Singularity CRI User Documentation

Sylabs

Mar 07, 2019

CONTENTS

CHAPTER

ONE

INSTALLATION

This document will guide you through the process of installing Singularity CRI on existing Kubernetes **v1.12+** cluster. If you don't have Kubernetes cluster already set up, please reference official installation guide. Further assumed Linux environment since it is the only operating system that can support containers because of kernel features like namespaces

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

1.1 Overview

Singularity CRI is nothing more than Singularity-specific implementation of Kubernetes CRI. It is currently under development and passes 70/74 validation tests. Detailed report can be found here.

To fully enable Singularity support in Kubernetes cluster Singularity CRI should be installed on each Kubernetes node. Further you may find installation steps for a single node.

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 socat, e.g

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

1.4 Install from source

The following commands will install Singularity from the GitHub repo **master branch** to `/usr/local`.

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.

1.4.1 Download Singularity CRI repo

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` is set.

The following assumes you want to set up Singularity CRI outside `$GOPATH`. To set up project do the following:

```
$ git clone https://github.com/sylabs/singularity-cri.git && \
    cd singularity-cri && \
    git checkout tags/v1.0.0-alpha.2 -b v1.0.0-alpha.2 && \
    make dep
```

1.4.2 Build and install Singularity CRI

Assuming you are in repository root directory:

```
$ make && sudo make install
```

After this command Singularity CRI will be installed in the `/usr/local` directory hierarchy.

1.4.2.1 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 Singularity CRI will look for. To override this behavior one can specify `-config` flag with path to the desired config file, e.g:

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

It is also possible to change log level with `-v` flag, e.g:

```
$ sudo sycri -v 10
```

1.4.3 Configure node to use Singularity CRI

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

1.4.3.1 Create Singularity CRI service

To create the systemd service do the following:

```
$ cat > /etc/systemd/system/sycri.service <<EOF
        [Unit]
        Description=Singularity CRI
        After=network.target
        StartLimitIntervalSec=0

        [Service]
        Type=simple
        Restart=always
        RestartSec=1
        ExecStart=/usr/local/bin/sycri -v 10
        Environment="PATH=/usr/local/libexec/singularity/bin:/bin:/sbin:/usr/local/
↪sbin:/usr/local/bin:/usr/sbin:/usr/bin"

        [Install]
        WantedBy=multi-user.target
  EOF

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

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 -v 10

Feb 22 15:59:02 kube01 systemd[1]: Started Singularity CRI.
Feb 22 15:59:02 kube01 sycri[31927]: I0222 15:59:02.061441   31927 network.go:112]␣
↪Network configuration found: bridge
Feb 22 15:59:02 kube01 sycri[31927]: I0222 15:59:02.061598   31927 main.go:102]␣
↪Singularity CRI server started on /var/run/singularity.sock
```

Optionally you may want to disable other runtime services, e.g. docker daemon.

1.4.3.2 Modify kubelet config

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

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

If you have changed `listenSocket` make sure you pass it to kubelet as well.

1.4.3.3 Restart kubelet service

```
$ sudo systemctl restart kubelet
```

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

1.5 Remove an old version

When you run `sudo make install`, the command lists files as they are installed. They must all be removed in order to completely remove Singularity CRI. For convenience we created uninstall command, so you can run the following to cleanup installation:

```
$ sudo make uninstall
```

CHAPTER

TWO

BASIC USAGE

When Singularity CRI is installed and configured and kubelet is restarted, you may use k8s as you usually do. Here we will show some basic k8s usage so you can verify your installation. Full list of examples can be found in Singularity CRI repo.

2.1 Hello, cats!

Here we will walk through a basic k8s example with SIF. We will deploy http file server that listens on port 8080 and create a k8s service to make it public on port 80.

YAML configuration that we will be using is the following:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: image-service-deployment
  namespace: default
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: image-service
      name: image-service
      namespace: default
    spec:
      containers:
      - name: image-server
        image: cloud.sylabs.io/sashayakovtseva/test/image-server
        imagePullPolicy: IfNotPresent
        command: ["./image-server"]
        workingDir: "/go/bin"
        ports:
        - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: image-service
spec:
  type: NodePort
  ports:
    - port: 80
```

(continues on next page)

(continued from previous page)

```
    targetPort: 8080
  selector:
    app: image-service
```

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 image-service.yaml
deployment.extensions/image-service-deployment created
service/image-service created
```

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 file server via NodePort service.

CHAPTER

THREE

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.

CHAPTER

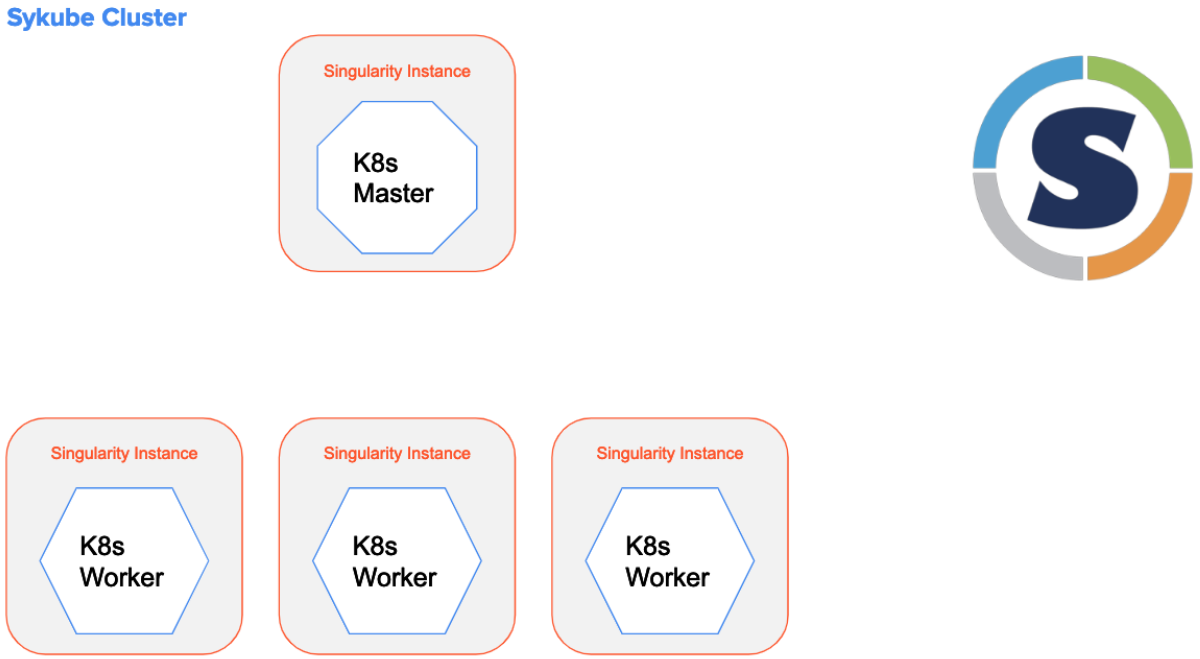
FOUR

SYKUBE

Imagine that you want to test your k8s 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 k8s 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.



4.1 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
```

4.2 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
Chain FORWARD (policy DROP)
target     prot opt source               destination
DOCKER-USER  all  --  0.0.0.0/0            0.0.0.0/0
DOCKER-ISOLATION-STAGE-1  all  --  0.0.0.0/0            0.0.0.0/0
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0           ctstate RELATED,
↪ESTABLISHED
DOCKER     all  --  0.0.0.0/0            0.0.0.0/0
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
```

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>
```

4.3 Cleaning up

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

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