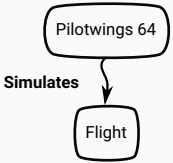
Introduction to OS-Level Virtualization on Linux

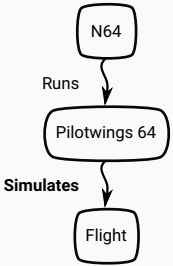
Pedro Bruel
phrb@ime.usp.br

May 25th, 2020

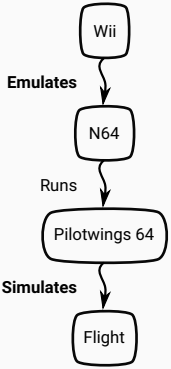
What are Simulation, Emulation, Virtualization?



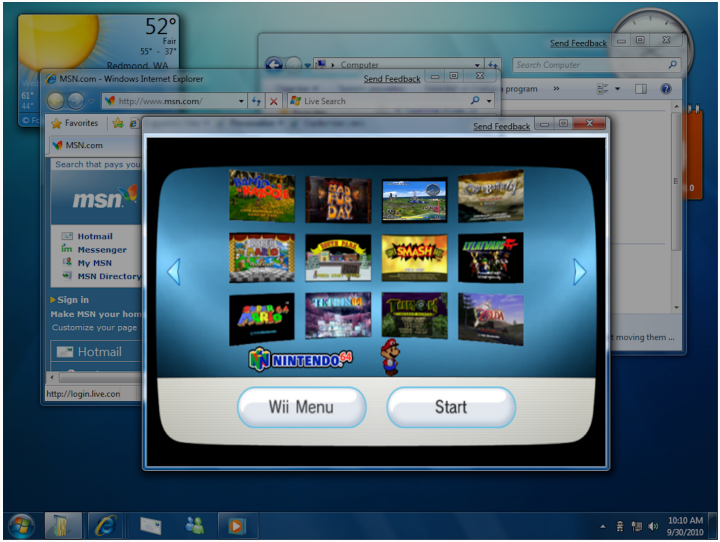
What are Simulation, Emulation, Virtualization?

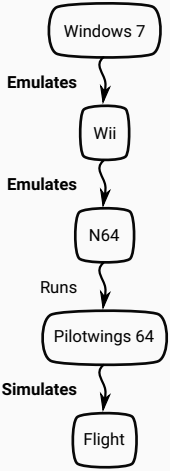


What are Simulation, Emulation, Virtualization?

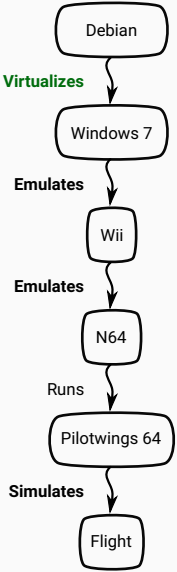


What are Simulation, Emulation, Virtualization?

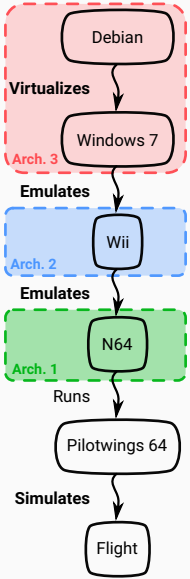




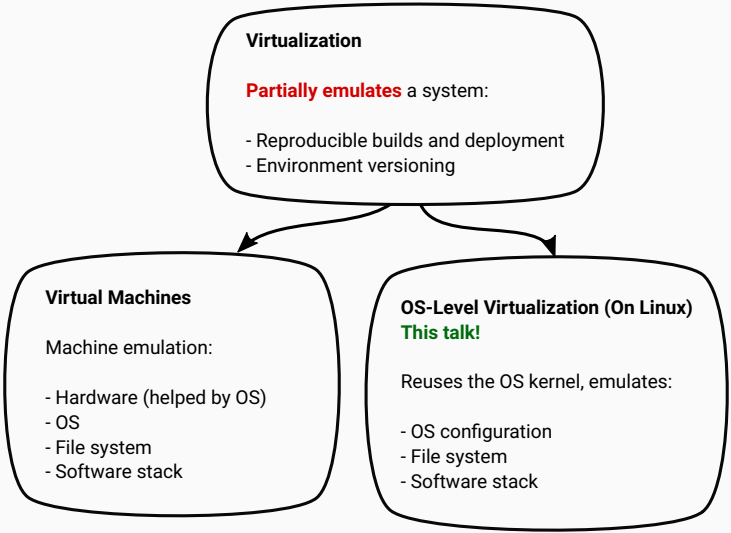
What are Simulation, Emulation, Virtualization?



What are Simulation, Emulation, Virtualization?



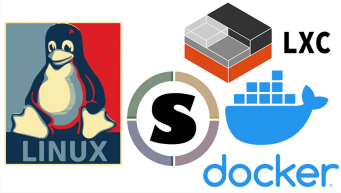
OS-Level Virtualization

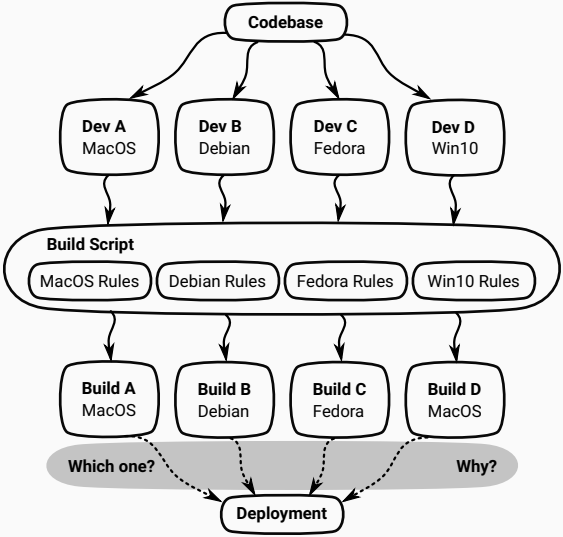


OS-Level Virtualization: Scope of this Talk

Scope

- Why should you use containers?
  - Reproducible builds
  - Environment versioning
  - It's also easier
- How do containers work?
- What tools are available?

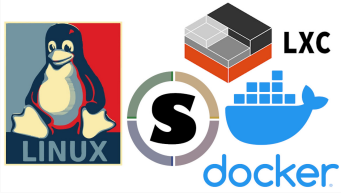


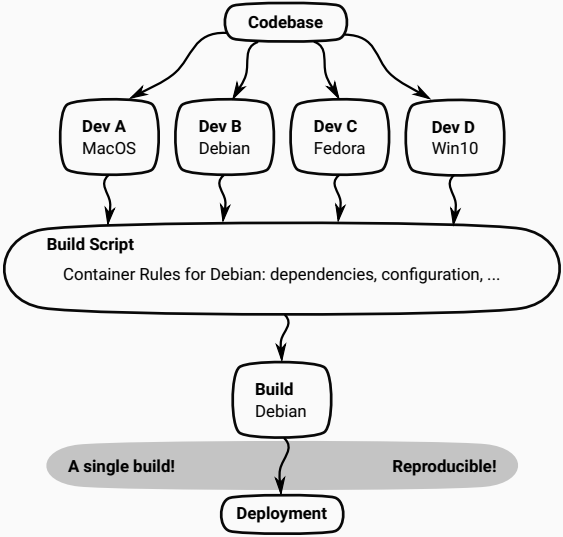


OS-Level Virtualization: Scope of this Talk

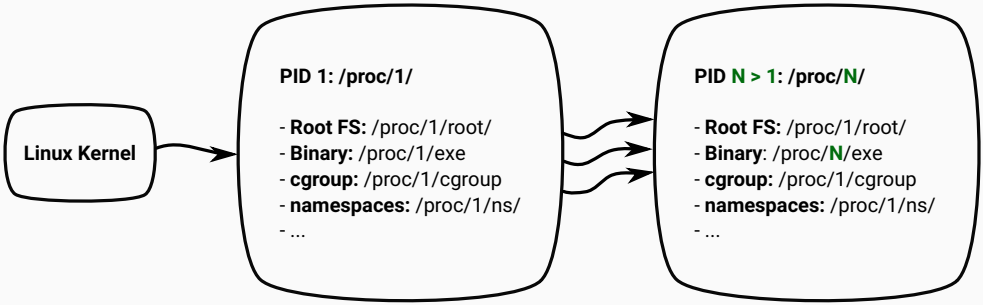
Scope

- Why should you use containers?
  - Reproducible builds
  - Environment versioning
  - It's also easier
- How do containers work?
- What tools are available?

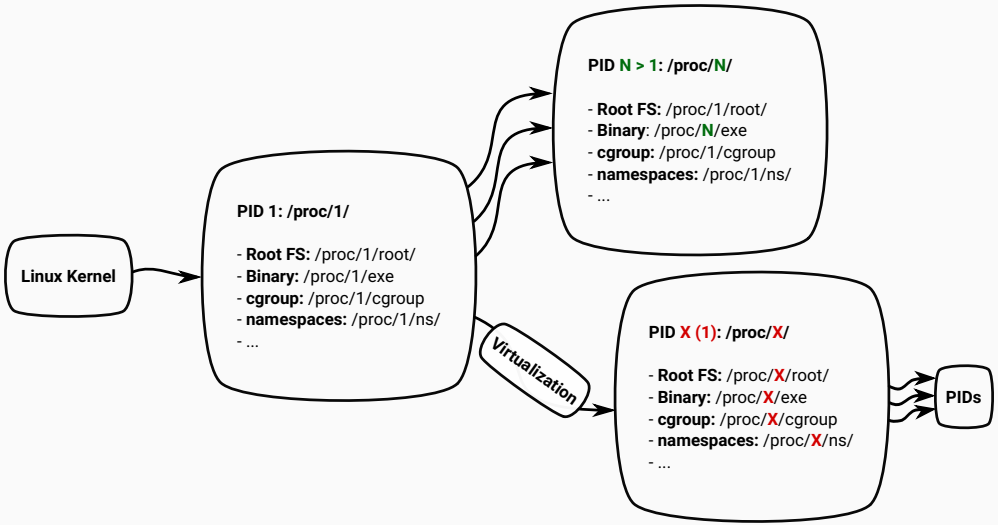




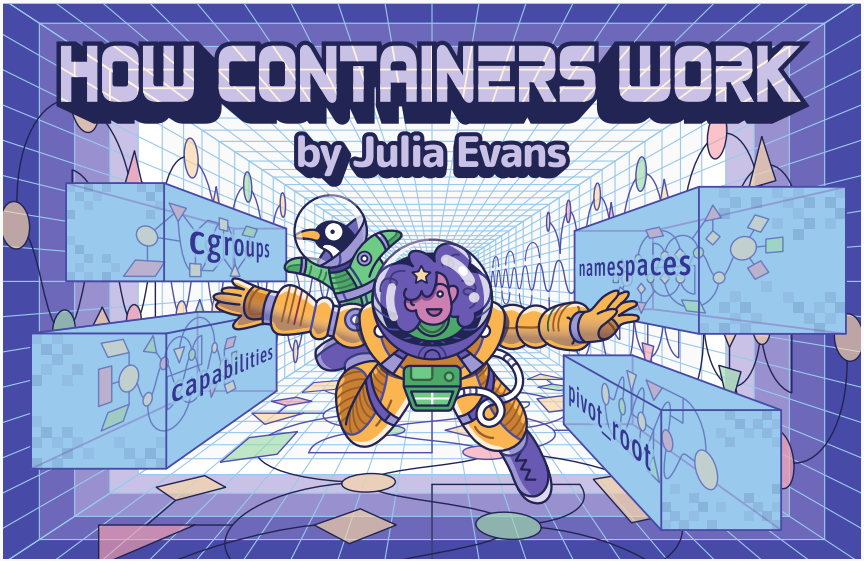
OS-Level Virtualization on Linux



OS-Level Virtualization on Linux

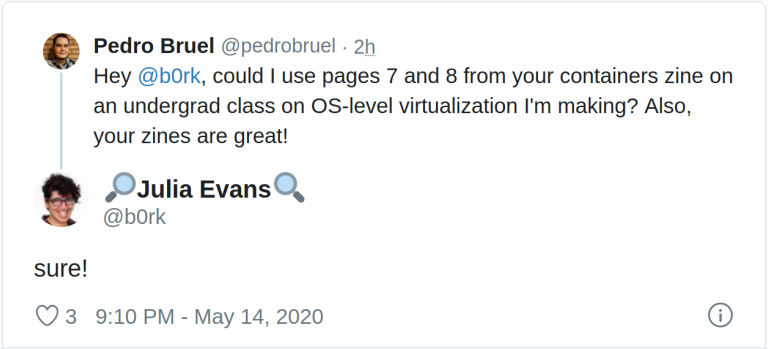


How do Containers Work?



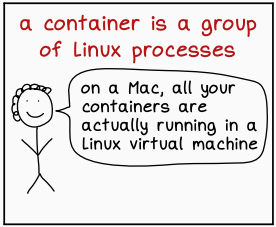
How do Containers Work?

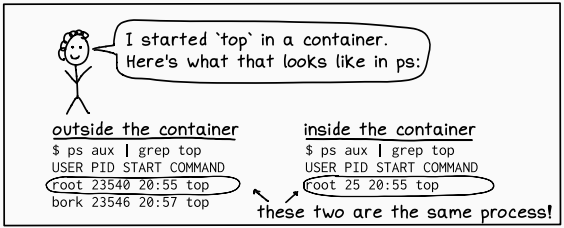
Images used with permission:

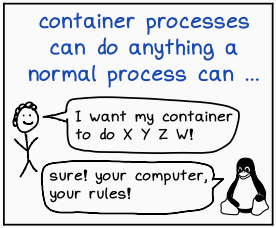


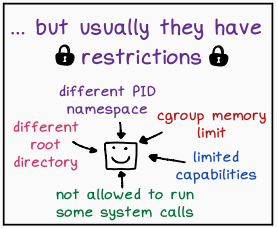
containers = processes

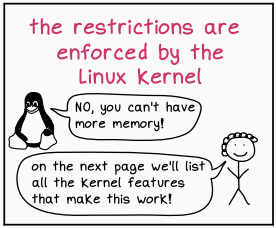
7











container kernel features

8

containers use these Linux kernel features

"container" doesn't have a clear definition, but Docker containers use all of these features.

♥ pivot_root ♥

set a process's root directory to a directory with the contents of the the container image

★ cgroups ★

limit memory/CPU usage for a group of processes

♥ namespaces ♥

allow processes to have their own:

- → network
- → mounts
- → PIDs
- → users
- → hostname
- + more

★ capabilities ★

security: give specific permissions

♥ seccomp-bpf ♥

security: prevent dangerous system calls

★ overlay filesystems ★

this is what makes layers work! Sharing layers saves disk space & helps containers start faster

Containers from Scratch: Obtaing an Image

An image usually means:

- A root file system, and
- Some metadata

We will use the Alpine distribution:

- It's root FS has only 2.4MB
- No need for metadata

Bash Script

```
#!/usr/bin/bash

IMG_DIR="alpine_img"
IMG_REPO="https://us.images.linuxcontainers.org/images"
IMG_URL="$IMG_REPO/alpine/3.11/amd64/default/20200521_13:00/rootfs.tar.xz"
[ ! -d $IMG_DIR ] && \
    mkdir -p $IMG_DIR && \
    curl $IMG_URL | tar xJ -C $IMG_DIR
cd $IMG_DIR
```

Containers from Scratch: Creating cgroups and Setting Limits

We will create a cgroup allowing up to:

- 50% CPU usage: 512/1024 shares
- 10GB of RAM

Script

```
CGROUP_ID="MAC0475-145"
sudo cgcreate -g "cpu,cpuacct,memory:$CGROUP_ID"
sudo cgset -r cpu.shares=512 "$CGROUP_ID"
sudo cgset -r memory.limit_in_bytes=10000000000 "$CGROUP_ID"
```

Containers from Scratch: Launching our Alpine Container

- cgexec: Runs using a cgroup
- unshare: Runs with new namespaces
- chroot: Changes root of the file system
- mount: Here, mounts a new proc directory
- sh: Starts a shell on the container
- We could install depencies now

Script

```
HOSTNAME="alpine-container"
sudo cgexec -g "cpu,cpuacct,memory:$CGROUP_ID" \
     unshare -fmuipn --mount-proc \
     chroot "$PWD/" \
     /bin/sh -c "PATH=/bin && mount -t proc proc /proc && hostname $HOSTNAME && sh"
```

And some cleanup after:

```
sudo cgdelete cpu,cpuacct,memory:/$CGROUP_ID
```