freja.nordsiek@gwdg.de

Freja Nordsiek

Containers

For HPC

Table of contents

1 The Problem

2 Containers

3 Outlook

4 Clean Up

5 Docker/OCI

6 Singularity/Apptainer

7 Plenary Discussion

When Users Need Software Admins Can’t Or Won’t Provide

When Users Need Software Admins Can’t Or Won’t Provide

Fully static linked compilation

- Works for single executables
- Needs machine/VM/container with all dependencies

When Users Need Software Admins Can't Or Won't Provide

Fully static linked compilation

- Works for single executables
- Needs machine/VM/container with all dependencies

$HOME Directory Build

- Works for most software
- Often brittle
- Username often baked into compiled code
- Not many good tools

When Users Need Software Admins Can't Or Won't Provide

Fully static linked compilation

- Works for single executables
- Needs machine/VM/container with all dependencies

$HOME Directory Build

- Works for most software
- Often brittle
- Username often baked into compiled code
- Not many good tools

Virtual Machines

- Always works
- High overhead
- Admin must enable virtualization for decent performance
- Difficult to provide VM access to data
- Difficult to get results out of VM
- GPU access requires root permissions

Isolating Services from Other Stuff on Servers

Isolating Services from Other Stuff on Servers

Virtual Machines

- Always works, but high overhead
- Extreme isolation, especially if CPU is fully emulated
- Easy to stop, pause, or move to another machine
- Difficult to provide file and raw hardware access to host

Isolating Services from Other Stuff on Servers

Virtual Machines

- Always works, but high overhead
- Extreme isolation, especially if CPU is fully emulated
- Easy to stop, pause, or move to another machine
- Difficult to provide file and raw hardware access to host

chroot

- Any level of isolation desired with extra tools
- Usually some overhead
- Often need to setup part of a linux install inside
- High isolation is a lot of work

Isolating Services from Other Stuff on Servers

VIRTUAL MACHINES – TECHNICALLY A HEAVY CONTAINER

- Always works, but high overhead
- Extreme isolation, especially if CPU is fully emulated
- Easy to stop, pause, or move to another machine
- Difficult to provide file and raw hardware access to host

CHROOT – MOST CONTAINER SYSTEMS DO THIS AND

- Any level of isolation desired with extra tools
- Usually some overhead
- Often need to setup part of a linux install inside
- High isolation is a lot of work

What Is A Container?

Overly Broad And Pedantic Definition

Container — A runnable item that carries most of its dependencies inside itself.

What Is A Container?

Overly Broad And Pedantic Definition

Container — A runnable item that carries most of its dependencies inside itself.

Used Definition

Container — A running or runnable item that carries all dependencies inside itself except for the OS kernel, a dedicated container runtime provided by the host OS, possibly things from other containers, and possibly a small selection of files/directories on the host.

Important Definitions Used In Container Land

Image

The physical file/s that containers are made from and can be transported from one machine to another.

Container Image

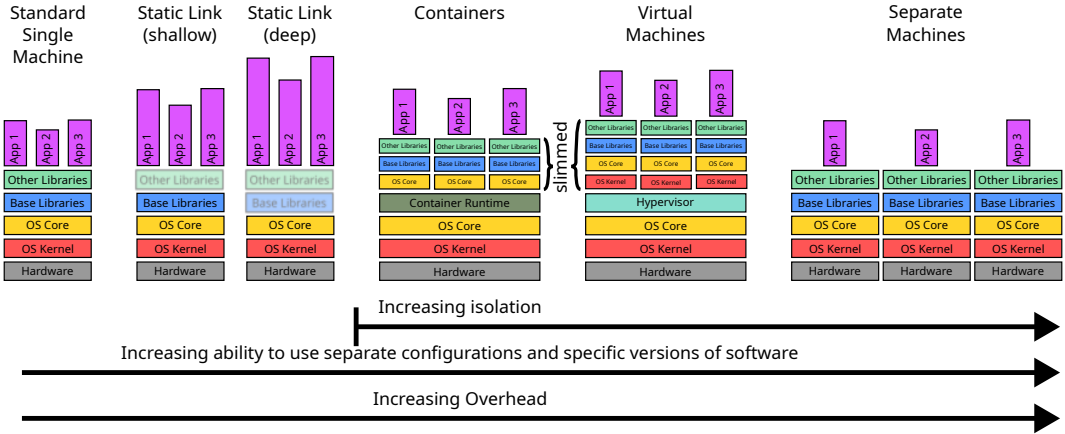
An image imported into the container system and ready to make containers from.

Container

A running or ready to run copy of a container image, its environment, and any changes made to it.

Note: these are often used interchangeably!

Where Containers Fit



Container Ecosystems

- Docker/OCI (Open Container Initiative)
  - Docker
  - Kubernetes
  - Podman, Buildah, Skopeo
  - etc.
- Singularity/Apptainer
- AppImage
- Flatpak
- Snap
- etc.

Container Ecosystems

- **Docker/OCI – for admins and users (limited)**
  - Docker
  - Kubernetes
  - Podman, Buildah, Skopeo
  - etc.
- **Singularity/Apptainer – for HPC users**
- AppImage
- Flatpak
- Snap
- etc.

Runtimes

- Containers need a runtime
  - Set them up from the container image
  - Run them
  - Interface between the container and the host
- Each container ecosystem has its own runtime/s
- Apptainer/Singularity ecosystem
  - Build tool is the runtime
  - Images are compatible between old Singularity, Apptainer, and SingularityCE/PRO
- OCI/Docker ecosystem – plethora of runtimes
  - high-level and low-level
  - compatible from the container side

Runtimes – Root Or Rootless

- Root runtimes
  - Requires root permissions to run
  - Often daemons
  - Great and seamless for services
  - Dangerous to allow users to do (they have sudo access)
  - Default for Docker, Kubernetes, etc.
- Rootless runtimes
  - Requires user namespaces and sometimes fuse, `/etc/subuid`, `/etc/subgid`, and/or a setUID helper program
  - While safer than root, these extra things do bring some dangers and/or extra configuration work
  - usually non-daemon
  - More limited, and some containers cannot build or run
  - Only way Podman, Apptainer/Singularity (with catches), etc.

Outlook

- Docker/OCI ecosystem
  - Rootless containers with Podman
    - Management and security considerations for admins
    - Building and running containers
    - Security considerations in choosing base image
    - Transfer container image to Docker and run it
- Singularity/Apptainer ecosystem
  - Management and security considerations for admins
  - Building and running containers
  - Overlays

File: Dockerfile

```
1 | FROM docker.io/library/alpine:3.17
2 |
3 | COPY outer_message.txt /
4 |
5 | RUN apk add python3 && \
6 |     echo "Message from inside." > /inner_message.txt
7 |
8 | LABEL org.opencontainers.image.authors="me"
```

File: singularity.def

```
1  | Bootstrap: docker
2  | From: docker.io/library/alpine:3.17
3  |
4  | %files
5  |     outer_message.txt /
6  |
7  | %post
8  |     apk add python
9  |     echo "Message from inside." > /inner_message.txt
10 |
11 | %labels
12 |     org.opencontainers.image.authors me
```

Install

```
[cloud@trunk ~]$ sudo dnf install podman apptainer
...
Complete!
```

Build

```
[cloud@trunk ~]$ echo "Message from outside." > outer_message.txt
[cloud@trunk ~]$ podman build --format oci --tag demo:latest --file demo_Dockerfile
...
[cloud@trunk ~]$ podman save --format oci-archive --output demo-latest.tar localhost/demo:latest
...
[cloud@trunk ~]$ apptainer build -F demo.sif demo_singularity.def
...
```

Run

```
[cloud@trunk ~]$ podman container run -it --rm localhost/demo:latest cat /outer_message.txt /inner_message.txt
...
Message from outside.
Message from inside.
[cloud@trunk ~]$ apptainer exec demo.sif cat /outer_message.txt /inner_message.txt
Message from outside.
Message from inside.
```

Cleaning up conflicting packages

Certain packages conflict with what we will be doing today

- Use the same names for executables due to compatibility and/or different builds
- Bug/issue in the currently available RPMs (distro specific)

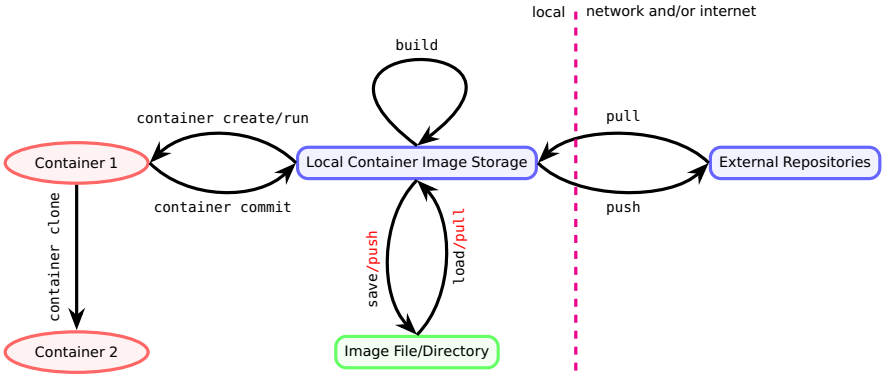
Remove conflicting packages

```
[cloud@trunk ~]$ sudo dnf remove singularity singularity-ce apptainer-suid docker-ce
```

What Is What

- Docker (https://www.docker.com)
  - Not the first: predated by chroot jails and LXC/LXD
  - Got the ball rolling and cemented a lot of the ideas
  - Most well known tool in ecosystem
  - Sets the tone in the ecosystem
- OCI (Open Container Initiative) (https://opencontainers.org)
  - Standardization of images, runtimes, metadata, and interchange
  - Based on Docker
  - Most tools are compatible with OCI and Docker
- Podman (https://podman.io)
  - $\approx$ rootless non-daemon Docker
  - Docker compatible CLI, but with extra things added
  - Can convert Singularity/Apptainer containers images to Docker/OCI
  - Related tools: Buildah (https://buildah.io) and Skopeo (https://github.com/containers/skopeo)

Podman/Docker Tool Picture



Install Podman – Tutorial item 1

Install

```
[cloud@trunk ~]$ sudo dnf install podman
...
Installed:
  conmon-3:2.1.5-1.module_el8.8.0+1254+78119b6e.x86_64
  container-selinux-2:2.195.1-1.module_el8.8.0+1254+78119b6e.noarch
  containernetworking-plugins-1:1.1.1-3.module_el8.7.0+1216+b022c01d.x86_64
  containers-common-2:1-49.module_el8.8.0+1254+78119b6e.x86_64
  criu-3.15-3.module_el8.7.0+1216+b022c01d.x86_64
  fuse-common-3.3.0-16.el8.x86_64
  fuse-overlayfs-1.10-1.module_el8.8.0+1254+78119b6e.x86_64
  fuse3-3.3.0-16.el8.x86_64
  fuse3-libs-3.3.0-16.el8.x86_64
  libnet-1.1.6-15.el8.x86_64
  libslirp-4.4.0-1.module_el8.7.0+1216+b022c01d.x86_64
  podman-3:4.3.1-2.module_el8.8.0+1254+78119b6e.x86_64
  podman-catatonit-3:4.3.1-2.module_el8.8.0+1254+78119b6e.x86_64
  policycoreutils-python-utils-2.9-21.1.el8.noarch
  protobuf-c-1.3.0-6.el8.x86_64
  runc-1:1.1.4-1.module_el8.7.0+1216+b022c01d.x86_64
  shadow-utils-subid-2:4.6-17.el8.x86_64
  slirp4netns-1.2.0-2.module_el8.7.0+1216+b022c01d.x86_64
  tar-2:1.30-8.el8.x86_64

Complete!
[cloud@trunk ~]$
```

Other important dependencies from `sudo dnf repoquery -deplist PACKAGE`

- iptables
- nftables
- libseccomp
- shadow-utils-subid
- gpgme
- libgpg-error

Install Podman – Tutorial item 1

Install

```
[cloud@trunk ~]$ sudo dnf install podman
...
Installed:
  conmon-3:2.1.5-1.module_el8.8.0+1254+78119b6e.x86_64
  container-selinux-2:2.195.1-1.module_el8.8.0+1254+78119b6e.noarch
  containernetworking-plugins-1:1.1.1-3.module_el8.7.0+1216+b022c01d.x86_64
  containers-common-2:1-49.module_el8.8.0+1254+78119b6e.x86_64
  criu-3.15-3.module_el8.7.0+1216+b022c01d.x86_64
  fuse-common-3.3.0-16.el8.x86_64
  fuse-overlayfs-1.10-1.module_el8.8.0+1254+78119b6e.x86_64
  fuse3-3.3.0-16.el8.x86_64
  fuse3-libs-3.3.0-16.el8.x86_64
  libnet-1.1.6-15.el8.x86_64
  libslirp-4.4.0-1.module_el8.7.0+1216+b022c01d.x86_64
  podman-3:4.3.1-2.module_el8.8.0+1254+78119b6e.x86_64
  podman-catatonit-3:4.3.1-2.module_el8.8.0+1254+78119b6e.x86_64
  policycoreutils-python-utils-2.9-21.1.el8.noarch
  protobuf-c-1.3.0-6.el8.x86_64
  runc-1:1.1.4-1.module_el8.7.0+1216+b022c01d.x86_64
  shadow-utils-subid-2:4.6-17.el8.x86_64
  slirp4netns-1.2.0-2.module_el8.7.0+1216+b022c01d.x86_64
  tar-2:1.30-8.el8.x86_64

Complete!
[cloud@trunk ~]$
```

Other important dependencies from `sudo dnf repoquery -deplist PACKAGE`

- iptables
- nftables
- libseccomp
- **shadow-utils-subid**
- gpgme
- libgpg-error

User subuids And subgids – Tutorial item 2

- Linux systems have many users and groups
- Thus, many containers have them
- Many tools crash if all files are non-root user
- Must trick containers
  - Make user look like root inside
  - And at least one of the following
    1. Many aliases that are just the user but different uids (shadow-utils-subid)
    2. Map all files to root and override internal permissions checks (fakeroot)

User subuids And subgids – Tutorial item 2

- Linux systems have many users and groups
- Thus, many containers have them
- Many tools crash if all files are non-root user
- Must trick containers
  - Make user look like root inside
  - And at least one of the following
    1. **Many aliases that are just the user but different uids (shadow-utils-subid)**
    2. Map all files to root and override internal permissions checks (fakeroot)

User subuids And subgids – Tutorial item 2

- Linux systems have many users and groups
- Thus, many containers have them
- Many tools crash if all files are non-root user
- Must trick containers
  - Make user look like root inside
  - And at least one of the following
    1. **Many aliases that are just the user but different uids (shadow-utils-subid)**
    2. Map all files to root and override internal permissions checks (fakeroot)

/etc/subuid

```
1 | cloud:100000:65536
```

/etc/subgid

```
1 | cloud:100000:65536
```

User subuids And subgids – Tutorial item 2

- Linux systems have many users and groups
- Thus, many containers have them
- Many tools crash if all files are non-root user
- Must trick containers
  - Make user look like root inside
  - And at least one of the following
    1. **Many aliases that are just the user but different uids (shadow-utils-subid)**
    2. Map all files to root and override internal permissions checks (fakeroot)

/etc/subuid

```
1 | cloud:100000:65536
```

/etc/subgid

```
1 | cloud:100000:65536
```

Set and remove

```
[cloud@trunk ~]$ sudo useradd -r -s /usr/bin/false junk-user
[cloud@trunk ~]$ sudo usermod --add-subuids 200000-265536 junk-user
[cloud@trunk ~]$ sudo usermod --add-subgids 200000-265536 junk-user
[cloud@trunk ~]$ cat /etc/subuid
cloud:100000:65536
junk-user:200000:65537
[cloud@trunk ~]$ cat /etc/subgid
cloud:100000:65536
junk-user:200000:65537
[cloud@trunk ~]$ sudo usermod --del-subuids 200000-265536 junk-user
[cloud@trunk ~]$ sudo usermod --del-subgids 200000-265536 junk-user
[cloud@trunk ~]$ sudo userdel junk-user
```

User Namespaces – Tutorial item 3

- **CRITICAL** to rootless containers
- Imitation of admin permissions in an isolated environment
- Look like root and subuids/subgids inside
- Relatively young, so there might still be some bugs allowing privilege escalation
- But is now reasonably safe in newer kernels (safer than sudo or setUID)
- Disabled by default in some distros
- Check and set with `sysctl`
- Permanent settings in `/etc/sysctl.d/`

User Namespaces – Tutorial item 3

- **CRITICAL** to rootless containers
- Imitation of admin permissions in an isolated environment
- Look like root and subuids/subgids inside
- Relatively young, so there might still be some bugs allowing privilege escalation
- But is now reasonably safe in newer kernels (safer than sudo or setUID)
- Disabled by default in some distros
- Check and set with `sysctl`
- Permanent settings in `/etc/sysctl.d/`

Check, disable, enable

```
[cloud@trunk etc]$ sudo sysctl -a | grep user.*namespace
user.max_cgroup_namespaces = 14527
user.max_ipc_namespaces = 14527
user.max_mnt_namespaces = 14527
user.max_net_namespaces = 14527
user.max_pid_namespaces = 14527
user.max_time_namespaces = 14527
user.max_user_namespaces = 14527
user.max_uts_namespaces = 14527
[cloud@trunk etc]$ sudo sysctl user.max_user_namespaces=0
user.max_user_namespaces = 0
[cloud@trunk etc]$ sudo sysctl user.max_user_namespaces=14527
user.max_user_namespaces = 14527
```

OCI/Docker Container Images

Addressing/name

`[TRANSPORT:]LOCATION/NAME[:TAG]`

Examples

- `localhost/mycont:latest`
- `docker.io/library/debian:bookworm`
- `quay.io/rockylinux/rockylinux:8.6`

All have

- image manifest
- stack of layers – each layer is the file/s and other changes on top of previous layer

Skopeo

Tool for inspecting and modifying metadata in registries, push, pull, and inter-registry copy.

OCI/Docker Container Images

Addressing/name

`[TRANSPORT:]LOCATION/NAME[:TAG]`

Examples

- `localhost/mycont:latest`
- `docker.io/library/debian:bookworm`
- `quay.io/rockylinux/rockylinux:8.6`

All have

- image manifest
- stack of layers – each layer is the file/s and other changes on top of previous layer

Skopeo

Tool for inspecting and modifying metadata in registries, push, pull, and inter-registry copy.

podman inspect

```
[cloud@trunk ~]$ podman pull docker.io/library/busybox:musl
...
[cloud@trunk ~]$ podman inspect docker.io/library/busybox:musl
```

skopeo inspect

```
[cloud@trunk ~]$ sudo dnf install skopeo
...
[cloud@trunk ~]$ skopeo inspect -n docker://docker.io/library/busybox:musl
```

OCI/Docker Container Images

Addressing/name

`[TRANSPORT:]LOCATION/NAME[:TAG]`

Examples

- `localhost/mycont:latest`
- `docker.io/library/debian:bookworm`
- `quay.io/rockylinux/rockylinux:8.6`

All have

- image manifest
- stack of layers – each layer is the file/s and other changes on top of previous layer

Skopeo

Tool for inspecting and modifying metadata in registries, push, pull, and inter-registry copy.

podman inspect

```
[cloud@trunk ~]$ podman pull docker.io/library/busybox:musl
...
[cloud@trunk ~]$ podman inspect docker.io/library/busybox:musl
```

skopeo inspect

```
[cloud@trunk ~]$ sudo dnf install skopeo
...
[cloud@trunk ~]$ skopeo inspect -n docker://docker.io/library/busybox:musl
```

metadata by Skopeo


```
{
    "Name": "docker.io/library/busybox",
    "Digest": "sha256:e7dc28a9c45363cb558fd4a03bc65a21b602a4fd744d48a4002790ea2c988178",
    "RepoTags": [],
    "Created": "2023-01-04T01:19:54.861751696Z",
    "DockerVersion": "20.10.12",
    "Labels": null,
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
        "sha256:e96e863613a1a11d149cad16301bcb8793a2295d677d783053ea489d65269d70"
    ],
    "LayersData": [
        {
            "MIMEType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
            "Digest": "sha256:e96e863613a1a11d149cad16301bcb8793a2295d677d783053ea489d65269d70",
            "Size": 855342,
            "Annotations": null
        }
    ],
    "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
    ]
}
```

Building A New Container Image Manually – Tutorial items 4 & 5

create

```
[cloud@trunk ~]$ podman container create -t --name=mycont \
> --env=MYVAR="what do we have here" docker.io/library/alpine:3.17
Trying to pull docker.io/library/alpine:3.17...
Getting image source signatures
Copying blob 63b65145d645 done
Copying config b2aa39c304 done
Writing manifest to image destination
Storing signatures
d14159214976c4f67b26e912a3d74bb8c5788a183fca65265adac2c65334c47f
[cloud@trunk ~]$ podman container init mycont
mycont
```

Building A New Container Image Manually – Tutorial items 4 & 5

create

```
[cloud@trunk ~]$ podman container create -t --name=mycont \
> --env=MYVAR="what do we have here" docker.io/library/alpine:3.17
Trying to pull docker.io/library/alpine:3.17...
Getting image source signatures
Copying blob 63b65145d645 done
Copying config b2aa39c304 done
Writing manifest to image destination
Storing signatures
d14159214976c4f67b26e912a3d74bb8c5788a183fca65265adac2c65334c47f
[cloud@trunk ~]$ podman container init mycont
mycont
```

start and edit

```
[cloud@trunk ~]$ podman container start -a -i mycont
/ # echo "Hello from inside" > inner_message.txt
/ # echo "#!/bin/sh" > launcher.sh
/ # echo "cat /outer_message.txt" >> launcher.sh
/ # echo "cat /inner_message.txt" >> launcher.sh
/ # echo "echo \"\$MYVAR\"" >> launcher.sh
/ # chmod +x launcher.sh
/ # exit
```

Building A New Container Image Manually – Tutorial items 4 & 5

create

```
[cloud@trunk ~]$ podman container create -t --name=mycont \
> --env=MYVAR="what do we have here" docker.io/library/alpine:3.17
Trying to pull docker.io/library/alpine:3.17...
Getting image source signatures
Copying blob 63b65145d645 done
Copying config b2aa39c304 done
Writing manifest to image destination
Storing signatures
d14159214976c4f67b26e912a3d74bb8c5788a183fca65265adac2c65334c47f
[cloud@trunk ~]$ podman container init mycont
mycont
```

start and edit

```
[cloud@trunk ~]$ podman container start -a -i mycont
/ # echo "Hello from inside" > inner_message.txt
/ # echo "#!/bin/sh" > launcher.sh
/ # echo "cat /outer_message.txt" >> launcher.sh
/ # echo "cat /inner_message.txt" >> launcher.sh
/ # echo "echo \"\$MYVAR\"" >> launcher.sh
/ # chmod +x launcher.sh
/ # exit
```

copy file into

```
[cloud@trunk ~]$ echo "Hello from outside" > outer_message.txt
[cloud@trunk ~]$ podman container cp outer_message.txt mycont:/outer_message.txt
```

Building A New Container Image Manually – Tutorial items 4 & 5

create

```
[cloud@trunk ~]$ podman container create -t --name=mycont \
> --env=MYVAR="what do we have here" docker.io/library/alpine:3.17
Trying to pull docker.io/library/alpine:3.17...
Getting image source signatures
Copying blob 63b65145d645 done
Copying config b2aa39c304 done
Writing manifest to image destination
Storing signatures
d14159214976c4f67b26e912a3d74bb8c5788a183fca65265adac2c65334c47f
[cloud@trunk ~]$ podman container init mycont
mycont
```

start and edit

```
[cloud@trunk ~]$ podman container start -a -i mycont
/ # echo "Hello from inside" > inner_message.txt
/ # echo "#!/bin/sh" > launcher.sh
/ # echo "cat /outer_message.txt" >> launcher.sh
/ # echo "cat /inner_message.txt" >> launcher.sh
/ # echo "echo \"\$MYVAR\"" >> launcher.sh
/ # chmod +x launcher.sh
/ # exit
```

copy file into

```
[cloud@trunk ~]$ echo "Hello from outside" > outer_message.txt
[cloud@trunk ~]$ podman container cp outer_message.txt mycont:/outer_message.txt
```

commit

```
[cloud@trunk ~]$ podman container commit --change CMD=/launcher.sh mycont mycont:latest
Getting image source signatures
Copying blob 7cd52847ad77 skipped: already exists
Copying blob 69676c0c01c5 done
Copying config 22d44f454f done
Writing manifest to image destination
Storing signatures
22d44f454f9e5830e174b77240e2b66b0495afe405c6d97edc207fbf97db71cd
[cloud@trunk ~]$ podman container rm mycont
mycont
```

Building A New Container Image Manually – Tutorial items 4 & 5

create

```
[cloud@trunk ~]$ podman container create -t --name=mycont \
> --env=MYVAR="what do we have here" docker.io/library/alpine:3.17
Trying to pull docker.io/library/alpine:3.17...
Getting image source signatures
Copying blob 63b65145d645 done
Copying config b2aa39c304 done
Writing manifest to image destination
Storing signatures
d14159214976c4f67b26e912a3d74bb8c5788a183fca65265adac2c65334c47f
[cloud@trunk ~]$ podman container init mycont
mycont
```

start and edit

```
[cloud@trunk ~]$ podman container start -a -i mycont
/ # echo "Hello from inside" > inner_message.txt
/ # echo "#!/bin/sh" > launcher.sh
/ # echo "cat /outer_message.txt" >> launcher.sh
/ # echo "cat /inner_message.txt" >> launcher.sh
/ # echo "echo \"\$MYVAR\"" >> launcher.sh
/ # chmod +x launcher.sh
/ # exit
```

copy file into

```
[cloud@trunk ~]$ echo "Hello from outside" > outer_message.txt
[cloud@trunk ~]$ podman container cp outer_message.txt mycont:/outer_message.txt
```

commit

```
[cloud@trunk ~]$ podman container commit --change CMD=/launcher.sh mycont mycont:latest
Getting image source signatures
Copying blob 7cd52847ad77 skipped: already exists
Copying blob 69676c0c01c5 done
Copying config 22d44f454f done
Writing manifest to image destination
Storing signatures
22d44f454f9e5830e174b77240e2b66b0495afe405c6d97edc207fbf97db71cd
[cloud@trunk ~]$ podman container rm mycont
mycont
```

run

```
[cloud@trunk ~]$ podman run -i -t --rm localhost/mycont:latest
Hello from outside
Hello from inside
what do we have here
```

Building A New Container Image Manually – Problems

- Scales poorly when there are many steps
- Hard to repeat
  - Memories fade
  - Commands run inside the container may not be recorded at all, the `~/.bash_history` or equivalent in the container may be short, etc.
- A bug means having to repeat every step from the beginning (time consuming)
- Doesn't record history (history must be set or externally documented manually)
- Hard to share build recipe with others
- Harder for others to trust your container images (can't easily just build themselves from a recipe)

Using A Dockerfile/Containerfile – Tutorial items 6 & 5

Containerfile

```
1  FROM docker.io/library/alpine:3.17
2
3  ENV MYVAR="what do we have here"
4
5  RUN echo "Hello from inside" > inner_message.txt && \
6      echo "#!/bin/sh" > launcher.sh && \
7      echo "cat /outer_message.txt" >> launcher.sh && \
8      echo "cat /inner_message.txt" >> launcher.sh && \
9      echo "echo \"\$MYVAR\"" >> launcher.sh && \
10     chmod +x launcher.sh
11
12 COPY outer_message.txt /outer_message.txt
13
14 CMD /launcher.sh
```

Docker, *Dockerfile reference*, 2023
https://docs.docker.com/engine/reference/builder

Each directive adds a layer, so “&& \” is used to stitch commands together in a `RUN`.

Using A Dockerfile/Containerfile – Tutorial items 6 & 5

Containerfile

```
1  FROM docker.io/library/alpine:3.17
2
3  ENV MYVAR="what do we have here"
4
5  RUN echo "Hello from inside" > inner_message.txt && \
6      echo "#!/bin/sh" > launcher.sh && \
7      echo "cat /outer_message.txt" >> launcher.sh && \
8      echo "cat /inner_message.txt" >> launcher.sh && \
9      echo "echo \"\$MYVAR\"" >> launcher.sh && \
10     chmod +x launcher.sh
11
12 COPY outer_message.txt /outer_message.txt
13
14 CMD /launcher.sh
```

build

```
[cloud@trunk ~]$ echo "Hello from outside" > outer_message.txt
[cloud@trunk ~]$ podman build --tag mycont:latest --file Containerfile
STEP 1/5: FROM docker.io/library/alpine:3.17
Trying to pull docker.io/library/alpine:3.17...
Getting image source signatures
Copying blob 63b65145d645 done
Copying config b2aa39c304 done
Writing manifest to image destination
Storing signatures
STEP 2/5: ENV MYVAR="what do we have here"
--> cb6823b1b9b
STEP 3/5: RUN echo "Hello from inside" > inner_message.txt &&     echo "#!/bin/sh" > launcher.sh &&
    echo "cat /outer_message.txt" >> launcher.sh &&     echo "cat /inner_message.txt" >> launcher.sh
 &&     echo "echo \"\$MYVAR\"" >> launcher.sh &&     chmod +x launcher.sh
--> 51f5bff3e29
STEP 4/5: COPY outer_message.txt /outer_message.txt
--> e6cf4226c9a
STEP 5/5: CMD /launcher.sh
COMMIT mycont:latest
--> dc49939d596
Successfully tagged localhost/mycont:latest
dc49939d5961e1655e03c50ae43203633f337a659302cc31084896a17a42f644
```

Docker, *Dockerfile reference*, 2023
https://docs.docker.com/engine/reference/builder

Each directive adds a layer, so “&& \” is used to stitch commands together in a RUN.

Using A Dockerfile/Containerfile – Tutorial items 6 & 5

Containerfile

```
1  FROM docker.io/library/alpine:3.17
2
3  ENV MYVAR="what do we have here"
4
5  RUN echo "Hello from inside" > inner_message.txt && \
6      echo "#!/bin/sh" > launcher.sh && \
7      echo "cat /outer_message.txt" >> launcher.sh && \
8      echo "cat /inner_message.txt" >> launcher.sh && \
9      echo "echo \"\$MYVAR\"" >> launcher.sh && \
10     chmod +x launcher.sh
11
12 COPY outer_message.txt /outer_message.txt
13
14 CMD /launcher.sh
```

Docker, *Dockerfile reference*, 2023
https://docs.docker.com/engine/reference/builder

Each directive adds a layer, so “&& \” is used to stitch commands together in a RUN.

build

```
[cloud@trunk ~]$ echo "Hello from outside" > outer_message.txt
[cloud@trunk ~]$ podman build --tag mycont:latest --file Containerfile
STEP 1/5: FROM docker.io/library/alpine:3.17
Trying to pull docker.io/library/alpine:3.17...
Getting image source signatures
Copying blob 63b65145d645 done
Copying config b2aa39c304 done
Writing manifest to image destination
Storing signatures
STEP 2/5: ENV MYVAR="what do we have here"
--> cb6823b1b9b
STEP 3/5: RUN echo "Hello from inside" > inner_message.txt &&     echo "#!/bin/sh" > launcher.sh &&
    echo "cat /outer_message.txt" >> launcher.sh &&     echo "cat /inner_message.txt" >> launcher.sh
 &&     echo "echo \"\$MYVAR\"" >> launcher.sh &&     chmod +x launcher.sh
--> 51f5bff3e29
STEP 4/5: COPY outer_message.txt /outer_message.txt
--> e6cf4226c9a
STEP 5/5: CMD /launcher.sh
COMMIT mycont:latest
--> dc49939d596
Successfully tagged localhost/mycont:latest
dc49939d5961e1655e03c50ae43203633f337a659302cc31084896a17a42f644
```

run

```
[cloud@trunk ~]$ podman run -i -t --rm localhost/mycont:latest
Hello from outside
Hello from inside
what do we have here
```

Choosing The Base Without Opening Pandora's Box – Part 1

Remember

Containerfile

```
1 | FROM docker.io/library/alpine:3.17
```

create

```
[cloud@trunk ~]$ podman container create -t --name=mycont \
> --env=MYVAR="what do we have here" docker.io/library/alpine:3.17
```

Choosing The Base Without Opening Pandora's Box – Part 1

Remember

Containerfile

```
1 | FROM docker.io/library/alpine:3.17
```

create

```
[cloud@trunk ~]$ podman container create -t --name=mycont \
> --env=MYVAR="what do we have here" docker.io/library/alpine:3.17
```

Chooses the base image

- From an external registry (in the example)
- From local storage (previously made image)
- `scratch` (empty container with no files or directories)

Choosing The Base Without Opening Pandora's Box – Part 1

Remember

Containerfile

```
1 | FROM docker.io/library/alpine:3.17
```

create

```
[cloud@trunk ~]$ podman container create -t --name=mycont \
> --env=MYVAR="what do we have here" docker.io/library/alpine:3.17
```

Chooses the base image

- From an external registry (in the example)
- From local storage (previously made image)
- `scratch` (empty container with no files or directories)

Do you trust the base image?

- Who made it?
- Are you actually getting the one you intended?
- Are you actually getting the version you trust?

Choosing The Base Without Opening Pandora's Box – Part 1

Remember

Containerfile

```
1 | FROM docker.io/library/alpine:3.17
```

create

```
[cloud@trunk ~]$ podman container create -t --name=mycont \
> --env=MYVAR="what do we have here" docker.io/library/alpine:3.17
```

Chooses the base image

- From an external registry (in the example)
- From local storage (previously made image)
- `scratch` (empty container with no files or directories)

Do you trust the base image?

- Who made it?
- Are you actually getting the one you intended?
- Are you actually getting the version you trust?

Why care?

- Registries of varying trustworthiness
- Many registries are a "Wild West" just like PyPI, NPM, etc.
- Base images are high value targets to compromise
- History can be forged

Choosing The Base Without Opening Pandora's Box – Part 2

- Research registry
- Research image and who made it
- Always include the registry URL and image namespace in the `LOCATION`
- Tags can be changed, so it can be very useful to pin to a specific version with its digest checksum
- If you have doubts, look for a more trustworthy base image
- If you still have doubts
  1. Find the Dockerfile/Containerfile, or rebuild it from the image's history
  2. Inspect it
  3. Build the container image yourself
- Building with a rootless tool reduces the damage of a container escape
- Further reduce permissions
- Don't forget the base image of your base image

Choosing The Base Without Opening Pandora's Box – Part 3

With tag

$$\overbrace{\textit{docker.io}}^{\text{registry}} / \overbrace{\textit{library}}^{\text{namespace}} / \overbrace{\textit{alpine}}^{\text{name}} : \overbrace{3.17}^{\text{tag}}$$

With pinned digest

$$\overbrace{\textit{docker.io}}^{\text{registry}} / \overbrace{\textit{library}}^{\text{namespace}} / \overbrace{\textit{alpine}}^{\text{name}} : \overbrace{@\textit{sha}256 : e2e16842c9b54d985bf1ef9242a313f36b856181f188de21313820e177002501}^{\text{digest}}$$

Exercise 1

Useful links during the exercise

- Docker, *Dockerfile reference*, 2023
  `https://docs.docker.com/engine/reference/builder`
- Podman documentation – `https://docs.podman.io`
- Docker documentation – `https://docs.docker.com`

Tools

- Podman (`https://podman.io`)
- Skopeo (`https://github.com/containers/skopeo`)
- Docker (`https://docker.com`)

What Is What

- Singularity – defunct
  - Container build system, image format, and runtime for HPC users
  - Users have access to `$HOME` by default
  - Easy to inject GPU libraries at runtime
  - SIF containers are compressed single file squashfs archives
- Apptainer (`https://apptainer.org`)
  - Opensource successor to Singularity
  - Renamed when Singularity community moved to the Linux Foundation
  - Defaults to non-setUID build
- SingularityCE/PRO (`https://sylabs.io/singularity`)
  - CE (opensource) and PRO (commercial) successors to Singularity
  - From Sylabs, one of the major players behind Singularity
  - Defaults to setUID build

Differences from Docker/OCI

- Designed more for letting users bring the exact dependencies they need or want rather than isolation (isolation is mostly a means to an end)
- Not designed for services
- rootless (non-setUID build) or quasi-root quasi-rootless (setUID build)
- Container images are SIF files and sandbox directories the user actually gets, rather than being stored in local storage
- SIF files are squashfs filesystem images with a header and can be mounted
- By default, containers are readonly
- By default, `$HOME` and `/tmp` are mounted inside
- Gives admins options to achieve functionality rather than fixed dependencies for some features

To setUID or not setUID Part 1

- Make user namespaces
  - Make one using the power of root if compiled with setUID
  - Do unprivileged if `user.max_user_namespaces` > 0
- Trick containers on file ownership and users/groups
  - Use `/etc/subuid` and `/etc/subgid` if possible (either using them directly if compiled with setUID or using `newuidmap` and `newgidmap` otherwise)
  - Map root to user and all files to root as root otherwise, and use `fakeroot` if available to make this work better
- Mounting
  - Use the power of root if compiled with setUID
  - Use `fuse` otherwise, though less so on newer kernels

To setUID or not setUID Part 2

setUID build

- Makes a helper program with setUID
- Can do most tasks without help from other programs and admin configuration
- Any bugs and design flaws in setUID helper risk privilege escalation
- Default build for SingularityCE/PRO and old Singularity

non-setUID build

- Needs admin to allow unprivileged user namespaces
- Needs more system packages as helper programs for functionality
- Often needs to use `fuse`
- Unprivileged user namespaces, the helper programs, and `fuse` have their own security considerations
- Default for Apptainer

Install Apptainer – Tutorial item 1

Install

```
[cloud@trunk ~]$ sudo dnf install apptainer
...
Installed:
  apptainer-1.1.5-2.el8.x86_64                          fakeroot-1.30.1-1.el8.x86_64
  fakeroot-libs-1.30.1-1.el8.x86_64                     fuse-common-3.3.0-16.el8.x86_64
  fuse-overlayfs-1.10-1.module_el8.8.0+1254+78119b6e.x86_64  fuse3-3.3.0-16.el8.x86_64
  fuse3-libs-3.3.0-16.el8.x86_64                        squashfuse-0.1.104-1.el8.x86_64
  squashfuse-libs-0.1.104-1.el8.x86_64

Complete!
[cloud@trunk ~]$
```

Other important dependencies from
`sudo dnf repoquery -deplist PACKAGE`

- libseccomp
- libz

Note, the package repo also provides `apptainer-suid` for a setUID build.

Subuids, subgids, and namespaces – Tutorial item 2

Already handled during Podman section

- subuids and subgids in `/etc/subuid` and `/etc/subgid`
- enabling unprivileged user namespaces (`user.max_user_namespaces`)

Network namespaces

- Seldom needed (`-net` argument is rarely used by users)
- Most exploits of user namespaces require the combination with network namespaces
- Recommend disabling if not needed for something else (Podman usage needs them more often)

Check, enable, and disable

```
[cloud@trunk etc]$ sudo sysctl -a | grep user.*namespace
user.max_cgroup_namespaces = 14527
user.max_ipc_namespaces = 14527
user.max_mnt_namespaces = 14527
user.max_net_namespaces = 14527
user.max_pid_namespaces = 14527
user.max_time_namespaces = 14527
user.max_user_namespaces = 14527
user.max_uts_namespaces = 14527
[cloud@trunk etc]$ sudo sysctl user.max_net_namespaces=14527
user.max_net_namespaces = 14527
[cloud@trunk etc]$ sudo sysctl user.max_net_namespaces=0
user.max_net_namespaces = 0
```

Other security considerations and non-considerations Part 1

fakeroot

- no issues – completely unprivileged
- users could build and use themselves by doing a `$HOME` directory build of `fakeroot`
- Many container images fail to build and/or run without it

shadow-utils-subid

- setUID `newuidmap` and `newgidmap` programs from package
- Optional
- Works without, just some container images will fail to build and/or run

Other security considerations and non-considerations Part 2

fuse

- Optional, and not used for building container images at all
- Often already present and allowed due to other tools (but can use selinux and other tools to disable for Singularity/Apptainer)
- Works without, but must unpack all SIF images to a directory before running and overlays don't work

setUID `fusermount` and/or `fusermount3`

- setUID to load the `fuse` module and allow users to do limited mounts
- If `fuse` and `ovleray` modules are loaded at boot, the program/s isn't/aren't actually needed at all and can be deleted or setUID disabled with `chmod u-s /usr/bin/fusermount*`

Resource starvation while building – Tutorial item 3

By default, when making SIF files

- All cores used
- Use as much RAM as it wants

By default, when downloading

- Use up to 3 download streams

Resource starvation while building – Tutorial item 3

By default, when making SIF files

- All cores used
- Use as much RAM as it wants

By default, when downloading

- Use up to 3 download streams

The problem

- Starve other users from resources
- Not much of an issue for small images
- But big issue for $>$ 1 GiB images
- Users don't know about this
- Users can't fix it in old Apptainer/Singularity

Resource starvation while building – Tutorial item 3

By default, when making SIF files

- All cores used
- Use as much RAM as it wants

By default, when downloading

- Use up to 3 download streams

The problem

- Starve other users from resources
- Not much of an issue for small images
- But big issue for $>$ 1 GiB images
- Users don't know about this
- Users can't fix it in old Apptainer/Singularity

`/etc/apptainer/apptainer.conf`

Set the following to more reasonable values for a shared system

- `mksquashfs procs`
- `mksquashfs mem`
- `download concurrency`

Building A SIF Manually – Tutorial item 4

create sandbox

```
[cloud@trunk ~]$ apptainer build --sandbox manual/ docker://docker.io/library/alpine:3.17
INFO:    Starting build...
Getting image source signatures
Copying blob 63b65145d645 skipped: already exists
Copying config 6a2bcc1c7b done
Writing manifest to image destination
Storing signatures
2023/02/18 11:46:27  info unpack layer: sha256:63b65145d645c1250c391b2d16ebe53b3747c295ca8ba2fcb6b0cf064a4dc21c
WARNING: The sandbox contain files/dirs that cannot be removed with 'rm'.
WARNING: Use 'chmod -R u+rwX' to set permissions that allow removal.
WARNING: Use the '--fix-perms' option to 'apptainer build' to modify permissions at build time.
INFO:    Creating sandbox directory...
INFO:    Build complete: manual/
```

Building A SIF Manually – Tutorial item 4

create sandbox

```
[cloud@trunk ~]$ apptainer build --sandbox manual/ docker://docker.io/library/alpine:3.17
INFO:    Starting build...
Getting image source signatures
Copying blob 63b65145d645 skipped: already exists
Copying config 6a2bcc1c7b done
Writing manifest to image destination
Storing signatures
2023/02/18 11:46:27  info unpack layer: sha256:63b65145d645c1250c391b2d16ebe53b3747c295ca8ba2fcb6b0cf064a4dc21c
WARNING: The sandbox contain files/dirs that cannot be removed with 'rm'.
WARNING: Use 'chmod -R u+rwX' to set permissions that allow removal.
WARNING: Use the '--fix-perms' option to 'apptainer build' to modify permissions at build time.
INFO:    Creating sandbox directory...
INFO:    Build complete: manual/
```

start a shell inside and edit

```
[cloud@trunk ~]$ apptainer shell --writable --fakeroot manual/
WARNING: Skipping mount /etc/localtime [binds]: /etc/localtime doesn't exist in container
Apptainer> cd /
Apptainer> echo "Hello from inside" > inner_message.txt
Apptainer> echo "#!/bin/sh" > launcher.sh
Apptainer> echo "cat /outer_message.txt" >> launcher.sh
Apptainer> echo "cat /inner_message.txt" >> launcher.sh
Apptainer> chmod +x launcher.sh
Apptainer> exit
```

Building A SIF Manually – Tutorial item 4

create sandbox

```
[cloud@trunk ~]$ apptainer build --sandbox manual/ docker://docker.io/library/alpine:3.17
INFO:    Starting build...
Getting image source signatures
Copying blob 63b65145d645 skipped: already exists
Copying config 6a2bcc1c7b done
Writing manifest to image destination
Storing signatures
2023/02/18 11:46:27  info unpack layer: sha256:63b65145d645c1250c391b2d16ebe53b3747c295ca8ba2fcb6b0cf064a4dc21c
WARNING: The sandbox contain files/dirs that cannot be removed with 'rm'.
WARNING: Use 'chmod -R u+rwX' to set permissions that allow removal.
WARNING: Use the '--fix-perms' option to 'apptainer build' to modify permissions at build time.
INFO:    Creating sandbox directory...
INFO:    Build complete: manual/
```

start a shell inside and edit

```
[cloud@trunk ~]$ apptainer shell --writable --fakeroot manual/
WARNING: Skipping mount /etc/localtime [binds]: /etc/localtime doesn't exist in container
Apptainer> cd /
Apptainer> echo "Hello from inside" > inner_message.txt
Apptainer> echo "#!/bin/sh" > launcher.sh
Apptainer> echo "cat /outer_message.txt" >> launcher.sh
Apptainer> echo "cat /inner_message.txt" >> launcher.sh
Apptainer> chmod +x launcher.sh
Apptainer> exit
```

copy file into

```
[cloud@trunk ~]$ echo "Hello from outside" > outer_message.txt
[cloud@trunk ~]$ cp outer_message.txt manual/
```

Building A SIF Manually – Tutorial item 4

create sandbox

```
[cloud@trunk ~]$ apptainer build --sandbox manual/ docker://docker.io/library/alpine:3.17
INFO:    Starting build...
Getting image source signatures
Copying blob 63b65145d645 skipped: already exists
Copying config 6a2bcc1c7b done
Writing manifest to image destination
Storing signatures
2023/02/18 11:46:27  info unpack layer: sha256:63b65145d645c1250c391b2d16ebe53b3747c295ca8ba2fcb6b0cf064a4dc21c
WARNING: The sandbox contain files/dirs that cannot be removed with 'rm'.
WARNING: Use 'chmod -R u+rwX' to set permissions that allow removal.
WARNING: Use the '--fix-perms' option to 'apptainer build' to modify permissions at build time.
INFO:    Creating sandbox directory...
INFO:    Build complete: manual/
```

start a shell inside and edit

```
[cloud@trunk ~]$ apptainer shell --writable --fakeroot manual/
WARNING: Skipping mount /etc/localtime [binds]: /etc/localtime doesn't exist in container
Apptainer> cd /
Apptainer> echo "Hello from inside" > inner_message.txt
Apptainer> echo "#!/bin/sh" > launcher.sh
Apptainer> echo "cat /outer_message.txt" >> launcher.sh
Apptainer> echo "cat /inner_message.txt" >> launcher.sh
Apptainer> chmod +x launcher.sh
Apptainer> exit
```

copy file into

```
[cloud@trunk ~]$ echo "Hello from outside" > outer_message.txt
[cloud@trunk ~]$ cp outer_message.txt manual/
```

convert to SIF

```
[cloud@trunk ~]$ apptainer build manual.sif manual/
INFO:    Starting build...
INFO:    Creating SIF file...
INFO:    Build complete: manual.sif
```

Building A SIF Manually – Tutorial item 4

create sandbox

```
[cloud@trunk ~]$ apptainer build --sandbox manual/ docker://docker.io/library/alpine:3.17
INFO:    Starting build...
Getting image source signatures
Copying blob 63b65145d645 skipped: already exists
Copying config 6a2bcc1c7b done
Writing manifest to image destination
Storing signatures
2023/02/18 11:46:27  info unpack layer: sha256:63b65145d645c1250c391b2d16ebe53b3747c295ca8ba2fcb6b0cf064a4dc21c
WARNING: The sandbox contain files/dirs that cannot be removed with 'rm'.
WARNING: Use 'chmod -R u+rwX' to set permissions that allow removal.
WARNING: Use the '--fix-perms' option to 'apptainer build' to modify permissions at build time.
INFO:    Creating sandbox directory...
INFO:    Build complete: manual/
```

start a shell inside and edit

```
[cloud@trunk ~]$ apptainer shell --writable --fakeroot manual/
WARNING: Skipping mount /etc/localtime [binds]: /etc/localtime doesn't exist in container
Apptainer> cd /
Apptainer> echo "Hello from inside" > inner_message.txt
Apptainer> echo "#!/bin/sh" > launcher.sh
Apptainer> echo "cat /outer_message.txt" >> launcher.sh
Apptainer> echo "cat /inner_message.txt" >> launcher.sh
Apptainer> chmod +x launcher.sh
Apptainer> exit
```

copy file into

```
[cloud@trunk ~]$ echo "Hello from outside" > outer_message.txt
[cloud@trunk ~]$ cp outer_message.txt manual/
```

convert to SIF

```
[cloud@trunk ~]$ apptainer build manual.sif manual/
INFO:    Starting build...
INFO:    Creating SIF file...
INFO:    Build complete: manual.sif
```

exec

```
[cloud@trunk ~]$ apptainer exec manual.sif /launcher.sh
Hello from outside
Hello from inside
```

Building A SIF Manually – Problems

- Same problems as for Docker/OCI container images
- But does let external tools operate on the contents of the sandbox
- Can’t setup environment, runscript, labels, and apps without another doing a multi-stage build

Using A Definition File – Tutorial item 5

def_build.def

```
1  Bootstrap: docker
2  From: docker.io/library/alpine:3.17
3
4  %files
5      outer_message.txt /
6
7  %post
8      echo "Message from inside." > /inner_message.txt
9      echo "#!/bin/sh" > /launcher.sh
10     echo "cat /outer_message.txt" >> /launcher.sh
11     echo "cat /inner_message.txt" >> /launcher.sh
12     chmod +x /launcher.sh
13
14 %runscript
15     exec /launcher.sh
```

Apptainer, *Definition Files*, 2023
https://apptainer.org/docs/user/main/definition_files.html

Using A Definition File – Tutorial item 5

def_build.def

```
1  Bootstrap: docker
2  From: docker.io/library/alpine:3.17
3
4  %files
5      outer_message.txt /
6
7  %post
8      echo "Message from inside." > /inner_message.txt
9      echo "#!/bin/sh" > /launcher.sh
10     echo "cat /outer_message.txt" >> /launcher.sh
11     echo "cat /inner_message.txt" >> /launcher.sh
12     chmod +x /launcher.sh
13
14 %runscript
15     exec /launcher.sh
```

build

```
[cloud@trunk ~]$ echo "Hello from outside" > outer_message.txt
[cloud@trunk ~]$ apptainer build def_build.sif def_build.def
INFO:    Starting build...
Getting image source signatures
Copying blob 63b65145d645 skipped: already exists
Copying config 6a2bcc1c7b done
Writing manifest to image destination
Storing signatures
2023/02/18 12:05:43  info unpack layer: sha256:63b65145d645c1250c391b2d16ebe53b3747c295ca8ba2fcb6b0cf064a4dc21c
INFO:    Copying outer_message.txt to /
INFO:    Running post scriptlet
+ echo 'Message from inside.'
+ echo '#!/bin/sh'
+ echo 'cat /outer_message.txt'
+ echo 'cat /inner_message.txt'
+ chmod +x /launcher.sh
INFO:    Adding runscript
INFO:    Creating SIF file...
INFO:    Build complete: def_build.sif
```

Apptainer, *Definition Files*, 2023
https://apptainer.org/docs/user/main/definition_files.html

Using A Definition File – Tutorial item 5

def_build.def

```
1  Bootstrap: docker
2  From: docker.io/library/alpine:3.17
3
4  %files
5      outer_message.txt /
6
7  %post
8      echo "Message from inside." > /inner_message.txt
9      echo "#!/bin/sh" > /launcher.sh
10     echo "cat /outer_message.txt" >> /launcher.sh
11     echo "cat /inner_message.txt" >> /launcher.sh
12     chmod +x /launcher.sh
13
14 %runscript
15     exec /launcher.sh
```

Apptainer, *Definition Files*, 2023
https://apptainer.org/docs/user/main/definition_files.html

build

```
[cloud@trunk ~]$ echo "Hello from outside" > outer_message.txt
[cloud@trunk ~]$ apptainer build def_build.sif def_build.def
INFO:    Starting build...
Getting image source signatures
Copying blob 63b65145d645 skipped: already exists
Copying config 6a2bcc1c7b done
Writing manifest to image destination
Storing signatures
2023/02/18 12:05:43  info unpack layer: sha256:63b65145d645c1250c391b2d16ebe53b3747c295ca8ba2fcb6b0cf064a4dc21c
INFO:    Copying outer_message.txt to /
INFO:    Running post scriptlet
+ echo 'Message from inside.'
+ echo '#!/bin/sh'
+ echo 'cat /outer_message.txt'
+ echo 'cat /inner_message.txt'
+ chmod +x /launcher.sh
INFO:    Adding runscript
INFO:    Creating SIF file...
INFO:    Build complete: def_build.sif
```

run

```
[cloud@trunk ~]$ ./def_build.sif
Hello from outside
Message from inside.
```

Bootstrap

- Determine the base
- Many different sources
  - docker – Docker/OCI images from a Docker registry
  - localimage – another Singularity/Apptainer container image on disk
  - oci-archive – OCI tar image file
  - docker-archive – Docker tar image file
  - shub – Singularity/Apptainer image from a Singularity Hub registry
- Each one uses different options to select which base
  - Many use a `From` option
  - Always be explicit with registry, namespace, and tag/digest
  - Can include a fingerprint for many, which will only allow the build to proceed if the file/archive/image has that checksum

Exercise 2

Useful links during the exercise

- Apptainer, *Definition Files*, 2023 `https://apptainer.org/docs/user/main/definition_files.html`
- Apptainer user documentation – `https://apptainer.org/docs/user/main`
- Apptainer admin documentation – `https://apptainer.org/docs/admin/main/`

Tools

- Apptainer (`https://apptainer.org`)
- SingularityCE (`https://sylabs.io/singularity`)
- SingularityPRO (`https://sylabs.io/singularity-pro`)

Plenary Discussion & References

Apptainer. *Definition Files*. Online. Viewed 2023-02-17. Feb. 2023. URL: https://apptainer.org/docs/user/main/definition_files.html.

Docker. *Dockerfile reference*. Online. Viewed 2023-02-17. Feb. 2023. URL: https://docs.docker.com/engine/reference/builder.