# OpenShift Container Platform 3.11

## CRI-O ランタイム

CRI-O ランタイムガイド

Last Updated: 2023-03-25

# OpenShift Container Platform 3.11 CRI-O ランタイム

CRI-O ランタイムガイド

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

## 法律上の通知



## 概要

CRI-O の使用方法について

# 目次

# 第1章 CRI-O コンテナーエンジンの使用

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

CRI-O の使用者向けに、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と、CRI-O ノードを既存の OpenShift Container Platform クラスターに追加する方法について説明します。本書では、CRI-O エンジンの設定方法およびトラブルシューティングの方法についての情報も提供します。

## 1.1. CRI-O について

CRI-O コンテナーエンジンは、Open Container Initiative (OCI) 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、より安全で高性能のプラットフォームを提供します。CRI-O コンテナーエンジンを使用して、runc、デフォルトの OCI ランタイム、または Kata Containers などの OCI 準拠のランタイムを使用することにより、コンテナーおよび Pod を起動できます。CRI-O の目的は、Docker サービスに代わって、OpenShift Container Platform および Kubernetes 用の Kubernetes Container Runtime Interface(CRI) を実装するコンテナーエンジンになることです。

CRI-O は効率的なコンテナーエンジンを提供しますが、他のコンテナー機能は、革新的で独立したコマンドの個別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ーエンジンであるという CRI-O の主な目標を妨げることなく、コンテナー管理機能を独自のペースで開発することができます。

CRI-O の安定性は、Kubernetes のメジャーリリースとマイナーリリースと並行して開発、テスト、リリースされ、OCI 標準に準拠しているという事実に基づいています。たとえば、CRI-O 1.11 は Kubernetes 1.11 と連携します。CRI-O の範囲は Container Runtime Interface (CRI) に関連付けられています。CRI は、コンテナーエンジンから Kubernetes サービス (kubelet) が必要とするものを正確に抽出し、標準化しました。CRI チームは、複数のコンテナーエンジンが開発され始めたときに、Kubernetes コンテナーエンジンの要件を安定させるためにこれを行いました。

CRI-O とコマンドラインで直接アクセス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ーリングのために CRI-O へのフルアクセスを提供し、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、一連のコンテナー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ものを置き換え、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 コンテナーエンジンと直接連動させるために使用
- **runc**: コンテナーイメージを実行するために使用
- **podman**: コンテナーエンジンの外部で Pod およびコンテナーイメージ (run、stop、start、ps、attach、exec など) を管理するために使用
- **buildah**: コンテナーイメージを構築、プッシュ、および署名するために使用
- **skopeo**: イメージをコピー、検証、削除、および署名するために使用

一部の Docker 機能は、CRI-O ではなく他のツールに含まれます。たとえば、**podman** は、数多くの **docker** コマンド機能と正確なコマンドラインの互換性を提供し、これらの機能を Pod の管理にも拡張します。**podman** でコンテナーまたは Pod を実行する上で、コンテナーエンジンは必要ありません。

**buildah** コマンドでは、同じくコンテナーエンジンでは必要とされない、コンテナーイメージの構築、プッシュ、および署名の機能を使用できます。**docker** に代わるこれらのコマンドの詳細は、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ーエンジンとしては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ー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 を使用せずにコンテナーを実行するには、podman を使用します。

CRI-O コンテナーエンジンを 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 をそのノードのコンテナー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。

次のセクションでは、新しい OpenShift Container Platform クラスターと共に CRI-O をインストールする方法について説明します。

### 1.2.1. 新規の OpenShift Container Platform クラスターと共に CRI-O をインストールする

CRI-O は、インストール時にノードごとに OpenShift Container Platform ノードのコンテナーエンジンとして選択できます。OpenShift Container Platform のインストール時に CRI-O コンテナーエンジンを有効化することについて、知っておくべき点を以下にいくつか示します。

- 以前は、ノードで CRI-O を使用するには Docker コンテナー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 コンテナーエンジンは、すべての場合で必要なくなりました。現在は、CRI-O のみのノードを OpenShift Container Platform クラスターに含めることができます。ただし、ビルドおよびプッシュ操作を実行するノードには、依然として Docker コンテナーエンジンを CRI-O と共にインストールしておく必要があります。
- CRI-O コンテナーを使用した 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、Configuring Your Inventory File で説明されているような Ansible インベントリーファイルを使用して、OpenShift Container Platform をインストールしていることを前提としています。

**注記**

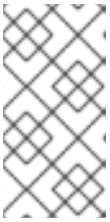
コンテナーエンジンとして CRI-O を使用する OpenShift Container Platform ノードの個別のマウントポイントとして **/var/lib/docker** を設定しないでください。CRI-O ノードのデプロイ時に、インストーラーは **/var/lib/docker** を **/var/lib/containers** へのシンボリックリンクにしようとします。このアクションは、既存の **/var/lib/docker** を削除してシンボリックリンクを作成することはできないために失敗します。

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. 選択し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 のインストール時に CRI-O コンテナーエンジンをノードにインストールできるようにするに

は、Ansible インベントリーファイルの [OSEv3:vars] セクションを見つけます。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 crio 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、ノードのコンテナーエンジンとしての CRI-O および Docker を許可し、overlay2 ストレージを持つノードで Docker ガベージコレクションを有効にします。



**注記**

CRI-O ノードでコンテナーをビルドできるようにするには、Docker コンテナーエンジンをインストールしておく必要があります。CRI-O のみのノードが必要な場合は、コンテナービルドを実行する他のノードを指定するだけで、可能となります。

```
[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。Defining Node Groups and Host Mappings 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 コンテナー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 コンテナーエンジンの使用から CRI-O の使用へのノードの直接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 コンテナー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します。
- CRI-O ノードが予想通りに実行されることを確認します。
- 必要に応じて CRI-O ノードをさらに追加します。
- クラスターが安定したら、Docker ノードをスケールダウンします。

CRI-O コンテナー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し、コンテナー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 コンテナー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。それ以降、CRI-O サービスは 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 によってデプロイ、アップグレード、および管理されることを目的としているため、CRI-O 設定ファイルは、OpenShift Container Platform を介してのみ、または CRI-O のテストまたはトラブルシューティングの目的でのみ変更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、リスニングソケット、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 (これらの設定を説明するコメントを確認するには、ファイル自体を調べてください)。

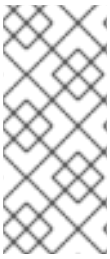
```
[crio]
root = "/var/lib/containers/storage"
runroot = "/var/run/containers/storage"
storage_driver = "overlay"
storage_option = [
      "overlay.override_kernel_check=1",
  ]

[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
conmon = "/usr/libexec/crio/conmon"
conmon_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、さまざまな 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、コンテナーエンジンとして CRI-O または Docker を使用しているかどうかに関係なく、OpenShift Container Platform で推奨される (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。

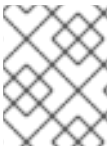


**注記**

devicemapper は CRI-O のサポートされるストレージ機能ですが、CRI-O ガベージコレクション機能はまだ devicemapper で機能しないため、実稼働環境での使用は推奨されません。また、CRI-O および **podman** の両方でコンテナーストレージを使用する方法に適用される他の 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 ストレージに関する以下のファクトが含まれます。

- 付随するレイヤーと共に各コンテナーの root ファイルシステムを格納することにより、イメージを保持します。
- Docker サービスで使用されるものと同じストレージレイヤーが組み込まれています。
- **container-storage-setup** を使用して、コンテナーストレージエリアを管理します。
- **/etc/containers/storage.conf** および **/etc/crio/crio.conf** ファイルの設定情報を使用します。
- デフォルトでデータを **/var/lib/containers** に保存します。そのディレクトリーは、CRI-O とコンテナーを実行するためのツール (**podman** など) 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を使用しますが、コンテナーエンジンとコンテナーツールは、コンテナーを個別に管理します。

- 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ます。

**container-storage-setup** を使用して CRI-O のストレージを設定する方法の詳細については、 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の保存場所 (**/etc/cni/net.d/**) および CNI プラグインバイナリーの保存場所 (**/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 で必要なネットワーク機能を理解するには、Kubernetes および OpenShift Container Platform の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 コンテナーエンジンをヘルスチェックし、問題のトラブルシューティングを行うために、**crictl** コマンドをいくつか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 コンテナー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、CRI-O の Pod を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-vktl6 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には、**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-vktl6 | grep cri-o
  Container ID:   cri-o://9a9209dc0608ce80f62bb4d7f7df61bcf8dd2abd77ef53075dee0542548238b7
```

CRI-O コンテナー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** ファイルで識別されます。

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

デフォルトで、**crictl.yaml** ファイルにより、crictl はローカルシステムの CRI-O ソケットをポイントします。**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。以下に例を示します。

```
$ 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 クラスターのノードにログインし、以下のコマンドを実行して CRI-O コンテナーエンジンの一般的なヘルスチェックをします。

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 デーモンと設定ファイル) 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  crio[889]: time="2018-10-16 18:30:55.128074704Z" level=error
```

### 1.4.2. CRI-O ログの検査

CRI-O コンテナーエンジンは 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. crio および origin-node ログの確認

crio 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 crio
-- 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 crio[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 crio[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 crio[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。以下に例を示します。

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

Oct 16 15:26:30 hostA origin-node[10624]: I1016 15:26:30.120889   10624
  kuberuntime_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
  kuberuntime_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
  kuberuntime_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 1 つ (cri-o // c94cc6 として示されている最後の Pod など) で何が起こっていたのかをさらに調査したかった場合は、**crictl logs** コマンドを使用できます。

```
$ sudo crictl logs c94cc6
/etc/openvswitch/conf.db does not exist ... (warning).
Creating empty database /etc/openvswitch/conf.db [  OK  ]
Starting ovsdb-server [  OK  ]
Configuring Open vSwitch system IDs [  OK  ]
Inserting openvswitch module [  OK  ]
Starting ovs-vswitchd [  OK  ]
Enabling remote OVSDB 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 に設定できます。

1. **/usr/lib/systemd/system/crio.service** ファイルを編集し、以下のように --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読み込みし、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。CRI-O サービスで進行中の処理を表す多くのデバッグメッセージが表示されるようになります。

```
# 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. 調査が完了したら **--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** コマンドを再度実行します。

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

### 1.4.3. CRI-O Pod およびコンテナー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 コンテナーエンジンと直接インターフェイスして、コンテナーエンジンに関連付けられたコンテナー、イメージ、および Pod を確認し、操作できます。**runc** コンテナーランタイムは、CRI-O と対話するもう 1 つの方法です。ノード上で support-tools を実行するなど、コンテナーを CRI-O コンテナー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 2 つのコマンドの説明とその違いについては、Crictl vs. Podman を参照してください。

開始するには、**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ステータスを確認できます。

```
$ 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、およびコンテナーの一覧表示

**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。以下は、イメージ、Pod、およびコンテナーについての情報を一覧表示するための **crictl** の使用例の一部になります。

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ーを表示するには、**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
1c96cfcfa10a7 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
```

実行中のコンテナーと、停止または終了したコンテナー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ーを一覧表示できます。この例では、CRI-O が実行されているコンテナーと実行されていないコンテナーの存在を検索します。次に、CRI-O が停止している場合でも、**runc** を使用してそのコンテナーを調査できることを示します。

```
$ 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
d36a99a9a40ecc4c830f10ed2d5bb3ce1c6deadcb1a4879ff342e315051a71ed   19477       running
  /run/containers/storage/overlay-
containers/d36a99a9a40ecc4c830f10ed2d5bb3ce1c6deadcb1a4879ff342e315051a71ed/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
```

ご覧のとおり、CRI-O サービスがオフの場合でも、さらに詳しく調べたい場合に備えて、**runc** はコンテナーの存在とファイルシステム内のその場所を示します。

#### 1.4.3.2. イメージ、Pod、およびコンテナーの調査

CRI-O 環境のイメージ、Pod またはコンテナー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** オプションを使用できます。

(**crictl ps** の出力からの) コンテナー ID が手元にあれば、そのコンテナー内でコマンドを実行できます。たとえば、コンテナー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)
```

コンテナー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>
```

別の方法として、**runc** コマンドを使用してコンテナーに"exec"することができます。

```
$ 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>
```

コンテナー内に **ps** コマンドがない場合、**runc** には **ps** オプションがあります。これは、コンテナーで実行されているプロセスを表示するのと同じ効果があります。

```
$ sudo runc ps e47b3a837aa3023c748c4c31a090266f014afba641a8ab9cfca31b065b4f2ddd
```

**runc** には完全なコンテナー ID が必要ですが、 **crictl** は最初のいくつかの固有の文字のみが必要である点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) が手元にある状態で、**crictl inspectp** を実行して Pod サンドボックスに関する情報を表示します。

```
$ sudo crictl pods | grep 5a60ac777aaa0
5a60ac777aaa0  8 days ago  SANDBOX_READY registry-console-1-vktl6  default  0
$ sudo crictl inspectp 5a60ac777aaa0
{
  "status": {
    "id": "5a60ac777aaa055f14b998a9f2ced3e146b3cddbe270154abb75decd583bf879",
    "metadata": {
      "attempt": 0,
      "name": "registry-console-1-vktl6",
      "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関する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": ""
  }
}
```

関連情報

- 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