From Bare Metal to Cloud

Andy Edmonds, Fabrice Manhart, Thomas Michael Bohnert, Christof Marti
ICCLab, ZHAW

www.cloudcomp.ch
Hello!

ICCLab
- Zurich University for Applied Sciences
- Cloud Computing Research

"From Research Themes to Projects creating Impact and Transfer for Educational Excellence"
We've Hardware for Cloud!

Nodes: 15
Core: 240
Memory: 960 GB
Storage: 64 TB
Challenges or Problems?

- Clouds are in essence big data centres
  - Means **lots** of servers:
    - Manual configuration **not an option**
    - Automation is **required**
Challenges or Problems?

Cloud frameworks can be complicated!
Challenges or Problems?

- Clouds are "cool" - Aayyy!

**BUT**

- How to deploy a "cloud"
  - with **minimal user interaction**?
  - **least number** of "hands"?
  - across **many servers**?
Challenges or Problems?

- How to **share/standardise** these processes?
  - Configuration - drift prevention
  - Testing - configuration, system functionality
  - Compliance - auditing, ITIL (eeek!)
  - Agility
  - Independence
    - Of physical/virtual deployment
    - Of infrastructure
Automation Toolchain

Create OS/VM Images → Bootstrap Image → Provision → Configuration

Monitor

Service Management start/stop/orch.
Automation Toolchain

Create OS/VM Images → Bootstrap Image → Provision → Configuration

Ubuntu Netboot apt-cache-ng
Foreman
Puppet

Monitor

Service Management start/stop/orch.
Provision - OS rollout

Baremetal  VM

?
Provision - Foreman

● "Single Address For All Machines Lifecycle Management".
● Manages or proxies to DNS, DHCP, TFTP, Virtual Machines, PuppetCA, CMDB
● Integrates with Puppet (and acts as web front end to it).
● Provisions:
  ○ most flavours of *NIX, Windows
  ○ Virtual machines - libvirt, oVirt
  ○ Cloud Resources - Amazon EC2, VMware vCenter, OpenStack
● Has an API! Automate your Automation :-)
Provision - Foreman

What does it look like?
Provision - Foreman Arch
Configuration - Puppet

- Declarative configuration language
  - Describe desired state of a system, not how to achieve it
  - Idempotence
- Different types of resources: software package, service, user, configuration file, mysql database, ...
- Dependencies can be formulated
- Grouping of resources by "class" concept:
  - Way of structuring your descriptions
- Abstraction layer for resources:
  - Independence from system type (different variants of linux, *bsd, mac os, windows, ...)
Configuration - Puppet's Model

current state

desired state

==?

sync

event
You describe system state...

current state

==?

desired state

sync

event

package {'sshd':
  ensure => present,
}
Puppet collects current state...

rmp –q sshd
dpkg-query – search sshd

current state

desired state

==?

sync

event

package {'sshd':
  ensure => present, }

rpm –q sshd
dpkg-query – search sshd
Puppet compares...

```
rpm -q sshd
------------------
dpkg-query - search sshd
```

Puppet compares the current state of the `sshd` package (obtained using `rpm -q sshd` or `dpkg-query - search sshd`) with the desired state (specified in the Puppet configuration). If the current state is `absent` or `!= present`, it triggers a sync event.

```
package {'sshd':
  ensure => present,
}
```
Puppet synchronizes...

- `rpm -q sshd`
- `dpkg-query -search sshd`
- `yum install sshd`
- `apt-get install sshd`
- `rpm --q sshd`
- `dpkg-query --search sshd`
- `yum install sshd`
- `apt-get install sshd`
- `sync`
- `event`
- `current state`
- `desired state`
- `absent` vs `present`
- `!=`
- `package {'sshd': ensure => present, }`
Puppet logs...

rpm –q sshd
dpkg-query -- search sshd

current state

desired state

package {'sshd':
  ensure =>
present,
}

absent

state transition: absent -> present

sync

event

rpm –q sshd
--------------------

dpkg-query – search sshd

yum install sshd
------------------------

apt-get install sshd

Puppet logs...
A more complete puppet manifest

```puppet
class ssh::install {
  package { "openssh":
    ensure => present,
  }
}
class ssh::config {
  file { "/etc/ssh/sshd_config":
    ensure    => present,
    owner     => 'root',
    group     => 'root',
    mode      => 0600,
    source    => "puppet:///modules/ssh/sshd_config",
    require   => Class["ssh::install"],
    notify    => Class["ssh::service"],
  }
}
class ssh::service {
  service { "sshd":
    ensure       => running,
    hasstatus    => true,
    hasrestart   => true,
    enable       => true,
    require      => Class["ssh::config"],
  }
}
class ssh {
  include ssh::install, ssh::config, ssh::service
}
```
OpenStack @ 10,000m, Looks Easy!

Everything has an API
Message-based
Discrete Pluggable Components

Essex was simpler!
OpenStack - The Ugly Close-up

Complicated
● Many Services
● Many Dependencies

Challenge to deploy
● 100's, 1000's of nodes?

You need an automated toolchain!
Apple Moment!

We have one! Want to see?
Demo - What Could Go Wrong?!  
Multi-node OpenStack Installation

- 1 controller node
  - "boss"

- 1 compute node
  - "worker1"

- More time? Easy to add more.
Demo: Deployment Architecture
Demo: OpenStack Component Deployment

OpenStack, EC2 and OCCI APIs

Controller VM
- nova-api
- nova-scheduler
- nova-console-auth
- nova-cert
- glance
- keystone
- rabbitmq, MySQL

Compute VM
- nova-compute
- nova-network
- nova-storage

Easy to add more
Just provision a new host with a "openstack/compute" hostgroup role
Demo: Code/Config Details

● There are 2 roles (hostgroups)
  ○ openstack/controller - controller.pp
  ○ openstack/compute - compute.pp

● Both have different puppet manifests
  ○ Same 'icclab' module
What's in a controller node? (excluding common params)
What's in a compute node?
(excluding common params)
In the ICCLab

- 15 nodes in ~15 minutes.
- Supports 2 networks
  - research (essex, soon to be folsom)
  - stable (essex)
Conclusions/Learnings

- Automation is **essential**
- Puppet **codifies** learnings, makes **sharing** easy
- Foreman a **central** management point, full lifecycle, adaptable to other services
- Dependence on infrastructure service management frameworks is lessened
  - Fast and efficient to install new ones with a tool chain
- Other than SLA guarantees, the only guarantee to maintain is the API between provider and customer and this is where standard APIs are need such as OCCI/CDMI/OVF.
Going Forward...

- Folsom to be Deployed
  - Only on research network
  - Essex remains on stable
- High Availability to be implemented
- Deploy and use Ceilometer
- Develop further Hadoop as a Service
- Consider/deploy other host OSes (SmartOS)
- Integrate OpenFlow switches
- ICCLab to run/support:
  - EU FP7 IP MobileCloud Network
  - EU FI-PPP KIARA (FIware)
  - KTI projects
Thanks!
Questions?
(ping me: @dizz, edmo@zhaw.ch)

**Everything** Presented is Documented at:
http://www.cloudcomp.ch

*Including:*
- HOWTOs
- Foreman, Puppet, OpenStack installs
  - Virtual Machine images
Backup Slides
Puppetmaster <-> agent interaction

Agent
- Request catalog
  (sends node name and facts)
- Catalog
- Apply
  - Query status
  - Enforce defined state
  - Defined system state

Master
- Classify
  (Who is this, and what do they need?)
- Class
- Class
- Class
- Compile
- Report
What are the common config params?

class icclab::params{
    /* ---------------Shared Connection Settings-------------*/
    ####### Important to set! #######
    $controller_node_address = '192.168.56.3'

    $controller_node_public = controller_node_address
    $controller_node_internal = controller_node_address
    $sql_connection = "mysql://nova:${icclab::params::nova_db_password}@${controller_node_internal}/nova"

    /* -------------------Shared Auth Settings----------------------*/
    $nova_user_password = 'nova_pass'
    $rabbit_password = 'rabbit_pass'
    $rabbit_user = 'rabbit_user'

    /* ----------------Shared Networking Settings----------------------*/
    $network_manager = 'nova.network.manager.FlatDHCPManager'
    $fixed_range = '10.0.0.0/24'
    $public_interface = 'eth0'
    $private_interface = 'eth1'

    */
}