Container
Yiying Zhang
Quiz 1 at the end of today’s lecture
  • 15 minutes, open everything, turn on your camera
Course turned into virtual only for this entire quarter
Project proposal due this Wed (1/26), no more extension!
Outline

• Motivation and overview
• Linux container techniques
• Docker
• Security of containers

Acknowledgment: some slides from “Introduction to Docker” (Docker Inc.)
Invented by IBM in 1960s for sharing expensive mainframes

Popular research ideas in 1960s and 1970s

Interest died as the adoption of cheap PCs and multi-user OSes surged in 1980s

A (somewhat accidental) research idea got transferred to VMware

Real adoption happened with the growth of cloud computing

New forms of virtualization: container and serverless, in the modern cloud era
Do VMs Fit All Today’s Cloud User Needs?

- Performance overhead of indirections (guest OS and hypervisor)
- Large memory footprint
- Slow startup time
- License and maintenance cost of guest OS
Do VMs Fit All Today’s Cloud User Needs?

• Performance overhead of indirections (guest OS and hypervisor)
• Large memory footprint
• Slow startup time
• License and maintenance cost of guest OS

• Do we really need to virtualize hardware and a full OS?
Do VMs Fit All Today’s Cloud User Needs?

- Performance overhead of indirections (guest OS and hypervisor)
- Large memory footprint
- Slow startup time
- License and maintenance cost of guest OS

- Do we really need to virtualize hardware and a full OS?
- What about DevOps?
The Challenge

Static website
nginx 1.5 + modsecurity + openssl + bootstrap 2

Background workers
Python 3.0 + celery + pyredis + libcurl + libmpg + libopencv + nodejs + phantomjs

User DB
postgresql + pgv8 + v8

Queue
Redis + redis-sentinel

Analytics DB
hadoop + hive + thrift + OpenJDK

Web frontend
Ruby + Rails + sass + Unicorn

API endpoint
Python 2.7 + Flask + pyredis + celery + psycopg + postresql-client

Development VM
QA server

Public Cloud

Customer Data Center

Disaster recovery

Production Servers

Production Cluster

Can I migrate smoothly and quickly?

Do services and apps interact appropriately?
### The Matrix from Hell

|----------------|----------------|-----------|--------------------|---------------|--------------|----------------------|------------------|
Cargo Transportation Pre-1960

- Multiplicity of Goods
- Multiplicity of methods for transporting/storing

Do I worry about how goods interact (e.g., coffee beans next to spices)?

Can I transport quickly and smoothly (e.g., from boat to train to truck)?
Also a Matrix from Hell

| ![Train] | ![Building] | ![Forklift] | ![Crane] | ![Ship] | ![Dock] | ![Truck] |
Solution: Shipping Container

A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

...in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another.
Docker: Container for Code

An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container...

...that can be manipulated using standard operations and run consistently on virtually any hardware platform.

- Development VM
- QA server
- Customer Data Center
- Public Cloud
- Production Cluster
- Contributor's laptop

Do services and apps interact appropriately?
Why Does It Work? Separation of Concerns

- **Dan the Developer**
  - Worries about what’s “inside” the container
    - His code
    - His Libraries
    - His Package Manager
    - His Apps
    - His Data
  - All Linux servers look the same

- **Oscar the Ops Guy**
  - Worries about what’s “outside” the container
    - Logging
    - Remote access
    - Monitoring
    - Network config
  - All containers start, stop, copy, attach, migrate, etc. the same way
Why Developers Care?

- Build once... (finally) run anywhere*
- A clean, portable runtime environment for your app
- No worries about missing dependencies, packages, etc. during deployments
- Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying
- Automate testing, integration, packaging...anything you can script
- Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers
- Deploy services like VM, but without the overhead of a VM

* "anywhere" means an x86 server running a modern Linux kernel (3.2+ generally or 2.6.32+ for RHEL 6.5+, Fedora, & related)
Why Administrators Care?

- Configure once... run anything
- Make the entire lifecycle more efficient, consistent, and repeatable
- Eliminate inconsistencies between development, test, production, and customer environments
- Support segregation of duties
- Significantly improves the speed and reliability of continuous deployment and continuous integration systems
- Address significant performance, costs, deployment, and portability issues normally associated with VMs
Containerization timeline

- **Docker containerization debuted** in 2013.
- In 2017, **20% of companies** have containers deployed.
- By 2020, **50% of companies** have containers deployed.

*Source: Gartner*
Linux Containers

- Run everywhere
  - Regardless of kernel version
  - Regardless of host distro
  - Physical or virtual, cloud or not
  - Container and host architecture must match...

- Run anything
  - If it can run on the host, it can run in the container
  - If it can run on a Linux kernel, it can run
At High-Level: It Looks Like a VM

- Own process space
- Own network interface
- Can run stuff as root
- Can have its own /sbin/init (different from the host)
At Low-Level: OS-Level Virtualization

• Containers run on a host OS directly (and share the OS)
• Run as processes
• OS provides resource isolation and namespace isolation
VM vs Container

Containers are isolated, but share OS and, where appropriate, bins/libraries

...result is significantly faster deployment, much less overhead, easier migration, faster restart
Why are Containers Lightweight?

**VMs**

- App A
  - Bins/Libs
  - Guest OS

Every app, every copy of an app, and every slight modification of the app requires a new virtual server

**Containers**

- App A
  - Bins/Libs

Original App (No OS to take up space, resources, or require restart)

- Copy of App
  - No OS. Can Share bins/libs

Modified App

Copy on write capabilities allow us to only save the diffs between container A and container A'
Using Namespaces to Separate “Views” of Users

- Namespace: naming domain for various resources
  - User IDs (UIDs)
  - Process IDs (PIDs)
  - File paths (mnt)
  - Network sockets
  - Pipe names
Namespaces Isolated by Kernel

- **Container1**
  - UIDs: 1, 2, 3, ...
  - PIDs: 1, 2, 3, ...
  - Paths: /, /usr, /home, ...

- **Container2**
  - UIDs: 1, 2, 3, ...
  - PIDs: 1, 2, 3, ...
  - Paths: /, /usr, /home, ...

---

**Linux Kernel**

- `setuid()`
- `getpid()`
- `open()`
- ...
Isolating Resources with cgroups

- Linux Control Groups (cgroups): collection of Linux processes
  - Limits resource usages at group level (e.g., memory, CPU, device)
  - Fair sharing of resources
  - Track resource utilization (e.g., could be used for billing/management)
  - Control processes (e.g., pause/resume, checkpoint/restore)
Efficiency: *almost* no overhead

- Processes are isolated, but run straight on the host.
- CPU performance = native performance.
- Memory performance = a few % shaved off for (optional) accounting.
- Network performance = small overhead; can be optimized to zero overhead.
Docker

- Docker Inc
  - Founded as dotCloud, Inc. in 2010 by Solomon Hykes (renamed to Docker Inc. in 2013)
  - Estimated to be valued at over $1 billion (101-250 employees)
- Docker the software
  - A container engine written in Go (based on Linux container)
- Docker community
  - Now 2150 contributors, 17.9K forks of docker engine on GitHub (called Moby)
25% of companies have adopted Docker.
Portion of Hosts Running Docker

DOCKER RUNS ON 20% OF HOSTS

Source: Datadog
DOCKER DEPLOYMENT SIZE HAS INCREASED 75% IN ONE YEAR
Top Technologies Running on Docker

- NGINX: 37%
- REDIS: 33%
- POSTGRES: 29%
- FLUENTD: 25%
- ELASTICSEARCH: 24%
- MONGO: 20%
- MYSQL: 18%
- ETCD: 12%
- RABBITMQ: 8%
- HAProxy: 7%

Source: Datadog
What are the Basics of a Docker System?
Docker Engine

- daemon: Rest API (receiving instructions) and other features
- containerd: Execution logic (e.g., start, stop, pause, unpause, delete containers)
- runc: A lightweight runtime CLI

Docker Images

- not a VHD, not a file system
- uses a Union File System to “overlay”
- a read-only Layer
- do not have state
- Basically a tar file
- Has hierarchy (arbitrary depth)
**overlay filesystems**

**Docker images can be really big**

Docker images contain Python packages installed in them.

**every container needs a "copy" of its image**

Installing new programs on top of the Ubuntu 16.04 image.

**solution: ★ overlays ★**

- Container's changes
- Merged result

- `/etc/apt/sources.list`
- `/files/something.txt`
- `/bin/bash /bin/cat`
- `/etc/apt/sources.list`

Containers can use the same base image without wasting disk space on copies!

**how to overlay**

Linux has an 'overlayfs' driver that you can use to overlay directories like this:

```
$ mount -t overlay overlay
-o lowerdir=/lower,upperdir=/upper,workdir=/work
/merged
```

- Base directory, will be read only
- Where changes will go
- Must be empty

Try it out! It's really easy.

**how Docker runs containers**

1. Unpack the base image into a directory
2. Make an empty directory for changes
3. Overlay the new directory on top of the base
4. Start the container!
Docker Image Registry

- Registry containing docker images
  - Local registry on the same host
  - Docker Hub Registry: Globally shared
  - Private registry on docker.com
Changes and Updates

Base Container Image

Container Mod A'

Container Mod A''

Docker Container Image Registry

App A

App A

Bins/Libs

Host is now running A''

Docker Engine

App A

App A

Bins/Libs

Docker Engine

Host running A wants to upgrade to A''. Requests update. Gets only diffs
Security Implications of Containers

TABLE 1. Threat model specifications for apps, containers, and host for the studied use cases. ‘Semi’ refers to semi-honest. Apps in semi-honest/malicious containers can be semi-honest or malicious too.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Apps can be honest</th>
<th>Apps can be semi malicious</th>
<th>Containers can be honest</th>
<th>Containers can be semi malicious</th>
<th>Host can be honest</th>
<th>Host can be semi malicious</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I) Protect container from applications</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(II) Inter-container protection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(III) Protect host from containers</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(IV) Protect containers from host</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Source: S. Sultan et al.: Container Security: Issues, Challenges, and the Road Ahead
Threats of Containers

- Unlike VMs whose interface is hardware instructions, containers’ interface is OS system calls.
- More difficult to protect syscalls:
  - Involve large amount of code in the OS.
  - And there are many syscalls.
Threats of Container Images

- Difficult to understand the source/provenance of images

Figure 1: Scenario of vulnerability spread

Source: B. Tak et al.: Understanding Security Implications of Using Containers in the Cloud