b. To view values of a specific compute machine set custom resource (CR), run the following command:

```
$ oc get machineset <machineset_name> \
  -n openshift-machine-api -o yaml
```

Example output

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-<role> 2
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: <role>
        machine.openshift.io/cluster-api-machine-type: <role>
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
    spec:
      providerSpec: 3
        ...
```

1 The cluster infrastructure ID.

2 A default node label.

NOTE

For clusters that have user-provisioned infrastructure, a compute machine set can only create **worker** and **infra** type machines.

3 The values in the **<providerSpec>** section of the compute machine set CR are platform-specific. For more information about **<providerSpec>** parameters in the CR, see the sample compute machine set CR configuration for your provider.

3. Create a **MachineSet** CR by running the following command:

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

Verification

- View the list of compute machine sets by running the following command:

```
$ oc get machineset -n openshift-machine-api
```

Example output

```
NAME                                DESIRED   CURRENT   READY   AVAILABLE   AGE
agl030519-vplxk-windows-worker-us-east-1a  1        1        1       1          11m
agl030519-vplxk-worker-us-east-1a        1        1        1       1          55m
agl030519-vplxk-worker-us-east-1b        1        1        1       1          55m
agl030519-vplxk-worker-us-east-1c        1        1        1       1          55m
agl030519-vplxk-worker-us-east-1d        0        0                          55m
agl030519-vplxk-worker-us-east-1e        0        0                          55m
agl030519-vplxk-worker-us-east-1f        0        0                          55m
```

When the new compute machine set is available, the **DESIRED** and **CURRENT** values match. If the compute machine set is not available, wait a few minutes and run the command again.

6.5.5. Additional resources

- Overview of machine management

CHAPTER 7. SCHEDULING WINDOWS CONTAINER WORKLOADS

You can schedule Windows workloads to Windows compute nodes.

Prerequisites

- You installed the Windows Machine Config Operator (WMCO) using Operator Lifecycle Manager (OLM).
- You are using a Windows container as the OS image.
- You have created a Windows compute machine set.

7.1. WINDOWS POD PLACEMENT

Before deploying your Windows workloads to the cluster, you must configure your Windows node scheduling so pods are assigned correctly. Since you have a machine hosting your Windows node, it is managed the same as a Linux-based node. Likewise, scheduling a Windows pod to the appropriate Windows node is completed similarly, using mechanisms like taints, tolerations, and node selectors.

With multiple operating systems, and the ability to run multiple Windows OS variants in the same cluster, you must map your Windows pods to a base Windows OS variant by using a **RuntimeClass** object. For example, if you have multiple Windows nodes running on different Windows Server container versions, the cluster could schedule your Windows pods to an incompatible Windows OS variant. You must have **RuntimeClass** objects configured for each Windows OS variant on your cluster. Using a **RuntimeClass** object is also recommended if you have only one Windows OS variant available in your cluster.

For more information, see Microsoft's documentation on Host and container version compatibility.

Also, it is recommended that you set the **spec.os.name.windows** parameter in your workload pods. The Windows Machine Config Operator (WMCO) uses this field to authoritatively identify the pod operating system for validation and is used to enforce Windows-specific pod security context constraints (SCCs). Currently, this parameter has no effect on pod scheduling. For more information about this parameter, see the Kubernetes Pods documentation.

IMPORTANT

The container base image must be the same Windows OS version and build number that is running on the node where the conainer is to be scheduled.

Also, if you upgrade the Windows nodes from one version to another, for example going from 20H2 to 2022, you must upgrade your container base image to match the new version. For more information, see Windows container version compatibility.

Additional resources

- Controlling pod placement using the scheduler
- Controlling pod placement using node taints
- Placing pods on specific nodes using node selectors

7.2. CREATING A RUNTIMECLASS OBJECT TO ENCAPSULATE SCHEDULING MECHANISMS

Using a **RuntimeClass** object simplifies the use of scheduling mechanisms like taints and tolerations; you deploy a runtime class that encapsulates your taints and tolerations and then apply it to your pods to schedule them to the appropriate node. Creating a runtime class is also necessary in clusters that support multiple operating system variants.

Procedure

1. Create a **RuntimeClass** object YAML file. For example, **runtime-class.yaml**:

```
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: windows2019 (1)
handler: 'runhcs-wcow-process'
scheduling:
  nodeSelector: (2)
    kubernetes.io/os: 'windows'
    kubernetes.io/arch: 'amd64'
    node.kubernetes.io/windows-build: '10.0.17763'
  tolerations: (3)
  - effect: NoSchedule
    key: os
    operator: Equal
    value: "windows"
  - effect: NoSchedule
    key: os
    operator: Equal
    value: "Windows"
```

(1) Specify the **RuntimeClass** object name, which is defined in the pods you want to be managed by this runtime class.

(2) Specify labels that must be present on nodes that support this runtime class. Pods using this runtime class can only be scheduled to a node matched by this selector. The node selector of the runtime class is merged with the existing node selector of the pod. Any conflicts prevent the pod from being scheduled to the node.

- For Windows 2019, specify the **node.kubernetes.io/windows-build: '10.0.17763'** label.
- For Windows 2022, specify the **node.kubernetes.io/windows-build: '10.0.20348'** label.

(3) Specify tolerations to append to pods, excluding duplicates, running with this runtime class during admission. This combines the set of nodes tolerated by the pod and the runtime class.

2. Create the **RuntimeClass** object:

```
$ oc create -f <file-name>.yaml
```

For example:

```
$ oc create -f runtime-class.yaml
```

3. Apply the **RuntimeClass** object to your pod to ensure it is scheduled to the appropriate operating system variant:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-windows-pod
spec:
  runtimeClassName: windows2019 (1)
# ...
```

(1) Specify the runtime class to manage the scheduling of your pod.

7.3. SAMPLE WINDOWS CONTAINER WORKLOAD DEPLOYMENT

You can deploy Windows container workloads to your cluster once you have a Windows compute node available.

NOTE

This sample deployment is provided for reference only.

Example Service object

```
apiVersion: v1
kind: Service
metadata:
  name: win-webserver
  labels:
    app: win-webserver
spec:
  ports:
    # the port that this service should serve on
  - port: 80
    targetPort: 80
  selector:
    app: win-webserver
  type: LoadBalancer
```

Example Deployment object

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: win-webserver
  name: win-webserver
spec:
```

```
selector:
  matchLabels:
    app: win-webserver
replicas: 1
template:
  metadata:
    labels:
      app: win-webserver
    name: win-webserver
  spec:
    containers:
    - name: windowswebserver
      image: mcr.microsoft.com/windows/servercore:ltsc2019 (1)
      imagePullPolicy: IfNotPresent
      command:
      - powershell.exe (2)
      - -command
      - $listener = New-Object System.Net.HttpListener; $listener.Prefixes.Add('http://*:80/');
$listener.Start();Write-Host('Listening at http://*:80/'); while ($listener.IsListening) { $context =
$listener.GetContext(); $response = $context.Response; $content='<html><body><H1>Red Hat
OpenShift + Windows Container Workloads</H1></body></html>'; $buffer =
[System.Text.Encoding]::UTF8.GetBytes($content); $response.ContentLength64 = $buffer.Length;
$response.OutputStream.Write($buffer, 0, $buffer.Length); $response.Close(); };
      securityContext:
        runAsNonRoot: false
        windowsOptions:
          runAsUserName: "ContainerAdministrator"
    os:
      name: "windows"
    runtimeClassName: windows2019 (3)
```

1. Specify the container image to use: **mcr.microsoft.com/powershell:<tag>** or **mcr.microsoft.com/windows/servercore:<tag>**. The container image must match the Windows version running on the node.

    - For Windows 2019, use the **ltsc2019** tag.
    - For Windows 2022, use the **ltsc2022** tag.

2. Specify the commands to execute on the container.

    - For the **mcr.microsoft.com/powershell:<tag>** container image, you must define the command as **pwsh.exe**.
    - For the **mcr.microsoft.com/windows/servercore:<tag>** container image, you must define the command as **powershell.exe**.

3. Specify the runtime class you created for the Windows operating system variant on your cluster.

7.4. SUPPORT FOR WINDOWS CSI DRIVERS

Red Hat OpenShift support for Windows Containers installs CSI Proxy on all Windows nodes in the cluster. CSI Proxy is a plug-in that enables CSI drivers to perform storage operations on the node.

To use persistent storage with Windows workloads, you must deploy a specific Windows CSI driver daemon set, as described in your storage provider's documentation. By default, the WMCO does not automatically create the Windows CSI driver daemon set. See the list of supported production drivers in the Kubernetes Container Storage Interface (CSI) Documentation.

7.5. SCALING A COMPUTE MACHINE SET MANUALLY

To add or remove an instance of a machine in a compute machine set, you can manually scale the compute machine set.

This guidance is relevant to fully automated, installer-provisioned infrastructure installations. Customized, user-provisioned infrastructure installations do not have compute machine sets.

Prerequisites

- Install an OpenShift Container Platform cluster and the **oc** command line.
- Log in to **oc** as a user with **cluster-admin** permission.

Procedure

1. View the compute machine sets that are in the cluster by running the following command:

   ```
   $ oc get machinesets.machine.openshift.io -n openshift-machine-api
   ```

   The compute machine sets are listed in the form of **<clusterid>-worker-<aws-region-az>**.

2. View the compute machines that are in the cluster by running the following command:

   ```
   $ oc get machines.machine.openshift.io -n openshift-machine-api
   ```

3. Set the annotation on the compute machine that you want to delete by running the following command:

   ```
   $ oc annotate machines.machine.openshift.io/<machine_name> -n openshift-machine-api
   machine.openshift.io/delete-machine="true"
   ```

4. Scale the compute machine set by running one of the following commands:

   ```
   $ oc scale --replicas=2 machinesets.machine.openshift.io <machineset> -n openshift-
   machine-api
   ```

   Or:

   ```
   $ oc edit machinesets.machine.openshift.io <machineset> -n openshift-machine-api
   ```

TIP

You can alternatively apply the following YAML to scale the compute machine set:

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 2
```

You can scale the compute machine set up or down. It takes several minutes for the new machines to be available.

IMPORTANT

By default, the machine controller tries to drain the node that is backed by the machine until it succeeds. In some situations, such as with a misconfigured pod disruption budget, the drain operation might not be able to succeed. If the drain operation fails, the machine controller cannot proceed removing the machine.

You can skip draining the node by annotating **machine.openshift.io/exclude-node-draining** in a specific machine.

Verification

- Verify the deletion of the intended machine by running the following command:

```
$ oc get machines.machine.openshift.io
```

CHAPTER 8. WINDOWS NODE UPGRADES

You can ensure your Windows nodes have the latest updates by upgrading the Windows Machine Config Operator (WMCO).

8.1. WINDOWS MACHINE CONFIG OPERATOR UPGRADES

When a new version of the Windows Machine Config Operator (WMCO) is released that is compatible with the current cluster version, the Operator is upgraded based on the upgrade channel and subscription approval strategy it was installed with when using the Operator Lifecycle Manager (OLM). The WMCO upgrade results in the Kubernetes components in the Windows machine being upgraded.

NOTE

If you are upgrading to a new version of the WMCO and want to use cluster monitoring, you must have the **openshift.io/cluster-monitoring=true** label present in the WMCO namespace. If you add the label to a pre-existing WMCO namespace, and there are already Windows nodes configured, restart the WMCO pod to allow monitoring graphs to display.

For a non-disruptive upgrade, the WMCO terminates the Windows machines configured by the previous version of the WMCO and recreates them using the current version. This is done by deleting the **Machine** object, which results in the drain and deletion of the Windows node. To facilitate an upgrade, the WMCO adds a version annotation to all the configured nodes. During an upgrade, a mismatch in version annotation results in the deletion and recreation of a Windows machine. To have minimal service disruptions during an upgrade, the WMCO only updates one Windows machine at a time.

After the update, it is recommended that you set the **spec.os.name.windows** parameter in your workload pods. The WMCO uses this field to authoritatively identify the pod operating system for validation and is used to enforce Windows-specific pod security context constraints (SCCs).

IMPORTANT

The WMCO is only responsible for updating Kubernetes components, not for Windows operating system updates. You provide the Windows image when creating the VMs; therefore, you are responsible for providing an updated image. You can provide an updated Windows image by changing the image configuration in the **MachineSet** spec.

For more information on Operator upgrades using the Operator Lifecycle Manager (OLM), see Updating installed Operators.

CHAPTER 9. USING BRING-YOUR-OWN-HOST (BYOH) WINDOWS INSTANCES AS NODES

Bring-Your-Own-Host (BYOH) allows for users to repurpose Windows Server VMs and bring them to OpenShift Container Platform. BYOH Windows instances benefit users looking to mitigate major disruptions in the event that a Windows server goes offline.

9.1. CONFIGURING A BYOH WINDOWS INSTANCE

Creating a BYOH Windows instance requires creating a config map in the Windows Machine Config Operator (WMCO) namespace.

Prerequisites

Any Windows instances that are to be attached to the cluster as a node must fulfill the following requirements:

- The instance must be on the same network as the Linux worker nodes in the cluster.
- Port 22 must be open and running an SSH server.
- The default shell for the SSH server must be the Windows Command shell, or **cmd.exe**.
- Port 10250 must be open for log collection.
- An administrator user is present with the private key used in the secret set as an authorized SSH key.
- If you are creating a BYOH Windows instance for an installer-provisioned infrastructure (IPI) AWS cluster, you must add a tag to the AWS instance that matches the **spec.template.spec.value.tag** value in the compute machine set for your worker nodes. For example, **kubernetes.io/cluster/<cluster_id>: owned** or **kubernetes.io/cluster/<cluster_id>: shared**.
- If you are creating a BYOH Windows instance on vSphere, communication with the internal API server must be enabled.
- The hostname of the instance must follow the RFC 1123 DNS label requirements, which include the following standards:
  - Contains only lowercase alphanumeric characters or '-'.
  - Starts with an alphanumeric character.
  - Ends with an alphanumeric character.

NOTE

Windows instances deployed by the WMCO are configured with the containerd container runtime. Because the WMCO installs and manages the runtime, it is recommended that you not manually install containerd on nodes.

Procedure

1. Create a ConfigMap named **windows-instances** in the WMCO namespace that describes the Windows instances to be added.

   NOTE

   Format each entry in the config map's data section by using the address as the key while formatting the value as **username=<username>**.

   Example config map

   ```
   kind: ConfigMap
   apiVersion: v1
   metadata:
     name: windows-instances
     namespace: openshift-windows-machine-config-operator
   data:
     10.1.42.1: |- 1
       username=Administrator 2
     instance.example.com: |-
       username=core
   ```

   1 The address that the WMCO uses to reach the instance over SSH, either a DNS name or an IPv4 address. A DNS PTR record must exist for this address. It is recommended that you use a DNS name with your BYOH instance if your organization uses DHCP to assign IP addresses. If not, you need to update the **windows-instances** ConfigMap whenever the instance is assigned a new IP address.

   2 The name of the administrator user created in the prerequisites.

9.2. REMOVING BYOH WINDOWS INSTANCES

You can remove BYOH instances attached to the cluster by deleting the instance's entry in the config map. Deleting an instance reverts that instance back to its state prior to adding to the cluster. Any logs and container runtime artifacts are not added to these instances.

For an instance to be cleanly removed, it must be accessible with the current private key provided to WMCO. For example, to remove the **10.1.42.1** instance from the previous example, the config map would be changed to the following:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: windows-instances
  namespace: openshift-windows-machine-config-operator
data:
  instance.example.com: |-
    username=core
```

Deleting **windows-instances** is viewed as a request to deconstruct all Windows instances added as nodes.

CHAPTER 10. REMOVING WINDOWS NODES

You can remove a Windows node by deleting its host Windows machine.

10.1. DELETING A SPECIFIC MACHINE

You can delete a specific machine.

IMPORTANT

Do not delete a control plane machine unless your cluster uses a control plane machine set.

Prerequisites

- Install an OpenShift Container Platform cluster.
- Install the OpenShift CLI (**oc**).
- Log in to **oc** as a user with **cluster-admin** permission.

Procedure

1. View the machines that are in the cluster by running the following command:

   ```
   $ oc get machine -n openshift-machine-api
   ```

   The command output contains a list of machines in the **<clusterid>-<role>-<cloud_region>** format.

2. Identify the machine that you want to delete.

3. Delete the machine by running the following command:

   ```
   $ oc delete machine <machine> -n openshift-machine-api
   ```

   ### IMPORTANT

   By default, the machine controller tries to drain the node that is backed by the machine until it succeeds. In some situations, such as with a misconfigured pod disruption budget, the drain operation might not be able to succeed. If the drain operation fails, the machine controller cannot proceed removing the machine.

   You can skip draining the node by annotating **machine.openshift.io/exclude-node-draining** in a specific machine.

   If the machine that you delete belongs to a machine set, a new machine is immediately created to satisfy the specified number of replicas.

CHAPTER 11. DISABLING WINDOWS CONTAINER WORKLOADS

You can disable the capability to run Windows container workloads by uninstalling the Windows Machine Config Operator (WMCO) and deleting the namespace that was added by default when you installed the WMCO.

11.1. UNINSTALLING THE WINDOWS MACHINE CONFIG OPERATOR

You can uninstall the Windows Machine Config Operator (WMCO) from your cluster.

Prerequisites

- Delete the Windows **Machine** objects hosting your Windows workloads.

Procedure

1. From the **Operators → OperatorHub** page, use the **Filter by keyword** box to search for **Red Hat Windows Machine Config Operator**.
2. Click the **Red Hat Windows Machine Config Operator** tile. The Operator tile indicates it is installed.
3. In the **Windows Machine Config Operator** descriptor page, click **Uninstall**.

11.2. DELETING THE WINDOWS MACHINE CONFIG OPERATOR NAMESPACE

You can delete the namespace that was generated for the Windows Machine Config Operator (WMCO) by default.

Prerequisites

- The WMCO is removed from your cluster.

Procedure

1. Remove all Windows workloads that were created in the **openshift-windows-machine-config-operator** namespace:

   ```
   $ oc delete --all pods --namespace=openshift-windows-machine-config-operator
   ```

2. Verify that all pods in the **openshift-windows-machine-config-operator** namespace are deleted or are reporting a terminating state:

   ```
   $ oc get pods --namespace openshift-windows-machine-config-operator
   ```

3. Delete the **openshift-windows-machine-config-operator** namespace:

   ```
   $ oc delete namespace openshift-windows-machine-config-operator
   ```

Additional resources

- Deleting Operators from a cluster
- Removing Windows nodes