Computer Systems & Network Administration

Lecture 14. Linux Container

請上網填寫健康關懷問卷

Outline

- Virtual Machine
- Linux Container
  - Brief Introduction
  - Technologies
- Docker

Virtual Machine

Virtual Machine

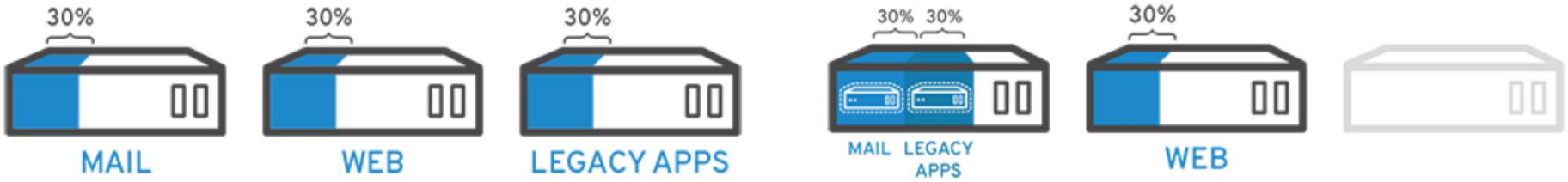
- A virtual machine (VM) is the virtualization/emulation of a computer system. Virtual machines are based on computer architectures and provide functionality of a physical computer. Their implementations may involve specialized hardware, software, or a combination.

Virtual Machine - Type

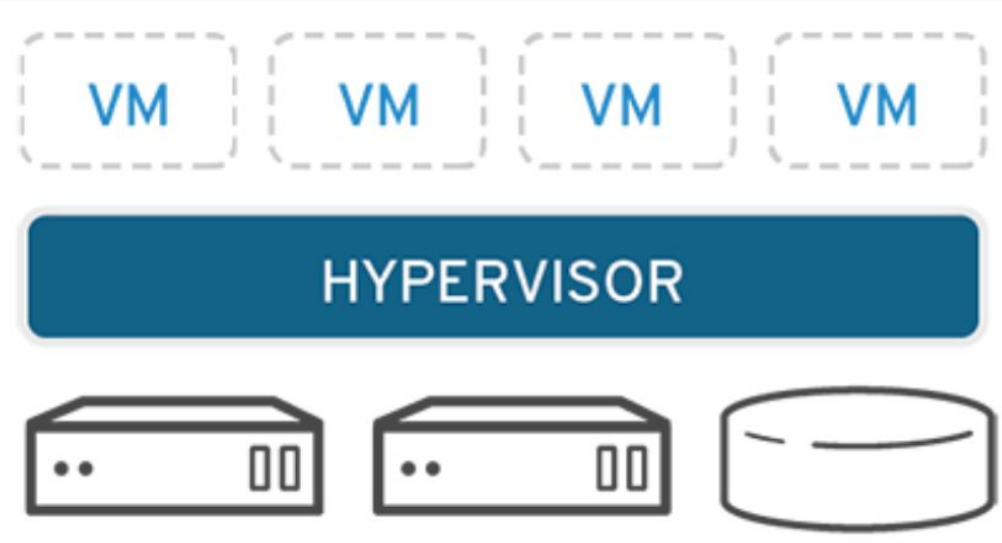
- System Virtual Machine
  - VMware Player / Oracle VirtualBox / QEMU
- Program Virtual Machine
  - Java VM / Python

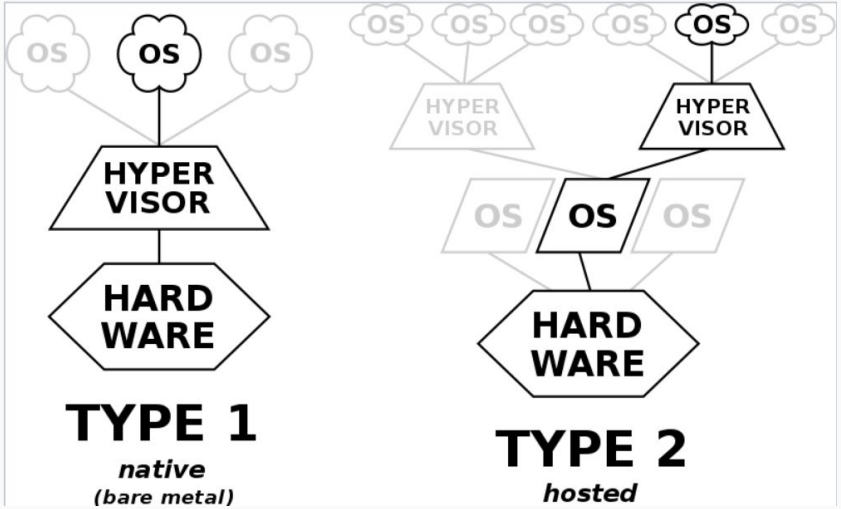
Virtual Machine - History

- Based on time-sharing and CTSS in 1960s
- Companies use one computer to run one program only
- The computer is underutilized
- Companies could partition their servers and run legacy apps on multiple operating system types and versions



Virtual Machine - Hypervisor



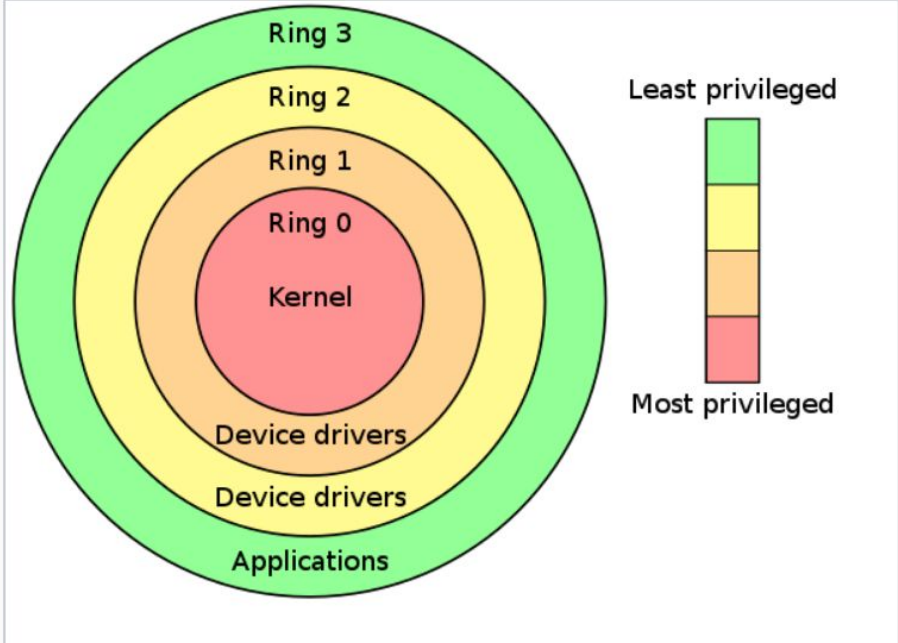


Virtual Machine - Type

- Full Virtualization
- Paravirtualization
- Hardware-assisted Virtualization

Virtual Machine - CPU ring

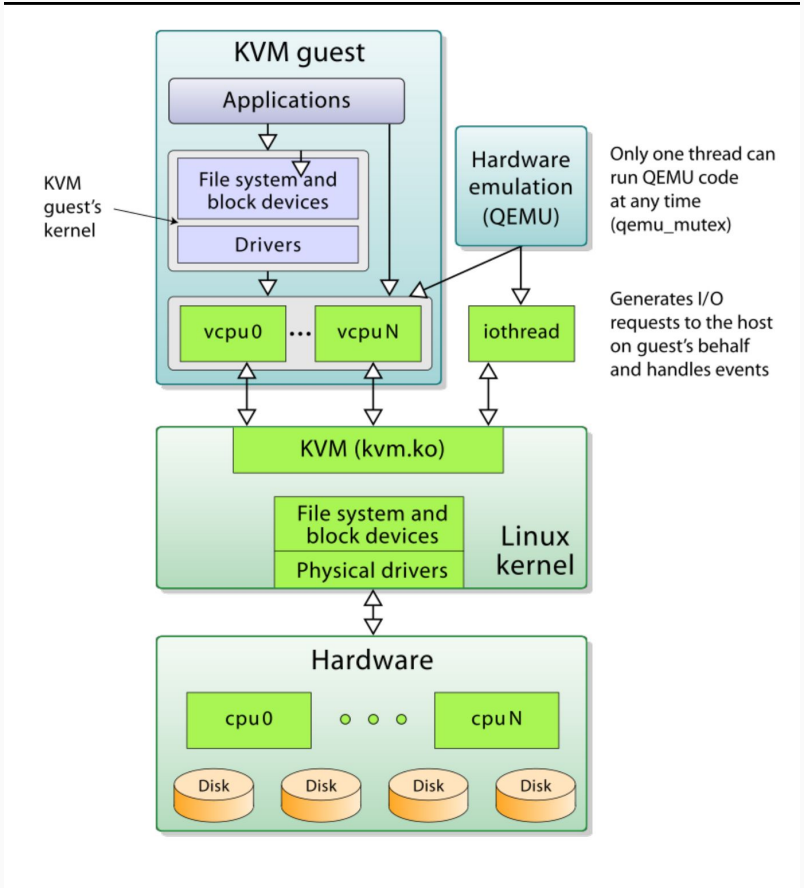
- Modern OS only implements Ring 3 and Ring 0
- For virtualizaiton, we have Ring -1
  - Guest OS can run Ring 0 natively without affecting other guests or host OS



Virtual Machine - QEMU

- Both emulator and virtualizer
- Emulator
  - Binary Translation
- Virtualizer
  - Using KVM (Kernel Virtual Machine)
  - Your VM is using this method

Virtual Machine - KVM



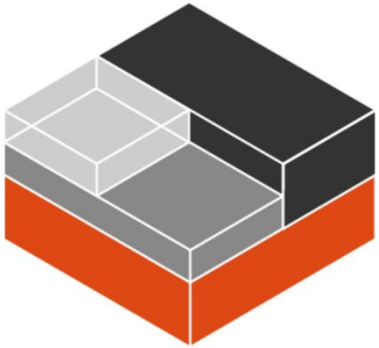
Linux Container

Linux Container? Docker?

Linux containers | Home | LXC | LXD | LXCFS | distrobuilder | CGManager | Language

Home

Infrastructure for container projects.



linuxcontainers.org is the umbrella project behind LXC, LXD and LXCFS.

The goal is to offer a distro and vendor neutral environment for the development of Linux container technologies.

Our main focus is system containers. That is, containers which offer an environment as close as possible as the one you'd get from a VM but without the overhead that comes with running a separate kernel and simulating all the hardware.

Active projects

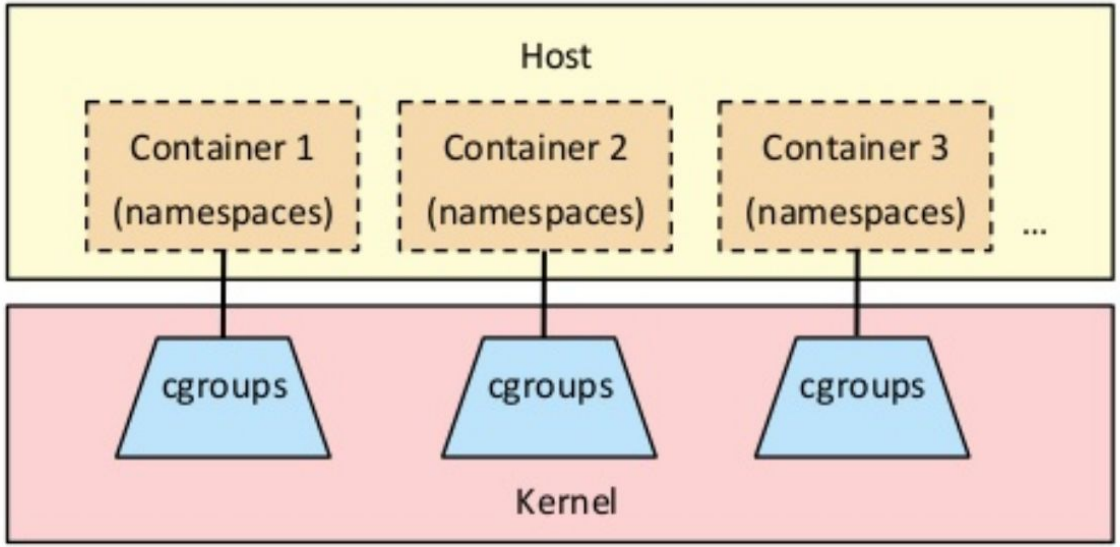
LXC | LXD | LXCFS

docker

Linux Container = namespaces & cgroups

Linux Containers

Container = combination of namespaces & cgroups



https://www.slideshare.net/brendangregg/container-performance-analysis

Linux Container - History

- 1979, Unix V7 and chroot
- 2000, FreeBSD Jails
- 2001, Linux VServer
- 2005, OpenVZ (Virtuzzo)
- 2006, Process Containers
- 2008, LXC
- 2013, Docker

Linux Container - Foundation

- Linux namespaces
- Linux cgroups

Linux Namespaces

- mount (CLONE_NEWNS, 2.4.19)
- UTS (CLONE_NEWUTS, 2.6.19)
- IPC (CLONE_NEWIPC, 2.6.19)
- PID (CLONE_NEWPID, 2.6.24)
- Network (CLONE_NEWNET, 2.6.29)
- User (CLONE_NEWUSER, 3.8)
- cgroups (CLONE_NEWCGROUP, 4.6)

Linux chroot

```
# Mount Kernel Virtual File Systems
TARGETDIR="/mnt/chroot"
mount -t proc proc $TARGETDIR/proc
mount -t sysfs sysfs $TARGETDIR/sys
mount -t devtmpfs devtmpfs $TARGETDIR/dev
mount -t tmpfs tmpfs $TARGETDIR/dev/shm
mount -t devpts devpts $TARGETDIR/dev/pts

# Copy /etc/hosts
/bin/cp -f /etc/hosts $TARGETDIR/etc/

# Copy /etc/resolv.conf
/bin/cp -f /etc/resolv.conf $TARGETDIR/etc/resolv.conf

# Link /etc/mtab
chroot $TARGETDIR rm /etc/mtab 2> /dev/null
chroot $TARGETDIR ln -s /proc/mounts /etc/mtab
```

Linux Namespace - clone

```c
#define _GNU_SOURCE
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <sched.h>
#include <signal.h>
#include <unistd.h>

#define STACK_SIZE (1024 * 1024)
static char container_stack[STACK_SIZE];

char* const container_args[] = {
    "/bin/bash",
    NULL
};

int container_main(void* arg)
{
    printf("Container - inside the container!\n");
    execv(container_args[0], container_args);
    printf("Something's wrong!\n");
    return 1;
}

int main()
{
    printf("Parent - start a container!\n");
    int container_pid = clone(container_main, container_stack+STACK_SIZE, SIGCHLD, NULL);
    waitpid(container_pid, NULL, 0);
    printf("Parent - container stopped!\n");
    return 0;
}
```

```
F74076310@F74076310:~/namespace_demo$ ./clone.out
Parent - start a container!
Container - inside the container!
F74076310@F74076310:~/namespace_demo$ exit
exit
Parent - container stopped!
F74076310@F74076310:~/namespace_demo$
```

Linux Namespace - UTS (Hostname)

```c
#define _GNU_SOURCE
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <sched.h>
#include <signal.h>
#include <unistd.h>

#define STACK_SIZE (1024 * 1024)
static char container_stack[STACK_SIZE];

char* const container_args[] = {
    "/bin/bash",
    NULL
};

int container_main(void* arg)
{
    printf("Container - inside the container!\n");
    sethostname("container",10);
    execv(container_args[0], container_args);
    printf("Something's wrong!\n");
    return 1;
}

int main()
{
    printf("Parent - start a container!\n");
    int container_pid = clone(container_main, container_stack+STACK_SIZE,
                              CLONE_NEWUTS | SIGCHLD, NULL);
    waitpid(container_pid, NULL, 0);
    printf("Parent - container stopped!\n");
    return 0;
}
```

```
F74076310@F74076310:~/namespace_demo$ ./uts.out
Parent - start a container!
Parent - container stopped!
F74076310@F74076310:~/namespace_demo$ sudo ./uts.out
Parent - start a container!
Container - inside the container!
root@container:/home/F74076310/namespace_demo# exit
exit
Parent - container stopped!
F74076310@F74076310:~/namespace_demo$
```

Linux Namespace - IPC

```c
#define _GNU_SOURCE
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <sched.h>
#include <signal.h>
#include <unistd.h>

#define STACK_SIZE (1024 * 1024)
static char container_stack[STACK_SIZE];

char* const container_args[] = {
    "/bin/bash",
    NULL
};

int container_main(void* arg)
{
    printf("Container - inside the container!\n");
    sethostname("container",10);
    execv(container_args[0], container_args);
    printf("Something's wrong!\n");
    return 1;
}

int main()
{
    printf("Parent - start a container!\n");
    int container_pid = clone(container_main, container_stack+STACK_SIZE,
                              CLONE_NEWUTS | CLONE_NEWIPC | SIGCHLD, NULL);
    waitpid(container_pid, NULL, 0);
    printf("Parent - container stopped!\n");
    return 0;
}
```

```
F74076310@F74076310:~/namespace_demo$ ipcmk -Q
Message queue id: 1
F74076310@F74076310:~/namespace_demo$ ipcs -q

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x379eb4b5 1          F74076310  644        0            0

F74076310@F74076310:~/namespace_demo$ ./ipc.out
Parent - start a container!
Parent - container stopped!
F74076310@F74076310:~/namespace_demo$ sudo ./ipc.out
Parent - start a container!
Container - inside the container!
root@container:/home/F74076310/namespace_demo# ipcs -q

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

root@container:/home/F74076310/namespace_demo# exit
exit
Parent - container stopped!
F74076310@F74076310:~/namespace_demo$ ipcs -q

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x379eb4b5 1          F74076310  644        0            0

F74076310@F74076310:~/namespace_demo$
```

Linux Namespace - PID

```c
#define _GNU_SOURCE
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <sched.h>
#include <signal.h>
#include <unistd.h>

#define STACK_SIZE (1024 * 1024)
static char container_stack[STACK_SIZE];

char* const container_args[] = {
    "/bin/bash",
    NULL
};

int container_main(void* arg)
{
    printf("Container [%5d] - inside the container!\n", getpid());
    sethostname("container",10);
    execv(container_args[0], container_args);
    printf("Something's wrong!\n");
    return 1;
}
int main()
{
    printf("Parent [%5d] - start a container!\n", getpid());
    int container_pid = clone(container_main, container_stack+STACK_SIZE,
            CLONE_NEWUTS | CLONE_NEWPID | SIGCHLD, NULL);
    waitpid(container_pid, NULL, 0);
    printf("Parent - container stopped!\n");
    return 0;
}
```

```
F74076310@F74076310:~/namespace_demo$ sudo ./pid.out
Parent [20007] - start a container!
Container [    1] - inside the container!
root@container:/home/F74076310/namespace_demo# echo $$
1
root@container:/home/F74076310/namespace_demo# exit
exit
Parent - container stopped!
F74076310@F74076310:~/namespace_demo$ echo $$
18469
F74076310@F74076310:~/namespace_demo$
```

Linux Namespace - mount

```c
#define _GNU_SOURCE
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <sched.h>
#include <signal.h>
#include <unistd.h>

#define STACK_SIZE (1024 * 1024)
static char container_stack[STACK_SIZE];

char* const container_args[] = {
    "/bin/bash",
    NULL
};

int container_main(void* arg)
{
    printf("Container [%5d] - inside the container!\n", getpid());
    sethostname("container",10);
    system("mount -t proc proc /proc");
    execv(container_args[0], container_args);
    printf("Something's wrong!\n");
    return 1;
}

int main()
{
    printf("Parent [%5d] - start a container!\n", getpid());
    int container_pid = clone(container_main, container_stack+STACK_SIZE,
            CLONE_NEWUTS | CLONE_NEWPID | CLONE_NEWNS | SIGCHLD, NULL);
    waitpid(container_pid, NULL, 0);
    printf("Parent - container stopped!\n");
    return 0;
}
```

```
F74076310@F74076310:~/namespace_demo$ sudo ./mount.out
Parent [20039] - start a container!
Container [    1] - inside the container!
root@container:/home/F74076310/namespace_demo# ps -elf
F S UID          PID    PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S root           1       0  0  80   0 -  2240 do_wai 23:01 pts/1    00:00:00 /bin/bash
0 R root          10       1  0  80   0 -  2654 -      23:01 pts/1    00:00:00 ps -elf
root@container:/home/F74076310/namespace_demo# exit
exit
Parent - container stopped!
F74076310@F74076310:~/namespace_demo$
```

Why?
3) Speed!

	Ships within ...	Manual deployment takes ...	Automated deployment takes ...	Boots in ...
Bare Metal	days	hours	minutes	minutes
Virtualization	minutes	minutes	seconds	less than a minute
Lightweight Virtualization	seconds	minutes	seconds	seconds

Why?
2) Footprint!

On a typical physical server, with average compute resources, you can easily run:

- 10-100 virtual machines
- 100-1000 containers

On disk, containers can be very light.
A few MB — even without fancy storage.

Why?
1) It's still virtualization!

Each container has:

- its own network interface (and IP address)
  - can be bridged, routed... just like $your_favorite_vm
- its own filesystem
  - Debian host can run Fedora container (&vice-versa)
- isolation (security)
  - container A & B can't harm (or even see) each other
- isolation (resource usage)
  - soft & hard quotas for RAM, CPU, I/O...

LXC lifecycle

- `lxc-create`
  Setup a container (root filesystem and config)
- `lxc-start`
  Boot the container (by default, you get a console)
- `lxc-console`
  Attach a console (if you started in background)
- `lxc-stop`
  Shutdown the container
- `lxc-destroy`
  Destroy the filesystem created with lxc-create

Docker

Docker

- dotCloud Inc.
- OS-level virtualization based on container technology
- Released in 2013
- Gets popular around 2014, 2015

Docker

- Build
- Share
- Run

Docker - keywords

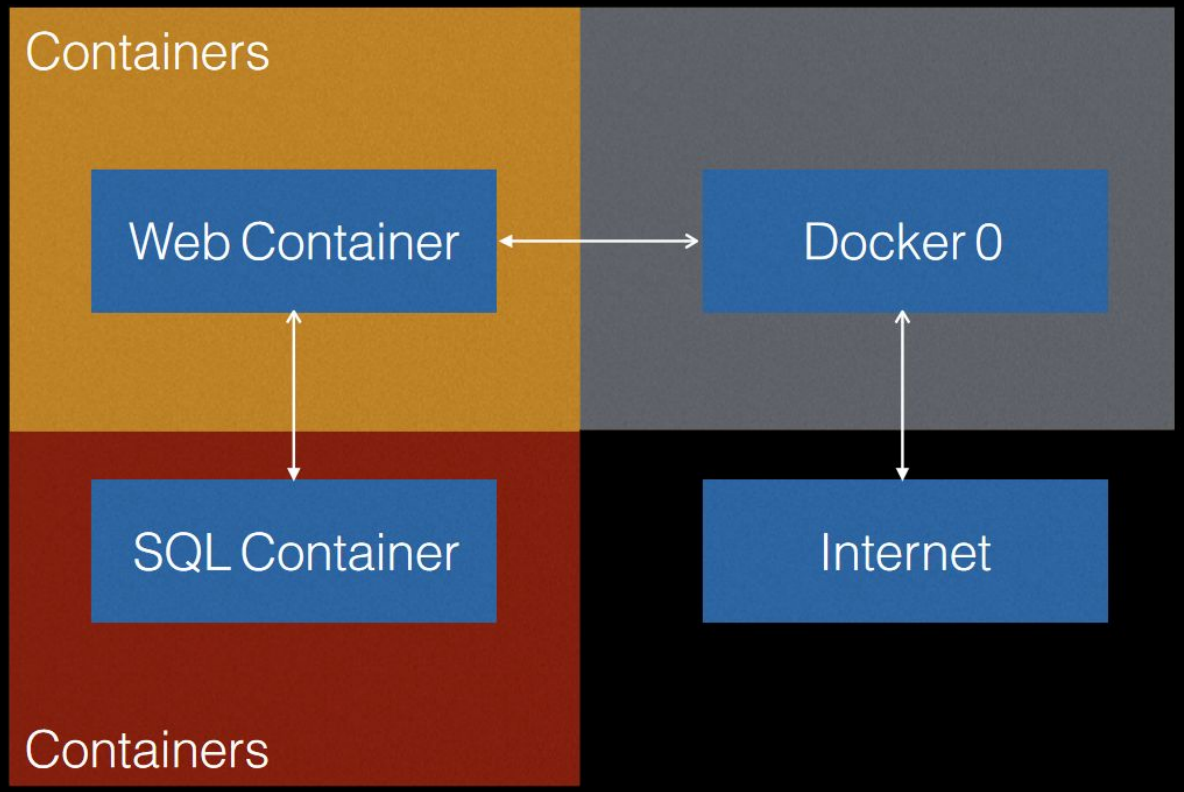
- Image
- Container
- Repository

Docker - Image

- Read-only Layer
- Loads into container
- Commit a container, and you get an image

Docker - Container

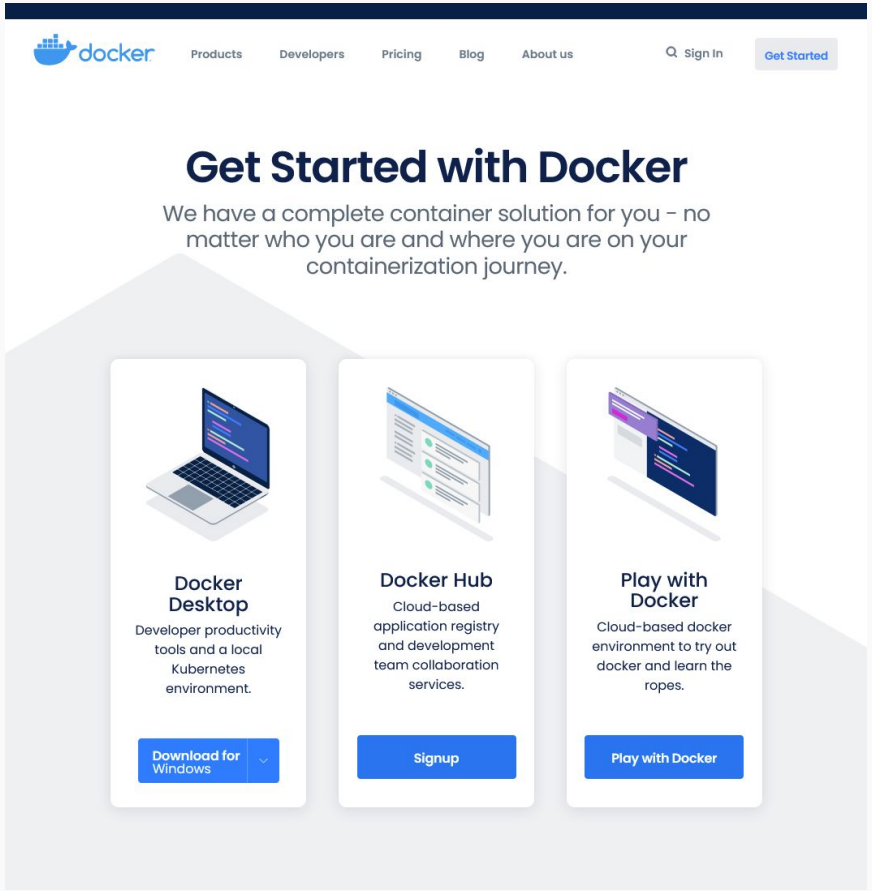
- Container comes from image
- Create / Start / Stop / Delete
- Can link containers together


Containers
Web Container
Docker 0
SQL Container
Internet
Containers

Docker - Repository

- Bunch of Images
- Use tag to track images

Get Started with Docker



curl -fsSL https://get.docker.com | sudo bash

```
F74076310@F74076310:~$ curl -fsSL https://get.docker.com | sudo bash
# Executing docker install script, commit: 7cae5f8b0decc17d6571f9f52eb840fbc13b2737
+ sh -c 'apt-get update -qq >/dev/null'
+ sh -c 'DEBIAN_FRONTEND=noninteractive apt-get install -y -qq apt-transport-https ca-certificates curl >/dev/null'
+ sh -c 'curl -fsSL "https://download.docker.com/linux/ubuntu/gpg" | apt-key add -qq - >/dev/null'
Warning: apt-key output should not be parsed (stdout is not a terminal)
+ sh -c 'echo "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable" > /etc/apt/sources.list.d/docker.list'
+ sh -c 'apt-get update -qq >/dev/null'
+ '[' -n '' ']'
+ sh -c 'apt-get install -y -qq --no-install-recommends docker-ce >/dev/null'
+ '[' -n 1 ']'
+ sh -c 'DEBIAN_FRONTEND=noninteractive apt-get install -y -qq docker-ce-rootless-extras >/dev/null'
+ sh -c 'docker version'
Client: Docker Engine - Community
 Version:           20.10.6
 API version:       1.41
 Go version:        go1.13.15
 Git commit:        370c289
 Built:             Fri Apr  9 22:47:17 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.6
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.13.15
  Git commit:       8728dd2
  Built:            Fri Apr  9 22:45:28 2021
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.4
  GitCommit:        05f951a3781f4f2c1911b05e61c160e9c30eaa8e
 runc:
  Version:          1.0.0-rc93
  GitCommit:        12644e614e25b05da6fd08a38ffa0cfe1903fdec
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0

================================================================================

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

    dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.


To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
         to root access on the host. Refer to the 'Docker daemon attack surface'
```

Docker

- docker pull
- docker run
- docker build
- docker tag
- docker push
- docker login

docker pull

- Get image from Docker Hub
- Docker Hub
  - Largest Docker Image registry in the world
  - Official images

docker run

- Turn image into container
- It’s like install VM into your computer, then run the VM
  - Just **a lot more** quicker

docker pull & docker run

```
F74076310@F74076310:~$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:5122f6204b6a3596e048758cabba3c46b1c937a46b5be6225b835d091b90e46c
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
F74076310@F74076310:~$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

docker run

- -d
  - Run container in background and print container ID
- -i
  - Keep STDIN open even if not attached
- -p
  - Publish a container’s port(s) to the host
- -t
  - Allocate a pseudo-TTY
- -v
  - Bind mount a volume

docker run -it ubuntu:20.04 /bin/bash

```
F74076310@F74076310:~$ docker run -it ubuntu:20.04 /bin/bash
Unable to find image 'ubuntu:20.04' locally
20.04: Pulling from library/ubuntu
345e3491a907: Pull complete
57671312ef6f: Pull complete
5e9250ddb7d0: Pull complete
Digest: sha256:b4aa552dd3f2ed84f3214b0e8add3648aee0205ef58295c92fe0899f96ad8755
Status: Downloaded newer image for ubuntu:20.04
root@1943a1f70bc7:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@1943a1f70bc7:/# apt update
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [27.6 kB]
Get:4 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [833 kB]
Get:5 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [721 kB]
Get:9 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [274 kB]
35% [7 Packages 1907 kB/11.3 MB 17%]
```

docker run -p 8080:80 nginx:alpine

```
F74076310@F74076310:~$ docker run -p 8080:80 nginx:alpine
Unable to find image 'nginx:alpine' locally
alpine: Pulling from library/nginx
540db60ca938: Pull complete
0ae30075c5da: Pull complete
9da81141e74e: Pull complete
b2e41dd2ded0: Pull complete
7f40e809fb2d: Pull complete
758848c48411: Pull complete
Digest: sha256:0f8595aa040ec107821e0409a1dd3f7a5e989501d5c8d5b5ca1f955f33ac81a0
Status: Downloaded newer image for nginx:alpine
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2021/05/26 22:07:56 [notice] 1#1: using the "epoll" event method
2021/05/26 22:07:56 [notice] 1#1: nginx/1.21.0
2021/05/26 22:07:56 [notice] 1#1: built by gcc 10.2.1 20201203 (Alpine 10.2.1_pre1)
2021/05/26 22:07:56 [notice] 1#1: OS: Linux 5.4.0-73-generic
2021/05/26 22:07:56 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2021/05/26 22:07:56 [notice] 1#1: start worker processes
2021/05/26 22:07:56 [notice] 1#1: start worker process 31
172.17.0.1 - - [26/May/2021:22:07:58 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.68.0" "-"
```

```
F74076310@F74076310:~$ curl localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
F74076310@F74076310:~$
```

docker run -v ...

```
F74076310@F74076310:~$ cd docker-demo/
F74076310@F74076310:~/docker-demo$ ls
bar
F74076310@F74076310:~/docker-demo$ cat bar
        File: bar
        foo
F74076310@F74076310:~/docker-demo$ docker run -v /home/F74076310/docker-demo:/docker-demo -it ubuntu:20.04 /bin/bash
root@4284ffa5b5d1:/# ls
bin  boot  dev  docker-demo  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@4284ffa5b5d1:/# cat docker-demo/bar
foo
root@4284ffa5b5d1:/#
```

docker run -d ...

```
F74076310@F74076310:~/Docker-Build$ docker run -d ubuntu:20.04 /bin/bash -c 'while true; do echo Hello World; sleep 1; done;'
54c82a115b16040edb33320e73886800f67d59fd3a66186922177ec13254e50f
F74076310@F74076310:~/Docker-Build$ docker logs 54c82a
Hello World
Hello World
Hello World
Hello World
Hello World
F74076310@F74076310:~/Docker-Build$
```

docker ps

```
F74076310@F74076310:~/docker-demo$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
F74076310@F74076310:~/docker-demo$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED              STATUS                       PORTS     NAMES
4284ffa5b5d1   ubuntu:20.04   "/bin/bash"              About a minute ago   Exited (0) 8 seconds ago               jolly_raman
c362934c59aa   nginx:alpine   "/docker-entrypoint.…"   4 minutes ago        Exited (0) 2 minutes ago               stupefied_taussig
1943a1f70bc7   ubuntu:20.04   "/bin/bash"              6 minutes ago        Exited (130) 4 minutes ago             awesome_joliot
43b3bb105c45   hello-world    "/hello"                 15 minutes ago       Exited (0) 15 minutes ago              musing_colden
F74076310@F74076310:~/docker-demo$
```

docker build

- Create a new image based on the definition
- Create a new file called Dockerfile
  - Define steps in Dockerfile
- docker build ...

docker build … (Demo Repo)

```
F74076310@F74076310:~$ git clone https://github.com/MicrosoftLearning/Docker-Build.git
Cloning into 'Docker-Build'...
remote: Enumerating objects: 77, done.
remote: Total 77 (delta 0), reused 0 (delta 0), pack-reused 77
Unpacking objects: 100% (77/77), 60.42 KiB | 519.00 KiB/s, done.
F74076310@F74076310:~$ cd Docker-Build/
F74076310@F74076310:~/Docker-Build$ ls
Dockerfile  LICENSE  _build.yml  attribution.md  package.js  package.json  readme.md  template.docx
F74076310@F74076310:~/Docker-Build$ vim Dockerfile
F74076310@F74076310:~/Docker-Build$ docker build -t microsoft-learning-docker-build .
Sending build context to Docker daemon  268.3kB
Step 1/5 : FROM node:8
8: Pulling from library/node
146bd6a88618: Pull complete
9935d0c62ace: Pull complete
db0efb86e806: Pull complete
e705a4c4fd31: Pull complete
c877b722db6f: Pull complete
645c20ec8214: Pull complete
db8fbd9db2fe: Pull complete
1c151cd1b3ea: Pull complete
fbd993995f40: Pull complete
Digest: sha256:a681bf74805b80d03eb21a6c0ef168a976108a287a74167ab593fc953aac34df
Status: Downloaded newer image for node:8
 ---> 8eeadf3757f4
Step 2/5 : RUN apt-get update
 ---> Running in 906eaa9d14d7
Get:1 http://security.debian.org/debian-security stretch/updates InRelease [53.0 kB]
Ign:2 http://deb.debian.org/debian stretch InRelease
Get:3 http://deb.debian.org/debian stretch-updates InRelease [93.6 kB]
Get:4 http://deb.debian.org/debian stretch Release [118 kB]
Get:5 http://security.debian.org/debian-security stretch/updates/main amd64 Packages [688 kB]
Get:6 http://deb.debian.org/debian stretch Release.gpg [2410 B]
Get:7 http://deb.debian.org/debian stretch/main amd64 Packages [7080 kB]
```

docker build … (cont.)

```
Setting up liblua5.1-0:amd64 (5.1.5-8.1+b2) ...
Setting up pandoc (1.17.2~dfsg-3) ...
Processing triggers for libc-bin (2.24-11+deb9u4) ...
Removing intermediate container 9e399cd14aaf
 ---> 80a536f413fa
Step 4/5 : COPY . ./
 ---> 9bd78721157e
Step 5/5 : RUN npm install
 ---> Running in 5361e72f24aa
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
npm notice created a lockfile as package-lock.json. You should commit this file.
added 273 packages from 168 contributors and audited 274 packages in 22.241s

2 packages are looking for funding
  run `npm fund` for details

found 2 vulnerabilities (1 moderate, 1 high)
  run `npm audit fix` to fix them, or `npm audit` for details
Removing intermediate container 5361e72f24aa
 ---> 677923e095b8
Successfully built 677923e095b8
Successfully tagged microsoft-learning-docker-build:latest
F74076310@F74076310:~/Docker-Build$ docker images
REPOSITORY                      TAG       IMAGE ID       CREATED              SIZE
microsoft-learning-docker-build latest    677923e095b8   About a minute ago   1.02GB
nginx                           alpine    a6eb2a334a9f   31 hours ago         22.6MB
ubuntu                          20.04     7e0aa2d69a15   4 weeks ago          72.7MB
hello-world                     latest    d1165f221234   2 months ago         13.3kB
node                            8         8eeadf3757f4   17 months ago        901MB
```

docker push & docker login

- docker push
  - Push your image to designated registry
- docker login
  - Log in to designated registry
- Both docker push and docker login direct to Docker Hub

Docker is good for...

- Continuous Integration
- Escape dependency hell
- Save resources
- Fast service deployment

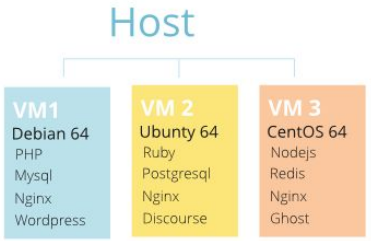
Docker is not (really) good for...

- Stateful Application
- Graphical Application
- Cross-platform Application
- Performance-critical Application
- Security-focused Application

flocport

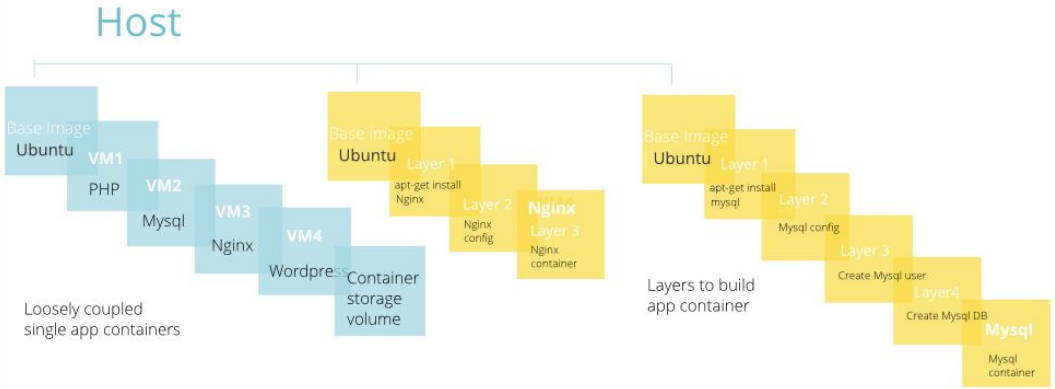
Key differences between LXC and Docker

LXC



- Filesystem neutral
- Containers are like VMs with a fully functional OS
- Data can be saved in a container or outside
- Build loosely coupled or composite stacks

Docker



- Containers are made up of read only layers via AUFS/Devicemapper
- Containers are designed to support a single applicaton.
- Instances are ephemeral, persistent data is stored in bind mounts to host or data volume containers

Docker - Learn More

- Docker Compose
- Docker Swarm
- Kubernetes
- Kata Containers (OpenStack)
- Podman