



# Workload Availability for Red Hat OpenShift 23.2

## Remediation, fencing, and maintenance

Workload Availability remediation, fencing, and maintenance



## Workload Availability for Red Hat OpenShift 23.2 Remediation, fencing, and maintenance

---

Workload Availability remediation, fencing, and maintenance

## 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<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

Node.js<sup>®</sup> 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<sup>®</sup> 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

Information about workload availability operators and their usage

## Table of Contents

<b>PREFACE</b> .....	<b>4</b>
<b>PROVIDING FEEDBACK ON WORKLOAD AVAILABILITY FOR RED HAT OPENSIFT DOCUMENTATION</b> .	<b>5</b>
<b>CHAPTER 1. ABOUT NODE REMEDIATION, FENCING, AND MAINTENANCE</b> .....	<b>6</b>
1.1. SELF NODE REMEDIATION	6
1.2. MACHINE HEALTH CHECK	6
1.3. NODE HEALTH CHECK	6
1.4. NODE MAINTENANCE	6
<b>CHAPTER 2. USING SELF NODE REMEDIATION</b> .....	<b>7</b>
2.1. ABOUT THE SELF NODE REMEDIATION OPERATOR	7
2.1.1. About watchdog devices	7
2.1.1.1. Understanding Self Node Remediation Operator behavior with watchdog devices	8
2.2. CONTROL PLANE FENCING	8
2.3. INSTALLING THE SELF NODE REMEDIATION OPERATOR BY USING THE WEB CONSOLE	9
2.4. INSTALLING THE SELF NODE REMEDIATION OPERATOR BY USING THE CLI	9
2.5. CONFIGURING THE SELF NODE REMEDIATION OPERATOR	12
2.5.1. Understanding the Self Node Remediation Operator configuration	12
2.5.2. Understanding the Self Node Remediation Template configuration	13
2.5.3. Troubleshooting the Self Node Remediation Operator	14
2.5.3.1. General troubleshooting	14
2.5.3.2. Checking the daemon set	14
2.5.3.3. Unsuccessful remediation	14
2.5.3.4. Daemon set and other Self Node Remediation Operator resources exist even after uninstalling the Operator	15
2.5.4. Gathering data about the Self Node Remediation Operator	15
2.5.5. Additional resources	15
<b>CHAPTER 3. REMEDIATING NODES WITH MACHINE HEALTH CHECKS</b> .....	<b>16</b>
3.1. ABOUT MACHINE HEALTH CHECKS	16
3.1.1. Limitations when deploying machine health checks	16
3.2. CONFIGURING MACHINE HEALTH CHECKS TO USE THE SELF NODE REMEDIATION OPERATOR	17
<b>CHAPTER 4. REMEDIATING NODES WITH NODE HEALTH CHECKS</b> .....	<b>19</b>
4.1. ABOUT THE NODE HEALTH CHECK OPERATOR	19
4.1.1. Understanding the Node Health Check Operator workflow	21
4.1.2. About how node health checks prevent conflicts with machine health checks	21
4.2. CONTROL PLANE FENCING	22
4.3. INSTALLING THE NODE HEALTH CHECK OPERATOR BY USING THE WEB CONSOLE	23
4.4. INSTALLING THE NODE HEALTH CHECK OPERATOR BY USING THE CLI	23
4.5. CREATING A NODE HEALTH CHECK	25
4.6. GATHERING DATA ABOUT THE NODE HEALTH CHECK OPERATOR	26
4.7. ADDITIONAL RESOURCES	26
<b>CHAPTER 5. PLACING NODES IN MAINTENANCE MODE WITH NODE MAINTENANCE OPERATOR</b> .....	<b>27</b>
5.1. ABOUT THE NODE MAINTENANCE OPERATOR	27
5.2. INSTALLING THE NODE MAINTENANCE OPERATOR	27
5.2.1. Installing the Node Maintenance Operator by using the web console	27
5.2.2. Installing the Node Maintenance Operator by using the CLI	28
5.3. SETTING A NODE TO MAINTENANCE MODE	30
5.3.1. Setting a node to maintenance mode by using the web console	30
5.3.2. Setting a node to maintenance mode by using the CLI	30

5.3.3. Checking status of current NodeMaintenance CR tasks	31
5.4. RESUMING A NODE FROM MAINTENANCE MODE	32
5.4.1. Resuming a node from maintenance mode by using the web console	32
5.4.2. Resuming a node from maintenance mode by using the CLI	33
5.5. WORKING WITH BARE-METAL NODES	34
5.5.1. Maintaining bare-metal nodes	34
5.5.2. Setting a bare-metal node to maintenance mode	34
5.5.3. Resuming a bare-metal node from maintenance mode	35
5.6. GATHERING DATA ABOUT THE NODE MAINTENANCE OPERATOR	35
5.7. ADDITIONAL RESOURCES	35



## PREFACE



## PROVIDING FEEDBACK ON WORKLOAD AVAILABILITY FOR RED HAT OPENSIFT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it. To do so:

1. Go to the [JIRA](#) website.
2. Enter a descriptive title in the **Summary** field.
3. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
4. Enter your username in the **Reporter** field.
5. Enter the affected versions in the **Affects Version/s** field.
6. Click **Create** at the bottom of the dialog.

# CHAPTER 1. ABOUT NODE REMEDIATION, FENCING, AND MAINTENANCE

Hardware is imperfect and software contains bugs. When node-level failures, such as the kernel hangs or network interface controllers (NICs) fail, the work required from the cluster does not decrease, and workloads from affected nodes need to be restarted somewhere. However, some workloads, such as ReadWriteOnce (RWO) volumes and StatefulSets, might require at-most-one semantics.

Failures affecting these workloads risk data loss, corruption, or both. It is important to ensure that the node reaches a safe state, known as **fencing** before initiating recovery of the workload, known as **remediation** and ideally, recovery of the node also.

It is not always practical to depend on administrator intervention to confirm the true status of the nodes and workloads. To facilitate such intervention, Red Hat OpenShift provides multiple components for the automation of failure detection, fencing and remediation.

## 1.1. SELF NODE REMEDIATION

The Self Node Remediation Operator is an Red Hat OpenShift add-on operator which implements an external system of fencing and remediation that reboots unhealthy nodes and deletes resources, such as, Pods and VolumeAttachments. The reboot ensures that the workloads are fenced, and the resource deletion accelerates the rescheduling of affected workloads. Unlike other external systems, Self Node Remediation does not require any management interface, like, for example, Intelligent Platform Management Interface (IPMI) or an API for node provisioning.

Self Node Remediation can be used by failure detection systems, like Machine Health Check or Node Health Check.

## 1.2. MACHINE HEALTH CHECK

Machine Health Check utilizes an Red Hat OpenShift built-in failure detection, fencing and remediation system, which monitors the status of machines and the conditions of nodes. Machine Health Checks can be configured to trigger external fencing and remediation systems, like Self Node Remediation.

## 1.3. NODE HEALTH CHECK

The Node Health Check Operator is an Red Hat OpenShift add-on operator which implements a failure detection system that monitors node conditions. It does not have a built-in fencing or remediation system and so must be configured with an external system that provides such features. By default, it is configured to utilize the Self Node Remediation system.

## 1.4. NODE MAINTENANCE

Administrators face situations where they need to interrupt the cluster, for example, replace a drive, RAM, or a NIC.

In advance of this maintenance, affected nodes should be cordoned and drained. When a node is cordoned, new workloads cannot be scheduled on that node. When a node is drained, to avoid or minimize downtime, workloads on the affected node are transferred to other nodes.

While this maintenance can be achieved using command line tools, the Node Maintenance Operator offers a declarative approach to achieve this by using a custom resource. When such a resource exists for a node, the operator cordons and drains the node until the resource is deleted.

## CHAPTER 2. USING SELF NODE REMEDIATION

You can use the Self Node Remediation Operator to automatically reboot unhealthy nodes. This remediation strategy minimizes downtime for stateful applications and ReadWriteOnce (RWO) volumes, and restores compute capacity if transient failures occur.

### 2.1. ABOUT THE SELF NODE REMEDIATION OPERATOR

The Self Node Remediation Operator runs on the cluster nodes and reboots nodes that are identified as unhealthy. The Operator uses the **MachineHealthCheck** or **NodeHealthCheck** controller to detect the health of a node in the cluster. When a node is identified as unhealthy, the **MachineHealthCheck** or the **NodeHealthCheck** resource creates the **SelfNodeRemediation** custom resource (CR), which triggers the Self Node Remediation Operator.

The **SelfNodeRemediation** CR resembles the following YAML file:

```
apiVersion: self-node-remediation.medik8s.io/v1alpha1
kind: SelfNodeRemediation
metadata:
  name: selfnoderemediation-sample
  namespace: openshift-operators
spec:
  remediationStrategy: <remediation_strategy> 1
status:
  lastError: <last_error_message> 2
```

- 1 Specifies the remediation strategy for the nodes.
- 2 Displays the last error that occurred during remediation. When remediation succeeds or if no errors occur, the field is left empty.

The Self Node Remediation Operator minimizes downtime for stateful applications and restores compute capacity if transient failures occur. You can use this Operator regardless of the management interface, such as IPMI or an API to provision a node, and regardless of the cluster installation type, such as installer-provisioned infrastructure or user-provisioned infrastructure.

#### 2.1.1. About watchdog devices

Watchdog devices can be any of the following:

- Independently powered hardware devices
- Hardware devices that share power with the hosts they control
- Virtual devices implemented in software, or **softdog**

Hardware watchdog and **softdog** devices have electronic or software timers, respectively. These watchdog devices are used to ensure that the machine enters a safe state when an error condition is detected. The cluster is required to repeatedly reset the watchdog timer to prove that it is in a healthy state. This timer might elapse due to fault conditions, such as deadlocks, CPU starvation, and loss of network or disk access. If the timer expires, the watchdog device assumes that a fault has occurred and the device triggers a forced reset of the node.

Hardware watchdog devices are more reliable than **softdog** devices.

### 2.1.1.1. Understanding Self Node Remediation Operator behavior with watchdog devices

The Self Node Remediation Operator determines the remediation strategy based on the watchdog devices that are present.

If a hardware watchdog device is configured and available, the Operator uses it for remediation. If a hardware watchdog device is not configured, the Operator enables and uses a **softdog** device for remediation.

If neither watchdog devices are supported, either by the system or by the configuration, the Operator remediates nodes by using software reboot.

#### Additional resources

[Configuring a watchdog device for the virtual machine](#)

## 2.2. CONTROL PLANE FENCING

In earlier releases, you could enable Self Node Remediation and Node Health Check on worker nodes. In the event of node failure, you can now also follow remediation strategies on control plane nodes.

Self Node Remediation occurs in two primary scenarios.

- API Server Connectivity
  - In this scenario, the control plane node to be remediated is not isolated. It can be directly connected to the API Server, or it can be indirectly connected to the API Server through worker nodes or control-plane nodes, that are directly connected to the API Server.
  - When there is API Server Connectivity, the control plane node is remediated only if the Node Health Check Operator has created a **SelfNodeRemediation** custom resource (CR) for the node.
- No API Server Connectivity
  - In this scenario, the control plane node to be remediated is isolated from the API Server. The node cannot connect directly or indirectly to the API Server.
  - When there is no API Server Connectivity, the control plane node will be remediated as outlined with these steps:
    - Check the status of the control plane node with the majority of the peer worker nodes. If the majority of the peer worker nodes cannot be reached, the node will be analyzed further.
      - Self-diagnose the status of the control plane node
        - If self diagnostics passed, no action will be taken.
        - If self diagnostics failed, the node will be fenced and remediated.
        - The self diagnostics currently supported are checking the **kubelet** service status, and checking endpoint availability using **opt in** configuration.
      - If the node did not manage to communicate to most of its worker peers, check the connectivity of the control plane node with other control plane nodes. If the node can communicate with any other control plane peer, no action will be taken. Otherwise, the

node will be fenced and remediated.

## 2.3. INSTALLING THE SELF NODE REMEDIATION OPERATOR BY USING THE WEB CONSOLE

You can use the Red Hat OpenShift web console to install the Self Node Remediation Operator.



### NOTE

The Node Health Check Operator also installs the Self Node Remediation Operator as a default remediation provider.

### Prerequisites

- Log in as a user with **cluster-admin** privileges.

### Procedure

1. In the Red Hat OpenShift web console, navigate to **Operators** → **OperatorHub**.
2. Search for the Self Node Remediation Operator from the list of available Operators, and then click **Install**.
3. Keep the default selection of **Installation mode** and **namespace** to ensure that the Operator is installed to the **openshift-operators** namespace.
4. Click **Install**.

### Verification

To confirm that the installation is successful:

1. Navigate to the **Operators** → **Installed Operators** page.
2. Check that the Operator is installed in the **openshift-operators** namespace and its status is **Succeeded**.

If the Operator is not installed successfully:

1. Navigate to the **Operators** → **Installed Operators** page and inspect the **Status** column for any errors or failures.
2. Navigate to the **Workloads** → **Pods** page and check the logs of the **self-node-remediation-controller-manager** pod and **self-node-remediation-ds** pods in the **openshift-operators** project for any reported issues.

## 2.4. INSTALLING THE SELF NODE REMEDIATION OPERATOR BY USING THE CLI

You can use the OpenShift CLI (**oc**) to install the Self Node Remediation Operator.

You can install the Self Node Remediation Operator in your own namespace or in the **openshift-operators** namespace.

To install the Operator in your own namespace, follow the steps in the procedure.

To install the Operator in the **openshift-operators** namespace, skip to step 3 of the procedure because the steps to create a new **Namespace** custom resource (CR) and an **OperatorGroup** CR are not required.

## Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

## Procedure

1. Create a **Namespace** custom resource (CR) for the Self Node Remediation Operator:
  - a. Define the **Namespace** CR and save the YAML file, for example, **self-node-remediation-namespace.yaml**:

```
apiVersion: v1
kind: Namespace
metadata:
  name: self-node-remediation
```

- b. To create the **Namespace** CR, run the following command:

```
$ oc create -f self-node-remediation-namespace.yaml
```

2. Create an **OperatorGroup** CR:
  - a. Define the **OperatorGroup** CR and save the YAML file, for example, **self-node-remediation-operator-group.yaml**:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: self-node-remediation-operator
  namespace: self-node-remediation
```

- b. To create the **OperatorGroup** CR, run the following command:

```
$ oc create -f self-node-remediation-operator-group.yaml
```

3. Create a **Subscription** CR:
  - a. Define the **Subscription** CR and save the YAML file, for example, **self-node-remediation-subscription.yaml**:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: self-node-remediation-operator
  namespace: self-node-remediation 1
spec:
  channel: stable
  installPlanApproval: Manual 2
```

```
name: self-node-remediation-operator
source: redhat-operators
sourceNamespace: openshift-marketplace
package: self-node-remediation
```

- 1 Specify the **Namespace** where you want to install the Self Node Remediation Operator. To install the Self Node Remediation Operator in the **openshift-operators** namespace, specify **openshift-operators** in the **Subscription** CR.
- 2 Set the approval strategy to Manual in case your specified version is superseded by a later version in the catalog. This plan prevents an automatic upgrade to a later version and requires manual approval before the starting CSV can complete the installation.

- b. To create the **Subscription** CR, run the following command:

```
$ oc create -f self-node-remediation-subscription.yaml
```

## Verification

1. Verify that the installation succeeded by inspecting the CSV resource:

```
$ oc get csv -n self-node-remediation
```

### Example output

```
NAME                                DISPLAY                                VERSION  REPLACES  PHASE
self-node-remediation.v.0.6.0      Self Node Remediation Operator  v.0.6.0
Succeeded
```

2. Verify that the Self Node Remediation Operator is up and running:

```
$ oc get deploy -n self-node-remediation
```

### Example output

```
NAME                                READY  UP-TO-DATE  AVAILABLE  AGE
self-node-remediation-controller-manager  1/1    1            1          28h
```

3. Verify that the Self Node Remediation Operator created the **SelfNodeRemediationConfig** CR:

```
$ oc get selfnoderemediationconfig -n self-node-remediation
```

### Example output

```
NAME                                AGE
self-node-remediation-config  28h
```

4. Verify that each self node remediation pod is scheduled and running on each worker node and control plane node:

```
$ oc get daemonset -n self-node-remediation
```

## Example output

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE
SELECTOR	AGE					
self-node-remediation-ds	6	6	6	6	<none>	28h

## 2.5. CONFIGURING THE SELF NODE REMEDIATION OPERATOR

The Self Node Remediation Operator creates the **SelfNodeRemediationConfig** CR and the **SelfNodeRemediationTemplate** Custom Resource Definition (CRD).

### 2.5.1. Understanding the Self Node Remediation Operator configuration

The Self Node Remediation Operator creates the **SelfNodeRemediationConfig** CR with the name **self-node-remediation-config**. The CR is created in the namespace of the Self Node Remediation Operator.

A change in the **SelfNodeRemediationConfig** CR re-creates the Self Node Remediation daemon set.

The **SelfNodeRemediationConfig** CR resembles the following YAML file:

```
apiVersion: self-node-remediation.medik8s.io/v1alpha1
kind: SelfNodeRemediationConfig
metadata:
  name: self-node-remediation-config
  namespace: openshift-operators
spec:
  safeTimeToAssumeNodeRebootedSeconds: 180 1
  watchdogFilePath: /dev/watchdog 2
  isSoftwareRebootEnabled: true 3
  apiServerTimeout: 15s 4
  apiCheckInterval: 5s 5
  maxApiErrorThreshold: 3 6
  peerApiServerTimeout: 5s 7
  peerDialTimeout: 5s 8
  peerRequestTimeout: 5s 9
  peerUpdateInterval: 15m 10
```

1 Specify the time duration that the Operator waits before recovering affected workloads running on an unhealthy node. Starting replacement pods while they are still running on the failed node can lead to data corruption and a violation of run-once semantics. To prevent this from occurring, the Operator ignores values lower than a minimum calculated from the **ApiServerTimeout**, **ApiCheckInterval**, **maxApiErrorThreshold**, **peerDialTimeout**, and **peerRequestTimeout** fields.

2 Specify the file path of the watchdog device in the nodes. If you enter an incorrect path to the watchdog device, the Self Node Remediation Operator automatically detects the softdog device path.

If a watchdog device is unavailable, the **SelfNodeRemediationConfig** CR uses a software reboot.

3 Specify if you want to enable software reboot of the unhealthy nodes. By default, the value of **isSoftwareRebootEnabled** is set to **true**. To disable the software reboot, set the parameter value to **false**.



- 4 Specify the timeout duration to check connectivity with each API server. When this duration elapses, the Operator starts remediation. The timeout duration must be greater than or equal to 10
- 5 Specify the frequency to check connectivity with each API server. The timeout duration must be greater than or equal to 1 second.
- 6 Specify a threshold value. After reaching this threshold, the node starts contacting its peers. The threshold value must be greater than or equal to 1 second.
- 7 Specify the duration of the timeout for the peer to connect the API server. The timeout duration must be greater than or equal to 10 milliseconds.
- 8 Specify the duration of the timeout for establishing connection with the peer. The timeout duration must be greater than or equal to 10 milliseconds.
- 9 Specify the duration of the timeout to get a response from the peer. The timeout duration must be greater than or equal to 10 milliseconds.
- 10 Specify the frequency to update peer information such as IP address. The timeout duration must be greater than or equal to 10 seconds.



## NOTE

You can edit the **self-node-remediation-config** CR that is created by the Self Node Remediation Operator. However, when you try to create a new CR for the Self Node Remediation Operator, the following message is displayed in the logs:

```
controllers.SelfNodeRemediationConfig
ignoring selfnoderemediationconfig CRs that are not named 'self-node-remediation-
config'
or not in the namespace of the operator:
'openshift-operators' {"selfnoderemediationconfig":
"openshift-operators/selfnoderemediationconfig-copy"}
```

## 2.5.2. Understanding the Self Node Remediation Template configuration

The Self Node Remediation Operator also creates the **SelfNodeRemediationTemplate** Custom Resource Definition (CRD). This CRD defines the remediation strategy for the nodes. The following remediation strategies are available:

### ResourceDeletion

This remediation strategy removes the pods and associated volume attachments on the node, rather than the removal of the node object. This strategy recovers workloads faster. **ResourceDeletion** is the default remediation strategy.

### OutOfServiceTaint

This remediation strategy implicitly causes the removal of the pods and associated volume attachments on the node, rather than the removal of the node object. It achieves this by placing the **OutOfServiceTaint** strategy, a feature supported since Kubernetes version 1.25, on the node. This strategy recovers workloads faster.

The Self Node Remediation Operator creates the **SelfNodeRemediationTemplate** CR for the strategy **self-node-remediation-resource-deletion-template**, which the **ResourceDeletion** remediation strategy uses.

The **SelfNodeRemediationTemplate** CR resembles the following YAML file:

```
apiVersion: self-node-remediation.medik8s.io/v1alpha1
kind: SelfNodeRemediationTemplate
metadata:
  creationTimestamp: "2022-03-02T08:02:40Z"
  name: self-node-remediation-<remediation_object>-deletion-template 1
  namespace: openshift-operators
spec:
  template:
    spec:
      remediationStrategy: <remediation_strategy> 2
```

1 Specifies the type of remediation template based on the remediation strategy. Replace **<remediation\_object>** with either **resource** or **node**; for example, **self-node-remediation-resource-deletion-template**.

2 Specifies the remediation strategy. The remediation strategy is **ResourceDeletion**.

## 2.5.3. Troubleshooting the Self Node Remediation Operator

### 2.5.3.1. General troubleshooting

#### Issue

You want to troubleshoot issues with the Self Node Remediation Operator.

#### Resolution

Check the Operator logs.

### 2.5.3.2. Checking the daemon set

#### Issue

The Self Node Remediation Operator is installed but the daemon set is not available.

#### Resolution

Check the Operator logs for errors or warnings.

### 2.5.3.3. Unsuccessful remediation

#### Issue

An unhealthy node was not remediated.

#### Resolution

Verify that the **SelfNodeRemediation** CR was created by running the following command:

```
$ oc get snr -A
```

If the **MachineHealthCheck** controller did not create the **SelfNodeRemediation** CR when the node turned unhealthy, check the logs of the **MachineHealthCheck** controller. Additionally, ensure that the **MachineHealthCheck** CR includes the required specification to use the remediation template.

If the **SelfNodeRemediation** CR was created, ensure that its name matches the unhealthy node or the machine object.

#### 2.5.3.4. Daemon set and other Self Node Remediation Operator resources exist even after uninstalling the Operator

##### Issue

The Self Node Remediation Operator resources, such as the daemon set, configuration CR, and the remediation template CR, exist even after after uninstalling the Operator.

##### Resolution

To remove the Self Node Remediation Operator resources, delete the resources by running the following commands for each resource type:

```
$ oc delete ds <self-node-remediation-ds> -n <namespace>
```

```
$ oc delete snrc <self-node-remediation-config> -n <namespace>
```

```
$ oc delete snrt <self-node-remediation-template> -n <namespace>
```

#### 2.5.4. Gathering data about the Self Node Remediation Operator

To collect debugging information about the Self Node Remediation Operator, use the **must-gather** tool. For information about the **must-gather** image for the Self Node Remediation Operator, see [Gathering data about specific features](#).

#### 2.5.5. Additional resources

- [Using Operator Lifecycle Manager on restricted networks](#) .
- [Deleting Operators from a cluster](#)

## CHAPTER 3. REMEDIATING NODES WITH MACHINE HEALTH CHECKS

Machine health checks automatically repair unhealthy machines in a particular machine pool.

### 3.1. ABOUT MACHINE HEALTH CHECKS



#### NOTE

You can only apply a machine health check to control plane machines on clusters that use control plane machine sets.

To monitor machine health, create a resource to define the configuration for a controller. Set a condition to check, such as staying in the **NotReady** status for five minutes or displaying a permanent condition in the node-problem-detector, and a label for the set of machines to monitor.

The controller that observes a **MachineHealthCheck** resource checks for the defined condition. If a machine fails the health check, the machine is automatically deleted and one is created to take its place. When a machine is deleted, you see a **machine deleted** event.

To limit disruptive impact of the machine deletion, the controller drains and deletes only one node at a time. If there are more unhealthy machines than the **maxUnhealthy** threshold allows for in the targeted pool of machines, remediation stops and therefore enables manual intervention.



#### NOTE

Consider the timeouts carefully, accounting for workloads and requirements.

- Long timeouts can result in long periods of downtime for the workload on the unhealthy machine.
- Too short timeouts can result in a remediation loop. For example, the timeout for checking the **NotReady** status must be long enough to allow the machine to complete the startup process.

To stop the check, remove the resource.

#### 3.1.1. Limitations when deploying machine health checks

There are limitations to consider before deploying a machine health check:

- Only machines owned by a machine set are remediated by a machine health check.
- If the node for a machine is removed from the cluster, a machine health check considers the machine to be unhealthy and remediates it immediately.
- If the corresponding node for a machine does not join the cluster after the **nodeStartupTimeout**, the machine is remediated.
- A machine is remediated immediately if the **Machine** resource phase is **Failed**.

## 3.2. CONFIGURING MACHINE HEALTH CHECKS TO USE THE SELF NODE REMEDIATION OPERATOR

Use the following procedure to configure the worker or control-plane machine health checks to use the Self Node Remediation Operator as a remediation provider.



### NOTE

To use the Self Node Remediation Operator as a remediation provider for machine health checks, a machine must have an associated node in the cluster.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

### Procedure

1. Create a **SelfNodeRemediationTemplate** CR:

- a. Define the **SelfNodeRemediationTemplate** CR:

```
apiVersion: self-node-remediation.medik8s.io/v1alpha1
kind: SelfNodeRemediationTemplate
metadata:
  namespace: openshift-machine-api
  name: selfnoderemediationtemplate-sample
spec:
  template:
    spec:
      remediationStrategy: ResourceDeletion 1
```

- 1** Specifies the remediation strategy. The default strategy is **ResourceDeletion**.

- b. To create the **SelfNodeRemediationTemplate** CR, run the following command:

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

2. Create or update the **MachineHealthCheck** CR to point to the **SelfNodeRemediationTemplate** CR:

- a. Define or update the **MachineHealthCheck** CR:

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: machine-health-check
  namespace: openshift-machine-api
spec:
  selector:
    matchLabels: 1
      machine.openshift.io/cluster-api-machine-role: "worker"
      machine.openshift.io/cluster-api-machine-type: "worker"
```

```
unhealthyConditions:  
- type: "Ready"  
  timeout: "300s"  
  status: "False"  
- type: "Ready"  
  timeout: "300s"  
  status: "Unknown"  
maxUnhealthy: "40%"  
nodeStartupTimeout: "10m"  
remediationTemplate: 2  
  kind: SelfNodeRemediationTemplate  
  apiVersion: self-node-remediation.medik8s.io/v1alpha1  
  name: selfnoderemediationtemplate-sample
```

- 1 Selects whether the machine health check is for **worker** or **control-plane** nodes. The label can also be user-defined.
- 2 Specifies the details for the remediation template.

b. To create a **MachineHealthCheck** CR, run the following command:

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

c. To update a **MachineHealthCheck** CR, run the following command:

```
$ oc apply -f <mhc-name>.yaml
```

## CHAPTER 4. REMEDIATING NODES WITH NODE HEALTH CHECKS

You can use the Node Health Check Operator to identify unhealthy nodes. The Operator uses the Self Node Remediation Operator to remediate the unhealthy nodes.

For more information on the Self Node Remediation Operator, see the [Using Self Node Remediation](#) chapter.

### 4.1. ABOUT THE NODE HEALTH CHECK OPERATOR

The Node Health Check Operator detects the health of the nodes in a cluster. The **NodeHealthCheck** controller creates the **NodeHealthCheck** custom resource (CR), which defines a set of criteria and thresholds to determine the health of a node.

The Node Health Check Operator also installs the Self Node Remediation Operator as a default remediation provider.

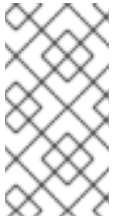
When the Node Health Check Operator detects an unhealthy node, it creates a remediation CR that triggers the remediation provider. For example, the controller creates the **SelfNodeRemediation** CR, which triggers the Self Node Remediation Operator to remediate the unhealthy node.

The **NodeHealthCheck** CR resembles the following YAML file, with self-node-remediation as the remediation provider:

```
apiVersion: remediation.medik8s.io/v1alpha1
kind: NodeHealthCheck
metadata:
  name: nodehealthcheck-sample
spec:
  minHealthy: 51% 1
  pauseRequests: 2
  - <pause-test-cluster>
  remediationTemplate: 3
    apiVersion: self-node-remediation.medik8s.io/v1alpha1
    name: self-node-remediation-resource-deletion-template
    namespace: openshift-operators
    kind: SelfNodeRemediationTemplate
  escalatingRemediations: 4
  - remediationTemplate:
    apiVersion: self-node-remediation.medik8s.io/v1alpha1
    name: self-node-remediation-resource-deletion-template
    namespace: openshift-operators
    kind: SelfNodeRemediationTemplate
  order: 1
  timeout: 300s
  selector: 5
  matchExpressions:
  - key: node-role.kubernetes.io/worker
    operator: Exists
  unhealthyConditions: 6
  - type: Ready
    status: "False"
    duration: 300s 7
```

- type: Ready  
 status: Unknown  
 duration: 300s **8**

- 1 Specifies the amount of healthy nodes (in percentage or number) required for a remediation provider to concurrently remediate nodes in the targeted pool. If the number of healthy nodes equals to or exceeds the limit set by **minHealthy**, remediation occurs. The default value is 51%.
- 2 Prevents any new remediation from starting, while allowing any ongoing remediations to persist. The default value is empty. However, you can enter an array of strings that identify the cause of pausing the remediation. For example, **pause-test-cluster**.



#### NOTE

During the upgrade process, nodes in the cluster might become temporarily unavailable and get identified as unhealthy. In the case of worker nodes, when the Operator detects that the cluster is upgrading, it stops remediating new unhealthy nodes to prevent such nodes from rebooting.

- 3 Specifies a remediation template from the remediation provider. For example, from the Self Node Remediation Operator. **remediationTemplate** is mutually exclusive with **escalatingRemediations**.
- 4 Specifies a list of **RemediationTemplates** with order and timeout fields. To obtain a healthy node, use this field to sequence and configure multiple remediations. This strategy increases the likelihood of obtaining a healthy node, instead of depending on a single remediation that might not be successful. The **order** field determines the order in which the remediations are invoked (lower order = earlier invocation). The **timeout** field determines when the next remediation is invoked. **escalatingRemediations** is mutually exclusive with **remediationTemplate**.
- 5 Specifies a selector that matches labels or expressions that you want to check. Avoid selecting both control-plane and worker nodes in one CR.
- 6 Specifies a list of the conditions that determine whether a node is considered unhealthy.
- 7 **8** Specifies the timeout duration for a node condition. If a condition is met for the duration of the timeout, the node will be remediated. Long timeouts can result in long periods of downtime for a workload on an unhealthy node.

The **NodeHealthCheck** CR resembles the following YAML file, with metal3 as the remediation provider:

```
apiVersion: remediation.medik8s.io/v1alpha1
kind: NodeHealthCheck
metadata:
  name: nhc-worker-metal3
spec:
  minHealthy: 30%
  remediationTemplate:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3RemediationTemplate
    name: metal3-remediation
    namespace: openshift-machine-api
  selector:
    matchExpressions:
      - key: node-role.kubernetes.io/worker
```



```

operator: Exists
unhealthyConditions:
- duration: 300s
  status: 'False'
  type: Ready
- duration: 300s
  status: 'Unknown'
  type: Ready

```



## NOTE

The **matchExpressions** are examples only; you must map your machine groups based on your specific needs.

The **Metal3RemediationTemplate** resembles the following YAML file, with metal3 as the remediation provider:

```

apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
kind: Metal3RemediationTemplate
metadata:
  name: metal3-remediation
  namespace: openshift-machine-api
spec:
  template:
    spec:
      strategy:
        retryLimit: 1
        timeout: 5m0s
        type: Reboot

```



## NOTE

In addition to creating a **NodeHealthCheck** CR, you must also create the **Metal3RemediationTemplate**.

### 4.1.1. Understanding the Node Health Check Operator workflow

When a node is identified as unhealthy, the Node Health Check Operator checks how many other nodes are unhealthy. If the number of healthy nodes exceeds the amount that is specified in the **minHealthy** field of the **NodeHealthCheck** CR, the controller creates a remediation CR from the details that are provided in the external remediation template by the remediation provider. After remediation, the kubelet updates the node's health status.

When the node turns healthy, the controller deletes the external remediation template.

### 4.1.2. About how node health checks prevent conflicts with machine health checks

When both, node health checks and machine health checks are deployed, the node health check avoids conflict with the machine health check.



## NOTE

Red Hat OpenShift deploys **machine-api-termination-handler** as the default **MachineHealthCheck** resource.

The following list summarizes the system behavior when node health checks and machine health checks are deployed:

- If only the default machine health check exists, the node health check continues to identify unhealthy nodes. However, the node health check ignores unhealthy nodes in a Terminating state. The default machine health check handles the unhealthy nodes with a Terminating state.

### Example log message

```
INFO MHCChecker ignoring unhealthy Node, it is terminating and will be handled by MHC
{"nodeName": "node-1.example.com"}
```

- If the default machine health check is modified (for example, the **unhealthyConditions** is **Ready**), or if additional machine health checks are created, the node health check is disabled.

### Example log message

```
INFO controllers.NodeHealthCheck disabling NHC in order to avoid conflict with custom
MHCs configured in the cluster {"NodeHealthCheck": "/nhc-worker-default"}
```

- When, again, only the default machine health check exists, the node health check is re-enabled.

### Example log message

```
INFO controllers.NodeHealthCheck re-enabling NHC, no conflicting MHC configured in the
cluster {"NodeHealthCheck": "/nhc-worker-default"}
```

## 4.2. CONTROL PLANE FENCING

In earlier releases, you could enable Self Node Remediation and Node Health Check on worker nodes. In the event of node failure, you can now also follow remediation strategies on control plane nodes.

Do not use the same **NodeHealthCheck** CR for worker nodes and control plane nodes. Grouping worker nodes and control plane nodes together can result in incorrect evaluation of the minimum healthy node count, and cause unexpected or missing remediations. This is because of the way the Node Health Check Operator handles control plane nodes. You should group the control plane nodes in their own group and the worker nodes in their own group. If required, you can also create multiple groups of worker nodes.

Considerations for remediation strategies:

- Avoid Node Health Check configurations that involve multiple configurations overlapping the same nodes because they can result in unexpected behavior. This suggestion applies to both worker and control plane nodes.
- The Node Health Check Operator implements a hardcoded limitation of remediating a maximum of one control plane node at a time. Multiple control plane nodes should not be remediated at the same time.

## 4.3. INSTALLING THE NODE HEALTH CHECK OPERATOR BY USING THE WEB CONSOLE

You can use the Red Hat OpenShift web console to install the Node Health Check Operator.

### Prerequisites

- Log in as a user with **cluster-admin** privileges.

### Procedure

1. In the Red Hat OpenShift web console, navigate to **Operators → OperatorHub**.
2. Search for the Node Health Check Operator, then click **Install**.
3. Keep the default selection of **Installation mode** and **namespace** to ensure that the Operator will be installed to the **openshift-operators** namespace.
4. Ensure that the **Console plug-in** is set to **Enable**.
5. Click **Install**.

### Verification

To confirm that the installation is successful:

1. Navigate to the **Operators → Installed Operators** page.
2. Check that the Operator is installed in the **openshift-operators** namespace and that its status is **Succeeded**.

If the Operator is not installed successfully:

1. Navigate to the **Operators → Installed Operators** page and inspect the **Status** column for any errors or failures.
2. Navigate to the **Workloads → Pods** page and check the logs in any pods in the **openshift-operators** project that are reporting issues.

## 4.4. INSTALLING THE NODE HEALTH CHECK OPERATOR BY USING THE CLI

You can use the OpenShift CLI (**oc**) to install the Node Health Check Operator.

To install the Operator in your own namespace, follow the steps in the procedure.

To install the Operator in the **openshift-operators** namespace, skip to step 3 of the procedure because the steps to create a new **Namespace** custom resource (CR) and an **OperatorGroup** CR are not required.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

## Procedure

1. Create a **Namespace** custom resource (CR) for the Node Health Check Operator:
  - a. Define the **Namespace** CR and save the YAML file, for example, **node-health-check-namespace.yaml**:

```
apiVersion: v1
kind: Namespace
metadata:
  name: node-health-check
```

- b. To create the **Namespace** CR, run the following command:

```
$ oc create -f node-health-check-namespace.yaml
```

2. Create an **OperatorGroup** CR:

- a. Define the **OperatorGroup** CR and save the YAML file, for example, **node-health-check-operator-group.yaml**:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: node-health-check-operator
  namespace: node-health-check
```

- b. To create the **OperatorGroup** CR, run the following command:

```
$ oc create -f node-health-check-operator-group.yaml
```

3. Create a **Subscription** CR:

- a. Define the **Subscription** CR and save the YAML file, for example, **node-health-check-subscription.yaml**:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: node-health-check-operator
  namespace: node-health-check 1
spec:
  channel: stable 2
  installPlanApproval: Manual 3
  name: node-healthcheck-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  package: node-healthcheck-operator
```

**1** Specify the **Namespace** where you want to install the Node Health Check Operator. To install the Node Health Check Operator in the **openshift-operators** namespace, specify **openshift-operators** in the **Subscription** CR.

**2**

Specify the channel name for your subscription. To upgrade to the latest version of the Node Health Check Operator, you must manually change the channel name for

- 3 Set the approval strategy to Manual in case your specified version is superseded by a later version in the catalog. This plan prevents an automatic upgrade to a later version and requires manual approval before the starting CSV can complete the installation.

- b. To create the **Subscription** CR, run the following command:

```
$ oc create -f node-health-check-subscription.yaml
```

## Verification

1. Verify that the installation succeeded by inspecting the CSV resource:

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

### Example output

```
NAME                                DISPLAY                               VERSION REPLACES PHASE
node-healthcheck-operator.v0.5.0. Node Health Check Operator 0.5.0      Succeeded
```

2. Verify that the Node Health Check Operator is up and running:

```
$ oc get deploy -n openshift-operators
```

### Example output

```
NAME                                READY UP-TO-DATE AVAILABLE AGE
node-health-check-operator-controller-manager 1/1    1            1      10d
```

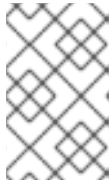
## 4.5. CREATING A NODE HEALTH CHECK

Using the web console, you can create a node health check to identify unhealthy nodes and specify the remediation type and strategy to fix them.

### Procedure

1. From the **Administrator** perspective of the Red Hat OpenShift web console, click **Compute** → **NodeHealthChecks** → **CreateNodeHealthCheck**.
2. Specify whether to configure the node health check using the **Form view** or the **YAML view**.
3. Enter a **Name** for the node health check. The name must consist of lower case, alphanumeric characters, '-' or '.', and must start and end with an alphanumeric character.
4. Specify the **Remediator** type, and **Self node remediation** or **Other**. The Self node remediation option is part of the Self Node Remediation Operator that is installed with the Node Health Check Operator. Selecting **Other** requires an **API version**, **Kind**, **Name**, and **Namespace** to be entered, which then points to the remediation template resource of a remediator.
5. Make a **Nodes** selection by specifying the labels of the nodes you want to remediate. The

selection matches labels that you want to check. If more than one label is specified, the nodes must contain each label. The default value is empty, which selects both worker and control-plane nodes.



#### NOTE

When creating a node health check with the Self Node Remediation Operator, you must select either **node-role.kubernetes.io/worker** or **node-role.kubernetes.io/control-plane** as the value.

6. Specify the minimum number of healthy nodes, using either a percentage or a number, required for a **NodeHealthCheck** to remediate nodes in the targeted pool. If the number of healthy nodes equals to or exceeds the limit set by **Min healthy**, remediation occurs. The default value is 51%.
7. Specify a list of **Unhealthy conditions** that if a node meets determines whether the node is considered unhealthy, and requires remediation. You can specify the **Type**, **Status** and **Duration**. You can also create your own custom type.
8. Click **Create** to create the node health check.

#### Verification

- Navigate to the **Compute** → **NodeHealthCheck** page and verify that the corresponding node health check is listed, and their status displayed. Once created, node health checks can be paused, modified, and deleted.

## 4.6. GATHERING DATA ABOUT THE NODE HEALTH CHECK OPERATOR

To collect debugging information about the Node Health Check Operator, use the **must-gather** tool. For information about the **must-gather** image for the Node Health Check Operator, see [Gathering data about specific features](#).

## 4.7. ADDITIONAL RESOURCES

- [Changing the update channel for an Operator](#)
- [Using Operator Lifecycle Manager on restricted networks](#).

## CHAPTER 5. PLACING NODES IN MAINTENANCE MODE WITH NODE MAINTENANCE OPERATOR

You can use the Node Maintenance Operator to place nodes in maintenance mode by using the **oc adm** utility or **NodeMaintenance** custom resources (CRs).

### 5.1. ABOUT THE NODE MAINTENANCE OPERATOR

The Node Maintenance Operator watches for new or deleted **NodeMaintenance** CRs. When a new **NodeMaintenance** CR is detected, no new workloads are scheduled and the node is cordoned off from the rest of the cluster. All pods that can be evicted are evicted from the node. When a **NodeMaintenance** CR is deleted, the node that is referenced in the CR is made available for new workloads.



#### NOTE

Using a **NodeMaintenance** CR for node maintenance tasks achieves the same results as the **oc adm cordon** and **oc adm drain** commands using standard Red Hat OpenShift CR processing.

### 5.2. INSTALLING THE NODE MAINTENANCE OPERATOR

You can install the Node Maintenance Operator using the web console or the OpenShift CLI (**oc**).



#### NOTE

If OpenShift Virtualization version 4.10 or less is installed in your cluster, it includes an outdated version of the Node Maintenance Operator.

#### 5.2.1. Installing the Node Maintenance Operator by using the web console

You can use the Red Hat OpenShift web console to install the Node Maintenance Operator.

##### Prerequisites

- Log in as a user with **cluster-admin** privileges.

##### Procedure

1. In the Red Hat OpenShift web console, navigate to **Operators** → **OperatorHub**.
2. Search for the Node Maintenance Operator, then click **Install**.
3. Keep the default selection of **Installation mode** and **namespace** to ensure that the Operator will be installed to the **openshift-operators** namespace.
4. Click **Install**.

##### Verification

To confirm that the installation is successful:

1. Navigate to the **Operators** → **Installed Operators** page.

2. Check that the Operator is installed in the **openshift-operators** namespace and that its status is **Succeeded**.

If the Operator is not installed successfully:

1. Navigate to the **Operators → Installed Operators** page and inspect the **Status** column for any errors or failures.
2. Navigate to the **Operators → Installed Operators → Node Maintenance Operator → Details** page, and inspect the **Conditions** section for errors before pod creation.
3. Navigate to the **Workloads → Pods** page, search for the **Node Maintenance Operator** pod in the installed namespace, and check the logs in the **Logs** tab.

## 5.2.2. Installing the Node Maintenance Operator by using the CLI

You can use the OpenShift CLI (**oc**) to install the Node Maintenance Operator.

You can install the Node Maintenance Operator in your own namespace or in the **openshift-operators** namespace.

To install the Operator in your own namespace, follow the steps in the procedure.

To install the Operator in the **openshift-operators** namespace, skip to step 3 of the procedure because the steps to create a new **Namespace** custom resource (CR) and an **OperatorGroup** CR are not required.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

### Procedure

1. Create a **Namespace** CR for the Node Maintenance Operator:
  - a. Define the **Namespace** CR and save the YAML file, for example, **node-maintenance-namespace.yaml**:

```
apiVersion: v1
kind: Namespace
metadata:
  name: nmo-test
```

- b. To create the **Namespace** CR, run the following command:

```
$ oc create -f node-maintenance-namespace.yaml
```

2. Create an **OperatorGroup** CR:
  - a. Define the **OperatorGroup** CR and save the YAML file, for example, **node-maintenance-operator-group.yaml**:

```
apiVersion: operators.coreos.com/v1
```



```
kind: OperatorGroup
metadata:
  name: node-maintenance-operator
  namespace: nmo-test
```

- b. To create the **OperatorGroup** CR, run the following command:

```
$ oc create -f node-maintenance-operator-group.yaml
```

3. Create a **Subscription** CR:

- a. Define the **Subscription** CR and save the YAML file, for example, **node-maintenance-subscription.yaml**:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: node-maintenance-operator
  namespace: nmo-test 1
spec:
  channel: stable
  installPlanApproval: Automatic
  name: node-maintenance-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  package: node-maintenance-operator
```

- 1** Specify the **Namespace** where you want to install the Node Maintenance Operator.



### IMPORTANT

To install the Node Maintenance Operator in the **openshift-operators** namespace, specify **openshift-operators** in the **Subscription** CR.

- b. To create the **Subscription** CR, run the following command:

```
$ oc create -f node-maintenance-subscription.yaml
```

### Verification

1. Verify that the installation succeeded by inspecting the CSV resource:

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

#### Example output

```
NAME                                DISPLAY                VERSION  REPLACES  PHASE
node-maintenance-operator.v5.0.0    Node Maintenance Operator  5.0.0    Succeeded
```

2. Verify that the Node Maintenance Operator is running:

```
$ oc get deploy -n openshift-operators
```

■

### Example output

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
node-maintenance-operator-controller-manager	1/1	1	1	10d

The Node Maintenance Operator is supported in a restricted network environment. For more information, see [Using Operator Lifecycle Manager on restricted networks](#).

## 5.3. SETTING A NODE TO MAINTENANCE MODE

You can place a node into maintenance mode from the web console or from the CLI by using a **NodeMaintenance** CR.

### 5.3.1. Setting a node to maintenance mode by using the web console

To set a node to maintenance mode, you can create a **NodeMaintenance** custom resource (CR) by using the web console.

#### Prerequisites

- Log in as a user with **cluster-admin** privileges.
- Install the Node Maintenance Operator from the **OperatorHub**.

#### Procedure

1. From the **Administrator** perspective in the web console, navigate to **Operators → Installed Operators**.
2. Select the Node Maintenance Operator from the list of Operators.
3. In the **Node Maintenance** tab, click **Create NodeMaintenance**.
4. In the **Create NodeMaintenance** page, select the **Form view** or the **YAML view** to configure the **NodeMaintenance** CR.
5. To apply the **NodeMaintenance** CR that you have configured, click **Create**.

#### Verification

In the **Node Maintenance** tab, inspect the **Status** column and verify that its status is **Succeeded**.

### 5.3.2. Setting a node to maintenance mode by using the CLI

You can put a node into maintenance mode with a **NodeMaintenance** custom resource (CR). When you apply a **NodeMaintenance** CR, all allowed pods are evicted and the node is rendered unschedulable. Evicted pods are queued to be moved to another node in the cluster.

#### Prerequisites

- Install the Red Hat OpenShift CLI **oc**.
- Log in to the cluster as a user with **cluster-admin** privileges.

## Procedure

1. Create the following **NodeMaintenance** CR, and save the file as **nodemaintenance-cr.yaml**:

```
apiVersion: nodemaintenance.medik8s.io/v1beta1
kind: NodeMaintenance
metadata:
  name: nodemaintenance-cr 1
spec:
  nodeName: node-1.example.com 2
  reason: "NIC replacement" 3
```

- 1 The name of the node maintenance CR.
- 2 The name of the node to be put into maintenance mode.
- 3 A plain text description of the reason for maintenance.

2. Apply the node maintenance CR by running the following command:

```
$ oc apply -f nodemaintenance-cr.yaml
```

## Verification

1. Check the progress of the maintenance task by running the following command:

```
$ oc describe node <node-name>
```

where **<node-name>** is the name of your node; for example, **node-1.example.com**

2. Check the example output:

```
Events:
  Type    Reason             Age          From          Message
  ----    -
  Normal  NodeNotSchedulable  61m         kubelet       Node node-1.example.com
status is now: NodeNotSchedulable
```

### 5.3.3. Checking status of current NodeMaintenance CR tasks

You can check the status of current **NodeMaintenance** CR tasks.

#### Prerequisites

- Install the Red Hat OpenShift CLI **oc**.
- Log in as a user with **cluster-admin** privileges.

#### Procedure

- Check the status of current node maintenance tasks, for example the **NodeMaintenance** CR or **nm** object, by running the following command:

```
$ oc get nm -o yaml
```

### Example output

```
apiVersion: v1
items:
- apiVersion: nodemaintenance.medik8s.io/v1beta1
  kind: NodeMaintenance
  metadata:
  ...
  spec:
    nodeName: node-1.example.com
    reason: Node maintenance
  status:
    drainProgress: 100 1
    evictionPods: 3 2
    lastError: "Last failure message" 3
    lastUpdate: "2022-06-23T11:43:18Z" 4
    phase: Succeeded
    totalPods: 5 5
  ...
```

- 1** The percentage completion of draining the node.
- 2** The number of pods scheduled for eviction.
- 3** The latest eviction error, if any.
- 4** The last time the status was updated.
- 5** The total number of pods before the node entered maintenance mode.

## 5.4. RESUMING A NODE FROM MAINTENANCE MODE

You can resume a node from maintenance mode from the web console or from the CLI by using a **NodeMaintenance** CR. Resuming a node brings it out of maintenance mode and makes it schedulable again.


### 5.4.1. Resuming a node from maintenance mode by using the web console

To resume a node from maintenance mode, you can delete a **NodeMaintenance** custom resource (CR) by using the web console.

#### Prerequisites

- Log in as a user with **cluster-admin** privileges.
- Install the Node Maintenance Operator from the **OperatorHub**.

#### Procedure

1. From the **Administrator** perspective in the web console, navigate to **Operators → Installed Operators**.
2. Select the Node Maintenance Operator from the list of Operators.
3. In the **Node Maintenance** tab, select the **NodeMaintenance** CR that you want to delete.
4. Click the Options menu  at the end of the node and select **Delete NodeMaintenance**.

### Verification

1. In the Red Hat OpenShift console, click **Compute → Nodes**.
2. Inspect the **Status** column of the node for which you deleted the **NodeMaintenance** CR and verify that its status is **Ready**.

## 5.4.2. Resuming a node from maintenance mode by using the CLI

You can resume a node from maintenance mode that was initiated with a **NodeMaintenance** CR by deleting the **NodeMaintenance** CR.

### Prerequisites

- Install the Red Hat OpenShift CLI **oc**.
- Log in to the cluster as a user with **cluster-admin** privileges.

### Procedure

- When your node maintenance task is complete, delete the active **NodeMaintenance** CR:

```
$ oc delete -f nodemaintenance-cr.yaml
```

### Example output

```
nodemaintenance.nodemaintenance.medik8s.io "maintenance-example" deleted
```

### Verification

1. Check the progress of the maintenance task by running the following command:

```
$ oc describe node <node-name>
```

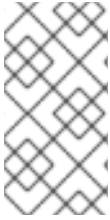
where **<node-name>** is the name of your node; for example, **node-1.example.com**

2. Check the example output:

```
Events:
  Type    Reason             Age          From    Message
  ----    -
  Normal  NodeSchedulable   2m          kubelet Node node-1.example.com status
is now: NodeSchedulable
```

## 5.5. WORKING WITH BARE-METAL NODES

For clusters with bare-metal nodes, you can place a node into maintenance mode, and resume a node from maintenance mode, by using the web console **Actions** control.



### NOTE


Clusters with bare-metal nodes can also place a node into maintenance mode, and resume a node from maintenance mode, by using the web console and CLI, as outlined. These methods, by using the web console **Actions** control, are applicable to bare-metal clusters only.

### 5.5.1. Maintaining bare-metal nodes

When you deploy Red Hat OpenShift on bare-metal infrastructure, you must take additional considerations into account compared to deploying on cloud infrastructure. Unlike in cloud environments, where the cluster nodes are considered ephemeral, reprovisioning a bare-metal node requires significantly more time and effort for maintenance tasks.


When a bare-metal node fails due to a kernel error or a NIC card hardware failure, workloads on the failed node need to be restarted on another node in the cluster while the problem node is repaired or replaced. Node maintenance mode allows cluster administrators to gracefully turn-off nodes, move workloads to other parts of the cluster, and ensure that workloads do not get interrupted. Detailed progress and node status details are provided during maintenance.

### 5.5.2. Setting a bare-metal node to maintenance mode

Set a bare-metal node to maintenance mode using the Options menu  found on each node in the **Compute** → **Nodes** list, or using the **Actions** control of the **Node Details** screen.

#### Procedure

1. From the **Administrator** perspective of the web console, click **Compute** → **Nodes**.
2. You can set the node to maintenance from this screen, which makes it easier to perform actions on multiple nodes, or from the **Node Details** screen, where you can view comprehensive details of the selected node:

- Click the Options menu  at the end of the node and select **Start Maintenance**.
- Click the node name to open the **Node Details** screen and click **Actions** → **Start Maintenance**.


3. Click **Start Maintenance** in the confirmation window.

The node is no longer schedulable. If it had virtual machines with the **LiveMigration** eviction strategy, then it will live migrate them. All other pods and virtual machines on the node are deleted and recreated on another node.

#### Verification


- Navigate to the **Compute** → **Nodes** page and verify that the corresponding node has a status of **Under maintenance**.

### 5.5.3. Resuming a bare-metal node from maintenance mode

Resume a bare-metal node from maintenance mode using the Options menu  found on each node in the **Compute** → **Nodes** list, or using the **Actions** control of the **Node Details** screen.

#### Procedure

1. From the **Administrator** perspective of the web console, click **Compute** → **Nodes**.
2. You can resume the node from this screen, which makes it easier to perform actions on multiple nodes, or from the **Node Details** screen, where you can view comprehensive details of the selected node:

- Click the Options menu  at the end of the node and select **Stop Maintenance**.
- Click the node name to open the **Node Details** screen and click **Actions** → **Stop Maintenance**.

3. Click **Stop Maintenance** in the confirmation window.

The node becomes schedulable. If it had virtual machine instances that were running on the node prior to maintenance, then they will not automatically migrate back to this node.

#### Verification

- Navigate to the **Compute** → **Nodes** page and verify that the corresponding node has a status of **Ready**.

## 5.6. GATHERING DATA ABOUT THE NODE MAINTENANCE OPERATOR

To collect debugging information about the Node Maintenance Operator, use the **must-gather** tool. For information about the **must-gather** image for the Node Maintenance Operator, see [Gathering data about specific features](#).

## 5.7. ADDITIONAL RESOURCES

- [Gathering data about your cluster](#)
- [Understanding how to evacuate pods on nodes](#)
- [Understanding how to mark nodes as unschedulable or schedulable](#)