



OpenShift Container Platform 4.19

インストール設定

インストール中のクラスター全体の設定

OpenShift Container Platform 4.19 インストール設定

インストール中のクラスター全体の設定

Legal Notice

Copyright © 2025 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

このドキュメントでは、OpenShift Container Platform クラスターの初期設定を実行する方法を説明します。

Table of Contents

第1章 ノードのカスタマイズ	3
1.1. BUTANE でのマシン設定の作成	3
1.2. DAY-1 カーネル引数の追加	5
1.3. カーネルモジュールのノードへの追加	6
1.4. インストール時のディスクの暗号化およびミラーリング	13
1.5. CHRONY タイムサービスの設定	27
1.6. 関連情報	28
第2章 ファイアウォールの設定	29
2.1. OPENSIFT CONTAINER PLATFORM のファイアウォールの設定	29
2.2. OPENSIFT CONTAINER PLATFORM ネットワークフローマトリックス	35

第1章 ノードのカスタマイズ

OpenShift Container Platform は、Ignition を介してクラスター全体の設定およびマシンごとの設定の両方をサポートしています。これにより、オペレーティングシステムに対して任意のパーティショニングやファイル内容の変更を行うことができます。一般に、設定ファイルが Red Hat Enterprise Linux (RHEL) で文書化されている場合は、Ignition を介した変更がサポートされます。

マシン設定の変更をデプロイするには 2 つの方法があります。

- **openshift-install** の実行時にクラスターを起動するためにマニフェストファイルに組み込まれるマシン設定を作成します。
- Machine Config Operator を使用して実行中の OpenShift Container Platform ノードに渡されるマシン設定を作成します。

さらに、ベアメタルノードのインストール時に **coreos-installer** に渡される Ignition 設定などの参照設定を変更すると、マシンごとの設定が可能になります。現在、これらの変更は Machine Config Operator に表示されません。

以下のセクションでは、この方法でノード上で設定する必要がある機能の説明します。

1.1. BUTANE でのマシン設定の作成

マシン設定は、ユーザーおよびファイルシステムの作成、ネットワークの設定、systemd ユニットのインストールなどを行う方法をマシンに指示することで、コントロールプレーンマシンおよびワーカーマシンを設定するために使用されます。

マシン設定の変更は困難である可能性があるため、Butane 設定を使用してマシン設定を作成することができます。これにより、ノードの設定がより容易になります。

1.1.1. Butane について

Butane は、OpenShift Container Platform が使用するコマンドラインユーティリティーで、マシン設定を作成するための便利で簡略化した構文を提供したり、マシン設定の追加検証を実行したりします。Butane が受け入れる Butane 設定ファイルの形式は、[OpenShift Butane config spec](#) で定義されています。

1.1.2. Butane のインストール

Butane ツール (**butane**) をインストールして、コマンドラインインターフェイスから OpenShift Container Platform マシン設定を作成できます。対応するバイナリーファイルをダウンロードし、Linux、Windows、または macOS に **butane** をインストールできます。

ヒント

Butane リリースは、古いリリースと、Fedora CoreOS Config Transpiler (FCCT) との後方互換性があります。

手順

1. Butane イメージのダウンロードページ (<https://mirror.openshift.com/pub/openshift-v4/clients/butane/>) に移動してください。
2. **butane** バイナリーを取得します。

- a. 最新バージョンの Butane の場合は、最新の **butane** イメージを現在のディレクトリーに保存します。

```
$ curl https://mirror.openshift.com/pub/openshift-v4/clients/butane/latest/butane --output butane
```

- b. オプション: aarch64 や ppc64le など、Butane をインストールする特定のタイプのアーキテクチャーの場合は、適切な URL を指定してください。以下に例を示します。

```
$ curl https://mirror.openshift.com/pub/openshift-v4/clients/butane/latest/butane-aarch64 --output butane
```

3. ダウンロード済みのバイナリーファイルを実行可能にします。

```
$ chmod +x butane
```

4. **butane** バイナリーファイルを **PATH** にあるディレクトリーに移動します。**PATH** を確認するには、ターミナルを開き、以下のコマンドを実行します。

```
$ echo $PATH
```

検証手順

- **butane** コマンドを実行して、Butane ツールを使用できるようになりました。

```
$ butane <butane_file>
```

1.1.3. Butane を使用した MachineConfig オブジェクトの作成

Butane を使用して **MachineConfig** オブジェクトを作成できるため、インストール時に、または Machine Config Operator を使用して、ワーカーノードまたはコントロールプレーンノードを設定できます。

前提条件

- **butane** ユーティリティーをインストールした。

手順

1. Butane 設定ファイルを作成します。以下の例では、**99-worker-custom.bu** という名前のファイルを作成します。このファイルは、カーネルデバッグメッセージを表示するようにシステムコンソールを設定し、chrony タイムサービスのカスタム設定を指定します。

```
variant: openshift
version: 4.19.0
metadata:
  name: 99-worker-custom
  labels:
    machineconfiguration.openshift.io/role: worker
openshift:
  kernel_arguments:
    - loglevel=7
```

```

storage:
  files:
    - path: /etc/chrony.conf
      mode: 0644
      overwrite: true
      contents:
        inline: |
          pool 0.rhel.pool.ntp.org iburst
          driftfile /var/lib/chrony/drift
          makestep 1.0 3
          rtcsync
          logdir /var/log/chrony

```



注記

99-worker-custom.bu ファイルは、ワーカーノードのマシン設定を作成するように設定されます。コントロールプレーンノードにデプロイするには、ロールを **worker** から **master** に変更します。どちらの方法でも、デプロイメントの種類ごとに異なるファイル名を使用して手順全体を繰り返すことができます。

2. 直前の手順で作成したファイルを Butane に指定して **MachineConfig** オブジェクトを作成します。

```
$ butane 99-worker-custom.bu -o ./99-worker-custom.yaml
```

MachineConfig オブジェクト YAML ファイルは、マシンの設定を終了するために作成されます。

3. 将来的に **MachineConfig** オブジェクトを更新する必要がある場合に備えて、Butane 設定を保存します。
4. クラスターがまだ起動していない場合は、マニフェストファイルを生成し、**MachineConfig** オブジェクト YAML ファイルを **openshift** ディレクトリーに追加します。クラスターがすでに実行中の場合は、ファイルを以下のように適用します。

```
$ oc create -f 99-worker-custom.yaml
```

関連情報

- [カーネルモジュールのノードへの追加](#)
- [インストール時のディスクの暗号化およびミラーリング](#)

1.2. DAY-1 カーネル引数の追加

多くの場合は、カーネル引数を day-2 アクティビティーとして変更することが推奨されますが、初期クラスターのインストール時にすべてのマスターまたはワーカーノードにカーネル引数を追加できます。以下は、クラスターのインストール時にカーネル引数を追加して、システムの初回起動前に有効にする必要が生じる可能性のある理由です。

- システムの起動前に、低レベルのネットワーク設定を実行する必要がある場合。

- SELinux などの機能を無効にし、初回起動時にシステムに影響を与えないようにする必要があります。



警告

実稼働環境の RHCOS での SELinux の無効化はサポートされていません。ノード上で SELinux が無効になったら、再プロビジョニングしてから実稼働クラスターに再び追加する必要があります。

カーネル引数をマスターまたはワーカーノードに追加するには、**MachineConfig** オブジェクトを作成し、そのオブジェクトをクラスターのセットアップ時に Ignition が使用するマニフェストファイルのセットに挿入できます。

起動時に RHEL 8 カーネルに渡すことのできる引数のリストは、[Kernel.org カーネルパラメーター](https://kernel.org/doc/Documentation/kernel-parameters.txt) を参照してください。カーネル引数が OpenShift Container Platform の初回インストールを完了するために必要な場合は、この手順でカーネル引数のみを追加することが推奨されます。

手順

1. インストールプログラムが含まれるディレクトリーに切り替え、クラスターの Kubernetes マニフェストを生成します。

```
$ ./openshift-install create manifests --dir <installation_directory>
```

2. カーネル引数をワーカーまたコントロールプレーンノードに追加するかどうかを決定します。
3. **openshift** ディレクトリーでファイル (例: **99-openshift-machineconfig-master-kargs.yaml**) を作成し、カーネル設定を追加するために **MachineConfig** オブジェクトを定義します。この例では、**loglevel=7** カーネル引数をコントロールプレーンノードに追加します。

```
$ cat << EOF > 99-openshift-machineconfig-master-kargs.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-openshift-machineconfig-master-kargs
spec:
  kernelArguments:
    - loglevel=7
EOF
```

カーネル引数をワーカーノードに追加する場合は、**master** を **worker** に切り替えます。マスターおよびワーカーノードの両方に追加するために別々の YAML ファイルを作成します。

クラスターの作成を継続できます。

1.3. カーネルモジュールのノードへの追加

大半の一般的なハードウェアの場合、Linux カーネルには、コンピューターの起動時にそのハードウェアを使用するために必要となるデバイスドライバモジュールが含まれます。ただし、一部のハードウェアの場合は、Linux でモジュールを利用できません。したがって、各ホストコンピューターにこれらのモジュールを提供する方法を確保する必要があります。この手順では、OpenShift Container Platform クラスターのノードにこれを実行する方法を説明します。

この手順に従ってカーネルモジュールを最初にデプロイする際、モジュールは現行のカーネルに対して利用可能になります。新規カーネルがインストールされると、kmods-via-containers ソフトウェアはモジュールを再ビルドし、デプロイしてそのモジュールの新規カーネルと互換性のあるバージョンが利用可能になるようにします。

この機能によって各ノードでモジュールが最新の状態に保てるようにするために、以下が実行されます。

- 新規カーネルがインストールされているかどうかを検出するために、システムの起動時に起動する各ノードに systemd サービスを追加します。
- 新規カーネルが検出されると、サービスはモジュールを再ビルドし、これをカーネルにインストールします。

この手順に必要なソフトウェアの詳細は、[kmods-via-containers github](#) サイトを参照してください。

以下の重要な点に留意してください。

- この手順はテクノロジープレビューです。
- ソフトウェアのツールおよびサンプルは公式の RPM 形式で利用できず、現時点ではこの手順に記載されている非公式の **github.com** サイトからしか取得できません。
- この手順で追加する必要がある可能性のあるサードパーティーのカーネルモジュールは、Red Hat はサポートしません。
- この手順では、カーネルモジュールのビルドに必要なソフトウェアは RHEL 8 コンテナにデプロイされます。モジュールは、ノードが新規カーネルを取得する際に各ノードで自動的に再ビルドされることに注意してください。このため、各ノードには、モジュールの再ビルドに必要なカーネルと関連パッケージを含む **yum** リポジトリへのアクセスが必要です。このコンテンツは、有効な RHEL サブスクリプションを使用して効果的に利用できます。

1.3.1. カーネルモジュールコンテナのビルドおよびテスト

カーネルモジュールを OpenShift Container Platform クラスターにデプロイする前に、プロセスを別の RHEL システムでテストできます。カーネルモジュールのソースコード、KVC フレームワーク、および kmod-via-containers ソフトウェアを収集します。次にモジュールをビルドし、テストします。RHEL 8 システムでこれを行うには、以下を実行します。

手順

1. RHEL 8 システムを登録します。

```
# subscription-manager register
```

2. RHEL 8 システムにサブスクリプションを割り当てます。

```
# subscription-manager attach --auto
```

3. ソフトウェアとコンテナのビルドに必要なソフトウェアをインストールします。

```
# yum install podman make git -y
```

4. **kmod-via-containers** リポジトリのクローンを作成します。

- a. リポジトリのフォルダーを作成します。

```
$ mkdir kmods; cd kmods
```

- b. リポジトリのクローンを作成します。

```
$ git clone https://github.com/kmods-via-containers/kmods-via-containers
```

5. RHEL 8 ビルドホストに KVC フレームワークインスタンスをインストールし、モジュールをテストします。これにより、**kmods-via-container** systemd サービスが追加され、読み込まれます。

- a. **kmod-via-containers** ディレクトリーに移動します。

```
$ cd kmods-via-containers/
```

- b. KVC フレームワークインスタンスをインストールします。

```
$ sudo make install
```

- c. systemd マネージャー設定を再読み込みします。

```
$ sudo systemctl daemon-reload
```

6. カーネルモジュールのソースコードを取得します。ソースコードは、制御下になく、他から提供されるサードパーティーモジュールをビルドするために使用される可能性があります。システムに対してクローン作成できる以下の **kvc-simple-kmod** サンプルのコンテンツと同様のコンテンツが必要になります。

```
$ cd .. ; git clone https://github.com/kmods-via-containers/kvc-simple-kmod
```

7. この例では、設定ファイル **simple-kmod.conf** を編集し、Dockerfile の名前を **Dockerfile.rhel** に変更します。

- a. **kvc-simple-kmod** ディレクトリーに移動します。

```
$ cd kvc-simple-kmod
```

- b. Dockerfile の名前を変更します。

```
$ cat simple-kmod.conf
```

Dockerfile の例

```
KMOD_CONTAINER_BUILD_CONTEXT="https://github.com/kmods-via-containers/kvc-simple-kmod.git"
```

```
KMOD_CONTAINER_BUILD_FILE=Dockerfile.rhel
KMOD_SOFTWARE_VERSION=dd1a7d4
KMOD_NAMES="simple-kmod simple-procfs-kmod"
```

8. この例ではカーネルモジュール **simple-kmod** の **kmods-via-containers@.service** のインスタンスを作成します。

```
$ sudo make install
```

9. **kmods-via-containers@.service** インスタンスを有効にします。

```
$ sudo kmods-via-containers build simple-kmod $(uname -r)
```

10. systemd サービスを有効にし、起動します。

```
$ sudo systemctl enable kmods-via-containers@simple-kmod.service --now
```

- a. サービスのステータスを確認します。

```
$ sudo systemctl status kmods-via-containers@simple-kmod.service
```

出力例

```
● kmods-via-containers@simple-kmod.service - Kmods Via Containers - simple-kmod
   Loaded: loaded (/etc/systemd/system/kmods-via-containers@.service;
           enabled; vendor preset: disabled)
   Active: active (exited) since Sun 2020-01-12 23:49:49 EST; 5s ago...
```

11. カーネルモジュールがロードされていることを確認するには、**lsmod** コマンドを使用してモジュールをリスト表示します。

```
$ lsmod | grep simple_
```

出力例

```
simple_procfs_kmod    16384 0
simple_kmod           16384 0
```

12. オプション: 他の方法を使用して **simple-kmod** のサンプルが機能していることを確認します。

- **dmesg** を使用してカーネルリングバッファで "Hello world" メッセージを探します。

```
$ dmesg | grep 'Hello world'
```

出力例

```
[ 6420.761332] Hello world from simple_kmod.
```

- **/proc** で **simple-procfs-kmod** の値を確認します。

```
$ sudo cat /proc/simple-procfs-kmod
```

出力例

```
simple-procfs-kmod number = 0
```

- **spkut** コマンドを実行して、モジュールの詳細情報を取得します。

```
$ sudo spkut 44
```

出力例

```
KVC: wrapper simple-kmod for 4.19.0-147.3.1.el8_1.x86_64
Running userspace wrapper using the kernel module container...
+ podman run -i --rm --privileged
  simple-kmod-dd1a7d4:4.19.0-147.3.1.el8_1.x86_64 spkut 44
simple-procfs-kmod number = 0
simple-procfs-kmod number = 44
```

その後は、システムの起動時に、このサービスは新規カーネルが実行中であるかどうかをチェックします。新規カーネルがある場合は、サービスは新規バージョンのカーネルモジュールをビルドし、これをロードします。モジュールがすでにビルドされている場合は、これをロードします。

1.3.2. カーネルモジュールの OpenShift Container Platform へのプロビジョニング

OpenShift Container Platform クラスターの初回起動時にカーネルモジュールを有効にする必要があるかどうかに応じて、以下のいずれかの方法でデプロイするようにカーネルモジュールを設定できます。

- **クラスターインストール時のカーネルモジュールのプロビジョニング (day-1)**: コンテンツを **MachineConfig** として作成し、これをマニフェストファイルのセットと共に組み込み、これを **openshift-install** に提供できます。
- **Machine Config Operator によるカーネルモジュールのプロビジョニング (day-2)**: カーネルモジュールを追加する際にクラスターが稼働するまで待機できる場合は、Machine Config Operator (MCO) を使用してカーネルモジュールソフトウェアをデプロイできます。

いずれの場合も、各ノードは、新しいカーネルが検出された際に、カーネルパッケージおよび関連するソフトウェアパッケージを取得できるようにする必要があります。該当するコンテンツを取得できるように各ノードをセットアップする方法はいくつかあります。

- 各ノードに RHEL エンタイトルメントを提供します。
- **/etc/pki/entitlement** ディレクトリーから、既存 RHEL ホストの RHEL エンタイトルメントを取得し、それらを Ignition 設定の作成時に提供する他のファイルと同じ場所にコピーします。
- Dockerfile 内で、カーネルおよびその他のパッケージを含む **yum** リポジトリへのポインターを追加します。これには、新たにインストールされたカーネルと一致させる必要があるため、新規のカーネルパッケージが含まれている必要があります。

1.3.2.1. MachineConfig オブジェクトを介したカーネルモジュールのプロビジョニング

MachineConfig オブジェクトでカーネルモジュールソフトウェアをパッケージ化することで、インストール時に、または Machine Config Operator を使用して、そのソフトウェアをワーカーノードまたはコントロールプレーンノードに配信できます。

手順

1. RHEL 8 システムを登録します。

```
# subscription-manager register
```

2. RHEL 8 システムにサブスクリプションを割り当てます。

```
# subscription-manager attach --auto
```

3. ソフトウェアのビルドに必要なソフトウェアをインストールします。

```
# yum install podman make git -y
```

4. カーネルモジュールおよびツールをホストするディレクトリを作成します。

```
$ mkdir kmods; cd kmods
```

5. **kmods-via-containers** ソフトウェアを取得します。

- a. **kmods-via-containers** リポジトリのクローンを作成します。

```
$ git clone https://github.com/kmods-via-containers/kmods-via-containers
```

- b. **kvc-simple-kmod** リポジトリのクローンを作成します。

```
$ git clone https://github.com/kmods-via-containers/kvc-simple-kmod
```

6. モジュールソフトウェアを取得します。この例では、**kvc-simple-kmod** が使用されます。

7. 先ほどクローン作成したリポジトリを使用して、**fakeroot** ディレクトリを作成し、Ignition 経由で配信するファイルをそのディレクトリに追加します。

- a. ディレクトリを作成します。

```
$ FAKEROOT=$(mktemp -d)
```

- b. **kmod-via-containers** ディレクトリに移動します。

```
$ cd kmods-via-containers
```

- c. KVC フレームワークインスタンスをインストールします。

```
$ make install DESTDIR=${FAKEROOT}/usr/local CONFDIR=${FAKEROOT}/etc/
```

- d. **kvc-simple-kmod** ディレクトリに移動します。

```
$ cd ../kvc-simple-kmod
```

- e. インスタンスを作成します。

```
$ make install DESTDIR=${FAKEROOT}/usr/local CONFDIR=${FAKEROOT}/etc/
```

8. fakeroot ディレクトリーのクローンを作成し、以下のコマンドを実行してシンボリックリンクをターゲットのコピーに置き換えます。

```
$ cd .. && rm -rf kmod-tree && cp -Lpr ${FAKEROOT} kmod-tree
```

9. カーネルモジュールツリーを埋め込む Butane 設定ファイル (**99-simple-kmod.bu**) を作成し、systemd サービスを有効にします。



注記

Butane の詳細は、「Butane を使用したマシン設定の作成」を参照してください。

```
variant: openshift
version: 4.19.0
metadata:
  name: 99-simple-kmod
  labels:
    machineconfiguration.openshift.io/role: worker ❶
storage:
  trees:
    - local: kmod-tree
systemd:
  units:
    - name: kmods-via-containers@simple-kmod.service
      enabled: true
```

- ❶ コントロールプレーンノードでデプロイするには、**worker** を **master** に変更します。コントロールプレーンおよびワーカーノードの両方にデプロイするには、それぞれのノードのタイプに対してこれらの残りの手順を1回ずつ実行します。

10. Butane を使用して、配信されるファイルおよび設定を含むマシン設定 YAML ファイルの **99-simple-kmod.yaml** を生成します。

```
$ butane 99-simple-kmod.bu --files-dir . -o 99-simple-kmod.yaml
```

11. クラスターがまだ起動していない場合は、マニフェストファイルを生成し、そのファイルを **openshift** ディレクトリーに追加します。クラスターがすでに実行中の場合は、ファイルを以下のように適用します。

```
$ oc create -f 99-simple-kmod.yaml
```

ノードは **kmods-via-containers@simple-kmod.service** サービスを起動し、カーネルモジュールがロードされます。

12. カーネルモジュールがロードされていることを確認するには、ノードにログインすることができます (**oc debug node/<openshift-node>** を使用してから **chroot /host** を使用します)。モジュールをリスト表示するには、**lsmod** コマンドを使用します。

```
$ lsmod | grep simple_
```

出力例

```
simple_procfs_kmod 16384 0
simple_kmod         16384 0
```

1.4. インストール時のディスクの暗号化およびミラーリング

OpenShift Container Platform のインストール時に、クラスターノードでブートディスクの暗号化およびミラーリングを有効にできます。

1.4.1. ディスクの暗号化について

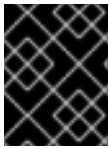
インストール時に、コントロールプレーンおよびコンピュートノードのブートディスクの暗号化を有効にできます。OpenShift Container Platform は Trusted Platform Module (TPM) v2 および Tang 暗号化モードをサポートします。

TPM v2

これは優先モードです。TPM v2 は、パスフレーズをサーバー上の安全な暗号プロセッサに保存します。このモードを使用すると、ディスクがサーバーから取り外された場合に、クラスターノード上のブートディスクデータの暗号化が解除されないようにすることができます。

Tang

Tang および Clevis は、ネットワークバインドディスク暗号化 (NBDE) を有効にするサーバーおよびクライアントコンポーネントです。クラスターノードのブートディスクデータを1つまたは複数の Tang サーバーにバインドできます。これにより、ノードが Tang サーバーにアクセスできる安全なネットワーク上にない限り、データの復号化が防止されます。Clevis は、クライアント側の復号化の実装に使用される自動復号化フレームワークです。



重要

Tang 暗号化モードを使用したディスクの暗号化は、user-provisioned infrastructure でのベアメタルおよび vSphere インストールでのみサポートされます。

以前のバージョンの Red Hat Enterprise Linux CoreOS (RHCOS) では、ディスク暗号化は Ignition 設定で `/etc/clevis.json` を指定して設定されました。このファイルは、OpenShift Container Platform 4.7 以降で作成されたクラスターではサポートされていません。次の手順を使用して、ディスク暗号化を設定します。

TPM v2 または Tang 暗号化モードを有効にすると、RHCOS ブートディスクは LUKS2 形式を使用して暗号化されます。

この機能には以下の特徴があります。

- installer-provisioned infrastructure、user-provisioned infrastructure、および Assisted Installer のデプロイメントで利用可能
- Assisted Installer のデプロイメントの場合:
 - 各クラスターは、Tang または TPM の1つの暗号化方式のみを持つことができる
 - 一部またはすべてのノードで暗号化を有効にできる
 - Tang のしきい値がないため、すべてのサーバーが有効で動作している必要がある
 - 暗号化はインストールディスクにのみ適用され、ワークロードディスクには適用されない

- Red Hat Enterprise Linux CoreOS (RHCOS) システムのみでサポートされる
- マニフェストのインストール段階でディスク暗号化を設定し、最初の起動以降、ディスクに書き込まれるすべてのデータを暗号化する
- パスフレーズを提供するのにユーザーの介入を必要としない。
- AES-256-XTS 暗号化を使用する

1.4.1.1. 暗号化しきい値の設定

OpenShift Container Platform では、複数の Tang サーバーの要件を指定できます。TPM v2 と Tang 暗号化モードを同時に設定することもできます。これにより、TPM セキュア暗号プロセッサが存在し、Tang サーバーがセキュアネットワーク経由でアクセスできる場合にのみ、ブートディスクデータの復号化が有効になります。

ブタン設定で **threshold** 属性を使用して、復号化が発生するために必要な TPM v2 および Tang 暗号化条件の最小数を定義できます。

宣言条件の組み合わせで指定値に到達した場合に、しきい値が満たされます。オフラインプロビジョニングの場合、オフラインサーバーは含まれているアドバタイズメントを使用してアクセスされ、オンラインサーバーの数が設定されたしきい値を満たさない場合にのみ、その提供されたアドバタイズメントを使用します。

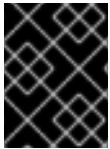
たとえば、次の設定の **threshold 2** は、オフラインサーバーをバックアップとして使用して2つの Tang サーバーにアクセスするか、TPM セキュア暗号プロセッサといずれかの Tang サーバーにアクセスすることで到達できます。

ディスク暗号化の Butane 設定例

```
variant: openshift
version: 4.19.0
metadata:
  name: worker-storage
  labels:
    machineconfiguration.openshift.io/role: worker
boot_device:
  layout: x86_64 ❶
  luks:
    tpm2: true ❷
    tang: ❸
      - url: http://tang1.example.com:7500
        thumbprint: jwGN5tRFK-kF6pIX89ssF3khxxX
      - url: http://tang2.example.com:7500
        thumbprint: VCJsvZFjBSIHsldw78rOrq7h2ZF
      - url: http://tang3.example.com:7500
        thumbprint: PLjNyRdGw03zIRoGjQYMahSZGu9
      advertisement: "{\"payload\": \"...\", \"protected\": \"...\", \"signature\": \"...\"}" ❹
    threshold: 2 ❺
openshift:
  fips: true
```

- ❶ このフィールドをクラスターノードの命令セットアーキテクチャーに設定します。いくつかの例には、**x86_64**、**aarch64**、または **ppc64le** が含まれます。

- 2 Trusted Platform Module (TPM) を使用してルートファイルシステムを暗号化する場合は、このフィールドを追加してください。
- 3 1台以上の Tang サーバーを使用する必要がある場合は、このセクションを追加してください。
- 4 オプション: オフラインプロビジョニングにこのフィールドを含めます。Ignition は、実行時にサーバーからアドバタイズメントを取得するのではなく、Tang サーバーバインディングをプロビジョニングします。これにより、プロビジョニング時にサーバーが利用できなくなります。
- 5 復号化を行うために必要な TPM v2 および Tang 暗号化条件の最小数を指定します。



重要

デフォルトの **しきい値** は **1** です。設定に複数の暗号化条件を追加してもしきい値を指定しない場合は、いずれかの条件が満たされると復号が行われます。



注記

復号に TPM v2 と Tang が必要な場合、**threshold** 属性の値は、指定された Tang サーバーの総数に 1 を加えた値と等しくする必要があります。**threshold** が低い場合は、単一の暗号化モードを使用することで、しきい値に達する可能性があります。たとえば、**tpm2** を **true** に設定し、2 つの Tang サーバーを指定すると、TPM セキュア暗号プロセッサが利用できない場合でも、2 つの Tang サーバーにアクセスすることでしきい値 **2** を満たすことができます。

1.4.2. ディスクのミラーリングについて

コントロールプレーンおよびワーカーノードでの OpenShift Container Platform のインストール時に、ブートおよびその他のディスクの 2 つ以上の冗長ストレージデバイスへのミラーリングを有効にできます。1 つのデバイスが使用可能なままであれば、ストレージデバイスの障害後もノードは機能し続けます。

ミラーリングは、障害の発生したディスクの置き換えをサポートしません。ノードを再プロビジョニングして、ミラーを本来の劣化していない状態に復元します。



注記

user-provisioned infrastructure のデプロイメントの場合、ミラーリングは RHCOS システムでのみ使用できます。ミラーリングのサポートは、BIOS または UEFI で起動した **x86_64** ノードおよび **ppc64le** ノードで利用できます。

1.4.3. ディスク暗号化およびミラーリングの設定

OpenShift Container Platform のインストール時に暗号化およびミラーリングを有効にし、設定できます。

前提条件

- インストールノードで OpenShift Container Platform インストールプログラムをダウンロードしている。
- インストールノードに Butane がインストールされている。



注記

Butane は、OpenShift Container Platform が使用するコマンドラインユーティリティであり、マシンの設定を記述および検証するための便利で簡潔な構文を提供します。詳細は、「Butane を使用したマシン設定の作成」を参照してください。

- Tang 交換キーのサムプリントの生成に使用できる Red Hat Enterprise Linux (RHEL) 8 マシンにアクセスできる。

手順

1. TPM v2 を使用してクラスターを暗号化する必要がある場合は、TPM v2 暗号化を各ノードのホストファームウェアで有効にする必要があるかどうかを確認します。これは、ほとんどの Dell システムで必要になります。特定のシステムのマニュアルを確認してください。
2. Tang を使用してクラスターを暗号化する必要がある場合は、以下の準備段階の手順に従います。
 - a. Tang サーバーを設定するか、既存のサーバーにアクセスします。手順は、[Network-bound disk encryption](#) を参照してください。
 - b. RHEL 8 マシンに **clevis** パッケージがインストールされていない場合はインストールします。

```
$ sudo yum install clevis
```

- c. RHEL 8 マシンで以下のコマンドを実行し、交換キーのサムプリントを生成します。**http://tang1.example.com:7500** を Tang サーバーの URL に置き換えます。

```
$ clevis-encrypt-tang '{"url":"http://tang1.example.com:7500"}' < /dev/null > /dev/null ❶
```

- ❶ この例では、**tangd.socket** は Tang サーバーのポート **7500** でリッスンしています。



注記

clevis-encrypt-tang コマンドは、交換キーの拇印を生成します。このステップでは、データは暗号化コマンドに渡されません。**/dev/null** は、プレーンテキストの代わりに入力としてここに存在します。この手順には必要ないため、暗号化された出力は **/dev/null** に送信されます。

出力例

```
The advertisement contains the following signing keys:
```

```
PLjNyRdGw03zIRoGjQYMahSZGu9 ❶
```

- ❶ エクスチェンジキーのサムプリント。

Do you wish to trust these keys? [ynYN] のプロンプトが表示されたら、**Y** と入力します。

d. オプション: オフライン Tang プロビジョニングの場合は、以下を行います。

- i. **curl** コマンドを使用して、サーバーからアドバタイズメントを取得します。**http://tang2.example.com:7500** を Tang サーバーの URL に置き換えます。

```
$ curl -f http://tang2.example.com:7500/adv > adv.jws && cat adv.jws
```

予想される出力

```
{"payload": "eyJrZXlzljogW3siYWxnljogIkV", "protected":  
"eyJhbGciOiJFUzUxMlslmN0eSI", "signature": "ADLgk7fZdE3Yt4FyYsm0pHiau7Q"}
```

- ii. 暗号化のためにアドバタイズメントファイルを Clevis に提供します。

```
$ clevis-encrypt-tang '{"url":"http://tang2.example.com:7500","adv":"adv.jws"}' <  
/dev/null > /dev/null
```

- e. ノードが静的 IP アドレス指定で設定されている場合は、RHCOS ノードをインストールするときに **coreos-installer iso customize --dest-karg-append** を実行するか、**coreos-installer --append-karg** オプションを使用して、インストール済みシステムの IP アドレスを設定します。ネットワークに必要な **ip=** およびその他の引数を追加します。



重要

一部の静的 IP の設定方法は、初回のブート後に `initramfs` に影響を与えず、Tang 暗号化では機能しない場合があります。これらには、**coreos-installer --copy-network** オプション、**coreos-installer iso customize --network-keyfile** オプション、および **coreos-installer pxe customize --network-keyfile** オプションが含まれるほか、インストール中のライブ ISO または PXE イメージのカーネルコマンドラインに **ip=** 引数が追加されます。静的 IP 設定が間違っていると、ノードの 2 回目のブートが失敗します。

- 3. インストールノードで、インストールプログラムが含まれるディレクトリーに切り替え、クラスタの Kubernetes マニフェストを生成します。

```
$ ./openshift-install create manifests --dir <installation_directory> ❶
```

- ❶ **<installation_directory>** は、インストールファイルを保存するディレクトリーへのパスに置き換えます。

- 4. ディスクの暗号化、ミラーリング、またはそれら両方を設定する Butane 設定を作成します。たとえば、コンピュートノードのストレージを設定するには、**\$HOME/clusterconfig/worker-storage.bu** ファイルを作成します。

起動デバイスの Butane 設定例

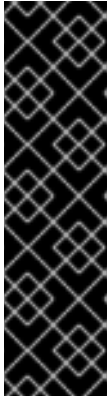
```
variant: openshift  
version: 4.19.0  
metadata:  
  name: worker-storage ❶  
  labels:
```

```

machineconfiguration.openshift.io/role: worker ❷
boot_device:
layout: x86_64 ❸
luks: ❹
  tpm2: true ❺
  tang: ❻
    - url: http://tang1.example.com:7500 ❼
      thumbprint: PLjNyRdGw03zlRoGjQYMahSZGu9 ❽
    - url: http://tang2.example.com:7500
      thumbprint: VCJsvZFjBSIHSlw78rOrq7h2ZF
      advertisement: '{"payload": "eyJrZXlzljogW3siYWxnljogIkV", "protected":
"eyJhbGciOiJFUzUxMlslmN0eSI", "signature": "ADLgk7fZdE3Yt4FyYsm0pHiau7Q"}' ❾
  threshold: 1 ❿
mirror: ⓫
  devices: ⓬
    - /dev/sda
    - /dev/sdb
openshift:
  fips: true ⓭

```

- ❶ ❷ コントロールプレーンの設定は、これらの両方の場所で **worker** を **master** に置き換えます。
- ❸ このフィールドをクラスターノードの命令セットアーキテクチャーに設定します。いくつかの例には、**x86_64**、**aarch64**、または **ppc64le** が含まれます。
- ❹ ルートファイルシステムを暗号化する必要がある場合は、このセクションを追加してください。詳細は、「ディスクの暗号化について」を参照してください。
- ❺ Trusted Platform Module (TPM) を使用してルートファイルシステムを暗号化する場合は、このフィールドを追加してください。
- ❻ 1 台以上の Tang サーバーを使用する必要がある場合は、このセクションを追加してください。
- ❼ Tang サーバーの URL を指定します。この例では、**tangd.socket** は Tang サーバーのポート **7500** でリスンしています。
- ❽ 前述のステップで生成された Exchange キーサムプリントを指定します。
- ❾ オプション: オフライン Tang サーバーのアドバタイズメントを有効な JSON 形式で指定します。
- ❿ 復号化に実行に必要な TPM v2 および Tang 暗号化条件の最小数を指定します。デフォルト値は **1** です。このトピックの詳細は、「暗号化しきい値の設定」を参照してください。
- ⓫ ブートディスクをミラーリングする必要がある場合は、このセクションを追加してください。詳細は、「ディスクのミラーリングについて」を参照してください。
- ⓬ RHCOS がインストールされるディスクを含む、ブートディスクミラーに含まれる必要があるすべてのディスクデバイスを一覧表示します。
- ⓭ クラスターで FIPS モードを有効にするためにこのディレクティブを追加します。



重要

クラスターで FIPS モードを有効にするには、FIPS モードで動作するように設定された Red Hat Enterprise Linux (RHEL) コンピューターからインストールプログラムを実行する必要があります。RHEL での FIPS モードの設定の詳細は、[FIPS モードでのシステムのインストール](#) を参照してください。ノードをディスク暗号化とミラーリングの両方を使用するように設定する場合は、両方の機能を同じ Butane 設定ファイルに設定する必要があります。FIPS モードが有効になっているノードでディスク暗号化を設定する際には、FIPS モードが別のマニフェストで有効になっている場合でも、同じ Butane 設定ファイルに **fips** ディレクティブを追加する必要があります。

5. 対応する Butane 設定ファイルからコントロールプレーンまたはコンピュートノードのマニフェストを作成し、**<installation_directory>/openshift** ディレクトリーに保存します。たとえば、コンピュートノードのマニフェストを作成するには、以下のコマンドを実行します。

```
$ butane $HOME/clusterconfig/worker-storage.bu -o <installation_directory>/openshift/99-worker-storage.yaml
```

ディスクの暗号化またはミラーリングを必要とするノード種別ごとに、この手順を繰り返します。

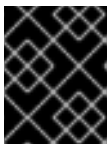
6. 暗号化を有効にする場合は、前の手順で生成されたマニフェストを編集し、暗号 **aes-cbc-essiv:sha256** を **aes-xts-plain64** に置き換えます。次の抜粋は、この変更後の暗号化設定のサンプルを示しています。

```
# ...
luks:
# ...
  options:
    - --cipher
    - aes-xts-plain64
```

7. 今後マニフェストを更新する必要がある場合は、Butane 設定ファイルを保存します。
8. 残りの OpenShift Container Platform インストールを継続します。

ヒント

インストール時に、ディスク暗号化またはミラーリングに関連するエラーメッセージがないか、RHCOS ノードでコンソールログをモニタリングできます。



重要

追加のデータパーティションを設定する場合、暗号化が明示的に要求されない限り、それらは暗号化されません。

検証

OpenShift Container Platform のインストール後に、ブートディスクの暗号化またはミラーリングがクラスターノードで有効にされているかどうかを確認できます。

1. インストールホストから、デバッグ Pod を使用してクラスターノードにアクセスします。
 - a. ノードのデバッグ Pod を開始します。次に例を示します。

```
$ oc debug node/compute-1
```

- b. **/host** をデバッグシェル内の root ディレクトリーとして設定します。デバッグ Pod は、Pod 内の **/host** にノードのルートファイルシステムをマウントします。root ディレクトリーを **/host** に変更すると、ノードの実行可能パスに含まれるバイナリーを実行できます。

```
# chroot /host
```



注記

Red Hat Enterprise Linux CoreOS (RHCOS) を実行する OpenShift Container Platform クラスターノードは変更できず、Operator を使用してクラスターの変更を適用します。SSH を使用したクラスターノードへのアクセスは推奨されません。ただし、OpenShift Container Platform API が利用できない場合や、**kubelet** がターゲットノードで適切に機能しない場合は、**oc** 操作がその影響を受けます。この場合は、代わりに **ssh core@<node>.<cluster_name>.<base_domain>** を使用してノードにアクセスできます。

2. ブートディスクの暗号化を設定している場合は、有効であるかどうかを確認します。

- a. デバッグシェルで、ノードでのルートマッピングのステータスを確認します。

```
# cryptsetup status root
```

出力例

```
/dev/mapper/root is active and is in use.
type: LUKS2 ❶
cipher: aes-xts-plain64 ❷
keysize: 512 bits
key location: keyring
device: /dev/sda4 ❸
sector size: 512
offset: 32768 sectors
size: 15683456 sectors
mode: read/write
```

- ❶ 暗号化形式。TPM v2 または Tang 暗号化モードを有効にすると、RHCOS ブートディスクは LUKS2 形式を使用して暗号化されます。
- ❷ LUKS2 ボリュームの暗号化に使用される暗号化アルゴリズム。
- ❸ 暗号化した LUKS2 ボリュームを含むデバイス。ミラーリングを有効にすると、値は **/dev/md126** などのソフトウェアミラーデバイスを表します。

- b. 暗号化されたデバイスにバインドされる Clevis プラグインを一覧表示します。

```
# clevis luks list -d /dev/sda4 ❶
```

- 1 前述のステップの出力の **device** フィールドに一覧表示されるデバイスを指定します。

出力例

```
1: sss '{"t":1,"pins":{"tang":{"url":"http://tang.example.com:7500"}}}' 1
```

- 1 この出力例では、Tang プラグインは、**/dev/sda4** デバイスの Shamir の Secret Sharing (SSS) Clevis プラグインにより使用されます。

3. ミラーリングを設定している場合は、有効かどうかを確認します。

- a. デバッグシェルから、ノードにあるソフトウェアの RAID デバイスのリストを表示します。

```
# cat /proc/mdstat
```

出力例

```
Personalities : [raid1]
md126 : active raid1 sdb3[1] sda3[0] 1
      393152 blocks super 1.0 [2/2] [UU]

md127 : active raid1 sda4[0] sdb4[1] 2
      51869632 blocks super 1.2 [2/2] [UU]

unused devices: <none>
```

- 1 **/dev/md126** ソフトウェア RAID ミラーデバイスは、クラスターノードの **/dev/sda3** および **/dev/sdb3** ディスクデバイスを使用します。
- 2 **/dev/md127** ソフトウェア RAID ミラーデバイスは、クラスターノードの **/dev/sda4** および **/dev/sdb4** ディスクデバイスを使用します。

- b. 上記のコマンドの出力に記載されている各ソフトウェア RAID デバイスの詳細を確認してください。以下の例は、**/dev/md126** デバイスの詳細を示しています。

```
# mdadm --detail /dev/md126
```

出力例

```
/dev/md126:
  Version : 1.0
  Creation Time : Wed Jul 7 11:07:36 2021
  Raid Level : raid1 1
  Array Size : 393152 (383.94 MiB 402.59 MB)
  Used Dev Size : 393152 (383.94 MiB 402.59 MB)
  Raid Devices : 2
  Total Devices : 2
  Persistence : Superblock is persistent
```

```

Update Time : Wed Jul  7 11:18:24 2021
State : clean 2
Active Devices : 2 3
Working Devices : 2 4
Failed Devices : 0 5
Spare Devices : 0

Consistency Policy : resync

Name : any:md-boot 6
UUID : ccfa3801:c520e0b5:2bee2755:69043055
Events : 19

Number Major Minor RaidDevice State
  0   252    3    0   active sync  /dev/sda3 7
  1   252   19    1   active sync  /dev/sdb3 8

```

- 1 デバイスの RAID レベルを指定します。**raid1** は、RAID 1 ディスクミラーリングを示します。
- 2 RAID デバイスの状態を指定します。
- 3 4 アクティブかつ機能している基礎となるディスクデバイスの数を示します。
- 5 ステータスが failed のディスクデバイスの数を示します。
- 6 ソフトウェア RAID デバイスの名前。
- 7 8 ソフトウェア RAID デバイスが使用する基礎となるディスクデバイスに関する情報を提供します。

c. ソフトウェア RAID デバイ스에 마운트されている 파일 시스템을 일괄 표시합니다.

```
# mount | grep /dev/md
```

出力例

```

/dev/md127 on / type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /etc type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /usr type xfs
(ro,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /sysroot type xfs
(ro,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /var type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /var/lib/containers/storage/overlay type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /var/lib/kubelet/pods/e5054ed5-f882-4d14-b599-99c050d4e0c0/volume-
subpaths/etc/tuned/1 type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /var/lib/kubelet/pods/e5054ed5-f882-4d14-b599-99c050d4e0c0/volume-
subpaths/etc/tuned/2 type xfs

```

```
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /var/lib/kubelet/pods/e5054ed5-f882-4d14-b599-99c050d4e0c0/volume-
subpaths/etc/tuned/3 type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /var/lib/kubelet/pods/e5054ed5-f882-4d14-b599-99c050d4e0c0/volume-
subpaths/etc/tuned/4 type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /var/lib/kubelet/pods/e5054ed5-f882-4d14-b599-99c050d4e0c0/volume-
subpaths/etc/tuned/5 type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md126 on /boot type ext4 (rw,relatime,seclabel)
```

この出力例では、**/boot** ファイルシステムが **/dev/md126** software RAID デバイスに、**root** ファイルシステムが **/dev/md127** にマウントされています。

4. OpenShift Container Platform ノードタイプごとに検証手順を繰り返します。

関連情報

- TPM v2 および Tang 暗号化モードの詳細は、[ポリシーベースの複号を使用して暗号化ボリュームの自動アンロックの設定](#) を参照してください。

1.4.4. RAID 対応のデータボリュームの設定

ソフトウェア RAID のパーティション設定を有効にして、外部データボリュームを提供できます。OpenShift Container Platform は、データ保護およびフォールトトレランスに対応するために RAID 0、RAID 1、RAID 4、RAID 5、RAID 6、および RAID 10 をサポートします。詳細は、「ディスクのミラーリングについて」を参照してください。



注記

OpenShift Container Platform 4.19 は、インストールドライブ上のソフトウェア RAID をサポートしていません。

前提条件

- インストールノードで OpenShift Container Platform インストールプログラムをダウンロードしている。
- インストールノードに Butane をインストールしている。



注記

Butane は、OpenShift Container Platform が使用するコマンドラインユーティリティーで、マシン設定を作成するための便利で簡略化した構文を提供したり、マシン設定の追加検証を実行したりします。詳細は、[Butane を使用したマシン設定の作成](#) セクションを参照してください。

手順

1. ソフトウェア RAID を使用してデータボリュームを設定する Butane 設定を作成します。

- ミラーリングされた起動ディスクに使用されるのと同じディスク上に RAID 1 を使用してデータボリュームを設定するには、**\$HOME/clusterconfig/raid1-storage.bu** ファイルを作成します。以下に例を示します。

ミラーリングされた起動ディスク上の RAID 1

```
variant: openshift
version: 4.19.0
metadata:
  name: raid1-storage
  labels:
    machineconfiguration.openshift.io/role: worker
boot_device:
  mirror:
    devices:
      - /dev/disk/by-id/scsi-3600508b400105e2100009000000490000
      - /dev/disk/by-id/scsi-SSEAGATE_ST373453LW_3HW1RHM6
storage:
  disks:
    - device: /dev/disk/by-id/scsi-3600508b400105e2100009000000490000
      partitions:
        - label: root-1
          size_mib: 25000 ①
        - label: var-1
    - device: /dev/disk/by-id/scsi-SSEAGATE_ST373453LW_3HW1RHM6
      partitions:
        - label: root-2
          size_mib: 25000 ②
        - label: var-2
  raid:
    - name: md-var
      level: raid1
      devices:
        - /dev/disk/by-partlabel/var-1
        - /dev/disk/by-partlabel/var-2
  filesystems:
    - device: /dev/md/md-var
      path: /var
      format: xfs
      wipe_filesystem: true
      with_mount_unit: true
```

- ① ② データパーティションをブートディスクに追加する場合は、25000 のメビバイトの最小値が推奨されます。値の指定がない場合や、指定した値が推奨される最小値よりも小さい場合は、生成されるルートファイルシステムのサイズが小さ過ぎるため、RHCOS の再インストールでデータパーティションの最初の部分が上書きされる可能性があります。

- セカンダリーディスク上に RAID 1 を使用してデータボリュームを設定するには、**\$HOME/clusterconfig/raid1-alt-storage.bu** ファイルを作成します。以下に例を示します。

セカンダリーディスク上の RAID 1

```

variant: openshift
version: 4.19.0
metadata:
  name: raid1-alt-storage
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  disks:
    - device: /dev/sdc
      wipe_table: true
      partitions:
        - label: data-1
    - device: /dev/sdd
      wipe_table: true
      partitions:
        - label: data-2
  raid:
    - name: md-var-lib-containers
      level: raid1
      devices:
        - /dev/disk/by-partlabel/data-1
        - /dev/disk/by-partlabel/data-2
  filesystems:
    - device: /dev/md/md-var-lib-containers
      path: /var/lib/containers
      format: xfs
      wipe_filesystem: true
      with_mount_unit: true

```

2. 前のステップで作成した Butane 設定から RAID マニフェストを作成し、それを **<installation_directory>/openshift** ディレクトリーに保存します。たとえば、コンピュータノードのマニフェストを作成するには、以下のコマンドを実行します。

```

$ butane $HOME/clusterconfig/<butane_config>.bu -o
<installation_directory>/openshift/<manifest_name>.yaml ❶

```

- ❶ **<butane_config>** および **<manifest_name>** を直前の手順のファイル名に置き換えます。たとえば、セカンダリーディスクの場合は、**raid1-alt-storage.bu** および **raid1-alt-storage.yaml** になります。

3. 今後マニフェストを更新する必要がある場合には、Butane 設定を保存します。
4. 残りの OpenShift Container Platform インストールを継続します。

1.4.5. Intel® Virtual RAID on CPU (VROC) データボリュームの設定

Intel® VROC はハイブリッド RAID の一種で、メンテナンスの一部はハードウェアにオフロードされますが、オペレーティングシステムにはソフトウェア RAID として表示されます。

次の手順では、Intel® VROC 対応 RAID1 を設定します。

前提条件

- Intel® Volume Management Device (VMD) が有効になっているシステムがある。

手順

1. 次のコマンドを実行して、Intel® Matrix Storage Manager (IMSM) RAID コンテナを作成します。

```
$ mdadm -CR /dev/md/imsm0 -e \
  imsm -n2 /dev/nvme0n1 /dev/nvme1n1 ①
```

- ① RAID のデバイス名。この例では、2つのデバイスがリストされています。2つ以上のデバイス名を指定する場合は、**-n** フラグを調整する必要があります。たとえば、3つのデバイスをリストするには、フラグ **-n3** を使用します。

2. コンテナ内に RAID1 ストレージを作成します。

- a. 次のコマンドを実行して、実際の RAID1 ボリュームの前にダミーの RAID0 ボリュームを作成します。

```
$ mdadm -CR /dev/md/dummy -l0 -n2 /dev/md/imsm0 -z10M --assume-clean
```

- b. 次のコマンドを実行して、実際の RAID1 アレイを作成します。

```
$ mdadm -CR /dev/md/coreos -l1 -n2 /dev/md/imsm0
```

- c. 次のコマンドを使用して、RAID0 と RAID1 のメンバーアレイを停止し、ダミーの RAID0 アレイを削除します。

```
$ mdadm -S /dev/md/dummy \
  mdadm -S /dev/md/coreos \
  mdadm --kill-subarray=0 /dev/md/imsm0
```

- d. 次のコマンドを実行して RAID1 アレイを再起動します。

```
$ mdadm -A /dev/md/coreos /dev/md/imsm0
```

3. RAID1 デバイスに RHCOS をインストールします。

- a. 次のコマンドを実行して、IMSM コンテナの UUID を取得します。

```
$ mdadm --detail --export /dev/md/imsm0
```

- b. 次のコマンドを実行して、RHCOS をインストールし、**rd.md.uuid** カーネル引数を含めます。

```
$ coreos-installer install /dev/md/coreos \
  --append-karg rd.md.uuid=<md_UUID> ①
...
```

- ① IMSM コンテナの UUID。

RHCOS をインストールするために必要な追加の **coreos-installer** 引数を含めます。

1.5. CHRONY タイムサービスの設定

chrony タイムサービス (**chronyd**) で使用されるタイムサーバーおよび関連する設定は、**chrony.conf** ファイルのコンテンツを変更し、それらのコンテンツをマシン設定としてノードに渡して設定できます。

手順

1. **chrony.conf** ファイルのコンテンツを含む Butane 設定を作成します。たとえば、ワーカーノードで chrony を設定するには、**99-worker-chrony.bu** ファイルを作成します。



注記

設定ファイルで指定する [Butane のバージョン](#) は、OpenShift Container Platform のバージョンと同じである必要があり、末尾は常に **0** です。たとえば、**4.19.0** などです。Butane の詳細は、「Butane を使用したマシン設定の作成」を参照してください。

```
variant: openshift
version: 4.19.0
metadata:
  name: 99-worker-chrony ❶
  labels:
    machineconfiguration.openshift.io/role: worker ❷
storage:
  files:
    - path: /etc/chrony.conf
      mode: 0644 ❸
      overwrite: true
      contents:
        inline: |
          pool 0.rhel.pool.ntp.org iburst ❹
          driftfile /var/lib/chrony/drift
          makestep 1.0 3
          rtcsync
          logdir /var/log/chrony
```

- ❶ ❷ コントロールプレーンノードでは、これらの両方の場所で **worker** の代わりに **master** を使用します。
- ❸ マシン設定ファイルの **mode** フィールドに 8 進数の値でモードを指定します。ファイルを作成し、変更を適用すると、**mode** は 10 進数の値に変換されます。**oc get mc <mc-name> -o yaml** コマンドで YAML ファイルを確認できます。
- ❹ DHCP サーバーが提供するものなど、有効な到達可能なタイムソースを指定します。



注記

全マシン間の通信の場合、UDP 上の Network Time Protocol (NTP) はポート **123** です。外部 NTP タイムサーバーが設定されている場合は、UDP ポート **123** を開く必要があります。

または、NTP サーバーの **1.rhel.pool.ntp.org**、**2.rhel.pool.ntp.org**、または **3.rhel.pool.ntp.org** のいずれかを指定できます。DHCP サーバーで NTP を使用する場合は、**chrony.conf** ファイルで **sourcedir/run/chrony-dhcp** パラメーターを設定する必要があります。

2. Butane を使用して、ノードに配信される設定を含む **MachineConfig** オブジェクトファイル (**99-worker-chrony.yaml**) を生成します。

```
$ butane 99-worker-chrony.bu -o 99-worker-chrony.yaml
```

3. 以下の 2 つの方法のいずれかで設定を適用します。

- クラスターがまだ起動していない場合は、マニフェストファイルを生成した後に、**MachineConfig** オブジェクトファイルを **<installation_directory>/openshift** ディレクトリーに追加してから、クラスターの作成を続行します。
- クラスターがすでに実行中の場合は、ファイルを適用します。

```
$ oc apply -f ./99-worker-chrony.yaml
```

1.6. 関連情報

- Butane の詳細は、[Butane を使用したマシン設定の作成](#) を参照してください。
- FIPS サポートの詳細は、[FIPS 暗号のサポート](#) を参照してください。

第2章 ファイアウォールの設定

ファイアウォールを使用する場合は、OpenShift Container Platform が機能するために必要なサイトにアクセスできるように設定する必要があります。一部のサイトには常にアクセスを付与する必要があります。また、クラスターをホストするために Red Hat Insights、Telemetry サービス、クラウドを使用する場合や、特定のビルドストラテジーを利用する場合は、追加でアクセスを付与する必要があります。

2.1. OPENSIFT CONTAINER PLATFORM のファイアウォールの設定

OpenShift Container Platform をインストールする前に、ファイアウォールを、OpenShift Container Platform が必要とするサイトへのアクセスを付与するように設定する必要があります。ファイアウォールを使用する場合は、OpenShift Container Platform が機能するために必要なサイトにアクセスできるように、ファイアウォールに追加の設定を行います。

ワーカーノードと比較して、コントローラーノードのみで実行されるサービスには、特別な設定上の考慮事項はありません。



注記

ご使用の環境で OpenShift Container Platform クラスターの前に専用のロードバランサーがある場合は、ファイアウォールとロードバランサーの間の許可リストを確認して、クラスターに対する不要なネットワーク制限を回避してください。

手順

1. ファイアウォールの許可リストに次のレジストリー URL を設定します。

URL	ポート	機能
registry.redhat.io	443	コアコンテナイメージを指定します。
access.redhat.com	443	コンテナクライアントが registry.access.redhat.com から取得したイメージを検証するのに必要な署名ストアをホストします。ファイアウォール環境では、このリソースが許可リストに含まれていることを確認してください。
registry.access.redhat.com	443	コアコンテナイメージを含め、Red Hat Ecosystem Catalog に保存されているすべてのコンテナイメージをホストします。
quay.io	443	コアコンテナイメージを指定します。
cdn.quay.io	443	コアコンテナイメージを指定します。
cdn01.quay.io	443	コアコンテナイメージを指定します。
cdn02.quay.io	443	コアコンテナイメージを指定します。

URL	ポート	機能
cdn03.quay.io	443	コアコンテナイメージを指定します。
cdn04.quay.io	443	コアコンテナイメージを指定します。
cdn05.quay.io	443	コアコンテナイメージを指定します。
cdn06.quay.io	443	コアコンテナイメージを指定します。
sso.redhat.com	443	https://console.redhat.com サイトは、 sso.redhat.com からの認証を使用します。
icr.io	443	IBM Cloud Pak コンテナイメージを提供します。このドメインは、IBM Cloud Paks を使用する場合にのみ必要です。
cp.icr.io	443	IBM Cloud Pak コンテナイメージを提供します。このドメインは、IBM Cloud Paks を使用する場合にのみ必要です。

- 許可リストでは、**cdn.quay.io** と **cdn0[1-6].quay.io** の代わりにワイルドカード ***.quay.io** を使用できます。
 - ワイルドカード ***.access.redhat.com** を使用すると、設定を簡素化し、**registry.access.redhat.com** を含むすべてのサブドメインを許可できます。
 - quay.io** などのサイトを許可リストに追加するには、***.quay.io** などのワイルドカードエントリを拒否リストに加えないでください。ほとんどの場合、イメージレジストリーはコンテンツ配信ネットワーク (CDN) を使用してイメージを提供します。ファイアウォールがアクセスをブロックすると、最初のダウンロード要求が **cdn01.quay.io** などのホスト名にリダイレクトされるときに、イメージのダウンロードが拒否されます。
- ファイアウォールの許可リストを設定し、ビルドに必要な言語またはフレームワークのリソースを提供するサイトをリストに含めます。
 - Telemetry を無効にしていない場合は、以下の URL へのアクセスを許可して Red Hat Insights にアクセスできるようにする必要があります。

URL	ポート	機能
cert-api.access.redhat.com	443	Telemetry で必須
api.access.redhat.com	443	Telemetry で必須
infogw.api.openshift.com	443	Telemetry で必須

URL	ポート	機能
console.redhat.com	443	Telemetry および insights-operator で必須

4. Alibaba Cloud、Amazon Web Services (AWS)、Microsoft Azure、または Google Cloud Platform (GCP) を使用してクラスターをホストする場合は、クラウドプロバイダー API およびそのクラウドの DNS を提供する URL へのアクセス権を付与する必要があります。

クラウド	URL	ポート	機能
Alibaba	*.aliyuncs.com	443	Alibaba Cloud のサービスとリソースにアクセスするために必要です。 Alibaba の endpoints_config.go ファイル を参照して、使用するリージョンを許可する正確なエンドポイントを確認してください。
AWS	aws.amazon.com	443	AWS 環境でのクラスターのインストールおよび管理に使用されます。
	*.amazonaws.com または、AWS API にワイルドカードを使用しない場合は、次の URL を許可リストに含める必要があります。	443	AWS サービスおよびリソースへのアクセスに必要です。AWS ドキュメントの AWS Service Endpoints を参照して、使用するリージョンを許可する正確なエンドポイントを確認してください。
	ec2.amazonaws.com	443	AWS 環境でのクラスターのインストールおよび管理に使用されます。
	events.amazonaws.com	443	AWS 環境でのクラスターのインストールおよび管理に使用されます。
	iam.amazonaws.com	443	AWS 環境でのクラスターのインストールおよび管理に使用されます。
	route53.amazonaws.com	443	AWS 環境でのクラスターのインストールおよび管理に使用されます。
	*.s3.amazonaws.com	443	AWS 環境でのクラスターのインストールおよび管理に使用されます。
	*.s3.<aws_region>.amazonaws.com	443	AWS 環境でのクラスターのインストールおよび管理に使用されます。

クラウド	URL	ポート	機能
	*.s3.dualstack. <aws_region>.amazonaws.com	443	AWS 環境でのクラスターのインストールおよび管理に使用されます。
	sts.amazonaws.com	443	AWS 環境でのクラスターのインストールおよび管理に使用されます。
	sts. <aws_region>.amazonaws.com	443	AWS 環境でのクラスターのインストールおよび管理に使用されます。
	tagging.us-east-1.amazonaws.com	443	AWS 環境でのクラスターのインストールおよび管理に使用されます。このエンドポイントは、クラスターがデプロイされているリージョンに関係なく、常に us-east-1 です。
	ec2. <aws_region>.amazonaws.com	443	AWS 環境でのクラスターのインストールおよび管理に使用されます。
	elasticloadbalancing. <aws_region>.amazonaws.com	443	AWS 環境でのクラスターのインストールおよび管理に使用されます。
	servicequotas. <aws_region>.amazonaws.com	443	必須。サービスをデプロイするためのクォータを確認するのに使用されます。
	tagging. <aws_region>.amazonaws.com	443	タグの形式で AWS リソースに関するメタデータを割り当てることができます。
	*.cloudfront.net	443	CloudFront へのアクセスを提供するために使用されます。AWS Security Token Service (STS) およびプライベート S3 バケットを使用する場合は、CloudFront へのアクセスを提供する必要があります。
GCP	*.googleapis.com	443	GCP サービスおよびリソースへのアクセスに必要です。GCP ドキュメントの Cloud Endpoints を参照して、お使いの API を許可するエンドポイントを確認してください。
	accounts.google.com	443	GCP アカウントへのアクセスに必要です。

クラウド	URL	ポート	機能
Microsoft Azure	management.azure.com	443	Microsoft Azure のサービスとリソースにアクセスするために必要です。Microsoft Azure ドキュメントの Microsoft Azure REST API reference を参照して、お使いの API を許可するエンドポイントを確認してください。
	*.blob.core.windows.net	443	Ignition ファイルのダウンロードに必要です。
	login.microsoftonline.com	443	Microsoft Azure のサービスとリソースにアクセスするために必要です。Microsoft Azure ドキュメントの Azure REST API reference を参照して、お使いの API を許可するエンドポイントを確認してください。

5. 以下の URL を許可リストに指定します。

URL	ポート	機能
*.apps.<cluster_name>.<base_domain>	443	Ingress ワイルドカードをインストール時に設定しない限り、デフォルトのクラスタールートへのアクセスに必要です。
api.openshift.com	443	クラスタトークンの両方が必要であり、クラスターに更新が利用可能かどうかを確認するために必要です。
console.redhat.com	443	クラスタトークンに必要です。
mirror.openshift.com	443	ミラーリングされたインストールのコンテンツおよびイメージへのアクセスに必要です。Cluster Version Operator には単一の機能ソースのみが必要ですが、このサイトはリリースイメージ署名のソースでもあります。
quayio-production-s3.s3.amazonaws.com	443	AWS で Quay イメージコンテンツにアクセスするために必要です。
rhcos.mirror.openshift.com	443	Red Hat Enterprise Linux CoreOS (RHCOS) イメージをダウンロードするために必要です。

URL	ポート	機能
sso.redhat.com	443	https://console.redhat.com サイトは、 sso.redhat.com からの認証を使用します。
storage.googleapis.com/openshift-release	443	リリースイメージ署名のソース。ただし、Cluster Version Operator には単一の機能ソースのみが必要です。

Operator にはヘルスチェックを実行するためのルートアクセスが必要です。具体的には、認証および Web コンソール Operator は 2 つのルートに接続し、ルートが機能することを確認します。クラスター管理者として操作を実行しており、***.apps.<cluster_name>.<base_domain>** を許可しない場合は、これらのルートを許可します。

- **oauth-openshift.apps.<cluster_name>.<base_domain>**
- **canary-openshift-ingress-canary.apps.<cluster_name>.<base_domain>**
- **console-openshift-console.apps.<cluster_name>.<base_domain>**、またはフィールドが空でない場合に **consoles.operator/cluster** オブジェクトの **spec.route.hostname** フィールドに指定されるホスト名

6. オプションのサードパーティーコンテンツに対する次の URL を許可リストに追加します。

URL	ポート	機能
registry.connect.redhat.com	443	すべてのサードパーティーのイメージと認定 Operator が必要です。
rhc4tp-prod-z8cxf-image-registry-us-east-1-evenkyleffocxqvofrk.s3.dualstack.us-east-1.amazonaws.com	443	registry.connect.redhat.com でホストされているコンテナイメージにアクセスできます。
oso-rhc4tp-docker-registry.s3-us-west-2.amazonaws.com	443	Sonatype Nexus、F5 Big IP Operator が必要です。

7. デフォルトの Red Hat Network Time Protocol (NTP) サーバーを使用する場合は、以下の URL を許可します。

- **1.rhel.pool.ntp.org**
- **2.rhel.pool.ntp.org**
- **3.rhel.pool.ntp.org**



注記

デフォルトの Red Hat NTP サーバーを使用しない場合は、プラットフォームの NTP サーバーを確認し、ファイアウォールでこれを許可します。

関連情報

- [AWS STS の OpenID Connect の要件](#)

2.2. OPENSIFT CONTAINER PLATFORM ネットワークフローマトリックス

このセクションのネットワークフローマトリックスでは、以下に示す環境の OpenShift Container Platform サービスへの Ingress フローを説明します。

- ベアメタル上の OpenShift Container Platform
- 他のプラットフォームと連携したシングルノード OpenShift
- Amazon Web Services (AWS) 上の OpenShift Container Platform
- AWS 上のシングルノード OpenShift

適切なネットワークフローマトリックスの情報を使用して、お客様の環境の Ingress トラフィックを管理してください。Ingress トラフィックを重要なフローに制限して、ネットワークセキュリティを強化できます。

さらに、ベアメタル環境とクラウド環境の両方の Ingress トラフィックを管理する場合は、次の動的ポート範囲を考慮してください。

- **9000-9999**: ホストレベルのサービス
- **30000-32767**: Kubernetes ノードポート
- **49152-65535**: 動的ポートまたはプライベートポート

環境の完全な raw CSV コンテンツを表示またはダウンロードするには、次のリソースを参照してください。

- [ベアメタル上の OpenShift Container Platform](#)
- [他のプラットフォームと連携したシングルノード OpenShift](#)
- [AWS 上の OpenShift Container Platform](#)
- [AWS 上のシングルノード OpenShift](#)



注記

ネットワークフローマトリックスでは、ベースの OpenShift Container Platform またはシングルノード OpenShift インストールの Ingress トラフィックフローを説明します。マトリックスは、Hosted Control Plane、Red Hat build of MicroShift、またはスタンドアロンクラスターには適用されません。

2.2.1. ベースネットワークフロー

次のマトリックスでは、OpenShift Container Platform サービスへのベース Ingress フローを説明します。



注記

シングルノード OpenShift クラスターへのベース Ingress フローは、**コントロールプレーンノードのベースフロー** マトリックスのみを参照してください。

表2.1 コントロールプレーンノードのベースフロー

方向	プロトコル	ポート	namespace	サービス	Pod	コンテナー	ノードのロール	任意
Ingress	TCP	22	Host system service	sshd			master	TRUE
Ingress	TCP	111	Host system service	rpcbind			master	TRUE
Ingress	TCP	2379	openshift-etcd	etcd	etcd	etcdctl	master	FALSE
Ingress	TCP	2380	openshift-etcd	healthz	etcd	etcd	master	FALSE
Ingress	TCP	6080	openshift-kube-apiserver		kube-apiserver	kube-apiserver-insecure-readyz	master	FALSE
Ingress	TCP	6443	openshift-kube-apiserver	apiserver	kube-apiserver	kube-apiserver	master	FALSE
Ingress	TCP	8080	openshift-network-operator		network-operator	network-operator	master	FALSE
Ingress	TCP	8798	openshift-machine-config-operator	machine-config-daemon	machine-config-daemon	machine-config-daemon	master	FALSE

方向	プロトコル	ポート	namespace	サービス	Pod	コンテナー	ノードのロール	任意
Ingress	TCP	9001	openshift-machine-config-operator	machine-config-daemon	machine-config-daemon	kube-rbac-proxy	master	FALSE
Ingress	TCP	9099	openshift-cluster-version	cluster-version-operator	cluster-version-operator	cluster-version-operator	master	FALSE
Ingress	TCP	9100	openshift-monitoring	node-exporter	node-exporter	kube-rbac-proxy	master	FALSE
Ingress	TCP	9103	openshift-ovn-kubernetes	ovn-kubernetes-node	ovnkube-node	kube-rbac-proxy-node	master	FALSE
Ingress	TCP	9104	openshift-network-operator	metrics	network-operator	network-operator	master	FALSE
Ingress	TCP	9105	openshift-ovn-kubernetes	ovn-kubernetes-node	ovnkube-node	kube-rbac-proxy-ovn-metrics	master	FALSE
Ingress	TCP	9107	openshift-ovn-kubernetes	egressip-node-healthcheck	ovnkube-node	ovnkube-controller	master	FALSE
Ingress	TCP	9108	openshift-ovn-kubernetes	ovn-kubernetes-control-plane	ovnkube-control-plane	kube-rbac-proxy	master	FALSE

方向	プロトコル	ポート	namespace	サービス	Pod	コンテナー	ノードのロール	任意
Ingress	TCP	9192	openshift-cluster-machine-approver	machine-approver	machine-approver	kube-rbac-proxy	master	FALSE
Ingress	TCP	9258	openshift-cloud-controller-manager-operator	machine-approver	cluster-cloud-controller-manager	cluster-cloud-controller-manager	master	FALSE
Ingress	TCP	9637	openshift-machine-config-operator	kube-rbac-proxy-crio	kube-rbac-proxy-crio	kube-rbac-proxy-crio	master	FALSE
Ingress	TCP	9978	openshift-etcd	etcd	etcd	etcd-metrics	master	FALSE
Ingress	TCP	9979	openshift-etcd	etcd	etcd	etcd-metrics	master	FALSE
Ingress	TCP	9980	openshift-etcd	etcd	etcd	etcd	master	FALSE
Ingress	TCP	10250	Host system service	kubelet			master	FALSE
Ingress	TCP	10256	openshift-ovn-kubernetes	ovnkube	ovnkube	ovnkube-controller	master	FALSE
Ingress	TCP	10257	openshift-kube-controller-manager	kube-controller-manager	kube-controller-manager	kube-controller-manager	master	FALSE

方向	プロトコル	ポート	namespace	サービス	Pod	コンテナ	ノードのロール	任意
Ingress	TCP	10259	openshift-kube-scheduler	scheduler	openshift-kube-scheduler	kube-scheduler	master	FALSE
Ingress	TCP	10357	openshift-kube-apiserver	openshift-kube-apiserver-healthz	kube-apiserver	kube-apiserver-check-endpoints	master	FALSE
Ingress	TCP	17697	openshift-kube-apiserver	openshift-kube-apiserver-healthz	kube-apiserver	kube-apiserver-check-endpoints	master	FALSE
Ingress	TCP	22623	openshift-machine-config-operator	machine-config-server	machine-config-server	machine-config-server	master	FALSE
Ingress	TCP	22624	openshift-machine-config-operator	machine-config-server	machine-config-server	machine-config-server	master	FALSE
Ingress	UDP	111	Host system service	rpcbind			master	TRUE

表2.2 ワーカーノードのベースフロー

方向	プロトコル	ポート	namespace	サービス	Pod	コンテナ	ノードのロール	任意
Ingress	TCP	22	Host system service	sshd			worker	TRUE
Ingress	TCP	111	Host system service	rpcbind			worker	TRUE

方向	プロトコル	ポート	namespace	サービス	Pod	コンテナー	ノードのロール	任意
Ingress	TCP	8798	openshift-machine-config-operator	machine-config-daemon	machine-config-daemon	machine-config-daemon	worker	FALSE
Ingress	TCP	9001	openshift-machine-config-operator	machine-config-daemon	machine-config-daemon	kube-rbac-proxy	worker	FALSE
Ingress	TCP	9100	openshift-monitoring	node-exporter	node-exporter	kube-rbac-proxy	worker	FALSE
Ingress	TCP	9103	openshift-ovn-kubernetes	ovn-kubernetes-node	ovnkube-node	kube-rbac-proxy-node	worker	FALSE
Ingress	TCP	9105	openshift-ovn-kubernetes	ovn-kubernetes-node	ovnkube-node	kube-rbac-proxy-ovn-metrics	worker	FALSE
Ingress	TCP	9107	openshift-ovn-kubernetes	egressip-node-healthcheck	ovnkube-node	ovnkube-controller	worker	FALSE
Ingress	TCP	9637	openshift-machine-config-operator	kube-rbac-proxy-crio	kube-rbac-proxy-crio	kube-rbac-proxy-crio	worker	FALSE
Ingress	TCP	10250	Host system service	kubelet			worker	FALSE
Ingress	TCP	10256	openshift-ovn-kubernetes	ovnkube	ovnkube	ovnkube-controller	worker	FALSE

方向	プロトコル	ポート	namespace	サービス	Pod	コンテナー	ノードのロール	任意
Ingress	UDP	111	Host system service	rpcbind			worker	TRUE

2.2.2. ベアメタル上の OpenShift Container Platform の追加ネットワークフロー

ベースネットワークフローとは別に、次のマトリックスで、ベアメタル上の OpenShift Container Platform に固有の OpenShift Container Platform サービスへの Ingress フローを説明します。

表2.3 ベアメタル上の OpenShift Container Platform

方向	プロトコル	ポート	namespace	サービス	Pod	コンテナー	ノードのロール	任意
Ingress	TCP	53	openshift-dns	dns-default	dnf-default	dns	master	FALSE
Ingress	TCP	6180	openshift-machine-api	metal3-state	metal3	metal3-httpd	master	FALSE
Ingress	TCP	6183	openshift-machine-api	metal3-state	metal3	metal3-httpd	master	FALSE
Ingress	TCP	6385	openshift-machine-api	metal3-state	metal3	metal3-httpd	master	FALSE
Ingress	TCP	6388	openshift-machine-api	metal3-state	metal3	metal3-httpd	master	FALSE
Ingress	TCP	9444	openshift-kni-infra		haproxy	haproxy	master	FALSE
Ingress	TCP	9445	openshift-kni-infra		haproxy	haproxy	master	FALSE

方向	プロトコル	ポート	namespace	サービス	Pod	コンテナー	ノードのロール	任意
Ingress	TCP	9454	openshift-kni-infra		haproxy	haproxy	master	FALSE
Ingress	TCP	18080	openshift-kni-infra		coredns	coredns	master	FALSE
Ingress	UDP	53	openshift-dns	dns-default	dnf-default	dns	master	FALSE
Ingress	UDP	6081	openshift-ovn-kubernetes	ovn-kubernetes-geneve			master	FALSE
Ingress	TCP	53	openshift-dns	dns-default	dnf-default	dns	worker	FALSE
Ingress	TCP	80	openshift-ingress	router-internal-default	router-default	router	worker	FALSE
Ingress	TCP	443	openshift-ingress	router-internal-default	router-default	router	worker	FALSE
Ingress	TCP	1936	openshift-ingress	router-internal-default	router-default	router	worker	FALSE
Ingress	TCP	18080	openshift-kni-infra		coredns	coredns	worker	FALSE
Ingress	UDP	53	openshift-dns	dns-default	dnf-default	dns	worker	FALSE
Ingress	UDP	6081	openshift-ovn-kubernetes	ovn-kubernetes-geneve			worker	FALSE

2.2.3. 他のプラットフォームとシングルノード OpenShift の追加のネットワークフロー

基本ネットワークフローに加えて、次のマトリックスは、インストール manifests で **platform: none** が設定されたシングルノード OpenShift に固有の OpenShift Container Platform サービスへの Ingress フローを説明しています。

表2.4 他のプラットフォームと連携したシングルノード OpenShift

方向	プロトコル	ポート	namespace	サービス	Pod	コンテナー	ノードのロール	任意
Ingress	TCP	80	openshift-ingress	router-internal-default	router-default	router	master	FALSE
Ingress	TCP	443	openshift-ingress	router-internal-default	router-default	router	master	FALSE
Ingress	TCP	1936	openshift-ingress	router-internal-default	router-default	router	master	FALSE

2.2.4. AWS 上の OpenShift Container Platform の追加ネットワークフロー

ベースネットワークフローとは別に、次のマトリックスで、AWS 上の OpenShift Container Platform に固有の OpenShift Container Platform サービスへの Ingress フローを説明します。

表2.5 AWS 上の OpenShift Container Platform

方向	プロトコル	ポート	namespace	サービス	Pod	コンテナー	ノードのロール	任意
Ingress	TCP	10258	openshift-cloud-controller-manager-operator	cloud-controller	cloud-controller-manager	cloud-controller-manager	master	FALSE
Ingress	TCP	80	openshift-ingress	router-default	router-default	router	worker	FALSE
Ingress	TCP	443	openshift-ingress	router-default	router-default	router	worker	FALSE
Ingress	UDP	6081	openshift-ovn-kubernetes	ovn-kubernetes-geneve			worker	FALSE

2.2.5. AWS 上のシングルノード OpenShift の追加ネットワークフロー

ベースネットワークフローとは別に、次のマトリックスでは、AWS 上のシングルノード OpenShift に固有の OpenShift Container Platform サービスへの Ingress フローを説明します。

表2.6 AWS 上のシングルノード OpenShift

方向	プロトコル	ポート	namespace	サービス	Pod	コンテナー	ノードのロール	任意
Ingress	TCP	80	openshift-ingress	router-default	router-default	router	master	FALSE
Ingress	TCP	443	openshift-ingress	router-default	router-default	router	master	FALSE
Ingress	TCP	10258	openshift-cloud-controller-manager-operator	cloud-controller	cloud-controller-manager	cloud-controller-manager	master	FALSE