



# OpenShift Container Platform 3.11

## CRI-O ランタイム

CRI-O ランタイムガイド





## 法律上の通知

Copyright © 2020 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

CRI-O の使用方法を理解する

---

## 目次

<b>第1章 CRI-O コンテナエンジンの使用</b> .....	<b>3</b>
1.1. CRI-O について	3
1.2. CRI-O の取得	4
1.2.1. 新規 OpenShift Container Platform クラスターでの CRI-O のインストール	4
1.2.2. CRI-O ノードの OpenShift Container Platform クラスターへの追加	6
1.3. CRI-O の設定	6
1.3.1. CRI-O ストレージの設定	7
1.3.2. CRI-O ネットワークの設定	8
1.4. CRI-O のトラブルシューティング	8
1.4.1. CRI-O の一般的なヘルスチェック	9
1.4.2. CRI-O ログの検査	10
1.4.2.1. cri-o および origin-node ログの確認	10
1.4.2.2. CRI-O のデバッグをオンにする	11
1.4.3. CRI-O Pod およびコンテナのトラブルシューティング	12
1.4.3.1. イメージ、Pod、およびコンテナの一覧表示	12
1.4.3.2. イメージ、Pod、およびコンテナの調査	14
その他のリソース	16



# 第1章 CRI-O コンテナエンジンの使用

CRI-O は、オープンソースのコミュニティ主導型のコンテナエンジンです。その主な目的は、OpenShift Container Platform などの、Kubernetes 実装のコンテナエンジンとしての Docker サービスを置き換えることにあります。

CRI-O の使用を開始されたい場合には、本書を参照してください。本書では、OpenShift Container Platform のインストール時に CRI-O をインストールする方法や、CRI-O ノードを既存の OpenShift Container Platform クラスタに追加する方法について説明します。また本書には、CRI-O エンジンの設定方法やトラブルシューティングの方法についての情報も記載しています。

## 1.1. CRI-O について

CRI-O コンテナエンジンは、OCI (Open Container Initiative) と互換性のあるランタイムを実行するための安定性があり、よりセキュアで高性能のプラットフォームを提供します。CRI-O コンテナエンジンを使用し、runc、デフォルト OCI ランタイム、または Kata Containers などの OCI と互換性のあるランタイムを使用してコンテナおよび Pod を起動できます。CRI-O は、OpenShift Container Platform および Kubernetes の Kubernetes Container Runtime Interface (CRI) を実装するコンテナエンジンとして機能し、Docker サービスを置き換えることを目的としています。

CRI-O は効率的なコンテナエンジンを提供しますが、他のコンテナ機能は革新的で独立したコマンドの別個のセットとして実装されます。このアプローチにより、Kubernetes ベースのインストール用のコンテナエンジンとして機能するという CRI-O の主な目的を達成すると同時に、コンテナの管理機能の開発を独自のペースで進めていくことができます。

CRI-O の安定性は、これが Kubernetes のメジャーリリースおよびマイナーリリースと連動して開発され、テストされ、リリースされていること、および OCI 標準に準拠しているということによって実現されています。たとえば、CRI-O 1.11 は Kubernetes 1.11 と連動します。CRI-O の範囲は CRI (Container Runtime Interface) に関連付けられています。CRI は、コンテナエンジンから Kubernetes サービス (kubelet) が必要とするものを抽出し、標準化しています。CRI チームは、複数のコンテナエンジンの開発が開始される過程で Kubernetes コンテナエンジンの要件の安定化を促進することを目的としてこれを実施しました。

CRI-O にコマンドラインで直接接続する必要はほとんどありません。ただし、テストおよびモニタリングのために CRI-O への完全アクセスを提供し、CRI-O が提供しないものの Docker で提供されることが予想される機能を提供するために、コンテナ関連のコマンドラインツールを利用できます。これらのツールは、**docker** コマンドおよびサービスで利用可能な機能を置き換え、拡張します。ツールには以下が含まれます。

- **crictl**: トラブルシューティングを行い、CRI-O コンテナエンジンと直接連動させるために使用します。
- **runc**: コンテナイメージを実行するために使用します。
- **podman**: コンテナエンジンの外部で Pod およびコンテナイメージを管理 (run、stop、start、ps、attach、exec など) するために使用します。
- **buildah**: コンテナイメージをビルドし、プッシュし、これに署名するために使用します。
- **skopeo**: イメージをコピーし、検査し、削除し、これに署名するために使用します。

一部の Docker 機能は、CRI-O ではなく他のツールに組み込まれます。たとえば、**podman** は、数多くの **docker** コマンド機能とのコマンドラインの互換性を提供し、Pod を管理できるようにそれらの機能を拡張します。**podman** でコンテナまたは Pod を実行するためにコンテナエンジンが必要になることはありません。

コンテナエンジンで必要とされないコンテナイメージのビルド、プッシュおよび署名機能は、**buildah** コマンドで利用できます。**docker** に代わるこれらのコマンドについての詳細は、「[Finding, Running and Building Containers without Docker](#)」を参照してください。

## 1.2. CRI-O の取得

CRI-O はスタンドアロンのコンテナエンジンとしてサポートされていません。CRI-O は、OpenShift Container Platform などの Kubernetes インストールのコンテナエンジンとして使用する必要があります。Kubernetes または OpenShift Container Platform なしでコンテナを実行するには、[podman](#) を使用します。

CRI-O コンテナエンジンを OpenShift Container Platform クラスターで使用できるように設定するには、以下を実行できます。

- CRI-O を新規の OpenShift Container Platform クラスターと共にインストールする。または、
- ノードを既存クラスターに追加し、CRI-O をノードのコンテナエンジンとして特定します。CRI-O および Docker ノードの両方を同じクラスター上に配置できます。

以下のセクションでは、CRI-O を新規の OpenShift Container Platform クラスターでインストールする方法を説明します。

### 1.2.1. 新規 OpenShift Container Platform クラスターでの CRI-O のインストール

CRI-O は、インストール時にノードごとに OpenShift Container Platform ノードのコンテナエンジンとして選択できます。以下では、OpenShift Container Platform のインストール時に CRI-O コンテナエンジンを有効にする方法について知っておく必要のあるいくつかの点を説明します。

- 以前は、ノード上で CRI-O を使用するには Docker コンテナエンジンも利用可能な状態である必要がありました。OpenShift Container Platform 3.10 以降では、Docker コンテナエンジンはいずれの場合でも不要になりました。CRI-O のみが利用可能なノードを OpenShift Container Platform クラスターで使用することができます。ただし、ビルドおよびプッシュ操作を実行するノードでは、依然として Docker コンテナエンジンを CRI-O と共にインストールしておく必要があります。
- CRI-O コンテナの使用による CRI-O の有効化はサポートされなくなりました。CRI-O の rpm ベースのインストールが必要です。

以下の手順では、「[インベントリーファイルの設定](#)」で説明されているように Ansible インベントリーファイルを使用して OpenShift Container Platform をインストールすることを前提としています。



#### 注記

CRI-O をコンテナエンジンとして使用し、**/var/lib/docker** を OpenShift Container Platform ノードの別個のマウントポイントとして設定しないでください。CRI-O ノードのデプロイ時に、インストーラーは **/var/lib/docker** を **/var/lib/containers** のシンボリックリンクにしようとします。このアクションは、既存の **/var/lib/docker** を削除してシンボリックリンクを作成することはできないために失敗します。

1. OpenShift Container Platform Ansible Playbook がインストールされた状態で、適切なインベントリーファイルを編集して CRI-O を有効にします。
2. 選択されたインベントリーファイルで CRI-O 設定を見つけます。OpenShift Container Platform インストール時に CRI-O コンテナエンジンをノードにインストールするには、Ansible インベントリーファイルの [OSEv3:vars] セクションを確認します。CRI-O 設定のセク



ションには以下が含まれる場合があります。

```
[OSEv3:vars]
...
# Install and run cri-o.
#openshift_use_crio=False
#openshift_use_crio_only=False
# The following two variables are used when openshift_use_crio is True
# and cleans up after builds that pass through docker. When openshift_use_crio is True
# these variables are set to the defaults shown. You may override them here.
# NOTE: You will still need to tag cri-o nodes with your given label(s)!
# Enable docker garbage collection when using cri-o
#openshift_crio_enable_docker_gc=True
# Node Selectors to run the garbage collection
#openshift_crio_docker_gc_node_selector={'runtime': 'cri-o'}
```

3. CRI-O 設定を有効にします。CRI-O のみを有効にするか、または Docker と共に CRI-O を有効にするかのいずれかを選択できます。以下の設定は、ノードのコンテナエンジンとしての CRI-O および Docker を許可し、overlay2 ストレージが設定されたノードで Docker ガベージコレクションを有効にします。



### 注記

CRI-O ノードでコンテナをビルドできるようにするには、Docker コンテナエンジンをインストールしておく必要があります。CRI-O のみが利用可能なノードを使用する必要がある場合も、これに対応することは可能であり、単純に他のノードにコンテナのビルドを実行させることができます。

```
[OSEv3:vars]
...
openshift_use_crio=True
openshift_use_crio_only=False
openshift_crio_enable_docker_gc=True
```

4. 各ノードの `openshift_node_group_name` を CRI-O ランタイム用に `kubelet` を設定する `configmap` に設定します。すべてのデフォルトノードグループに対応する CRI-O `configmap` があります。「[ノードグループおよびホストマッピングの定義](#)」では、ノードグループおよびマッピングを詳細に説明しています。

```
[nodes]
ocp-crio01 openshift_node_group_name='node-config-all-in-one-crio'
ocp-docker01 openshift_node_group_name='node-config-all-in-one'
```

これにより、必要な CRI-O パッケージが自動的にインストールされます。

結果として作成される OpenShift Container Platform 設定は OpenShift Container Platform インストールのノードで CRI-O コンテナエンジンを実行します。`oc` コマンドを使用してノードのステータスを確認し、CRI-O を実行しているノードを特定します。

```
$ oc get nodes -o wide
NAME          STATUS ROLES          AGE ... CONTAINER-RUNTIME
ocp-crio01    Ready  compute,infra,master 16d ... cri-o://1.11.5
ocp-docker01  Ready  compute,infra,master 16d ... docker://1.13.1
```

## 1.2.2. CRI-O ノードの OpenShift Container Platform クラスターへの追加

OpenShift Container Platform は、docker コンテナエンジンの使用から CRI-O の使用へのノードの直接のアップグレードをサポートしません。既存の OpenShift Container Platform クラスターを CRI-O を使用するようにアップグレードするには、以下を実行します。

- CRI-O コンテナエンジンを使用するように設定されているノードをスケールアップする。
- CRI-O ノードが予想通りに実行されることを確認する。
- 必要に応じて CRI-O ノードを追加する。
- クラスターが安定していく過程で Docker ノードをスケールダウンする。

CRI-O コンテナエンジンでノードを作成する際に実行するアクションを確認するには、「[Upgrading to CRI-O with Ansible](#)」を参照してください。



### 注記

OpenShift Container Platform クラスター全体を OpenShift Container Platform 3.10 以降にアップグレードする場合や、コンテナ化されたバージョンの CRI-O がノードで実行されている場合、CRI-O コンテナはそのノードから削除され、CRI-O rpm がインストールされます。CRI-O サービスはそれ以降 systemd サービスとして実行されます。詳細は、[BZ#1618425](#) を参照してください。

## 1.3. CRI-O の設定

CRI-O は OpenShift Container Platform でデプロイされ、アップグレードされ、管理されることが意図されているため、CRI-O 設定ファイルの変更は OpenShift Container Platform で実行するか、または CRI-O のテストまたはトラブルシューティングを目的として実行する必要があります。実行中の OpenShift Container Platform ノードでは、ほとんどの CRI-O 設定内容は `/etc/crio/crio.conf` ファイルに保持されます。

`crio.conf` ファイルの設定は、ストレージ、リッスンしているソケット、ランタイム機能、およびネットワークが CRI-O に設定される方法を定義します。以下はデフォルトの `crio.conf` ファイルの例です (ファイル自体を調べて、これらの設定について記述しているコメントを確認できます)。

```
[crio]
root = "/var/lib/containers/storage"
runroot = "/var/run/containers/storage"
storage_driver = "overlay"
storage_option = [
    "overlay.override_kernel_check=1",
]

[crio.api]
listen = "/var/run/crio/crio.sock"
stream_address = ""
stream_port = "10010"
file_locking = true

[crio.runtime]
runtime = "/usr/bin/runc"
runtime_untrusted_workload = ""
default_workload_trust = "trusted"
```

```

no_pivot = false
common = "/usr/libexec/crio/common"
common_env = [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
]
selinux = true
seccomp_profile = "/etc/crio/seccomp.json"
apparmor_profile = "crio-default"
cgroup_manager = "systemd"
hooks_dir_path = "/usr/share/containers/oci/hooks.d"
default_mounts = [
    "/usr/share/rhel/secrets:/run/secrets",
]
pids_limit = 1024
enable_shared_pid_namespace = false
log_size_max = 52428800

[crio.image]
default_transport = "docker://"
pause_image = "docker.io/openshift/origin-pod:v3.11"
pause_command = "/usr/bin/pod"
signature_policy = ""
image_volumes = "mkdir"
insecure_registries = [
    ""
]
registries = [
    "docker.io"
]

[crio.network]
network_dir = "/etc/cni/net.d/"
plugin_dir = "/opt/cni/bin"

```

以下のセクションでは、異なる CRI-O 設定が **crio.conf** ファイルで使用される可能性のある方法について説明しています。

### 1.3.1. CRI-O ストレージの設定

OverlayFS2 は、コンテナエンジンとして CRI-O または Docker のいずれを使用している場合でも、OpenShift Container Platform の推奨される (およびデフォルトの) ストレージドライバーになります。利用可能なストレージデバイスの詳細は、「[Choosing a graph driver](#)」を参照してください。



#### 注記

devicemapper は CRI-O のサポートされているストレージ機能ですが、CRI-O のガベージコレクション機能はまだ devicemapper と連動しないため、実稼働環境での使用には推奨されません。また、CRI-O および **podman** の両方でコンテナストレージを使用する方法に関する他の devicemapper の問題については、[BZ1625394](#) および [BZ1623944](#) を参照してください。

CRI-O ストレージについて知っておく必要のある点には、CRI-O ストレージが実行する以下の点が含まれます。

- 付随するレイヤーと共に、各コンテナのルートファイルシステムを格納することによりイメージを保持する。
- Docker サービスで 사용되는ものと同じストレージレイヤーを組み込む。
- **container-storage-setup** を使用してコンテナストレージエリアを管理する。
- **/etc/containers/storage.conf** および **/etc/crio/crio.conf** ファイルから設定情報を使用する。
- デフォルトでデータを **/var/lib/containers** に格納する。このディレクトリーは CRI-O および (**podman** などの) コンテナを実行するためのツールの両方で使用されます。



### 注記

同じストレージディレクトリーが使用されるものの、コンテナエンジンおよびコンテナツールはコンテナを別個に管理します。

- Docker バージョン 1 およびバージョン 2 スキーマの両方を格納できる。

**container-storage-setup** を使用して CRI-O のストレージを設定する方法については、「[Using container-storage-setup](#)」を参照してください。

### 1.3.2. CRI-O ネットワークの設定

CRI-O は [Container Network Interface](#) (CNI) と互換性のあるネットワーク機能をサポートします。サポートされるネットワーク機能には、ネットワークプラグインとして実装される loopback、flannel、および openshift-sdn が含まれます。

デフォルトで、OpenShift Container Platform は openshift-sdn ネットワークを使用します。**crio.conf** ファイルの以下の設定は、CNI ネットワーク設定ファイルが保管される場所 (**/etc/cni/net.d/**) および CNI プラグインバイナリーが保管される場所 (**/opt/cni/bin/**) を定義します。

```
[crio.network]
network_dir = "/etc/cni/net.d/"
plugin_dir = "/opt/cni/bin/"
```

OpenShift Container Platform の CRI-O で必要とされるネットワーク機能を理解するには、[Kubernetes](#) および [OpenShift Container Platform](#) ネットワーク要件の両方を参照してください。

## 1.4. CRI-O のトラブルシューティング

CRI-O コンテナエンジンの健全性を確認し、問題のトラブルシューティングを行うために、**crictl** コマンドを、一部のよく知られている Linux および OpenShift Container Platform コマンドと共に使用することができます。すべての OpenShift Container Platform コンテナエンジンの場合と同様に、**oc** および **kubectl** などのコマンドを使用して Pod を CRI-O で調査することもできます。

たとえば、Pod を一覧表示するには、以下を実行します。

```
$ sudo oc get pods -o wide
NAME                READY STATUS RESTARTS AGE IP          NODE   NOMINATED NODE
docker-registry-1-fb2g8 1/1   Running 1      5d 10.128.0.4  hostA <none>
registry-console-1-vk1l6 1/1   Running 0      5d 10.128.0.6  hostA <none>
router-1-hjfm7       1/1   Running 0      5d 192.168.122.188 hostA <none>
```

Pod が CRI-O で実行されていることを確認するために、**cri-o** の **describe** オプションおよび **grep** を使用します。

```
$ sudo oc describe pods registry-console-1-vk1l6 | grep cri-o
Container ID: cri-o://9a9209dc0608ce80f62bb4d7f7df61bcf8dd2abd77ef53075dee0542548238b7
```

CRI-O コンテナランタイムをクエリーし、デバッグするには、**crictl** コマンドを実行して CRI-O と直接通信します。**crictl** が使用する CRI-O インスタンスは **crictl.yaml** ファイルで特定されます。

```
# cat /etc/crictl.yaml
runtime-endpoint: /var/run/crio/crio.sock
```

デフォルトで、**crictl.yaml** ファイルは **crictl** にローカルシステムの CRI-O ソケットをポイントさせます。**crictl** で利用可能なオプションを確認するには、引数を指定せずに **crictl** を実行します。特定のオプションについてのヘルプを参照するには、**--help** を追加します。以下は例になります。

```
$ sudo crictl ps --help
NAME:
  crictl ps - List containers

USAGE:
  crictl ps [command options] [arguments...]

OPTIONS:
  --all, -a          Show all containers
  --id value         Filter by container id
  --label value     Filter by key=value label
  ...
```

#### 1.4.1. CRI-O の一般的なヘルスチェック

CRI-O を実行している OpenShift Container Platform クラスターにログインし、以下のコマンドを実行して CRI-O コンテナエンジンの一般的なヘルスチェックを行います。

CRI-O 関連のパッケージがインストールされていることを確認します。これには、**crio** (CRI-O daemon および設定ファイル) および **cri-tools** (**crictl** コマンド) パッケージが含まれます。

```
# rpm -qa | grep ^cri-
cri-o-1.11.6-1.rhaos3.11.git2d0f8c7.el7.x86_64
cri-tools-1.11.1-1.rhaos3.11.gitedabfb5.el7_5.x86_64
```

**crio** サービスが実行中であることを確認します。

```
# systemctl status -l crio
● crio.service - Open Container Initiative Daemon
  Loaded: loaded (/usr/lib/systemd/system/crio.service; enabled; vendor preset: disabled)
  Active: active (running) since Tue 2018-10-16 15:15:49 UTC; 3h 30min ago
  Docs: https://github.com/kubernetes-sigs/cri-o
  Main PID: 889 (crio)
  Tasks: 14
  Memory: 2.3G
  CGroup: /system.slice/crio.service
          └─889 /usr/bin/crio
```

```
Oct 16 15:15:48 hostA systemd[1]: Starting Open Container Initiative Daemon...
Oct 16 15:15:49 hostA systemd[1]: Started Open Container Initiative Daemon.
Oct 16 18:30:55 hostA cri-o[889]: time="2018-10-16 18:30:55.128074704Z" level=error
```

## 1.4.2. CRI-O ログの検査

CRI-O コンテナエンジンは systemd サービスとして実装されるため、標準の **journalctl** コマンドを使用して CRI-O のログメッセージを検査できます。

### 1.4.2.1. cri-o および origin-node ログの確認

cri-o サービスからの情報のジャーナルを確認するには、**-u** オプションを使用します。この例ではサービスが実行されていることを確認できますが、Pod は起動に失敗しています。

```
$ sudo journalctl -u cri-o
-- Logs begin at Tue 2018-10-16 15:01:31 UTC, end at Tue 2018-10-16 19:10:52 UTC. --
Oct 16 15:05:42 hostA systemd[1]: Starting Open Container Initiative Daemon...
Oct 16 15:05:42 hostA systemd[1]: Started Open Container Initiative Daemon.
Oct 16 15:06:35 hostA systemd[1]: Stopping Open Container Initiative Daemon...
Oct 16 15:06:35 hostA cri-o[4863]: time="2018-10-16 15:06:35.018523314Z" level=error msg="Failed to start streaming server: http: Server closed"
Oct 16 15:06:35 hostA systemd[1]: Starting Open Container Initiative Daemon...
Oct 16 15:06:35 hostA systemd[1]: Started Open Container Initiative Daemon.
Oct 16 15:10:27 hostA cri-o[6874]: time="2018-10-16 15:10:26.900411457Z" level=error msg="Failed to start streaming server: http: Server closed"
Oct 16 15:10:26 hostA systemd[1]: Stopping Open Container Initiative Daemon...
Oct 16 15:10:27 hostA systemd[1]: Stopped Open Container Initiative Daemon.
-- Reboot --
Oct 16 15:15:48 hostA systemd[1]: Starting Open Container Initiative Daemon...
Oct 16 15:15:49 hostA systemd[1]: Started Open Container Initiative Daemon.
Oct 16 18:30:55 hostA cri-o[889]: time="2018-10-16 18:30:55.128074704Z" level=error msg="Error adding network: CNI request failed with status 400: 'pods "
```

origin-node サービスで CRI-O 関連のメッセージを確認することもできます。以下は例になります。

```
$ sudo journalctl -u origin-node | grep -i cri-o

Oct 16 15:26:30 hostA origin-node[10624]: I1016 15:26:30.120889 10624
  kubernetes_manager.go:186] Container runtime cri-o initialized,
  version: 1.11.6, apiVersion: v1alpha1
Oct 16 15:26:30 hostA origin-node[10624]: I1016 15:26:30.177213 10624
  factory.go:157] Registering CRI-O factory
Oct 16 15:27:27 hostA origin-node[11107]: I1016 15:27:27.449197 11107
  kubernetes_manager.go:186] Container runtime cri-o initialized,
  version: 1.11.6, apiVersion: v1alpha1
Oct 16 15:27:27 hostA origin-node[11107]: I1016 15:27:27.507030 11107
  factory.go:157] Registering CRI-O factory
Oct 16 19:27:56 hostA origin-node[8326]: I1016 19:27:56.224770 8326
  kubernetes_manager.go:186] Container runtime cri-o initialized,
  version: 1.11.6, apiVersion: v1alpha1
Oct 16 19:27:56 hostA origin-node[8326]: I1016 19:27:56.282138 8326
  factory.go:157] Registering CRI-O factory
Oct 16 19:27:57 hostA origin-node[8326]: I1016 19:27:57.783304 8326
  status_manager.go:375] Status Manager: adding pod:
```

```
"db1f45e3-d157-11e8-8645-42010a8e0002", with status: ("x01", {Running ...
docker.io/openshift/origin-node:v3.11 docker.io/openshift/origin-node@sha256:6f9b0fbdd...
cri-o://c94cc6
2c27d021d61e8b7c1a82703d51db5847e74f5e57c667432f90c07013e4}} Burstable)) to
podStatusChannel
```

一覧表示されている Pod のいずれかで実行されていることをさらに調査する必要がある場合には (例: cri-o//c94cc6 として表示されている最後のもの)、**crictl logs** コマンドを使用することができます。

```
$ sudo crictl logs c94cc6
/etc/openvswitch/conf.db does not exist ... (warning).
Creating empty database /etc/openvswitch/conf.db [ OK ]
Starting ovssdb-server [ OK ]
Configuring Open vSwitch system IDs [ OK ]
Inserting openvswitch module [ OK ]
Starting ovs-vswitchd [ OK ]
Enabling remote OVSSDB managers [ OK ]
```

#### 1.4.2.2. CRI-O のデバッグをオンにする

CRI-O のロギング機能の詳細情報を取得するには、以下のようにログレベルを一時的に debug に設定することができます。

1. 以下に表示されているように **/usr/lib/systemd/system/crio.service** ファイルを編集し、**--loglevel=debug** を **ExecStart=** 行に追加します。

```
ExecStart=/usr/bin/crio --log-level=debug \
    $CRIO_STORAGE_OPTIONS \
    $CRIO_NETWORK_OPTIONS
```

2. 以下のように設定ファイルを再度読み込み、サービスを再起動します。

```
# systemctl daemon-reload
# systemctl restart crio
```

3. **journalctl** コマンドを再度実行します。処理が CRI-O サービスで実行されていることを表す数多くのデバッグメッセージが表示され始めます。

```
# journalctl -u crio
Oct 18 08:41:31 mynode01-crio crio[21998]:
    time="2018-10-18 08:41:31.839702058-04:00" level=debug
    msg="ListContainersRequest
    &ListContainersRequest{Filter:&ContainerFilter{Id:.,State:nil,PodSandboxId:
    ,LabelSelector:map[string]string{,},}"
Oct 18 08:41:31 mynode01-crio crio[21998]: time="2018-10-18
    08:41:31.839928476-04:00" level=debug msg="no filters were applied,
    returning full container list"
Oct 18 08:41:31 mynode01-crio crio[21998]: time="2018-10-18 08:41:31.841814536-04:00"
    level=debug msg="ListContainersResponse: &ListContainersResponse{Containers:
    [&Container{Id:e1934cc46696ff821bc35154f281764e80ac1122563ffd95aa92d01477225603,
    PodSandboxId:d904d45e6e46110a044758f20047805d8832b6859e10dc903c104cf757894e8d,
```

```
Metadata:&ContainerMetadata{Name:c,Attempt:0,},Image:&ImageSpec{
Image:e72de76ca8d5410497ae3171b6b059e7c7d11e4d1f3225df8d05812f29e205b7,},
ImageRef:docker.io/openshift/origin-template-service-broker@sha256:fd539 ...
```

4. 調査が終了したら **--loglevel=debug** オプションを削除し、生成されるメッセージの量を減らします。次に2つの **systemctl** コマンドを返します。

```
# systemctl daemon-reload
# systemctl restart cri-o
```

### 1.4.3. CRI-O Pod およびコンテナのトラブルシューティング

**crictl** コマンドを使用すると、CRI-O コンテナエンジンと直接インターフェースでき、コンテナエンジンに関連付けられたコンテナ、イメージ、および Pod をチェックし、操作できます。**runc** コンテナランタイムの使用は、CRI-O と対話するもう1つの方法です。サポートツールをノードで実行するなど、コンテナを CRI-O コンテナエンジンの外部で実行する必要がある場合は、**podman** コマンドを使用できます。

これらの2つのコマンドおよびそれらの違いについての説明は、「[Crictl vs. Podman](#)」を参照してください。

まず始めに、**crictl info** および **crictl version** コマンドを使用して CRI-O サービスの一般的な状態を確認することができます。

```
$ sudo crictl info
{
  "status": {
    "conditions": [
      {
        "type": "RuntimeReady",
        "status": true,
        "reason": "",
        "message": ""
      },
      {
        "type": "NetworkReady",
        "status": true,
        "reason": "",
        "message": ""
      }
    ]
  }
}

$ sudo crictl version
Version: 0.1.0
RuntimeName: cri-o
RuntimeVersion: 1.11.6
RuntimeApiVersion: v1alpha1
```

#### 1.4.3.1. イメージ、Pod、およびコンテナの一覧表示

**crictl** コマンドは CRI-O 環境でコンポーネントを調査するためのオプションを提供します。以下に、イメージ、Pod、およびコンテナについての情報を一覧表示するための **crictl** の使用方法についてのいくつかの例を示します。



ローカル CRI-O ノードにプルされているイメージを表示するには、**crictl images** コマンドを実行します。

```
$ sudo crictl images
IMAGE                                TAG  IMAGE ID  SIZE
docker.io/openshift/oauth-proxy     v1.1.0 90c45954eb03e 242MB
docker.io/openshift/origin-haproxy-router v3.11 13f40ad4d2e21 410MB
docker.io/openshift/origin-node     v3.11 93d2aeddcd6db 1.17GB
docker.io/openshift/origin-pod      v3.11 89ceff8fb1907 263MB
docker.io/openshift/prometheus-alertmanager v0.15.2 68bbd00063784 242MB
docker.io/openshift/prometheus-node-exporter v0.16.0 f9f775bf6d0ef 225MB
quay.io/coreos/cluster-monitoring-operator v0.1.1 4488a207a5bca 531MB
quay.io/coreos/configmap-reload     v0.0.1 3129a2ca29d75 4.79MB
quay.io/coreos/kube-rbac-proxy      v0.3.1 992ac1a5e7c79 40.4MB
quay.io/coreos/kube-state-metrics   v1.3.1 a9c8f313b7aad 22.2MB
```

CRI-O 環境で現在アクティブな Pod を表示するには、**crictl pods** を実行します。

```
$ sudo crictl pods

POD ID      CREATED      STATE NAME          NAMESPACE          ATTEMPT
09997515d7729 5 hours ago Ready kube-state-metrics-... openshift-monitoring 0
958b0789e0552 5 hours ago Ready node-exporter-rkbzp openshift-monitoring 0
4ec0498dacec8 5 hours ago Ready alertmanager-main-0 openshift-monitoring 0
2873b697df1d2 5 hours ago Ready cluster-monitoring-... openshift-monitoring 0
b9e221481fb7e 5 hours ago Ready router-1-968t4     default             0
f02ce4a4b4186 5 hours ago Ready sdn-c45cm           openshift-sdn       0
bdf5b1dcc0a08 5 hours ago Ready ovs-kdvzs           openshift-sdn       0
49dbc57455c8f 5 hours ago Ready sync-hgfvb        openshift-node      0
```

現在実行されているコンテナを表示するには、**crictl ps** コマンドを実行します。

```
$ sudo crictl ps
CONTAINER ID IMAGE                                CREATED      STATE NAME          ATTEMPT
376eb13e3cb37 quay.io/coreos/kube-state-metrics... 4 hours ago Running kube-state-metrics 0
72d61c3d393b5 992ac1a5e7c79d627321dc7877f741a00... 4 hours ago Running kube-rbac-proxy-
self 0
5fa8c93484055 992ac1a5e7c79d627321dc7877f741a00... 4 hours ago Running kube-rbac-proxy-
main 0
a2d35508fc0ee quay.io/coreos/kube-rbac-proxy... 4 hours ago Running kube-rbac-proxy 0
9adda43f3595f docker.io/openshift/prometheus-no... 4 hours ago Running node-exporter 0
7f4ce5b25cfdb docker.io/openshift/oauth-proxy... 4 hours ago Running alertmanager-proxy 0
85418badbf6ae quay.io/coreos/configmap-reload... 4 hours ago Running config-reloader 0
756f20138381c docker.io/openshift/prometheus-al... 4 hours ago Running alertmanager 0
5e6d8ff4852ba quay.io/coreos/cluster-monitoring... 4 hours ago Running cluster-monitoring- 0
1c96cfca10a7 docker.io/openshift/origin-haprox... 5 hours ago Running route 0
8f90bb4cded60 docker.io/openshift/origin-node... 5 hours ago Running sdn 0
59e5fb8514262 docker.io/openshift/origin-node... 5 hours ago Running openvswitch 0
73323a2c26abe docker.io/openshift/origin-node... 5 hours ago Running sync 0
```

実行中のコンテナおよび停止しているか、または終了しているコンテナの両方を表示するには、**crictl ps -a** を実行します。

```
$ sudo crictl ps -a
```

CRI-O サービスが停止しているか、または正常に機能していない場合、**runc** コマンドを使用して CRI-O で実行されたコンテナを一覧表示できます。以下の例では、CRI-O が実行されている/実行されていないコンテナの有無を確認します。次に、CRI-O が停止している場合でも、コンテナを **runc** で調査できることを示しています。

```
$ crictl ps | grep d36a99a9a40ec
d36a99a9a40ec    062cd20609d3895658e54e5f367b9d70f42db4f86ca14bae7309512c7e0777fd
  11 hours ago    CONTAINER_RUNNING  sync          2
$ sudo systemctl stop crio
$ sudo crictl ps | grep d36a99a9a40ec
2018/10/25 11:22:16 grpc: addrConn.resetTransport failed to create client transport:
  connection error: desc = "transport: dial unix /var/run/crio/crio.sock: connect:
  no such file or directory"; Reconnecting to {/var/run/crio/crio.sock <nil>}
  FATA[0000] listing containers failed: rpc error: code = Unavailable desc = grpc:
  the connection is unavailable
$ sudo runc list | grep d36a99a9a40ec
d36a99a9a40ecc4c830f10ed2d5bb3ce1c6deadc1a4879ff342e315051a71ed 19477    running
/run/containers/storage/overlay-
containers/d36a99a9a40ecc4c830f10ed2d5bb3ce1c6deadc1a4879ff342e315051a71ed/userdata
2018-10-25T04:44:29.47950187Z  root
$ ls /run/containers/storage/overlay-containers/d36*/userdata/
attach config.json ctl pidfile run
$ less /run/containers/storage/overlay-containers/d36*/userdata/config.json
{
  "ociVersion": "1.0.0",
  "process": {
    "user": {
      "uid": 0,
      "gid": 0
    },
    "args": [
      "/bin/bash",
      "-c",
      "#!/bin/bash\nset -euo pipefail\n\n# set by the node
        image\nunset KUBECONFIG\n\ntrap 'kill $(jobs -p);
        exit 0' TERM\n\n# track the current state of the ...
$ sudo systemctl start crio
```

上記から分かるように、CRI-O サービスをオフにした場合でも、**runc** は、詳細を確認したい場合にコンテナの存在とファイルシステムにおける位置を示します。

#### 1.4.3.2. イメージ、Pod、およびコンテナの調査

CRI-O 環境のイメージ、Pod またはコンテナの内部で実行されていることについての詳細を確認するには、いくつかの **crictl** オプションを使用することができます。

コンテナ ID (**crictl ps** の出力) を参照できる状態で、コンテナ内でコマンドを実行できます。たとえば、コンテナ内のオペレーティングシステムの名前およびリリースを確認するには、以下を実行します。

```
$ crictl exec 756f20138381c cat /etc/redhat-release
CentOS Linux release 7.5.1804 (Core)
```

コンテナ内で実行されているプロセスの一覧を表示するには、以下を実行します。

```
$ crictl exec -t e47b3a837aa30 ps -ef
UID      PID PPID C STIME TTY      TIME CMD
1000130+  1  0  0 Oct17 ?       00:38:14 /usr/bin/origin-web-console --au
1000130+ 15894 0  0 15:38 pts/0    00:00:00 ps -ef
1000130+ 17518  1  0 Oct23 ?       00:00:00 [curl] <defunct>
```

または、**runc** コマンドを使用してコンテナに対して「exec」を実行できます。

```
$ sudo runc exec -t e47b3a837aa3023c748c4c31a090266f014afba641a8ab9cfca31b065b4f2ddd ps
-ef
UID      PID PPID C STIME TTY      TIME CMD
1000130+  1  0  0 Oct17 ?       00:38:16 /usr/bin/origin-web-console --audit-log-path=- -v=0 --
config=/var/webconsole-config/webc
1000130+ 16541  0  0 15:48 pts/0    00:00:00 ps -ef
1000130+ 17518  1  0 Oct23 ?       00:00:00 [curl] <defunct>
```

コンテナ内に **ps** コマンドがない場合、**runc** では、コンテナで実行中のプロセスを表示する同じ機能を持つ **ps** オプションを使用できます。

```
$ sudo runc ps e47b3a837aa3023c748c4c31a090266f014afba641a8ab9cfca31b065b4f2ddd
```

**runc** には完全なコンテナ ID が必要ですが、**crictl** には最初のいくつかの固有の文字のみが必要であることに注意してください。

Pod サンドボックス ID (**crictl pods** からの出力) が参照できる状態で、**crictl inspectp** を実行し、Pod サンドボックスについての情報を表示します。

```
$ sudo crictl pods | grep 5a60ac777aaa0
5a60ac777aaa0 8 days ago SANDBOX_READY registry-console-1-vk1l6 default 0
$ sudo crictl inspectp 5a60ac777aaa0
{
  "status": {
    "id": "5a60ac777aaa055f14b998a9f2ced3e146b3cddb270154abb75decd583bf879",
    "metadata": {
      "attempt": 0,
      "name": "registry-console-1-vk1l6",
      "namespace": "default",
      "uid": "6af860cc-d20b-11e8-b094-525400535ba1"
    },
    "state": "SANDBOX_READY",
    "createdAt": "2018-10-17T08:53:22.828511516-04:00",
    "network": {
      "ip": "10.128.0.6"
    }
  }
}
```

ローカルシステムの CRI-O で利用可能なイメージについてのステータス情報を確認するには、**crictl inspecti** を実行します。

```
$ sudo crictl inspecti ff5dd2137a4ff
{
  "status": {
    "id": "ff5dd2137a4ffd5ccb9837d5a0aa0a5d10729f9c186df02e54e58748a32d08b0",
    "repoTags": [
      "quay.io/coreos/etcd:v3.2.22"
    ],
  }
}
```

```
"repoDigests": [  
  "quay.io/coreos/etcd@sha256:43fbc8a457aa0cb887da63d74a48659e13947cb74b96a53ba8f47abb6172a948"  
  ],  
  "size": "37547599",  
  "username": ""  
}  
}
```

## その他のリソース

- [CRI-O - OCI-based implementation of Kubernetes Container Runtime Interface](#)
- [CRI-O Lightweight Container Runtime for Kubernetes](#)
- [CRI-O Command Line Interface: crictl](#)
- [Finding, Running, and Building Containers without Docker](#)
- [Container Commandos Coloring Book](#)
- [CRI-O now running production workloads in OpenShift Online](#)
- [CRI-O How Standards Power a Container Runtime](#)
- [A Practical Introduction to Container Terminology](#)