Deploying workloads with Juju and MAAS in Ubuntu 14.04 LTS

A Dell Technical White Paper

Kent Baxley

Canonical Field Engineer

Jose De la Rosa

Dell Software Engineer

Mark Wenning

Canonical Field Engineer

May 2014

Contents

1. Introduction

1.1. Purpose of this document

This whitepaper will show you how to instantly deploy, integrate and scale software services on top of Dell PowerEdge servers. These software services are provided by Ubuntu Juju charms, which are deployed to hardware via Ubuntu MAAS (Metal-as-a-Service). We will explain in detail what Juju and MAAS can do together.

The intended audience of this whitepaper is IT system administrators (and enthusiasts with access to a few servers) who want to quickly deploy Linux-based applications on Dell PowerEdge servers and automate the configuration steps required for each application.

1.2. Assumptions & Disclaimers

It is assumed that the reader is familiar with Ubuntu Server and with the Linux operating system in general. You don't have to be an expert, but past experience will make it a lot easier following this document. We assume that you are familiar with applications and services such as Wordpress, MySQL and OpenStack.

It is assumed that the reader is familiar with the concepts of service orchestration and the dynamic allocation of resources to implement on-demand services. You don't have to have expertise in cloud services to follow this whitepaper, but familiarity with such concepts will be very helpful.

2. MAAS

2.1. What is MAAS

Metal-As-A-Service is hardware provisioning software from Canonical intended to quickly commission and deploy physical servers to run a wide array of software services or workloads via Juju charms (more on Juju in Chapter 3).

Servers can be dynamically associated or connected together to scale up services, and can also be disconnected to scale down as demand requires it. MAAS treats physical servers as compute commodities that can be quickly manipulated to meet customer demand, similar to how a cloud environment creates and removes virtual resources to adjust to computing demands.

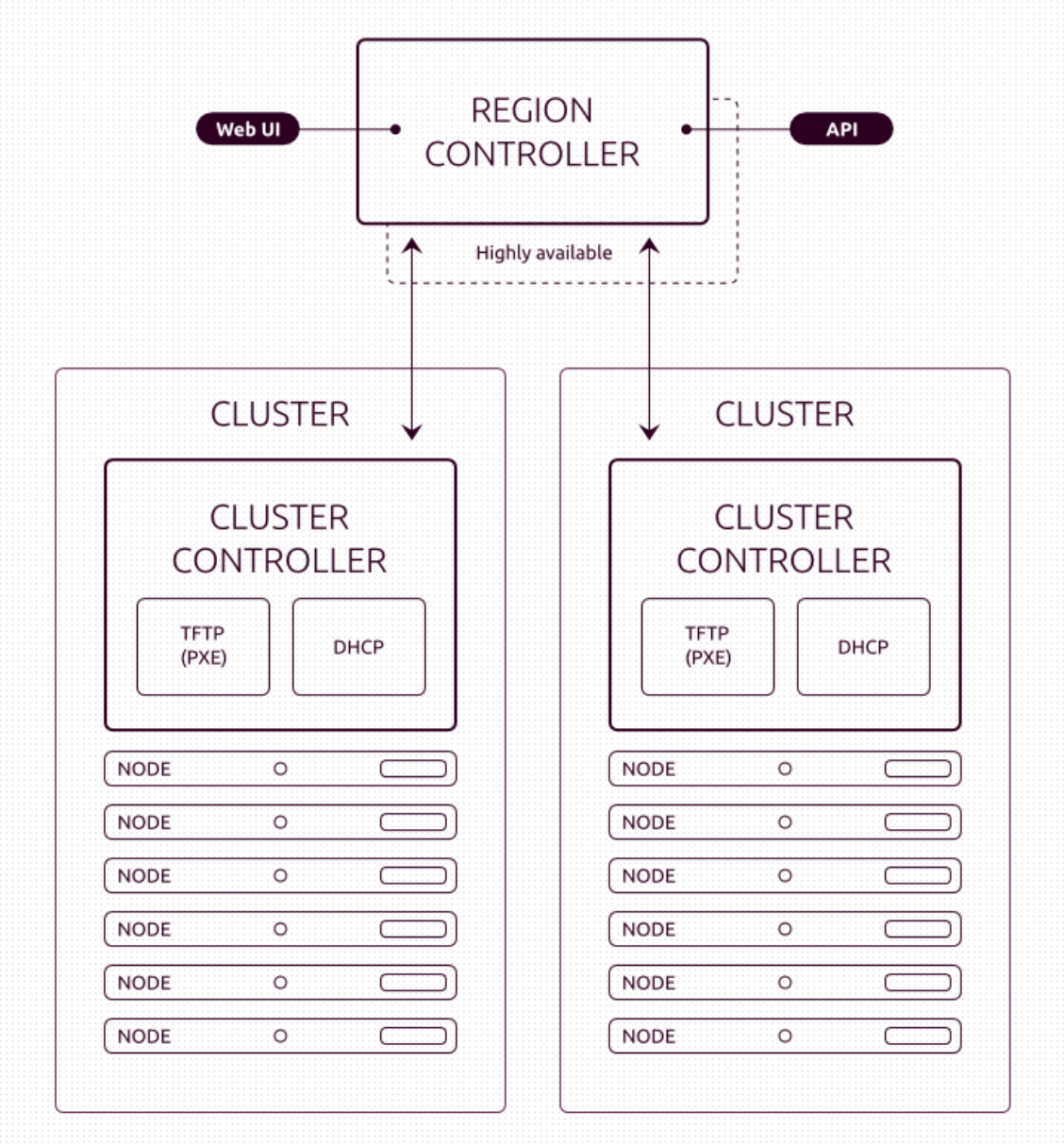

Figure 1 – MAAS deployment with a Regional Controller and two Cluster Controllers.

A MAAS deployment consists of one Region Controller, one or more Cluster Controllers, and as many physical servers as you will require to deploy your applications. Figure 1 (borrowed from http://maas.ubuntu.com) depicts a MAAS deployment with a Region controller, two Cluster controllers and 6 servers (nodes) per cluster. **The terms 'servers' and 'nodes' are used interchangeably throughout this whitepaper.**

A Region controller consists of a web user interface, an API, the metadata server for cloud-init and an optional DNS server.

A Cluster controller is in charge of provisioning and consists of a TFTP server and an optional DHCP server. It is also responsible for powering servers on and off.

Note that both the DNS and DHCP services are optional because you have the option of using your own DNS and DHCP services already running in your environment.

2.2. Hardware Requirements

2.2.1. MAAS server

For simplicity purposes, we will do a MAAS deployment with the Region controller and one Cluster controller (one subnet) on the same machine. Table 1 lists the minimum requirements for the MAAS server.

Component	Minimum
CPU	Dual-core or better
Memory	4 GB
Disk	Single disk or RAID 1 boot device of 150 GB
Network	2 x 1GB NICs, external internet access for downloading media
Operating System	Ubuntu Server 14.04 LTS

Table 1 – Minimum hardware requirements for MAAS server

Canonical does not mention that the MAAS server has to run on bare metal. Even though we see no reason why you can't install and run it in a VM, we installed it on a physical machine.

2.2.2. Nodes

a. **In a lab setting used for testing purposes, any server with a minimum dual-core CPU, 4 GB of memory, 150 GB disk, 1 GB NIC will do.** The quantity and specification of PowerEdge servers that you will need will depend on the type of applications that you intend to use and the loads you expect.

b. **All servers must be equipped with out-of-band management such as an iDRAC or a BMC controller**, as they will be powered on and off via IPMI commands, so be sure to turn on IPMI over LAN. If using an iDRAC, virtual media must be set to auto-attached, otherwise the OS provisioning process will fail.

c. **All servers must be set to boot from the network.** Since servers are provisioned via PXE, they must always boot from the network first. Also, because in this example we only use one Cluster controller, all servers must be in the same subnet.

2.3. Installation

There are two ways to install MAAS, either during OS installation or after the OS has been installed. We recommend following the instructions described here as there are a couple of extra steps we had to take to get a working environment.

a. We installed MAAS after installing the OS by following the instructions at http://maas.ubuntu.com/docs/install.html. Since we are installing the region and cluster controller in the same server, we install the 'maas' package, which will install both the 'maas-region-controller' and the 'maas-cluster-controller' packages.

   $ sudo apt-get install maas

   The installation can take 5-15 minutes, depending on your network connection. According to Canonical's documentation, expect about 200 MB of packages to be installed.

   As of this writing, the latest MAAS version in the repository is 1.5+bzr2269-0ubuntu0.1.

b. MAAS requires DNS and DHCP services, which MAAS can provide. However, you can skip this step if you decide to use your existing network's DNS and DHCP services.

   $ sudo apt-get install maas-dns maas-dhcp

2.4. Post-Install Tasks

a. Create a super user login id. It's ok to accept the default name of 'root'.

   $ sudo maas-region-admin createsuperuser

b. Login to the web UI by going to http://<your-maas-server-name>/MAAS. You will see a login screen as shown in Figure 2. Use the credentials created in the previous step to login.

c. If you use a proxy server for remote access, you will need to specify it. Click on Settings (the gear icon in the top-right corner) and enter your proxy server in the "Network Configuration" section.

d. Go to the home page. As you can see from Figure 3, there are two pending activities: 1) import boot images for our nodes and 2) specify whether we need Third-party drivers for booting nodes.

   Click on the "Initiated by hand" link in the yellow box, which will take you to the Clusters tab, click on "Import boot images" towards the bottom of the page. The import process will start in the background and will take about 30-90 minutes, depending on the speed of your network. You will continue to see the yellow block until the import process has finished.

   We do not require Third-Party drivers on PowerEdge servers. Go to the home page and click on the "settings" link in the blue box. This will take you back to the Settings page, uncheck the Third Party Drivers option and click Save.

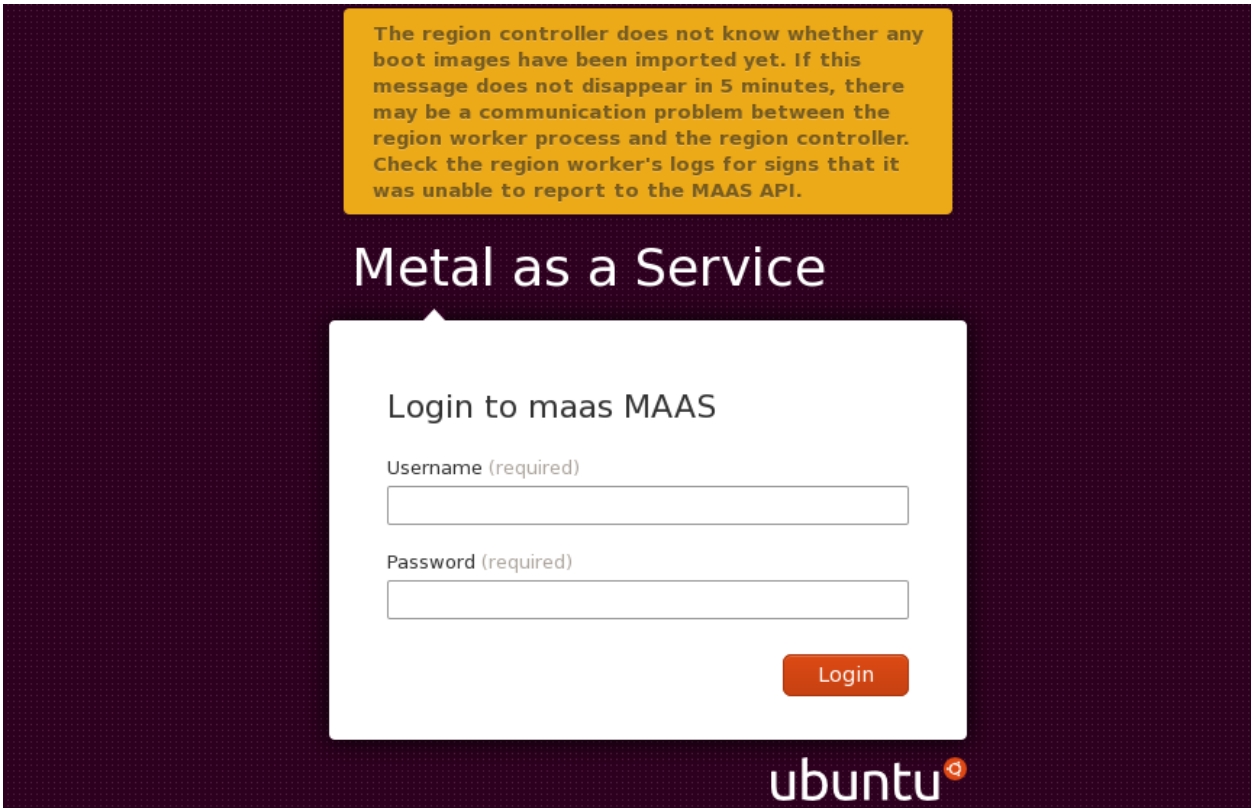

Figure 2 – MAAS web UI login screen

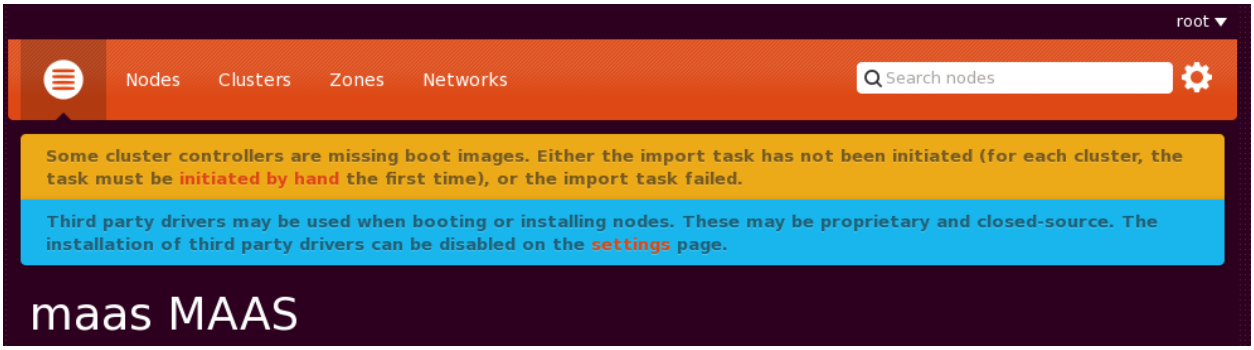

Figure 3 – Import boot images and disable Third-Party drivers

e. Add a public SSH key that will be used to access the nodes. In the top-right corner, click on "root" and select "Preferences" from the drop-down menu. Click on "Add SSH Key" and paste your public SSH key in the field.

f. Select the default distribution used for commissioning and deployment. In the MAAS web home page, click on the gear icon in the top right corner to go the Settings page, scroll down to the Commissioning and Ubuntu section and select a release. We highly recommend you use the default distribution already selected.

g. We did not add SSL support for our web UI, but you can easily follow the steps in http://maas.ubuntu.com/docs/configure.html#ssl-support if you'd prefer to have SSL support.

2.5. Configuring DNS and DHCP services

2.5.1. Using your existing DNS and DHCP services

If you choose to use your existing DNS and DHCP services, make sure you follow the guidelines for how to configure your existing DHCP server:

http://maas.ubuntu.com/docs/configure.html#manual-dhcp

In our lab, we used our existing DHCP server and all we had to do was change the IP address for the 'next-server' parameter in our DHCP services configuration file to that of our MAAS server (where TFTP is running). Be sure to restart the DHCP service for the change to take effect.

2.5.2. Using MAAS to provide DNS and DHCP services

If instead you want to use MAAS to control DNS and DHCP, you can use the MAAS cli (see Appendix A) to configure these services by following the instructions listed here:

http://maas.ubuntu.com/docs/maascli.html#cli-dhcp

Alternatively, you can use the web UI to configure DNS and DHCP. In the MAAS web home page, click on the Clusters tab and then click on the "Cluster master" link, as indicated in Figure 4.

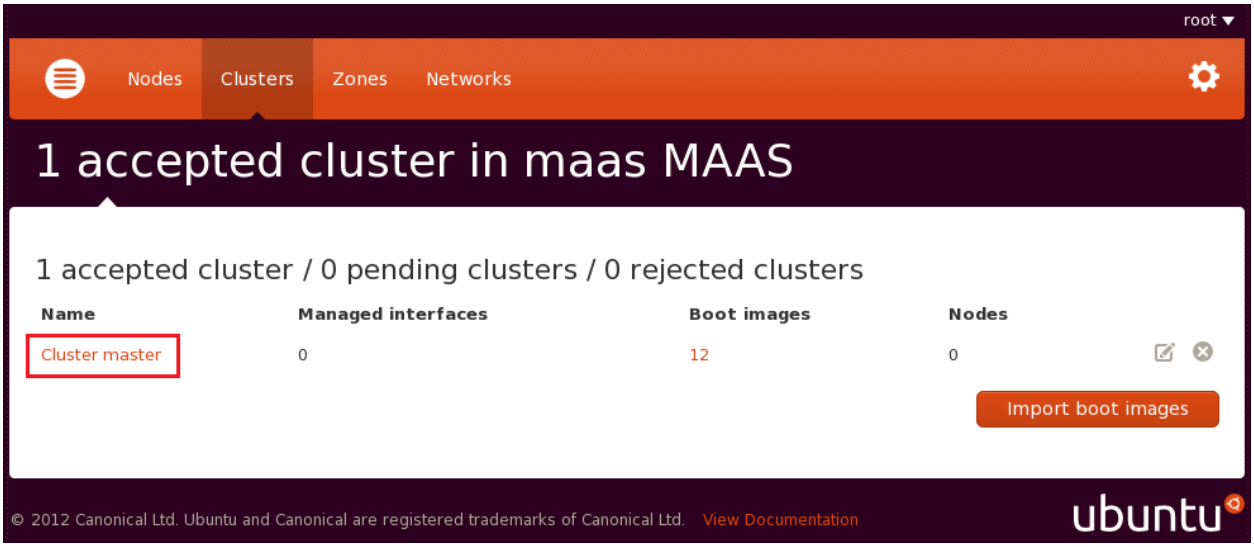

Figure 4 – Cluster page

Next, in the Interfaces section of the Edit Cluster controller page, click the edit icon for the interface you wish to use. You may have to add an interface if the one you wish to use isn't visible, as shown in Figure 5.

After you click on the Interface edit icon, in the 'Management' drop-down menu, select whether you want the interface to manage DHCP alone or both DNS and DHCP and fill out all the required fields. For the easiest experience, the preferred option is to let MAAS manage both DNS and DHCP. When done, click "Save Interface".

If you need to make additional, fine-grained changes to your maas-dns setup, you can manually edit the files in /etc/maas/templates/dns/. Make sure to restart the bind9 service after any changes are made.

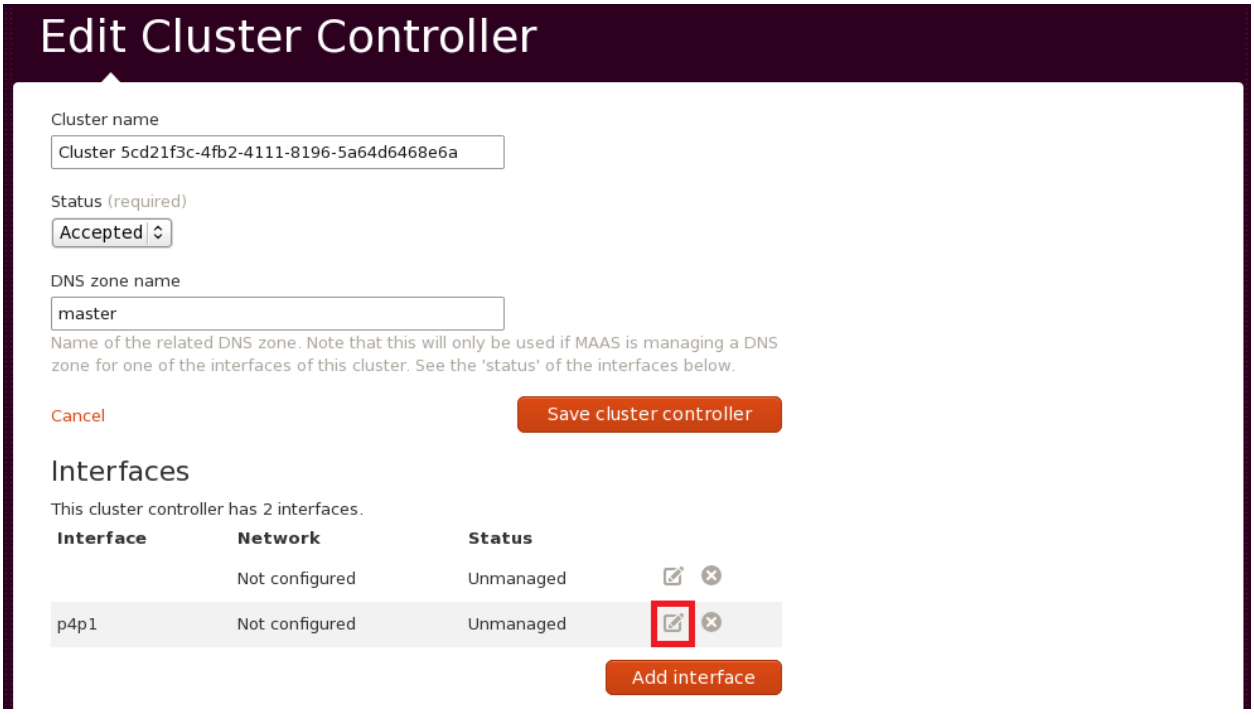

Figure 5 – Edit Cluster Controller

2.6. Provisioning

Once we've met hardware requirements, completed post-install activities and have MAAS properly configured, we are ready to start provisioning servers. This process can be broken down into 3 stages: Enlistment, Commissioning and Deployment.

2.6.1. Enlistment

The enlistment process starts when a server is manually powered on (by pressing the power button or through the iDRAC) and registers itself with MAAS. When the server boots, it obtains an IP address and then PXE boots from the Cluster Controller. The server then loads an ephemeral image that runs and performs an initial discovery process via a preseed fed to cloud-init. The discovery process obtains basic information such as network interfaces, MAC addresses and the machine's architecture.

Once this information is gathered, a request to register the machine is made to the MAAS Region Controller. When done, the server will be listed in MAAS with a 'Declared' state.

MAAS creates a 'maas' login ID and password in the iDRAC / BMC to power a server on or off. To verify, click on a server's FQDN (as shown in Figure 6), then click on "Edit node" and verify the power type and parameters, which should include an IP address, login and password. The power type should already be set to 'IPMI' and the power parameter driver should be set to 'auto-detect'. If you have any issues powering servers, try 'IPMI 1.5' for PowerEdge blades or 'IPMI 2.0' for PowerEdge rack and tower servers.

2.6.2. Accept and Commission

Once a server is in the 'Declared' state, it must be accepted into MAAS in order for the commissioning processes to begin and for it to be ready for deployment. The commissioning process is where MAAS collects hardware information such as the number of CPU cores, memory, disk size, etc. which can be later used as constraints.

To accept a node, click on a server and then click on "Commission Node". Once you commission a node, it will PXE boot from the MAAS Cluster Controller and will be instructed to run the same ephemeral image. This time, however, the commissioning process will be instructed to gather more information about the server, which will be sent back to the MAAS region controller via cloud-init.

Once this process has finished, the server information will be updated and its state will change to 'Ready', which means it's ready for deployment. Figure 6 shows 4 servers that are ready for use.

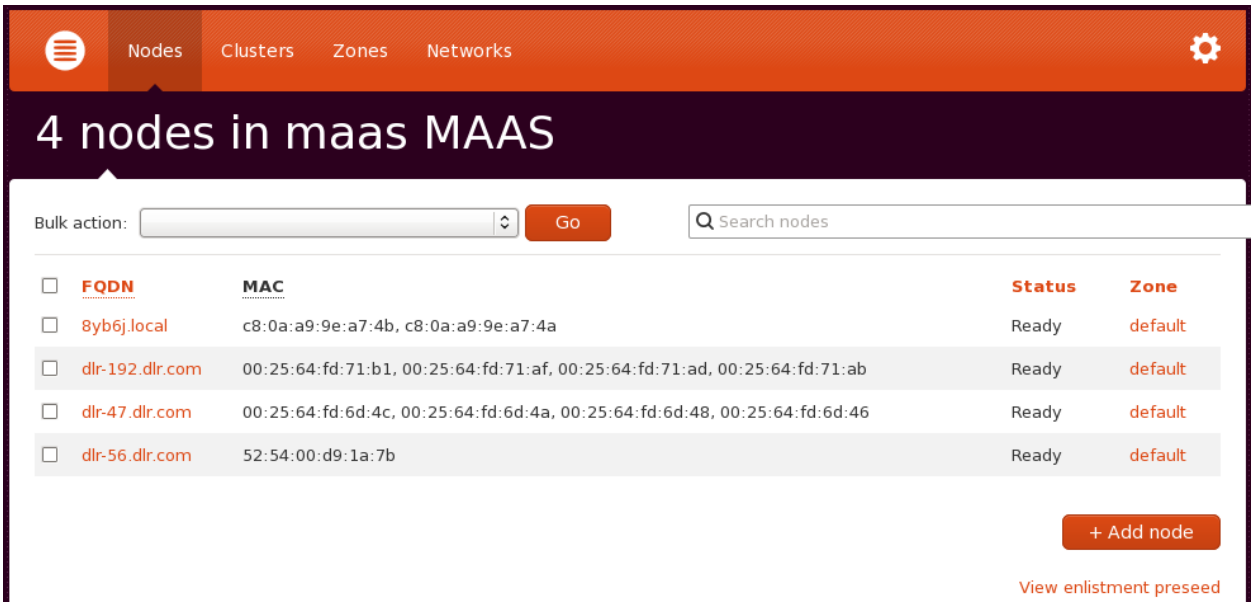

Figure 6 – Four servers (nodes) in the 'Ready' state

2.6.3. Deploy

Once a server is in the 'Ready' state, it can be used for deployment, which we will cover in the next chapter. When a server has been deployed, its state will change to 'Allocated to <user>'. This state means that the server is in use by the user who requested its deployment.

3. Juju

3.1. What is Juju

Juju is software that provides service orchestration and is used to easily and quickly deploy and manage services using components called 'charms'.

For example, say you want to deploy a blog web site using the open source Wordpress application. This can be done in 3 easy steps with Juju:

1. Deploy the Wordpress Juju charm
2. This charm requires a database backend, deploy the MySQL Juju charm.
3. Associate both charms together.

This process illustrates the simplicity involved in deploying a new application in a matter of minutes. In addition, you can easily scale up or down in one simple step by adding or removing charm units.

Juju charms can be deployed on cloud services such as Amazon Web Services (AWS), Microsoft Azure and OpenStack. It can also be used on bare metal using MAAS, which is what we cover in this whitepaper.

3.2. Installation and configuration

3.2.1. Package installation

For simplicity purposes, we installed the juju packages in the same server where we installed MAAS. Installation and configuration is simple and fast; you can follow the instructions at https://juju.ubuntu.com/docs/getting-started.html. Install package:

$ sudo apt-get update && sudo apt-get install juju

3.2.2. The environments.yaml file

We now need to configure Juju to be used with MAAS, which is done by generating and editing the file ~/.juju/environments.yaml. A template of this file is generated with

$ juju generate-config

You will have to comment out a lot of extra lines that are generated for other providers such as Amazon's AWS. To use with MAAS, this is what our environments.yaml file looks like:

```
default: maas
environments:
  maas:
    type: maas
    maas-server: 'http://<MAAS-server-IP-address>/MAAS/'
    maas-oauth: ' <MAAS-API-Key>'
```

Table 2 lists the parameters that will be different for your deployment.

Parameter	Value
maas-server	The URL of your MAAS server. Use an IP address, and be sure to add port 80.
maas-oauth	The API key. Refer to Appendix A for instructions on how to extract it.

Table 2 – Parameters to set in environments.yaml

Table 3 lists useful (but optional) parameters in environments.yaml:

Parameter	Value
http-proxy, https-proxy, ftp-proxy	Specifies http, https and ftp proxy server. See https://juju.ubuntu.com/docs/howto-proxies.html
apt-http-proxy, apt-https-proxy, apt-ftp-proxy	Specifies http, https and ftp proxy server specific to apt. See https://juju.ubuntu.com/docs/howto-proxies.html
no-proxy	Specifies a list of host names and addresses to bypass proxy server
bootstrap-timeout	Time allowed (in seconds) for juju to bootstrap a node. Default is 10 minutes, which should be increased for slow network connections.
bootstrap-retry-delay	How often juju polls a started node to see if it's ready. Default is 5 seconds.
default-series	Force charms to be from this series (i.e. "precise", "trusty"). Can be overridden on the command line with option "—series".

Table 3 – Optional parameters in environments.yaml

3.3. Bootstrap node

Before you can deploy any Juju charms, you need to deploy a Juju bootstrap node. This node is used by Juju to deploy and control other services. Table 4 lists the recommended hardware requirements for the bootstrap node.

Component	Recommended
CPU	Dual-core or better
Memory	4 GB
Disk	Single disk or RAID 1 boot device of 150 GB
Network	2 x 1GB NICs, external internet access for downloading media

Table 4 – Recommended hardware requirements for Juju bootstrap node

To launch the bootstrap node, run:

```
$ juju sync-tools
$ juju bootstrap
```

You can view a more verbose output by passing the '-v' or '--debug' parameter in the above commands.

3.3.1 Proxy Server

You can set environment variables specifying proxy settings in your network prior to syncing tools or bootstrapping. For example, you can add the following lines to your */etc/environment* file or *.bashrc*:

export http_proxy=http://<your-proxy>:<port>/
export https_proxy=http://<your-proxy>:<port>/
export no_proxy="locahost,192.168.0.2,127.0.0.1"

If your MAAS server is on a separate, private network that doesn't require a proxy, you may want to include a "no_proxy" variable like above. In this example, the MAAS server is at 192.168.0.2.

3.3.2 Restricted or No Internet Access

If you are in a situation where external internet access is not possible, you can still bootstrap with a local copy of the tools. In this example, we share the tools from a webserver:

a. Prior to bootstrapping, you will need to download a copy of the tools from a computer that has internet access, for example:

   $ wget -c https://streams.canonical.com/juju/tools/releases/juju-1.18.1-trusty-amd64.tgz

   Make sure the release version matches the version of juju you are running. Here we are running the version of juju that ships with Ubuntu 14.04 LTS at the time of this writing.

b. On the node you intend to host the tools from, set up the directory structure for the tools (this assumes you already have an apache webserver up and running). The "tools/releases" portion of the directory structure is mandatory:

   $ sudo mkdir -p /var/www/html/juju-metadata/tools/releases

c. Copy the tar file you downloaded in step 'a' to the 'releases' directory created in step b.

d. Generate the simplestreams tools metadata by running the following:

   $ sudo juju --debug metadata generate-tools -d /var/www/html/juju-metadata

   This will generate a "streams/v1" directory under tools. Make the files accessible:

   $ sudo chmod 644 /var/www/html/juju-metadata/tools/streams/v1/*

e. In your ~/.juju/environments.yaml file you will need to specify the URL of your newly shared tools location under the "maas:" stanza. This will override juju from trying to get out to https://streams.canonical.com/ to get the tools. For example:

```
default: maas
environments:
  maas:
    type: maas
    maas-server: 'http://<MAAS-server-IP-address>/MAAS/'
    maas-oauth: ' <MAAS-API-Key>'
```

tools-metadata-url: http://192.168.0.2/juju-metadata/tools

6) Once this is all done, you can run "juju bootstrap" to get your environment up and running.

4. Juju charms

The magic behind Juju is a collection of software components called charms. These are the encapsulated knowledge of how to properly deploy and configure the services you will want to deploy on your hardware or on your cloud. Charms make it quick and easy to reliably and repeatedly deploy services.

4.1. Downloading charms

When deploying charms you can either 1) download them from external repositories as you deploy, or 2) download them first to a local repository and then point to the repository as you deploy.

If you are in a network environment where there may be some restrictions on internet access, it is recommended that you download the charms to a local repository to prevent any network connectivity issues during deployment. In our case, we downloaded the charms to the same server where we installed juju (which if you remember from section 3.2.2, is the same server where we installed MAAS). Charms can be downloaded using the Bazaar version control tool:

Install Bazaar if needed:

$ sudo apt-get install bzr

To see a full list of available charms, go to https://code.launchpad.net/~charmers. To download a specific charm, run:

$ mkdir –p /opt/charms/trusty; cd /opt/charms/trusty
$ bzr branch lp:charms/<charm-name>

For example, to download the 'mediawiki' charm, run:

$ bzr branch lp:charms/mediawiki

Alternatively, if you have a direct connection to the internet (you're not using a proxy), you can use:

$ sudo apt-get install charm-tools
$ charm get <charm>

To deploy a downloaded charm from your local repo:

$ juju deploy –repository=/opt/charms local:trusty/<charm-name>

Juju will deploy the charm from the local repository when directed to. If the local repository is not found, Juju will attempt to deploy the charm from the online charm store.

4.2. Constraints

Constraints allow you to have some degree of control over which hardware you wish to deploy services to. Constraints can be set for environments as well as services.

A simple example of using constraints can be something like this:

$ juju deploy mysql --constraints mem=10G

This tells Juju to deploy the MySQL charm on a system with at least 10 GB of memory. Juju will attempt to use the machine with the minimum specified amount of RAM and smallest number of CPUs available to deploy the charm on.

More details and examples of how to utilize constraints can be found on the Juju documentation site:

https://juju.ubuntu.com/docs/charms-constraints.html

4.3. Deploying Wordpress

The following is an example of how to deploy Wordpress, an open source web-based blogging application. It is assumed you have bootstrapped your environment (section 3.3) and you have at least 2 servers in the 'Ready' state in MAAS to deploy to.

1. Download charms if you haven't already. Since the Wordpress application requires a database backend, you will need to download both the wordpress and the MySQL charms.

   $ mkdir –p /opt/charms/trusty; cd /opt/charms/trusty
   $ bzr branch lp:charms/wordpress
   $ bzr branch lp:charms/mysql

2. Deploy charms. Assuming we downloaded the charms to local directory /opt/charms/trusty, we specify the location of the repository in our deploy command:

   $ juju deploy –-repository=/opt/charms local:trusty/wordpress
   $ juju deploy –-repository=/opt/charms local:trusty/mysql

   The 'local' option tells Juju to look for the charm in a local repository.

   If you get an error about the system not knowing what the default series is or if it can't find charm in /opt/charms/trusty, try this instead:

   $ juju set-env default-series=trusty
   $ juju deploy –-repository=/opt/charms local:wordpress
   $ juju deploy –-repository=/opt/charms local:mysql

   As mentioned in section 4.2, you can use constraints to have more control over what servers you deploy charms to. For more details, see https://juju.ubuntu.com/docs/charms-constraints.html

If you go to the MAAS home page, you should see that two servers have changed status from 'Ready' to 'Allocated to root'. The servers will go through a bootstrapping process where they will boot via PXE, the OS will be installed and then the corresponding application packages will be installed and configured.

3. Get status:

   $ juju status wordpress
   $ juju status mysql

   For ease of readability, we omit the output here, but the critical parameter to look for is "agent-state". While the bootstrapping process is ongoing, agent-state will be "pending". Check again after 5-15 minutes (depending on your network connection); when done, agent-state should be 'started'.

4. Add charm relation. Because wordpress requires a database backend, we need to add a database relation to it. A charm relation is how juju charms are tied together. Juju takes care of it for us with one simple command:

   $ juju add-relation wordpress mysql

5. Get status:

   $ juju status wordpress

```
environment: maas
machines:
  "1":
    agent-state: started
    agent-version: 1.18.1
    dns-name: dlr-112.dlr.com
    instance-id: /MAAS/api/1.0/nodes/node-36c1be14-301b-11e3-8f20-5254004c0416/
    series: trusty
services:
  wordpress:
    charm: local:trusty/wordpress-93
    exposed: false
    relations:
      db:
      - mysql
      loadbalancer:
      - wordpress
    units:
      wordpress/0:
        agent-state: started
        agent-version: 1.18.1
        machine: "1"
        open-ports:
```

- 80/tcp
public-address: dlr-112.dlr.com

This output provides information about the machine where the service was deployed to and about the service itself. Pay close attention to the items in bold:

- The 'relations" field denotes relationships to other services. Since we added a relation to a MySQL service (step 4), we see it listed here.
- The 'agent-state' field denotes the status of service. When is set to 'started', the service is ready for use.
- The 'public-address' field denotes the URL where the service can be accessed at.
- The 'charm' field denotes where the charm came from. In this case, it was deployed from a local repository.

We omitted the status of the MySQL service, but you should see a similar output, with its 'relations' field showing a relation to the wordpress service.

6. We now have a working Wordpress deployment. Figure 7 illustrates the welcome screen when accessing the service via a web browser.

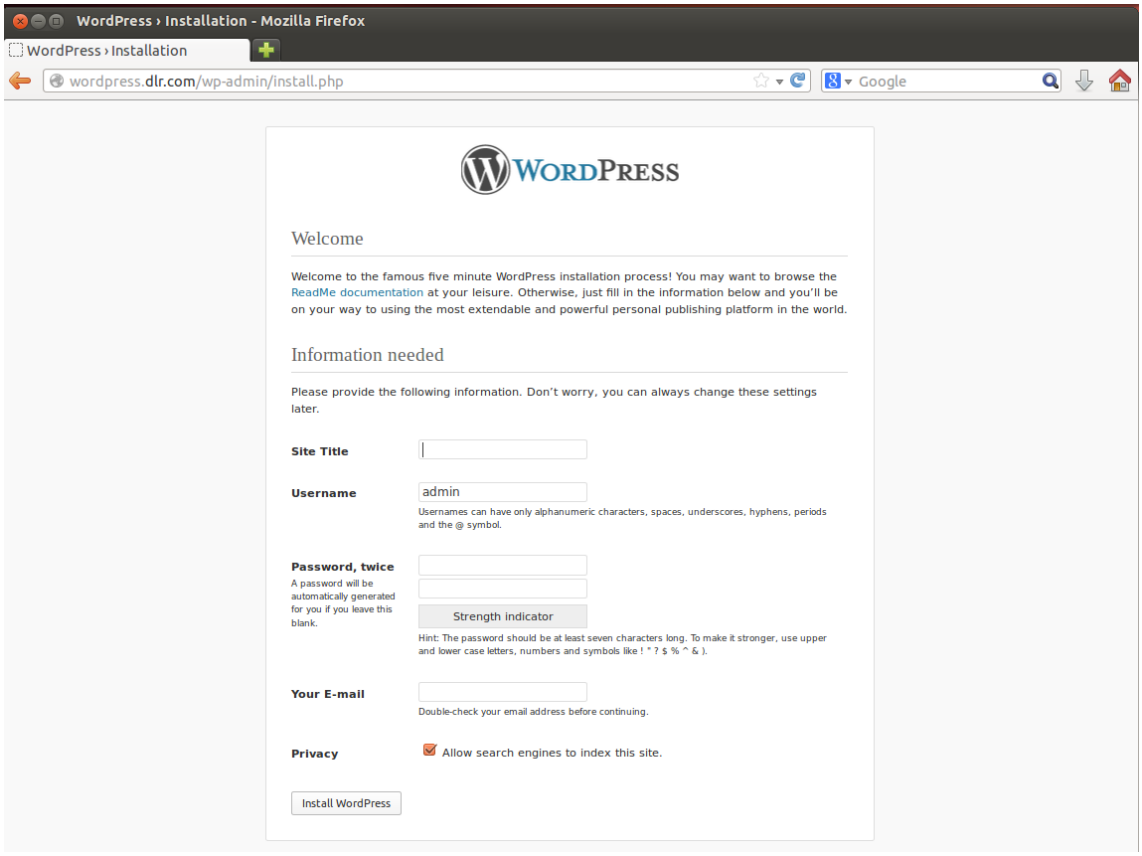

Figure 7 – Wordpress welcome screen

7. For additional configuration options for the Wordpress service, see https://jujucharms.com/trusty/wordpress

4.4. Deploying Juju-GUI

The following is an example of how to deploy Juju-GUI, a graphical tool that allows you to see and manage charms instead. Because juju-gui has a small footprint and for demonstration purposes, in this example we're going to deploy it to the same server as the bootstrap node.

1. Download charm.

   $ mkdir –p /opt/charms/trusty; cd /opt/charms/trusty
   $ bzr branch lp:charms/juju-gui

2. Deploy charm. Run:

   $ juju deploy --to 0 –-repository=/opt/charms local:trusty/juju-gui

   The "--to <machine-id>" parameter tells Juju to deploy the charm to the bootstrap node, which usually has id 0. You can verify the machine id with the "juju status" command.

3. Get status:

   $ juju status juju-gui

   environment: maas
   machines:
     "0":
       agent-state: started
       agent-version: 1.18.1
       dns-name: dlr-89.dlr.com
       instance-id: /MAAS/api/1.0/nodes/node-1e7ad04e-1657-11e3-9a32-c80aa99e96ca/
       series: trusty
   services:
     juju-gui:
       charm: local:trusty/juju-gui-80
       exposed: false
       units:
         juju-gui/0:
           agent-state: started
           agent-version: 1.18.1
           machine: "0"
           public-address: dlr-89.dlr.com

4. After a few minutes, the deployment should be complete. You can access the juju-gui site at its public address, in this case http://dlr-89.dlr.com. Figure 8 shows the Juju-gui console depicting 3 deployed charms: the juju-gui charm itself, plus the Wordpress & MySQL charms deployed in section 4.3, which also shows the relation established between them.

5. For full configuration options, see https://jujucharms.com/trusty/juju-gui

Figure 8 – Juju GUI console showing three deployed charms

4.5. Deploying OpenStack

Deploying a basic OpenStack environment is significantly complex because of the large number of charms required (one for each OpenStack component) and the relationships between them. In this example we include all the steps we took to get us there, but should only be taken as a reference, not as a definitive guide.

We are assuming that you are very familiar with OpenStack. This is a very simple example, and some additional configuration might be needed in your specific environment to get everything working. Note that Swift or Ceph was not deployed here.

This setup uses five servers registered to the MAAS server. The services deployed to them will be:

- One node running as the juju bootstrap node.
- One node running mysql, nova-cloud-controller, glance, keystone, rabbitmq-server, and openstack-dashboard.
- One node running cinder.
- One node running quantum-gateway.
- One node running nova-compute.

1. Download charms

```
$ mkdir -p /opt/charms/trusty; cd /opt/charms/trusty
$ for i in mysql rabbitmq-server cinder quantum-gateway glance keystone nova-cloud-controller
nova-compute cinder openstack-dashboard ; do
> bzr branch lp:charms/trusty/$i
> done
```

2. Create a configuration file in your home directory and call it 'openstack.cfg'. This file will contain any specific settings for each charm. You can read about these settings in each charm's config.yaml file. Some of the charms also contain a README that will help explain some of the settings as well as any other important information. An example configuration file is below:

```
keystone:
  admin-password: 'admin'
  admin-token: 'admin'
nova-cloud-controller:
  network-manager: 'Neutron'
  quantum-security-groups: 'yes'
nova-compute:
  config-flags: 'auto_assign_floating_ip=False'
  enable-live-migration: 'False'
  virt-type: 'kvm'
cinder:
  block-device: '/srv/cinder.img|100G'
  overwrite: 'true'
quantum-gateway:
  ext-port: eth1
openstack-dashboard:
  offline-compression: 'no'
```

3. Bootstrap the environment (if not done already) and deploy services.

```
$ juju bootstrap
$ juju add-machine constraints "mem=12G"
```

The "add-machine" command brings up one of the nodes from MAAS which will become machine 1. We will use this machine to co-locate some of our services onto it using LXC containers. Since we will deploy OpenStack services to 6 LXC containers and since each container can take up to 2GB of memory, we bring up a server with at least 12GB of memory.

```
$ juju deploy --to lxc:1 –-repository=/opt/charms/ local:trusty/mysql
$ juju deploy --to lxc:1 –-repository=/opt/charms/ local:trusty/rabbitmq-server
$ juju deploy --to lxc:1 –-repository=/opt/charms/ local:trusty/keystone --config=openstack.cfg
$ juju deploy --to lxc:1 –-repository=/opt/charms/ local:trusty/openstack-dashboard --
config=openstack.cfg
$ juju deploy --to lxc:1 –-repository=/opt/charms/ local:trusty/nova-cloud-controller --
config=openstack.cfg
$ juju deploy --to lxc:1 –-repository=/opt/charms/ local:trusty/glance
```

Note the use of juju's ability to co-locate multiple services onto one machine using the "--to" parameter. We use the "lxc:1" parameter to tell juju to deploy the given service onto an LXC container on machine 1. This is useful if you are limited on the number of machines you have on hand and you want to install a small cluster for testing or development.

The LXC containers allow us to place each service in its own container, which will prevent charms from potentially stepping on each other's configuration files or trying to bind several services to the same network ports. LXC containers help provide a clean separation of services.

Most OpenStack charms, with the exception of nova-compute, ceph, swift, and quantum-gateway, can be placed safely into LXC containers. The last OpenStack services left to deploy are brought up on separate servers:

```
$ juju deploy –-repository=/opt/charms/ local:trusty/cinder --config=openstack.cfg
$ juju deploy –-repository=/opt/charms/ local:trusty/quantum-gateway --config=openstack.cfg
$ juju deploy –-repository=/opt/charms/ local:trusty/nova-compute –config=openstack.cfg
```

In total, we've deployed OpenStack charms to four machines.

4. When the services are deployed, set up all of the relationships:

```
$ juju add-relation keystone mysql
$ juju add-relation nova-cloud-controller mysql
$ juju add-relation nova-cloud-controller rabbitmq-server
$ juju add-relation nova-cloud-controller glance
$ juju add-relation nova-cloud-controller keystone
$ juju add-relation nova-compute nova-cloud-controller
$ juju add-relation nova-compute mysql
$ juju add-relation nova-compute rabbitmq-server:amqp
$ juju add-relation nova-compute glance
$ juju add-relation glance mysql
$ juju add-relation glance keystone
$ juju add-relation glance cinder
$ juju add-relation cinder mysql
$ juju add-relation cinder rabbitmq-server
$ juju add-relation cinder nova-cloud-controller
$ juju add-relation cinder keystone
$ juju add-relation quantum-gateway mysql
$ juju add-relation quantum-gateway rabbitmq-server
$ juju add-relation quantum-gateway nova-cloud-controller
$ juju add-relation openstack-dashboard keystone
```

5. Point your browser to the machine that is running the openstack-dashboard (Horizon) and log in to the web interface using the admin username and keystone admin-password you set in your configuration file. If you are not sure which system the dashboard is deployed to, you can run the following command and look for the 'public-address' parameter:

```
$ juju status openstack-dashboard
```

You can then navigate to the dashboard using http://<dashboard-address>/horizon

6. You can use the Horizon dashboard as well as the command line to complete any post-deployment tasks, such as setting up networks, uploading images, configuring security groups, adding users, etc.

7. You can easily scale up your OpenStack environment by adding more nodes. For example, once you enlist and commission a new node via MAAS, you can then add it as an OpenStack compute node by running:

```
$ juju add-unit nova-compute
```

4.6. Saving your work with bundles

Once you have all your charms set up and working you can save the configuration as a "bundle" that you can then use to recreate your charms and their relationships in a different deployment.

To export your charm configuration, go to the juju-gui screen and click on the Export Bundle icon on the Juju GUI masthead or use the keyboard shortcut "shift-d". This will save the configuration to a file named "*export.yaml*" (although you can name it differently) and share with others. Here is what our bundle configuration file would look like when we export our environment after deploying the Wordpress and MySQL charms in sections 4.3:

```
envExport:
  services:
    mysql:
      charm: "local:trusty/mysql-312"
      num_units: 1
      annotations:
        "gui-x": "481.1520386633123"
        "gui-y": "542.9394368158863"
    wordpress:
      charm: "local:trusty/wordpress-93"
      num_units: 1
      expose: true
      annotations:
        "gui-x": "882.8479613366877"
        "gui-y": "542.9394368158863"
  relations:
    - - "wordpress:db"
      - "mysql:db"
  series: trusty
```

Note: some of these values, especially the x and y coordinates, will be different in your environment.

To import a bundle, you can drag and drop the configuration file into the juju GUI or use the Import Bundle icon on the Juju GUI masthead. You can also use the CLI using quickstart:

```
$ sudo apt-get install juju-quickstart
$ juju quickstart -n <bundle-file>
```

For more details, see https://juju.ubuntu.com/docs/charms-bundles.html

4.7. Handling charm deployment errors

From time to time, something can go wrong in the deployment of a charm. For example,

```
$ juju status wordpress

 [ ...]
 wordpress:
   charm: local:trusty/wordpress-94
   exposed: false
   relations:
     loadbalancer:
     - wordpress
   units:
     wordpress/0:
       agent-state: error
       agent-state-info: 'hook failed: "install"'
       agent-version: 1.18.1
       machine: "5"
       public-address: dlr-62.dlr.com
```

Here something failed in the deployment of the wordpress charm. The 'hook failed" error message is not very descriptive, so you will have to ssh to machine 5 and look at the log files in /var/log/juju:

```
$ juju ssh 5
```

Be sure you are using the SSH public key you uploaded to MAAS in section 2.4, otherwise you will not be able to ssh to the node.

Once you have fixed the hook error, you must re-run the charm deployment process:

```
$ juju resolved --retry wordpress/0
```

If the error wasn't really an error and you would like to continue as if the hook had completed successfully, run:

```
$ juju resolved wordpress/0
```

For more information, see https://juju.ubuntu.com/docs/authors-hook-errors.html

4.8. Removing services

When you no longer need a service (charm), you can easily remove it so that you can repurpose the underlying hardware. After verifying that the service is no longer being used, run this:

$ juju remove-service <service-name>

The service may take a few moments to shut down. Keep checking with "juju status". After the service has been removed, the underlying hardware (machine) is still in use. To release the machine (assuming no other services are also using it), run this:

$ juju terminate-machine <machine>

If there is another charm deployed to this machine, the above command will fail.

To destroy your entire Juju environment, which includes all services, machines and the bootstrap node, run this:

$ juju destroy-environment <environment-name>

Because this will destroy everything and return all nodes to the 'Ready' state in MAAS, you will be prompted to confirm the action.

5. Conclusion

Using MAAS and Juju together can provide an easy and powerful way to quickly deploy applications without having to learn the installation and configuration details for each application. An application can be deployed in one step and configured with a few simple commands.

Applications can be quickly scaled up or down as requirements change. When an application is no longer required, it can quickly be disposed of and return the underlying hardware to a common pool so it can readily be used again.

For a more detailed implementation of MAAS and Juju, you can refer to "Ubuntu Cloud Documentation – 14.04.LTS".

If you are ready to try MAAS and Juju on Dell PowerEdge or PowerEdge-C servers, contact your Dell or Canonical sales representatives, we are here to help you.

6. Appendices

Appendix A – MAAS Command Level Interface

MAAS provides Command Level Interface (CLI) commands that exposes the APIs that you can then use to script out functionality or if you simply prefer working with a command console instead of a web UI. The documentation at http://maas.ubuntu.com/docs/maascli.html#maas-cli-commands provides a fairly complete list of all available commands, which we will not cover here except for a few useful examples.

Getting the API Key

To be able to run MAAS CLI commands, you need the API key for your MAAS environment, which can be easily obtained by logging in to the web UI:

http://maas.ubuntu.com/docs/maascli.html#api-key

All CLI commands require a profile name, which defaults to 'maas' and is what we will use here.

Logging in

$ maas login <profile> http://<maas-server-name>/MAAS/api/1.0 <api-key>

The command above would then look something like:

$ maas login maas http://maas.dell.com/MAAS/api/1.0 ppJ7AaR6Dk9:jagaFK9yz4FwwQcMWGJ4FF

Listing nodes

Return list of all registered nodes in JSON format:

$ maas maas nodes list

An example output for a node looks like:


```
  {
    "status": 4,
    "macaddress_set": [
      {
        "resource_uri": "/MAAS/api/1.0/nodes/node-418b24b6-1658-11e3-9a32-
c80aa99e96ca/macs/60:eb:69:dc:3c:d2/",
        "mac_address": "60:eb:69:dc:3c:d2"
      },
      {
        "resource_uri": "/MAAS/api/1.0/nodes/node-418b24b6-1658-11e3-9a32-
c80aa99e96ca/macs/60:eb:69:dc:3c:d3/",
        "mac_address": "60:eb:69:dc:3c:d3"
      }
    ],
```

```
    "netboot": true,
    "hostname": "dlr-194.dlr.com",
    "power_type": "ipmi",
    "system_id": "node-418b24b6-1658-11e3-9a32-c80aa99e96ca",
    "architecture": "amd64/generic",
    "tag_names": [
        "metal"
    ],
    "resource_uri": "/MAAS/api/1.0/nodes/node-418b24b6-1658-11e3-9a32-c80aa99e96ca/"
}
```


Accepting nodes

To accept all nodes that are in the 'Declared' state (so they can be commissioned):

$ maas maas nodes accept-all

Defining nodes power parameters

Specify power parameters to start and shut down servers:

$ maas maas node update <system-id> power_type="ipmi" power_parameters_power_address=<ip-address> power_parameters_power_user=<idrac-login> power_parameters_power_pass=<idrac-password>;

To get a node's system-id, list all nodes and extract information from there. Note that the parameters will depend on the systems management type. The above example is for a system that uses IPMI.

Deleting nodes

If a node is allocated to a user, it must first be released:

$ maas maas node release <system-id>

Then delete it:

$ maas maas node delete <system-id>

To get a node's system-id, list all nodes and extract information from there.

Appendix B - References

Guide	URL
MAAS User Guide	http://maas.ubuntu.com/docs/
Juju User Guide	https://juju.ubuntu.com/docs/
Juju Charm Store	https://jujucharms.com/
Ubuntu OpenStack HA reference architecture	https://wiki.ubuntu.com/ServerTeam/OpenStackHA
Ubuntu Cloud Infrastructure	https://help.ubuntu.com/community/UbuntuCloudInfrastructure
Juju Public Mailing List	https://lists.ubuntu.com/mailman/listinfo/juju