On-demand rootless containers with systemd and podman

Rich Lucente

who am i?

Husband, Dad, Technical professional

30+ years in the industry as developer, software engineer, manager, consultant, sales engineer

Like to tinker with technology

Also spend free time with family, trips to Disney world (probably too much), tabletop games, Legos (so many), fantasy and sci/fi books, shows, and movies

also, didn’t bring the ansible

serverless

key features

Containers made easy

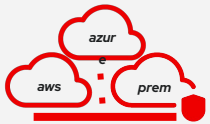
Simplified developer experience to deploy applications/code on serverless containers abstracting infrastructure & focusing on what matters.

Immutable revisions

Deploy new features: performing canary, A/B or blue-green testing with gradual traffic rollout with no sweat and following best practices.

Automatic scaling

No need to configure number of replicas, or idling. Scale to zero when not in use, auto scale to thousands during peak, with built-in reliability and fault-tolerance.



Ready for the Hybrid Cloud

Truly portable serverless that is on-premises or on any public cloud. Leverage data locality and SaaS when needed.

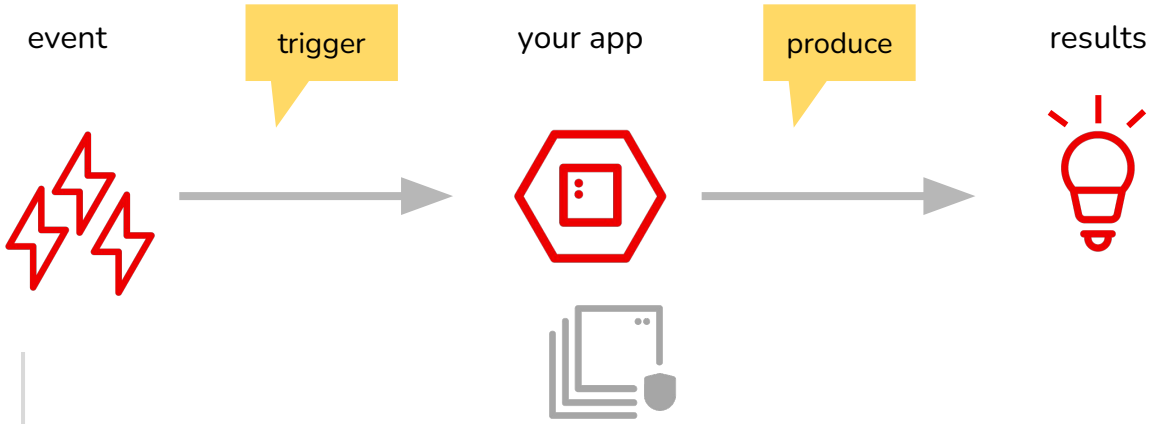
Any programming language

Use any programming language or runtime of choice. From Java, Python, Go and JavaScript to Quarkus, SpringBoot or Node.js.

Event Driven Architectures

Build loosely coupled & distributed apps connecting with a variety of built-in or third-party event sources or connectors.

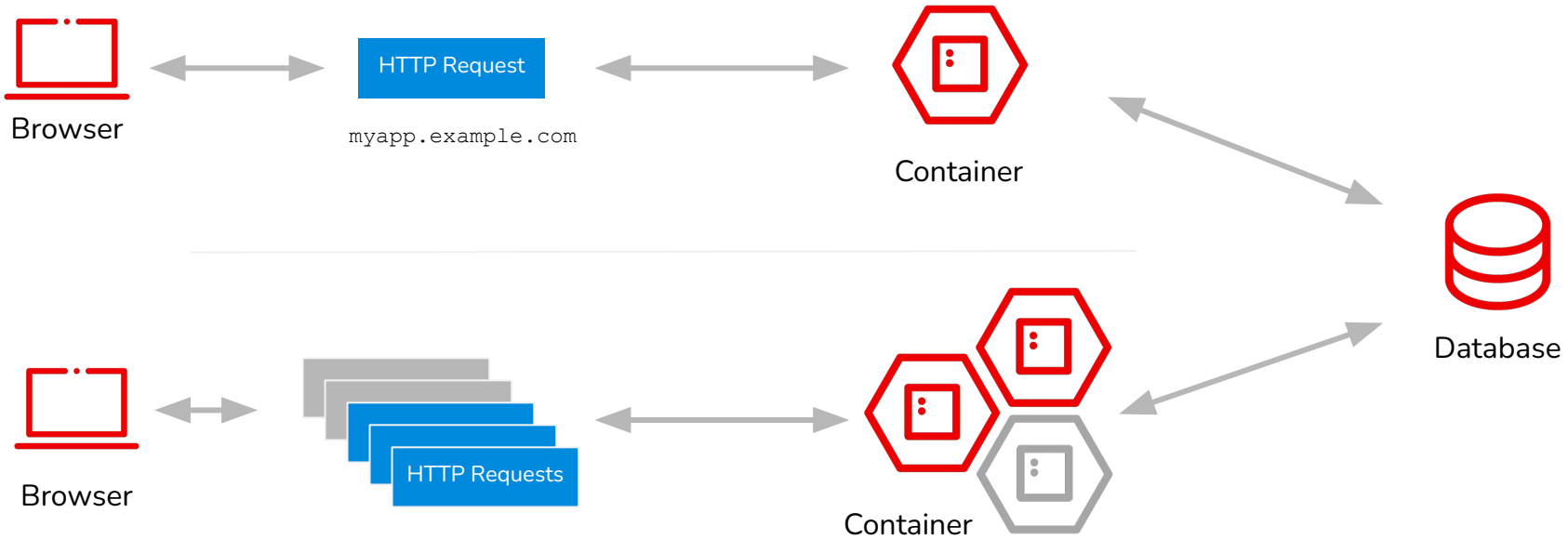
the "serverless" pattern



- HTTP Requests
- Kafka Messages
- Image Uploaded
- New Order
- Login from user

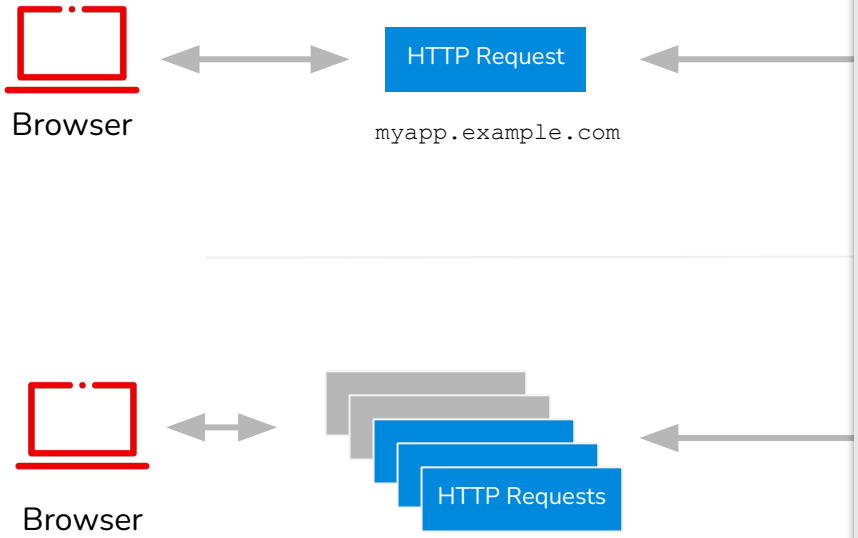
the "serverless" pattern

A serverless web application



the "serverless" pattern

A serverless web application



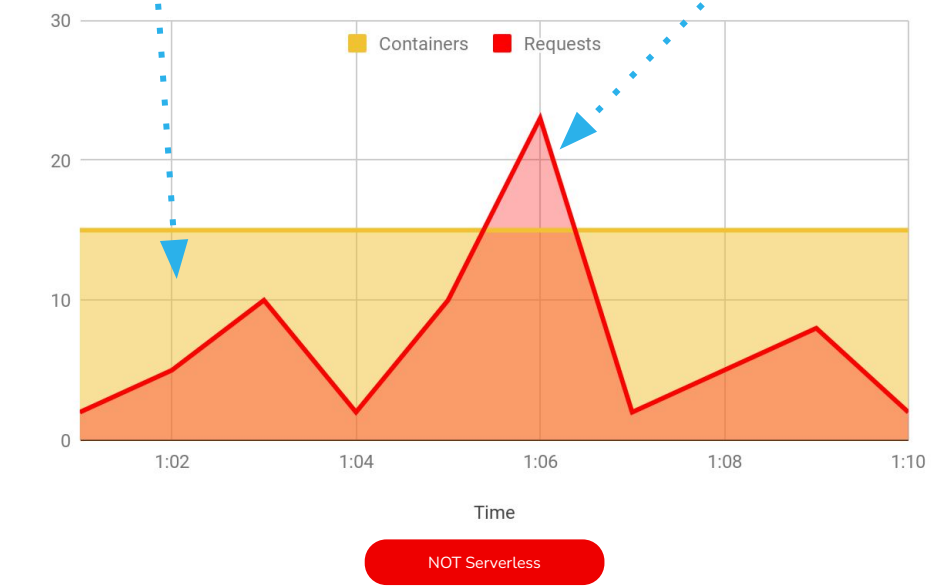
Benefits of this model:

- No need to setup auto-scaling and load balancers
  - Scale down and save resources when needed.
  - Scale up to meet the demand.
- No tickets to configure SSL for applications
- Enable Event Driven Architectures (EDA) patterns
- Enable teams to associate cost with IT
- Modernize existing applications to run as serverless containers

serverless operational benefits

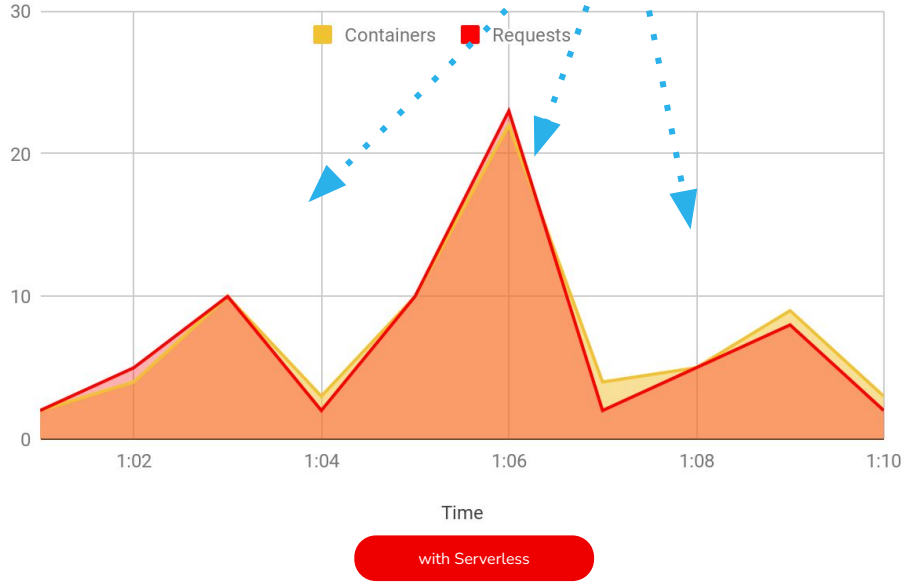
Over provisioning
Time in capacity planning
IT cost of idle resources

Under provisioning
Lost business revenue
Poor quality of service



NOT Serverless

More applications
Direct line between IT
costs & business revenue



with Serverless

systemd in a nutshell

unit files

```
[Socket]
ListenStream=172.16.244.128:8080

[Install]
WantedBy=sockets.target
```

```
[Unit]
Requires=container-httpd.service
After=container-httpd.service
Requires=container-httpd-proxy.socket
After=container-httpd-proxy.socket

[Service]
ExecStart=/usr/lib/systemd/systemd-socket-proxyd --exit-idle-time=10 127.0.0.1:8080
```

unit types

- service
- socket
- target
- timer
- slice
- scope

- path
- device
- mount
- automount
- swap

```
man systemd.unit && man systemd.<unit type>
```

unit file locations

maintainer

```
/usr/lib/systemd/system
```

administrator

```
/etc/systemd/system
```

non-persistent, runtime

```
/run/systemd/system
```

identify and compare overriding unit files

```
systemd-delta
```

Note

Unit files in `/etc` take precedence over `/run`, and `/run` over `/usr`. The full systemd unit path list and how to override it can be found in `man 5 systemd.unit`

Very basic usage

`systemctl` - Primary command for interacting with systemd e.g. start, stop, reload, restart, enable, disable, status

- **`systemctl enable --now httpd`**
- **`systemctl set-property --runtime CPUShares=2048 httpd`**

`journalctl` - View and filter the system journal

- **`journalctl -fu chronyd`**

socket activation

The operation is pretty straightforward

- Systemd listens for an incoming request
- When request received, file descriptor is passed to service with matching name
- Service handles the request and also listens for additional connections
- When service terminates, systemd once again listens for incoming requests

Set up is to define a **socket** unit and **service** unit with the same name

Only the socket is started at boot when enabled

unprivileged systemd units

systemd – user

```
$ systemctl --user status --no-pager -l
● localhost.localdomain
    State: running
     Jobs: 0 queued
   Failed: 0 units
    Since: Wed 2022-03-23 12:20:10 EDT; 2h 1min ago
   CGroup: /user.slice/user-1000.slice/user@1000.service
           ├─app.slice
           │ └─dbus-broker.service
           │   ├─1798 /usr/bin/dbus-broker-launch --scope user
           │   └─1799 dbus-broker --log 4 --controller 9 --machine-id
215e9bf02e8e4a70928fa87f25dcc2f7 --max-bytes 100000000000000 --max-fds
25000000000000 --max-matches 5000000000
           ├─init.scope
           │ ├─1616 /usr/lib/systemd/systemd --user
           │ └─1618 "(sd-pam)"
           └─user.slice
             └─podman-pause-637686879523354398.scope
               └─1760 podman
```

systemd – user

User units

```
~/.config/systemd/user
```

Maintainer user units

```
/usr/lib/systemd/user &
~/.local/share/systemd/user
```

Global user units (all users)

```
/etc/systemd/user
```

Note:

`.bashrc` and `.bash_profile` are not sourced by systemd

```
~/.config/environment.d
```

```
systemctl --user import-environment VAR
```

```
systemctl --user show-environment
```

systemd – user

Interact with the systemd user instance

- `systemctl --user`

e.g. start, stop, restart, enable, disable, status

- `systemctl --user enable --now foo.service`

Filter the journal by user unit(s)

- `journalctl --user-unit=foo.service`

Enable/disable systemd user outside of sessions (start on boot)

- `loginctl enable-linger $USER`
- `loginctl disable-linger $USER`
- `Loginctl show-user $USER`

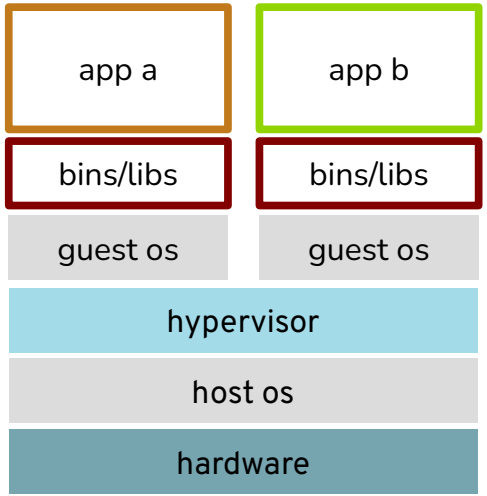
“Shame back” view of user’s disgusting use of system resources

- `loginctl user-status`

switching gears to podman ...

what’s a container?

helps to start with virtual machines



Virtual machine definition includes ...

- Virtual hardware boundaries
- Hypervisor
- One instance per VM
- IaaS paradigm

And often adds ...

- Can run any OS
- Full software stack
- Very isolated guests

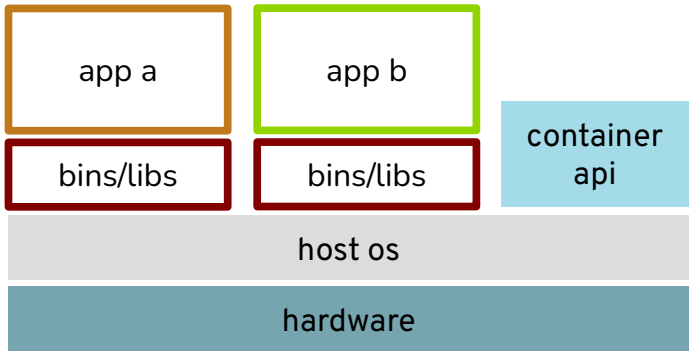
containers “virtualize” the host os

Containers are characterized by ...

- Horizontal segmentation
- Container API
- Single OS instance
- Multi-tenancy
- Bare metal, virtual, cloud

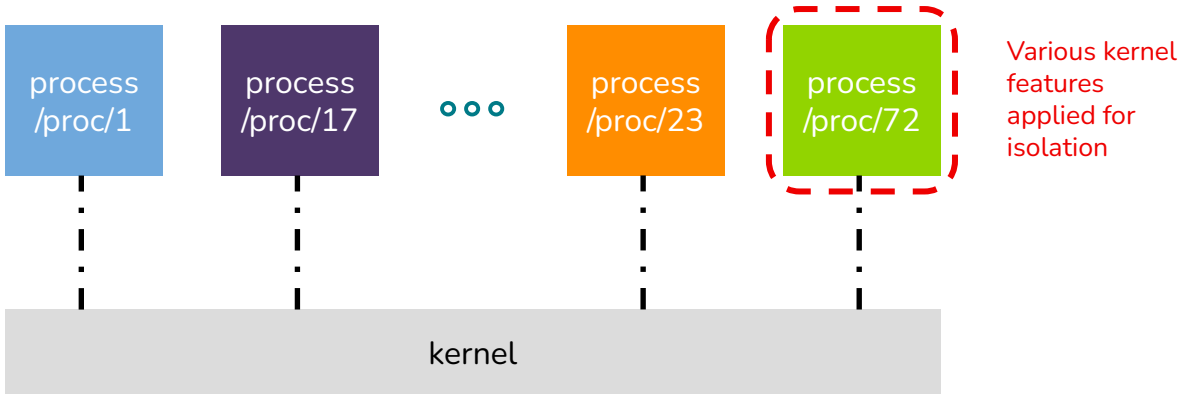
And often ...

- Near zero boot time
- Cannot run any OS



containers are a process

- Containers are just like any other process
- Leverage specific kernel features for isolation (e.g. namespaces, cgroups, SELinux, seccomp, capabilities, etc)



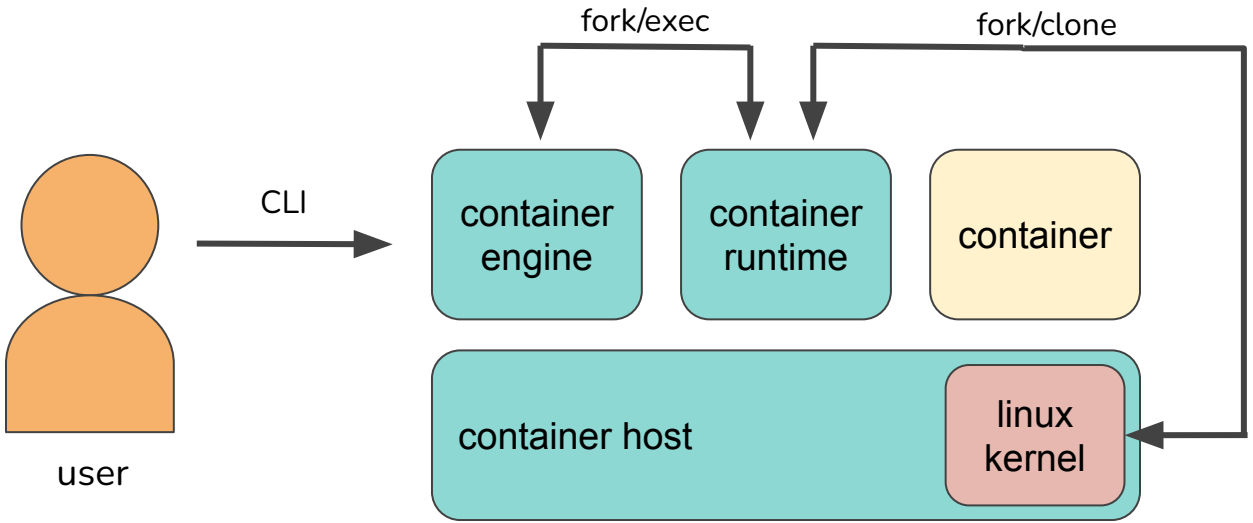
containers are standard

- 2015: OCI Announced
- https://www.opencontainers.org/
- Runtime specification
- Image specification

podman

how are containers instantiated?

```
podman run -it ubi8/ubi:latest /bin/bash
```



what is podman?

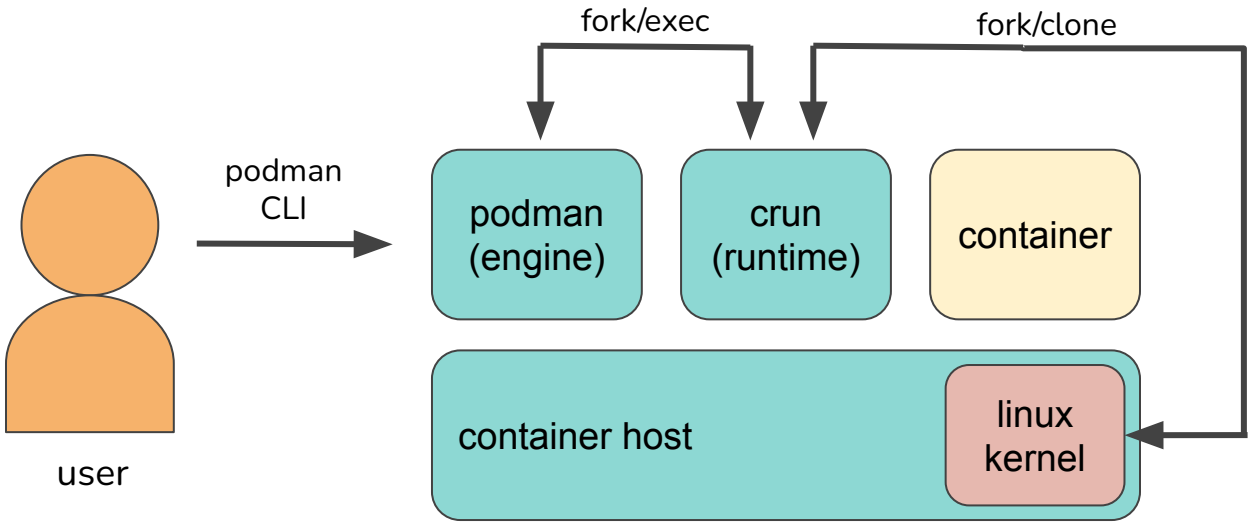
Podman is a daemonless container engine for developing, managing, and running OCI Containers on your Linux System.

Containers can either be run as root or in rootless mode.

Simply put: `alias docker=podman`

how are containers instantiated?

```
podman run -it ubi8/ubi:latest /bin/bash
```



useful podman commands

Manage OCI container images

- podman images
- podman pull
- podman push
- podman search
- podman tag
- podman rmi

Manage services

- podman auto-update
- podman generate systemd

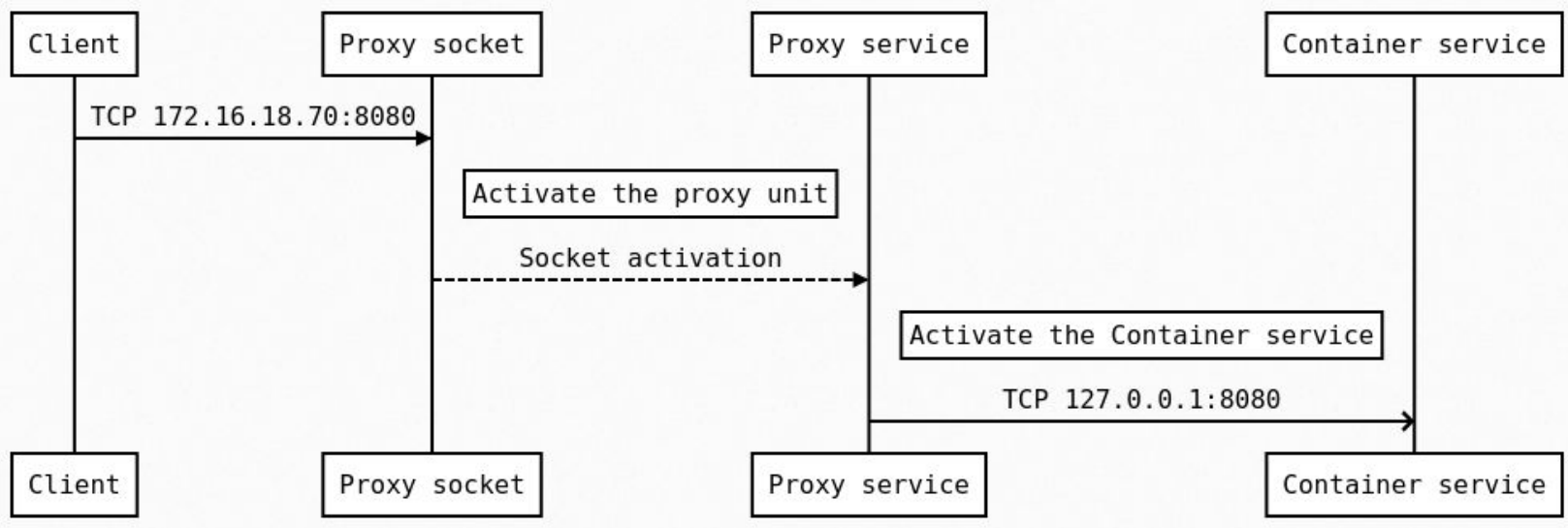
Manage OCI containers

- podman create
- podman ps
- podman rm
- podman run
- podman start
- podman stop

demo time!

what it is

Huge shout out to Pietro Bertera and his awesome blog ...
[Painless services: Implementing serverless with rootless podman and systemd]



questions? (and thanks!)