Date printed: 7/17/2014

Event Listener based UCS Integration with OpenStack

BETA

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

Table of Contents

BETA

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

List of Tables and/or List of Figures are optional and may be deleted.

BETA

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

1 Overview

An OpenStack installation comes with its challenges – from OS installation to the right modules being loaded on the nodes. Tools from various vendors that help deliver a smooth setup process have addressed most of these challenges. This, of course, assumes that the bare metal is already setup and running outside the purview of OpenStack itself. Cisco UCS addresses the challenges associated with bare metal setup by delivering a policy driven framework and an open, fully functional XML API allowing seamless integration with tools.

This document explores and addresses the integration of bare metal bring-up and the subsequent setup of Applications (like OpenStack, cobbler, etc). The Integration occurs in two stages:

- ✓ The First stage configures UCS bare metal using the Python SDK
- ✓ Next stage starts an event listener that listens for the change events of service-profiles. Based on the event type, it either updates/removes the host details in the integrated application

The result is a painless, one-step scripted auto configuration of an application on a completely new UCS setup.

This integration script quickly configures UCS Manager with required policies so that it automatically discovers chassis and servers, and creates and associates service-profiles with the servers. This script also starts the event listener program in the integration application. This event listener adds the UCS servers to an Application cluster after it receives the association complete event from UCS Manager.

Then it also does 'puppet apply' with the updated site.pp file.

So the servers can automatically PXE boot from the Host (Build-node in case of OpenStack, or DHCP server used with Cobbler setup).

BETA

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

1.1 Integration architecture

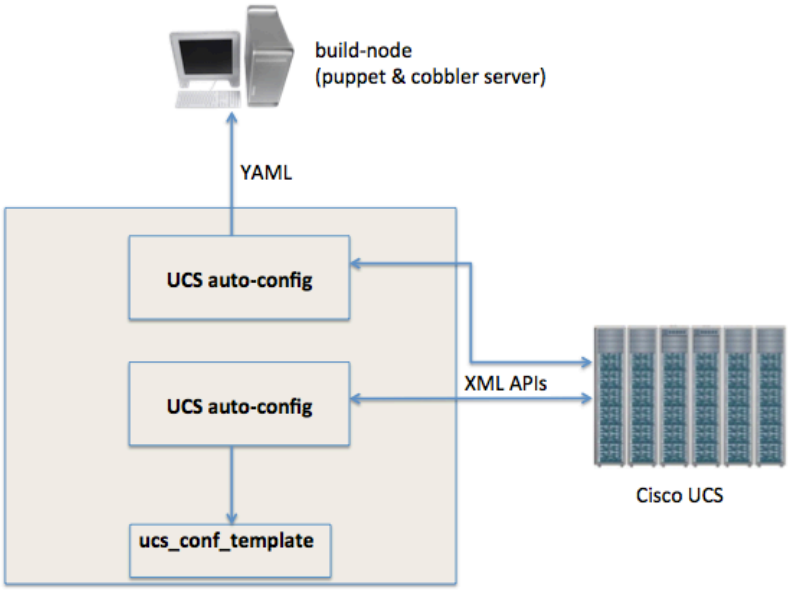


Fig 1. Auto-config UCS to integrate with OpenStack

There are multiple components (scripts) in this Integration. The details of the scripts and their purpose are explained below:

ucs_integration.py: This is the main integration script that a user would execute from command line. It does UCSM configuration and starts the event listener to listen for change events from UCSM.

SPConsumer.py: This script has the implementation of different types of service-profile consumers (Integration applications). It holds the application-specific code to add/update/remove nodes in their applications.

UcsUtils.py: This is utility script is used to configure policies in UCSM. Also used for retrieving the information related to service-profiles.

CobblerUtils.py: This script is specific to the UCSM integration with Cobbler. This utility is used to update the Cobbler DB and also to do 'cobbler sync' operations.

ConfUtils.py: This holds some config variable used across the integration scripts.

IPList.py: This is the key script that a user has to edit before starting the integration process. The file holds the information of different node types supported by OpenStack as well as service-profile to hostname mapping. There is no direct way to specify the hostname and ip-address from UCSM, so for now, we've chosen this method. The sample configuration is provided in section 5.

ucs_conf_template.conf: This is the most important configuration file a user will modify. It contains the policies, ports, VLANs and other configuration information that has to be configured in UCSM as part of the integration. Please refer to section 6 for details with examples.

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

The solution is divided into two steps with the intention that it can be used with applications beyond those documented here.

Step 1: Configure UCSM with required policies, VLANs, ports, and other required configuration. This step is optional, In case the UCSM is already configured with all the policies.

Step 2: Start event listener. This listens for the service-profile change events. Based on the event type, the listener updates the service-profile consumer (OpenStack, Cobbler, Hadoop etc) DB and also does other required operations like 'puppet apply', 'cobbler sync', etc.

These integration scripts use the UCS python SDK to communicate with the UCS Manager. The communication uses XML APIs over http(s) protocol.

On the other side, YAML is used to update the nodes info in a temporary DB. Later, this DB is used to generate the final os_site.pp to be used by puppet.

Jinja is used to generate the site.pp based on the template site.pp.template file with the node values in DB.

2 Installation

2.1 Pre-install check list

- ✓ Folsom OpenStack setup, with build-node (Please refer to Cisco OpenStack setup instructions available here).
- ✓ New or pre configured UCS System.
- ✓ Python 2.7.3 or later version.
- ✓ Daemon, logging, YAML, jinja2, passlib.apps python modules
- ✓ UCS Python SDK version 0.4 or later installed on Build server or on a separate server. The Python SDK can be downloaded from here (http://developer.cisco.com/web/unifiedcomputing/sdk).
- ✓ UCS admin account login credentials to configure the policies on UCS Manager.
- ✓ Update site.yaml @ /etc/puppet/manifests with the existing OpenStack cluster details. Along with control-node details.

Note: This integration was tested with Ubuntu 12.04.1 distribution OS versions and Folsom OpenStack version.

2.2 OpenStack and network requirements

- ✓ OpenStack Folsom installation with build-node.

BETA

- ✓ It is expected that the build-node will act as both the DHCP server and Image server
- ✓ It is expected that the build-node is on the same subnet as the servers which are trying to network boot.

2.3 UCS config template requirements

- ✓ It is expected that the ucs_config_template file has been updated with the required values. Please refer to section 6 for sample configuration and detailed use cases.

2.4 Installation Steps

- ✓ Please read carefully the terms and conditions of the End User License Agreement before installing or using the plug-in.
- ✓ Run the ucs_integration.py script from the build-node by providing ucsm host and login details along with configuration file location. If UCSM is already configured, run ucs_integration.py script with –listener-only=True option.
- ✓ This program also starts an event listener process in the background.

3 How the integration script works

The integration scripts use the UCS python SDK to communicate with the UCS Manager.

Stage 1:

First, it reads the ucs_config_template, and configures these policies on UCS Manager.

Once it completes the configuration, the UCS Manager starts chassis and server discovery. It automatically creates the service-profiles based on the server qualifications.

Or, if the UCSM has all chassis and servers discovered, the user can just create N number of service profiles associated with server pool. It automatically associates service-profiles to servers if and when servers are available in the server pool. This configuration can be specified in the ucs_config_template file.

This stage can be skipped if the UCSM is already configured with these policies and you just want to listen for change events and respond to the event types. User can use '—listener-only=True' option while executing ucs_integration.py to skip configuring the policies.

BETA

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

Stage 2:

After the script completes the configuration of policies in UCSM, it starts an event listener process in the background. The event listener will listen for the lsServer events. Once the server association is complete, it adds the servers to local nodes DB on build-node and also generates the new os_site.pp config based on local nodes DB and site.pp.template.

The next step is applying the new configuration on to Puppet master (running on build-node), using 'puppet apply –v site.pp'.

It repeats the steps; updating config file and 'puppet apply' for each server it adds to OpenStack cluster.

Note: puppet apply will take care of adding the system entries on to Cobbler DB.

User needs to update the service-profile to hostname, ip-address, and node-type mapping in IPList:iplist.

4 Known limitations

- Renaming of service-profiles is not supported.
- service-profile to ip-address mapping is not dynamically handled.

5 Hostname mapping for OpenStack integration

One of the limitations listed in section 4 is ip address and hostname mapping. There is no direct option to specify hostname, ip address in service-profile. This info is maintained by in iplist.yaml file.

This file contains the list of node types supported in OpenStack. Also maintains service-profile to hostname, ip address, and node-type mappings.

The sample content of this file is provided below.

```
iplist:
  alpha-control01:
    name: alpha-control01
    ip: 172.29.75.198
    type: control
  alpha-compute01:
    name: alpha-compute01
    ip: 172.29.75.199
    type: compute
  alpha-storage01:
    name: alpha-storage01
```

BETA

ip: 172.29.75.207

type: storage

iplist::iplist holds the service-profile mappings. Here 'alpha-control01', 'alpha-compute01', and 'alpha-storage-proxy01' are service–profile names. It maps to the hostname, ip address and node-type being used while adding the node to OpenStack cluster.

iplist::nodetypes holds the different types of nodes that are supported by OpenStack. The valid values are:

'build:', 'compute:', 'control:', ' storage:', 'storageproxy:', 'computeproxy:'

NOTE: Similar to this representation, update this file contents with service-profile to hostname, ip address node-type mappings. These details are used while adding any node to the OpenStack cluster.

6 Use cases and sample configuration

There are three different use cases supported as part of the integration. This section explains each one of the use cases. But, before we start learning about the use cases, let's dive into the configuration file details and different configuration parameters used. Refer section 9 for detailed description of the policies.

6.1 ucs_conf_template.conf

This section explains the configuration policies used to configure in UCSM.

NOTE: The keys shown in the key value pair is case sensitive. And policy definition should be in single line.

MacPool: Mac pool of addresses used while associating with the service-profile. This mac pool is specified in the vnic definitions.

Ex: MacPool {'Name':'compute-nics', 'From':'00:25:B5:F2:00:00', 'To':'00:25:B5:F2:00:FF'}

User has to modify to appropriate values they wish to use.

IpPool: mgmt. ip pool used to access the server KVM, vMedia, and SoL sessions associated with the service- profiles. If user wish to use separate ip-pool for compute, storage, control etc, user have to add one entry for these definitions. By

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

default ext-mgmt pool is used, if no pool is specified. Configuring this pool alone would be sufficient incase user doesn't want to use separate ip pools.

Ex: IpPool {'Name':'compute-nodes-ip', 'From':'172.20.231.49', 'To':'172.20.231.67', 'DefGw':'172.20.231.1', 'PrimDns':'172.29.74.154', 'SecDns':'172.29.74.155', 'Subnet':'255.255.255.128'}

IpPool {'Name':'ext-mgmt', 'From':'172.20.231.89', 'To':'172.20.231.104', 'DefGw':'172.20.231.1', 'PrimDns':'172.29.74.154', 'SecDns':'172.29.74.155', 'Subnet':'255.255.255.128'}

StorageLocalDiskConfigPolicy: This policy is used to configure the local-disks on the server with RAID configuration.

Ex:

StorageLocalDiskConfigPolicy { 'Name':'openstack-nodes', 'Mode':'raid-striped', 'ProtectConfig':'no'}

The supported configuration values for Mode are:

Value	**Description**
no-local-storage	: No Local Storage
raid-striped	: RAID 0 Striped
raid-mirrored	: RAID 1 Mirrored
any-configuration	: Any Configuration
no-raid	: No RAID
raid-striped-parity	: RAID 5 Striped Parity
raid-striped-dual-parity	: RAID 6 Striped Dual Parity
raid-mirrored-striped	: RAID10 Mirrored and Striped
raid-striped-parity-striped	: RAID 50 Striped Parity and Striped
raid-striped-dual-parity-striped	: RAID 60 Striped Dual Parity and Striped

This local disk config policy is specified in the service-profile definition.

ServiceProfile: This is server-profile template created for each-node type.

Ex: ServiceProfile {'Name':'control-node', 'Type':'updating-template', 'BootPolicyName':'OpenStackDef', 'Uuid':'derived', 'HostFwPolicyName':'default',

BETA

'ScrubPolicyName':'control-nodes', 'ExtIPPoolName':'control-nodes-ip', 'LocalDiskPolicyName': 'openstack-nodes'}

Details of parameters:

Name: Name is of the service-profile template. Choose the name appropriately for different node types.

Type: Type of the template, it is either 'initial-template' or 'updating-template'. In case of updating template any updates to the service-profile template results in config change on the service-profiles derived from this template.

BootPolicyName: Boot policy to be used. BootPolicy section explains more about boot-policy options.

Uuid: UUID identification pool. For now its derived uses the hardware default value of the server.

HostFwPolicyName: Host firmware update policy. No explicit policy creation support is added. Uses default policy value.

ScrubPolicyName: Scrub policy, used to scrub the disk/boot order once the service-profile is disassociated from server.

ExtIPPoolName: Ip pool used for mgmt address of the server.

LocalDiskPolicyName: Local disk policy to be used. Explained in the above sections about this policy.

LsRequirement: This is server pool to service-profile template using qualifier.

Ex: LsRequirement {'SrcTemplDn':'org-root/ls-control-node', 'Name':'control-pool', 'Qualifier':'control-pool'}

Details of parameters:

SrcTemplDn: service-profile template dn to be used for instantiation of service-profiles.

Name: Name of the server-pool to be used with the service-profile.

Qualifier: Is the match criteria that decides, whether given server is qualified to be associated from server-profile Instantiated from SrcTemplDn.

BETA

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

ComputeAutoconfigPolicy: This auto-config policy creates service-profiles automatically and associates them with servers once the discovery is completed. If the servers were already discovered, this will not have any effect.

Ex:

ComputeAutoconfigPolicy {'Name':'compute-nodes', 'DstDn':'org-root', 'Qualifier':'compute-pool', 'SrcTemplName':'compute-node' }

Details of parameters:

Name: is the name of the auto-config policy.

DstDn: destination OrgDN, where the service-profiles will be placed.

Qualifier: service-profile will be instanted only if server matches this server qualification.

SrcTemplName: service-profile template to be used to instance the service-profiles.

Vlan: Named VLAN configuration on the fabric-interconnects.

Ex: Vlan { 'Name':'Alpha-756', 'SwitchId':'dual', 'Id': '756', 'Sharing':'none', 'DefaultNet':'no'}

Details of the parameters:

Name: name of the VLAN.

SwitchId: switch id on which the LVAN would be created.

Sharing: If it is a shared VLAN.

DefaultNet: yes or no, is this the default VLAN.

Vnic: Vnic

Ex: Vnic { 'Name':'eth0', 'IdentPoolName':'compute-nics', 'VlanName':'Alpha-756', 'SwitchId':'A', 'LsServer':'compute-node', 'DefaultNet':'yes'}

Details of the parameters:

Name: Name of the Vnic

IdentPoolName: the mac pool to be used to derive the Vnic identity.

VlanName: Name of vlan to be associated with this Vnic.

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

SwitchId: Switch Id

LsServer: service-profile associated with the Vnic.

DefaultNet:

ServerPool: Server pool contains set of servers. Server qualifier is used to specify the servers that are being added to this pool.

Ex: ServerPool {'Name':'compute-pool'}

Details of the parameters:

Name: Name of the server pool.

ServerQualifier: Server Pools can be built manually from individual blades, or they can be populated automatically through Server Pool Policy Qualifiers.

Ex: ServerQualifier {'Name':'control-pool', 'MinId':'1', 'MaxId':'1', 'SlotMinId':'7', 'SlotMaxId':'7'}

Details of the parameters:

Name: name of the server qualifier.

MinId: minimum chassis ID.

MaxId: maximum chassis ID.

SlotMinId: Slot minimum ID.

SlotMaxId: Slot maximum ID.

ServerPoolingPolicy: This policy is invoked during the server discovery process. It determines what happens if server pool policy qualifications match a server to the target pool specified in the policy. If a server qualifies for more than one pool and those pools have server pool policies, the server is added to all those pools.

Ex: ServerPoolingPolicy {'Name':'compute-nodes', 'Qualifier':'compute-pool', 'PoolDn':'org-root/compute-pool-compute-pool'}

Details of the parameters:

Name: name of the server pooling policy.

Qualifier: ServerQualifier used to determine the qualification.

BETA

PoolDn: the server pool to which this pooling policy is associated with.

ScrubPolicy: This policy determined whether to scrub the disk and BIOS settings once the server is diassociated from the service-profile.

Ex: ScrubPolicy {'Name':'compute-nodes', 'BiosSettingsScrub':'no', 'DiskScrub':'yes'}

Details of the parameters:

Name: name of the scrub policy.

BiosSettingsScrub: yes or no, whether to scrub or not to scrub the BIOS settings on disassociation of service-profile.

DiskScrub: yes or no, whether to scrub or not to scrub the disk settings on disassociation of service-profile.

ServiceProfileInstance: This is used to create the N no.of service profiles based on the service-profile template.

Ex: ServiceProfileInstance {'SrcTempl':'compute-node', 'NamePrefix':'alpha-compute0', 'NumberOf':'8', "TargetOrg":"org-root/org-Production"}

Details of the parameters:

SrcTempl: service-profile template to be used to create service profile instances.

NamePrefix: The name prefix used in the service-profile naming.

NumberOf: no.of service profiles need to be created.

TargetOrg: the target org dn where the service-profiles are created.

ServerPort: Server ports handle data traffic between the fabric interconnect and the adapter cards on the servers. You can only configure server ports on the fixed port module. Expansion modules do not include server ports.

Ex: ServerPort {'PortId':'1', 'SlotId':'1', 'SwitchId':"A"}

Details of the parameters:

PortId: the Port ID

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

SlotId: the slot ID

SwitchId: The fabrics interconnect ID on which this port will be activated.

Note: User has to update/add the entries based on their setup configurations.

6.2 Use cases:

The list of use cases supported is explained below.

Auto configuration policy: creates service-profiles automatically during the server discovery. This is more suitable to bring up a new UCS with no policies configured. Auto configuration policy automatically creates service-profiles and associates them with servers after discovery. In this case, the user doesn't have to pre-create service-profiles.

The policy user has to modify or add is **'ComputeAutoconfigPolicy'**.

Ex:

ComputeAutoconfigPolicy {'Name':'compute-nodes', 'DstDn':'org-root', 'Qualifier':'compute-pool', 'SrcTemplName':'compute-node' }

This definition adds an auto config policy named 'compute-nodes'. This creates the service and places them under DstDn org which is 'org-root; in this case. The server qualifier is 'compute-pool' and service-profiles are instantiated from 'SrcTemplName' template, which is 'compute-node'.

Add more entries for different types of nodes.

ServerProfileInstance: This allows the user to create N number of service profiles in UCSM associated with server-pool. It automatically associates the service-profiles with the servers when they are available. This is more suitable in cases where UCSM is already configured with required policies and servers are already discovered. Auto configuration policy would not create service-profiles if the servers were already discovered.

The policy user has to modify or add is **'ServiceProfileInstance'.**

Ex:

ServiceProfileInstance {'SrcTempl':'compute-node', 'NamePrefix':'alpha-compute0', 'NumberOf':'8', "TargetOrg":"org-root/org-Production"}

BETA

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

The source template used to create service-profiles instances is 'SrcTempl', which is 'compute-node'. NumberOf parameter, how many service-profiles user wants to create? So, SrcTempl would be associated with server-pool, all the service profiles created with this policy are associated with servers in this server-pool.

Add more entries for different types of nodes.

Listener Mode: In cases where UCSM is already configured with the required policies as specified in the above use cases, or the user has explicitly created them, using the Listener mode would simply update the node inventory and listens for the change events.

7 Execution status

7.1 Auto config UCS

```
root@alpha-build:~/UcsInt-0.2# /usr/bin/python ./ucs_integration.py --ucsm dc1-r2-ucsm.ctocllab
.cisco.com --ucsm_user admin --ucsm_password [redacted] --debug=True --app-name openstack --liste
ner-only=True
root@alpha-build:~/UcsInt-0.2#
```

BETA

7.2 Sample ucs_conf_template file

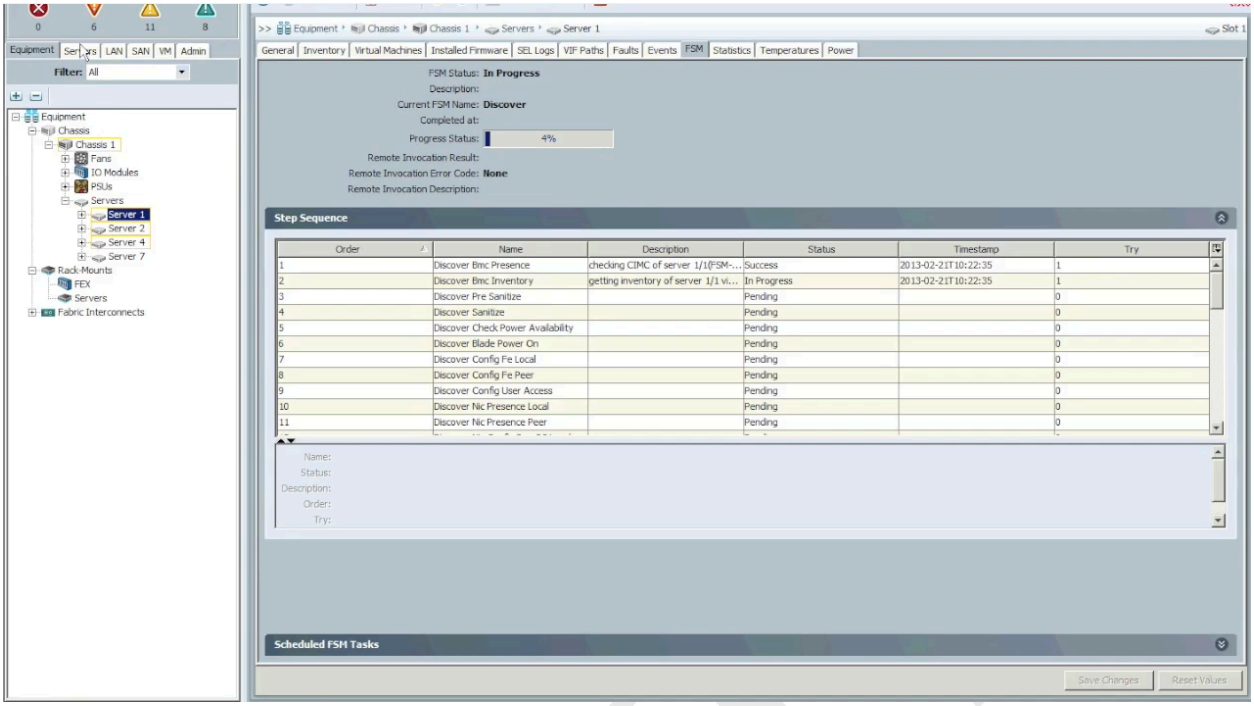
1. ssh

```
MacPool {'Name':'compute-nics', 'From':'00:25:b5:44:55:66', 'To':'00:25:b5:44:55:69'}
#MacPool {'Name':'control-nics', 'From':'00:25:b5:44:55:70', 'To':'00:25:b5:44:55:73'}
IpPool {'Name':'compute-nodes-ip', 'From':'10.105.214.175', 'To':'10.105.214.178', 'DefGw':'10.
105.214.1', 'PrimDns':'72.163.128.140', 'SecDns':'171.70.168.183', 'Subnet':'255.255.255.0'}
IpPool {'Name':'ext-mgmt', 'From':'10.105.214.243', 'To':'10.105.214.246', 'DefGw':'10.105.214.
1', 'PrimDns':'72.163.128.140', 'SecDns':'171.70.168.183', 'Subnet':'255.255.255.0'}
#IpPool {'Name':'control-nodes-ip', 'From':'10.105.214.177', 'To':'10.105.214.178', 'DefGw':'10
.105.214.1', 'PrimDns':'72.163.128.140', 'SecDns':'171.70.168.183', 'Subnet':'255.255.255.0'}
#ServiceProfile {'Name':'control-node', 'Type':'initial-template', 'BootPolicyName':'default',
'Uuid':'derived', 'HostFwPolicyName':'default', 'ScrubPolicyName':'control-nodes', 'ExtIPPoolNa
me':'compute-nodes-ip'}
ServiceProfile {'Name':'compute-node', 'Type':'initial-template', 'BootPolicyName':'default', '
Uuid':'derived', 'HostFwPolicyName':'default', 'ScrubPolicyName':'compute-nodes', 'ExtIPPoolNam
e':'control-nodes-ip'}
ComputeAutoconfigPolicy {'Name':'compute-nodes', 'DstDn':'org-root', 'Qualifier':'compute-pool'
, 'SrcTemplName':'compute-node' }
#ComputeAutoconfigPolicy {'Name':'control-nodes', 'DstDn':'org-root', 'Qualifier':'control-pool
', 'SrcTemplName':'control-node' }
UplinkPort {'SwitchId':'A', 'SlotId':'1', 'PortId':'8', 'AdminSpeed':'1gbps'}

Vlan { 'Name':'vlan106', 'SwitchId':'dual', 'Id': '106', 'Sharing':'none', 'DefaultNet':'yes'}
Vnic { 'Name':'compute-vnic', 'IdentPoolName':'compute-nics', 'VlanName':'vlan106', 'SwitchId':
'A', 'LsServer':'compute-node', 'DefaultNet':'yes'}
#Vnic { 'Name':'controller-vnic', 'IdentPoolName':'control-nics', 'VlanName':'vlan107', 'Switch
Id':'A', 'LsServer':'control-node', 'DefaultNet':'yes'}

ServerPool {'Name':'compute-pool'}
#ServerPool {'Name':'control-pool'}
ServerQualifier {'Name':'compute-pool', 'MinId':'1', 'MaxId':'1', 'SlotMinId':'1', 'SlotMaxId':
'6'}
#ServerQualifier {'Name':'control-pool', 'MinId':'1', 'MaxId':'1', 'SlotMinId':'7', 'SlotMaxId'
:'8'}
ServerPoolingPolicy {'Name':'compute-nodes', 'Qualifier':'compute-pool', 'PoolDn':'org-root/com
pute-pool-compute-pool'}
#ServerPoolingPolicy {'Name':'control-nodes', 'Qualifier':'control-pool', 'PoolDn':'org-root/co
mpute-pool-control-pool'}
ScrubPolicy {'Name':'compute-nodes', 'BiosSettingsScrub':'no', 'DiskScrub':'yes'}
#ScrubPolicy {'Name':'control-nodes', 'BiosSettingsScrub':'no', 'DiskScrub':'yes'}

# create server-ports first.
# server-prots config goes here.
ServerPort {'PortId':'1', 'SlotId':'1', 'SwitchId':"A"}
ServerPort {'PortId':'1', 'SlotId':'1', 'SwitchId':"B"}
```

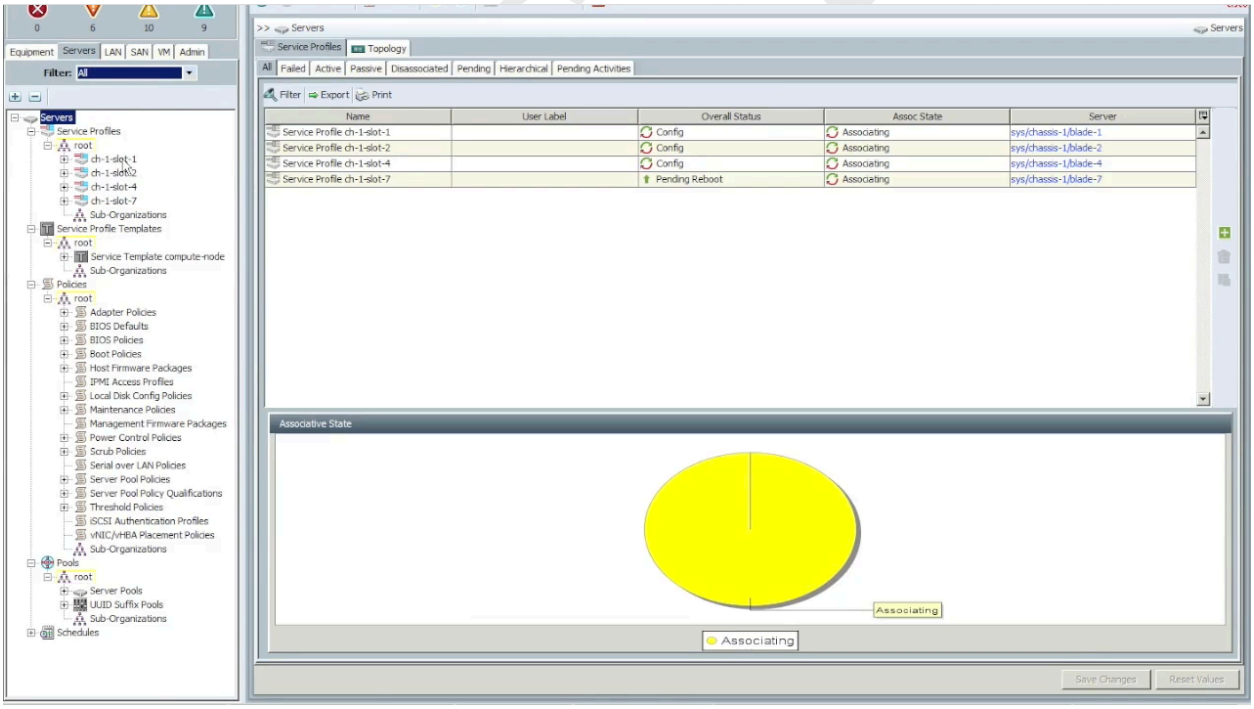
Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

7.3 UCS chassis and server discovery status



7.4 UCS service-profile association status



Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

7.5 Updated local YAML DB

```
root@alpha-build:~/UcsInt-0.2# cat /etc/puppet/manifests/ucs_site_nodes.yaml
  alpha-compute03:
    name: alpha-compute03
    type: compute
    ip: 172.29.75.201
    power_address: dc1-r2-ucsm.ctocllab.cisco.com:org-Production
    power_id: alpha-compute03
    power_admin: admin
    power_password: [redacted]
    power_type: ucs
    mac: 00:25:B5:F2:00:EF
  alpha-compute02:
    name: alpha-compute02
    type: compute
    ip: 172.29.75.200
    power_address: dc1-r2-ucsm.ctocllab.cisco.com:org-Production
    power_id: alpha-compute02
    power_admin: admin
    power_password: [redacted]
    power_type: ucs
    mac: 00:25:B5:F2:00:DF
  alpha-compute01:
    name: alpha-compute01
    type: compute
    ip: 172.29.75.199
    power_address: dc1-r2-ucsm.ctocllab.cisco.com:org-Production
    power_id: alpha-compute01
    power_admin: admin
    power_password: [redacted]
    power_type: ucs
    mac: 00:25:B5:F2:00:FF
```

7.6 Initial cobbler DB & puppet config status

```
root@alpha-build:~/UcsInt-0.2# cobbler system list
   alpha-build
root@alpha-build:~/UcsInt-0.2#
```

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

```
root@alpha-build:~/UcsInt-0.2# /usr/bin/python ./ucs_integration.py --ucsm dc1-r2-ucsm.ctocllab
.cisco.com --ucsm_user admin --ucsm_password [redacted] --debug=True --app-name openstack --liste
ner-only=True
info: Connecting to mysql database: puppet
warning: Dynamic lookup of $puppetversion at /etc/puppet/modules/cobbler/manifests/node.pp:83 i
s deprecated.  Support will be removed in Puppet 2.8.  Use a fully-qualified variable name (e.g
., $classname::variable) or parameterized classes.
info: Caching catalog for alpha-build.ctocllab.cisco.com
info: /Service[mysqld]: Provider upstart does not support features enableable; not managing att
ribute enable
info: Applying configuration version '1373349034'
info: mount[files]: allowing * access
notice: /Stage[main]/Graphite/Package[graphite-web]/ensure: created
notice: /Stage[main]/Collectd/Package[collectd ]/ensure: created
info: /Stage[main]/Collectd/Package[collectd ]: Scheduling refresh of Service[collectd]
notice: /Stage[main]/Graphite/Exec[graphite-syncdb]/returns: Creating tables ...
notice: /Stage[main]/Graphite/Exec[graphite-syncdb]/returns: Installing custom SQL ...
notice: /Stage[main]/Graphite/Exec[graphite-syncdb]/returns: Installing indexes ...
notice: /Stage[main]/Graphite/Exec[graphite-syncdb]/returns: Installed 0 object(s) from 0 fixtu
re(s)
```

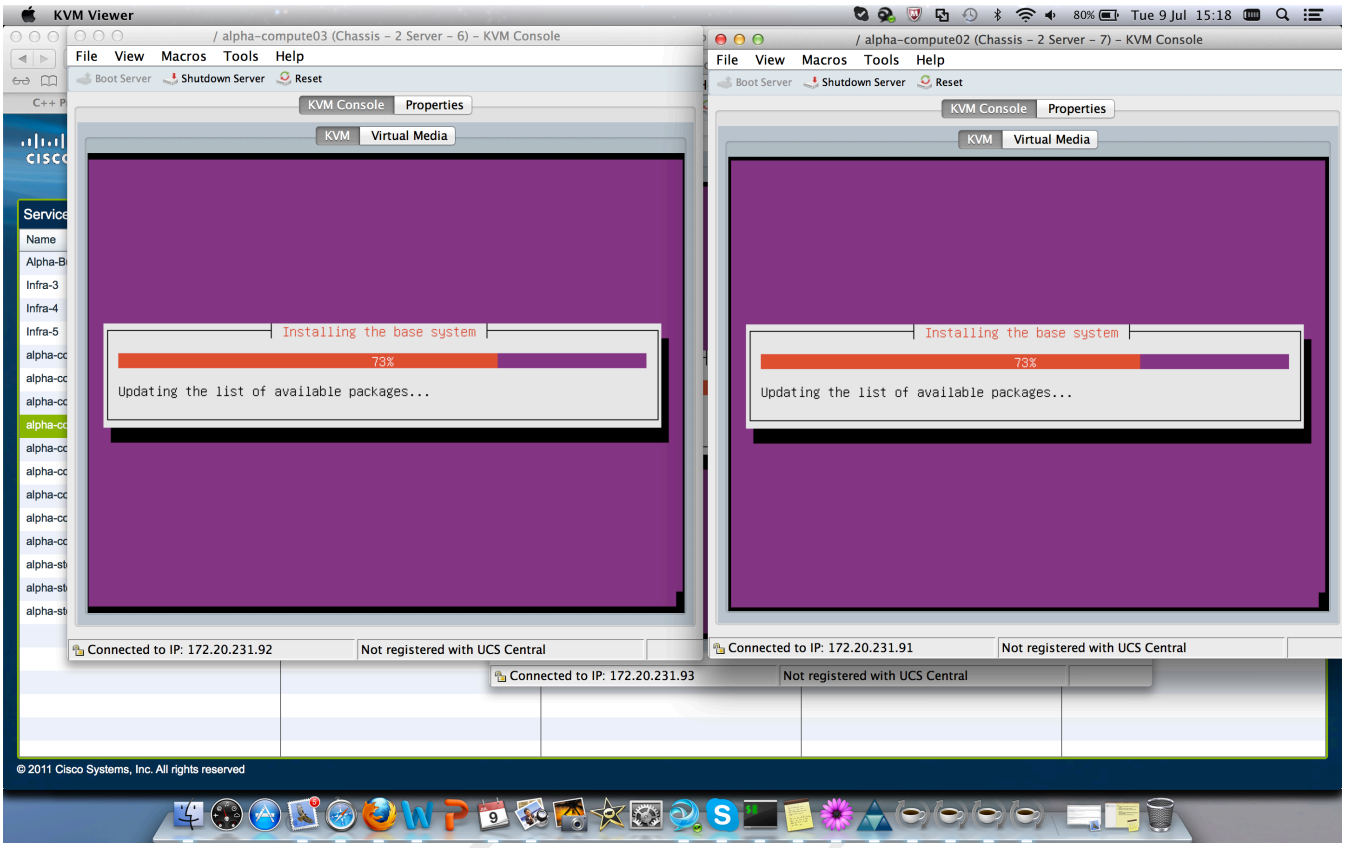
7.7 Updated cobbler DB

```
root@alpha-build:~/UcsInt-0.2# cobbler system list
   alpha-build
   alpha-compute01
   alpha-compute02
   alpha-compute03
   alpha-compute04
   alpha-compute05
   alpha-compute06
   alpha-compute07
   alpha-compute08
   alpha-control01
root@alpha-build:~/UcsInt-0.2#
```

Copyright 2011 Cisco Systems

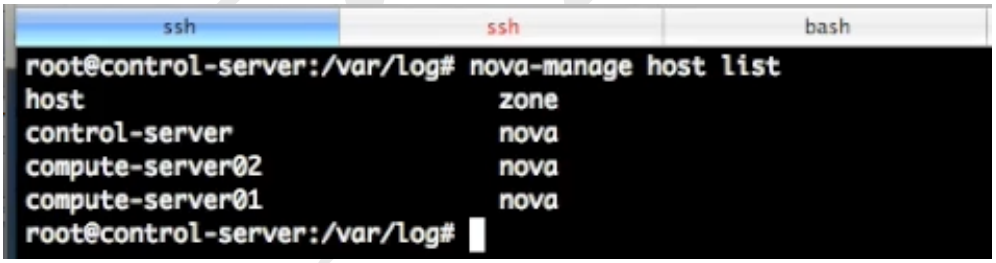
A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

7.8 PXE boot status

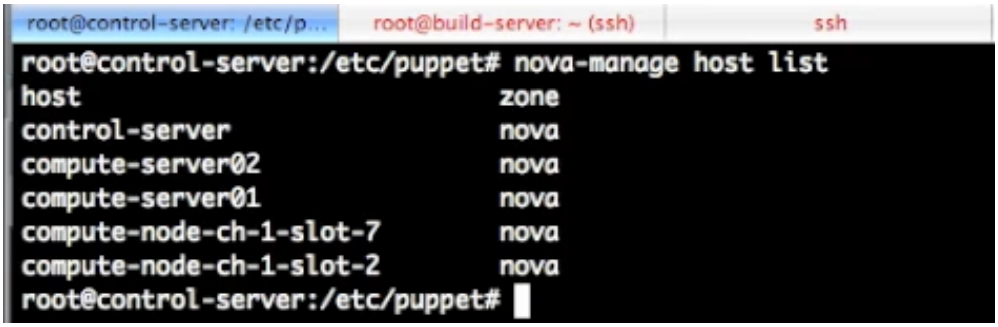


7.9 nova-manage status on control-node

7.9.1 Before executing auto config scripts



7.9.2 After executing auto config script



8 UCS_CONF_TEMPLATE details

8.1 Mac Pool

A MAC pool is a collection of network identities, or MAC addresses, that are unique in their layer 2 environments and are available to be assigned to vNICs on a server. If you use MAC pools in service profiles, you do not have to manually configure the MAC addresses to be used by the server associated with the service profile.

In a system that implements multi-tenancy, you can use the organizational hierarchy to ensure that MAC pools can only be used by specific applications or business services. Cisco UCS Manager uses the name resolution policy to assign MAC addresses from the pool.

You can specify your own MAC addresses or use a group of MAC addresses provided by Cisco.

8.2 Management IpPool

The default management IP pool **ext-mgmt** is a collection of external IP addresses. Cisco UCS Manager reserves each block of IP addresses in the management IP pool for external access that terminates in the CIMC on a server.

You can configure service profiles and service profile templates to use IP addresses from the management IP pool. You cannot configure servers to use the management IP pool.

All IP addresses in the management IP pool must be in the same subnet as the IP address of the fabric-interconnect.

8.3 Service Profiles and Templates

Cisco Unified Computing System service profiles are a powerful tool for streamlining the management of modern data centers. They provide a mechanism

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

for rapidly provisioning servers and their associated network and storage connections with consistency in all details of the environment, and they can be set up in advance of the physical installation of the servers, which is extremely useful in most organizations. They also enable new operational policies for high availability, since a service profile can be applied to a spare server in another rack or blade slot in the event of a server failure, or the profile can be applied to a bare-metal replacement in the failed unit physical slot. However, these basic service profile use cases show only a small fraction of the power and utility of service profiles and primarily provide a transition methodology as users transition from older operating processes to new processes that fully utilize the capabilities of the Cisco Unified Computing System.

The real power of the service profile becomes evident in templates. A service profile template parameterizes the UIDs that differentiate one instance of an otherwise identical server from another. Templates can be categorized into two types: initial and updating.

- **Initial Template:** The initial template is used to create a new server from a service profile with UIDs, but after the server is deployed, there is no linkage between the server and the template, so changes to the template will not propagate to the server, and all changes to items defined by the template must be made individually to each server deployed with the initial template.
- **Updating Template:** An updating template maintains a link between the template and the deployed servers, and changes to the template (most likely to be firmware revisions) cascade to the servers deployed with that template on a schedule determined by the administrator.

Service profiles, templates, and other management data is stored in high-speed persistent storage on the Cisco Unified Computing System fabric interconnects, with mirroring between fault-tolerant pairs of fabric interconnects.

8.4 Server auto configuration policy

Cisco UCS Manager uses this policy to determine how to configure a new server. If you create a server autoconfiguration policy, the following occurs when a new server starts:

- ✓ The qualification in the server autoconfiguration policy is executed against the server.
- ✓ If the server meets the required qualifications, the server is associated with a service profile created from the service profile template configured in the server autoconfiguration policy. The name of that service profile is based on

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

the name given to the server by Cisco UCS Manager.

The service profile is assigned to the organization configured in the server autoconfiguration policy.

8.5 Uplink Ethernet port

Uplink Ethernet ports handle Ethernet traffic between the fabric interconnect and the next layer of the network. All network-bound Ethernet traffic is pinned to one of these ports. By default, Ethernet ports are unconfigured. However, you can configure them to function in the following ways:

- ✓ Uplink
- ✓ FCoE
- ✓ Appliance

You can configure uplink Ethernet ports on either the fixed module or an expansion module.

8.6 Server Pool

A server pool contains a set of servers. These servers typically share the same characteristics. Those characteristics can be their location in the chassis, or an attribute such as server type, amount of memory, local storage, type of CPU, or local drive configuration. You can manually assign a server to a server pool, or use server pool policies and server pool policy qualifications to automate the assignment.

If your system implements multi-tenancy through organizations, you can designate one or more server pools to be used by a specific organization. For example, a pool that includes all servers with two CPUs could be assigned to the Marketing organization, while all servers with 64 GB memory could be assigned to the Finance organization.

A server pool can include servers from any chassis in the system. A given server can belong to multiple server pools.

8.7 Server Pool Policy

This policy is invoked during the server discovery process. It determines what happens if server pool policy qualifications match a server to the target pool specified in the policy. If a server qualifies for more than one pool and those pools have server pool policies, the server is added to all those pools.

BETA

Copyright 2011 Cisco Systems

A printed copy of this document is considered uncontrolled. Refer to the online version for the controlled revision.

8.8 Server pool policy qualifications

This policy qualifies servers based on the inventory of a server conducted during the discovery process. The qualifications are individual rules that you configure in the policy to determine whether a server meets the selection criteria. For example, you can create a rule that specifies the minimum memory capacity for servers in a data center pool.

Qualifications are used in other policies to place servers, not just by the server pool policies. For example, if a server meets the criteria in a qualification policy, it can be added to one or more server pools or have a service profile automatically associated with it.

You can use the server pool policy qualifications to qualify servers according to the following criteria:

- Adapter type
- Chassis location
- Memory type and configuration
- Power group
- CPU cores, type, and configuration
- Storage configuration and capacity
- Server model

8.9 Scrub Policy

This policy determines what happens to local data and to the BIOS settings on a server during the discovery process and when the server is disassociated from a service profile. Depending upon how you configure a scrub policy, the following can occur at those times:

Disk Scrub

One of the following occurs to the data on any local drives on disassociation:

- If enabled, destroys all data on any local drives
- If disabled, preserves all data on any local drives, including local storage configuration

BIOS Settings Scrub

One of the following occurs to the BIOS settings when a service profile containing the scrub policy is disassociated from a server:

BETA

- If enabled, erases all BIOS settings for the server and and resets them to the BIOS defaults for that server type and vendor.
- If disabled, preserves the existing BIOS settings on the server.

8.10 Server port

Server ports handle data traffic between the fabric interconnect and the adapter cards on the servers. You can only configure server ports on the fixed port module. Expansion modules do not include server ports.

9 References

- Cisco UCS Manager
- UCS Manager – configuring service-profiles
- UCS Python SDK (Need Cisco CCO account to download this SDK.)
- OpenStack Folsom

End of Document

BETA