# Singularity Container Documentation

***Release 3.0***

## User Docs

**Oct 09, 2018**

# CONTENTS

CHAPTER

ONE

# QUICK START

This guide is intended for running Singularity on a computer where you have root (administrative) privileges.

If you need to request an installation on your shared resource, see the requesting an installation help page for information to send to your system administrator.

For any additional help or support contact the Sylabs team: https://www.sylabs.io/contact/

## 1.1 Quick Installation Steps

You will need a Linux system to run Singularity.

See the installation page for information about installing older versions of Singularity.

### 1.1.1 Install system dependencies

You must first install development libraries to your host. Assuming Ubuntu (apply similar to RHEL derivatives):

```
$ sudo apt-get update && sudo apt-get install -y \
    build-essential \
    libssl-dev \
    uuid-dev \
    libgpgme11-dev
```

### 1.1.2 Install Go

Singularity 3.0 is written primarily in Go, and you will need Go installed to compile it from source.

This is one of several ways to install and configure Go.

First, visit the Go download page and pick the appropriate Go archive (>=1.11.1). Copy the link address and download with `wget` like so:

```
$ export VERSION=1.11 OS=linux ARCH=amd64
$ cd /tmp
$ wget https://dl.google.com/go/go$VERSION.$OS-$ARCH.tar.gz
```

Then extract the archive to `/usr/local`

```
$ sudo tar -C /usr/local -xzf go$VERSION.$OS-$ARCH.tar.gz
```

Finally, set up your environment for Go

```
$ echo 'export GOPATH=${HOME}/go' >> ~/.bashrc
$ echo 'export PATH=/usr/local/go/bin:${PATH}:${GOPATH}/bin' >> ~/.bashrc
$ source ~/.bashrc
```

### 1.1.3 Clone the Singularity repository

Go is a bit finicky about where things are placed. Here is the correct way to build Singularity from source.

```
$ mkdir -p $GOPATH/src/github.com/sylabs
$ cd $GOPATH/src/github.com/sylabs
$ git clone https://github.com/sylabs/singularity.git
$ cd singularity
```

### 1.1.4 Install Go dependencies

Dependencies are managed using Dep. You can use go get to install it like so:

```
$ go get -u -v github.com/golang/dep/cmd/dep
```

### 1.1.5 Compile the Singularity binary

Now you are ready to build Singularity. Dependencies will be automatically downloaded. You can build Singularity using the following commands:

```
$ cd $GOPATH/src/github.com/sylabs/singularity
$ ./mconfig
$ make -C builddir
$ sudo make -C builddir install
```

Singularity must be installed as root to function properly.

## 1.2 Overview of the Singularity Interface

Singularity's command line interface allows you to build and interact with containers transparently. You can run programs inside a container as if they were running on your host system. You can easily redirect IO, use pipes, pass arguments, and access files, sockets, and ports on the host system from within a container.

The `help` command gives an overview of Singularity options and subcommands as follows:

```
$ singularity help

Linux container platform optimized for High Performance Computing (HPC) and
Enterprise Performance Computing (EPC)

Usage:
  singularity [global options...]

Description:
  Singularity containers provide an application virtualization layer enabling
  mobility of compute via both application and environment portability. With
```

(continues on next page)

(continued from previous page)

```
  Singularity one is capable of building a root file system that runs on any
  other Linux system where Singularity is installed.

Options:
  -d, --debug              print debugging information (highest verbosity)
  -h, --help               help for singularity
  -q, --quiet              suppress normal output
  -s, --silent             only print errors
  -t, --tokenfile string   path to the file holding your sylabs
                           authentication token (default
                           "/home/david/.singularity/sylabs-token")
  -v, --verbose            print additional information

Available Commands:
  build       Build a new Singularity container
  capability  Manage Linux capabilities on containers
  exec        Execute a command within container
  help        Help about any command
  inspect     Display metadata for container if available
  instance    Manage containers running in the background
  keys        Manage OpenPGP key stores
  pull        Pull a container from a URI
  push        Push a container to a Library URI
  run         Launch a runscript within container
  run-help    Display help for container if available
  search      Search the library
  shell       Run a Bourne shell within container
  sign        Attach cryptographic signatures to container
  test        Run defined tests for this particular container
  verify      Verify cryptographic signatures on container
  version     Show application version

Examples:
  $ singularity help <command>
      Additional help for any Singularity subcommand can be seen by appending
      the subcommand name to the above command.


For additional help or support, please visit https://www.sylabs.io/docs/
```

Information about subcommand can also be viewed with the `help` command.

```
$ singularity help verify
Verify cryptographic signatures on container

Usage:
  singularity verify [verify options...] <image path>

Description:
  The verify command allows a user to verify cryptographic signatures on SIF
  container files. There may be multiple signatures for data objects and
  multiple data objects signed. By default the command searches for the primary
  partition signature. If found, a list of all verification blocks applied on
  the primary partition is gathered so that data integrity (hashing) and
  signature verification is done for all those blocks.

Options:
```

(continues on next page)

(continued from previous page)

```
  -g, --groupid uint32   group ID to be verified
  -h, --help             help for verify
  -i, --id uint32        descriptor ID to be verified
  -u, --url string       key server URL (default "https://keys.sylabs.io")


Examples:
  $ singularity verify container.sif


For additional help or support, please visit https://www.sylabs.io/docs/
```

Singularity uses positional syntax (i.e. the order of commands and options matters).

Global options affecting the behavior of all commands follow the main `singularity` command. Then sub commands are passed followed by their options and arguments.

For example, to pass the `--debug` option to the main `singularity` command and run Singularity with debugging messages on:

```
$ singularity --debug run library://sylabsed/examples/lolcow
```

To pass the `--containall` option to the `run` command and run a Singularity image in an isolated manner:

```
$ singularity run --containall library://sylabsed/examples/lolcow
```

Singularity 2.4 introduced the concept of command groups. For instance, to list Linux capabilities for a particular user, you would use the `list` command in the `capabilities` command group like so:

```
$ singularity capability list --user dave
```

Container authors might also write help docs specific to a container or for an internal module called an `app`. If those help docs exist for a particular container, you can view them like so.

```
$ singularity help container.sif  # See the container's help, if provided

$ singularity help --app foo container.sif  # See the help for foo, if provided
```

## 1.3 Download pre-built images

You can use the `search` command to locate groups, collections, and containers of interest on the Container Library .

```
$ singularity search alp
No users found for 'alp'

Found 1 collections for 'alp'
    library://jchavez/alpine

Found 5 containers for 'alp'
    library://jialipassion/official/alpine
            Tags: latest
    library://dtrudg/linux/alpine
            Tags: 3.2 3.3 3.4 3.5 3.6 3.7 3.8 edge latest
    library://sylabsed/linux/alpine
```

(continues on next page)

(continued from previous page)

```
            Tags: 3.6 3.7 latest
    library://library/default/alpine
            Tags: 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 latest
    library://sylabsed/examples/alpine
            Tags: latest
```

You can use the pull and build commands to download pre-built images from an external resource like the Container Library or Docker Hub.

When called on a native Singularity image like those provided on the Container Library, `pull` simply downloads the image file to your system.

```
$ singularity pull library://sylabsed/linux/alpine
```

You can also use `pull` with the `docker://` uri to reference Docker images served from a registry. In this case `pull` does not just download an image file. Docker images are stored in layers, so `pull` must also combine those layers into a usable Singularity file.

```
$ singularity pull docker://godlovedc/lolcow
```

Pulling Docker images reduces reproducibility. If you were to pull a Docker image today and then wait six months and pull again, you are not guaranteed to get the same image. If any of the source layers has changed the image will be altered. If reproducibility is a priority for you, try building your images from the Container Library.

You can also use the `build` command to download pre-built images from an external resource. When using `build` you must specify a name for your container like so:

```
$ singularity build ubuntu.sif library://ubuntu

$ singularity build lolcow.sif docker://godlovedc/lolcow
```

Unlike `pull`, `build` will convert your image to the latest Singularity image format after downloading it.

`build` is like a "Swiss Army knife" for container creation. In addition to downloading images, you can use `build` to create images from other images or from scratch using a definition file. You can also use `build` to convert an image between the container formats supported by Singularity.

## 1.4 Interact with images

You can interact with images in several ways. It is not actually necessary to `pull` or `build` an image to interact with it. The commands listed here will work with image URIs in addition to accepting a local path to an image.

For these examples we will use a `lolcow_latest.sif` image that can be pulled from the Container Library like so.

```
$ singularity pull library://sylabsed/examples/lolcow
```

### 1.4.1 Shell

The shell command allows you to spawn a new shell within your container and interact with it as though it were a small virtual machine.

```
$ singularity shell lolcow_latest.sif

Singularity lolcow_latest.sif:~>
```

The change in prompt indicates that you have entered the container (though you should not rely on that to determine whether you are in container or not).

Once inside of a Singularity container, you are the same user as you are on the host system.

```
Singularity lolcow_latest.sif:~> whoami
david

Singularity lolcow_latest.sif:~> id
uid=1000(david) gid=1000(david) groups=1000(david),4(adm),24(cdrom),27(sudo),30(dip),
↪46(plugdev),116(lpadmin),126(sambashare)
```

`shell` also works with the `library://`, `docker://`, and `shub://` URIs. This creates an ephemeral container that disappears when the shell is exited.

```
$ singularity shell library://sylabsed/examples/lolcow
```

### 1.4.2 Executing Commands

The exec command allows you to execute a custom command within a container by specifying the image file. For instance, to execute the `cowsay` program within the `lolcow_latest.sif` container:

```
$ singularity exec lolcow_latest.sif cowsay moo
 _____
< moo >
 -----
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

`exec` also works with the `library://`, `docker://`, and `shub://` URIs. This creates an ephemeral container that executes a command and disappears.

```
$ singularity exec library://sylabsed/examples/lolcow cowsay "Fresh from the library!"
 _________________________
< Fresh from the library! >
 -------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

### 1.4.3 Running a container

Singularity containers contain runscripts. These are user defined scripts that define the actions a container should perform when someone runs it. The runscript can be triggered with the run command, or simply by calling the container as though it were an executable.

```
$ singularity run lolcow_latest.sif
 _______________________________________
/ You have been selected for a secret \
\ mission.                            /
 ---------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||

$ ./lolcow_latest.sif
 ______________________________________
/ Q: What is orange and goes "click, \
\ click?" A: A ball point carrot.    /
 --------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

`run` also works with the `library://`, `docker://`, and `shub://` URIs. This creates an ephemeral container that runs and then disappears.

```
$ singularity run library://sylabsed/examples/lolcow
 _____________________________________
/ Is that really YOU that is reading \
\ this?                              /
 -------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

### 1.4.4 Working with Files

Files on the host are reachable from within the container.

```
$ echo "Hello from inside the container" > $HOME/hostfile.txt

$ singularity exec lolcow_latest.sif cat $HOME/hostfile.txt

Hello from inside the container
```

This example works because `hostfile.txt` exists in the user's home directory. By default Singularity bind mounts `/home/$USER`, `/tmp`, and `$PWD` into your container at runtime.

You can specify additional directories to bind mount into your container with the `--bind` option. In this example, the `data` directory on the host system is bind mounted to the `/mnt` directory inside the container.

```
$ echo "Drink milk (and never eat hamburgers)." > /data/cow_advice.txt

$ singularity exec --bind /data:/mnt lolcow_latest.sif cat /mnt/cow_advice.txt
Drink milk (and never eat hamburgers).
```

Pipes and redirects also work with Singularity commands just like they do with normal Linux commands.

```
$ cat /data/cow_advice.txt | singularity exec lolcow_latest.sif cowsay
 ________________________________________
< Drink milk (and never eat hamburgers). >
 ----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

## 1.5 Build images from scratch

Singularity v3.0 produces immutable images in the Singularity Image File (SIF) format. This ensures reproducible and verifiable images and allows for many extra benefits such as the ability to sign and verify your containers.

However, during testing and debugging you may want an image format that is writable. This way you can `shell` into the image and install software and dependencies until you are satisfied that your container will fulfill your needs. For these scenarios, Singularity also supports the `sandbox` format (which is really just a directory).

For more details about the different build options and best practices, read about the Singularity flow.

### 1.5.1 Sandbox Directories

To build into a `sandbox` (container in a directory) use the `build --sandbox` command and option:

```
$ sudo singularity build --sandbox ubuntu/ library://ubuntu
```

This command creates a directory called `ubuntu/` with an entire Ubuntu Operating System and some Singularity metadata in your current working directory.

You can use commands like `shell`, `exec` , and `run` with this directory just as you would with a Singularity image. If you pass the `--writable` option when you use your container you can also write files within the sandbox directory (provided you have the permissions to do so).

```
$ sudo singularity exec --writable ubuntu touch /foo

$ singularity exec ubuntu/ ls /foo
/foo
```

### 1.5.2 Converting images from one format to another

The `build` command allows you to build a container from an existing container. This means that you can use it to convert a container from one format to another. For instance, if you have already created a sandbox (directory) and want to convert it to the default immutable image format (squashfs) you can do so:

```
$ singularity build new-sif sandbox
```

Doing so may break reproducibility if you have altered your sandbox outside of the context of a definition file, so you are advised to exercise care.

### 1.5.3 Singularity Definition Files

For a reproducible, production-quality container you should build a SIF file using a Singularity definition file. This also makes it easy to add files, environment variables, and install custom software, and still start from your base of choice (e.g., the Container Library).

A definition file has a header and a body. The header determines the base container to begin with, and the body is further divided into sections that do things like install software, setup the environment, and copy files into the container from the host system.

Here is an example of a definition file:

```
BootStrap: library
From: ubuntu:16.04

%post
    apt-get -y update
    apt-get -y install fortune cowsay lolcat

%environment
    export LC_ALL=C
    export PATH=/usr/games:$PATH

%runscript
    fortune | cowsay | lolcat

%labels
    Author GodloveD
```

To build a container from this definition file (assuming it is a file named lolcow.def), you would call build like so:

```
$ sudo singularity build lolcow.sif lolcow.def
```

In this example, the header tells Singularity to use a base Ubuntu 16.04 image from the Container Library.

The `%post` section executes within the container at build time after the base OS has been installed. The `%post` section is therefore the place to perform installations of new applications.

The `%environment` section defines some environment variables that will be available to the container at runtime.

The `%runscript` section defines actions for the container to take when it is executed.

And finally, the `%labels` section allows for custom metadata to be added to the container.

This is a very small example of the things that you can do with a definition file. In addition to building a container from the Container Library, you can start with base images from Docker Hub and use images directly from official repositories such as Ubuntu, Debian, CentOS, Arch, and BusyBox. You can also use an existing container on your host system as a base.

If you want to build Singularity images but you don't have administrative (root) access on your build system, you can build images using the Remote Builder.

This quickstart document just scratches the surface of all of the things you can do with Singularity!

If you need additional help or support, contact the Sylabs team: https://www.sylabs.io/contact/

CHAPTER

TWO

# SIGNING AND VERIFYING CONTAINERS

Singularity 3.0 introduces the abilities to create and manage PGP keys and use them to sign and verify containers. This provides a trusted method for Singularity users to share containers. It ensures a bit-for-bit reproduction of the original container as the author intended it.

## 2.1 Verifying containers from the Container Library

The `verify` command will allow you to verify that a container has been signed using a PGP key. To use this feature with images that you pull from the container library, you must first generate an access token to the Sylabs Cloud. If you don't already have a valid access token, follow these steps:

1. Go to : https://cloud.sylabs.io/
2. Click "Sign in to Sylabs" and follow the sign in steps.
3. Click on your login id (same and updated button as the Sign in one).
4. Select "Access Tokens" from the drop down menu.
5. Click the "Manage my API tokens" button from the "Account Management" page.
6. Click "Create".
7. Click "Copy token to Clipboard" from the "New API Token" page.
8. Paste the token string into your `~/.singularity/sylabs-token` file.

Now you can verify containers that you pull from the library, ensuring they are bit-for-bit reproductions of the original image.

```
$ singularity pull library://alpine

$ singularity verify alpine_latest.sif
Verifying image: alpine_latest.sif
Data integrity checked, authentic and signed by:
    Sylabs Admin <support@sylabs.io>, KeyID 51BE5020C508C7E9
```

In this example you can see that **Sylabs Admin** has signed the container.

## 2.2 Signing your own containers

### 2.2.1 Generating and managing PGP keys

To sign your own containers you first need to generate one or more keys.

If you attempt to sign a container before you have generated any keys, Singularity will guide you through the interactive process of creating a new key. Or you can use the `newpair` subcommand in the `key` command group like so:.

```
$ singularity keys newpair
Enter your name (e.g., John Doe) : Dave Godlove
Enter your email address (e.g., john.doe@example.com) : d@sylabs.io
Enter optional comment (e.g., development keys) : demo
Generating Entity and OpenPGP Key Pair... Done
Enter encryption passphrase :
```

The `list` subcommand will show you all of the keys you have created or saved locally.‘

```
$ singularity keys list
Public key listing (/home/david/.singularity/sypgp/pgp-public):

0) U: Dave Godlove (demo) <d@sylabs.io>
   C: 2018-10-08 15:25:30 -0400 EDT
   F: 135E426D67D8416DE1D6AC7FFED5BBA38EE0DC4A
   L: 4096
   --------
```

In the output above, the letters stand for the following:

- U: User
- C: Creation date and time
- F: Fingerprint
- L: Key length

After generating your key you can optionally push it to the Keystore using the fingerprint like so:

```
$ singularity keys push 135E426D67D8416DE1D6AC7FFED5BBA38EE0DC4A
public key `135E426D67D8416DE1D6AC7FFED5BBA38EE0DC4A` pushed to server successfully
```

This will allow others to verify images that you have signed.

If you delete your local public PGP key, you can always locate and download it again like so.

```
$ singularity keys search Godlove
Search results for 'Godlove'

Type bits/keyID     Date       User ID
-------------------------------------------------------------------------------
pub  4096R/8EE0DC4A 2018-10-08 Dave Godlove (demo) <d@sylabs.io>
-------------------------------------------------------------------------------

$ singularity keys pull 8EE0DC4A
1 key(s) fetched and stored in local cache /home/david/.singularity/sypgp/pgp-public
```

But note that this only restores the *public* key (used for verifying) to your local machine and does not restore the *private* key (used for signing).

### 2.2.2 Signing and validating your own containers

Now that you have a key generated, you can use it to sign images like so:

```
$ singularity sign my_container.sif
Signing image: my_container.sif
Enter key passphrase:
Signature created and applied to my_container.sif
```

Because your public PGP key is saved locally you can verify the image without needing to contact the Keystore.

```
$ singularity verify my_container.sif
Verifying image: my_container.sif
Data integrity checked, authentic and signed by:
    Dave Godlove (demo) <d@sylabs.io>, KeyID FED5BBA38EE0DC4A
```

If you've pushed your key to the Keystore you can also verify this image in the absence of a local key. To demonstrate this, first delete your local keys, and then try to use the `verify` command again.

```
$ rm ~/.singularity/sypgp/*

$ singularity verify my_container.sif
Verifying image: my_container.sif
INFO:    key missing, searching key server for KeyID: FED5BBA38EE0DC4A...
INFO:    key retreived successfully!
Store new public key 135E426D67D8416DE1D6AC7FFED5BBA38EE0DC4A? [Y/n] y
Data integrity checked, authentic and signed by:
    Dave Godlove (demo) <d@sylabs.io>, KeyID FED5BBA38EE0DC4A
```

Answering yes at the interactive prompt will store the Public key locally so you will not have to contact the Keystore again the next time you verify your container.