



# Workload Availability for Red Hat OpenShift 23.2

修復、フェンシング、およびメンテナンス

ワークロードの可用性の修復、フェンシング、およびメンテナンス



## Workload Availability for Red Hat OpenShift 23.2 修復、フェンシング、およびメンテナンス

---

ワークロードの可用性の修復、フェンシング、およびメンテナンス

## 法律上の通知

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.

## 概要

ワークロード可用性演算子とその使用方法に関する情報

## 目次

はじめに .....	3
WORKLOAD AVAILABILITY FOR RED HAT OPENSIFT に関するフィードバック .....	4
第1章 ノードの修復、フェンシング、およびメンテナンスについて .....	5
1.1. 自己ノード修復	5
1.2. MACHINE HEALTH CHECK (マシンのヘルスチェック)	5
1.3. ノードのヘルスチェック	5
1.4. ノードのメンテナンス	5
第2章 自己ノード修復の使用 .....	7
2.1. SELF NODE REMEDIATION OPERATOR について	7
2.2. コントロールプレーンフェンシング	8
2.3. WEB コンソールを使用した SELF NODE REMEDIATION OPERATOR のインストール	9
2.4. CLI を使用した SELF NODE REMEDIATION OPERATOR のインストール	9
2.5. SELF NODE REMEDIATION OPERATOR の設定	12
第3章 マシンヘルスチェックを使用したノードの修復 .....	16
3.1. マシンのヘルスチェック	16
3.2. SELF NODE REMEDIATION OPERATOR を使用するためのマシンヘルスチェックの設定	17
第4章 ノードヘルスチェックを使用したノードの修復 .....	19
4.1. NODE HEALTH CHECK OPERATOR について	19
4.2. コントロールプレーンフェンシング	22
4.3. WEB コンソールを使用したノードヘルスチェックオペレーターのインストール	23
4.4. CLI を使用した NODE HEALTH CHECK OPERATOR のインストール	23
4.5. ノードヘルスチェックの作成	25
4.6. NODE HEALTH CHECK OPERATOR に関するデータの収集	26
4.7. 関連情報	26
第5章 NODE MAINTENANCE OPERATOR を使用してノードをメンテナンスモードにする .....	27
5.1. NODE MAINTENANCE OPERATOR について	27
5.2. NODE MAINTENANCE OPERATOR のインストール	27
5.3. ノードのメンテナンスモードへの設定	30
5.4. メンテナンスモードからのノードの再開	32
5.5. ベアメタルノードの操作	34
5.6. NODE MAINTENANCE OPERATOR に関するデータの収集	35
5.7. 関連情報	35



# はじめに

## WORKLOAD AVAILABILITY FOR RED HAT OPENSIFT に関するフィードバック

Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。これを行うには、以下を行います。

1. [JIRA](#) の Web サイトにアクセスします。
2. **Summary** フィールドにわかりやすいタイトルを入力します。
3. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
4. **Reporter** フィールドにユーザー名を入力します。
5. **Affects Version/s** フィールドに、影響を受けるバージョンを入力します。
6. ダイアログの下部にある **Create** をクリックします。

# 第1章 ノードの修復、フェンシング、およびメンテナンスについて

ハードウェアは不完全であり、ソフトウェアにはバグが含まれています。カーネルのハングやネットワークインターフェイスコントローラー (NIC) の障害などのノードレベルの障害が発生した場合、クラスターから必要な作業は減少せず、影響を受けるノードからのワークロードをどこかで再起動する必要があります。ただし、ReadWriteOnce (RWO) ボリュームや StatefulSet などの一部のワークロードでは、最大で1つのセマンティクスが必要になる場合があります。

これらのワークロードに影響を与える障害は、データの損失、破損、またはその両方のリスクを伴います。ワークロードの回復 (**remediation** と理想的にはノードの回復とも呼ばれます) を開始する前に、**fencing** と呼ばれる安全な状態にノードが達していることを確認することが重要です。

ノードとワークロードの true のステータスを確認するために管理者の介入に依存することは、必ずしも現実的ではありません。このような介入を容易にするために、Red Hat OpenShift は、障害検出、フェンシング、および修復を自動化するための複数のコンポーネントを提供します。

## 1.1. 自己ノード修復

Self Node Remediation Operator は Red Hat OpenShift アドオンオペレーターであり、異常なノードを再起動し、Pod や VolumeAttachments などのリソースを削除するフェンシングと修復の外部システムを実装します。再起動により、ワークロードが隔離され、リソースの削除により、影響を受けるワークロードの再スケジュールが加速されます。他の外部システムとは異なり、自己ノード修復には、Intelligent Platform Management Interface (IPMI) やノードプロビジョニング用の API などの管理インターフェイスは必要ありません。

セルフノード修復は、マシンヘルスチェックやノードヘルスチェックなどの障害検出システムで使用できます。

## 1.2. MACHINE HEALTH CHECK (マシンのヘルスチェック)

Machine Health Check は、マシンのステータスとノードの状態をモニターする Red Hat OpenShift ビルトインの失敗検出、フェンシング、修復システムを利用します。マシンヘルスチェックは、セルフノード修復などの外部フェンシングおよび修復システムをトリガーするように設定できます。

## 1.3. ノードのヘルスチェック

Node Health Check Operator は、ノードの状態をモニターする障害検出システムを実装する Red Hat OpenShift アドオンオペレーターです。フェンシングまたは修復システムが組み込まれていないため、そのような機能を提供する外部システムで設定する必要があります。デフォルトでは、セルフノード修復システムを利用するように設定されています。

## 1.4. ノードのメンテナンス

管理者は、ドライブ、RAM、または NIC を交換するなど、クラスターを中断する必要がある状況に直面します。

このメンテナンスの前に、影響を受けるノードを封鎖して排水する必要があります。ノードが封鎖されると、そのノードで新しいワークロードをスケジュールできなくなります。ノードがドレインされると、ダウンタイムを回避または最小化するために、影響を受けるノードのワークロードが他のノードに転送されます。

このメンテナンスはコマンドラインツールを使用して実行できますが、Node Maintenance Operator は、カスタムリソースを使用してこれを達成するための宣言的なアプローチを提供します。そのようなリソースがノードに存在する場合、オペレーターは、リソースが削除されるまでノードを遮断してドレ

インします。

## 第2章 自己ノード修復の使用

Self Node Remediation Operator を使用して、異常なノードを自動的に再起動できます。この修復戦略は、ステートフルアプリケーションと ReadWriteOnce(RWO) ボリュームのダウンタイムを最小限に抑え、一時的な障害が発生した場合に計算能力を回復します。

### 2.1. SELF NODE REMEDIATION OPERATOR について

Self Node Remediation Operator はクラスターノードで実行され、正常でないと特定されるノードを再起動します。Operator は、**MachineHealthCheck** または **NodeHealthCheck** コントローラーを使用して、クラスター内のノードの正常性を検出します。ノードが異常であると識別されると、**MachineHealthCheck** または **NodeHealthCheck** リソースが **SelfNodeRemediation** カスタムリソース (CR) を作成し、Self NodeRemediationOperator をトリガーします。

**SelfNodeRemediation** CR は、次の YAML ファイルに似ています。

```
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 ノードの修復ストラテジーを指定します。

② 修復中に発生した最後のエラーを表示します。修復が正常に実行されるか、エラーが発生しない場合は、このフィールドは空になります。

Self Node Remediation Operator は、ステートフルアプリケーションのダウンタイムを最小限に抑え、一時的な障害が発生した場合に計算能力を回復します。この Operator は、IPMI や API などの管理インターフェイスに関係なくノードをプロビジョニングするために使用できます。また、クラスターのインストールタイプ (インストーラーでプロビジョニングされたインフラストラクチャーやユーザーでプロビジョニングされたインフラストラクチャーなど) に関係なく使用できます。

#### 2.1.1. ウォッチドッグデバイスについて

ウォッチドッグデバイスは、次のいずれかになります。

- 電源が独立しているハードウェアデバイス
- 制御するホストと電源を共有するハードウェアデバイス
- ソフトウェアまたは **softdog** に実装された仮想デバイス

ハードウェアウォッチドッグデバイスと **softdog** デバイスには、それぞれ電子タイマーまたはソフトウェアタイマーがあります。これらのウォッチドッグデバイスは、エラー状態が検出されたときにマシンが安全な状態になるようにするために使用されます。クラスターは、ウォッチドッグタイマーを繰り返しリセットして、正常な状態にあることを証明する必要があります。このタイマーは、デッドロック、CPU の枯渇、ネットワークまたはディスクアクセスの喪失などの障害状態が原因で経過する可能性があります。タイマーが時間切れになると、ウォッチドッグデバイスは障害が発生したと見なし、デバイスがノードの強制リセットをトリガーします。

ハードウェアウォッチドッグデバイスは、**softdog** デバイスよりも信頼性があります。

### 2.1.1.1. ウォッチドッグデバイスを使用した Self Node Remediation Operator の動作の理解

Self Node Remediation Operator は、存在するウォッチドッグデバイスに基づいて修復戦略を決定します。

ハードウェアウォッチドッグデバイスが設定されて使用可能である場合、Operator はそれを修復に使用します。ハードウェアウォッチドッグデバイスが設定されていない場合、Operator は修復のために **softdog** デバイスを有効にして使用します。

システムまたは設定のどちらかで、いずれのウォッチドッグデバイスもサポートされていない場合、Operator はソフトウェアの再起動を使用してノードを修復します。

## 関連情報

[仮想マシンのウォッチドッグデバイスの設定](#)

## 2.2. コントロールプレーンフェンシング

以前のリリースでは、ワーカーノードでセルフノード修復とノードヘルスチェックを有効にすることができました。ノード障害が発生した場合、コントロールプレーンノードの修復戦略に従うこともできるようになりました。

自己ノード修復は、主に 2 つのシナリオで発生します。

- API サーバー接続
  - このシナリオでは、修復されるコントロールプレーンノードは分離されていません。API サーバーに直接接続することも、API サーバーに直接接続されているワーカーノードまたはコントロールプレーンノードを介して API サーバーに間接的に接続することもできます。
  - API サーバー接続がある場合、コントロールプレーンノードは、Node Health Check Operator がノードの **SelfNodeRemediation** カスタムリソース (CR) を作成した場合にのみ修復されます。
- API サーバー接続なし
  - このシナリオでは、修復されるコントロールプレーンノードは API サーバーから分離されています。ノードは API サーバーに直接または間接的に接続できません。
  - API サーバー接続がない場合、コントロールプレーンノードは次の手順で説明されているように修正されます。
    - ピアワーカーノードの大部分を含むコントロールプレーンノードのステータスを確認します。ピアワーカーノードの大部分に到達できない場合、ノードはさらに分析されます。
      - コントロールプレーンノードのステータスを自己診断する
        - 自己診断に合格した場合、アクションは実行されません。
        - 自己診断に失敗した場合、ノードは隔離され、修復されます。
        - 現在サポートされている自己診断では、**kubelet** サービスのステータスをチェックし、**opt in** 設定を使用してエンドポイントの可用性をチェックしています。

- ノードがほとんどのワーカーピアと通信できなかった場合は、コントロールプレーンノードと他のコントロールプレーンノードとの接続を確認します。ノードが他のコントロールプレーンピアと通信できる場合、アクションは実行されません。それ以外の場合、ノードは隔離され、修正されます。

## 2.3. WEB コンソールを使用した SELF NODE REMEDIATION OPERATOR のインストール

Red Hat OpenShift Web コンソールを使用して、Self Node Remediation Operator をインストールできます。



### 注記

Node Health Check Operator は、Self Node Remediation Operator をデフォルトの修復プロバイダーとしてインストールします。

### 前提条件

- **cluster-admin** 権限を持つユーザーとしてログインしている。

### 手順

1. Red Hat OpenShift Web コンソールで、**Operators** → **OperatorHub** に移動します。
2. 使用可能なオペレーターのリストから Self Node Remediation Operator を検索し、**Install** をクリックします。
3. Operator が **openshift-operators** namespace にインストールされるように、**Installation mode** と **namespace** のデフォルトの選択を維持します。
4. **Install** をクリックします。

### 検証

インストールが正常に行われたことを確認するには、以下を実行します。

1. **Operators** → **Installed Operators** ページに移動します。
2. Operator が **openshift-operators** の namespace に設置されていることと、その状態が **Succeeded** になっていることを確認してください。

Operator が正常にインストールされていない場合、以下を実行します。

1. **Operators** → **Installed Operators** ページに移動し、**Status** 列でエラーまたは失敗の有無を確認します。
2. **Workloads** → **Pod** ページに移動し、**openshift-operators** プロジェクト内の **self-node-remediation-controller-manager** Pod および **self-node-remediation-ds** Pod のログで、報告された問題がないか確認します。

## 2.4. CLI を使用した SELF NODE REMEDIATION OPERATOR のインストール

OpenShift CLI (**oc**) を使用して、Self Node Remediation Operator をインストールできます。

Self Node Remediation Operator は、独自の namespace または **openshift-operators** namespace にインストールできます。

独自の namespace に Operator をインストールするには、手順に従います。

**openshift-operators** namespace に Operator をインストールするには、手順の 3 にスキップします。これは、新しい **Namespace** カスタムリソース (CR) と **OperatorGroup** CR を作成する必要がないためです。

## 前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

## 手順

1. Self Node Remediation Operator の **Namespace** カスタムリソース (CR) を作成します。

a. **Namespace** CR を定義し、YAML ファイルを保存します (例: **self-node-remediation-namespace.yaml**)。

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

b. **Namespace** CR を作成するには、次のコマンドを実行します。

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

2. **OperatorGroup** を作成します。

a. **OperatorGroup** CR を定義し、YAML ファイルを保存します (例: **self-node-remediation-operator-group.yaml**)。

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

b. **OperatorGroup** CR を作成するには、次のコマンドを実行します。

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

3. **Subscription** CR を作成します。

a. **Subscription** CR を定義し、YAML ファイルを保存します (例: **self-node-remediation-subscription.yaml**)。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: self-node-remediation-operator
```

```

namespace: self-node-remediation ❶
spec:
  channel: stable
  installPlanApproval: Manual ❷
  name: self-node-remediation-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  package: self-node-remediation

```

- ❶ Self Node Remediation Operator をインストールする **Namespace** を指定します。セルフノード修復 Operator を **openshift-operators** namespace にインストールするには、**Subscription** CR で **openshift-operators** を指定します。
- ❷ 指定したバージョンがカタログの新しいバージョンに置き換えられる場合に備えて、承認ストラテジーを Manual に設定します。これにより、新しいバージョンへの自動アップグレードが阻止され、最初の CSV のインストールが完了する前に手動での承認が必要となります。

b. **Subscription**CR を作成するには、次のコマンドを実行します。

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

## 検証

1. CSV リソースを調べて、インストールが成功したことを確認します。

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

出力例

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

2. Self Node Remediation Operator が稼働していることを確認します。

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

出力例

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

3. Self Node Remediation Operator が **SelfNodeRemediationConfig** CR を作成していることを確認します。

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

出力例

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

- 各セルフノード修復 Pod がスケジュールされ、各ワーカーノードとコントロールプレーンノードで実行されていることを確認します。

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

出力例

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

## 2.5. SELF NODE REMEDIATION OPERATOR の設定

Self Node Remediation Operator は、**SelfNodeRemediationConfig** CR と **SelfNodeRemediationTemplate** カスタムリソース定義 (CRD) を作成します。

### 2.5.1. Self Node Remediation Operator 設定について

Self Node Remediation Operator は、**self-node-remediation-config** という名前の **SelfNodeRemediationConfig** CR を作成します。CR は Self Node Remediation Operator の namespace に作成されます。

**SelfNodeRemediationConfig** CR の変更により、Self Node Remediation デーモンセットが再作成されます。

**SelfNodeRemediationConfig** CR は以下の YAML ファイルのようになります。

```
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
```

- Operator が、正常でないノードで実行中の影響を受けるワークロードを復元するまで待機する期間を指定します。障害が発生したノードで実行中の代替 Pod を開始すると、データの破損や 1 回実行セマンティクスの違反が生じる可能性があります。これを防ぐために、Operator は、**ApiServerTimeout**、**ApiCheckInterval**、**maxApiErrorThreshold**、**peerDialTimeout**、および **peerRequestTimeout** フィールドから計算された最小値よりも小さい値を無視します。

- 2 ノード内のウォッチドッグデバイスのファイルパスを指定します。ウォッチドッグデバイスへの誤ったパスを入力すると、Self Node Remediation Operator がソフトドッグデバイスのパスを自動ウォッチドッグデバイスが使用できない場合、**SelfNodeRemediationConfig** CR はソフトウェアの再起動を使用します。
- 3 異常なノードのソフトウェア再起動を有効にするかどうかを指定します。デフォルトでは、**is Software Reboot Enabled** の値は **true** に設定されています。ソフトウェアの再起動を無効にするには、パラメーター値を **false** に設定します。
- 4 各 API サーバーとの接続を確認するためのタイムアウト期間を指定します。この期間が経過すると、Operator は修復を開始します。タイムアウト期間は 10 ミリ秒以上である必要があります。
- 5 各 API サーバーとの接続を確認する頻度を指定します。タイムアウト期間は 1 秒以上である必要があります。
- 6 しきい値を指定します。このしきい値に達した後、ノードはピアへの接続を開始します。しきい値は 1 秒以上である必要があります。
- 7 ピアが API サーバーに接続するためのタイムアウトの期間を指定します。タイムアウト期間は 10 ミリ秒以上である必要があります。
- 8 ピアで接続を確立するためのタイムアウトの期間を指定します。タイムアウト期間は 10 ミリ秒以上である必要があります。
- 9 ピアから応答を取得するためのタイムアウトの期間を指定します。タイムアウト期間は 10 ミリ秒以上である必要があります。
- 10 IP アドレスなどのピア情報を更新する頻度を指定します。タイムアウト期間は 10 秒以上である必要があります。

### 注記

Self NodeRemediationOperator によって作成された **self-node-remediation-config** CR を編集できます。ただし、Self Node Remediation Operator の新しい CR を作成しようとすると、次のメッセージがログに表示されます。

```

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. 自己ノード修復テンプレートの設定を理解する

Self Node Remediation Operator は、**SelfNodeRemediationTemplate** カスタムリソース定義 (CRD) も作成します。この CRD は、ノードの修復ストラテジーを定義します。次の修復戦略が利用可能です。

### ResourceDeletion

この修復戦略では、ノードオブジェクトではなく、ノード上の Pod と関連するボリュームアタッチメントが削除されます。この戦略により、ワークロードがより迅速に回復されます。**ResourceDeletion** は、デフォルトの修復戦略です。

## OutOfServiceTaint

この修復戦略により、ノードオブジェクトではなく、ノード上の Pod および関連するボリュームアタッチメントが暗黙的に削除されます。これは、**OutOfServiceTaint** 戦略をノードに配置することで実現します。この戦略により、ワークロードがより迅速に回復されます。この戦略は、OpenShift Container Platform バージョン 4.13 以降、および OpenShift Container Platform バージョン 4.15 以降、一般公開(GA)でサポートされます。

Self Node Remediation Operator は、**ResourceDeletion** 修復戦略が使用する戦略 **self-node-remediation-resource-deletion-template** の **SelfNodeRemediationTemplate** CR を作成します。

**SelfNodeRemediationTemplate** CR は以下の YAML ファイルのようになります。

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

- ❶ 修復戦略に基づいて修復テンプレートのタイプを指定します。<remediation\_object> をリソース または **node** のいずれかに置き換えます (例: **self-node-remediation-resource-deletion-template**)。
- ❷ 修復戦略を指定します。修復戦略は **ResourceDeletion** です。

## 2.5.3. Self Node Remediation Operator のトラブルシューティング

### 2.5.3.1. 一般的なトラブルシューティング

#### 問題

Self Node Remediation Operator の問題のトラブルシューティングが必要です。

#### 解決方法

Operator ログを確認してください。

### 2.5.3.2. デモンセットの確認

#### 問題

Self Node Remediation Operator はインストールされていますが、デモンセットはインストールされません。

#### 解決方法

エラーまたは警告がないか、オペレーターログを確認してください。

### 2.5.3.3. 失敗した修復

#### 問題

不健康なノードは修正されませんでした。

#### 解決方法

以下のコマンドを実行して、**SelfNodeRemediation** CR が作成されていることを確認します。

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

**MachineHealthCheck** コントローラーがノードが正常でない状態で **SelfNodeRemediation** CR を作成しなかった場合、**MachineHealthCheck** コントローラーのログを確認します。さらに、**MachineHealthCheck** CR に、修復テンプレートを使用するために必要な仕様が含まれていることを確認してください。

**SelfNodeRemediation** CR が作成される場合、その名前が正常でないノードまたはマシンオブジェクトと一致することを確認します。

#### 2.5.3.4. Operator をアンインストールした後でも、デーモンセットおよびその他の Self Node Remediation Operator リソースが存在する

##### 問題

デーモンセット、設定 CR、修復テンプレート CR などの Self Node Remediation Operator リソースは、Operator をアンインストールした後も存在します。

##### 解決方法

Self Node Remediation Operator リソースを削除するには、リソースタイプごとに次のコマンドを実行してリソースを削除します。

```
$ 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. Self Node Remediation Operator に関するデータの収集

Self Node Remediation Operator に関するデバッグ情報を収集するには、**must-gather** ツールを使用します。Self Node Remediation Operator の **must-gather** イメージは、**特定の機能に関するデータの収集** を参照してください。

#### 2.5.5. 関連情報

- [ネットワークが制限された環境での Operator Lifecycle Manager の使用](#)
- [クラスターからの Operator の削除](#)

## 第3章 マシンヘルスチェックを使用したノードの修復

マシンのヘルスチェックは特定のマシンプールの正常ではないマシンを自動的に修復します。

### 3.1. マシンのヘルスチェック



#### 注記

マシンのヘルスチェックは、コントロールプレーンマシンセットを使用するクラスター上のコントロールプレーンマシンにのみ適用できます。

マシンの正常性を監視するには、リソースを作成し、コントローラーの設定を定義します。5分間 **NotReady** ステータスにすることや、`node-problem-detector` に永続的な条件を表示すること、および監視する一連のマシンのラベルなど、チェックする条件を設定します。

**MachineHealthCheck** リソースを監視するコントローラーは定義済みのステータスをチェックします。マシンがヘルスチェックに失敗した場合、このマシンは自動的に検出され、その代替りとなるマシンが作成されます。マシンが削除されると、**machine deleted** イベントが表示されます。

マシンの削除による破壊的な影響を制限するために、コントローラーは1度に1つのノードのみをドレイン (解放) し、これを削除します。マシンのターゲットプールで許可される **maxUnhealthy** しきい値を上回る数の正常でないマシンがある場合、修復が停止するため、手動による介入が可能になります。



#### 注記

タイムアウトについて注意深い検討が必要であり、ワークロードと要件を考慮してください。

- タイムアウトの時間が長くなると、正常でないマシンのワークロードのダウンタイムが長くなる可能性があります。
- タイムアウトが短すぎると、修復ループが生じる可能性があります。たとえば、**NotReady** ステータスを確認するためのタイムアウトについては、マシンが起動プロセスを完了できるように十分な時間を設定する必要があります。

チェックを停止するには、リソースを削除します。

#### 3.1.1. マシンヘルスチェックのデプロイ時の制限

マシンヘルスチェックをデプロイする前に考慮すべき制限事項があります。

- マシンセットが所有するマシンのみがマシンヘルスチェックによって修復されます。
- マシンのノードがクラスターから削除される場合、マシンヘルスチェックはマシンが正常ではないとみなし、すぐにこれを修復します。
- **nodeStartupTimeout** の後にマシンの対応するノードがクラスターに加わらない場合、マシンは修復されます。
- **Machine** リソースフェーズが **Failed** の場合、マシンはすぐに修復されます。

## 3.2. SELF NODE REMEDIATION OPERATOR を使用するためのマシンヘルスチェックの設定

次の手順を使用して、Self Node Remediation Operator を修復プロバイダーとして使用するようによりワーカーまたはコントロールプレーンマシンのヘルスチェックを設定します。



### 注記

Self Node Remediation Operator をマシンの健全性チェックの修復プロバイダーとして使用するには、マシンに、クラスター内に関連付けられたノードが配置されている必要があります。

### 前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

### 手順

1. **SelfNodeRemediationTemplate** CR を作成します。

- a. **SelfNodeRemediationTemplate** CR を定義します。

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

- ❶ 修復ストラテジーを指定します。デフォルトのストラテジーは **ResourceDeletion** です。

- b. **SelfNodeRemediationTemplate** CR を作成するには、以下のコマンドを実行します。

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

2. **MachineHealthCheck** CR を作成し、**SelfNodeRemediationTemplate** CR を参照するよう更新します。

- a. **MachineHealthCheck** を定義または更新します。

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: machine-health-check
  namespace: openshift-machine-api
spec:
  selector:
    matchLabels: ❶
```

```
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 マシンのヘルスチェックの対象が **worker** ノードか **control-plane** ノードかを選択します。ラベルはユーザー定義にすることもできます。

2 修復テンプレートの詳細を指定します。

b. **MachineHealthCheck** CR を作成するには、次のコマンドを実行します。

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

c. **MachineHealthCheck** CR を更新するには、次のコマンドを実行します。

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

## 第4章 ノードヘルスチェックを使用したノードの修復

Node Health Check Operator を使用して、不健全なノードを特定できます。Operator は、Self Node Remediation Operator を使用して、不健全なノードを修復します。

Self Node Remediation Operator の詳細は、[自己ノード修復の使用](#) の章を参照してください。



### 注記

Red Hat OpenShift Service on AWS (ROSA) クラスタにプリインストールされたマシンヘルスチェックが存在するため、Node Health Check Operator はこのような環境では機能しません。

### 4.1. NODE HEALTH CHECK OPERATOR について

Node Health Check Operator は、クラスタ内のノードの健全性を検出します。**NodeHealthCheck** コントローラーは、**NodeHealthCheck** カスタムリソース (CR) を作成します。これは、ノードの状態を判断するための一連の基準としきい値を定義します。

Node Health Check Operator は、Self Node Remediation Operator をデフォルトの修復プロバイダーとしてインストールします。

Node Health Check Operator は異常なノードを検出すると、修復プロバイダーをトリガーする修復 CR を作成します。たとえば、コントローラーは **SelfNodeRemediation** CR を作成し、Self Node Remediation Operator をトリガーして正常でないノードを修復します。

**NodeHealthCheck** CR は、修復プロバイダーとして自己ノード修復を使用した次の YAML ファイルに似ています。

```
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** 修復プロバイダーがターゲットプール内のノードを同時に修復するために必要な正常なノードの数 (パーセンテージまたは数) を指定します。正常なノードの数が **minHealthy** で設定された制限以上の場合、修復が行われます。デフォルト値は 51% です。
- 2** 新しい修復が開始されないようにし、進行中の修復を継続できるようにします。デフォルト値は空です。ただし、修復を一時停止する原因を特定する文字列の配列を入力できます。たとえば、**pause-test-cluster**。



#### 注記

アップグレードプロセス中に、クラスター内のノードが一時的に使用できなくなり、異常として識別される場合があります。ワーカーノードの場合、オペレーターはクラスターがアップグレード中であることを検出すると、新しい異常なノードの修正を停止して、そのようなノードが再起動しないようにします。

- 3** 修復プロバイダーからの修復テンプレートを指定します。たとえば、Self Node Remediation Operator のようになります。 **remediationTemplate** は **escalatingRemediation** と相互排他的です。
- 4** 順序フィールドとタイムアウトフィールドを含む **RemediationTemplate** のリストを指定します。正常なノードを取得するには、このフィールドを使用して複数の修復を順序付けし、設定します。この戦略により、成功しない可能性のある単一の修復に依存するのではなく、正常なノードを取得できる可能性が高まります。 **order** フィールドは、修復が呼び出される順序を決定します (低い順序 = 早い呼び出し)。 **timeout** フィールドは、次の修復がいつ呼び出されるかを決定します。 **escalatingRemediation** は **remediationTemplate** と相互排他的です。
- 5** チェックするラベルまたは式に一致する **selector** を指定します。1つの CR でコントロールプレーンノードとワーカーノードの両方を選択しないでください。
- 6** ノードが異常と見なされるかどうかを決定する条件のリストを指定します。
- 7** **8** ノード条件のタイムアウト期間を指定します。タイムアウトの期間中に条件が満たされた場合、ノードは修正されます。タイムアウトが長いと、異常なノードのワークロードで長期間のダウンタイムが発生する可能性があります。

**NodeHealthCheck** CR は、修復プロバイダーとして **metal3** を使用した、次の YAML ファイルに似ています。

```
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

```



#### 注記

**matchExpressions** は例です。特定のニーズに基づいてマシングループをマッピングする必要があります。

**Metal3RemediationTemplate** は、修復プロバイダーとして metal3 を使用した、次の YAML ファイルに似ています。

```

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

```



#### 注記

**NodeHealthCheck** CR の作成に加えて、**Metal3RemediationTemplate** も作成する必要があります。

### 4.1.1. Node Health Check Operator のワークフローを理解する

ノードが異常であると識別されると、Node Health Check Operator は他にいくつかのノードが異常であるかをチェックします。健康なノードの数が **NodeHealthCheck** CR の **minHealthy** フィールドで指定された量を超えた場合、コントローラーは、修復プロバイダーによって外部の修復テンプレートで提供される詳細から修復 CR を作成します。修復後、kubelet はノードのヘルスステータスを更新します。

ノードが正常になると、コントローラーは外部修復テンプレートを削除します。

### 4.1.2. ノードのヘルスチェックによるマシンヘルスチェックの競合

ノードヘルスチェックとマシンヘルスチェックの両方がデプロイメントされている場合、ノードヘルスチェックはマシンヘルスチェックとの競合を回避します。



#### 注記

Red Hat OpenShift は **machine-api-termination-handler** をデフォルトの **MachineHealthCheck** リソースとしてデプロイします。

次のリストは、ノードヘルスチェックとマシンヘルスチェックがデプロイメントされたときのシステムの動作をまとめたものです。

- デフォルトのマシンヘルスチェックのみが存在する場合、ノードヘルスチェックは引き続き異常なノードを識別します。ただし、ノードヘルスチェックは、Terminating 状態の異常なノードを無視します。デフォルトのマシンヘルスチェックは、異常なノードを Terminating 状態で処理します。

#### ログメッセージの例

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

- デフォルトのマシンヘルスチェックが変更された場合 (たとえば、**unhealthyConditions** が **Ready** の場合)、または追加のマシンヘルスチェックが作成された場合、ノードヘルスチェックは無効になります。

#### ログメッセージの例

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

- ここでも、デフォルトのマシンヘルスチェックのみが存在する場合、ノードヘルスチェックが再度有効になります。

#### ログメッセージの例

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

## 4.2. コントロールプレーンフェンシング

以前のリリースでは、ワーカーノードでセルフノード修復とノードヘルスチェックを有効にすることができました。ノード障害が発生した場合、コントロールプレーンノードの修復戦略に従うこともできるようになりました。

ワーカーノードとコントロールプレーンノードに同じ **NodeHealthCheck** CR を使用しないでください。ワーカーノードとコントロールプレーンノードと一緒にグループ化すると、正常なノードの最小数が正しく評価されず、予期しない修復または欠落した修復が発生する可能性があります。これは、Node Health Check Operator がコントロールプレーンノードを処理する方法が原因です。コントロールプレーンノードを独自のグループにグループ化し、ワーカーノードを独自のグループにグループ化する必要があります。必要に応じて、複数のワーカーノードグループを作成することもできます。

修復戦略に関する考慮事項:

- 予期しない動作が発生する可能性があるため、同じノードに重複する複数の設定を含むノードヘルスチェック設定は避けてください。この提案は、ワーカープレーンノードとコントロールプレーンノードの両方に適用されます。
- Node Health Check Operator は、一度に最大1つのコントロールプレーンノードを修正するというハードコーディングされた制限を実装します。複数のコントロールプレーンノードを同時に修復しないでください。

### 4.3. WEB コンソールを使用したノードヘルスチェックオペレーターのインストール

Red Hat OpenShift Web コンソールを使用して、Node Health Check Operator をインストールできます。

#### 前提条件

- **cluster-admin** 権限を持つユーザーとしてログインしている。

#### 手順

1. Red Hat OpenShift Web コンソールで、**Operators** → **OperatorHub** に移動します。
2. Node Health Check Operator を検索し、**Install**をクリックします。
3. Operator が **openshift-operators** namespace にインストールされるように、**Installation mode** と **namespace** のデフォルトの選択を維持します。
4. **Console plug-in** が **Enable** に設定されていることを確認します。
5. **Install** をクリックします。

#### 検証

インストールが正常に行われたことを確認するには、以下を実行します。

1. **Operators** → **Installed Operators** ページに移動します。
2. Operator が **openshift-operators** の namespace 内に設置されていることと、その状態が **Succeeded** となっていることを確認してください。

Operator が正常にインストールされていない場合、以下を実行します。

1. **Operators** → **Installed Operators** ページに移動し、**Status** 列でエラーまたは失敗の有無を確認します。
2. **Workloads** → **Pods** ページにナビゲートし、問題を報告している **openshift-operators** プロジェクトの Pod のログを確認します。

### 4.4. CLI を使用した NODE HEALTH CHECK OPERATOR のインストール

OpenShift CLI(**oc**)を使用して、Node Health Check Operator をインストールできます。

独自の namespace に Operator をインストールするには、手順に従います。

**openshift-operators** namespace に Operator をインストールするには、手順の 3 にスキップします。これは、新しい **Namespace** カスタムリソース (CR) と **OperatorGroup** CR を作成する必要がないためです。

## 前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

## 手順

1. Node Health Check Operator の **Namespace** カスタムリソース (CR) を作成します。

- a. **Namespace** CR を定義し、YAML ファイルを保存します (例: **node-health-check-namespace.yaml**)。

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

- b. **Namespace** CR を作成するには、次のコマンドを実行します。

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

2. **OperatorGroup** を作成します。

- a. **OperatorGroup** CR を定義し、YAML ファイルを保存します (例: **node-health-check-operator-group.yaml**)。

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

- b. **OperatorGroup** CR を作成するには、次のコマンドを実行します。

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

3. **Subscription** CR を作成します。

- a. **Subscription** CR を定義し、YAML ファイルを保存します (例: **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 Node Health Check Operator をインストールする **Namespace** を指定します。Node Health Check Operator を **openshift-operators** namespace にインストールするには、**Subscription** CR で **openshift-operators** を指定します。
- 2 サブスクリプションのチャンネル名を指定します。Node Health Check Operator の最新バージョンにアップグレードするには、サブスクリプションのチャンネル名を **candidate** から **stable** に手動で変更する必要があります。
- 3 指定したバージョンがカタログの新しいバージョンに置き換えられる場合に備えて、承認ストラテジーを Manual に設定します。これにより、新しいバージョンへの自動アップグレードが阻止され、最初の CSV のインストールが完了する前に手動での承認が必要となります。

b. **Subscription**CR を作成するには、次のコマンドを実行します。

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

## 検証

1. CSV リソースを調べて、インストールが成功したことを確認します。

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

### 出力例

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

2. Node Health CheckOperator が稼働していることを確認します。

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

### 出力例

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

## 4.5. ノードヘルスチェックの作成

Web コンソールを使用して、ノードヘルスチェックを作成して異常なノードを特定し、修正する修復タイプとストラテジーを指定できます。

### 手順

1. Red Hat OpenShift Web コンソールの **Administrator** の観点から、**Compute** → **NodeHealthChecks** → **CreateNodeHealthCheck** をクリックします。

2. **Form ビュー** または **YAML ビュー** を使用してノードのヘルスチェックを設定するかどうかを指定します。
3. ノードヘルスチェックの **名前** を入力します。名前は小文字、英数字、 '-'、または..で設定され、英数字で開始および終了する必要があります。
4. **Remediator** タイプ、および **Self node remediation** または **Other** を指定します。Self node remediation オプションは、Node Health Check Operator でインストールされる Self Node Remediation Operator の一部です。**Other** を選択するには、**API バージョン**、**Kind**、**Name**、および **Namespace** を入力する必要があります。これは、修正の修復テンプレートリソースを指します。
5. 修復する **ノード** のラベルを指定して、ノードを選択します。選択した内容は、確認するラベルと一致します。複数のラベルを指定する場合、ノードには各ラベルが含まれている必要があります。デフォルト値は空で、ワーカーノードとコントロールプレーンノードの両方を選択します。



#### 注記

Self Node Remediation Operator を使用してノードヘルスチェックを作成する場合、**node-role.kubernetes.io/worker** または **node-role.kubernetes.io/control-plane** のいずれかを値として選択する必要があります。

6. **NodeHealthCheck** がターゲットプール内のノードを修復するために必要な、正常なノードの最小数をパーセンテージまたは数値で指定します。正常なノードの数が **Min healthy** によって設定された制限以上の場合には、修復が行われます。デフォルト値は 51% です。
7. ノードが一致した場合に、ノードが異常であると見なされ、修復が必要かどうかを決定する、**異常な条件** のリストを指定します。**Type**、**Status**、および **Duration** を指定できます。独自のカスタムタイプを作成することもできます。
8. **Create** をクリックしてノードヘルスチェックを作成します。

#### 検証

- **Compute** → **NodeHealthCheck** ページに移動し、対応するノードヘルスチェックが一覧表示され、それらのステータスが表示されることを確認します。作成が完了すると、ノードヘルスチェックを一時停止、変更、および削除できます。

## 4.6. NODE HEALTH CHECK OPERATOR に関するデータの収集

Node Health Check Operator に関するデバッグ情報を収集するには、**must-gather** ツールを使用します。Node Health Check Operator の must-gather イメージについては、**特定の機能に関するデータの収集** を参照してください。

## 4.7. 関連情報

- [Operator の更新チャネルの変更](#)
- [ネットワークが制限された環境での Operator Lifecycle Manager の使用](#)

## 第5章 NODE MAINTENANCE OPERATOR を使用してノードをメンテナンスモードにする

**oc adm** ユーティリティまたは **NodeMaintenance** カスタムリソース (CR) を使用して、Node Maintenance Operator を使用してノードをメンテナンスモードにすることができます。

### 5.1. NODE MAINTENANCE OPERATOR について

Node Maintenance Operator は、新規または削除された **NodeMaintenance** CR をモニタリングします。新規の **NodeMaintenance** CR が検出されると、新規ワークロードはスケジュールされず、ノードは残りのクラスターから遮断されます。エビクトできるすべての Pod はノードからエビクトされます。**NodeMaintenance** CR が削除されると、CR で参照されるノードは新規ワークロードで利用可能になります。



#### 注記

ノードのメンテナンスタスクに **NodeMaintenance** CR を使用すると、標準の Red Hat OpenShift CR 処理を使用して **oc adm cordon** および **oc adm drain** コマンドの場合と同じ結果が得られます。

### 5.2. NODE MAINTENANCE OPERATOR のインストール

Node Maintenance Operator は、Web コンソールまたは Open Shift CLI (**oc**) を使用してインストールできます。



#### 注記

OpenShift Virtualization バージョン 4.10 以下がクラスターにインストールされている場合は、古いバージョンの Node Maintenance Operator が含まれています。

#### 5.2.1. Web コンソールを使用した Node Maintenance Operator のインストール

Red Hat OpenShift Web コンソールを使用して、Node Maintenance Operator をインストールできます。

##### 前提条件

- **cluster-admin** 権限を持つユーザーとしてログインしている。

##### 手順

1. Red Hat OpenShift Web コンソールで、**Operators** → **OperatorHub** に移動します。
2. Node Maintenance Operator を検索し、**Install** をクリックします。
3. Operator が **openshift-operators** namespace にインストールされるように、**Installation mode** と **namespace** のデフォルトの選択を維持します。
4. **Install** をクリックします。

##### 検証

インストールが正常に行われたことを確認するには、以下を実行します。

1. **Operators** → **Installed Operators** ページに移動します。
2. Operator が **openshift-operators** の namespace 内に設置されていることと、その状態が **Succeeded** となっていることを確認してください。

Operator が正常にインストールされていない場合、以下を実行します。

1. **Operators** → **Installed Operators** ページに移動し、**Status** 列でエラーまたは失敗の有無を確認します。
2. **Operators** → **Installed Operators** → **Node Maintenance Operator** → **Details** ページに移動し、Pod を作成する前に **Conditions** セクションでエラーを調べます。
3. **Workloads** → **Pods** ページに移動し、インストールされた namespace で **Node Maintenance Operator** Pod を検索し、**Logs** タブでログを確認します。

### 5.2.2. CLI を使用した Node Maintenance Operator のインストール

OpenShift CLI (**oc**) を使用して、Node Maintenance Operator をインストールできます。

Node Maintenance Operator は、独自の namespace または **openshift-operators** namespace にインストールできます。

独自の namespace に Operator をインストールするには、手順に従います。

**openshift-operators** namespace に Operator をインストールするには、手順の 3 にスキップします。これは、新しい **Namespace** カスタムリソース (CR) と **OperatorGroup** CR を作成する必要がないためです。

#### 前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

#### 手順

1. Node Maintenance Operator の **Namespace** CR を作成します。
  - a. **Namespace** CR を定義し、YAML ファイルを保存します (例: **node-maintenance-namespace.yaml**)。

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

- b. **Namespace** CR を作成するには、次のコマンドを実行します。

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

2. **OperatorGroup** を作成します。
  - a. **OperatorGroup** CR を定義し、YAML ファイルを保存します (例: **node-maintenance-operator-group.yaml**)。

■

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: node-maintenance-operator
  namespace: nmo-test
```

- b. **OperatorGroup** CR を作成するには、次のコマンドを実行します。

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

3. **Subscription** CR を作成します。

- a. **Subscription** CR を定義し、YAML ファイルを保存します (例: **node-maintenance-subscription.yaml**)。

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

- ❶ Node Maintenance Operator をインストールする **Namespace** を指定します。



### 重要

Node Maintenance Operator を **openshift-operators** namespace にインストールするには、**Subscription** CR で **openshift-operators** を指定します。

- b. **Subscription** CR を作成するには、次のコマンドを実行します。

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

### 検証

1. CSV リソースを調べて、インストールが成功したことを確認します。

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

### 出力例

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

2. Node Maintenance Operator が実行されていることを確認します。

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

出力例

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

Node Maintenance Operator は、制限されたネットワーク環境でサポートされています。詳細は、[ネットワークが制限された環境での Operator Lifecycle Manager の使用](#) を参照してください。

### 5.3. ノードのメンテナンスモードへの設定

**NodeMaintenance** CR を使用して、Web コンソールまたは CLI からノードをメンテナンスモードにすることができます。

#### 5.3.1. Web コンソールでのノードのメンテナンスモードへの設定

ノードをメンテナンスモードに設定するために、Web コンソールを使用して **NodeMaintenance** カスタムリソース (CR) を作成できます。

##### 前提条件

- **cluster-admin** 権限を持つユーザーとしてログインしている。
- **OperatorHub** から Node Maintenance Operator をインストールします。

##### 手順

1. Web コンソールの **Administrator** パースペクティブで、**Operators** → **Installed Operators** に移動します。
2. Operator リストから Node Maintenance Operator を選択します。
3. **Node Maintenance** タブで **Create NodeMaintenance** をクリックします。
4. **Create NodeMaintenance** ページで、**Form view** または **YAML view** を選択して **NodeMaintenance** CR を設定します。
5. 設定した **NodeMaintenance** CR を適用するには、**Create** をクリックします。

##### 検証

**Node Maintenance** タブで **Status** 列を調べ、そのステータスが **Succeeded** であることを確認します。

#### 5.3.2. CLI を使用してノードをメンテナンスモードに設定する場合

**NodeMaintenance** カスタムリソース (CR) を使用して、ノードをメンテナンスモードにすることができます。**NodeMaintenance** CR を適用すると、許可されているすべての Pod が削除され、ノードがスケジューリング不能になります。エビクトされた Pod は、クラスター内の別のノードに移動するようにキューに置かれます。

##### 前提条件

- Red Hat OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしている。

## 手順

1. 次の **NodeMaintenance** CR を作成し、ファイルを **nodemaintenance-cr.yaml** として保存します。

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

- ❶ ノードメンテナンス CR の名前。
- ❷ メンテナンスモードにするノードの名前。
- ❸ メンテナンスの理由を説明するプレーンテキスト。

2. 次のコマンドを実行して、ノードメンテナンス CR を適用します。

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

## 検証

1. 次のコマンドを実行して、メンテナンスタスクの進捗状況を確認します。

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

**<node-name>** はノードの名前です。たとえば、**node-1.example.com** などになります。

2. 出力例を確認します。

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

### 5.3.3. 現在の NodeMaintenance CR タスクのステータスの確認

現在の **NodeMaintenance** CR タスクのステータスを確認できます。

#### 前提条件

- Red Hat OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

## 手順

- 以下のコマンドを実行して、現在のノードのメンテナンスタスクのステータスを確認します (例: **NodeMaintenance** CR または **nm**)。

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

### 出力例

```
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 ノードのドレインの完了率。
- ② エビクションにスケジュールされる Pod 数。
- ③ 最新のエビクションエラー (ある場合)。
- ④ ステータスが最後に更新された時刻。
- ⑤ ノードがメンテナンスモードに入る前の Pod の総数。

## 5.4. メンテナンスモードからのノードの再開

**NodeMaintenance** CR を使用して、Web コンソールまたは CLI から、メンテナンスモードからノードを再開できます。ノードを再起動することにより、ノードをメンテナンスモードから切り替え、再度スケジュール可能な状態にできます。

### 5.4.1. Web コンソールを使用してノードをメンテナンスモードから再開する方法

ノードをメンテナンスモードから再開するために、Web コンソールを使用して **NodeMaintenance** カスタムリソース (CR) を削除できます。

#### 前提条件

- cluster-admin** 権限を持つユーザーとしてログインしている。
- OperatorHub** から Node Maintenance Operator をインストールします。

## 手順

1. Web コンソールの **Administrator** パースペクティブで、**Operators** → **Installed Operators** に移動します。
2. Operator リストから Node Maintenance Operator を選択します。
3. **Node Maintenance** タブで、削除する **NodeMaintenance** CR を選択します。
4. ノードの末尾の Options メニュー  をクリックし、**Delete NodeMaintenance** を選択します。

## 検証

1. Red Hat OpenShift コンソールで、**Compute** → **Nodes** をクリックします。
2. **NodeMaintenance** CR を削除したノードの **Status** 列を調べて、その状況が **Ready** であることを確認します。

### 5.4.2. CLI を使用してノードをメンテナンスモードから再開する方法

**NodeMaintenance** CR を削除することにより、**NodeMaintenance** CR で開始されたメンテナンスモードからノードを再開できます。

#### 前提条件

- Red Hat OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしている。

## 手順

- ノードのメンテナンスタスクが完了したら、アクティブな **NodeMaintenance** CR を削除します。

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

#### 出力例

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

## 検証

1. 次のコマンドを実行して、メンテナンスタスクの進捗状況を確認します。

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

**<node-name>** はノードの名前です。たとえば、**node-1.example.com** などになります。

2. 出力例を確認します。

```
Events:
```

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

## 5.5. ベアメタルノードの操作

ベアメタルノードを含むクラスターの場合、Web コンソールの **アクション** コントロールを使用して、ノードをメンテナンスモードにしたり、ノードをメンテナンスモードから再開したりできます。



### 注記

ベアメタルノードを含むクラスターは、概説したように、Web コンソールと CLI を使用して、ノードをメンテナンスモードにしたり、ノードをメンテナンスモードから再開したりすることもできます。これらのメソッドは、Web コンソールの **アクション** コントロールを使用して、ベアメタルクラスターにのみ適用できます。

### 5.5.1. ベアメタルノードのメンテナンス

Red Hat OpenShift をベアメタルインフラストラクチャーにデプロイする場合、クラウドインフラストラクチャーにデプロイする場合と比較すると、追加で考慮する必要のある点があります。クラスターノードが一時的とみなされるクラウド環境とは異なり、ベアメタルノードを再プロビジョニングするには、メンテナンスタスクにより多くの時間と作業が必要になります。

カーネルエラーや NIC カードのハードウェア障害が原因でベアメタルノードに障害が発生した場合には、障害のあるノードが修復または置き換えられている間に、障害が発生したノード上のワークロードをクラスターのノードで再起動する必要があります。ノードのメンテナンスモードにより、クラスター管理者はノードを正常にオフにし、ワークロードをクラスターの他の部分に移動させ、ワークロードが中断されないようにします。詳細な進捗とノードのステータスについての詳細は、メンテナンス時に提供されます。

### 5.5.2. ベアメタルノードをメンテナンスモードに設定する

**Compute → Nodes** 一覧で各ノードにある Options メニュー  を使用するか、**Node Details** 画面の **Actions** コントロールを使用して、ベアメタルノードをメンテナンスモードに設定します。

#### 手順

1. Web コンソールの **管理者** パースペクティブから、**Compute → Nodes** をクリックします。
2. この画面からノードをメンテナンスモードに設定することができます。これにより、複数のノードでアクションを簡単に実行できるようになります。または、選択したノードの包括的な詳細を表示できる **Node Details** 画面からも実行できるようになります。

- ノードの末尾の Options メニュー  をクリックし、**Start Maintenance** を選択します。
- ノード名をクリックし、**Node Details** 画面を開いて **Actions → Start Maintenance** をクリックします。

3. 確認ウィンドウで **Start Maintenance** をクリックします。

ノードはスケジュール可能ではなくなりました。**LiveMigration** エビクションストラテジーを使用する仮想マシンがあった場合は、それらをライブマイグレーションします。このノードの他のすべての Pod および仮想マシンは削除され、別のノードで再作成されます。

## 検証

- **Compute → Nodes** ページに移動し、対応するノードのステータスが **Under maintenance** であることを確認します。

### 5.5.3. メンテナンスモードからのベアメタルノードの再開

**Compute → Nodes** リストの各ノードにあるオプションメニュー  を使用して、または **Node Details** 画面の **Actions** コントロールを使用して、メンテナンスモードからベアメタルノードを再開します。

## 手順

1. Web コンソールの **管理者** パースペクティブから、**Compute → Nodes** をクリックします。
2. 複数のノードでアクションを簡単に実行できるこの画面からノードを再開できます。または、選択したノードの包括的な詳細を表示できる **Node Details** 画面からもノードを再開できます。

- ノードの末尾の Options メニュー  をクリックし、**Stop Maintenance** を選択します。
- ノード名をクリックし、**Node Details** 画面を開いて **Actions → Stop Maintenance** をクリックします。

3. 確認ウィンドウで **Stop Maintenance** をクリックします。

ノードがスケジュール可能になります。メンテナンス前にノードで実行されていた仮想マシンインスタンスがあった場合、それらは自動的にこのノードに移行されません。

## 検証

- **Compute → Nodes** ページに移動し、対応するノードのステータスが **Ready** であることを確認します。

## 5.6. NODE MAINTENANCE OPERATOR に関するデータの収集

Node Maintenance Operator に関するデバッグ情報を収集するには、**must-gather** ツールを使用します。Node Maintenance Operator の **must-gather** イメージは、**特定の機能に関するデータの収集** を参照してください。

## 5.7. 関連情報

- [クラスターに関するデータの収集](#)
- [ノード上の Pod を退避させる方法](#)

- ノードをスケジュール対象外 (Unschedulable) またはスケジュール対象 (Schedulable) としてマークする方法