Virtual Machines

CS 423 - University of Illinois
Wade Fagen-Ulmschneider
(Slides built from Adam Bates and Tianyin Xu previous work on CS 423.)
Cloud Computing (Generation 1)

★ Dominated by Infrastructure-as-a-Service (IaaS) clouds (and storage services)
  ○ Big winner was Amazon EC2

★ Hypervisors that virtualized the hardware-software interface

★ Customers were responsible for provisioning the software stack from the kernel up
Cloud Computing (Generation 1)

- Guest OS
- Guest OS
- Guest OS

Hypervisor
(Generally Type 2, limited Type 1)

Host Kernel

Hardware
Cloud Computing (Generation 1)

★ Type 2 Hypervisors:
  ○ Strong isolation between different customer’s virtual machines
  ○ VMM is ‘small’ compared to the kernel
    ■ Less LoC means ⇒ less bugs
    ■ Fewer bugs ⇒ usually more security
Cloud Computing (Generation 1)

- Most “practical” attacks on IaaS clouds relied on side channels to detect co-location between attacker and victim VM
  - E.g., we could correlate the performance of a shared resource
  - network RTT's, cache performance

- After co-resident, make inferences about victim’s activities
Cloud Computing (Generation 1)

★ Overall:
  ○ Centralizing the management of hardware ⇒ Increased reliability, Decreased IT costs
  ○ Cheap VMs allows services to run in their own environments (further increasing reliability)
  ○ Extremely high flexibility (you build the OS!), but was all that flexibility needed?
Cloud Computing (Generation 2)

★ Introduction of various service models:
  ○ **CaaS**: Container as a Service
  ○ **PaaS**: Platform as a Service
  ○ **FaaS**: Function as a Service
  ○ **SaaS**: Software as a Service
Why Choose CaaS?

★ Containers provide a known, configurable runtime environment ("user land") without managing an OS or Kernel.

★ **AWS**: Elastic Container Service (ECS)
★ **Google**: Google App Engine
★ ...*many others*...
Why Choose PaaS?

★ Lots of user-level services require configuration, maintenance, and performance optimization ("systems knowledge"). What if this is provided for us?

★ **Databases:** SQL, NoSQL (mongodb), In-Memory (redis), etc

★ **AI/ML Algorithms:** AutoML, Speech Recognition, Image Classification, etc

★ **Build Tools:** Test Suites, Data Pipelines, etc

★ *...hundreds of development platforms...*
Why Choose FaaS?

★ Common to need software to run “on-demand” to some event for short bursts of computation.
  ○ **Examples:**
    - Profile Photo Upload ⇒ Need conversation to many different sizes for various layouts
    - On-Demand Data ⇒ Need creation of a CSV w/ processed data based on user inputs
    - Many computational tasks that are expensive but uncommon

★ **AWS:** Lambdas
★ **Google:** Cloud Functions
Why Choose SaaS?

★ What if you never want to see source code?
  ○ Almost any website you log into can be considered “SaaS”

★ **Systems tools are used to create SaaS platforms** -- but generally SaaS is beyond the scope of systems.
Based on Google Cloud prices for non-preemptable, always-on, and on-demand services with no long-term commitment, sourced from [https://cloud.google.com/appengine/pricing](https://cloud.google.com/appengine/pricing) in April 2021.
Motivation

- Rather than virtualize both user space and kernel space... why not just ‘virtualize’ user space?
- Meets the needs of most customers, who don’t require significant customization of the OS.
- Sometimes called ‘OS virtualization,’ which is highly misleading given our existing taxonomy of virtualization techniques.
- Running natively on host, containers enjoy bare metal performance without reliance on advanced virtualization support from hardware.
Cloud Computing (Generation 1)

- Host Kernel
- Hardware
- Guest Container
- Guest Container
- Guest Container
- Guest Container
Cloud Computing (Generation 1)

- Looks like a VM from the inside!
- Acts like a process from the outside!
Containers vs VMs

A container is a group of processes.

A virtual machine is a fake computer, each one has its own operating system!

Containers use less RAM. This is because they share a single Linux kernel. I can easily run thousands of small containers!

Containers start faster because they're processes and processes start fast.

Containers are more complicated to secure. I'm totally isolated from other VMs on this computer!

It's harder to figure out what you can do in a container. Just pretend I'm a computer! It's easy!

Um, my operating system is still booting.

Um, it really depends on how you configured me...
Containers Aren’t New...

★ Lots of work on containers dating back decades:
  ○ BSD Jails
  ○ Solaris Zones
  ○ Linux containers
  ○ ...etc...

★ ...but weren’t well advertised, not user-friendly (used low-level system interfaces), not easily deployable (usually required root).
Enter: Docker
Docker

★ **Big Idea:** “Build, Ship, and Run App, Anywhere”
  ○ Debug your app, not your environment
  ○ Securely build and share any application, anywhere
  ○ Accomplished by including *everything* in a container
the big idea: include EVERY dependency

containers package EVERY dependency together

to make sure this program will run on your laptop, I'm going to send you every single file on my computer. Exaggeration, but it's the basic idea.

a container image is a tarball of a filesystem

Here's what's in a typical Rails app's container:

- your app's code
- libc + other system libraries
- Ubuntu 18.04 base OS
- Ruby interpreter
- Rails + other Ruby gems

how images are built

0. Start with a base OS
1. Install program + dependencies
2. Configure it how you want
3. Make a tarball of the WHOLE FILESYSTEM
   (This is what 'docker build' does)

running an image

1. Download the tarball
2. Unpack it into a directory
3. Run a program and pretend that directory is its whole filesystem
   (This is what 'docker run' does)

images let you "install" programs really easily

Wow, I can get a Postgres test database running in 45 seconds!
Container Support on OSes

CS 423 - University of Illinois
Wade Fagen-Ulmschneider
(Slides built from Adam Bates and Tianyin Xu previous work on CS 423.)
Containers are Build on Linux Utilities

- **Linux Containers (LXC):**
  - chroot
  - namespace
    - PID, Network, User, IPC, uts, mount
  - cgroups for HW isolation
  - Security profiles and policies
  - Apparmor, SELinux, Seccomp
Containers are Build on Linux Utilities

- Linux Containers (LXC):
  - chroot
  - namespace
    - PID, Network, User, IPC, uts, mount
  - cgroups for HW isolation
  - Security profiles and policies
  - Apparmor, SELinux, Seccomp
Containers are Build on Linux Utilities

* **chroot** changes the apparent root directory for a given process and all of its children.
  - An old idea! POSIX call dating back to 1979
  - **Ex:** `/usr/home/waf/myapp ⇒ /`
    - Process is no longer able to “see” below myapp directory!

* Not intended to defend against privileged attackers.
  - With root access you can do all sorts of things to break out (like chroot’ing again)

* Does not hide processes, network, etc!
Containers are Build on Linux Utilities

★ Linux Containers (LXC):
  ○ chroot
  ○ namespace
    ■ PID, Network, User, IPC, uts, mount
  ○ cgroups for HW isolation
  ○ Security profiles and policies
  ○ Apparmor, SELinux, Seccomp
Namespaces

**namespaces** are the key feature enabling containerization!
- Partition practically all OS functionalities so that different process domains see different things
  - **Mount (mnt):** Controls mount points
  - **Process ID (pid):** Exposes a new set of process IDs distinct from other namespaces (i.e., the hosts)
  - **Network (net):** Dedicated network stack per container; each interface present in exactly one namespace at a time.
  - **IPC (inter-process comm.):** Isolate processes from various methods of POSIX IPC
    - No shared memory between containers!
  - **UTS:** Allows the host to present different host/domain names to different containers.
  - **User ID (user) and cgroup namespace** -- allows the container to **think** its root!
  - ...

★
namespaces

inside a container, things look different
I only see 4 processes in `ps aux`, that's weird...

commands that will look different
- `ps aux` (less processes!)
- `mount` & `df`
- `netstat -tulpn` (different open ports!)
- `hostname`
... and LOTS more

why those commands look different:
:: namespaces ::
I'm in a different PID namespace so `ps aux` shows different processes!

every process has 7 kinds of namespaces

network  mount  user

these 3 came up the most
- PID
- Cgroup

there's a default ("host") namespace
"outside a container" just means "using the default namespaces"

processes can have any combination of namespaces
I'm using the host network namespace but my own mount namespace!
Containers are Build on Linux Utilities

- Linux Containers (LXC):
  - chroot
  - namespace
    - PID, Network, User, IPC, uts, mount
  - cgroups for HW isolation
  - Security profiles and policies
  - Apparmor, SELinux, Seccomp
Namespaces

★ **cgroups** limit, track and isolate utilization of hardware resources including CPU, memory, and disk.
- Important for ensuring QoS between customers! Protects against bad neighbors

★ **Features:**
- Resource limitation
- Prioritization
- Accounting (for billing customers!)
- Control, e.g., freezing groups
- The cgroup namespace prevents containers from viewing or modifying their own group assignment...
Containers are Build on Linux Utilities

★ Linux Containers (LXC):
  ○ chroot
  ○ namespace
    ■ PID, Network, User, IPC, uts, mount
  ○ cgroups for HW isolation
  ○ Security profiles and policies (Apparmor, SELinux, Seccomp)
Security

“Containers do not contain.”
- Dan Walsh (SELinux contributor)
Containers are Build on Linux Utilities

★ It is **real hard** to prove that every feature of the operating system is namespaced.
  ○ Root access to any of these enables pwning the host

★ Solution?
  ○ Secure Linux distributions (ex: SELinux) provide good support for namespace labeling. *Does not prevent against physical attacks (physical security is part of security)*!
  ○ Much easier to express a correct isolation policy over a coarse-grained namespace than, say, individual processes.
Julia Evans @b0rk

seccomp-bpf

all programs use system calls

- read 2000 bytes from this file

- program

- here you go!

Linux

some programs have security vulnerabilities

- I know ffmpeg codecs can be exploited but I really need to process these untrusted videos...

- I know ffmpeg codecs can be exploited but I really need to process these untrusted videos...

rarely used syscalls can help an attacker

- process-vm-readv

- reboot

- read memory from another process

- request-key

- ffmpeg DEFINITELY doesn't need access to read memory from other programs!

seccomp-BPF: make Linux run a tiny program before every system call

- reboot the computer!

- process

- the BPF program I was given returned false, that's a no from me!

- Docker

Docker blocks dozens of syscalls by default

- most programs don't need those system calls so I told Linux to block them for you

2 ways to block scary system calls

1. Limit a container's capabilities
2. Use a seccomp-BPF whitelist

Usually people do both!
Figure 1: LSM Hook Architecture
Containers aren't magic

These 15 lines of bash will start a container running the fish shell. Try it!
(download this script at bit.ly-containers-arent-magic)

```
wget bit.ly/fish-container -O fish.tar
mkdir container-root; cd container-root
tar -xf ../fish.tar
cgroup_id="cgroup_$(shuf -i 1000-2000 -n 1)"
cgcreate -g "cpu,cpuaacct,memory:$cgroup_id"
cgset -r cpu.shares=512 "$cgroup_id"
cgset -r memory.limit_in_bytes=1000000000 "$cgroup_id"
cgexec -g "cpu,cpuaacct,memory:$cgroup_id" \
    unshare -fmuipn --mount-proc \
    chroot "$PWD" \
    /bin/sh -c \
    /bin/mount -t proc proc /proc && 
    hostname container-fun-times && 
    /usr/bin/fish"
```

# 1. download the image
# 2. unpack image into a directory
# 3. generate random cgroup name
# 4. make a cgroup & set CPU/memory limits
# 5. use the cgroup
# 6. make + use some namespaces
# 7. change root directory
# 8. use the right /proc
# 9. change the hostname
# 10. finally, start fish!