KUBAM: Simplifying Bare Metal Operating System Deployment on Cisco Unified Compute System

February, 2018
Vallard Benincosa, Software Solutions Engineer
Michael Doherty, Techincal Solutions Architect

Executive Overview

In December, 2017 a major travel platform provider had over 500 **Cisco Unified Computing System** (UCS) servers running in production. The engineering team faced the task of redeploying all of these servers from the Microsoft Windows Operating System (OS) to the Red Hat Linux OS. While many OSes today are deployed trivially on virtual machines, installing bare metal OSes can be much more difficult.

This document details how the team used KUBAM, an open source deployment tool developed by Cisco, to accomplish the task in less than 3 weeks. The significance of this undertaking is set forth in the following:

- The team had to use the production network where real production workloads could not be put at risk.
- No PXE/DHCP services could be used as it would introduce security concerns as well as unwanted changes to the network operating model.
- No existing automated provisioning tools were in use
- Much of the time taken in the effort involved getting remote provider support and testing images. Once the solution was in place the effort was only a few days.
- The team now has a tool in which they can redeploy the entire 500+ nodes with a different operating system in less than a day.

KUBAM proved itself to be invaluable, flexible, and meet the needs of the engineering team. The hope is that after reading this document the reader may find KUBAM could be a good fit for their environment as well.

Introduction

We first begin by introducing the components of the UCS deployment system and then outline the requirements. The purpose of this section is to help the reader determine whether this solution could meet their needs at a high level.

UCS

The solution used by the customer is exclusive to UCS. More information on UCS can be found at https://cisco.com/go/ucs.

UCS provides the most complete API for total control over server management. With over 100 plus API touch points, no other server platform comes out of the box with as many control and automation capabilities as UCS. UCS also offers converged networking and network consolidation meaning less dollars are spent on networking equipment and more goes towards buying actual workload power.

With all its capabilities, UCS is usually found in enterprise organizations rather than large scale internet companies. The primary reason for this is that large scale internet providers invest a great deal in infrastructure software developers since this is core to their business. Organizations who chose to invest resources in other software projects can still realize the automation capabilities created by large scale internet companies by using UCS.

Those who decide not to invest in UCS must develop solutions that include the following:
- A firmware update process. This is especially relevant in the advent of platform bugs found in Intel processors announced in late 2017 with Meltdown and Spectre. UCS includes an automated firmware update process that can be done with little to no downtime.
- Consistent BIOS settings. Usually these come from the factory, but there are many quality factors that show up in performance tests that trace their roots back to inconsistent BIOS settings. UCS includes a way to assure that all servers have the same BIOS settings. Changes can be done once and applied everywhere. This is done on a large scale without having to PXE boot to service images.
- Decide on the right network links up front. UCS comes with the ability to create virtual links and save on wiring. The UCS adage is: “Wire once, configure any way”. This gives greater flexibility as servers take on different identities or business requirements change.

It is for many of these reasons that UCS is the number one blade platform world-wide[1] and continues to grow despite the trends of many companies putting more compute resources unto public cloud providers.

UCS has also had recent innovations in its HyperFlex platform which gives customers a converged platform that includes networking, compute, and storage. More information on HyperFlex is available at https://cisco.com/go/hyperflex.

KUBAM

KUBAM was originally built to deploy Kubernetes on UCS Bare Metal (and hence the name) in a consistent and fully automated way. KUBAM is an open source solution from Cisco’s Applied Science Technical Laboratories (CASTL) and is sponsored by DEVNET[2]. It was conceived in 2017 and made available 3 months later in October of the same year for general use. More information and code can be found at https://kubam.io.

The biggest issue KUBAM addresses is getting an OS deployed to a UCS server in a simplified manner. As the KUBAM team investigated OS deployment tools they found numerous solutions that had been available for many years including some recent solutions that all used PXE for OS deployment.

PXE is short for Pre-Execution Environment. It is the method by which a server boots from the network, requests an IP address from a DHCP server and with several other options can be given a disk image with which to install an OS. This practice has been the common way to get an OS deployed on an x86 server for over 20 years and has had very little changes during that time[3].

PXE is complicated in several ways:
- It requires a DHCP server configured with the appropriate options.
- It requires network-helpers or dhcp-relay for complex deployments that exist different layer 3 domains.
- TFTP servers must be deployed to pass out the required boot files.
- MAC addresses must be collected and mapped to the appropriate server and IP addresses.
- The system must not overwrite existing OS deployments and respond back to the TFTP server image to update if the server is already in a deployed state.

---

[1] https://blogs.cisco.com/datacenter/cisco-ucs-is-1-in-x86-blade-servers accessed Feb 12, 2018
[2] https://developer.cisco.com/
[3] https://tools.ietf.org/html/draft-henry-remote-boot-protocol-00 shows the process in place since June 24, 1999. Access Feb 12, 2018

Popular solutions that use PXE to deploy OS images include Cisco UCS Director, Puppet Razor, BMC Software, OpenStack Ironic, Cobbler, and Foreman. The advantage of these solutions is that they can deploy any type of x86 server, since nearly all servers support PXE.

What the KUBAM team also found was that most enterprise organizations in reality do not use PXE. Several informal surveys showed that less than 10% of Cisco UCS customers were actually using PXE to automatically deploy bare metal OSes. Most simply manually mounted remote ISO images[4] with KVM[5] to perform OS installations.

Given this information, the KUBAM team sought to create a tool that was extremely light weight, but could provide a simple and powerful way to deploy bare metal servers without the complexity of PXE. Coupled with this the team sought to make the experience of deploying the OS and Kubernetes as simple and enjoyable as possible using a fully automated approach.

In the case of the customer, they could not use PXE because they were in a production network and the server team did not have ability to change the networking practices in the allotted time. While it was a possible path, doing so would have required architectural changes, subsequent approvals, and would have used up valuable time.

KUBAM met the requirements because it does not use traditional PXE methods. Instead it automates the process of creating images for virtual media, and then uses UCS APIs to apply the media to a service profile. (For more information on what a service profile is see this post.) That means the following traditional requirements could be eliminated:

- No PXE server
- No DHCP server
- No MAC address to IP address mapping
- No network changes
- No DHCP Relay or DHCP Helper or DHCP forwarders required.

Eliminating these complications means bare-metal OS automation is available to more organizations.

KUBAM currently supports deploying VMware ESXi 6.0/6.5, RedHat 7.2/7.3/7.4, and CentOS 7.2/7.3/7.4

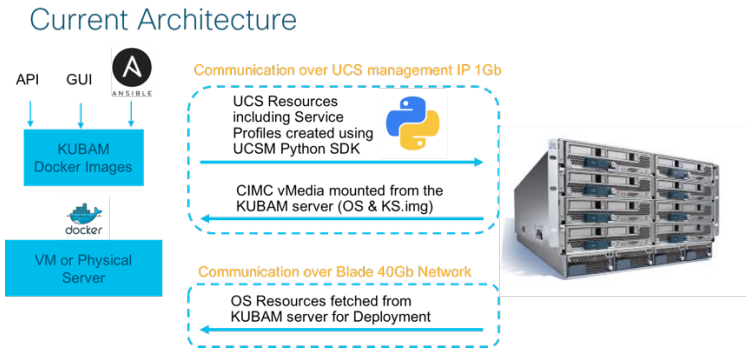
Because KUBAM itself is a light weight set of container images it can be deployed on a physical server or a virtual machine. KUBAM today only works with UCS as the deployment target. To use the full power of KUBAM, Fabric Interconnects are required. However, the way the customer used it, stand-alone UCS C-series servers[6] would have also worked.

High Level Architecture

The requirements for KUBAM include a virtual machine or physical server that can:

1. Reach UCS Manager. This means that the virtual IP address of UCS Manager must be accessible from the KUBAM server.
2. The CIMCs of the servers to be deployed must be able to reach the KUBAM server. This could be done through routed networks or be on the same subnet.
3. The network interface of the provisioned servers must be able to reach the KUBAM server. This is different than the CIMC interface which traverses a different network path.

Meeting these networking requirements is all that is necessary for network setup. Generally, the person accessing the KUBAM GUI should not use a bastion or jump server, but advanced connections are possible[7].



[4] ISO image is an operating system image. See: https://www.lifewire.com/iso-file-2625923 accessed Feb 12, 2018
[5] KVM is remote keyboard, mouse, and video. See: https://en.wikipedia.org/wiki/KVM_switch#Remote_KVM_devices
[6] Stand Alone servers that do not have Fabric Interconnects. See: https://www.cisco.com/c/en/us/products/servers-unified-computing/ucs-c-series-rack-servers/index.html
[7] https://ciscoucs.github.io/kubam/docs/connect

The requirements for the KUBAM server are outlined in the following table.

Table 1. KUBAM Server Requirements

Item	Requirements
KUBAM Server	Virtual Machine or Physical Machine
Memory	8GB
Disk Space	20GB is recommended
Operating System	Linux Operating System (Ubuntu, CoreOS, CentOS, RedHat)
Container Runtime	Tested with Docker but other compatible container engines may work.

With respect to these requirements the customer chose to deploy an Ubuntu VM with the Docker runtime engine. The KUBAM Ubuntu image was given the IP address 172.32.50.147/24 while the target UCS blades in the first batch were deployed with IP addresses in the 10.7.42.0/24 network. Because network routing was setup already, the KUBAM IP address was reachable from the blade CIMCS[8] and the 10.7.42.0/24 network. Due to the simplicity of KUBAM, it can install operating systems over complex routed networks, something that traditional PXE servers take days or weeks to set up and get working due to network technicalities.

KUBAM Server

KUBAM is composed of two microservices that run containers: The API container image kubam/kubam and the web container image kubam/web. To ease deployment both container images are installed using docker compose[9].

Deployment Process

Once the customer had deployed the Docker runtime engine and Docker Compose installing kubam was as simple as typing the following commands:

```
cd ~
curl -O https://raw.githubusercontent.com/CiscoUcs/KUBaM/master/docker-compose.yml
docker-compose up –d
```

The advantages of using a container for deploying the software are:

- Simple updates. When software is to be updated the customer only needs to download a new container image and restart the containers.
- No prerequisites. The container images include libraries for both node, Ansible, Python, React, and other software components. Using containers means no fussing with ensuring compatible libraries.
- Multiple Linux distributions supported. The KUBAM server can run on Ubuntu, RedHat, CentOS, or CoreOS and many others.

There were other caveats that the customer didn't have to work through like SELinux, Proxy's etc. For more detailed instructions for creating the KUBAM server the reader is encouraged to see the KUBAM install documentation.[10]

Configuration Process

KUBAM has several ways in which it can be used. The customer used KUBAM only to create the OS image and customized OS configuration image. This involved them doing the following actions after the container was deployed:

[8] CIMCs are Cisco Integrated Management Controllers: https://www.cisco.com/c/en/us/support/servers-unified-computing/ucs-c-series-integrated-management-controller/tsd-products-support-series-home.html
[9] https://docs.docker.com/compose/
[10] https://ciscoucs.github.io/kubam/docs/kubam-node

1. Copy OS ISO image into ~/kubam directory that was created by the container.
2. Copy Kickstart Template into ~/kubam directory
3. Update ~/kubam/kubam.yaml file with the target IP addresses and server names.

Each of these three details are discussed as follows:

ISO Image

As part of the installation process KUBAM requires the operating system image to be placed in the ~/kubam directory of the KUBAM server. KUBAM uses this ISO image to build a boot ISO image (which is smaller than the normal image) and direct the boot image to automatically install the operating system. If not using KUBAM, the user would need to do this extraction themselves. The nasty details of what is done with the ISO extraction process is documented on several web URLs[11].

Kickstart File

A Kickstart File is a configuration file used by RedHat Linux and its derivatives for automatically installing the Red Hat Operating system.[12] KUBAM actually includes a default kickstart image that can be used for generic installs but most likely the end user has specifications desired and would want to create their own image.

The kickstart file is used to install each server in the target server range. Because of this, the kickstart file required by KUBAM uses a templating language called jinja 2[13]. This allows IP addresses and other attributes to be left as variables inserted by KUBAM in an extensible file. A snippet of a kickstart template is shown below:

```
install
text
network --activate --bootproto=static --ip={{ ip }} --netmask={{ netmask }} --gateway={{ gateway }}
--nameserver={{ nameserver }} --device=enp6s0
network  --hostname={{ name }}
url --url="http://{{ masterIP }}/kubam/redhat7.2/"
```

In the above example the IP address is a variable that is filled in by KUBAM when deploying the image. It is written as {{ ip }} in the jinja2 template. A separate kickstart file embedded into a hard disk image is created for every target server deployed by KUBAM. The other attributes {{ netmask }}, {{ gateway }}, {{ nameserver }}, {{ name }}, and {{ masterIP }} are also filled in.

For a complete example of a kickstart template, see the KUBAM source tree[14]

Creating the kubam.yaml file

The kubam.yaml file specifies how the end user wants to use KUBAM to deploy and configure target servers. KUBAM uses yaml as the configuration file[15]. YAML is a standard format used by many systems like Kubernetes, Docker, etc for specifying configuration parameters. The format of the kubam.yaml file is specified in the documentation[16]. As an example the company only provided a few of the required parameters to make their use case work. This included the following:

```
hosts:
- ip: 10.7.42.71
  name: shlpngfas2252
  os: redhat7.2
  role: generic
...
iso_map:
- file: /kubam/rhel-server-7.2-x86_64-dvd.iso
  os: redhat7.2
kubam_ip: 172.32.50.147
network:
  gateway: 10.7.58.1
  nameserver: customer-name.server
  netmask: 255.255.255.0
  ntpserver: customer-company-ntp.server
public_keys:
```

[11] https://ciscoucs.github.io/os/2017/04/20/centos-redhat-baremetal
[12] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/installation_guide/s1-kickstart2-file
[13] http://jinja.pocoo.org/
[14] https://github.com/CiscoUcs/KUBaM/tree/master/kubam/templates
[15] http://yaml.org/
[16] https://github.com/CiscoUcs/KUBaM/tree/master/kubam

```
- <some public ssh key>
```

The fields specify the hostname, IP address, several networking settings, as well as a mapping from OS to ISO image file. The challenge with this format was that with 500+ servers typing this information manually is error prone and difficult to troubleshoot using a GUI. To fill out the entire kubam.yaml file a python script was created given the csv file[17] that was provided by the customer.

The company provided a CSV file in the format:

nodename01.domain.com,10.7.0.1
nodename02.domain.com,10.7.0.2
nodename03.domain.com,10.7.0.3
…

The script that was created read the file in, stripped the domain name, and converted to yaml. The script used looked similar to the below

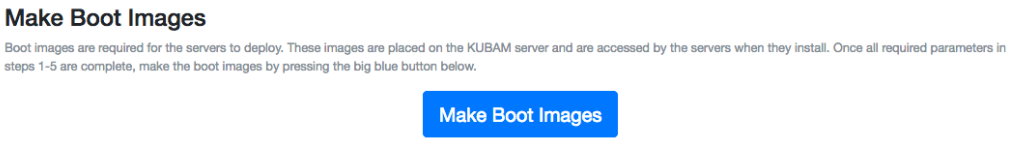
```
#!/usr/bin/env python2
import re

with open("./allPods.csv") as f:
    for i in f:
        host, ip = i.split(',')
        host = re.sub('.domain.com', '', host)
        print "- ip: %s" % ip.strip()
        print "  name: %s" % host.strip()
        print "  os: redhat7.2"
        print "  role: generic"
```

Adding this to the kubam.yaml file allowed KUBAM to create all the individual kickstart images so that each node would boot with the appropriate image.

The one issue encountered in this process was that KUBAM was not designed to configure different routers for distinct server groups. In the current efforts of KUBAM this is being adjusted to account for server pools of different name servers and routers into “server groups”.

Once the 3 files were in place and correct, the customer was able to use the KUBAM web interface to press the button “Make boot Images” and all images were ready for OS deployment.



Alternatively, the kubam server could have been called using the API to create target server images as shown below:

```
curl –X POST http://<kubamIP>>/api/v1/isos/boot
```

Other KUBAM capabilities

The customer engineers chose to only use a subset of what is possible with KUBAM and had no current need of the other functions. Other ways that could have been used are:

1. Embed KUBAM in Ansible playbook
2. Use KUBAM API directly to configure installation process
3. Use KUBAM Web GUI for total deployment including service profiles

In each of these processes the installation process with kickstart images specified above could be used.

UCS Configuration

The customer had previously created the UCS Service Profiles and didn’t require automation to update those. The requirement for

[17] https://en.wikipedia.org/wiki/Comma-separated_values

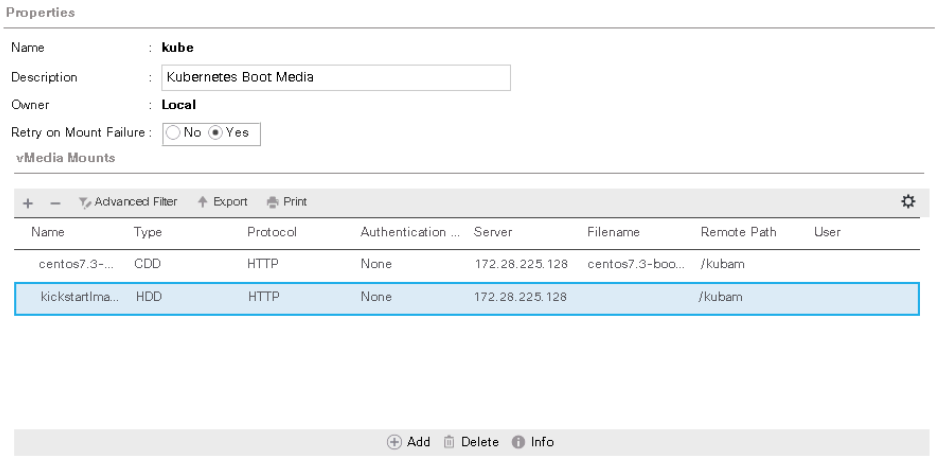
KUBAM is that the name of the service profile match the name of the OS image that the customer put in the kubam.yaml file.

The last step for automation was to ensure that the KUBAM images were present in a vMedia policy.

vMedia Policy

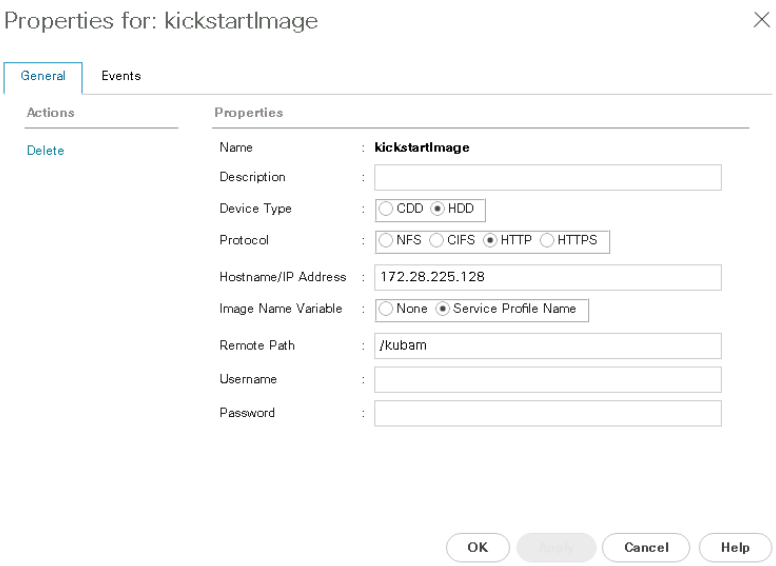
One of the attributes of server that was introduced to UCS was the idea of creating a policy that tells the server what remote media it should mount. On other platforms the user must do this by clicking each individual server. With UCS this can be done once in a service profile template and all child templates inherit this policy. The customer created a policy that was added to a service profile that included the mounting of two remote images.

The final policy was similar to the diagram below



Here there are two images the server will automatically mount when the physical server is associated with the service profile. The first image is the ISO image that KUBAM creates from the RedHat/CentOS media. Every server mounts this same image.

The second image mounted is unique for each server as it is based off the service profile name.



The above example shows that each server using this policy will mount an image from the web server http://172.28.225.128/kubam/<service-profile-name>. This is a hard drive image that was created by KUBAM. It is a 1MB image that contains one file: The ks.cfg file that was built from the template. The First CDD image contains an isolinux file that is configured to point to the KUBAM image for setup instructions. For more information on this process see the blog article[18].

Once the vMedia policy is in place a boot policy[19] was created in UCS that specified for the machine to boot off the virtual media if no

[18] https://ciscoucs.github.io/os/2017/04/20/centos-redhat-baremetal
[19] https://www.cisco.com/c/en/us/td/docs/unified_computing/ucs/sw/gui/config/guide/2-

hard drive was installed. The disks were purged as part of dissociating the previous service profile by creating a scrub policy[20].

Troubleshooting

Amazingly, some things don't work exactly as planned and there were several places where debugging KUBAM and related UCS policies were necessary. Many of those steps are documented currently in a KUBAM tutorial.[21] We encourage readers of this document or those wishing to perform similar steps as outlined to visit this site for additional help. It should also be noted that the KUBAM user interface comes with a button for feedback or support on the bottom right. Because it is one-way communication those who would like support should include their contact details when posting feedback.

Conclusion

This document has outlined the steps the customer took to redeploy several pods of UCS servers. While the process wasn't entirely painless it simplified exponentially what could have been a very arduous task.

KUBAM delivers a lot of value in little time. The engineers weren't required to change any network configurations nor did they have to deploy each server individually but could provision very quickly. KUBAM developers were concerned with 100+ servers hitting the web server and not being able to deliver but were surprised at the resiliency and scalability of what was able to be delivered. KUBAM also has a very low barrier to entry. Simply installing docker containers made the installation simple and painless. Finally, KUBAM fits into existing processes and didn't demand that any current DevOps practices be changed or modified. If the customer was using another automation tool, KUBAM could have been placed as a component in the process without much fuss.

It is the hope of the KUBAM developers that those who read this document will consider using KUBAM as part of the deployment process for UCS. KUBAM is under constant development and lessons learned from this installation are already being developed into the next iteration of the project.

The original purpose of KUBAM to install Kubernetes on UCS Bare Metal as simple a possible is still very much the goal of the project. The KUBAM team would also encourage those wishing to dip their toes into container orchestration to use KUBAM for Bare Metal Kubernetes deployments.

0/b_UCSM_GUI_Configuration_Guide_2_0/b_UCSM_GUI_Configuration_Guide_2_0_chapter_011101.html#concept_2313A9F59CD94237819D14A3CA608DFD

[20] https://www.cisco.com/c/en/us/td/docs/unified_computing/ucs/sw/gui/config/guide/2-0/b_UCSM_GUI_Configuration_Guide_2_0/b_UCSM_GUI_Configuration_Guide_2_0_chapter_011100.html#d129007e4747a1635

[21] https://github.com/vallard/KUBAM-Tutorial/blob/master/docs/trouble.md