Singularity CRI User Documentation

Release 1.0.0

Sylabs

Aug 06, 2019

CONTENTS

CHAPTER

ONE

INSTALLATION

This document will guide you through the process of installing Singularity-CRI on a Linux host.

1.1 Overview

Singularity-CRI is nothing more than Singularity-specific implementation of Kubernetes CRI.

It is currently under development and passes 71/74 validation tests. Note that used test suite is taken from *v1.13.0* tag. Detailed report can be found here.

1.2 Before you begin

If you have an earlier version of Singularity-CRI installed, you should *remove it* before executing the installation commands. You will also need to install some dependencies as described below.

1.3 Install Dependencies

1. Install git
2. Install Singularity 3.1+ with OCI support
3. Install Go 1.11+
4. Install inotify to enable GPU device plugin
5. Install *socat* if you want to enable fort-forwarding, e.g

```
$ sudo apt-get install socat
```

1.4 Install from source

The following commands will install Singularity-CRI from the GitHub repo to the `/usr/local/bin`.

The `master` branch contains the latest, bleeding edge version of Singularity-CRI. This is the default branch when you clone the source code, so you don't have to check out any new branches to install it. The `master` branch changes quickly and may be unstable. Thus installing from tag is the preferred way and is described below.

Since Singularity-CRI is now built with go modules there is no need to create standard go workspace. If you still prefer keeping source code under `GOPATH` make sure `GO111MODULE=on` is set.

The following assumes you want set up Singularity-CRI outside `GOPATH`.

```
$ git clone https://github.com/sylabs/singularity-cri.git && \
  cd singularity-cri && \
  git checkout tags/v1.0.0-beta.5 -b v1.0.0-beta.5 && \
  make && \
  sudo make install
```

After these commands Singularity-CRI will be installed in the `/usr/local/bin` directory.

Refer to *configuration section* to see how Singularity-CRI can be configured.

1.5 Remove an old version

When you run `install`, the command lists files as they are installed. They must be removed in order to completely remove Singularity-CRI from your host.

For convenience we created `uninstall` command, so you can run the following to cleanup installation:

```
$ sudo make uninstall
```

CHAPTER

TWO

KNOWN ISSUES AND CONSTRAINTS

2.1 Differentiate same image with different tags

Because images external to the Library are in a format other than SIF, when pulled they are converted to this native format for use by Singularity. Each time a SIF file is created through this conversion process a timestamp is automatically generated and captured as SIF metadata. Unfortunately, changes in the timestamp result in uniquely tagged images - even though the only difference is the timestamp in the SIF metadata. This matter has been classified as a known issue for documentation; refer to issue for additional details.

2.2 Using image from private registry

Unfortunately, it is not possible to use *on-prem* Sylabs library or pull private SIF images when working with Singularity-CRI. However, this is expected to change soon as we work with Kubernetes maintainers on the issue. As a current workaround we suggest to configure each node individually.

It is still possible to pull private SIF images from Cloud Library using image pull secrets. This will require creating a secret of type *kubernetes.io/dockerconfigjson* (a proposal is already created in order to change this hardcoded part of Kubernetes). The full flow is the following:

1. Create an access token to the Cloud Library (see docs)
2. Create pull secret

```
$ kubectl create secret docker-registry cloud-secret \
  --docker-server=cloud.sylabs.io \
  --docker-username=<any-name-here> \
  --docker-password=<cloud-token>
```

3. Use pull secret during pod creation

```
apiVersion: v1
kind: Pod
metadata:
  name: secret
spec:
  containers:
    - name: secret
      image: cloud.sylabs.io/sashayakovtseva/default/secret:interactive
      tty: true
      stdin: true
  imagePullSecrets:
    - name: cloud-secret
```

CHAPTER

THREE

CONFIGURATION

Singularity-CRI can be configured before the startup with a YAML config file. Example configuration can be found here.

Upon installation this default `sycri.yaml` config is copied to `/usr/local/etc/sycri/sycri.yaml` and that is the default location of the config file Singularity-CRI will look for. To override this behavior one can specify `--config` flag with path to the desired config file:

```
$ sycri --config ~/my-config.yaml
```

It is also possible to change logging level with `-v` flag. Singularity-CRI follows Kubernetes logging convention

Additionally you may specify log level 6 to enable Singularity runtime debug logging:

```
$ sycri -v 6
```

CHAPTER

FOUR

INTEGRATING WITH KUBERNETES

This document will guide you through the process of integrating Singularity-CRI with existing Kubernetes **v1.12+** cluster. If you don't have Kubernetes cluster already set up, please reference official installation guide.

Kubernetes cluster set up from scratch with Singularity-CRI can be found in wlm-operator repo's vagrant scripts.

If you are looking for trying Singularity-CRI locally, follow *local testing guide with Sykube*.

To fully enable Singularity support in Kubernetes cluster, Singularity-CRI should be installed on each Kubernetes node. However, one may choose to have a heterogeneous cluster with multiple container runtimes. In that case only dedicated Kubernetes nodes should be integrated, and no changes should be done to the rest.

To make Kubernetes work with Singularity-CRI a couple of steps are needed:

1. create Singularity-CRI service
2. modify kubelet config
3. restart kubelet with new config

4.1 Create Singularity-CRI service

Create a systemd service with the following content:

```
[Unit]
Description=Singularity-CRI
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=always
RestartSec=1
ExecStart=/usr/local/bin/sycri
Environment="PATH=/usr/local/libexec/singularity/bin:/bin:/sbin:/usr/local/sbin:/usr/
↪local/bin:/usr/sbin:/usr/bin"

[Install]
WantedBy=multi-user.target
```

Enable and start service afterwards:

```
$ sudo systemctl enable sycri && \
  sudo systemctl start sycri
```

Note: Singularity-CRI service above uses default configuration and log level. You can modify both of them if you wish, refer to *configuration section*.

Note: Latest Singularity plugin system is not stable and leads to panic when no `HOME` and `GOPATH` environments are set. There is an open issue related to this problem, so until it is open or if you have a bugged version, you may need to add the following line to Singularity-CRI service definition:

```
[Service]
...
Environment="GOPATH=/home/<your-name>/go"
```

To verify Singularity-CRI is running do the following:

```
$ sudo systemctl status sycri
```

You should see the following output:

```
* sycri.service - Singularity-CRI
   Loaded: loaded (/etc/systemd/system/sycri.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2019-02-22 15:59:02 UTC; 2min 54s ago
 Main PID: 31927 (sycri)
        Tasks: 9 (limit: 4915)
   CGroup: /system.slice/sycri.service
                   └─31927 /usr/local/bin/sycri

Jun 20 16:01:38 ubuntu-bionic systemd[1]: Started Singularity-CRI.
Jun 20 16:01:38 ubuntu-bionic sycri[1989]: I0620 16:01:38.980735    1989 network.
↪go:112] Network configuration found: bridge
Jun 20 16:01:38 ubuntu-bionic sycri[1989]: I0620 16:01:38.993158    1989 main.go:193]␣
↪Singularity-CRI server started on /var/run/singularity.sock
Jun 20 16:01:39 ubuntu-bionic sycri[1989]: I0620 16:01:39.095791    1989 device.
↪go:97] Loading NVML
Jun 20 16:01:39 ubuntu-bionic sycri[1989]: E0620 16:01:39.096779    1989 device.
↪go:99] Could not initialize NVML library: could not load NVML library
Jun 20 16:01:39 ubuntu-bionic sycri[1989]: W0620 16:01:39.097603    1989 main.go:209]␣
↪GPU support is not enabled: unable to load: check libnvidia-ml.so.1 library and␣
↪graphic drivers
```

Note: We recommend disabling other runtime services, e.g. docker daemon.

4.2 Modify kubelet config

Kubelet needs to be reconfigured so that it connects to Singularity-CRI. If you haven't changed default config, the following will be enough:

```
$ cat > /etc/default/kubelet <<EOF
  KUBELET_EXTRA_ARGS=--container-runtime=remote \
  --container-runtime-endpoint=unix:///var/run/singularity.sock \
  --image-service-endpoint=unix:///var/run/singularity.sock
  EOF
```

If you have changed `listenSocket` in Singularity-CRI configuration, make sure you pass that to kubelet instead of a default */var/run/singularity.sock*.

4.3 Restart kubelet service

```
$ sudo systemctl restart kubelet
```

That's it! After you completed those steps for a node, consider it configured to use Singularity as a container runtime. For examples refer to *examples section*.

4.4 GPU device plugin

Singularity-CRI is shipped with built-in NVIDIA GPU device plugin. It will automatically register itself in Kubernetes if node has any GPUs that can be discovered with NVML.

If GPU device-plugin was not enabled, you will see log line with the following content in Singularity-CRI logs, and Singularity-CRI will continue serving requests as usual:

```
GPU support is not enabled: <reason>
```

CHAPTER

FIVE

EXTENDING KUBERNETES SCHEDULER

Since SIF images do not support multiarch builds, its arch should be respected properly when it comes to pod scheduling. There are two options to make sure your SIF-based pod is scheduled on the node with a matching architecture: using *nodeSelector* in pod spec or configuring arch-scheduler extender.

This document will guide you through the second approach. To extend Kubernetes with arch extender the following steps are needed:

1. launch arch-scheduler pod
2. modify Kubernetes scheduler config
3. restart Kubernetes scheduler

5.1 Launch arch-scheduler pod

Clone arch-scheduler repo:

```
$ git clone https://github.com/sylabs/arch-scheduler
```

Launch arch-scheduler pod:

```
$ cd arch-scheduler && \
  kubectl apply -f deploy/extender.yaml
```

This will make arch-scheduler run on the same node where default Kubernetes scheduler is located. Location of arch-scheduler is important since later Kubernetes scheduler will query it on localhost.

5.2 Modify Kubernetes scheduler config

This step requires changing current scheduler policy to include arch-scheduler extender in extenders list.

If you have already modified scheduler policy you probably know how to do that. Further assumed default scheduler has not been altered and arch-scheduler extender is the first change applied.

Copy config and policy files under default Kubernetes directory:

```
$ sudo mkdir /etc/kubernetes/scheduler && \
  sudo cp deploy/config.yaml /etc/kubernetes/scheduler && \
  sudo cp deploy/policy.yaml /etc/kubernetes/scheduler
```

Modify kube-scheduler pod:

```
$ sudo cp deploy/scheduler.yaml /etc/kubernetes/manifests/kube-scheduler.yaml
```

Note: */etc/kubernetes/manifests/kube-scheduler.yaml* is the default location of scheduler pod specification. If you changed that, make sure to update to the appropriate one.

5.3 Restart kube-scheduler pod

By default, Kubernetes watches for any changes on static pods, and default scheduler is one of them. This means right after you updated kube-scheduler pod changes should automatically be applied in some reasonable amount of time. In the end you should see all system pods running without any issues:

```
$ kubectl get po --namespace=kube-system
```

However, if kube-scheduler is not in a *Running* state, try to simply delete it and let Kubernetes recreate it once again correctly:

```
$ kubectl delete po --namespace=kube-system kube-scheduler
```

Note: Kube-scheduler pod name may vary, make sure you are using correct one when deleting pod.

CHAPTER

SIX

COMPARING SINGULARITY-CRI WITH OTHERS

You may wonder, what makes Singularity-CRI different from other CRI implementations.

Well, there are couple of reasons. First of all, that is the only implementation fully compatible and designed specially for Singularity.

Singularity is known for its security and performance, especially when it comes to HPC. Unlike other popular runtimes, Singularity is **not** run as a daemon on a node, which prevents lots of security leaks.

Secondly, Singularity-CRI makes use of SIF images, which allows you to use all SIF benefits out of the box. For instance, Singularity-CRI will automatically check SIF signatures upon pulling an image. Also all pulled images that are not in SIF format will be automatically converted to SIF.

Thirdly, aiming HPC users needs, Singularity-CRI makes it possible to leverage pre-pulled SIF images to launch pods. To use this feature, specify *local.file* prefix before full SIF image path on host and Singularity-CRI will do the rest.

Last, but not least, Singularity is aimed at compute, that is why Singularity-CRI has built-in NVIDIA GPU support. With it, your Kubernetes cluster won't need any additional tuning to use GPUs. You use Kubernetes as usual and Singularity-CRI handles the rest.

CHAPTER

SEVEN

EXAMPLES

When Singularity-CRI is installed and configured and kubelet is restarted, you may use Kubernetes as you usually do. Here we will show some examples so you can verify your installation and probably discover some interesting use cases.

All examples can be found in Singularity-CRI repo. Further assumed you are located in Singularity-CRI repository root.

7.1 Hello, Kubernetes!

This is a dummy HTTP server suitable to verify Kubernetes installation.

```
$ kubectl apply -f examples/k8s/hello-kubernetes.yaml
```

After pods are up you should be able to hit hello-kubernetes Kubernetes service and see a web page with "Hello, World!" greeting.

7.2 Cat images server

Here we will walk through a basic Kubernetes example with SIF. We will deploy http file server that listens on port 8080 and creates a Kubernetes service to make it public on port 80.

Note: To make Singularity-CRI pull image from cloud library an explicit **cloud.sylabs.io** prefix should be specified in image field.

To create a deployment and a service run the following:

```
$ kubectl apply -f examples/k8s/image-service.yaml
```

To verify objects are indeed created you can do:

```
$ kubectl get deploy && \
  kubectl get svc
```

If everything is fine you should be able to access the cats server through the Kubernetes service. that has just been created. There are couple URLs that where created for testing purposes, `/cats/good` is the one with cats.

7.3 Bookshelf service

This is service for storing and searching books with MongoDB as a storage. Consists of two parts: MongoDB and a back-end application that provides API.

```
$ kubectl apply -f mongo.yaml && \
  kubectl apply -f bookshelf.yaml
```

After that you should be able to interact with bookshelf Kubernetes service.

The API is the following:

- List books

```
GET /books
```

- Create new book

```
POST /books
```

```
{
"title": "Les Misérables",
"author": "Victor Hugo",
"published_date": "1862",
"description": "Examining the nature of law and grace, the novel elaborates upon the␣
↪history of France, the architecture and urban design of Paris, politics, moral␣
↪philosophy, antimonarchism, justice, religion, and the types and nature of romantic␣
↪and familial love."
}
```

- Update existing book

```
PUT /books/<id>
```

```
{
"title": "Les Misérables",
"author": "Victor Hugo",
"published_date": "June 1862",
"description": "Examining the nature of law and grace, the novel elaborates upon the␣
↪history of France, the architecture and urban design of Paris, politics, moral␣
↪philosophy, antimonarchism, justice, religion, and the types and nature of romantic␣
↪and familial love."
}
```

- Get existing book

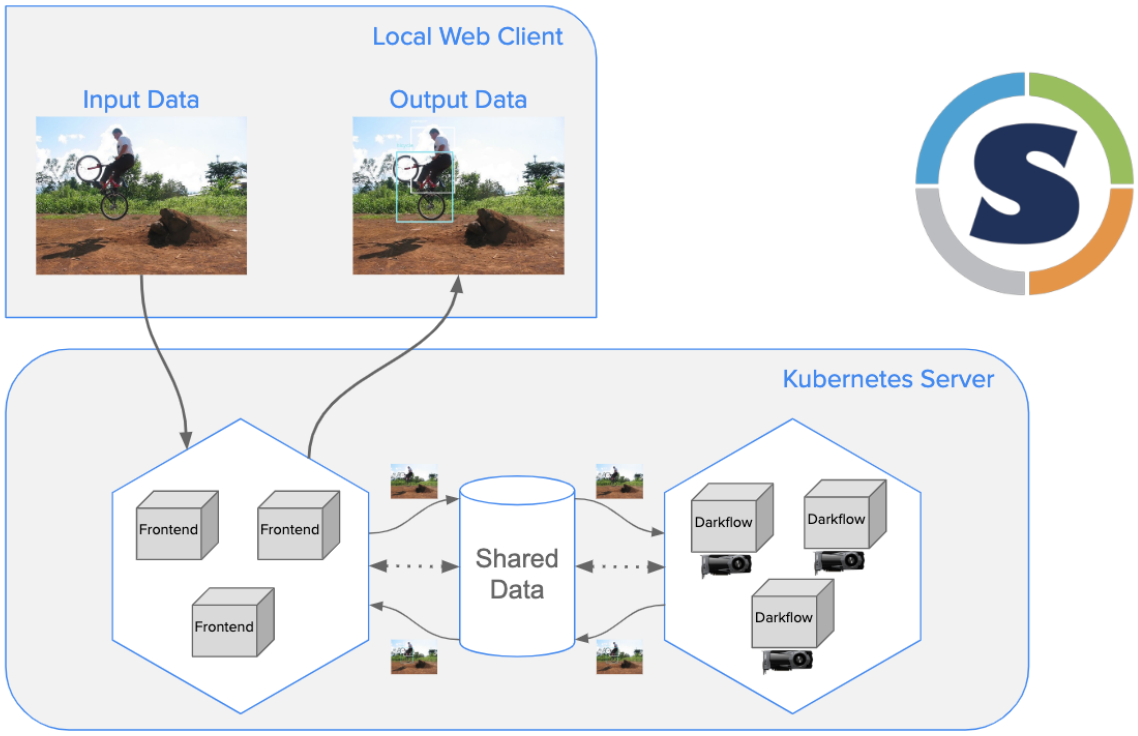
```
GET /books/<id>
```

- Delete existing book

```
POST /books/<id>:delete
```

7.4 Image recognition using NVIDIA GPU

Here we will deploy image recognition application that uses NVIDIA GPUs.



To create a deployment and a service run the following:

```
$ kubectl apply -f darkflow.yaml
```

To verify that objects are indeed created you can do:

```
$ kubectl get deploy && \
  kubectl get svc
```

If everything is fine you should be able to access Darkflow UI that is exposed with *darkflow-web* Kubernetes service.

Note: You may need to change *serverURL* value in a ConfigMao from the example above according to your cluster configuration. It should point to the darkflow-front Kubernetes service.

Also you can change *input* and *output* directories location on your host.

CHAPTER

EIGHT

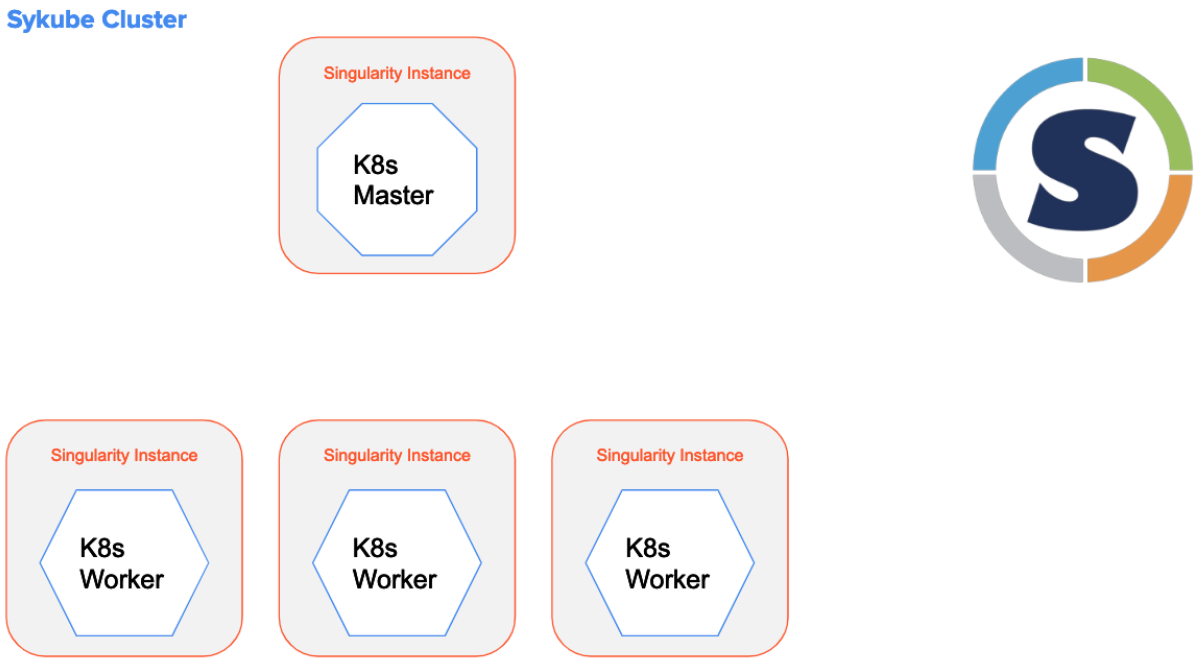
SYKUBE

Imagine that you want to test your Kubernetes cluster with Singularity-CRI locally, but don't want to waste time setting is all up or messing with Minikube.

Sykube comes to the rescue! Inspired by Minikube, it allows you to run Kubernetes cluster locally within **Singularity instances** with Singularity-CRI baked in. Moreover, unlike Minikube, it is capable of spawning not only one, but arbitrary amount of nodes.

Another nice feature is *ephemeral* clusters. With this option on, Sykube will create local cluster on *tmpfs* making sure nothing is left after host reboot.

For more info refer to Sykube repo.



8.1 Installation

Assuming you already have Singularity 3.1+ installed, do the following:

```
$ sudo singularity run library://sykube
```

This with pull the Sykube image and add a binary in */usr/local/bin*. To verify your installation you can check usage options with the following command:

```
$ sykube -h
```

8.2 Running local cluster

Before creating new Sykube cluster make sure you *removed any existing*. To create new Sykube cluster do the following:

```
$ sykube init
```

Warning: Make sure you don't have any restrictions applied to iptables *FORWARD* target. To check this do the following:

```
$ sudo iptables -nL
```

Problems are often caused by Docker daemon since it adds custom iptables rules. That prevents Sykube instance network from being correctly setup.

To disable docker and reboot you should do the following:

```
$ sudo service docker stop && \
  sudo systemctl disable docker && \
  reboot
```

After that Sykube should work correctly. Note this workaround may be redundant soon as there is an open GitHub issue referencing it here.

This may take a few minutes, stay patient.

After that if you already have kubectl installed, you may want to configure it to work with Sykube cluster. To do that run the following:

```
$ sykube config > ~/.kube/config
```

If you don't have kubectl, you can use Sykube, e.g:

```
$ sykube exec master kubectl <args>
```

8.3 Cleaning up

After testing you may want to remove the cluster. To do that run the following:

```
$ sykube stop && \
  sykube delete
```

CHAPTER

NINE

LOCAL TESTING

Further you may find a way to smoketest Singularity-CRI without Kubernetes installation.

9.1 Setting up environment

1. Install crictl:

```
$ VERSION="v1.12.0" && \
  wget https://github.com/kubernetes-sigs/cri-tools/releases/download/$VERSION/crictl-
↪$VERSION-linux-amd64.tar.gz && \
  sudo tar zxvf crictl-$VERSION-linux-amd64.tar.gz -C /usr/local/bin && \
  rm -f crictl-$VERSION-linux-amd64.tar.gz
```

2. Configure it work with Singularity-CRI. Create */etc/crictl.yaml* config file and add the following:

```
runtime-endpoint: unix:///var/run/singularity.sock
image-endpoint: unix:///var/run/singularity.sock
timeout: 10
debug: false
```

For details on all available options see crictl install page.

3. Build and launch Singularity-CRI server (optionally: *configure*):

```
$ make clean && \
  make && \
  sudo make install && \
  sudo sycri
```

9.2 Interact with Singularity-CRI

We will walk through basic examples of interaction with Singularity-CRI running. Further assumed you are located in Singularity-CRI repository root.

9.2.1 Running Nginx

1. Run a pod which exposes port 80:

```
$ sudo crictl runp examples/net-pod.json
```

That will return you ID of a freshly created pod. You will also see it when listing all pods on host:

```
$ sudo crictl pods
```

2. Create & start nginx container inside the just created pod:

```
$ sudo crictl pull nginx && \
  sudo crictl create <POD_ID> examples/nginx.json examples/net-pod.json
```

Here `POD_ID` is the ID of a pod you want your container to appear in. If everything is fine you will get ID of a created container. Also you will see container when listing all containers on host:

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

Unlike pods, containers should be explicitly started after creation:

```
$ sudo crictl start <CONTAINER_ID>
```

Verify Nginx container is running by opening localhost:80 in any browser. You should see the Nginx welcome page.

9.2.2 Running info container

1. Run any pod (we will use the same pod from the previous example):

```
$ sudo crictl runp examples/net-pod.json
```

2. Create & start container that outputs some system info:

```
$ sudo crictl pull cloud.sylabs.io/sashayakovtseva/test/test-info && \
  sudo crictl create <POD_ID> examples/info-cont.json examples/net-pod.json && \
  sudo crictl start <CONTAINER_ID>
```

Verify container executed correctly by opening logs:

```
$ sudo crictl logs <CONTAINER_ID>
```

The expected output is something like the following:

```
args: [./test]
mounts: 602 548 0:57 / / rw,relatime - overlay overlay rw,lowerdir=/var/run/
↪singularity/containers/
↪fa96e2cdaec1081a8b229fe2d8f64ac80b698b7a07f303629fb60b36abbeec8e/bundle/rootfs,
↪upperdir=/var/run/singularity/containers/
↪fa96e2cdaec1081a8b229fe2d8f64ac80b698b7a07f303629fb60b36abbeec8e/bundle/overlay/
↪upper,workdir=/var/run/singularity/containers/
↪fa96e2cdaec1081a8b229fe2d8f64ac80b698b7a07f303629fb60b36abbeec8e/bundle/overlay/work
603 602 0:50 / /proc rw,nosuid,nodev,noexec,relatime - proc proc rw
604 602 0:59 / /dev rw,nosuid - tmpfs tmpfs rw,size=65536k,mode=755
605 604 0:60 / /dev/pts rw,nosuid,noexec,relatime - devpts devpts rw,gid=5,mode=620,
↪ptmxmode=666
606 604 0:61 / /dev/shm rw,nosuid,nodev,noexec,relatime - tmpfs shm rw,size=65536k
607 604 0:49 / /dev/mqueue rw,nosuid,nodev,noexec,relatime - mqueue mqueue rw
608 602 0:56 / /sys ro,nosuid,nodev,noexec,relatime - sysfs sysfs ro
609 602 0:22 /singularity/pods/
↪85d02f45ee7fdf05aa199abafad6b1617fd018b3aacf30883c4724ebb025dac2/hostname /etc/
↪hostname ro,relatime shared:5 - tmpfs tmpfs rw,size=403956k,mode=755
610 602 8:1 /var/lib/singularity /mounted1 ro,relatime - ext4 /dev/sda1 rw,
↪errors=remount-ro,data=ordered
```

(continues on next page)

(continued from previous page)

```
hostname: networking <nil>
pwd: / <nil>
content of /
              Lrwxrwxrwx        0        .exec -> .singularity.d/actions/exec
              Lrwxrwxrwx        0        .run -> .singularity.d/actions/run
              Lrwxrwxrwx        0        .shell -> .singularity.d/actions/shell
              drwxr-xr-x        0        .singularity.d ->
              Lrwxrwxrwx        0        .test -> .singularity.d/actions/test
              drwxr-xr-x        0        bin ->
              drwxr-xr-x        0        dev ->
              Lrwxrwxrwx        0        environment -> .singularity.d/env/90-
↪environment.sh
              drwxr-xr-x        0        etc ->
              drwxr-xr-x        0        home ->
              drwxr-xr-x        0        lib ->
              drwxr-xr-x        0        media ->
              drwxr-xr-x        0        mnt ->
              drwxr-xr-x        0        mounted1 ->
              dr-xr-xr-x        0        proc ->
              drwx------        0        root ->
              drwxr-xr-x        0        run ->
              drwxr-xr-x        0        sbin ->
              Lrwxrwxrwx        0        singularity -> .singularity.d/runscript
              drwxr-xr-x        0        srv ->
              dr-xr-xr-x        0        sys ->
              -rwxr-xr-x        0        test ->
              dtrwxr-xr-x       0        tmp ->
              drwxr-xr-x        0        usr ->
              drwxr-xr-x        0        var ->
uid=0 gid=0 euid=0 egid=0
pid=30 ppid=0
envs=[LD_LIBRARY_PATH=/.singularity.d/libs SHLVL=1 MY_ANOTHER_VAR=is-awesome␣
↪PS1=Singularity>  TERM=xterm PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/
↪bin:/sbin:/bin PWD=/ MY_CUSTOM_VAR=singularity-cri]
...
```

9.2.3 Cleanup examples

The quickest way to cleanup is simply by removing containing pods:

```
$ sudo crictl stopp <POD_ID> && \
  sudo crictl rmp <POD_ID>
```