Operating Systems

Virtualization

Carlos Alberto Llano R.

9 de noviembre de 2015

Contents

Overview

The fundamental idea behind a virtual machine is to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each separate environment is running on its own private computer.

Overview - Components

host it's a physical machine.

hypervisor or Virtual Machine Manager, it's runs the virtual machines.

guest it's the virtual machine.

Overview - The implementation of VMMs

- **Hardware-based solutions via firmware**, these are generally known as type 0 hypervisors.
- **Operating-system-like software built to provide virtualization**: VMware ESX(mentioned above), Joyent SmartOS, and Citrix XenServer, these are known as type 1 hypervisors.
- **General-purpose operating systems**: Microsoft Windows Server with HyperV and RedHat Linux with the KVM feature.

Overview - The implementation of VMMs

- **Applications that run on standard operating systems**: VMware Workstation and Fusion, Parallels Desktop, and Oracle VirtualBox, are type 2 hypervisors.
- **Paravirtualization**, a technique in which the guest operating system is modified to work in cooperation with the VMM to optimize performance.
- **Programming-environment virtualization**, in which VMMs do not virtualize real hardware but instead create an optimized virtual system: Oracle Java and Microsoft.Net.

Overview - The implementation of VMMs

- **Emulators** that allow applications written for one hardware environment to run on a very different hardware environment, such as a different type of CPU.
- **Operating System Level Virtualization or containers**. Under this system, there is only one kernel installed - the host kernel. Each container is simply an isolation of the userland processes.

History

Virtual machines first appeared commercially on IBM mainframes in 1972.

Virtualization was provided by the IBM VM operating system.

A major difficulty with the VM approach involved disk systems. The solution was to provide virtual disks—termed minidisks in IBM's VM operating system.

Benefits and Features

- Ability to share the same hardware yet run several different execution environments concurrently.
- The host system is protected from the virtual machines, just as the virtual machines are protected from each other.
- Energy costs.
- Increases Business Agility.
- Increases IT Operational Flexibility.
- Reduces IT Operations Costs.
- High Availability.
- Disaster Recovery.
- Is Green.

Benefits and Features

- Suspend
- Snapshots
- Resume
- Clone
- Templating
- Live migration

Open Virtual Machine Format

The DMTF's Open Virtualization Format (OVF) standard provides the industry with a standard packaging format for software solutions based on virtual systems, solving critical business needs for software vendors and cloud computing service providers. OVF has been adopted and published by the International Organization for Standardization (ISO) as ISO 17203.

Cloud Computing

is made possible by virtualization in which resources such as CPU, memory, and I/O are provided as services to customers using Internet technologies. By using APIs, a program can tell a cloud computing facility to create thousands of VMs, all running a specific guest operating system and application, which others can access via the Internet.

Virtual CPU - VCPU

The VCPU does not execute code. Rather, it represents the state of the CPU as the guest machine believes it to be.

When the guest is context-switched onto a CPU by the VMM, information from the VCPU is used to load the right context, much as a general-purpose operating system would use the PCB.

Requirements for Virtualization

Safety : the hypervisor should have full control of the virtualized resources.

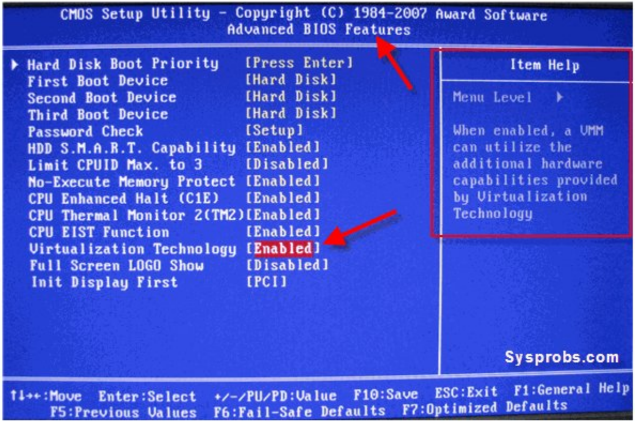
Fidelity : the behavior of a program on a virtual machine should be identical to that of the same program running on bare hardware.

Efficiency : much of the code in the virtual machine should run without intervention by the hypervisor.

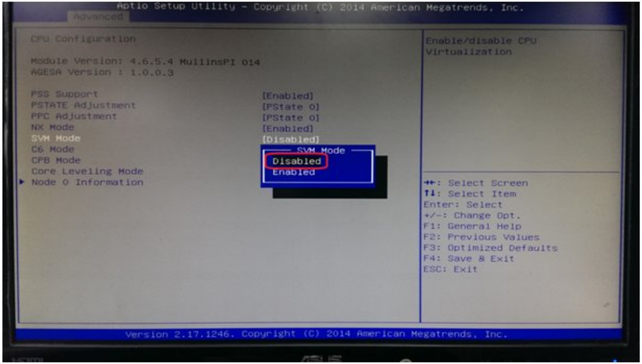
Virtualization Technology - Intel and AMD

On the Intel CPUs it is called VT (Virtualization Technology); on the AMD CPUs it is called SVM (Secure Virtual Machine).

Both were inspired by the IBM VM/370 work.



Virtualization Technology - Intel and AMD



cat /proc/cpuinfo | egrep '(vmx|svm)'

Comparison of platform virtualization software

https://en.wikipedia.org/wiki/Comparison_of_platform_virtualization_software

VirtualBox

VirtualBox allows you to run practically any operating system right inside your current OS.

Installation:
https://help.ubuntu.com/community/VirtualBox/Installation
Ubuntu iso server: http://www.ubuntu.com/download/server

VirtualBox - Network types

Network Address Translation (NAT) means the virtual machines will have private IP addresses that are not routable from outside.

Example: Your host is 192.168.1.1. The VirtualBox NAT device will be marked as 10.0.2.1. Therefore, the virtual machines will be given any address in the 10.0.2.x range.

VirtualBox - Network types

Bridge Adapter means that any virtual machine running will try to obtain an IP address from the same source your currently active, default network address got its IP address.

Example: Your host has leased an address of 192.168.1.100 from the router. The virtual machine leases an address of 192.168.1.103 from the router. The two machines now share the same network and all standard rules apply. For all practical purposes, the virtual machine is another IP address on your LAN.

VirtualBox - Network types

Host-only Adapter It's very similar to Bridged Adapter, except that is uses a dedicated network device, called vboxnet0, to lease IP addresses.

Example: Your host has the IP address of 192.168.56.1. Your virtual machine has the IP address of 192.168.56.101.

VMware Server has its two virtual adapters called vmnet1 and vmnet8, which are used assign NAT and host-only IP addresses to guests.

VirtualBox - Network types

Internal network It's similar to Host-only + NAT, except the networking takes place inside the virtual network of guest machines, without any access for the host, plus there is no real NAT. What you get is a private LAN for your guests only, without any access to the external world.

VirtualBox - Ubuntu interfaces

```
ifconfig —a

In /etc/network/interfaces file:

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface NAT
auto eth0
iface eth0 inet dhcp

# The seconday network interface  HOST—ONLY ADAPTER
auto eth1
iface eth1 inet static
address 192.168.56.103      <——————————————————— Ip of range of adapter
netmask 255.255.255.0
```

VirtualBox - Exercise

Install two ubuntu servers and to configurate the network.

In any machine create a snapshot after install apache, after retore the snapshot.

VirtualBox - Ubuntu interfaces

List all vms
============
VBoxManage list vms
VBoxManage list ostypes

List all properties
===================
VBoxManage guestproperty enumerate testMachine

Create a new machine
====================
VBoxManage createvm ——name testMachine ——ostype Ubuntu_64 ——**register**
Output:
Virtual machine 'testMachine' is created and registered.
UUID: 8f368cc1—7f9e—4378—a0c3—1f84dffe87c8
Settings file: '/home/callanor/VirtualBox VMs/testMachine/testMachine.vbox'

Checking existing Virtual machine
=================================
VBoxManage showvminfo testMachine

Change memory
=============
VBoxManage modifyvm testMachine ——memory 1024

VirtualBox - Ubuntu interfaces

Set cores
=========
VBoxManage modifyvm testMachine ——cpus 1 ——ioapic on

Create a bridge adapter
========================
VBoxManage modifyvm testMachine ——bridgeadapter1 eth0
VBoxManage modifyvm testMachine ——nic1 bridged

Create an HDD and attach
=========================
VBoxManage createhd ——filename testMachine.vdi ——size 18000 ——format VDI
VBoxManage storagectl testMachine ——name "SATA Controller" ——add sata ——controller IntelAhci
VBoxManage storageattach "testMachine" ——storagectl "SATA Controller" ——port 0 ——device 0 ——type hdd ——medium testMachine.vdi
VBoxManage storagectl testMachine ——name "IDE Controller" ——add ide ——controller PIIX4
VBoxManage storageattach testMachine ——storagectl "IDE Controller" ——port 1 ——device 0 ——type dvddrive ——medium /tmp/ubuntu.iso

Attach VBoxGuestAdditions
==========================
VBoxManage storageattach testMachine ——storagectl "IDE Controller" ——port 1 ——device 0 ——type dvddrive ——medium /usr/share/virtualbox/
VBoxGuestAdditions.iso
VBoxManage modifyvm $vm ——dvd /usr/share/virtualbox/VBoxGuestAdditions.iso

Vagrant

Vagrant makes it really easy to work with virtual machines. According to the Vagrant docs.

Installation: http://www.vagrantup.com/downloads

```
vagrant box add precise32 http://files.vagrantup.com/precise32.box

mkdir prueba

cd prueba

vagrant init precise32

vagrant up --provider virtualbox

vagrant ssh
```

http://www.vagrantbox.es/

Vagrant - Shell provisioning

```
mkdir provisioning
cd provisioning
touch setup.sh

edit setup.sh and put this content:

echo "Installing Git"
sudo apt-get install git -y

echo "Installing Apache"
sudo apt-get install -y apache2
sudo service apache2 restart

echo "You've been provisioned"

and save file.
```

Vagrant - Shell provisioning

```
edit VagrantFile:
Put this line  (wlan0 or eth0 depends of the network):
        config.vm.network :public_network, :public_network => "wlan0"

and put this too:
        config.vm.provider "virtualbox" do |vb|
                # Don’t boot with headless mode
                vb.gui = true
                v.name = "my_vm"

                # Use VBoxManage to customize the VM.
                # For example to change memory:
                vb.customize ["modifyvm", :id, "--memory", "1024"]
        end

        config.vm.provision "shell" do |s|
            s.path = "provisioning/setup.sh"
        end

save file.

vagrant up (and wait)
```

Exercise with Vagrant and Virtualbox

`https://www.leaseweb.com/labs/2011/07/`
`high-availability-load-balancing-using-haproxy-on-ubuntu-p`

Is the same:

`https://www.howtoforge.com/tutorial/`
`ubuntu-load-balancer-haproxy/`

Vagrant customizations

```
config.vm.provider "virtualbox" do |v|
  v.customize ["modifyvm", :id, "--cpuexecutioncap",
    "50"]
  v.memory = 1024
  v.cpus = 2
  v.name = "my_vm_name"
end
```

The VM is modified to have a host CPU execution cap of 50 %, meaning that no matter how much CPU is used in the VM, no more than 50 % would be used on your own host machine.

Vagrant multiple machines

```
$ mkdir vagrant_multi_machine
$ cd vagrant_multi_machine
$ vagrant init precise32
$ Updated the Vagrantfile with this:
# Adding Bridged Network Adapter
        config.vm.network "public_network"
        # Iterating the loop for three times
        (1..3).each do |i|
                # Defining VM properties
                config.vm.define "machine_vm#{i}" do |node|
                        # Specifying the provider as VirtualBox and
    naming the VM's
                        config.vm.provider "virtualbox" do |node|
                                # The VM will be named as edureka_vm{i}
                                node.name = "machine_vm#{i}"
                        end
                end
        end
```

Vagrant destroy machine

vagrant destroy

-f or –force

Vagrant box

```
vagrant box add precise32
    http://files.vagrantup.com/precise32.box

vagrant box list
vagrant box remove NAME
vagrant box update --box VALUE
vagrant box outdated
```

SSH - Secure SHell

It was designed and created to provide the best security when accessing another computer remotely.

```
Connect to a remote machine
===========================
ssh username@remotehost

Note: it will ask you if you wish to add the remote host to a list of
     known_hosts, go ahead and say yes.

Running Commands Over SSH
=========================
ssh [USER-NAME]@[REMOTE-HOST] [command or script]

Examples:
ssh username@remotehost ls -l
ssh username@remotehost 'df -H'
```

Containers

Containers have a long and storied history in computing. Unlike hypervisor virtualization, where one or more independent machines run virtually on physical hardware via an intermediation layer, containers instead run user space on top of an operating system's kernel.

As a result, container virtualization is often called operating system-level virtualization.

Container technology allows multiple isolated user space instances to be run on a single host.

Containers

As a result of their status as guests of the operating system, containers are sometimes seen as less flexible: they can generally only run the same or a similar guest operating system as the underlying host.

For example, you can run Red Hat Enterprise Linux on an Ubuntu server, but you can't run Microsoft Windows on top of an Ubuntu server.

Containers

Containers have also been seen as less secure than the full isolation of hypervisor virtualization.

Despite these limitations, containers have been deployed in a variety of use cases.

They are popular for hyperscale deployments of multi-tenant services, for lightweight sandboxing, and, despite concerns about their security, as process isolation environments.

(Multi-tenancy)

Multi-tenancy is an architecture in which a single instance of a software application serves multiple customers. Each customer is called a tenant. Tenants may be given the ability to customize some parts of the application, such as color of the user interface (UI) or business rules, but they cannot customize the application's code.

Single-tenancy, an architecture in which each customer has their own software instance and may be given access to code.

With a multi-tenancy architecture, the provider only has to make updates once.

Containers - lxc

Linux Containers (LXC) provide a Free Software virtualization system for computers running GNU/Linux. This is accomplished through kernel level isolation. It allows one to run multiple virtual units simultaneously. Those units, similar to chroots, are sufficiently isolated to guarantee the required security, but utilize available resources efficiently, as they run on the same kernel.

Containers - lxc - components

- The liblxc library
- Several language bindings for the API:
  - python3 (in-tree, long term support in 1.0.x)
  - lua (in tree, long term support in 1.0.x)
  - Go
  - ruby
  - python2
  - Haskell
- A set of standard tools to control the containers
- Distribution container templates

Containers - lxc - installation and commands

sudo apt-get install lxc lxc-templates

sudo lxc-checkconfig

View all lxc containers:

sudo lxc-ls –fancy

Create one lxc container:

lxc-create -t <template> -n <container name>
Example:

lxc-create -t ubuntu -n cn-01

Containers - lxc - installation and commands

Start one lxc container:

lxc-start -d -n <container name>
Example:

sudo lxc-start -d -n cn-01
sudo lxc-ls –fancy
ssh ubuntu@ip the password is ubuntu

Containers - lxc - installation and commands

Stop one lxc container:

lxc-stop -d -n <container name>
Example:

sudo lxc-stop -n cn-01

Clone one lxc container:

lxc-clone -o <existing container> -n <new container name>
Example:

sudo lxc-clone -o cn-01 cn-02

Containers - lxc - templates

```
$ ls /usr/share/lxc/templates
```

lxc-alpine lxc-archlinux lxc-centos lxc-debian lxc-fedora lxc-openmandriva lxc-oracle lxc-sshd lxc-ubuntu-cloud lxc-altlinux lxc-busybox lxc-cirros lxc-download lxc-gentoo lxc-opensuse lxc-plamo lxc-ubuntu

Containers - docker

Docker is an open-source engine that automates the deployment of applications into containers. It was written by the team at Docker, Inc (formerly dotCloud Inc, an early player in the Platform-as-a-Service (PAAS) market), and released by them under the Apache 2.0 license.

Containers - docker

Docker recommends that each container run a single application or process. This promotes a distributed application model where an application or service is represents by a series of inter-connected containers. This makes it very easy to distribute, scale, debug and introspect your applications.

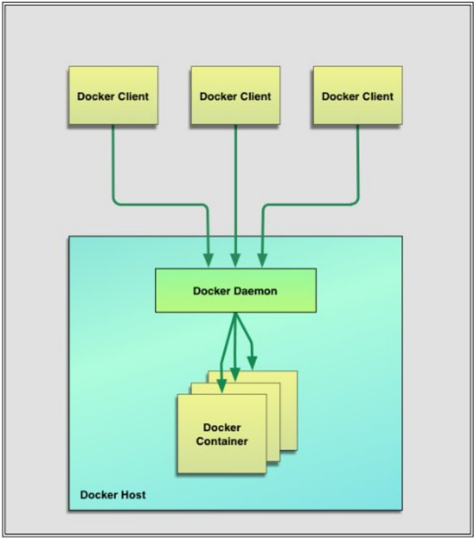
Containers - docker

Docker runs on a number of other platforms, including Debian, SuSE, Arch Linux, CentOS, and Gentoo. It's also supported on several Cloud platforms including Amazon EC2, Rackspace Cloud, and Google Compute Engine.

Containers - docker - components

- The Docker client and server
- Docker Images
- Registries
- Docker Containers

Containers - docker - architecture



Containers - docker - technical components

Docker can be run on any x64 host running a modern Linux kernel (kernel version 3.8 and later).

- libcontainer
- Linux kernel namespaces (isolation for filesystems, processes and networks).
- Resource isolation and grouping: (using the cgroups).
- Copy-on-write: (require limited disk usage).
- Logging: STDOUT, STDERR and STDIN from the container are collected.
- Interactive shell.

Containers - docker - installing on Ubuntu

`https://docs.docker.com/installation/ubuntulinux/`

$ **sudo docker info**

Containers - docker - installing on OSX

To install Boot2Docker on OSX we need to download its installer from Git Hub.

Current release (Downloading the Boot2Docker PKG file):

wget https://github.com/boot2docker/osx-installer/releases/download/v1.1.1/Boot2Docker-1.1.1.pkg

Launch the downloaded installer and follow the instructions to install Boot2Docker.

Containers - docker - installing on Windows

To install Boot2Docker on Windows we need to download its installer from Git Hub.

Current release (Downloading the Boot2Docker PKG file):

wget https://github.com/boot2docker/windows-installer/releases/download/v1.1.1/docker-install.exe

Launch the downloaded installer and follow the instructions to install Boot2Docker.

Containers - docker - creating our first container

sudo docker run -i -t ubuntu /bin/bash

Docker will automatically generate a name at random for each container we create.

sudo docker run –name bob_the_container -i -t ubuntu /bin/bash

sudo docker start bob_the_container

sudo docker stop bob_the_container

Containers - docker - commands

List containers:
sudo docker ps -a or sudo docker ps -l

Deleting a container:
sudo docker rm ID

Attaching to a container:
sudo docker attach bob_the_container

Containers - docker - Daemonized containers

sudo docker run –name daemon_dave -d ubuntu /bin/sh -c "while true; do echo hello world; sleep 1; done"

sudo docker logs -ft daemon_dave

sudo docker top daemon_dave

Containers - docker - inspect

sudo docker inspect daemon_dave

sudo docker inspect –format=' .State.Running ' daemon_dave

sudo docker inspect –format='.NetworkSettings.IPAddress' $INSTANCE_ID

docker inspect –format='.NetworkSettings.MacAddress' $INSTANCE_ID

sudo docker inspect –format='range $p, $conf := .NetworkSettings.Ports $p -> (index $conf 0).HostPort end' $INSTANCE_ID

Containers - docker - Dockerfile

create one file called Dockerfile with this content:

```
FROM ubuntu:14.04
MAINTAINER Carlos Llano <carlos_llano@hotmail.com>

RUN apt-get update && apt-get install -y openssh-server
RUN mkdir /var/run/sshd
RUN echo 'root:screencast' | chpasswd
RUN sed -i 's/PermitRootLogin without-password/PermitRootLogin yes/'
    /etc/ssh/sshd_config

# SSH login fix. Otherwise user is kicked off after login
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional
    pam_loginuid.so@g' -i /etc/pam.d/sshd

ENV NOTVISIBLE "in users profile"
RUN echo "export VISIBLE=now" >> /etc/profile

EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

Containers - docker - Dockerfile

Build the image:
sudo docker build -t eg_sshd .

Run a test_sshd container:
sudo docker run -d -P –name test_sshd eg_sshd
sudo docker port test_sshd 22
0.0.0.0:49154

And now you can ssh as root on the containerś IP address (you can find it with docker inspect) or on port 49154:
ssh root@192.168.1.2 -p 49154
The password is screencast.

Containers - docker - Dockerfile

create one file with this content:

```
# Version: 0.0.1
FROM ubuntu:14.04
MAINTAINER Carlos Llano "carlos_llano@hotmail.com"
RUN apt-get update
RUN apt-get install -y nginx
RUN echo 'Hi, I am in your container' \
>/usr/share/nginx/html/index.html
EXPOSE 80
```

Containers - docker - Dockerfile

Build the image (sudo docker images):

sudo docker build -t my-nginx .

—> Running in d794cc94a2cd

—> fe9bd24b1e5c

Removing intermediate container d794cc94a2cd

Step 5 : EXPOSE 80

—> Running in 6b6bf39279fa

—> 6438b9941e95

Removing intermediate container 6b6bf39279fa

Successfully built 6438b9941e95

sudo docker run -t -i 6438b9941e95 /bin/bash

root@735cacf8eaba:/# exit

sudo docker images

Containers - docker - Dockerfile

Launching a container from our new image:

sudo docker run -d -p 127.0.0.1:80:80 –name static_web my-nginx nginx -g "daemon off;"

sudo docker run -d -p 80 –name static_web my-nginx nginx -g "daemon off;"

sudo docker port dc62e37e033a0
80/tcp -> 0.0.0.0:32773

curl localhost:32773

Containers - docker - images

Listing Docker images:
sudo docker images

Public repository:
`https://hub.docker.com/`

docker search sinatra
docker search centos

Getting a new image:
docker pull centos
docker pull ubuntu
docker run -t -i centos /bin/bash

Containers - docker - images

Creating our own images:
docker run -t -i training/sinatra /bin/bash

Inside our running container let's add the mc program:
sudo apt-get install mc
Once this has completed let's exit our container using the exit command.
docker commit -m .ᴬdded mc programa "Kate Smith" INSTANCE_ID ouruser/sinatra:v2

sudo docker images
To use our new image to create a container you can then:
docker run -t -i ouruser/sinatra:v2 /bin/bash

Containers - docker - images

Setting tags on an images:
docker tag 5db5f8471261 ouruser/sinatra:devel

Push an image to Docker Hub:
docker push ouruser/sinatra

Remove an image from the host: **docker rmi training/sinatra**

Containers - kernel features to contain processes

- Kernel namespaces (ipc, uts, mount, pid, network and user)
- Apparmor and SELinux profiles
- Seccomp policies
- Chroots (using pivot_root)
- Kernel capabilities
- CGroups (control groups)

Containers - chroot

chroot is an operation that changes the apparent root directory for the current running process and their children. A program that is run in such a modified environment cannot access files and commands outside that environmental directory tree. This modified environment is called a chroot jail.

- Privilege separation for unprivileged process such as Web-server or DNS server.
- Setting up a test environment.
- Run old programs or ABI in-compatibility programs without crashing application or system.

Containers - chroot - example

Build a mini-jail for testing purpose with bash and ls command only. First, set jail location using mkdir command:

```
$ J=$HOME/jail

Create directories inside $J:
$ mkdir -p $J
$ mkdir -p $J/{bin,lib64,lib}
$ cd $J

Copy /bin/bash and /bin/ls into $J/bin/ location using cp command:
$ cp -v /bin/{bash,ls} $J/bin
```

Containers - chroot - example

```
Copy required libs in $J. Use ldd command to print shared library
     dependencies for bash:

$ ldd /bin/bash

        linux-vdso.so.1 =>  (0x00007fff8d987000)
        libtinfo.so.5 => /lib64/libtinfo.so.5 (0x00000032f7a00000)
        libdl.so.2 => /lib64/libdl.so.2 (0x00000032f6e00000)
        libc.so.6 => /lib64/libc.so.6 (0x00000032f7200000)
        /lib64/ld-linux-x86-64.so.2 (0x00000032f6a00000)

Copy libs in $J correctly from the above output:
$ cp -v /lib64/libtinfo.so.5 /lib64/libdl.so.2 /lib64/libc.so.6
     /lib64/ld-linux-x86-64.so.2 $J/lib64/
```

Containers - chroot - example

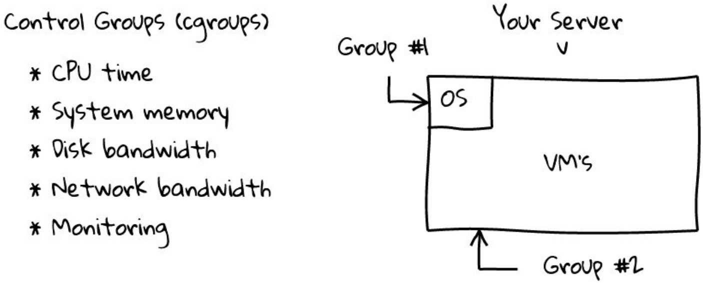
```
Copy required libs in $J for ls command. Use ldd command to print
     shared library dependencies for ls command:
$ ldd /bin/ls

You can copy libs one-by-one or try bash shell for loop as follows:

list="$(ldd /bin/ls | egrep -o '/lib.*\.[0-9]')"
for i in $list; do cp  -v "$i" "${J}${i}"; done

Finally, chroot into your new jail:
$ sudo chroot $J /bin/bash

Try browsing /etc or /var:
# ls /
# ls /etc/
# ls /var/

How do I exit from chrooted jail?
# exit
```

Containers - cgroups (control groups)

Cgroups provide a mechanism for easily managing and monitoring system resources, by partitioning things like cpu time, system memory, disk and network bandwidth, into groups, then assigning tasks to those groups.

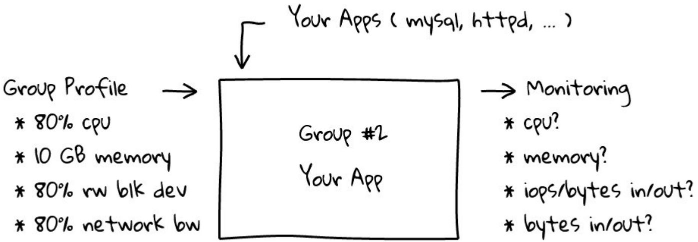
Linux is great at sharing resources between all of the processes on a system, but in some cases, you want to allocate, or guarantee, a greater amount to a specific application, or a set of applications, this is where control groups are useful.

Containers - cgroups (control groups)



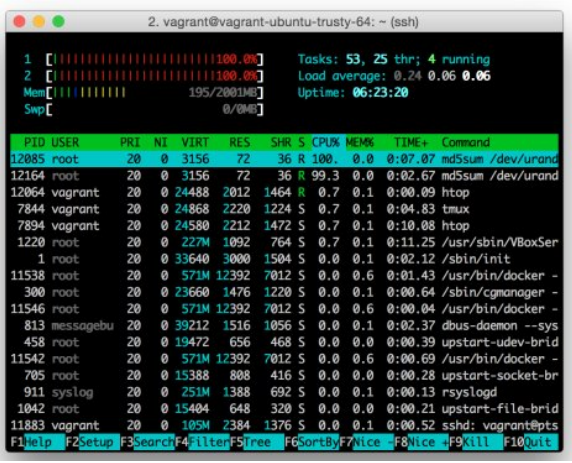
Containers - cgroups (control groups)

Once the group is created, you simple need to add your applications process ids, or pids, into a file, and your applications are automatically throttled.



Containers - cgroups (control groups)

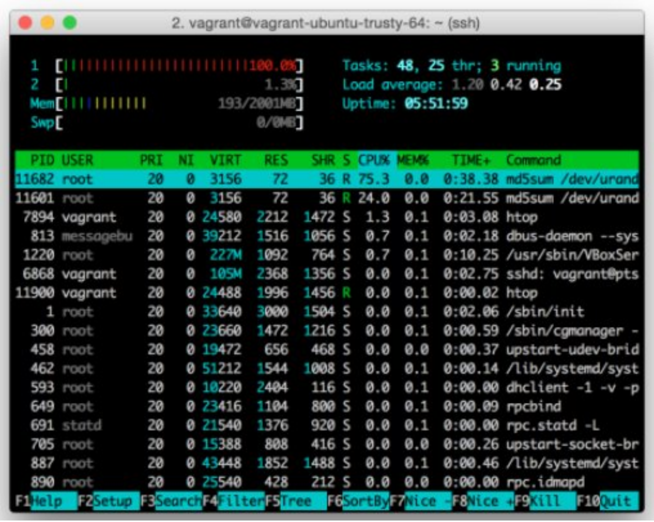
$ docker run -d busybox md5sum /dev/urandom
$ docker run -d busybox md5sum /dev/urandom



Containers - cgroups (control groups)

```
Start container with low priority:

$ docker run -d --name='low_prio' \
    --lxc-conf="lxc.cgroup.cpu.shares=250" \
    --lxc-conf="lxc.cgroup.cpuset.cpus=0" \
    busybox md5sum /dev/urandom

Start container with high priority:

$ docker run -d --name='high_prio' \
    --lxc-conf="lxc.cgroup.cpu.shares=750" \
    --lxc-conf="lxc.cgroup.cpuset.cpus=0" \
    busybox md5sum /dev/urandom
```

Containers - cgroups (control groups)



Containers - cgroups some controllers

blkio Sets limits on input/output access to and from block devices such as physical drives (disk, solid state, USB, etc.).

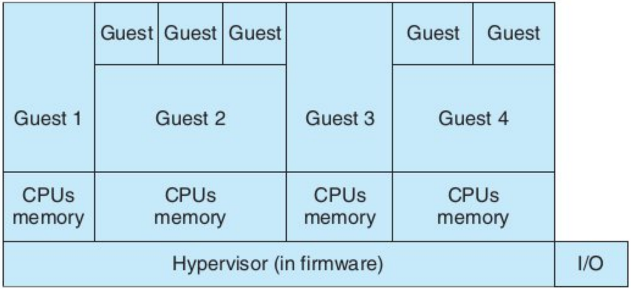
cpu Controlling the priorization of processes in the group.

memory Allows for setting limits on RAM and swap.

Type 0 and Type 1 Hypervisors

Goldberg (1972) distinguished between two approaches to virtualization.

- **type 0 hypervisor** Essentially, the guest operating systems in a type 0 hypervisor are native operating systems with a subset of hardware made available to them.



Type 0 and Type 1 Hypervisors

- **type 1 hypervisor** They are special-purpose operating systems that run natively on the hardware, but rather than providing system calls and other interfaces for running programs, they create, run, and manage guest operating systems.

Trap and Emulate

The virtual machine guest can execute only in user mode. The kernel, of course, runs in kernel mode, and it is not safe to allow user-level code to run in kernel mode.

Trap and Emulate

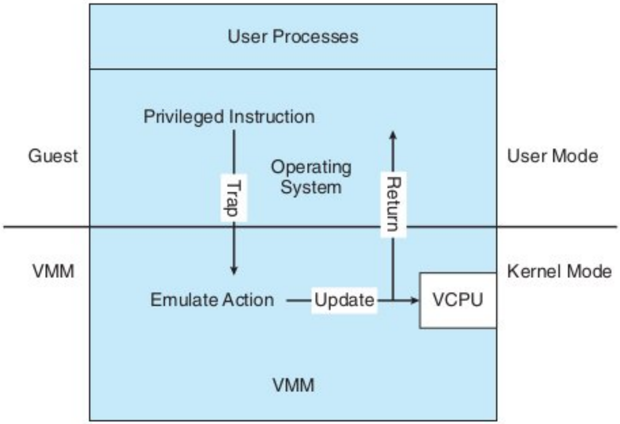
What happens when the guest operating system (which thinks it is in kernel mode) executes an instruction that is allowed only when the CPU really is in kernel mode?

Trap and Emulate

When the kernel in the guest attempts to execute a privileged instruction, that is an error (because the system is in user mode) and causes a trap to the VMM in the real machine.

The VMM gains control and executes (or “emulates”) the action that was attempted by the guest kernel on the part of the guest. It then returns control to the virtual machine.

Trap and Emulate



Binary Translation

Some CPUs do not have a clean separation of privileged and nonprivileged instructions.

Unfortunately for virtualization implementers, the Intel x86 CPU line is one of them.

Binary Translation

Some CPUs do not have a clean separation of privileged and nonprivileged instructions.

Unfortunately for virtualization implementers, the Intel x86 CPU line is one of them.

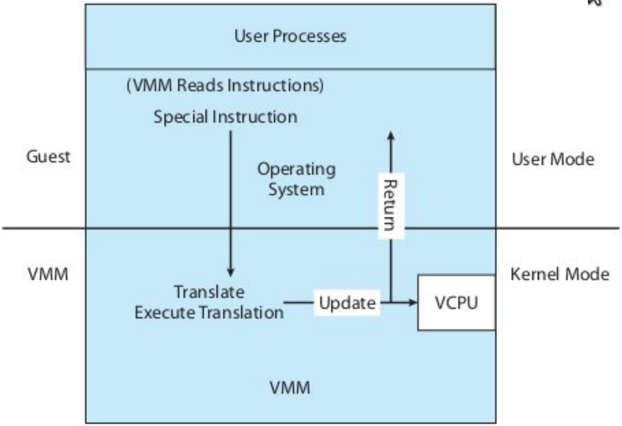
If the CPU is in privileged mode, the command's behaviour is diferent if the CPU is in user mode.

Binary Translation

Binary translation is fairly simple in concept but complex in implementation.

- If the guest VCPU is in user mode, the guest can run its instructions natively on a physical CPU.
- If the guest VCPU is in kernel mode, then the guest believes that it is running in kernel mode.
  - The VMM examines every instruction the guest executes in virtual kernel mode by reading the next few instructions that the guest is going to execute, based on the guest's program counter.
  - Instructions other than special instructions are run natively.
  - Special instructions are translated into a new set of instructions that perform the equivalent task.

Binary Translation



Memory Managment

nested page tables (NPTs): Each guest operating system maintains one or more page tables to translate from virtual to physical memory.

The VMM maintains NPTs to represent the guest's page-table state, just as it creates a VCPU to represent the guest's CPU state.

Memory Managment

The VMM knows when the guest tries to change its page table, and it makes the equivalent change in the NPT.

If the guest needs to modify the page table (for example, fulfilling a page fault), then that operation must be intercepted by the VMM and appropriate changes made to the nested and system page tables.