Migrating an Application into the Cloud with Docker and CoreOS

Zurich University of Applied Sciences

Presenter: Martin Blöchlinger
e-mail: bloe@zhaw.ch
Content

• Introduction
• Why docker, CoreOS?
• CoreOS & Components
  • fleet
  • etcd
• Service Discovery (confd)
• Current Implementation
• Demo
• Questions & Discussion
Introduction

• InIT Cloud Computing Lab

• Initiative: Cloud Native Applications

• Current Project:
  • Transform Business Application to Cloud
  • Open-Source Application not designed for Cloud
  • Application Goal: Resiliency, Scalability
Zurmo Architecture

- **Core:**
  - PHP Application
- **Components:**
  - Webserver
  - Database
  - Cache
Target Architecture

Application Ecosystem

Zurmo CRM
- Load Balancer
- Web Server n
  - Zurmo Application Core
- Web Server 1
  - Zurmo Application Core
- DB Master
- DB Slave
- Cache n
- Cache 1

Monitoring Systems
- System-Monitoring
- Application-Monitoring
- Log Collector

Management Systems
- Health-Management
- Auto-Scaling
- Configuration/Service Discovery

Logs

User
Why docker?

• lightweight
• easy setup of environment
• testing applications on the fly
• fast startup
• fine grained scheduling

• reusable in cloud -> CoreOS
CoreOS

• Linux Operating System
• Intended to be used with docker
• No package manager

• Preinstalled Tools
  • fleet → Run docker containers
  • etcd → Configuration Management
  • docker
fleet

- distributed init system
- based on systemd (init system)
- declare services in fleet unit files
- declare dependencies of services
- declare environment of process
- control lifecycle of long running processes
  - start process/container
  - stop process/container
  - manage process/container
fleet - example unit file

[Unit]
Description=Apache web server service
Before=zurmo_apache_discovery@.service
After=etcd.service
After=docker.service
Requires=zurmo_config.service
Wants=zurmo_apache_discovery@.service

[Service]
TimeoutStartSec=0
Restart=always
EnvironmentFile=/etc/environment
ExecStartPre=/usr/bin/docker kill zurmo_apache
ExecStartPre=/usr/bin/docker rm zurmo_apache
ExecStartPre=/usr/bin/docker pull user_name/zurmo_apache
ExecStart=/usr/bin/docker run --name zurmo_apache -p 80:80 --volumes-from zurmo_config user_name/zurmo_apache
ExecStop=/usr/bin/docker stop zurmo_apache

[X-Fleet]
X-Conflicts=zurmo_apache*.service
MachineMetadata=public=true

---

Process lifecycle management
# ExecStartPost  -> what to do after starting the process
# ExecReload    -> what to do on reload
# ExecStopPost  -> what to do after stopping the process

---

Scheduling constraints
fleet - start/stop services

- **fleetctl**: command line tool

make the service known to fleet

> fleetctl submit <unit_file>

schedule the service on a machine

> fleetctl load <service_name>

start / stop service

> fleetctl start <service_name>
> fleetctl stop <service_name>
fleet - using templates

• In fleet (save as zurmo_apache@.service)

 [...]  
 ExecStartPre=-/usr/bin/docker kill zurmo_apache_%i  
 ExecStartPre=/usr/bin/docker pull user_name/zurmo_apache_%i  
 ExecStart=/usr/bin/docker run --name zurmo_apache_%i  
 user_name/zurmo_apache_%i  

 [...]  

• Start instance based on template

 > fleetctl start <template_name@instance_name.service>  
 > fleetctl start zurmo_apache@0.service
etcd

- distributed key value store
- designed for: shared configuration & service discovery
- implements Raft consensus algorithm
- handles machine failures, master election etc.
- actions: read, write, listen
- data structure
  - /folder
  - /folder/key
- REST-API
- easy to use client: etcdctl
etcd - example

read/write a value
> etcdctl get /folder/key
> etcdctl set /folder/key

read/create directory
> etcdctl mkdir /folder
> etcdctl ls /folder

listen to changes
> etcdctl watch /folder/key
> etcdctl exec-watch /folder/key -- /bin/bash -c "touch /tmp/test"
etcd - service discovery
etcd - service discovery

/services/cache
/services/cache/f7d43e22-2e11-4d29-a5a4-c55741bcae99
/services/cache/f7d43e22-2e11-4d29-a5a4-c55741bcae99/host
/services/cache/f7d43e22-2e11-4d29-a5a4-c55741bcae99/port
/services/cache/f7d43e22-2e11-4d29-a5a4-c55741bcae99/ip
/services/loadbalancer
/services/loadbalancer/9259e4d8-f3a0-4947-9d9f-48b98d731cda
/services/loadbalancer/9259e4d8-f3a0-4947-9d9f-48b98d731cda/port
/services/loadbalancer/9259e4d8-f3a0-4947-9d9f-48b98d731cda/host
/services/loadbalancer/9259e4d8-f3a0-4947-9d9f-48b98d731cda/ip
/services/webserver
/services/webserver/87e60800-9b3f-43f9-875a-0954ac04059a
/services/webserver/87e60800-9b3f-43f9-875a-0954ac04059a/host
/services/webserver/87e60800-9b3f-43f9-875a-0954ac04059a/port
/services/webserver/87e60800-9b3f-43f9-875a-0954ac04059a/ip
[...]
confd - configuration reloading

- tool to rewrite configuration files if etcd key changes
- not part of CoreOS (used in containers)
Current Implementation

Containers & fleet files for:

- HAPProxy
- Apache
- Zurmo Application
- Zurmo Configuration
- Memcached
- MySQL

Sidekicks

- Service Discovery
Demo

Zurmo in Action

• Service Discovery
  • Stopping Service
• Resilience
  • Container failure
  • VM failure
Conclusion / Next Steps

Status

• Running cluster with core of application (OpenStack, Vagrant)
• Resiliency implemented with CoreOS, fleet, etcd, docker and confd
• Now implementing logging

Problems faced

• Time to build/download docker images
• Managing multiple git branches
• Complexity compared to size of application

Future challenges

• Stateful containers (e.g database)
• Scaling (tools like kubernetes)
Links

ICCLab:
   •  http://blog.zhaw.ch/icclab/

Cloud-Native Applications Initiative:
   •  http://blog.zhaw.ch/icclab/category/research-approach/themes/cloud-native-applications/

ZHAW InIT

Docker Hub - Docker Images
   •  https://hub.docker.com/u/icclabcna/

GitHub Repository
   •  https://github.com/icclab/cna-seed-project
Links II

Operating System: CoreOS
CoreOS components: etcd, fleet
Tools: confd
Software in Containers: MySQL, Memcached, Apache HTTP Server, HAPProxy
Web-Services: docker hub, github
Cloud-Software: OpenStack
CRM-Software: Zurmo
Questions