Advanced Virtualization in Computing

Jorge Gomes <jorge@lip.pt>
Virtualization many types ... 

<table>
<thead>
<tr>
<th>Type</th>
<th>Some examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network Virtualization:</strong></td>
<td>VLANs, vswitches, ...</td>
</tr>
<tr>
<td><strong>Storage Virtualization:</strong></td>
<td>Logical Volumes, ...</td>
</tr>
<tr>
<td><strong>Computer Virtualization:</strong></td>
<td>Virtual Machines, ...</td>
</tr>
<tr>
<td><strong>Operating System Virtualization:</strong></td>
<td>Containers, ...</td>
</tr>
</tbody>
</table>
Virtual Machine

“Virtual Machine an efficient, isolated duplicate of a real computer machine.”

Formal Requirements for Virtualizable Third Generation Architectures (1974)

Gerald J. Popek and Robert P. Goldberg
Computer virtualization

We are going to focus on Virtual Machines (VM).
History

• 1966 CP/40 for S/360-40
  • research project, introduced CP and CMS
  • first full virtualization capable system
• 1967 IBM CP/CMS for S/360-67
  • first virtualization in production
• 1972 IBM VM/370 => CP/CMS for IBM S/370

Applications
Cambridge Monitor System (CMS)
Control Program(CP) ➔ hypervisor
Mainframe Hardware
History

• Late 90’s the microprocessors become more powerful and multiprocessor machines (SMP) cheap.
• A single microprocessor based machine could now support multiple services and/or applications.

• Virtualization gained interest again.
  • 1999 VMware workstation
  • 2000 User Mode Linux (UML)
  • 2003 Xen for Linux
Types of Virtualization
Bare metal vs Virtualization

Bare metal

- App A
- OS Y
- Physical Machine 1
- App B
- App C
- OS Z
- Physical Machine 2

Virtualization

- App A
- OS Y
- VM 1
- Hypervisor / Host OS
- VM 2
- OS Z
- Physical Machine
- App B
- App C

Consolidation

- Less space
- Less energy
- Less hardware
- Faster provisioning
- More flexibility
- Burst to cloud
- Easier to manage
Common types of virtualization

**Emulation**
- Both kernels unchanged
- Emulated hardware
- Ex. QEMU

**Paravirtualization**
- Both kernels changed
- Emulation replaced by hypercalls to the host
- Ex. Xen

**Hardware assisted virtualization**
- Both kernels unchanged
- Emulation replaced by hardware assisted hypervisor
- Ex. KVM
**Rings and hardware virtualization**

- Rings are hierarchical protection domains within the CPU
- Lower rings have higher privileges in the processor
- Intel VT-x and AMD-V add a ring -1 for hypervisors

<table>
<thead>
<tr>
<th>RING 3</th>
<th>RING 2</th>
<th>RING 1</th>
<th>RING 0</th>
<th>RING -1</th>
</tr>
</thead>
<tbody>
<tr>
<td>App</td>
<td>App</td>
<td>App</td>
<td>App</td>
<td>HYPERVERSOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OS KERNEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OS KERNEL</td>
</tr>
<tr>
<td>OS KERNEL</td>
<td>OS KERNEL</td>
<td>OS KERNEL</td>
<td>OS KERNEL</td>
<td>HYPERVERSOR</td>
</tr>
</tbody>
</table>

**Image:**

- A diagram showing the different rings and their respective applications and operating systems.
Operating System Level Virtualization
a.k.a Containers
Why do we need hypervisors?

| Use different operating system implementations in the same physical machine (eg. Linux and Windows simultaneously) | X |
| Limit security breaches (isolation between applications or operating systems) | X |
| Better resource allocation and consumption control (memory, CPU, IO bandwidth, etc) | X |
| Flexible infrastructure (easier provisioning, capacity and resource management in large facilities) | X |
| Use same OS or very similar but with different system environments customized for several applications | X |

We need VM hypervisors because OSes are not capable...

“hypervisors are the living proof of operating system's incompetence.”

The Failure of Operating Systems & How We Can Fix It
Depending on the purpose hypervisors and virtual machines can be the wrong tool for the job ...
Operating system level virtualization

- Multiple environments via OS isolation features
- Limits what processes can do and see
- Same OS kernel is shared and directly used
- More efficient than VMs
OS level virtualization advantages

- Less memory consumption
  - No need of duplicated kernels and related processes
  - No duplication of buffering and shared memory
  - Less memory split across execution domains

- Faster I/O and execution and less latency
  - Direct execution on top of one single kernel
  - No emulation, No hypercalls, No buffer copies

- Don’t need to run OS services in each isolated environment
  - No need of duplicated NTP, SNMP, CRON, DHCP, SYSLOG, SMART, etc

- Much faster start–up times
  - No OS boot, smaller images to transfer and store

- Less management effort
  - Only the host machine needs to be managed (many-core is great)
### OS level virtualization also not new

<table>
<thead>
<tr>
<th></th>
<th>Year</th>
<th>File system isolation</th>
<th>I/O limits</th>
<th>Memory limits</th>
<th>CPU quotas</th>
<th>Network isolation</th>
<th>Root priv isolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>chroot</td>
<td>1982</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jail</td>
<td>FreeBSD</td>
<td>1998</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Linux-VServer</td>
<td>Linux</td>
<td>2001</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Virtuozzo Containers</td>
<td>Linux- Windows</td>
<td>2001</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Zones</td>
<td>Solaris</td>
<td>2004</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>OpenVZ</td>
<td>Linux</td>
<td>2005</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HP Containers</td>
<td>HP/UX</td>
<td>2007</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>LXC</td>
<td>Linux</td>
<td>2008</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Docker</td>
<td>Linux</td>
<td>2013</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Linux kernel features

- **Kernel namespaces**: isolate system resources from process perspective
  - **Mount** namespaces: isolate mount points
  - **UTS** namespaces: hostname and domain isolation
  - **IPC** namespaces: inter process communications isolation
  - **PID** namespaces: isolate and remap process identifiers
  - **Network** namespaces: isolate network resources
  - **User** namespaces: isolate and remap user/group identifiers
  - **Cgroup** namespaces: isolate Cgroup directories
- **Seccomp**: system call filtering
- **Cgroups**: process grouping and resource consumption limits
- **POSIX capabilities**: split/enable/disable root privileges
- **chroot**: isolated directory trees
- **AppArmor** and **SELinux**: kernel access control
Namespaces

$ ls -l /proc/$$/ns

You are already using them!
Container

- Runs programs as processes in a standard way
- No emulation or hypervisors
- Just process isolation
- Therefore much more efficient
Containers
Container putting it together

- Host Ubuntu
  - CentOS 6
  - loopback
  - Volume
  - /loop10
  - /etc

- Mount namespace
  - /home

- Pid namespace
  - /etc

- Mount filesystem
  - /

- Bind
  - /home

Program
Container putting it together

To create a container image:
- Add the required libraries, programs and data to the file-system
- Add the required programs to the container file-system

Can I run another Linux distribution using containers?
- Yes sure
- The Linux kernel ABI remains largely unchanged across versions

Containers are usually started by the root user:
- Some operations require privileges
- Can be root user inside a container without affecting the host or the other containers (with POSIX capabilities, seccomp and namespaces)
LXC/LXD
Linux Containers project (LXC)

• First open source project to provide a toolset for containers

• Create and manage containers using the Linux Kernel features:
  – liblxc library
  – Bindings for several languages (python, ruby, lua, Go)
  – Templates
  – Tools to create/manage containers

• Tools:
  – lxc-create, lxc-destroy, lxc-start, lxc-stop, lxc-execute, lxc-console,
  – lxc-monitor, lxc-wait, lxc-cgroup, lxc-ls, lxc-ps, lxc-info, lxc-freeze,
  – lxc-unfreeze

• Limitations:
  – Requires considerable knowledge and effort
**LXD**

- Newer development from the original Linux Containers project
- Pushed and supported by Canonical (Ubuntu)

**Objective:**
- Provide an environment to run complete Linux OS distributions within containers
- More similar to an hypervisor but using namespaces
- "boot" the complete OS distribution
- Images are tarballs

**Limitations:**
- Limited support and adoption beyond Ubuntu
- Fairly recent
docker
Docker containers are oriented to services composition:
- (Services or Applications) + (runtime environment)
- Self-contained and lightweight
- Run it everywhere (Linux)

DevOps \implies integration of IT development and operations
- DevOps requires strong automation
- Developers: focus on what's inside the container
- Operations: may focus in the underlying infrastructure

$ docker run -i -t centos:centos6
[root@28f89ada747e /]# cat /etc/redhat-release
CentOS release 6.8 (Final)
Docker

- Docker images can be fetched from the Docker Hub repository
  - There are other Docker container repositories besides Docker Hub
  - Very convenient to transfer and share containers pull/push

<table>
<thead>
<tr>
<th>Supported tags and respective Dockerfile links</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.10  . artful-20171019  . artful  . rolling  . devel (artful/Dockerfile)</td>
</tr>
<tr>
<td>14.04  . trusty-20170917  . trusty (trusty/Dockerfile)</td>
</tr>
<tr>
<td>16.04  . xenial-20171005  . xenial  . latest (xenial/Dockerfile)</td>
</tr>
<tr>
<td>17.04  . zesty-20170915  . zesty (zesty/Dockerfile)</td>
</tr>
</tbody>
</table>
Docker

- Docker container image is composed of:
  I. Multiple file-system layers each one:
     a. metadata
     b. tarball with the files for the layer
  II. Manifesto
  III. Ancestry

- Layers have unique ids and can be shared by multiple images
- Layers decrease storage space and transfer time
  - e.g. the same OS layer can be shared by many services and applications, avoiding duplication and downloading
Docker

- **Common format to distribute and manage images:**
  - Layered file-system based
    - At the host level implemented by AUFS, device-mapper thin snapshots
    - New images can be easily created from existing ones
      - Created by using **Dockerfiles** and **docker build**

```
FROM centos:centos6
RUN yum install --y httpd php
COPY /my/app /var/www/app
EXPOSE 80
ENTRYPOINT /usr/sbin/httpd
CMD ["-D", "FOREGROUND"]
```

**Dockerfile**
Docker in numbers

- DockerCon conference 2017 (> 5500 attendees)

  - More than 14M Docker hosts
  - More than 900K Docker apps in repositories
  - 77,000% growth in Docker job listings
  - More than 12B image pulls (accounting for 390,000% growth)
  - More than 3,300 contributors
  - More than 280 cities hold Docker meetups, which accounts for more than 170K members worldwide

- Large ecosystem of tools and frameworks
Scientific Computing and containers

Is this thing for me
Scientific computing and containers

Running applications still requires considerable effort

Need a consistent portable way of running applications
but ...
Limitations
Containers in general ...

Wizard with root powers

- pid namespace
- mount namespace
- ipc namespace
- seccomp
- user namespace
- net namespace

Container

cgroup
apparmor
selinux
uts namespaces
posix capabilities
udocker
Cloud PaaS easy execution across systems cloud, grid, etc

Linux Containers
INDIGO-DataCloud containers for batch

• How to run Docker in batch systems?
  • Can we run Docker in batch system?
  • If so how to integrate it with the batch system?
  • How to make it respect batch system policies?
  • How to make it respect batch system actions?
  • How to collect accounting?

• How to run containers without Docker?
  • Can we download container images?
  • Can we run without a layered filesystem?
  • Can we run them as normal user?
  • Can we enforce container metadata?
udocker

- Run applications encapsulated in docker containers:
  - without using docker
  - without using privileges
  - without system administrators intervention
  - without additional system software

- and run:
  - as a normal user
  - with the normal process controls and accounting
  - in interactive or batch systems
INDIGO-DataCloud udocker

udocker in open source

https://github.com/indigo-dc/udocker
  • https://github.com/indigo-dc/udocker/tree/master
  • https://github.com/indigo-dc/udocker/tree/devel

https://github.com/indigo-dc/udocker/tree/master/doc
udocker: install from github

$ curl https://raw.githubusercontent.com/indigo-dc/udocker/master/udocker.py > udocker

$ chmod u+rx udocker

$ ./udocker install

or devel

Does not require compilation or system installation
Tools are delivered statically compiled
udocker: pull images from repository

$ udocker pull ubuntu:14.04

Search for names and tags at: https://hub.docker.com/
udocker: list local images

$ udocker images

<table>
<thead>
<tr>
<th>REPOSITORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoffice:latest</td>
</tr>
<tr>
<td>iscampos/openqcd:latest</td>
</tr>
<tr>
<td>fedora:25</td>
</tr>
<tr>
<td>docker.io/susymastercode/mastercode:latest</td>
</tr>
<tr>
<td><strong>ubuntu:14.04</strong></td>
</tr>
<tr>
<td>ubuntu:16.10</td>
</tr>
<tr>
<td>ubuntu:latest</td>
</tr>
<tr>
<td>indigodatacloud/disvis:latest</td>
</tr>
<tr>
<td>jorge/private:latest</td>
</tr>
<tr>
<td>busybox:latest</td>
</tr>
<tr>
<td>jorge_fedora22_32bit:latest</td>
</tr>
<tr>
<td>debian:oldstable</td>
</tr>
</tbody>
</table>
udocker: create container from image

$ udocker create --name=ub14 ubuntu:14.04

container-alias

9fe2f9e7-ce37-3be5-b12d-829a3236d2a6

container-id
udocker: list containers

$ udocker ps

<table>
<thead>
<tr>
<th>CONTAINER ID</th>
<th>NAMES</th>
<th>IMAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>9fe2f9e7-ce37-3be5-b12d-829a3236d2a6</td>
<td>['ub14']</td>
<td>ubuntu:14.04</td>
</tr>
<tr>
<td>5c7bd29b-7ab3-3d73-95f9-4438443aa6d6</td>
<td>['myoffice']</td>
<td>msoffice:lastest</td>
</tr>
<tr>
<td>676eb77d-335e-3e9a-bf62-54ad08330b99</td>
<td>['fedora_25']</td>
<td>fedora:25</td>
</tr>
<tr>
<td>c64afe05-adfa-39de-bf15-dcd45f284249</td>
<td>['debianold']</td>
<td>debian:oldstable</td>
</tr>
<tr>
<td>7e76a4d7-d27e-3f09-a836-abb4ded0df34</td>
<td>['ubuntu16', 'S']</td>
<td>ubuntu:16.10</td>
</tr>
<tr>
<td>9d12f52d-f0eb-34ae-9f0e-412b1f8f2639</td>
<td>['f25']</td>
<td>fedora:25</td>
</tr>
</tbody>
</table>
udocker: run container

$ udocker run ub14

udocker respects container metadata, if the container has a default cmd to run it will be run otherwise starts a shell

executing: bash
root@nbjorge:/# cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu 14.04.5 LTS"
root@nbjorge:/# apt-get install firefox

root emulation
udocker: run container as yourself

$$
udocker \text{ run } --\text{user=jorge} \ -v \ /home/jorge \ \\
-e \text{HOME=/jorge/home} \ --\text{workdir=/home/jorge} \ \\
ub14
$$

Warning: non-existing user will be created

executing: bash
jorge@nbjorge:~$$ \text{id}
uid=1000(jorge) gid=1000(jorge) groups=1000(jorge),10(uucp)
jorge@nbjorge:~$$ \text{pwd}
/home/jorge
jorge@nbjorge:~$$
udocker

How does it work ...
udocker:

- Implemented
  - python, C, C++, go

- Can run:
  - CentOS 6, CentOS 7, Fedora >= 23
  - Ubuntu 14.04, Ubuntu 16.04
  - Any distro that supports python 2.7

- Components:
  - Command line interface docker like
  - Pull of containers from Docker Hub
  - Local repository of images and containers
  - Execution of containers with modular engines
udocker:

- Containers
  - Are produced from the layers by flattening them
  - Each layer is extracted on top of the previous
  - Whiteouts are respected, protections are changed
  - The obtained directory trees are stored under `~/.udocker/containers` in the user home directory
udocker: directories and execution

- Execution
- chroot-like

udocker directory tree
$HOME/.udocker

chroot to this directory becomes the new root for container processes

container tree in udocker
udocker: Execution methods

- udocker supports several techniques to achieve the equivalent to a chroot without using privileges
  - They are selected per container id via execution modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Base</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>PRoot</td>
<td>PTRACE accelerated (with SECCOMP filtering) ← DEFAULT</td>
</tr>
<tr>
<td>P2</td>
<td>PRoot</td>
<td>PTRACE non-accelerated (without SECCOMP filtering)</td>
</tr>
<tr>
<td>R1</td>
<td>runC</td>
<td>rootless unprivileged using user namespaces</td>
</tr>
<tr>
<td>F1</td>
<td>Fakechroot</td>
<td>with loader as argument and LD_LIBRARY_PATH</td>
</tr>
<tr>
<td>F2</td>
<td>Fakechroot</td>
<td>with modified loader, loader as argument and LD_LIBRARY_PATH</td>
</tr>
<tr>
<td>F3</td>
<td>Fakechroot</td>
<td>modified loader and ELF headers of binaries + libs changed</td>
</tr>
<tr>
<td>F4</td>
<td>Fakechroot</td>
<td>modified loader and ELF headers dynamically changed</td>
</tr>
<tr>
<td>S1</td>
<td>Singularity</td>
<td>where locally installed using chroot or user namespaces</td>
</tr>
</tbody>
</table>
udocker & Lattice QCD

Scaling performance as a function of the cores for the computation of application of the Dirac operator to a spinor field.

OpenQCD is a very advanced code to run lattice simulations.

Using OpenMPI

udocker in P1 mode
udocker & Biomolecular complexes

Better performance with Ubuntu 16 container

DisVis is being used in production with udocker

Performance with docker and udocker are the same and very similar to the host.

Using OpenCL and NVIDIA GPGPUs

udocker in P1 mode
Gromacs is widely used both in biochemical and non-biochemical systems.

udocker P mode have lower performance, udocker F mode same as Docker.

Using OpenCL and OpenMP

udocker in P1 mode
udocker in F3 mode

udocker & Molecular dynamics

udocker in P1 mode
udocker in F3 mode

PTRACE

SHARED LIB CALL
udocker & Phenomenology

Performance Degradation

<table>
<thead>
<tr>
<th></th>
<th>Compiling</th>
<th>Running</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>DOCKER</td>
<td>10%</td>
<td>1.0%</td>
</tr>
<tr>
<td>udocker</td>
<td>7%</td>
<td>1.3%</td>
</tr>
<tr>
<td>VirtualBox</td>
<td>15%</td>
<td>1.6%</td>
</tr>
<tr>
<td>KVM</td>
<td>5%</td>
<td>2.6%</td>
</tr>
</tbody>
</table>

udocker in P1 mode

MasterCode connects several complex codes. Hard to deploy.

Scanning through large parameter spaces. High Throughput Computing

C++, Fortran, many authors, legacy code

Jorge Gomes
udocker & Phenomenology

```bash
export MASTERDIR=/gpfs/csic_users/userabc/mastercode
export UDOCKER_DIR=$MASTERDIR/.udocker

udocker.py run --hostauth
  -v /home/csic/cdi/ica/mcpp-master
  -v /home/csic/cdi/ica
  -user=user001
  -w /home/csic/cdi/ica/mcpp-master mastercode
/bin/bash -c "pwd; ./udocker-mastercode.sh"
```
Scientific computing and containers
Thank you

https://github.com/indigo-dc/udocker