



# Containers without Docker



# Frederick Lawler

Systems Engineer @ Cloudflare

- `security_create_user_ns()`
- Primarily works in security
- Dabbles in containerization

**I want to open as many IPv4 unicast  
TCP connections as fast as possible  
on my machine without breaking my  
machine**

[connect\(\) - why are you so slow?](#)

# Requirements

- Client must connect to a server
- System must be configured to have 56,512 ephemeral ports
- Connections must balance over one or more IP addresses
- Latencies must be reported

# Requirements

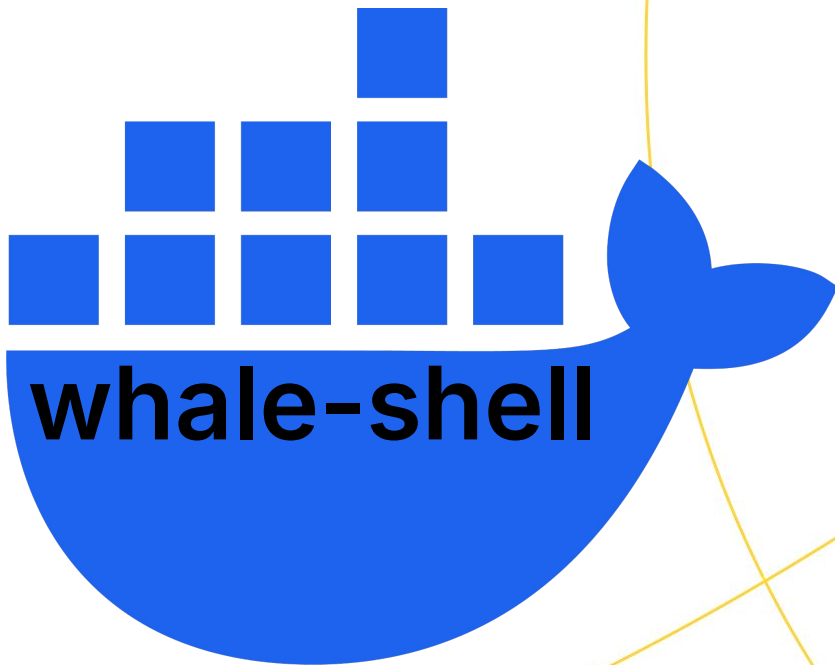
- Client must connect to a server
  - Socket must be created and bound to a port
- System must be configured to have 56,512 ephemeral ports
  - Configured through system sysctl
  - Ulimits increased
- Connections must balance over one or more IP addresses
  - A network interface must be created
  - IP addresses must be assigned to this interface
- Latencies must be reported
  - External libraries likely needed to make pretty pictures

**Whoops!**

**Need root permissions**

Container time:

# Docker in a whale-shell



# Composition

## Images

- System independent\*
- Creates a filesystem
- Sharable
- Composable
- Buildtime

## Containers

- System dependent\*
- Uses images
- Non-sharable
- Horizontal scalable
- Runtime

\* Architecture + Linux + correct kernel features enabled





**Write environment once,  
ship anywhere**

# Cowsay: Image

```
FROM debian:bookworm-slim
```

```
RUN apt-get update && \  
    apt-get install -y cowsay
```

```
ENTRYPOINT ["/usr/games/cowsay", "hello container!"]
```

```
$ docker build -t cowsay:latest -f Dockerfile.cowsay .
```

# Cowsay: Container run

```
$ cat /etc/os-release | grep PRETTY_NAME
```

```
PRETTY_NAME="Ubuntu 24.04.2 LTS"
```

```
$ docker run --rm cowsay:latest
```

```
-----  
< hello container! >
```

```
-----
```

```
  \      ^__^
   \    (oo)\_______
      (__)\\       )\/\
           ||----w |
           ||     ||
```

# Let's try my problem: Image

```
FROM debian:bookworm-slim
```

```
RUN apt-get update && \  
    apt-get install -y procps \  
        python3
```

```
WORKDIR work
```

```
COPY ./generate-data.py .
```

```
COPY ./runner.sh .
```

```
ENTRYPOINT ["../runner.sh"]
```

```
$ docker build -t connections:latest -f Dockerfile.connections .
```

# Let's try my problem: Image

```
$ cat runner.sh

#!/usr/bin/env -S bash -e

set -x

id

ulimit -Sn $(ulimit -Hn)

/usr/sbin/sysctl -w net.ipv4.ip_local_port_range="9024 65535"

./generate-data.py --random_offset_window 500 9024,65535 results
```

[Source files](#)

# Let's try my problem: Container run

```
$ docker run --rm connections:latest  
  
+ id  
  
uid=0(root) gid=0(root) groups=0(root)  
  
++ ulimit -Hn  
  
+ ulimit -Sn 1048576  
  
+ /usr/sbin/sysctl -w 'net.ipv4.ip_local_port_range=9024 65535'  
  
sysctl: permission denied on key "net.ipv4.ip_local_port_range"  
  
exit status 1
```

# Container capabilities

```
c$ cat /proc/self/status | grep CapEff
```

```
CapEff: 00000000a80425fb
```

```
$ capsh --decode=00000000a80425fb
```

```
0x00000000a80425fb=cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap
```

Missing capabilities: cap\_net\_admin, cap\_sys\_admin

# This can be fixed, but there's more!

```
$ capsh --decode=00000000a80425fb
```

```
0x00000000a80425fb=cap_chown,cap_dac_override,  
,cap_fowner,cap_fsetid,cap_kill,cap_setgid,ca  
p_setuid,cap_setpcap,cap_net_bind_service,cap  
_net_raw,cap_sys_chroot,cap_mknod,cap_audit_w  
rite,cap_setfcap
```

Missing capabilities: cap\_net\_admin,  
cap\_sys\_admin

? + id  
uid=0(root) gid=0(root) groups=0(root)

- Docker runs containers through the containerd daemon with runc runtime
- runc applies seccomp policies by default
- runc applies default capabilities
- These must be overwritten to solve my problem
- containerd is a root process!



~~To hell with Docker!~~

**Let's make our own  
container!**

# Namespaces

A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes.

**One use of namespaces is to implement containers.**

[man 7 namespaces](#)

# Types of namespaces

- **Mount**
- **Network**
- **Process ID (PID)**
- **UNIX Time-Sharing (UTS)**
- **User**
- **Control group (cgroup)**
- **Inter-process Communication (IPC)**
- **Time**

# How do we make them?

- clone
- setns
- **unshare**
- Other tools:
  - docker / podman / etc...
  - ip netns (for networking)
  - **systemd-nspawn**

```
$ echo $$
```

```
192566
```

```
$ unshare
```

```
c$ echo $$
```

```
193138
```

```
c$ echo $PPID
```

```
192566
```

# Let us dive in!

1. User
2. UNIX Time-Sharing (UTS)
3. Network
4. Process ID (PID)
5. Mount

# User

- Isolation of user capabilities
- Map users to other users
- Process specific
- Used to enable & fully utilize containerization

```
$ unshare --map-root
```

```
c$ id
```

```
uid=0(root) gid=0(root)  
groups=0(root),65534(nogroup)
```

```
c$ cat /proc/self/status | grep CapEff  
CapEff: 000001ffffffffffff
```

```
c$ cat /proc/self/user/uid_map  
  
0          1000          1
```

## User: A controversy

- Containers with `--map-root` are effectively root in permissions
- Bugs existing in Linux kernel can be surfaced & exploited through user namespaces
- Harmless for prototyping, but needs protection for production environments

## User: A filesystem example

- Root user doesn't mean root filesystem permissions
- Filesystem permissions are still mapped to the user making the namespace

```
$ unshare --map-root
```

```
c$ touch /file
```

```
touch: cannot touch '/file': Permission denied
```

```
c$ touch file
```

```
c$ stat -c '%u %g' file
```

```
0 0
```

```
c$ exit
```

```
$ stat -c '%u %g' file
```

```
1000 1000
```



## User: A Docker example

- Docker doesn't use them by default\*
- Can't use `unshare()` or `clone(CLONE_NEWUSER)` within container due to seccomp policy
- Not be confused with **`docker run --user`** (defaults to root)

\* `docker run --userns=...`  
(docker daemon needs [userns-remap](#) configured)

```
FROM debian:bookworm-slim
ENTRYPOINT ["/usr/bin/unshare"]
```

```
$ docker run --rm unshare:latest
unshare: unshare failed: Operation not
permitted
```

```
$ sudo strace -f -p $(pgrep containerd) -e
trace=clone,unshare,setns,seccomp,execve
...
[pid 1880403] unshare(0)                                = -1
EPERM (Operation not permitted)
```

## User: A Docker example

- Run as actual root
- Still not using CLONE\_NEWUSER
- Seccomp policies disabled
- Access to system devices
- Dangerous for Docker!

```
$ sudo docker run --privileged --rm  
unshare:latest
```

```
$ echo $?  
0
```

# User a summary

- Each process has a user namespace by default
- Needed for system-security isolation
- Filesystems respect original user
- Not used by Docker by default to make unpriv containers

[man 7 user\\_namespaces](#)

# UNIX Time-sharing (UTS)

- Isolate a hostname for a container
- By default assumes callers hostname
- Mostly useful for networking
- No seriously...

```
$ unshare --map-root --uts
```

```
c$ hostname  
CMGLRV3
```

```
c$ hostname container
```

```
c$ hostname  
container
```

# UNIX Time-sharing (UTS): A Docker example

```
$ docker run --hostname=hostname.com --rm cowsay:latest
```

```
-----  
< hostname.com >
```

```
-----  
 \   ^__^  
  \ (oo)\_____  
    (__)\\       )\\/\  
        ||----w |  
        ||     ||
```

[man 7 uts\\_namespaces](man 7 uts_namespaces)

# Network

- Isolates networks
- Powerful tool to configure interfaces
  - Firewall rules
  - Packet filtering/routing/etc...
  - IP addresses
  - Basically any network protocol stack manipulation

```
$ docker run --network=none
```

```
$ docker run --network=bridge
```

```
$ docker run --network=container:name|id
```

```
$ docker run --network=host
```

```
$ docker run  
--network=network-name|network-id
```

# Network: A Linux example

```
$ unshare --map-root --net
c$ ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

c$ ping 127.0.0.1
ping: connect: Network is unreachable

c$ ip link set lo up
c$ ip ping 1.1.1.1
ping: connect: Network is unreachable

c$ ip ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.079 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.102 ms
^C
--- 127.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1007ms
rtt min/avg/max/mdev = 0.079/0.090/0.102/0.011 ms
```

[man 7 network\\_namespaces](#)

# Process ID (PID)

- Isolate processes from host & other containers
- Your program becomes PID 1
- Killing container kills all sub-processes
- Docker does this for you

```
$ unshare --map-root --pid --fork
```

```
c$ ps ax
```

PID	TTY	STAT	TIME	COMMAND
1	pts/6	S	0:00	-zsh
122	pts/6	R+	0:00	ps ax

```
c$ echo $$
```

```
1
```

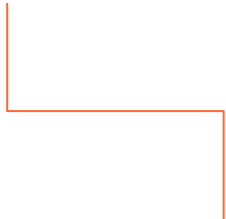
[man 7 pid\\_namespaces](#)



# Mount

- Isolate a filesystem from other processes
- Typically used for root-filesystems for the container
- Can be used to move files from container to host or other containers
- Default mount namespace is current host filesystem (containers should isolate)

FROM debian:bookworm-slim



root filesystem  
/  
/bin  
/etc  
/home  
...

# Mount: A Docker example

```
FROM debian:bookworm-slim
```

```
RUN apt-get update && \  
    apt-get install -y cowsay
```

```
VOLUME /output
```

```
ENTRYPOINT ["/bin/bash", "-c", "/bin/mount | /bin/grep output; /usr/games/cowsay \"hello container\" >  
/output/cowsay"]
```

```
$ docker build -t cowsay:latest -f Dockerfile.cowsay .
```

# Mount: A Docker example

```
$ mkdir cowsay-mount
```

```
$ docker volume create --name cowsay-volume --opt type=none --opt  
device=$(pwd)/cowsay-mount --opt o=bind
```

```
$ docker run --mount source=cowsay-volume,target=/output --rm cowsay:latest  
/dev/mapper/vgubuntu-root on /output type ext4 (rw,relatime,errors=remount-ro)
```

# Mount: A Docker example

```
$ cat cowsay-mount/cowsay
```

```
-----  
< hello container >  
-----
```

```
  \   ^__^  
  \  (oo)\_____   
      (__) \       )\/\  
          ||----w |  
          ||     ||
```

```
$ stat -c '%u %g' cowsay-mount/cowsay  
0 0
```

Created & owned by root!



# Mount: A Linux example: mount isolation

```
$ unshare --map-root --mount
```

```
c$ mount -t tmpfs none ./mount-point
```

```
c$ mount | grep mount-point
```

```
none on
```

```
/home/fred/Presentations/containers/mount-point
```

```
type tmpfs
```

```
(rw,relatime,uid=1000,gid=1000,inode64)
```

```
c$ touch mount-point/hello-container
```

```
c$ ls mount-point/hello-container
```

```
mount-point/hello-container
```

```
$ cat ./mount-point/hello-container
```

```
cat: ./mount-point/hello-container: No such  
file or directory
```

```
$ ls mount-point
```

```
$ mount | grep mount-point
```



Host has no knowledge of the mount!

# Mount: A Linux example: implement Docker

## More on this later!



(it's complicated—but not too complicated)

# Mount a summary

- By default uses host filesystem
- Important for host & container filesystem isolation
- Powerful tool to share mounts between other containers & host
- Whole presentations are dedicated to this subject

[man 7 mount\\_namespaces](#)

# Control Group (cgroup)

Used for managing resources such as memory, CPU, I/O on a per-process hierarchy level.

Main interaction with these are through the `/sys/fs/cgroup` filesystem.  
Per process cgroup information is found at `/proc/<pid>/cgroup`

[man 7 cgroups](#)



# Control Group (cgroup)

Used for managing resources such as memory, CPU, I/O on a per-process hierarchy level.

Main interaction with these are through the `/sys/fs/cgroup` filesystem.  
Per process cgroup information is found at `/proc/<pid>/cgroup`

**Out of scope of talk :(**

[man 7 cgroups](#)

# Tying it all together

```
$ cat runner.sh

#!/usr/bin/env -S bash -e

unshare --map-root --net --pid --fork --kill-child -- /bin/bash --init-file <(
cat <<EOF
    set -x
    ip link set dev lo up
    ulimit -Sn $(ulimit -Hn)
    /usr/sbin/sysctl -w net.ipv4.ip_local_port_range="9024 65535"
    ./generate-data.py --random_offset_window 500 9024,65535 results
    exit 0
EOF
)
```

```
$ ./run-generate-data.sh  
  
+ ip link set dev lo up  
+ ulimit -Sn 1048576  
+ sysctl -w 'net.ipv4.ip_local_port_range=9024 65535'  
net.ipv4.ip_local_port_range = 9024 65535  
+ ./generate-data.py --random_offset_window 500 9024,65535 results  
Namespace(random_offset_window=500, port_range='9024,65535', data_filename='results')  
expected connections: 56512 min port: 9024 max port: 65535  
filename: results.csv  
server: started on 127.9.9.2:999  
server: closing 55499 connections  
server: server closed  
+ exit 0
```

# No mount?

```
unshare --map-root --net --pid --fork  
--kill-child -- /bin/bash --init-file
```



--mount?

- Recall by default we're using host filesystem
- Prototyping & testing doesn't really need to be completely isolated (unless you want)

# Tying it all together

## 10x: Patching & debugging systemd

# Systemd quick refresher

- Many Linux systems use systemd as init process
- systemd is a tool of many tools that spawns processes and initializes a Linux operating system
- Init process is process ID (PID) 1 on machines
- If PID 1 breaks, your machine is on life support (or worse) until reboot

[man 1 systemd](#)

# Create root filesystem

- Tools for creating filesystems:
  - Arch: [mkarchroot](#)
  - Debian/Ubuntu: [debootstrap](#)
  - Many distros: [mkosi](#)
  - Download raw images & extract from distros
  - Docker/OCI images
- Not dissimilar to creating Docker images

```
$ mkosi -d ubuntu -p make -p libseccomp-dev  
-p udev -p util-linux -p python3 -p  
python3-systemd -p tmux -p systemd-coredump  
-t directory -o basefs --force build
```



FROM ubuntu:noble



# Install systemd

- Install systemd into a different folder to overlay on top of root filesystem
- Setup a configuration file to overlay on top of root filesystem
- Systemd & conf file can be installed into the ./basefs root filesystem, but maybe we want to reuse this filesystem for other projects?

```
$ mkdir ./systemd
```

```
$ DESTDIR=$(pwd)/systemd meson install -C  
/path/to/systemd/code/build/
```

```
$ mkdir -p ./config
```

```
$ cat
```

```
config/etc/systemd/system/console-getty.service.d/  
autologin.conf
```

```
[Service]
```

```
ExecStart=
```

```
ExecStart=-/sbin/agetty -o '-p -f -- \\u' --noclear  
--keep-baud --autologin root - 115200,38400,9600 $TERM
```

# Create an experiment

- Just a trivial Makefile to run within the container

```
$ tree /home/fred/Work/systemd-bug-hashmap
/home/fred/systemd-bug-hashmap
├─ a.service
├─ b.service
├─ c.service
├─ d.service
└─ Makefile

$ cat /home/fred/Work/systemd-bug-hashmap/Makefile
.PHONY: run
run:
    systemctl enable ./a.service
    systemctl enable ./b.service
    systemctl enable ./c.service
    systemctl enable ./d.service
    systemctl start a.service
```

# Overlay systemd + config + experiment on top of filesystem & execute

```
$ sudo systemd-nspawn --directory ./basefs --boot  
--overlay=+/:/home/fred/Work/ez-systemd/config:/home/fred/Work/ez-systemd/systemd:/home/fred/Work/systemd-bug-ha  
shmap::/
```

Spawning container basefs on /home/fred/Work/ez-systemd/basefs.

Press Ctrl-] three times within 1s to kill container.

```
systemd 257.2 running in system mode (+PAM -AUDIT +SELINUX -APPARMOR +IMA +IPE +SMACK +SECCOMP -GCRYPT -GNUTLS  
+OPENSSL +ACL +BLKID -CURL +ELFUTILS -FIDO2 -IDN2 -IDN -IPTC -KMOD -LIBCRYPTSETUP -LIBCRYPTSETUP_PLUGINS  
-LIBFDISK +PCRE2 -PWQUALITY -P11KIT -QRENCODE -TPM2 +BZIP2 -LZ4 +XZ +ZLIB +ZSTD -BPF_FRAMEWORK -BTF -XKBCOMMON  
+UTMP +SYSVINIT -LIBARCHIVE)
```

Detected virtualization systemd-nspawn.

Detected architecture x86-64.

Detected first boot.

Welcome to Ubuntu 24.04.1 LTS!

...

# Overlay systemd + config + experiment on top of filesystem & execute

```
root@basefs:/# ls / | grep service
```

```
a.service  
b.service  
c.service  
d.service
```

```
root@basefs:/# make run
```

```
systemctl enable ./a.service
```

```
Created symlink '/etc/systemd/system/a.service' → '/a.service'.
```

```
Created symlink '/etc/systemd/system/multi-user.target.wants/a.service' → '/a.service'.
```

```
systemctl enable ./b.service
```

```
Created symlink '/etc/systemd/system/b.service' → '/b.service'.
```

```
Created symlink '/etc/systemd/system/multi-user.target.wants/b.service' → '/b.service'.
```

```
systemctl enable ./c.service
```

```
Created symlink '/etc/systemd/system/c.service' → '/c.service'.
```

```
Created symlink '/etc/systemd/system/multi-user.target.wants/c.service' → '/c.service'.
```

```
systemctl enable ./d.service
```

```
Created symlink '/etc/systemd/system/d.service' → '/d.service'.
```

```
Created symlink '/etc/systemd/system/multi-user.target.wants/d.service' → '/d.service'.
```

```
systemctl start a.service
```

# Mounting a root filesystem

(Docker behaves similarly)

```

3400630 execve("/usr/bin/systemd-nspawn", ["systemd-nspawn", "-D", "/home/fred/Work/ez-systemd/basefs"...], "-b",
"--overlay=+:/home/fred/Work/ez-...", 0x7ffe0f9339d0 /* 17 vars */) = 0
3400630 clone(child_stack=NULL, flags=CLONE_NEWNS|SIGCHLD) = 3400633 <-- creates mount namespace to setup root filesystem
3400633 mount(NULL, "/", NULL, MS_REC|MS_SLAVE, NULL) = 0 <-- "remounts" root fs for changes. Changes in host root fs reflected here, but
not other way
3400633 mount("/home/fred/Work/ez-systemd/basefs", "/proc/self/fd/9", NULL, MS_BIND|MS_REC, NULL) = 0 <-- "remounts" basefs for changes
3400633 mount(NULL, "/home/fred/Work/ez-systemd/basefs", NULL, MS_REC|MS_PRIVATE, NULL) = 0
3400633 mount("overlay", "/proc/self/fd/9", "overlay", 0, "lowerdir=/home/fred/Work/systemd...") = 0 <-- "merges" our systemd, experiment,
with basefs to make a root filesystem
3400633 mount("tmpfs", "/proc/self/fd/9", "tmpfs", MS_NOSUID|MS_NODEV|MS_STRICTATIME, "mode=01777,size=10%,nr_inodes=40"... ) = 0
...
3400633 fchdir(9</home/fred/Work/ez-systemd/basefs>) = 0 <-- changes directory to basefs
3400633 pivot_root(".", ".") = 0 <-- makes basefs the new root
3400633 mount(NULL, ".", NULL, MS_REC|MS_SHARED, NULL) = 0
3400633 mount(NULL, "/run/host/incoming", NULL, MS_SLAVE, NULL) = 0
3400633 clone(child_stack=NULL, flags=CLONE_NEWNS|CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWPID|SIGCHLD) = 3400638 <-- fully container'd here

```

# Mounting: Host vs container

```
$ mount | grep fred
```

```
/dev/sda1 on /media/fred/SCARLETT type vfat  
(ro,nosuid,nodev,relatime,uid=1000,gid=1000,f  
mask=0022,dmask=0022,codepage=437,icharset=i  
so8859-1,shortname=mixed,showexec,utf8,flush,  
errors=remount-ro,uhelper=udisks2)
```

Overlay'd directory contents  
are mutable on host fs

```
root@basefs:~# mount | grep fred
```

```
overlay on / type overlay  
(rw,relatime,lowerdir=/home/fred/Work/systemd  
-bug-hashmap:/home/fred/Work/ez-systemd/syste  
md:/home/fred/Work/ez-systemd/config:/home/fr  
ed/Work/ez-systemd/basefs/,upperdir=/var/tmp/  
nspawn-temp-EwD69P/src,workdir=/var/tmp/nspaw  
n-temp-EwD69P/.#src7ebcd0180836e227,uid=on,n  
ouserxattr)
```

# Takeaways

- Docker does full-isolation containers on easy mode
- Docker can be a burden for systems prototyping/testing
- Containers are created through namespaces
- Any process can become a container
- You don't need a isolated root filesystem
- Tools like unshare and systemd-nspawn trivialize containerization

## Learn more

- [An introduction to control groups \(cgroups\) v2](#) - As the name implies by Michael Kerrisk
- [Linux containers in \(less than\) 100 lines of shell](#) - Step-by-step Implement "Docker" with shell scripts by Michael Kerrisk
- [CVE-2022-47929: traffic control noqueue no problem?](#) - An example of a container crashing a host system & potential dangers of user namespaces by Frederick Lawler



# Questions?

✉ [fred@cloudflare.com](mailto:fred@cloudflare.com)