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.
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

Convergence

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
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
## Kubernetes Charm as a Case Study

<table>
<thead>
<tr>
<th>Uncontained Delivery</th>
<th>Containerized Delivery</th>
</tr>
</thead>
<tbody>
<tr>
<td>5317 total LOC</td>
<td>2283 total LOC</td>
</tr>
<tr>
<td>- Required a Build Env</td>
<td>- No Build Env</td>
</tr>
<tr>
<td>- 15 Min Delivery</td>
<td>- 8 Min Delivery</td>
</tr>
<tr>
<td>- 8 min upgrade cycle</td>
<td>- ~1 min upgrade cycle</td>
</tr>
<tr>
<td>- Different model than suggested by google</td>
<td>- Same model suggested by google</td>
</tr>
</tbody>
</table>

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
- Configure and interact with a Docker Daemon
  - Manage DOCKEROP TS

def main()
    
    opts = DockerOpts()
    opts.add('allow-insecure-registry', True)
    opts.to_string()

http://github.com/juju-solutions/charms.docker
- Interact with a docker-engine

```python
from charms.docker import Docker

d = Docker()

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

http://github.com/juju-solutions/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/tripledes/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.docker.ng.html

Create / Manage Application Containers
Thanks for your time

Come see us @ CFGMGMTCAMP 2016 in Gent

http://summit.juju.solutions