

Config Management and Containers

Charles Butler
Fosdem 2016



@lazypower

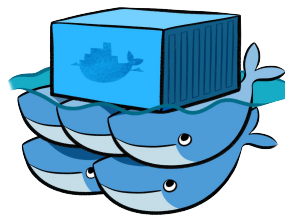
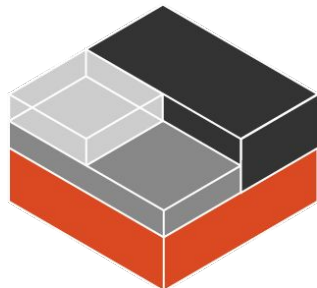
charles.butler@ubuntu.com

<http://blog.dasroot.net>

<http://github.com/chuckbutler>



We are the company
behind Ubuntu.




JUJU



kubernetes

Genesis



“Operational pain can
neither be created nor
destroyed - only moved
to someone else”

- Nick Galbreath



Well... You can
create it... :)

- Joshua Corman

System Management Patterns

Divergence

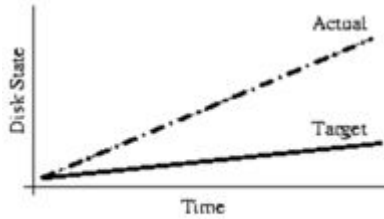


Figure 1: Divergence.

Convergence

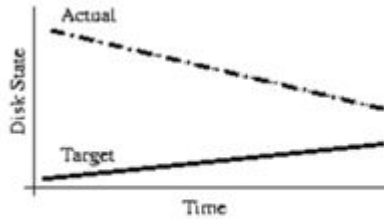


Figure 2: Convergence.

Congruence

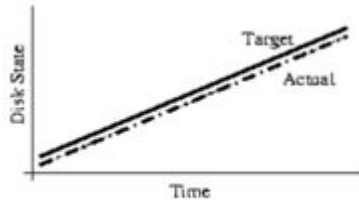


Figure 3: Congruence.

Config Management Solved Problems

- 1 Stopped divergent delivery patterns from a pre-virtualized world
- 2 Best Attempt to eliminate snowflakes
- 3 Frameworks to describe machine state
- 4 Support upstream packaging (or from source deployments)
- 5 resource abstraction

Emergent issues w/ Config Management

1

Domain specific
configuration
managers

2

Context Sensitive
Knowledge barriers.

3

10% technological—
the rest is improved
management,
process, and user
training. [1]

Enter Containers

The New Stack

Containers offer a way to virtualize an operating system.

This virtualization isolates processes, providing limited visibility and resource utilization to each, such that the processes appear to be running on separate machines.

Flavors

Application Containers

- Single Process
- No init
- No amenities like cron
- No SSH
- typically run/handled as immutable objects

System Containers

- Many processes
- runs /sbin/init
- Has amenities like cron
- SSH'able
- Can be treated as immutable or mutable. But designed to be mutable

```

x - [ ] Terminal
1 [|||||] 10.3% Tasks: 13, 3 thr: 1 running
2 [|||||] 13.1% Load average: 1.20 1.07 1.40
Mem [|||||] 11/1882MB Uptime: 00:08:44
Swp [|||||] 938/1916MB

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
1296 root 20 0 26164 3652 2612 R 0.0 0.2 0:00.43 /bin
1 root 20 0 36968 3096 2156 S 0.0 0.2 0:00.36 /sbin/init
524 root 20 0 35256 2868 2580 S 0.0 0.1 0:00.05 /lib/systemd/systemd-journald
765 root 20 0 23460 7092 196 S 0.0 0.4 0:00.00 dhclient -1 -v -pf /run/dhclient.eth0.pid -lf /va
835 syslog 20 0 250M 2400 1980 S 0.0 0.1 0:00.00 /usr/sbin/rsyslogd -n
836 syslog 20 0 250M 2400 1980 S 0.0 0.1 0:00.00 /usr/sbin/rsyslogd -n
837 syslog 20 0 250M 2400 1980 S 0.0 0.1 0:00.00 /usr/sbin/rsyslogd -n
800 syslog 20 0 250M 2400 1980 S 0.0 0.1 0:00.01 /usr/sbin/rsyslogd -n
903 root 20 0 20904 2340 2116 S 0.0 0.1 0:00.00 /usr/sbin/cron -f
962 root 20 0 69932 3260 2508 S 0.0 0.2 0:00.01 /usr/sbin/sshd -D
1029 root 20 0 15676 1852 1696 S 0.0 0.1 0:00.00 /sbin/agetty --noclear --keep-baud pts/2 115200 3
1044 root 20 0 15676 1852 1708 S 0.0 0.1 0:00.00 /sbin/agetty --noclear --keep-baud pts/1 115200 3
1058 root 20 0 15676 1860 1708 S 0.0 0.1 0:00.00 /sbin/agetty --noclear --keep-baud pts/0 115200 3
1072 root 20 0 15676 1844 1704 S 0.0 0.1 0:00.00 /sbin/agetty --noclear --keep-baud pts/3 115200 3
1086 root 20 0 15676 1840 1696 S 0.0 0.1 0:00.00 /sbin/agetty --noclear --keep-baud console 115200
1237 root 20 0 21124 3296 2796 S 0.0 0.2 0:00.01 /bin/bash

1 [|||||] 11.7% Tasks: 165, 484 thr: 1 running
2 [|||||] 16.4% Load average: 1.31 1.09 1.41
Mem [|||||] 1056/1893MB Uptime: 18:29:59
Swp [|||||] 938/1916MB

NI VIRT RES SHR S CPU% MEM% TIME+ Command
0 359M 30956 19124 S 3.8 1.6 13:01.28
0 26660 4528 3040 R 1.9 0.2 0:02.87
0 1492M 34964 18816 S 0.9 1.8 17:45.72
0 349M 2184 1600 S 0.9 0.1 7:58.60
0 349M 2184 1600 S 0.9 0.1 5:17.12
0 1146M 94676 32484 S 0.9 4.9 0:02.47
0 629M 18676 10484 S 0.5 1.0 3:23.78
0 278M 2884 2316 S 0.5 0.1 2:40.34
0 1242M 24356 1824 S 0.5 1.3 0:02.03
0 1173M 93232 31948 S 0.5 4.8 0:06.71
0 344M 2252 2164 S 0.5 0.1 0:01.49
0 33612 2492 2152 S 0.0 0.1 0:11.10
0 981M 60692 27956 S 0.0 3.1 2:04.14
0 69504 11820 4500 S 0.0 0.6 0:03.37
0 793M 36956 19576 S 0.0 1.9 1:18.65
0 981M 60692 27956 S 0.0 3.1 0:18.75
0 26164 3652 2612 S 0.0 0.2 0:00.43
0 285M 3180 2620 S 0.0 0.2 1:11.37
0 770M 32344 15280 S 0.0 1.7 0:05.24
0 981M 60692 27956 S 0.0 3.1 0:01.68
15 1242M 24356 1824 S 0.0 1.3 0:26.44
0 981M 60692 27956 S 0.0 3.1 0:00.84
0 547M 6244 4720 S 0.0 0.3 0:04.17
0 522M 9752 7068 S 0.0 0.5 0:18.85
0 482M 4984 4104 S 0.0 0.3 0:00.81
18 20204 6724 1764 S 0.0 0.3 0:36.75
0 529M 10880 7872 S 0.0 0.6 0:27.25
0 315M 7364 624 S 0.0 0.4 0:00.83
0 29852 2500 211 S 0.0 0.0 0:06.47
0 44100 3612 211 S 0.0 0.0 0:16.47
0 29760 2588 184 S 0.0 0.0 0:51.11
0 344M 2252 216 S 0.0 0.0 0:38.77
0 483M 984 8 S 0.0 0.0 0:04.30
0 620M 18664 9956 S 0.0 1.0 0:31.19
0 692M 12108 7956 S 0.0 0.6 0:07.78
0 1580M 40188 22704 S 0.0 2.1 0:39.14
0 340M 5684 3156 S 0.0 0.3 0:13.92
0 620M 18664 9956 S 0.0 1.0 0:10.52
0 340M 5684 3156 S 0.0 0.3 0:08.98
0 1146M 94676 32484 S 0.0 4.9 0:00.03

```

LXC HOST

Benefits of “the new stack”

- 1 Resource Constraints
- 2 Density
- 3 Super Fast (often sub second)
- 4 No VM Overhead

Why Config Management & Containers

A critical look

Model Everything



Model containers and non-containers

manage not only the containers, but the environments around the containers

This is especially important, as containerized applications are nearly always talking to components

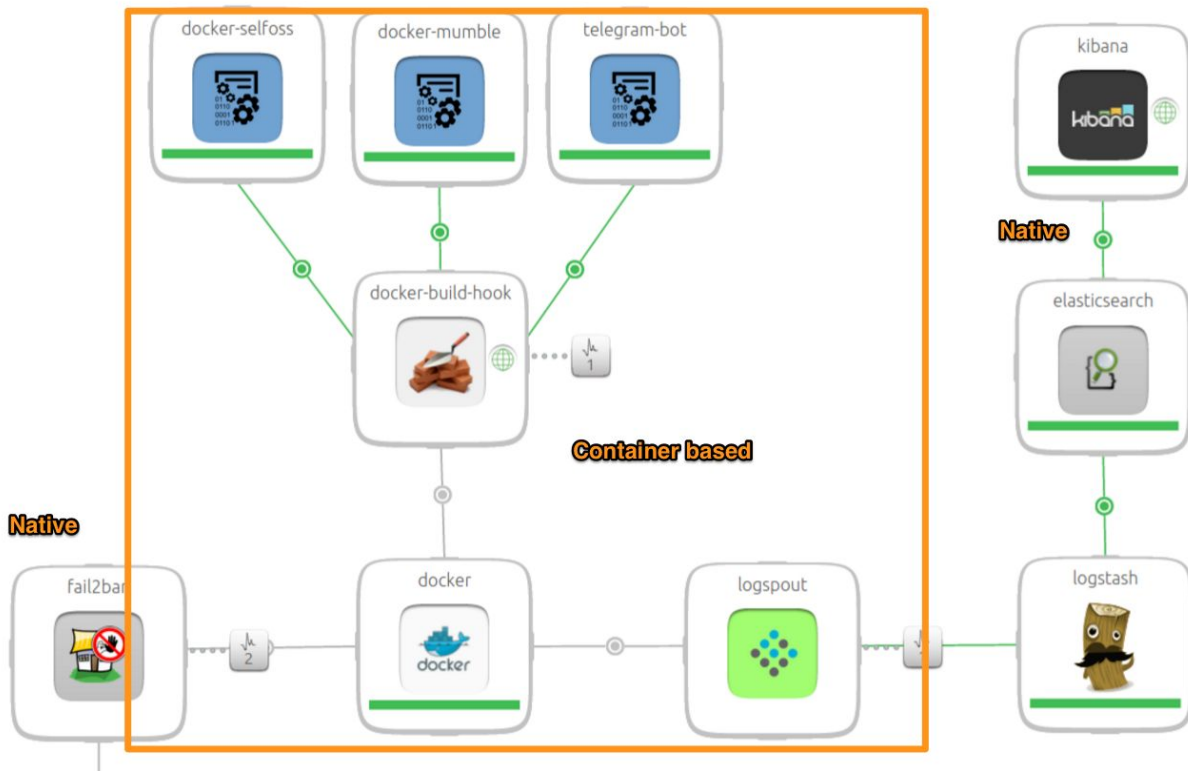
- storage
- database
- networking

that are not in containers, and in some (rare) cases: unable to be placed in a container.

Chuck's Adventure

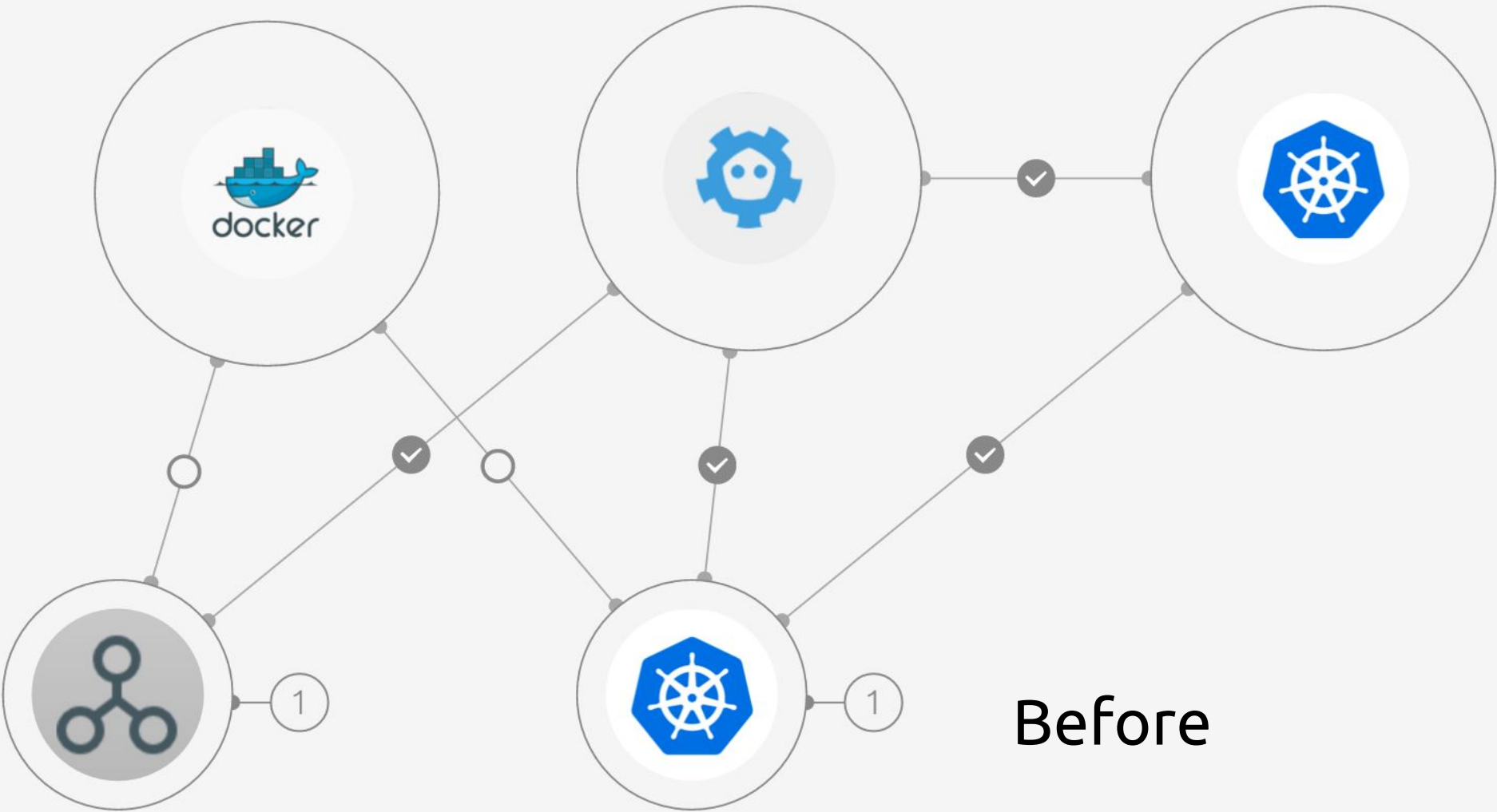


Chuck's Adventure



Delivery Patterns

Application containers vs uncontained delivery



Before

Kubernetes Charm as a Case Study

uncontained Delivery

5317 total LOC

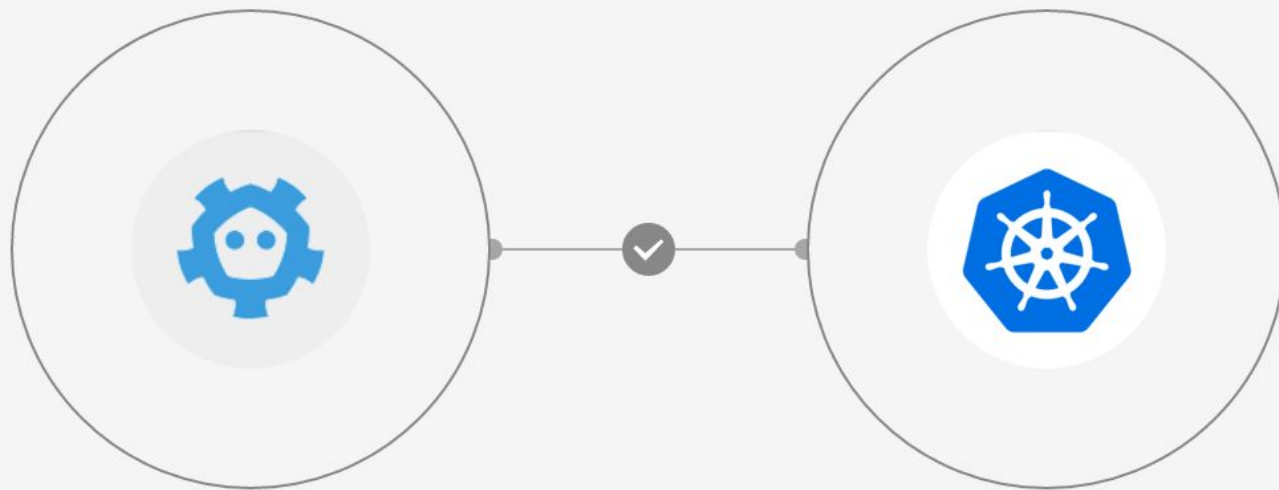
- Required a Build Env
- 15 Min Delivery
- 8 min upgrade cycle
- Different model than suggested by google

containerized Delivery

2283 total LOC

- No Build Env
- 8 Min Delivery
- ~ 1 min upgrade cycle
- Same model suggested by google

3,034 LOC reduction in cost of ownership



After (mid flight)

**It doesn't matter how many
resources you have...**



**If you don't know how to use
them, it will never be enough.**

Take a closer look @ the
Kubernetes Example

layer-docker

- Delivers the latest -Stable engine from Docker's PPA
- Provides a consistent interface to work with charming application containers.
- meaningful synthetic states - `@when('docker.ready')`
- Includes charms.docker

<http://github.com/juju-solutions/layer-docker>

charms.docker

- Configure and interact with a Docker Daemon
 - Manage DOCKEROPTS

```
opts = DockerOpts()
```

```
opts.add('allow-insecure-registry', True)
```

```
opts.to_string()
```

charms.docker

- Interact with a docker-engine

```
from charms.docker import Docker
d = Docker()
pid = d.up('lazypower/idlerpg:latest',
          dirs={"files/idlerpg":"/files/idlerpg"},
          ports=["8000:8000"])
```

charms.docker

- Manage docker-compose templates

```
from charms.docker.compose import Compose
```

```
compose = Compose('files/tikiwiki')
```

```
compose.up('mysql')
```

```
compose.kill()
```

```
compose.rm()
```

<http://github.com/juju-solutions/charms.docker>

Containers as Payloads

Containers as Payloads

- System Containers can be delivered in a similar fashion
 - Pack in a quick-configuration script to carry your CM configuration values into the environment
 - `lxd run /opt/configure_my_service foo=bar baz=bam`
- Generate the pre-configured containers with CM tooling
 - Juju, Chef, Puppet, Ansible, Saltstack, Foreman, CFEngine, or whatever strikes your fancy

LXD ships with everything you need

LXD can act as a hosting image server

- Warehouse base images
- Push container snapshots for migration / distribution
- Trusted Registry by default, they're all your containers

Where is charms.lxd then?

Simply stated:

LXC/LXD is natively supported in Juju. These “primitives” are exposed as a native “machine” to create units for an Application.

These principles work in every CM toolkit



Ansible Modules

<https://github.com/kbrebanov/ansible-lxd>

http://docs.ansible.com/ansible/lxc_container_module.html

Deliver and manage System Containers

http://docs.ansible.com/ansible/docker_module.html

Deliver and manage Application Containers

Chef Cookbooks

<https://supermarket.chef.io/cookbooks/container>

Deliver and manage System Containers

<https://supermarket.chef.io/cookbooks/docker>

Deliver and manage Application Containers

Puppet Modules

<https://github.com/tripledees/sjimenez-lxc>

Deliver and manage System Containers

<https://forge.puppetlabs.com/garethr/docker>

Deliver and manage Application Containers

Salt Stack

<https://docs.saltstack.com/en/latest/topics/cloud/lxc.html>

Create / Manage System Containers

<https://docs.saltstack.com/en/latest/ref/states/all/salt.states.dockerng.html>

Create / Manage Application Containers

Thanks for your time

Come see us @ CFGMGMTCAMP 2016 in Gent

<http://summit.juju.solutions>