# Abusing the Bridge

Using Docker Images
(and a little Linux magic)
To Provision Baremetal Machines

# Dramatis personæ

- Docker (why we're here)
- DHCP (a venerable protocol for dynamically configuring hosts since RFC 1531 *est. 1993*)
- TFTP (an even older protocol (1981) for, you guessed it, trivial file transfer)
- GRUB (a bootloader)
- CoreOS (an operating system)
- Network Bridging (802.1D)
- And BusyBox (a multicall binary)

# Objective

Deploy a bunch of hosts to the Cloud! Yay! I clicked “Launch” and we’re done!

# Objective

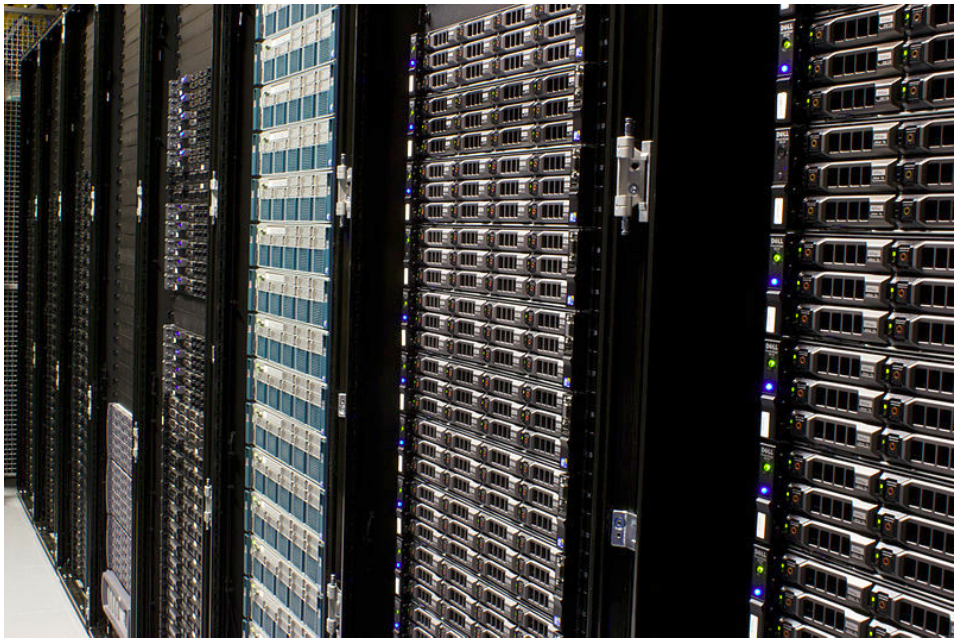
~~Deploy a bunch of hosts to the Cloud! Yay! I clicked “Launch” and we’re done!~~

DEPLOY ON METAL

# How do orgs deploy on baremetal?

A question nearly as old as the processor:

I have a big stack of these machines, how do I deploy something on them?



Wikimedia Foundation Servers
Victorgrigas

# Manually!

1. Insert install CD
2. Click
3. Type
4. Click, Type
5. Type click click type click

Challenges:
Why you don’t want to do it:

- Time consuming
- BORING!

How to sell to other stakeholders:

- Very hard to keep quality consistent when primarily performed by humans.
- Hard to modify procedures (people build specific series of habits).

# Have it baked at the factory

The mechanism is essentially that a set of images is provided to the machine builder. This is probably the easiest way from a time-I-spend perspective. HOWEVER:

- Potentially expensive
- Have to build machine images.
- Out of your control
- More challenging to change build process.
- No capability for redo/deploy in the field.
- Maybe somebody else has more experience?

# PXE

What? I thought those were just in fairytales.

- Stands for Preboot Execution Environment
  - Why did they not use the first letter of all words?
    - ☑ bathroom joke
- Could existing in one of three places:
  - Baked into the BIOS itself (some simple/older systems)
  - Built into the networking card (some fun things can be done with this)
  - Executed by the bootloader as a second stage

- Bootloader? What's that?

# A Quick Overview of the Linux Boot Process

Operating systems are in some sense just another program that a computer runs.  Usually there are a few steps to a working computer.

# A Quick Overview of the Linux Boot Process

The motherboard has a special built-in program called the BIOS* that loads a (very small?) program (usually from a disk drive).

* Recently, BIOS has been replaced by a successor called UEFI, or Unified Extensible Firmware Interface. Intended to make securing computers easier, it has been an interesting technology to play with, but it may actually make computers less secure, and has already been bungled to a degree: http://arstechnica.com/security/2016/08/microsoft-secure-boot-firmware-snafu-leaks-golden-key/

# A Quick Overview of the Linux Boot Process

Usually, the BIOS starts another program called the bootloader. On Linux systems there are two main versions that are used:

- GRUB for desktops/servers/etc.
- Das U-Boot is popular on “Embedded” devices
- fastboot on android

# A Quick Overview of the Linux Boot Process

The Bootloader

- “Initializes” hardware
  - Usually a disk
- Loads a program
  - Linux kernel/”Chainloader”
- May load other information
  - Initrd, recently password/key

# A Quick Overview of the Linux Boot Process

Just like the bootloader the “kernel” (linux) is essenially just another program.

Sometime the bootload loads an “Initial RAM Disk” (initrd)

# A Quick Overview of the Linux Boot Process

Aside: Bootloaders historically have also been used to load diagnostic programs. Examples:

- Memtest (when RAM goes bad)
- VerifyMedium (for verifying the CD you're booting burned correctly)
- You can write your own if you want! In Rust!

  Boot2Rust efi payload

# A Quick Overview of the Linux Boot Process

When the Linux kernel starts, one of the first things it looks for is an initrd.

- Kinda like a zip or tarball (cpio)
- Can be embedded directly into the same file as the kernel
  - Change requires rebuild
- Can be compressed

# A Quick Overview of the Linux Boot Process

If there is an initrd, then start the program called “init” in the initrd.

- Unlock encrypted drives
- Start RAID arrays
- Initialize external hardware
- Mount network shares

# A Quick Overview of the Linux Boot Process

Once init in the initrd is done, then start the program called “init” in the regular root of the filesystem.

- These days in many cases this is systemd everywhere.

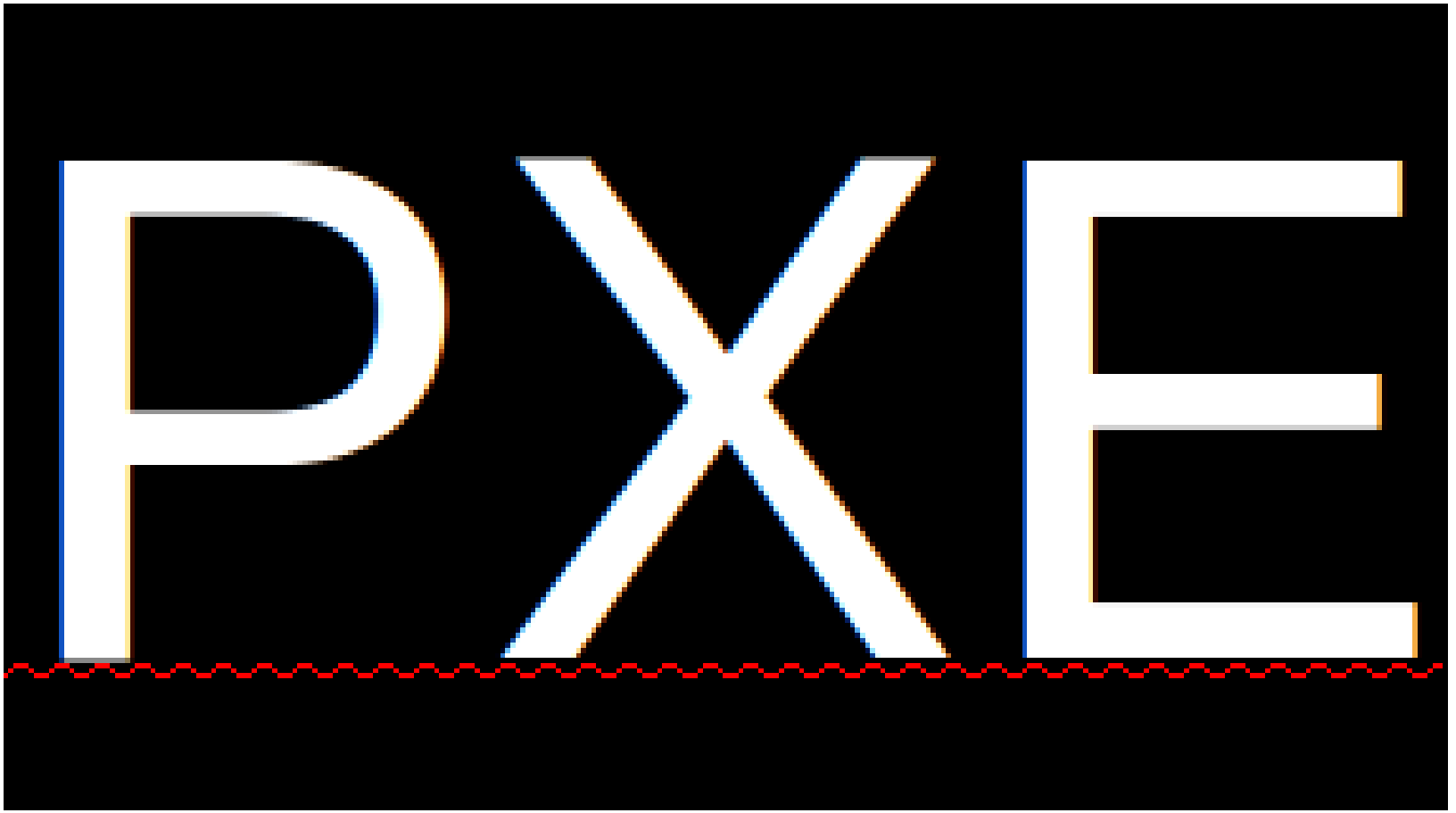
# A Quick Overview of the Linux Boot Process

Initrd: So often required that almost all Linux systems use them, even if they’re not strictly necessary.

- Pet peeve: Ubuntu will rebuild your initrd (sometimes multiple times) when updating. If you are waiting for an update, it’s terrible!

# Whew! That was a lot! What now?

All that was important (maybe) to understand where we are going next.



# PXE

- Basically everything we just talked about, but over the network.
  - BIOS/EFI (or NIC firmware (or bootloader)) loads a bootloader (over network, dhcp directed)
  - Bootloader loads kernel/initrd (over network)
  - PROFIT?

# PXE

- Basically everything we just talked about, but over the network.
  - BIOS/EFI (or NIC firmware (or bootloader)) loads a bootloader (over network, dhcp directed)
  - Bootloader loads kernel/initrd (over network)
  - PROFIT? Not yet, we don’t even have the OS started!
  - Kernel starts, runs init. PROFIT?

# PXE

- Basically everything we just talked about, but over the network.
  - BIOS/EFI (or NIC firmware (or bootloader)) loads a bootloader (over network)
  - Bootloader loads kernel/initrd (over network)
  - PROFIT? Not yet, we don’t even have the OS started!
  - Kernel starts, runs init. PROFIT? NOT YET!
  - Start docker, run services, finally PROFIT!

# PXE

Doesn’t sound too bad. Why doesn’t everybody do it?

- Several services to run:
  - DHCP tells which server and what files to pick for pxe booting.
    - Managing DHCP at this scale is not always awesome.
  - TFTP serves the files. (Not secure!)
  - Repacking custom initrd’s and updating kernels is not awesome either.
  - Not always easy to debug (can’t just unrack a machine and bring it back, because turning it off may lose debug info)

# PXE

Doesn’t sound too bad. Why doesn’t everybody do it?

- The Cattle-vs-Pets distinction is new for many organizations.
- It opens questions of how to automate provisioning, replication, and allocation of the persistent storage that likely exist on these machines.
- No one owns it per-se. Many organizations build products around it, (openstack, etc), but those can be quite complex.

# PXE

Can we make it easier with docker?

Yes!

- Docker can run the services (this is what it does!)
- The servers have to be hooked into some kind of network interface to PXE. We can connect to that.
- Docker uses a special kind of networking tool called a bridge to let containers talk to each other. We can put that on the larger network.

# CoreOS

- PXE booting is in its DNA
- Runs docker natively, so we know the kernel will have all the special goodies
- Actively maintained
- Also has cons:
  - Huge initrd means longer startuptimes (>2 minutes)
  - Company has investment in own tools.
  - Kinda complicated

# Go to demo