
Learning Objectives

This is the tutorial on using Apptainer (https://apptainer.org). The learning objectives in the tutorial are

- Install Apptainer (https://apptainer.org)
- Make sure network namespaces are disabled
- Configure Apptainer so users don't starve everyone else of resources
- Build and run basic containers

Tools

- Apptainer (https://apptainer.org)
- network namespaces

Contents

Install Apptainer 1: Tutorial (1 min) 1

Setup network namespaces 2: Tutorial (2 min) 2

Limiting resource usage when users build images 3: Tutorial (3 min) 2

Create a container image manually and run it 4: Tutorial (3 min) 2

Use a definition file 5: Tutorial (2 min) 3

For the Apptainer (https://apptainer.org) in the Singularity/Apptainer ecosystem part of this course,

Install Apptainer 1: Tutorial (1 min)

Steps

1. `$ sudo dnf install apptainer`

Further Reading

- `dnf` – https://dnf.readthedocs.io

Note, if you had wanted to do a setUID install, you would have needed to install `apptainer-suid`.

Setup network namespaces 2: Tutorial (2 min)

Steps

1. Check the maximum number of network namespaces that are allowed by
   `$ sudo sysctl -a | grep net_namespaces`
2. If it is zero, network namespaces are disable since none can be created. Singularity/Apptainer users rarely need them and they pose a security risk when combined with user namespaces, so it is recommended to disable them. If you need to change it:
   a) Change it just for the current boot by
      `$ sudo sysctl user.max_net_namespaces=0`
   b) To set a different value on boot, it needs to be added to the `sysctl` config at `/etc/sysctl.conf` and the files in `/etc/sysctl.d/`. You need to first see if it is already set in one of those files and change its value. A fast way to check is with
      `$ grep -r net_namespaces /etc/sysctl.*`
      If it isn't in any of the files, add a file in `/etc/sysct.d` with the following line:

```
user.max_net_namespaces = 0
```

Further Reading

- `sysctl` – https://man7.org/linux/man-pages/man8/sysctl.8.html
- `/etc/sysctl.d/*` – https://man7.org/linux/man-pages/man5/sysctl.d.5.html
- `net_namespaces` – https://man7.org/linux/man-pages/man7/network_namespaces.7.html

Limiting resource usage when users build images 3: Tutorial (3 min)

Apptainer's default configuration forces users to use many download streams and all cores (and as much RAM as possible) when making the final SIF file. Users cannot disable this. But admins can. This is controlled in the configuration file `/etc/apptainer/apptainer.conf`. Change the following settings to more reasonable values:

- `mksquashfs procs` – number of cores used to compress
- `mksquashfs mem` – maximum RAM used to compress
- `download concurrency` – maximum number of simultaneous download streams

Create a container image manually and run it 4: Tutorial (3 min)

Steps

1. Container images have two forms, SIF files and sandbox directories. The former are readonly, but the latter can be modified. Create a sandbox container with Apptainer's `build` command using the `--sandbox` option, by
   `$ apptainer build --sandbox DIRECTORY/ docker://docker.io/library/busybox:musl`
   where `DIRECTORY` is the directory you want it to be placed.

2. Now, you can run the shell inside the container and look around, do modifications, etc. Use Apptainer's `apptainer shell` to enter its shell with the `--writable` option so that it isn't readonly (some tasks like installing packages might also require the `--fakeroot` option). Then, when you are done making modifications (do a quick modification), exit. Note, by default, you start somewhere in your `$HOME` directory outside the container and thus if you aren't careful, you will modify files in your `$HOME` directory instead of the container. This can be done by
   ```
   $ apptainer shell --writable [--fakeroot] DIRECTORY/
   $ cd /
   $ WHATEVER_MODIFICATION_YOU_DO
   $ exit
   ```

3. Now, convert this modified sandbox container image to a SIF container image (with name `NAME.sif`) using Apptainer's `build` command, by
   ```
   $ apptainer build NAME.sif DIRECTORY/
   ```

4. Now, run something inside the container image using Apptainer's `exec` command, by
   ```
   $ apptainer exec NAME.sif EXECUTABLE [ARGS]
   ```
   An example would be `/bin/ls -lh /bin`

Further Reading

- `apptainer build` – https://apptainer.org/docs/user/main/cli/apptainer_build.html
- `apptainer exec` – https://apptainer.org/docs/user/main/cli/apptainer_exec.html
- `apptainer shell` – https://apptainer.org/docs/user/main/cli/apptainer_shell.html

Use a definition file 5: Tutorial (2 min)

Steps

1. Build a new container image the same way you did in the previous tutorial, but with a singularity defition file. First, write a singularity definition file like

   Listing 1: definition.def

   ```
   Bootstrap: docker
   From: docker.io/library/busybox:musl

   %files
       ANY_FILES_YOU_WANT_TO_COPY

   %post
       RUN WHATEVER_MODIFICATION_YOU_DO

   %runscript
       exec EXECUTABLE [DEFINED_ARGS]
   ```

   Unlike `Dockerfile/Containerfile`s, you don't need to use `&&` to join subcommands. Also, unlike `Dockerfile/Containerfiles`s, you can have only one of each section in a single stage build. The syntax for the `%files` section is that each line is a file or directry with two space sparated parts. The first part is the path on the host system (either absolute or relative to where you run Apptainer's `build` command), and the second part is the path inside the container.

2. Build it with Apptainer's `build` command, by
   ```
   $ apptainer build NAME2.sif definition.def
   ```

3. The resulting SIF file is executable. If you run it directly as in
   `$ ./NAME2.sif`
   or with Apptainer's `run` command like
   `$ apptainer run NAME2.sif`
   whatever you put in the `%runscript` section will run. Run it.

Further Reading

- `apptainer build` – https://apptainer.org/docs/user/main/cli/apptainer_build.html
- `apptainer run` – https://apptainer.org/docs/user/main/cli/apptainer_run.html