Lightweight Virtualization with Linux Containers (LXC)

The 5th China Cloud Computing Conference
June 7th, 2013
China National Convention Center, Beijing

The 5th China Cloud Computing Conference

Outline

- Introduction : who, what, why?

- Linux Containers (LXC)
- Namespaces
- Control Groups (cgroups)
- AUFS and BTRFS
- Docker

- Conclusion, Questions

But, first...

I would like to thank CSDN for inviting me to share our knowledge about containers!

谢谢

CSDN

全球最大中文IT社区

Introduction: Who am I ?

Jérôme Petazzoni

dotCloud (San Francisco)

Senior Software Engineer

SRE Team (=DevOps)

Introduction:
What is this about ?

- LXC (LinuX Containers) let you run a Linux system within another Linux system.
- A container is a group of processes on a Linux machine.
- Those processes form an isolated environment.
- Inside the container, it looks like a VM.
- Outside the container, it looks like normal processes running on the machine.
- It looks like a VM, but it is more efficient:
  Containers = Lightweight Virtualization

Introduction:
Why should I care ?

- Shipping containers changed world trade

  By sharply cutting costs and enhancing reliability, container-based shipping enormously increased the volume of international trade and made complex supply chains possible. (New York Times, 2006)

- Linux Containers will change the world, too

  The goal of a Standard Container is to encapsulate a software component and all its dependencies in a format that is self-describing and portable, so that any compliant runtime can run it without extra dependency, regardless of the underlying machine and the contents of the container.

Linux Containers:
They are awesome!

Three reasons why Linux Containers are awesome

Linux Containers: They are *fast*

	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	less than a second	minutes	less than a second	less than a second

Linux Containers:
They are *lightweight*

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 megabytes per container.

No need for special storage like SAN, NAS...

Linux Containers: They are *almost* Virtual Machines

Each container has:

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

Linux Containers:
Some use-cases

You can use
Linux Containers for…

Use-case:
Cost-efficient hosting

- For big websites and applications:
  less overhead than Virtual Machines
- For small websites and applications:
  10-1000x more efficient than Virtual Machines
- Automatic shutdown when there is no trafic
- Boot when a HTTP request arrives

dotCloud uses Linux Containers to run its Platform-as-a-Service (PAAS) offering

Use-case: Development

- Automatic testing with continuous integration
  - After each commit, run 100 tests in 100 VMs
- No more problems with library version etc.
  - Build and run in a controlled environment
- Put every project, big or small, in a VM
  - Isolation + easy to clone, transfer ...

Use-case:
Deployment & Software Delivery

- Work in a Linux Container
  on your local computer
- Deploy the same Linux Container
  to any server: on premise, IAAS...
- Guaranteed « repeatabiity »

Use-case: Better Virtual Machines

- Look inside your VMs
  - You can see (and kill) individual processes
  - You can browse (and change) the filesystem
- Do whatever you did with VMs
  - ... But faster

How to use Linux Containers?

- On Debian or Ubuntu Linux:

  apt-get install lxc

Linux Containers Basic Commands

- 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

Technical Details

namespaces

+

cgroups

=

Linux Containers

Namespaces

Partition essential kernel structures to create virtual environments e.g., you can have multiple processes with PID 42, in different environments

Different kinds of namespaces

- **pid (processes)**
- **net (network interfaces, routing...)**
- ipc (System V IPC)
- mnt (mount points, filesystems)
- uts (hostname)
- user (UIDs)

Creating namespaces

- Use the « clone() » system call with extra flags
- Command-Line tool « unshare »
  (a bit like « chroot »)

- Notes:
  - You don't have to use all namespaces
  - A new process inherits its parent's namespace
  - Use lxc-attach to enter an existing namespace
    (requires kernel 3.8)

Namespaces: pid

- Processes in a pid namespace don't see processes of the whole system
- Each pid namespace has a PID #1
- pid namespaces are actually nested
- A given process can have multiple PIDs
  - One in each namespace it belongs to
  - So you can easily access processes of children ns
- Can't see/affect processes in parent/sibling ns

Namespaces: net

- Each net namespace has its own…
  - Network interfaces (and its own lo/127.0.0.1)
  - IP address(es)
  - routing table(s)
  - iptables rules
- Communication between containers:
  - UNIX domain sockets (=on the filesystem)
  - Pairs of veth interfaces

One word about... Software-Defined Networking

It is possible to configure a network namespace from outside.

Example to set the IP address of eth0 in the container containing process 42:

```
ln -s /proc/42/ns/net /var/run/netns/nihao
ip netns exec nihao ifconfig eth0
```

This allows:

- Put containers in VLANs, VPNs...
- Openvswitch integration
- etc.

Namespaces: ipc

- Remember "System V IPC"?
  - msgget, semget, shmget
- Have been (mostly) superseded by POSIX alternatives: mq_open, sem_open, shm_open
- However, some stuff still uses "legacy" IPC.
- **Most (only?) notable example: PostgreSQL**
- The problem: xxxget() asks for a key, usually derived from the inode of a well-known file
- The solution: ipc namespace

Namespaces: mnt

- Deluxe chroot()
- A mnt namespace can have its own rootfs
- Filesystems mounted in a mnt namespace are visible only in this namespace
- You need to remount special filesystems, e.g.:
  - procfs (to see your processes)
  - devpts (to see your pseudo-terminals)

Namespaces:
uts

- Deals with just two syscalls: gethostname(),sethostname()
- Allows containers to have their own hostname
- Some tools (e.g.: sudo) behave differently depending on the hostname (sudo)

Namespaces: user

- User ID 42 in container 1
  is different from
  user ID 42 in container 2
- If you use the pid namespace, containers are already isolated anyway
- Useful for system-wide, per-user resource limits if you don't use cgroups

Control groups

Control Groups
Create as many cgroups as you like.
Put processes within cgroups.
Limit, account, and isolate resource usage.

Similar to ulimit, but for groups of processes
… and with fine-grained accounting.

Cgroups: The basics

- Everything exposed through a virtual filesystem
  - Older systems: /cgroup
  - New systems: /sys/fs/cgroup
- Create a cgroup:

  mkdir /cgroup/nihao
- Move process with PID 1234 to the cgroup:

  echo 1234 > /cgroup/nihao/tasks
- Limit memory usage:

  echo 10000000 > /cgroup/nihao/memory.limit_in_bytes

Cgroups: memory

- Limit
  - memory usage, swap usage
  - soft limits and hard limits
  - can be nested
- Account
  - cache vs. rss
  - active vs. inactive
  - file-backed pages vs. anonymous pages
  - page-in/page-out
- Isolate
  - Use hard limits to reserve memory

Cgroups: cpu

- Limit
  - Set cpu.shares (defines relative weights)
- Account
  - Check cpustat.usage for user/system breakdown
- Isolate
  - Use cpuset.cpus (also for NUMA systems)

Cgroups: blkio (block input/output)

- Limit & Isolate
  - blkio.throttle.{read,write}.{iops,bps}.device
  - Drawback: only for sync I/O

    (i.e.: "classical" reads; not writes; not mapped files)
- Account
  - Number of IOs, bytes, service time...
  - Drawback: same as previously

Cgroups aren't perfect if you want to limit I/O.

Limiting the amount of dirty memory helps a bit.

This will probably improve in future kernel versions.

AUFS

Writable single-system images

or

Copy-on-write at the filesystem level

AUFS
Quick Example

You have the following directories:

```
/images/ubuntu
/containers/nihao/rootfs
/containers/nihao/rw
```

Run this command:

```
mount -t aufs \
  -o br=/containers/nihao/rw=rw:/images/ubuntu=ro \
  none /containers/nihao/rootfs
```

Now, you can write to /containers/nihao/rootfs: changes will go to the /containers/nihao/rw directory.

AUFS
Good things

- Start thousands of containers,
  with a single image
- Create a new container = instantaneous
  (no need to copy the image)
- Easy to see modifications
  (in the « rw » directory)

AUFS
Bad things

- Not integrated in normal Linux Kernel
- Integrated with Debian and Ubuntu kernels
- Big problem for users of CentOS, RedHat...

BTRFS

Snapshotting filesystem

BTRFS
Quick Example

Create a « subvolume »

```
btrfs subvolume create /images/ubuntu
```

Create a basic Ubuntu system in the volume

```
debootstrap raring /images/ubuntu
```

Create an image from the volume

```
btrfs subvolume snapshot /images/ubuntu /containers/nihao
```

Available in all kernels

Warning: Linux distributions use ext4 by default

Docker

The Linux Container Engine

Docker:
What it is

- Open Source project using LXC + AUFS
  (soon BTRFS)

  automates the deployment of applications as highly portable, self-sufficient containers which are independent of hardware, language, framework, packaging system and hosting provider

- Written in Go
- Official distribution: Ubuntu 13.04
  (Works on others as well)

Docker:
What people say

« Docker looks heaven-sent »
« Awesome project »
« Docker is git for VMs »
« Exciting to see stuff like this being developed! »
« Just wow! »
« Having my mind blown by LXC and Docker »
« Looks really promising »
« Containerized app deployment is the future »
« It's all kinds of magic »
« Pure excellence »
« I've just build an elasticsearch container based on a openjdk container in less than 10 minutes! »
« This will change how we build operating systems »

Docker: Quick example

Installation (on Ubuntu 13.04)

```
laptop:~# curl get.docker.io|sh
```

Download, create, and start a new container

```
laptop:~# docker run -t -i ubuntu bash
root@93b769e86404:/#
```

Inside the container, change a few files and exit

```
root@93b769e86404:/# touch yi er san
root@93b769e86404:/# exit
```

See what was changed

```
laptop:~# docker diff 93b7
A /er
A /san
A /yi
```

Docker:
Creating a new image

Start a new container, install MySQL in it:

```
laptop:~# docker run -t -i ubuntu bash
root@bc6abb7bce23:/# apt-get install -qqy mysql-server
root@bc6abb7bce23:/# exit
```

Create a new image:

```
laptop:~# docker commit bc6abb jpetazzo/mysql
```

Send the image to the Docker index:

```
laptop:~# docker push jpetazzo/mysql
```

Then, on another server...

```
alibaba-srv-01:~# docker run -d jpetazzo/mysql
```

Remember: images are automatically downloaded.

Docker: Very fast development

March 23: version 0.1
April 23: version 0.2
May 6: version 0.3
June 3: version 0.4
+ approx. 1 minor version per week

- Big engineering commitment by dotCloud
- Many external contributors
- Last night, we just announced openstack-docker

Thank you!

Docker website:
http://www.docker.io/

Docker source code:
https://github.com/dotcloud/docker/

dotCloud technical blog:
http://blog.dotcloud.com/

Follow us on Twitter:
@dot_cloud @getdocker @jpetazzo

Those slides:
http://db.tt/jCBxCydh