Developers and operations have their specificities.
Developers have been automating software building

- Make-based tools:
  - make, Rake, Snakemake

- Non-make-based tools:
  - Apache Ant, Buildr, Maven, MSBuild, Scons

- Continuous integration tools:
  - Jenkins, Magnum, Bamboo, BuildBot
Current demands:

- **Continuous deliver** is required (up to production)
- **Incident response** must be fast
- **Operations** have to **automate** environment and application deployment
- **Operations and developers** need to work **closer**
DevOps?
DevOps?

Worked fine in Dev

Dev-Ops

One does not simply

SAY DEVS

I should learn DevOps

Yo dawg I heard you like DevOps

So I put Dev in your Ops so you can Dev when you Ops!

Not sure if DevOps is about collaboration

Or just Ops losing responsibility

One does not simply

Walk into DevOps

Deploy to production

DevOps

Deliver business value faster

With less risk

Say DevOps

One more time

I don't always Deploy Django

But when I do, I use Heroku

Brace yourselves

DevOps is coming

You wanna break the build

Or you wanna go home?

Just found

Where dev and ops meet

OMG!

DevOps is coming!

Prepare to die
• Operations and development engineers participating together in the entire service lifecycle

• From design through the development process to production support

• Operations staff making use many of the same techniques as developers for their systems work
(Some) Emerging DevOps solutions

docker
OpenVZ Linux Containers
PACKER
SALTSTACK
CRUSH
VAGRANT
ANSIBLE
puppet labs
CHEF
• Configuration management:
  • Puppet, Chef, Ansible, SaltStack …
  • Describe environment + automatic configuration in execution time

• Containerization tools:
  • LXC, OpenVZ, Docker, Kubernetes, DxEnterprise …
  • Basic environment + libs and apps in a lightweight image

• Virtualization platforms:
  • Xen, KVM, VirtualBox, VMWare, Vagrant …
  • VMM + Guest OS image with the environment, libs and apps

• Infrastructure-as-a-Service:
  • AWS, Rackspace, Azure, OpenStack …
  • On-demand resources + VM image(s)
• Configuration management:
  • Puppet, Chef, Ansible, SaltStack …
  • Describe environment + automatic configuration in execution time

• Containerization tools:
  • LXC, OpenVZ, **Docker**, Kubernetes, DxEnterprise …
  • Basic environment + libs and apps in a lightweight image

• Virtualization platforms:
  • Xen, KVM, VirtualBox, VMWare, Vagrant …
  • VMM + Guest OS image with the environment, libs and apps

• Infrastructure-as-a-Service:
  • AWS, Rackspace, Azure, OpenStack …
  • On-demand resources + VM image(s)
Virtual machines vs. containers

Virtual Machine Monitor

Host OS

Hardware

Guest OS

Bins/Libs

A

A

B

Guest OS

Bins/Libs

Virtual Machine

Container Manager

Host OS

Hardware

Bins/Libs

A

A

B
What is Docker?

An open **platform** for developers and sysadmins to **build**, **ship**, and **run** distributed applications

IT can ship faster and run the same app, **unchanged**, on laptops, data center VMs, and any cloud

- **Docker Client**
  - Interaction tool (e.g., GUI, CLI ...)
- **Docker Daemon**
  - Lightweight runtime and packaging tool
- **Docker Registry**
  - Private catalog of Docker images
- **Docker Hub**
  - Cloud service for sharing applications and automating workflows
- **Docker Image**
  - The libs+apps image volume
- **Docker Container**
  - A running Docker image
Docker infrastructure

- Docker Hub
- Docker Registry
- Docker Image
- Docker Daemon
- Docker Image
- Docker Container
- Docker Client
Docker architecture

- Docker Daemon
- LXC
- LayerFS
- CGroups
- Namespaces
- Linux Kernel
- Host OS
Docker architecture - LayerFS

Use union mounts

Lower layers are read-only

Only the top layer is writable

Old versions of files are invisible but they are in the lower layers
Docker architecture - CGroups

Resource management

Groups of processes

Allocate resources for each group

CPU, RAM, Network, and Disk resources
### Docker architecture - Namespaces

<table>
<thead>
<tr>
<th>Docker Daemon</th>
<th>LXC</th>
<th>LayerFS</th>
<th>CGroups</th>
<th>Namespaces</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux Kernel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host OS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Isolate groups of processes

PID, NICs, mounts, users ...

One group does not see the resources from others
Docker architecture - LXC

<table>
<thead>
<tr>
<th>LayerFS</th>
<th>CGroups</th>
<th>Namespaces</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Docker Daemon

LXC

Linux Kernel

Host OS

OS-level virtualization

Provide isolated environment for applications

LXC is one of Docker’s execution drivers
Docker and the SuperCloud project

- Efficiency
- Performance
- Isolation
- Security
- Live migration and state transfer
- Diversity
- Nested virtualization
The same Docker image could be efficiently deployed on anything from a fraction of a core to an entire machine.

Sequential/random read/writes

Throughput and latency in Redis NoSQL

- Docker equals or exceeds KVM performance in every case they tested.

- Both KVM and Docker introduce negligible overhead for CPU and memory performance.

- For I/O-intensive workloads, both forms of virtualization should be used carefully.

- Deploying containers inside VMs imposes the performance overheads of VMs while giving no benefit compared to deploying containers directly on non-virtualized Linux.
Isolation

- Docker uses for example LXC as execution driver
- LXC uses Kernel’s namespaces and cgroups
- Different users can run different containers in the same host
- Apps from one container do not see or affect resources from other containers
Security

- Security at **namespace** level = LXC
- Security at **resource** accounting and limiting = cgroups
- **Increased attack surface** in Docker daemon, which requires root privileges
- Docker daemon can **expose** a **REST API** over HTTP to remote clients
- Foreseen **secure improvements**:  
  - Map root of a container to a non-root user in the host  
  - Run unprivileged daemon
- Running a VM inside a container can create an extra layer of security since an attacker who can exploit QEMU would still be inside the container
Live migration and state transfer

• Docker **doesn’t** support **live migration**

• **Flocker** is a project trying to migrate active containers within a cluster (still with downtime)

• **CRIU** (Checkpoint/Restore In Userspace) is a project trying to perform live migration of LXC containers

• Three options from CRIU:
  • Stop old, copy, start new, kill old
  • Pre-dump old, stop old, copy diff, start new, kill old
  • Disk-less migration (transfer pages instead of disks)

• But it is not straightforward, and still is unstable
Diversity

- Multiple platforms supporting Docker:
  - OS-X and Windows **don’t** support Docker containers by default

- **Boot2Docker** is a “tiny” VirtualBox VM to run Docker in OS-X and Windows

- Docker supporting multiple platforms:
  - Docker containers run **only Linux** containers (Ubuntu, Debian, Fedora, RedHat ...)

- It means, OS-X or Windows apps don’t run on Docker

- DxEnterprise is a competitor for running Windows apps
Nested virtualization
Docking Dockers

Docker within a VM

Original container size

VMs are used to grow

Works in current clouds!

Only Linux containers

No diversity
Docking VMMs

VMM within a container

Container size grows (a lot!)

Doesn’t work in clouds!

OS and apps diversity

Secure VMMs (within the namespace)
Conclusions

- Both VMs and containers are mature technology
- Containers do not replace virtual machines
- Docker does not support live migration (Flocker does, with downtime)
- Docker within VMs (scenario 01) seems to be more appropriate for SuperCloud
- Docker probably will cause less overhead in nested virtualization
- However, we probably would need to code or hack Docker to have all desired features
Thank you!