From Bare Metal to Cloud

Andy Edmonds, @dizz, ICCLab, ZHAW
Piotr Kasprzak, GWDG
Intros

ICCLab
- Zurich University for Applied Sciences
- Cloud Computing Research

GWDG
- Service Provider for Max Planck Society and University of Goettingen
- Research
We've Hardware for Cloud!

**GWDG Cloud Hardware**

- Nodes: 38
- CPUs: 152
- Core: 2432
- Memory: 9728 TB

**ICCLab Cloud Hardware**

- Nodes: 20
- CPUs: 80
- Core: 1280
- Memory: 1920 TB
Challenges or Problems?

- Clouds in essence are 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?

● But 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
  - 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
Provision - OS rollout

- Fedora
- Ubuntu
- SUSE
- [Unknown]

Choice:
- 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
- Has an API! :-)
Provision - Foreman Arch
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 ➔ ==? ➔ sync ➔ event

desired state ➔ ==?
You describe system state...

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

current

state

desired

state

==?

sync

event
Puppet collects current state...

```
rpm -q sshd
-------------
dpkg-query -s sshd
```

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

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

Puppet compares the current state with the desired state.

- `rpm -q sshd` checks if `sshd` is present.
- `dpkg-query --search sshd` searches for `sshd`.

If the `sshd` package is **absent** or **present** differently than expected, a **sync event** is triggered.

Desired state:
```
package {‘sshd’: ensure => present, }
```
Puppet synchronizes...

Puppet synchronizes the current state of the package `sshd` with the desired state. If the current state is `absent`, it performs an operation to synchronize with the `desired state` being `present`. The operations to achieve this are:

- `rpm -q sshd`
- `dpkg-query -search sshd`
- `yum install sshd`
- `apt-get install sshd`

The package is ensured to be present in the desired state. The synchronization event triggers the necessary commands to achieve the desired state.
Puppet logs...

rpm –q sshd
---------------------
dpkg-query – search sshd

package {‘sshd’: ensure => present, }

rpm –q sshd
---------------------
dpkg-query – search sshd

yum install sshd
-----------------------------
apt-get install sshd

state transition:
absent -> present
A more complete puppet manifest

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!

Software as a Service
Platform as a Service
Infrastructure as a Service

Everything has an API
Message based
Discrete Pluggable Components
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!

BOOM!

TOOLCHAIN!
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?

class icclab::controller{
  include icclab::params

  $admin_password   = 'admin_pass'
  $keystone_admin_token = 'keystone_pass'

  class { 'openstack::controller':

    public_address   => $icclab::params::controller_node_public,
    public_interface => $icclab::params::public_interface,
    private_interface => $icclab::params::private_interface,
    internal_address => $icclab::params::controller_node_internal,
    floating_range  => '192.168.56.128/25',
    fixed_range     => $icclab::params::fixed_range,
    multi_host      => true,
    network_manager => $icclab::params::network_manager,
    verbose         => true,
    auto_assign_floating_ip => false,
    mysql_root_password => 'mysql_root_password',
    admin_email     => 'admin@ownz.you',
    admin_password  => $admin_password,
    keystone_db_password => 'keystone_db_password',
    keystone_admin_token => $keystone_admin_token,
    glance_db_password => 'glance_pass',
    glance_user_password => 'glance_pass',
    nova_user_password => 'nova_pass',
    nova_user_password => $icclab::params::nova_user_password,
    rabbit_password => $icclab::params::rabbit_password,
    rabbit_user     => $icclab::params::rabbit_user,
    export_resources => false,
  }

  # Optional: include if you want authorisation information
  # stored in a local file, located in /root/
  class { 'openstack::auth_file':

    admin_password => $admin_password,
    keystone_admin_token => $keystone_admin_token,
    controller_node => $icclab::params::controller_node_internal,
  }
}
What's in a compute node?

class iclab::compute{

  include iclab::params

class {
  'openstack::compute':

    public_interface => iclab::params::public_interface,
    private_interface => iclab::params::private_interface,
    internal_address => $ip_address_eth0,
    libvirt_type => 'kvm',
    fixed_range => iclab::params::fixed_range,
    network_manager => iclab::params::network_manager,
    multi_host => true,
    sql_connection => iclab::params::sql_connection,
    nova_user_password => iclab::params::nova_user_password,
    rabbit_host => iclab::params::controller_node_internal,
    rabbit_password => iclab::params::rabbit_password,
    rabbit_user => iclab::params::rabbit_user,
    glance_api_servers => "${iclab::params::controller_node_internal}:9292",
    vncproxy_host => iclab::params::controller_node_public,
    vnc_enabled => true,
    verbose => true,
    manage_volumes => true,
    nova_volume => 'nova-volumes'
  }
}
}
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.
Next Steps

● OpenStack to be rolled-out in ICCLab
  ○ New data centre, rolled-out within the month
  ○ Will include all OS Nova (Essex) and Swift services
    ■ Including OCCI interface
      ● puppetlab-nova pull-request available

● OpenStack to be rolled-out in GWDG
  ○ Will include all OS Nova (Essex) and Swift services
  ○ Providing production-quality OpenStack services
Thanks!
Questions?

**Everything** Presented is Documented at:

http://www.cloudcomp.ch
http://cloud.gwdg.de

Including:
- HOWTOs
- Foreman, Puppet, OpenStack installs
  - Virtual Machine images
Backup slides
Toolchain map

- Dashboard
- ITIL CMDB
- Config-DB (SQL)
- Git
- Puppet Master
- Puppet Agent
  - VM
- Puppet Agent
  - Host
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

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'

    /* ---------------------*/
GWDG Cloud topology

- SAN
  - Fabric A
  - Fabric B
- Core
- EoR/MoR
  - End/Middle of Row
- ToR
  - Top of Rack
- CNA und FCoE
  - Converged Network Adapter/
    Fiber Channel over Ethernet