Decomposing Container Tools

About Swiss Army Knives and Containers

Valentin Rothberg
rothberg@redhat.co
m
@vlntnrthbrg

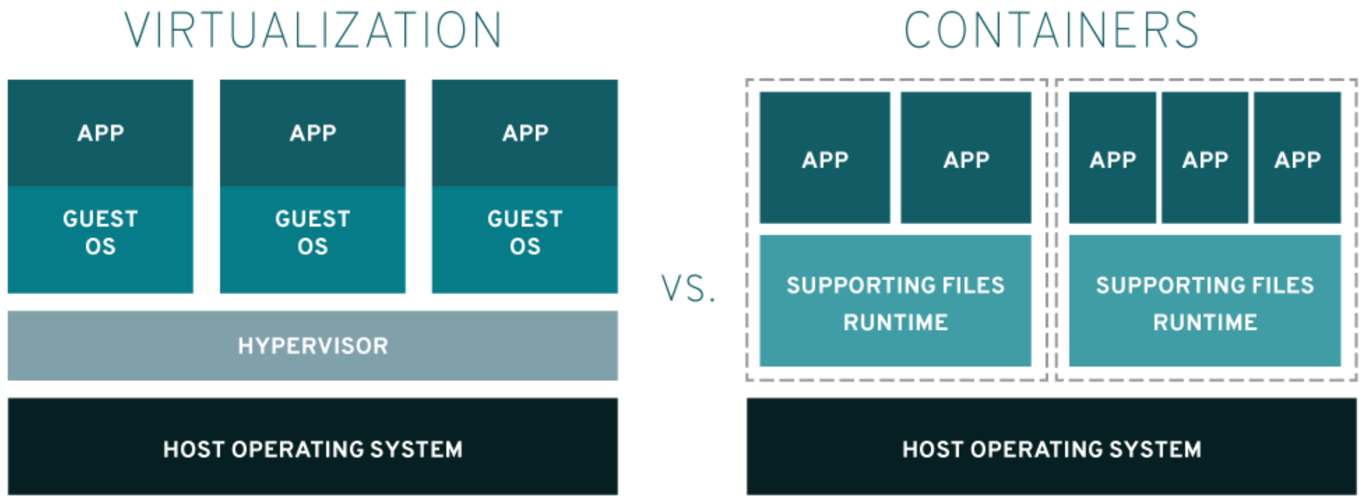
Who has been working with containers?

Please raise your hand.

Why are we using containers?

- An easy and cheap way to ship and deploy applications
- Scalability
- “Build once, run everywhere”
    - Portability
    - Reproducibility
    - Flexibility
- Great tooling and support
- Huge investments from the industry

Virtualization vs. Containers



The Life Cycle of a Container

Build

Distribute

Run

Orchestrate

Docker can do all of these things

Red Hat’s Containers Philosophy

- No *one-size-fits-all* solution
- Have use-case dedicated and specialized tools
  - *Open standards*
  - *Open development*
  - *Open source*
- Interoperability
- Reduced scope
- Allows for innovation

Building Containers

buildah

Buildah ABC

- Name originates from Dan Walsh’s Bostonian accent
- Buildah’s functionality goes beyond Dockerfiles
- Meant to be used as a low-level coreutils for building images
- Other tools should be able to embed buildah
- Developed at *github.com/containers/buildah*

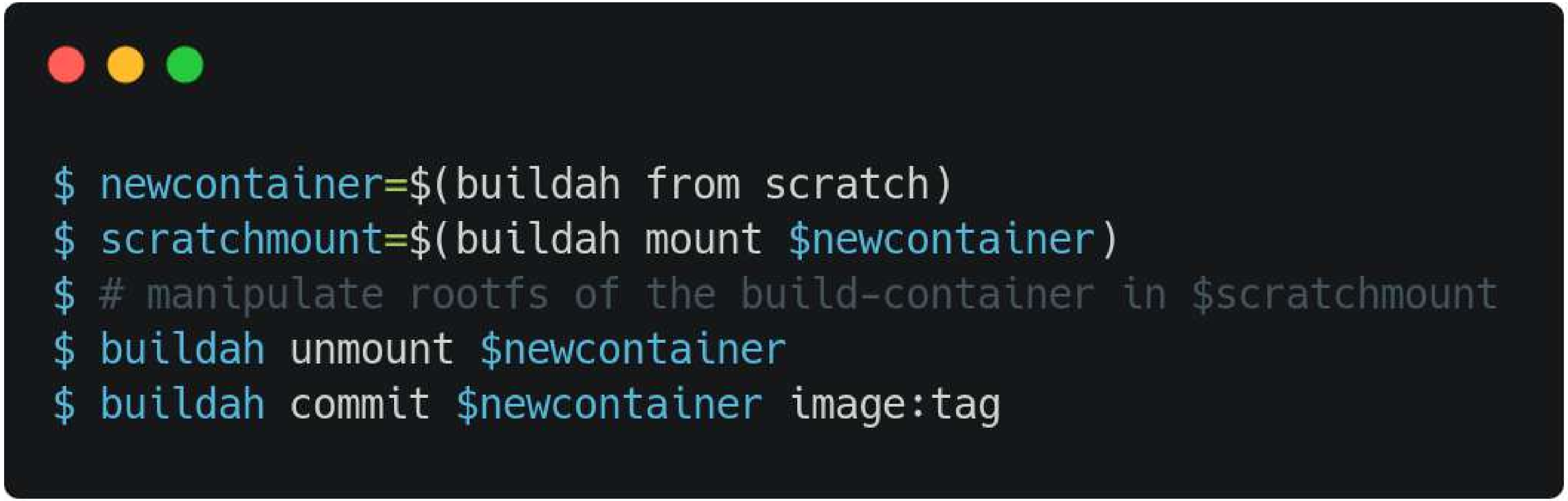
Buildah ABC

- Supports Dockerfiles
  - $ buildah **b**uild-**u**sing-**d**ockerfile -f Dockerfile .
  - Or shorter via $ buildah **bud** ...
- Can run rootless
- Daemon-less architecture
- Focus on OCI standards and open development
- Exposes a golang library
- Easy to integrate into K8s pipelines
  - Official images available at **quay.io/buildah/stable:latest**

Does *Buildah* have a scripting language?

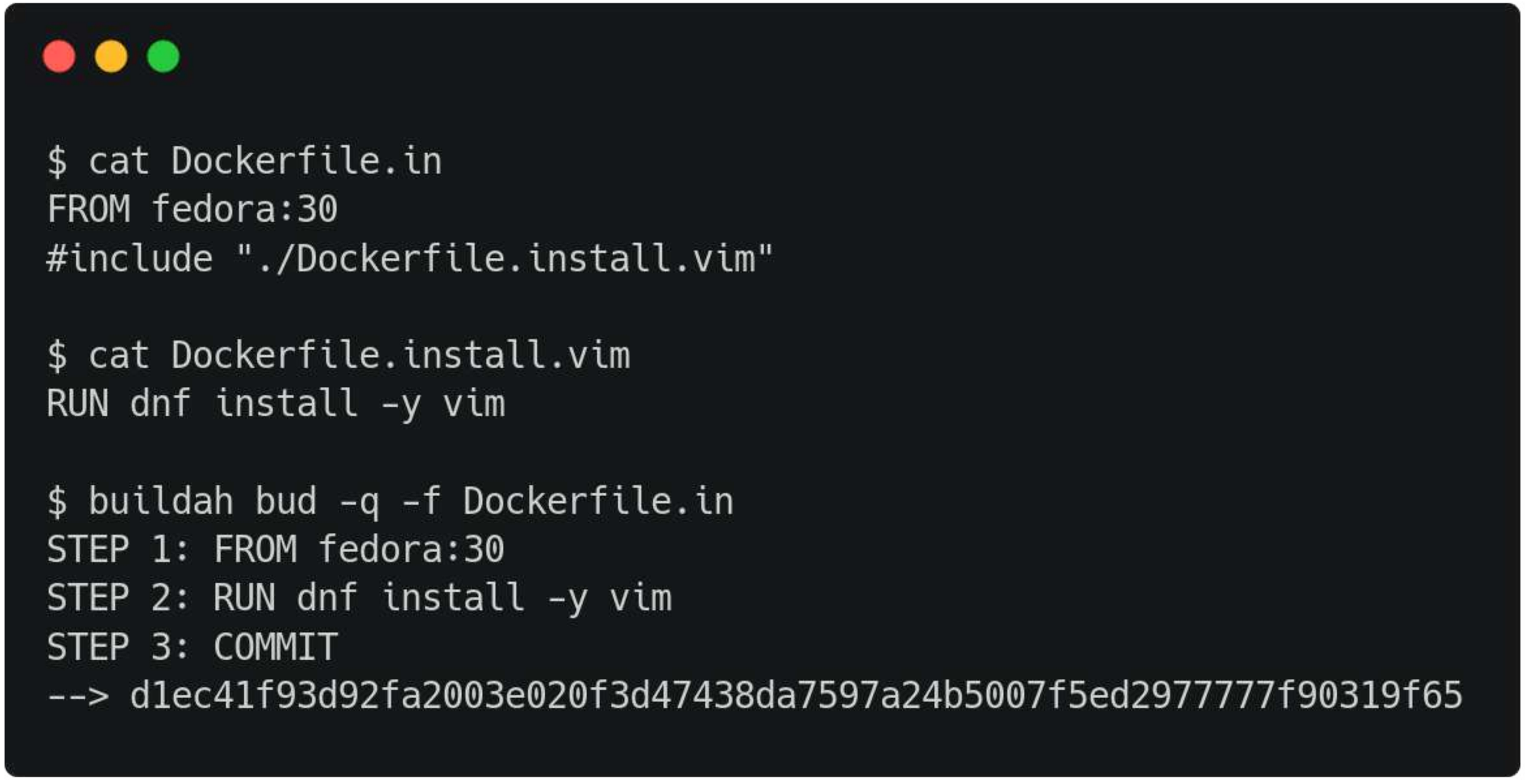
Perhaps *Buildahfile*?

BASH - Buildah’s scripting language



```
$ newcontainer=$(buildah from scratch)
$ scratchmount=$(buildah mount $newcontainer)
$ # manipulate rootfs of the build-container in $scratchmount
$ buildah unmount $newcontainer
$ buildah commit $newcontainer image:tag
```

Decomposing Dockerfiles



```
$ cat Dockerfile.in
FROM fedora:30
#include "./Dockerfile.install.vim"

$ cat Dockerfile.install.vim
RUN dnf install -y vim

$ buildah bud -q -f Dockerfile.in
STEP 1: FROM fedora:30
STEP 2: RUN dnf install -y vim
STEP 3: COMMIT
--> d1ec41f93d92fa2003e020f3d47438da7597a24b5007f5ed2977777f90319f65
```

Why a dedicated building tool?

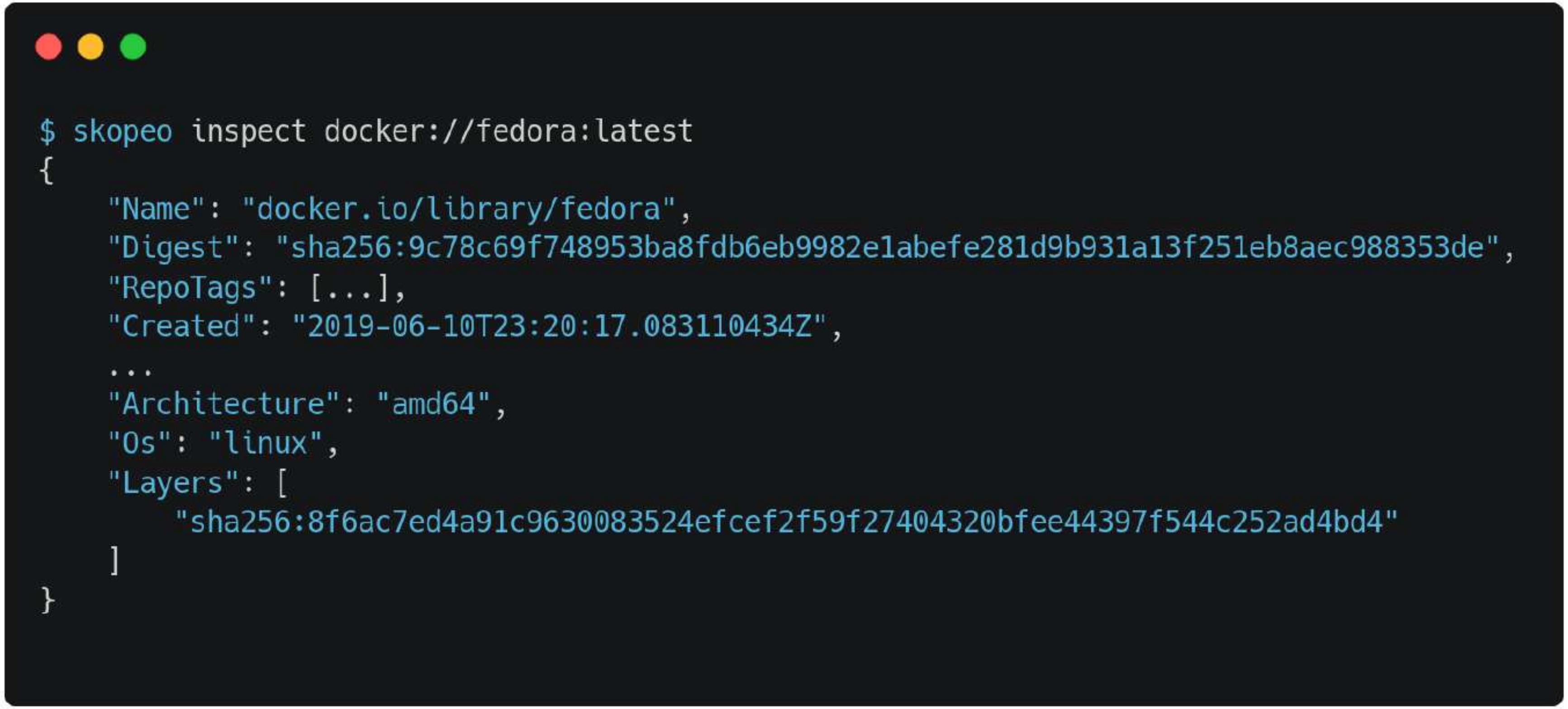
- Dedicated CLI without the fear of cluttering
- Use cases beyond Dockerfiles
- Cleaner and smaller code base due to limited focus
- Specialisation
  - A Buildah container is less restricted than a Podman container
- Independent release cycles
  - New features reach users faster
- Innovation
  - Not being blocked by non-building requirements

Distributing Containers

Skopeo ABC

- Used in many non-Docker pipelines to distribute images (e.g., Open Build Service)
- Developed at *github.com/containers/skopeo*
- Does not run as a daemon and does not require root privileges
- Can copy single images and lists of images (manifest lists)
- Supports different image formats
  - Docker v2s1 & v2s1
  - Open-Container Initiative (OCI)
- 23 MB binary size vs ~210 MB of Docker (Fedora 31)
  - A comparatively small “army knife”

Skopeo - born by the desire to inspect remote images



```
$ skopeo inspect docker://fedora:latest
{
    "Name": "docker.io/library/fedora",
    "Digest": "sha256:9c78c69f748953ba8fdb6eb9982e1abefe281d9b931a13f251eb8aec988353de",
    "RepoTags": [...],
    "Created": "2019-06-10T23:20:17.083110434Z",
    ...
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
        "sha256:8f6ac7ed4a91c9630083524efcef2f59f27404320bfee44397f544c252ad4bd4"
    ]
}
```

Skopeo - Supported Transports

- Containers-storage
  - Local container storage (e.g., overlay or btrfs)
- Directory
  - Non-standardized format to “explode” an image to a specified path
- Docker
  - Image on a registry (e.g., docker.io/library/fedora:latest)
  - Archive in the docker-save(1) format
  - From a local docker-daemon
- OCI
  - As specified by the OCI image spec
  - Can also be compressed as a tar(1) archive

Skopeo - Registries Configuration

- /etc/containers/registries.conf
- Unqualified search registries - *pull fedora*
- Namespaced registry settings
  - Insecure (without TLS verification)
  - Blocked (any attempt to contact the server is blocked)
  - Mirrors (will be contact prior to the registry)
- Shared by all tools in this talk

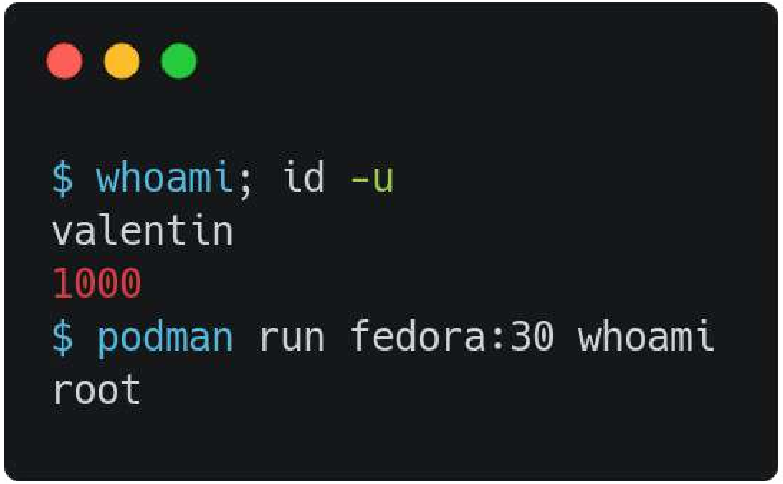
Running Containers

What is Podman?

- Container engine for managing containers and pods
- *Pod man*ager
- CLI is based on Docker
  - De facto standard CLI for managing containers
  - Allows for an easier transition of users *and* scripts
- Fastest migration
  - *alias docker=podman*

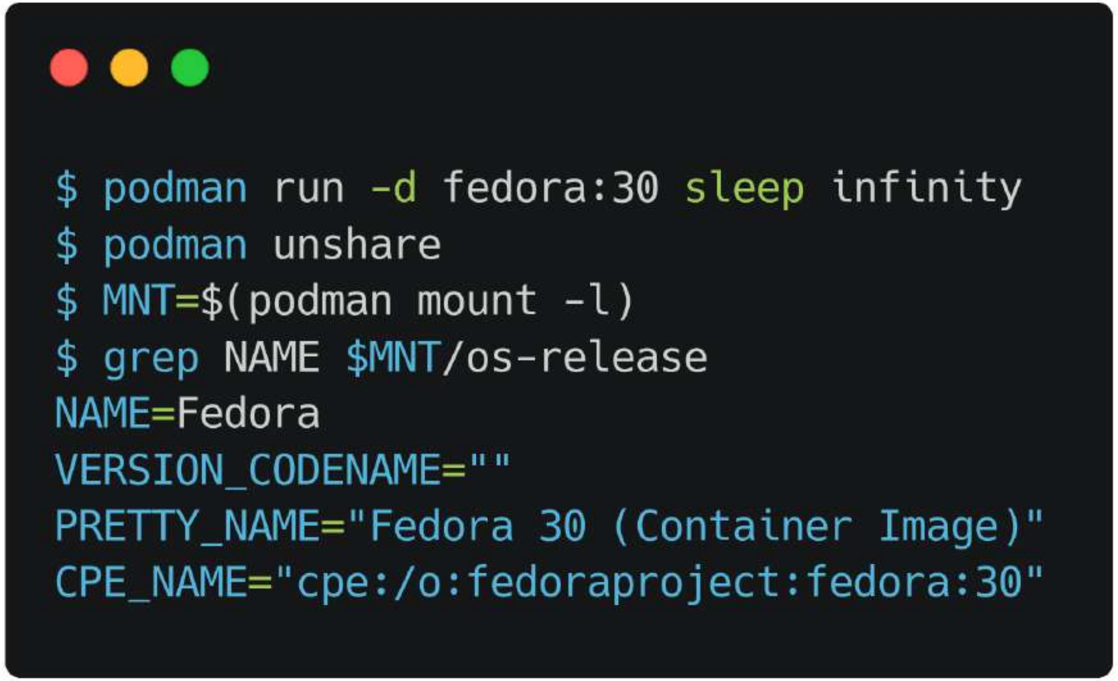
Enough said, let’s have a look!

No Daemon. *No* Root.



```
$ whoami; id -u
valentin
1000
$ podman run fedora:30 whoami
root
```

Podman Mount/Unmount



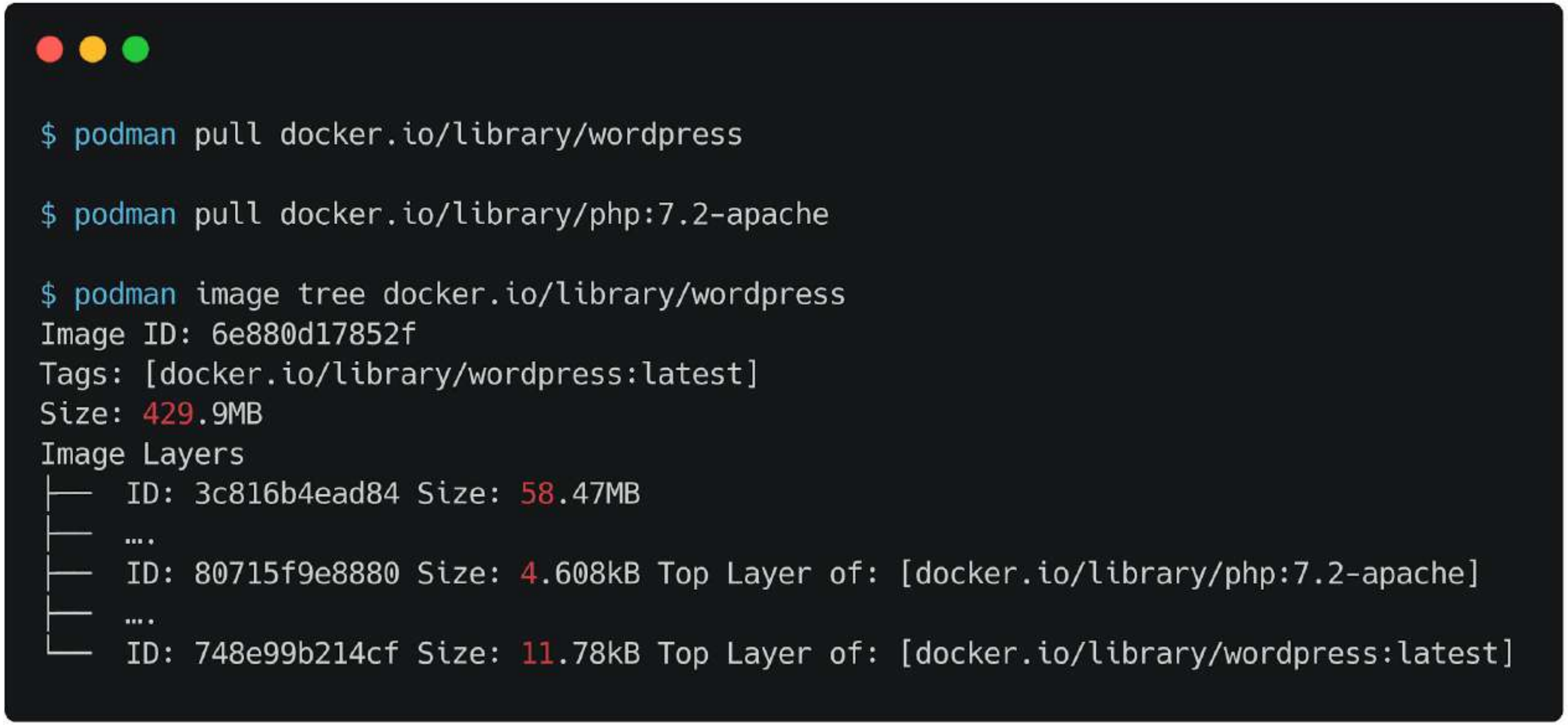
```
$ podman run -d fedora:30 sleep infinity
$ podman unshare
$ MNT=$(podman mount -l)
$ grep NAME $MNT/os-release
NAME=Fedora
VERSION_CODENAME=""
PRETTY_NAME="Fedora 30 (Container Image)"
CPE_NAME="cpe:/o:fedoraproject:fedora:30"
```

Managing Container Images Is Tough

```
$ podman images -q | wc -l
182
```

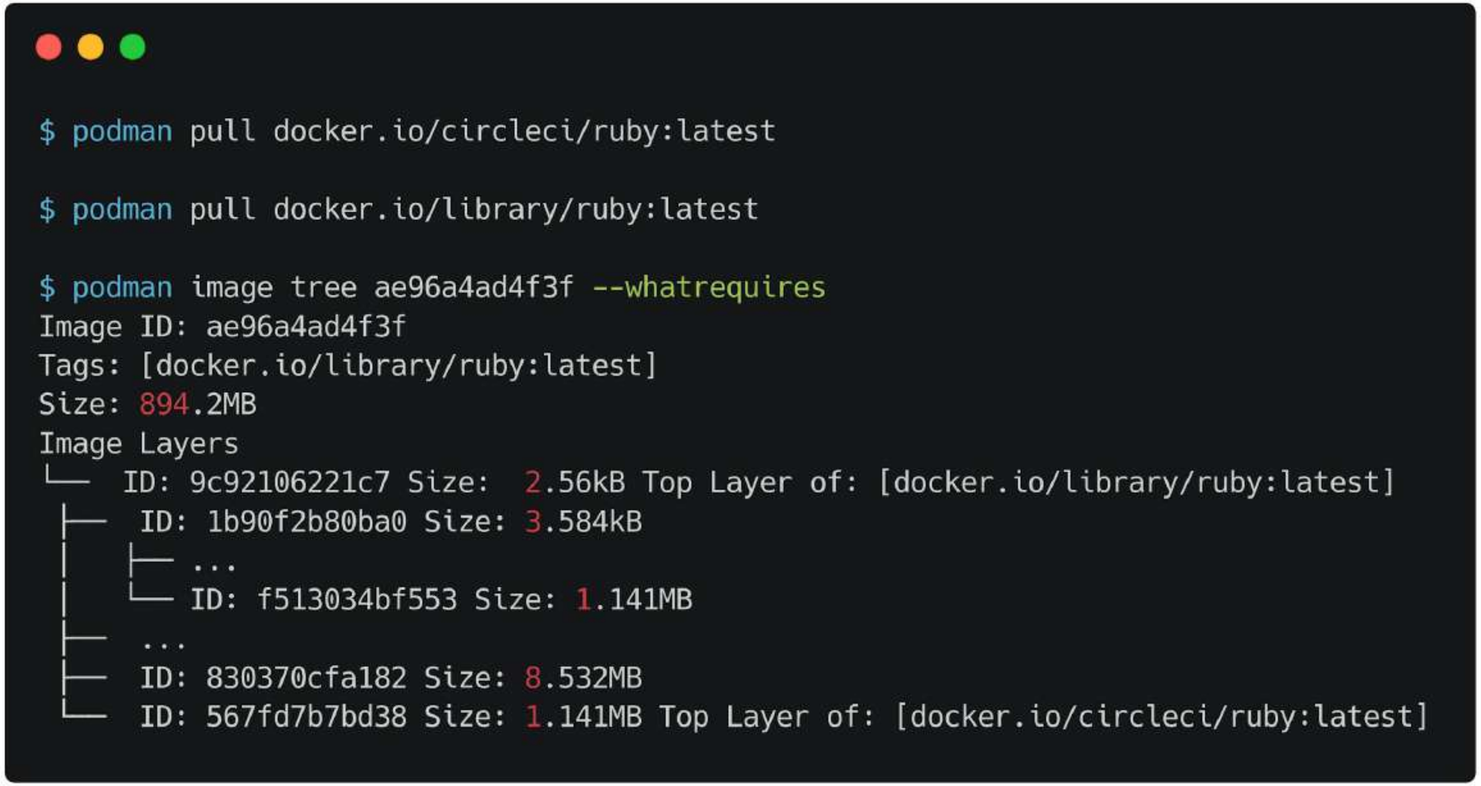
- The local image storage can quickly become a mess
  - Development, testing, and everything’s containerized
- Which images does an image use?
  - Do I really need them? Can I rebase my application on something less complex?
- Which image is required by other images?

Podman Image Tree - Which layers does X use?



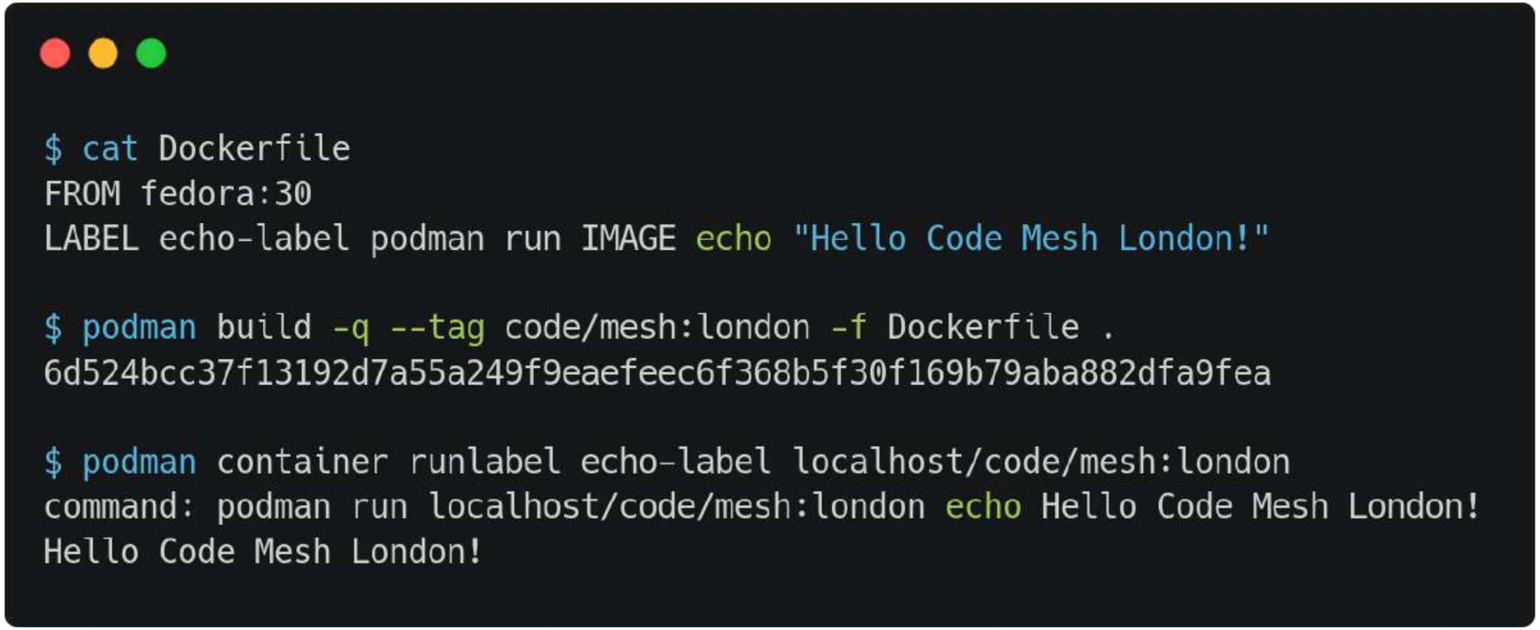
```
$ podman pull docker.io/library/wordpress

$ podman pull docker.io/library/php:7.2-apache

$ podman image tree docker.io/library/wordpress
Image ID: 6e880d17852f
Tags: [docker.io/library/wordpress:latest]
Size: 429.9MB
Image Layers
├──  ID: 3c816b4ead84 Size: 58.47MB
├──  ….
├──  ID: 80715f9e8880 Size: 4.608kB Top Layer of: [docker.io/library/php:7.2-apache]
├──  ….
└──  ID: 748e99b214cf Size: 11.78kB Top Layer of: [docker.io/library/wordpress:latest]
```

Podman Image Tree - Which layers require X?



```
$ podman pull docker.io/circleci/ruby:latest

$ podman pull docker.io/library/ruby:latest

$ podman image tree ae96a4ad4f3f --whatrequires
Image ID: ae96a4ad4f3f
Tags: [docker.io/library/ruby:latest]
Size: 894.2MB
Image Layers
└── ID: 9c92106221c7 Size:  2.56kB Top Layer of: [docker.io/library/ruby:latest]
 ├── ID: 1b90f2b80ba0 Size: 3.584kB
 │   ├── ...
 │   └── ID: f513034bf553 Size: 1.141MB
 ├──  ...
 ├── ID: 830370cfa182 Size: 8.532MB
 └── ID: 567fd7b7bd38 Size: 1.141MB Top Layer of: [docker.io/circleci/ruby:latest]
```

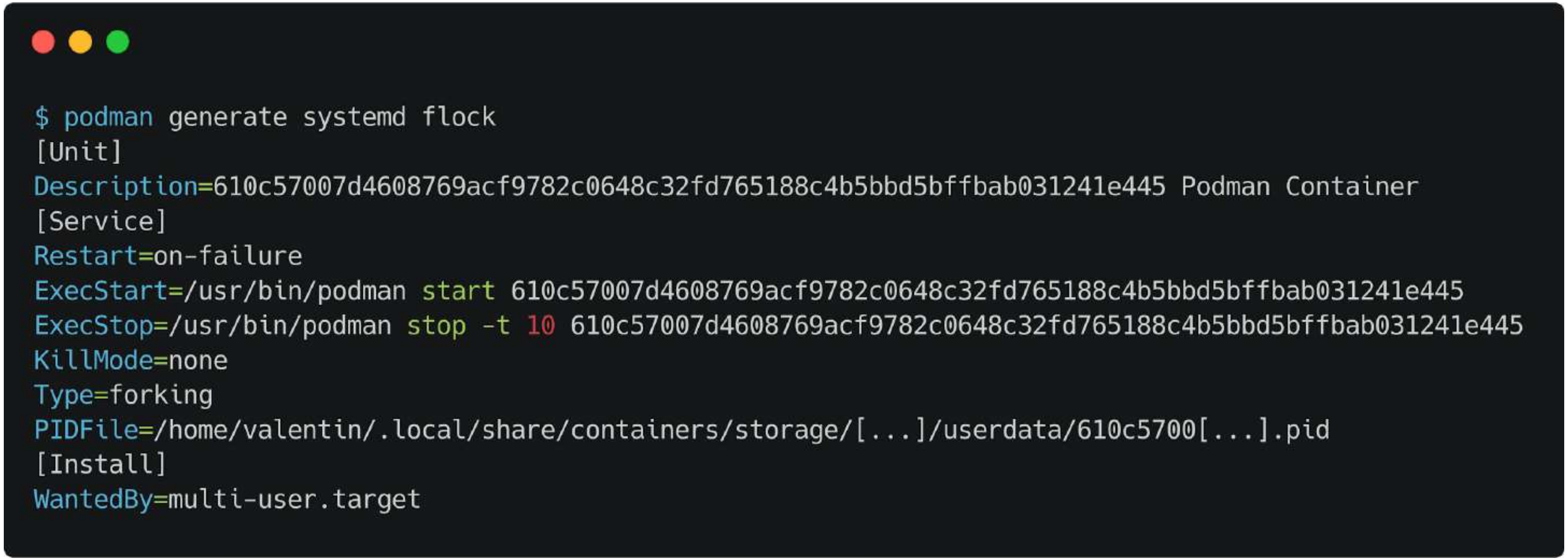
Container Runlabel - Let’s get straight to it!



```
$ cat Dockerfile
FROM fedora:30
LABEL echo-label podman run IMAGE echo "Hello Code Mesh London!"

$ podman build -q --tag code/mesh:london -f Dockerfile .
6d524bcc37f13192d7a55a249f9eaefeec6f368b5f30f169b79aba882dfa9fea

$ podman container runlabel echo-label localhost/code/mesh:london
command: podman run localhost/code/mesh:london echo Hello Code Mesh London!
Hello Code Mesh London!
```

“Runlabel can execute any command on the host”

Podman Generate Systemd



```
$ podman generate systemd flock
[Unit]
Description=610c57007d4608769acf9782c0648c32fd765188c4b5bbd5bffbab031241e445 Podman Container
[Service]
Restart=on-failure
ExecStart=/usr/bin/podman start 610c57007d4608769acf9782c0648c32fd765188c4b5bbd5bffbab031241e445
ExecStop=/usr/bin/podman stop -t 10 610c57007d4608769acf9782c0648c32fd765188c4b5bbd5bffbab031241e445
KillMode=none
Type=forking
PIDFile=/home/valentin/.local/share/containers/storage/[...]/userdata/610c5700[...].pid
[Install]
WantedBy=multi-user.target
```

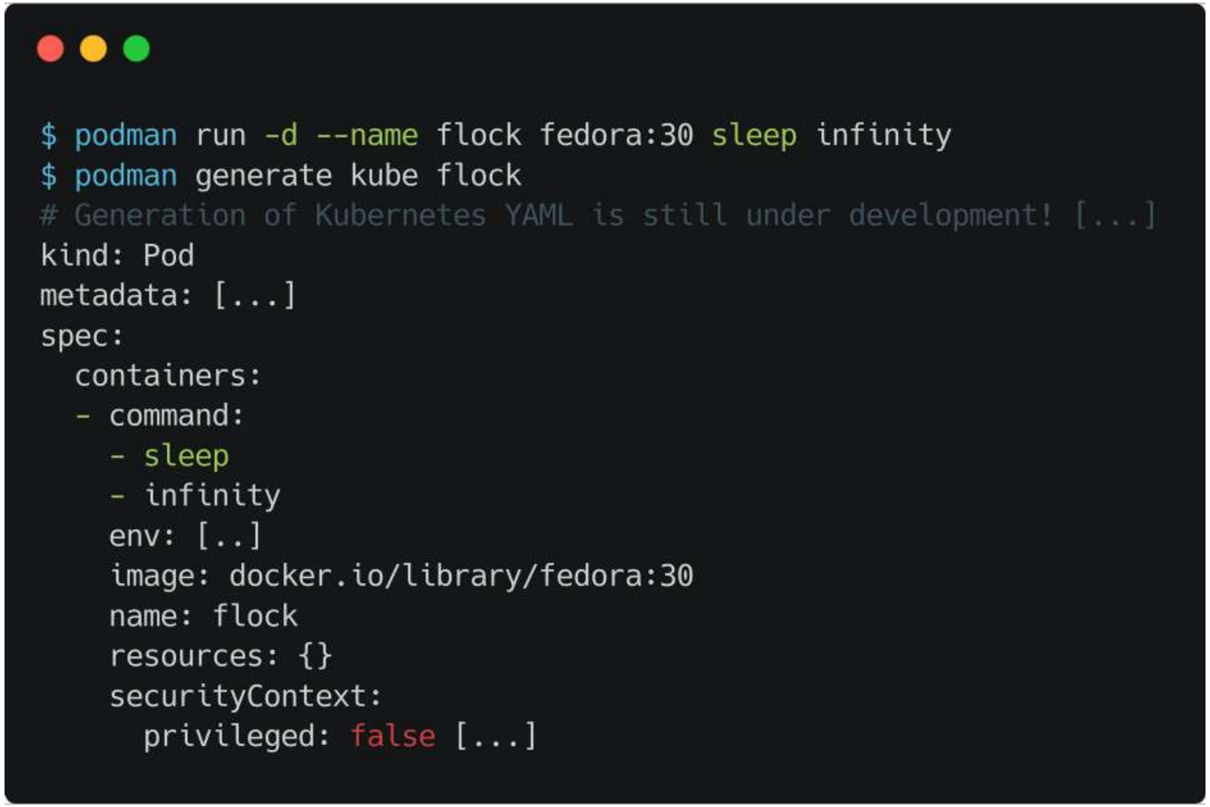
What is .../userdata/**conmon.pid**?

- **Conmon** is the container monitor and sits between Podman and the runtime
- Provides a socket for attaching to the container
- Streams to a log file or the systemd journal
- Keeps file descriptors and ports open
- Records container’s exit time and code
- It’s actually a **daemon** to prevent Podman from being one
  - But a really small one (i.e., 76K)

Systemd in Containers

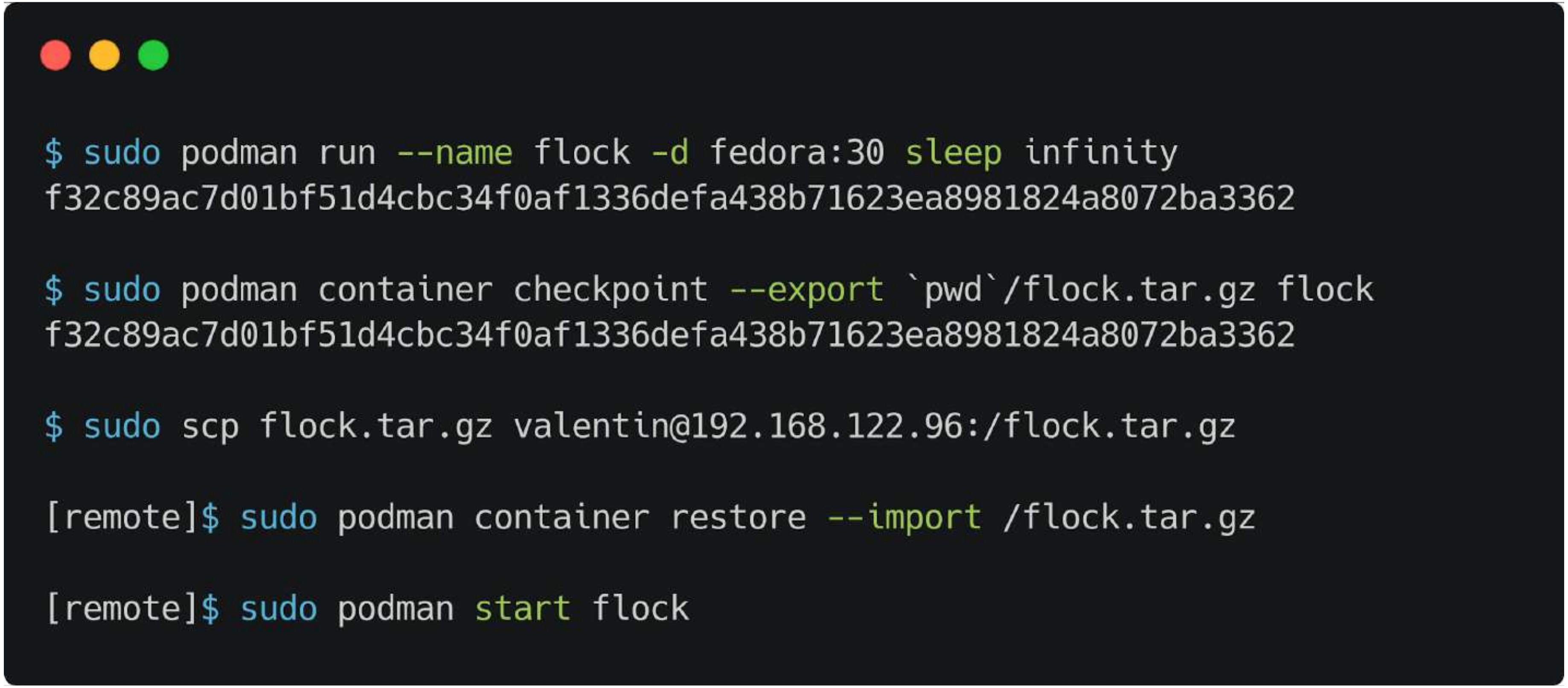
- Many packages need it but it wasn’t supported for a long while
  - Workarounds and hand-written init scripts
- Systemd OCI hook for Docker
- Podman has built-in support if --systemd or command[0] == “systemd” or “init”
  - Mounts /run, /run/lock, /tmp, /var/log/journal as tmpfs
  - Bind mounts /sys/fs/cgroup
- No workarounds needed anymore, just install the packages

Podman generate kube



```
$ podman run -d --name flock fedora:30 sleep infinity
$ podman generate kube flock
# Generation of Kubernetes YAML is still under development! [...]
kind: Pod
metadata: [...]
spec:
  containers:
  - command:
    - sleep
    - infinity
    env: [..]
    image: docker.io/library/fedora:30
    name: flock
    resources: {}
    securityContext:
      privileged: false [...]
```

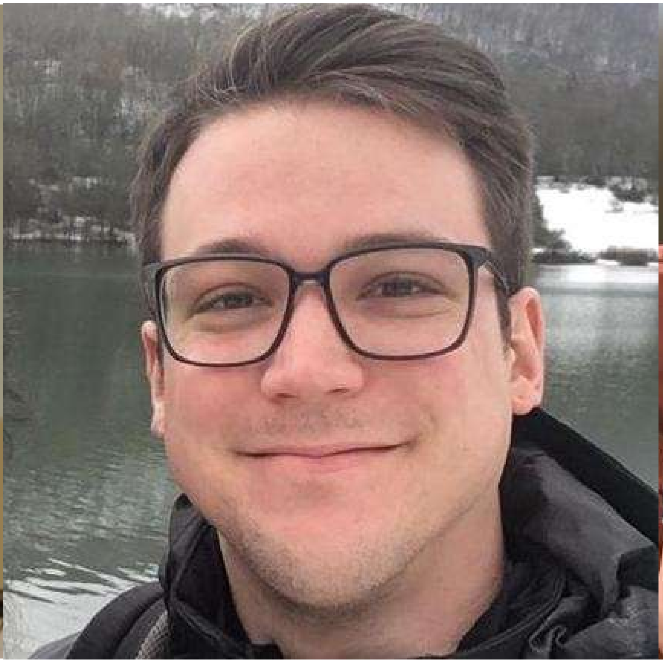
Podman Checkpoint & Restore



```
$ sudo podman run --name flock -d fedora:30 sleep infinity
f32c89ac7d01bf51d4cbc34f0af1336defa438b71623ea8981824a8072ba3362

$ sudo podman container checkpoint --export `pwd`/flock.tar.gz flock
f32c89ac7d01bf51d4cbc34f0af1336defa438b71623ea8981824a8072ba3362

$ sudo scp flock.tar.gz valentin@192.168.122.96:/flock.tar.gz

[remote]$ sudo podman container restore --import /flock.tar.gz

[remote]$ sudo podman start flock
```

Google Summer of Code Project 2019

Divyansh Kamboj



Valentin Rothberg

Dan Walsh

Generate Seccomp Profiles with Podman and eBPF

- Seccomp is a Linux security mechanism to filter syscalls
- Containers commonly use a default seccomp profile
  - Allows more than 300 of the 435 syscalls on Linux 5.3 x86_64
  - Average container uses 40 to 70 syscalls (Aqua Sec)
  - *~80% of attack surface reduction*
- We use eBPF to trace executed syscalls to generate custom profiles for each workload
- Please visit *podman.io* for more information on the GSoC project

Orchestrating Containers

CRI-O

- OCI-based Kubernetes Runtime
  - *The only use case is Kubernetes*: nothing more nothing less
- CNCF project since April 2019
- Supports all OCI compatible container images
  - Including all older Docker formats
- Supports any container registry
- Supports all OCI container runtimes
- 100+ contributors, 90+ releases, 1500+ per PR
- Collaboration across the industry (Red Hat, SUSE, Intel, IBM, lyft)

The Life Cycle of a Container

Build
Buildah

Distribute
Skopeo

Run
Podman

Orchestrate
CRI-O
&
Kubernetes

- All tools share the same code
  - github.com/containers/image
  - github.com/containers/storage
- Packaged for major Linux distributions
  - RHEL, Fedora, CentOS
  - openSUSE, SLES
  - Ubuntu, Arch Linux, Manjaro, Debian (soon)
- More information at
  - CRI-O.IO
  - BUILDAH.IO
  - PODMAN.IO

podman

cri-o

buildah