Content

• Introduction to virtualization
• Virtualization through VM
• Virtualization through Containers
From mono-core to exa-scale computer

Unreliable not very fast network

Virtualisation

Abstraction

No central control

Reliable fast network

central control

Bigger systems

central control

Aggregation

CPP Landscape

... From ~ 1986 to ~ 2023
Virtualisation environment components

Service container allows for running multiple services on one computer/OS

- Web Services, WSDL
- Services isolation

Web based Application and Web Services

Y. Demchenko 27-28 Nov 2012, HK PolyU
Abstract Pool automate
Content

• Virtualization
• Virtualization through VM
• Virtualization through Containers
Origins of the Virtualization

• During the initial surge of interest in virtualization in the 1960s the **motivating factors** were strong.
  – Operating systems ran directly on hardware, providing services directly to applications
  – **BUT compatibility** was a **major issue** due to the number of **different architectures** being pioneered at the time
First attempt

- IBM developed the VM370 in the early 70’s,
  “the limitations of the hardware of the time along with inadequate/awkward architectures hampered progress”

- In 1998, VMware figured out how to virtualize the x86 platform, once thought to be impossible, and created the market for x86 virtualization.

An old idea: x86 hardware virtualization
http://www.os2museum.com/wp/?p=1213
Virtualization types

- **Operating system**
- **Paravirtualization** (OS assisted Virtualization)
- **Full virtualization**
- **Hardware assisted**
Virtualization is now a must

• as **data centres** and **server farm** populations grew from **hundreds** to **hundreds of thousands** of servers
  ➢ maintaining Large collection of **small physical machines** became inefficient and expensive to run

**Virtualization offers** the opportunity to consolidate a large number of small machines on one larger server, easing manageability and allowing resources to be effectively prioritized
Virtualisation and hypervisor allows for running multiple OS on one computer/OS

- Cloud Management Software provides flexible VM management
- Hypervisor provides VM isolation and CPU, Memory, I/O virtualisation
Content

• Virtualization
• Virtualization through VM
• Virtualization through Containers
Containers from Scratch

containers are just isolated groups of processes(s) running on a single host. That isolation leverages several underlying technologies built into the Linux kernel:

– namespaces
– cgroups

prominent advantages of containers

• Flexibility
• Convenience
• Consistent
• Reproducibility

Source: Eric Chiang “Containers from Scratch”
https://ericchiang.github.io/post/containers-from-scratch/
Linux Kernel Support

• **cgroups**: limit how much resource the process can use
  – CPU
  – Memory
  – Network

• **namespaces**: limit what the process can see
  – pid
  – net
  – mnt
  – ipc
Process of creating a container

**Virtual bridge**

**System Net Namespace**
- Networks (eth)
- Other network resources

**System mnt Namespace**
Isolation of mountpoints /, /dev, /boot, ...

**System ipc Namespace**
Isolation of system V IPC objects and message queues

**System uts Namespace**
Isolation of system identifiers (hostname)

**System PID Namespace**

<table>
<thead>
<tr>
<th>process</th>
<th>PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>System (init)</td>
<td>1</td>
</tr>
<tr>
<td>[kthread]</td>
<td>2</td>
</tr>
<tr>
<td>docker process</td>
<td>1181</td>
</tr>
<tr>
<td>Shell</td>
<td>1194</td>
</tr>
<tr>
<td>top</td>
<td>7826</td>
</tr>
</tbody>
</table>

**Container Net Namespace**
- Networks (veth)
- Other network resources

**Container mnt Namespace**
Isolation of mountpoints /, /dev, /boot, ...

**Container ipc Namespace**
Isolation of system V IPC objects and message queues

**Container uts Namespace**
Isolation of system identifiers (hostname)

**Container PID Namespace**

<table>
<thead>
<tr>
<th>PID</th>
<th>process</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>shell</td>
</tr>
<tr>
<td>2</td>
<td>top</td>
</tr>
</tbody>
</table>

**container uid Namespace**

<table>
<thead>
<tr>
<th>user</th>
<th>uid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root privileged user</td>
<td>0</td>
</tr>
</tbody>
</table>

**System uid Namespace**

<table>
<thead>
<tr>
<th>uid</th>
<th>user</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>root</td>
</tr>
<tr>
<td>1</td>
<td>bin</td>
</tr>
<tr>
<td>2</td>
<td>daemon</td>
</tr>
<tr>
<td>2310</td>
<td>Unprivileged user</td>
</tr>
<tr>
<td>72</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>user1</td>
</tr>
</tbody>
</table>

**cgroups**
Container vs Container-image

**Container-Image**

- Binary representation of the container stored on disk

- **image layering**: Parent child relationships
  - Image can be created starting from an existing image
  - The tree structure of image helps to fix vulnerability

- Images can be created/built from “configuration file” (dockerfile)
  - but also from running containers (save container instance)

**Container**

- a **running** instance of Container-Image

- packaged with its dependencies
  - Nothing is installed on the host
  - multiple container with conflicting libs can run on the same host.
  - When the container stops everything go to clean state

- The lifecycle of a process running into the container is tight to the lifecycle of the container
  - process state
Container vs Container-image

Base Image: 
- ubuntu:latest

New Image: iid1

Container: cid1
- cmd → new state

Container: cid1
- run

Containers: cid2, cid3, cid4
- run
Terminology

**Container Image**
- Persisted snapshot that can be run
  - images: List all local images
  - run: Create a container from an image & execute a command in it
  - tag: Tag an image
  - pull: Download image from repository
  - rmi: Delete a local image
    - This will also remove intermediate images if no longer used

**Container**
- Runnable instance of an image
  - ps: List all running containers
  - ps  --a: List all container (incl. stopped)
  - top: Display processes of a container
  - start: Start a stopped container
  - stop: Stop a running container
  - pause: Pause all processes within a container
  - rm: Delete a container
  - commit: Create an image from a container
Example of container technology
Docker

• ...
• 2008: LXC (linux Container)
• 2013: Docker ➔ build on LXC
• 2016: Docker 1.10 ➔ runc and containerd

Docker image
• is a the union of one more read-only layers
• Created following instruction defined in Dockerfile
• Cache option to share layers.
• Uses volumes to store data outside the container
• Default docker storage /var/lib/docker/
Sharing layers across containers

(a) 
(b) 
(c) 
(d) 
(e)
Content

• Virtualization
• Virtualization through VM
• Virtualization through Containers
• VM vs Containers
VM vs containers

Virtual Machines
Each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB.
Why are containers lightweight?

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

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 & container A'.
Changes and Updates

Docker Engine

Host running A wants to upgrade to A''. Requests update. Gets only diffs

Host is now running A''
Containers lifecycle management tools

1. Containerization
2. Discovery and Global Configuration Stores
3. Networking Tools
4. Scheduling, Cluster Management, and Orchestration

Reference: The Docker Ecosystem: An Introduction to Common Components
https://www.digitalocean.com/community/tutorials/the-docker-ecosystem-an-introduction-to-common-components
Container technologies

• Docker (1)
• Singularity (2)
• Charliecloud (3)
• Shifter (4)
• LXC, OpenVZ, uDocker …

(1) https://docker.org
(2) https://singularity.lbl.gov
(3) https://charliecloud.readthedocs.io/en/latest/
(4) https://www.nersc.gov/research-and-development/user-defined-images/