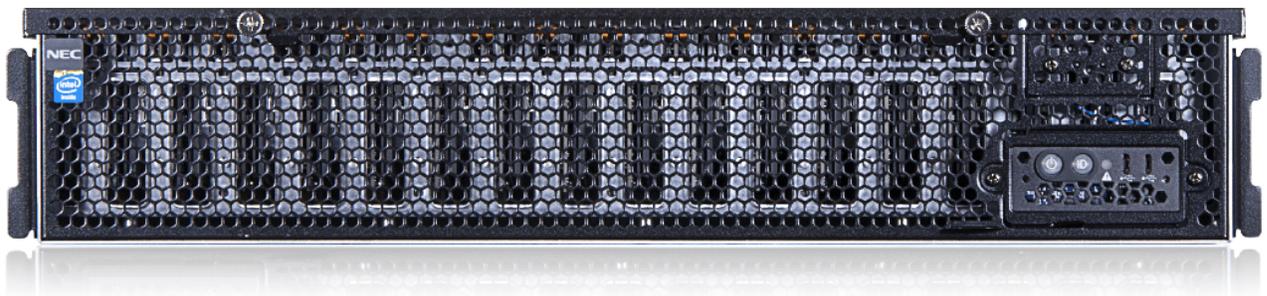


DEPLOYMENT GUIDE

Deploying an OpenStack  
Private Cloud on an  
**NEC DX1000  
MicroServer Chassis**



A Principled Technologies deployment guide commissioned by NEC.

# TABLE OF CONTENTS

<b>Table of contents .....</b>	<b>2</b>
<b>Introduction .....</b>	<b>3</b>
<b>About microservers.....</b>	<b>3</b>
<b>The NEC Microserver and OpenStack solution.....</b>	<b>4</b>
The NEC DX1000 MicroServer Chassis.....	4
The Intel Atom processor C2000 product family .....	6
Ubuntu OpenStack.....	6
<b>Private cloud deployment guide .....</b>	<b>9</b>
Deployment considerations.....	9
Configuring the DX1000 Microserver management interface .....	9
Installing and configuring the MAAS server .....	10
Installing and configuring the Juju server .....	14
Configuring OpenStack components .....	21
Expanding your OpenStack Environment .....	21
Validating the OpenStack installation .....	22
<b>Conclusion .....</b>	<b>28</b>
<b>Appendix A – Detailed configuration steps .....</b>	<b>29</b>
Configuring DX1000 Micro Modular server networking .....	29
Installing MAAS Ubuntu 14.04 Server Edition as a Virtual Machine	30
Configuring Ubuntu MAAS Server .....	31
Adding nodes to the system .....	32
Configuring Juju Management Server .....	32
Creating Juju bootstrap environment.....	34
Deploying OpenStack using Juju GUI .....	34
Configuring OpenStack and deploy .....	34
Scaling OpenStack components using Juju GUI.....	35
Launching an OpenStack instance .....	35
Accessing an OpenStack instance .....	36
Resizing an OpenStack instance .....	36
<b>Appendix B – YAML File for OpenStack on the NEC DX1000 .....</b>	<b>37</b>
<b>Appendix C – OpenStack_config.sh file .....</b>	<b>42</b>
<b>Appendix D – Load testing with DVD Store.....</b>	<b>44</b>
DVD Store.....	44
Configuring the database workload client.....	44
Configuring the database.....	44
<b>About Principled Technologies .....</b>	<b>45</b>

## INTRODUCTION

Companies ranging from medium-sized businesses to enterprises to service providers are increasingly looking to cloud-based computing for their private, in-house computing requirements. The flexibility and agility that cloud computing offers bring many advantages in management, self-service, power, and efficiency to companies. Provisioning of resources that once took days or weeks can now happen in seconds or minutes with a well-designed private cloud.

Unlike a public cloud solution, a private cloud also gives you the security, control, and performance that come with a dedicated environment. Those who can benefit from a private cloud deployment include companies with small databases and Web servers, service providers, large labs, training facilities, and others.

Of late, there has been a convergence of microserver technologies, such as the new NEC DX1000, and open source private cloud solutions, such as OpenStack. The Intel-powered NEC DX1000 nodes include technologies such as multi-core C2000 family of Intel Atom processors and onboard SSD storage. OpenStack has evolved from an open-source project combining code from NASA and RackSpace for compute and object storage, to now a mature and constantly updated platform that includes a mature compute platform, advanced software-defined networking, block storage, and other components. This deployment guide will show you how to prepare for, deploy, provision to, and scale an OpenStack private cloud using a NEC DX000 Micro Modular server.

## ABOUT MICROSERVERS

A microserver solution such as the NEC DX1000 comprises a chassis that contains many standalone compute nodes and some shared infrastructure components, such shared networking, power supplies, and management interfaces. This concept is similar to that of more traditional blade server environments, but a microserver may have lower power and cooling requirements and greater densities, and a lower cost per unit.

Microservers typically are used in applications using a large numbers of server nodes, such as private cloud environments, Web server farms, HPC environments for parallel processing, and compute nodes for Hadoop and big data. Their use in cloud computing is growing and continued growth is likely.

# THE NEC MICROSERVER AND OPENSTACK SOLUTION

In this section, we provide an overview of the hardware and software we use in our private cloud deployment.

Figure 1 shows the server chassis, modules, and switches in our solution.

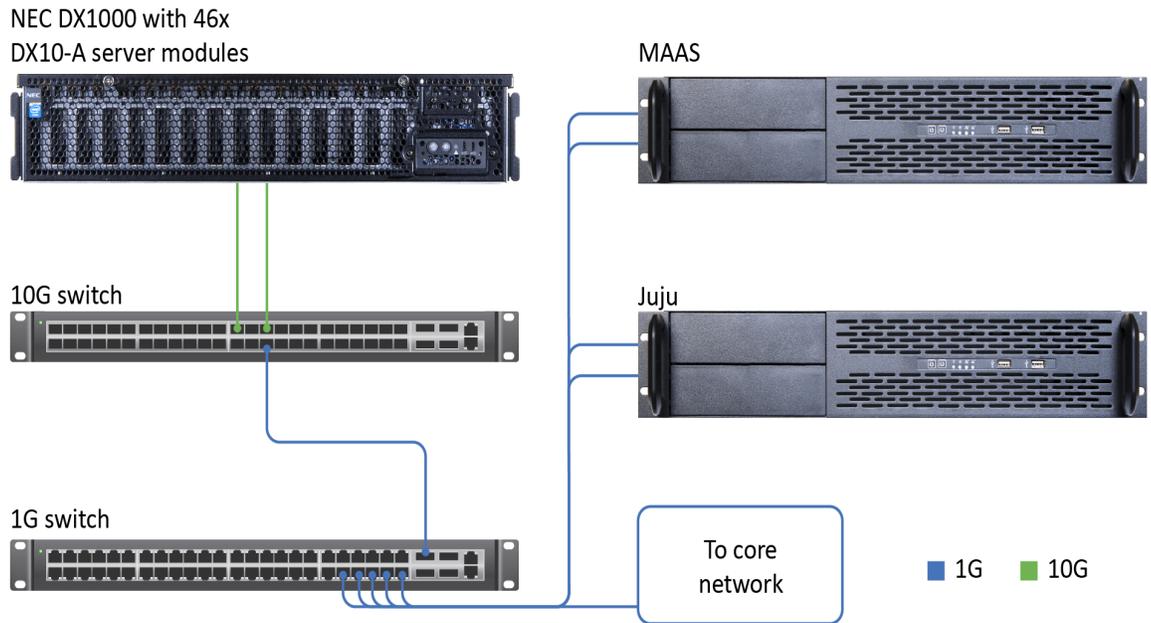
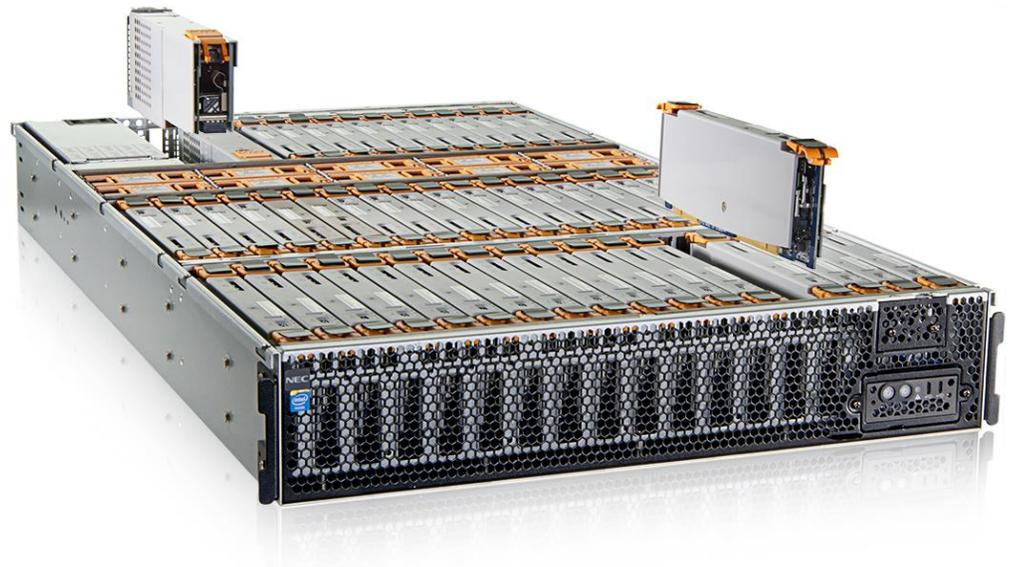


Figure 1: The hardware components of the solution.

## The NEC DX1000 MicroServer Chassis

NEC designed the DX1000 MicroServer platform to support open solution platforms while satisfying business requirements for cloud services and big-data analytics. Companies can use this platform to lower their energy and operational costs, and increase space efficiencies.

The NEC DX1000 MicroServer chassis is an ultra-dense server system with a maximum of 46 one-way servers in a 2U enclosure. Powered by redundant 1600W power supplies similar to those found in traditional two-socket, two-rack unit systems, the DX1000 chassis contains two integrated switches, each with redundant 40Gbps uplinks providing ample bandwidth for every node in the chassis. Each node contains an eight-core Intel Atom processor, 32 GB of high-speed RAM, and a 128GB SSD for low-latency, high-speed storage. A single standard server rack can house up to 16 chassis, for a maximum of 736 servers per rack.



**Figure 2: The ultra-dense NEC DX1000 Microserver Chassis.**

Powered by the Intel Atom C2000 processor and the high-port-count, low-latency Intel FM5224 Ethernet Switch, the NEC DX1000 MicroServer is optimized for innovative applications for hyper-scale datacenters. The NEC DX1000 MicroServer contains two switches, shown below in Figure 3, providing up to four 40Gb uplinks to your core network.

**Figure 3: The NEC DX1000 contains two redundant switching modules, offering high-bandwidth uplinks and low-latency intra-chassis communication.**



Inside the chassis, each DX10a server node connects to the redundant switches using 2.5Gbps connections—more than double the bandwidth available in traditional gigabit Ethernet networking. This greater bandwidth, along with other advanced switching functions, means the DX1000 MicroServer offers fast, flexible, low-latency networking, and the ability to contain traffic within the chassis, reducing the load on your core network.

The DX10a server nodes, shown in Figure 4, are built on the Intel Atom C2000 processor family. In addition to the multi-core CPU architecture, each node contains 32

GB of RAM, an integrated 128GB SSD drive, and dual 2.5Gb NICs, making the DX10a nodes an excellent choice for cloud-based distributed computing.

Figure 4: The NEC DX1000 houses up to 46 DX10a compute nodes.



## The Intel Atom processor C2000 product family

The Intel Atom processor C2000 family incorporates five models ranging from a two-core, 1.7 GHz CPU to an eight-core, 2.4 GHz CPU. The C2000 family is made of 64-bit system-on-chip (SoC) processors for lightweight scale-out workloads such as Web servers or cold data analytics. The Intel Atom C2000 family is designed for extreme density and energy-efficiency for these workloads. These systems have built-in GbE, SATA, and USB ports and have server-class features such as Error Correcting Code (ECC) memory and Intel Virtualization Technology®.

To learn more about the Intel Atom processor C2000 product family and the C2750 we used in our tested, visit [www.intel.com/content/www/us/en/intelligent-systems/rangeley/atom-c2000-product-family-based-platforms-overview.html](http://www.intel.com/content/www/us/en/intelligent-systems/rangeley/atom-c2000-product-family-based-platforms-overview.html)

## Ubuntu OpenStack

### Ubuntu OpenStack overview

Ubuntu OpenStack is a modular Infrastructure-as-a-Service (IaaS) cloud software built on Ubuntu Linux operating system. It is the most commonly deployed major OpenStack distribution deployed in production cloud environments.<sup>1</sup>

Ubuntu OpenStack provides a variety of tools that makes rapid deployment, configuration, and management of your underlying cloud resources straightforward. The LTS versions of Ubuntu provide long-term support and stability for the base operating systems powering these tools.

---

<sup>1</sup> [www.ubuntu.com/cloud/openstack](http://www.ubuntu.com/cloud/openstack)



Figure 5: The OpenStack components in our cloud environment.

## Ubuntu OpenStack toolset

Ubuntu MAAS, Ubuntu’s Metal-as-a-Service provider, provides network boot and IPMI capabilities, allowing unused hardware to remain in a powered-down state until additional cloud resources are commissioned. Once discovered via PXE, new hardware is configured for network booting and powered down until needed for provisioning. A Metal-as-a-Service provider is required for deploying headless servers – servers without physical consoles, such as the kind typically found in microserver platforms. Ubuntu MAAS can also perform hardware-specific tasks such as burn-in tests, firmware and RAID upgrades — and confirming that your hardware is Ubuntu certified.

MAAS provisions hardware dynamically. A Web interface or API makes it easy to add, commission, update, deploy, and recycle physical servers. In response to changing needs, you can quickly add nodes, redeploy nodes from one service to another, and retire them for use outside MAAS.

Working in conjunction with MAAS, Ubuntu Juju is an orchestration service that allows for rapid design and deployment of cloud services to MAAS-managed hardware. Juju provides a graphical interface for selection and deployment of a variety of cloud applications by using “charms.” Juju users drag the charms required for their environment onto a canvas or workspace, define the relationships between the charms,

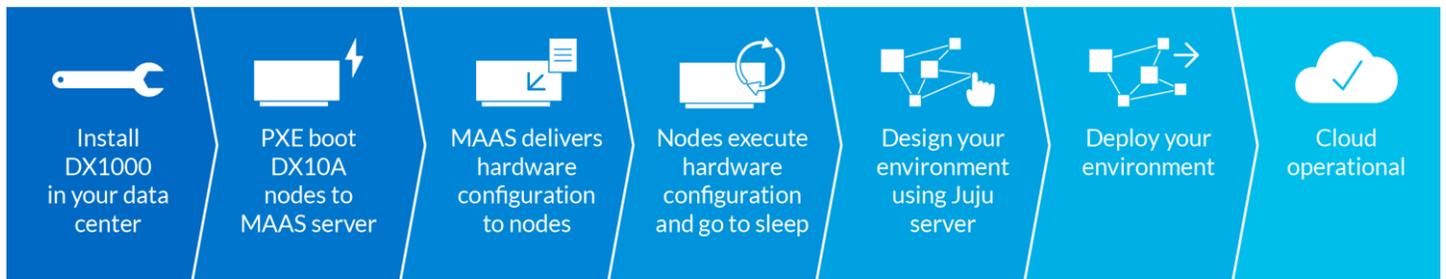
customize the settings of each component, and deploy their new environments with relative ease.

You can download individual charms or even complete bundles that contain both charms and blueprints for entire environments already populated with all the necessary relationships already defined. Visit [www.jujucharms.com](http://www.jujucharms.com) for details.

Ubuntu Landscape is a cloud systems management tool used to monitor OpenStack clouds for issues; it can be used for alerting to ensure you always know when problems arise. Implementing Ubuntu Landscape is beyond the scope of this guide. For our deployment guide, we focus on Ubuntu Server, MAAS, and Juju.

## How it works

This section describes the deployment process, shown in Figure 6 below.



**Figure 6: The OpenStack deployment process for the NEC DX1000 MicroServer chassis.**

When a network bootable node is introduced into the deployment environment, such as the DX10a nodes contained within the NEC DX1000, the node sends out a request for boot instructions. The MAAS server answers the request and provides configuration instructions to the new node, including settings for IPMI control. The MAAS server records the system information of the requesting node, and instructs the DX10a node to power down.

When a new cloud service is defined and deployed within Juju, a configuration script is created and passed from Juju to the MAAS server. The configuration script contains information regarding the number of servers, hardware requirements needed to deploy the service, and the various software parameters used to configure the cloud components for interoperability.

The MAAS server determines which hardware is available and matches the parameters in the script. The required number of DX10a nodes are designated for a particular role as defined by the Juju configuration script, and are sent power-on instructions by the MAAS server. In our simple deployment, each node receives only one role.

At power-on, each NEC DX10a server performs a network boot, and receives specific instructions from the MAAS server, based on the role designated to it. These

instructions include the location of bootable media for operating system installation, the post-installation packages needed for OpenStack configuration, and post configuration instructions that connect the components to each other, ensuring that all the components work together.

Once the cloud system deployment is complete, you provision virtual machine instances through the OpenStack Dashboard.

## PRIVATE CLOUD DEPLOYMENT GUIDE

In following section, we provide a walk-through of the process of deploying an OpenStack private cloud on the NEC DX1000 MicroServer Chassis. For a greater level of detail, see [Appendix A](#).

### Deployment considerations

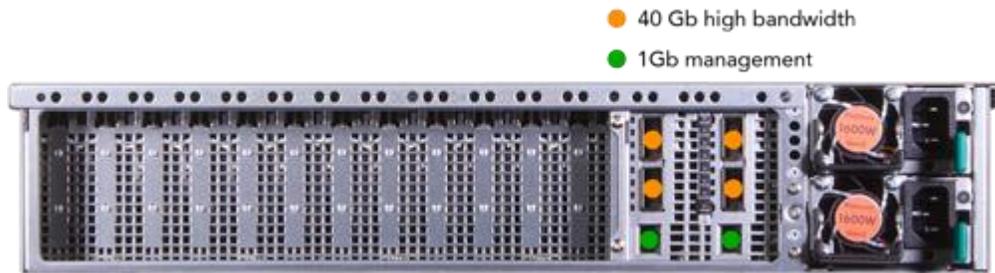
Before beginning the deployment, consider the following points:

- **Internet connectivity.** To execute the following procedure, you'll need to make sure your environment has access to the Internet in order to download installation and update packages for the Linux operating systems. Once the OpenStack private cloud is deployed, you may also need to obtain cloud-ready images you can use for launching instances.
- **Instance sizing constraints.** Each of the DX10a nodes contains eight processor cores, 32 GB of RAM, and 128 GB of onboard SSD storage. These specifications make the DX1000 a great choice for cloud deployments where many small, purpose-built VMs will be rapidly deployed to meet ever-changing load requirements. In cases where OpenStack default X-Large or custom sized instances will be used, you should add external storage and configure OpenStack Cinder nodes dedicated for storage provisioning. The addition of external storage is outside the scope of this deployment guide.
- **Traffic isolation.** PXE services are used for deploying nodes and applications within your MAAS managed hardware. Though outside the scope of this guide, in most cases PXE traffic should be isolated from production data traffic either by using separate data networks or by confining PXE traffic to a dedicated VLAN.

### Configuring the DX1000 Microserver management interface

To integrate the NEC DX1000 into the deployment environment, you must ensure that you have connectivity to the NEC DX1000 microserver chassis. To do so, follow these general steps. For specific details on each of these steps, see [Appendix A](#).

First, connect the 1Gb management ports on each switch within the NEC DX1000 to a top-of-rack (TOR) switch with access to a DHCP provided management network. The connections are indicated with green in Figure 7.



**Figure 7: The NEC DX1000 network connections.**

Next, connect the high-bandwidth connections on each switch within the NEC DX1000 to a TOR switch with access to PXE, management, and data networks. The high-bandwidth connections are marked above, in Figure 7, as orange.

Configure each of the DX1000 integrated switches using the front-panel micro USB ports. You'll connect a management laptop to these ports using a USB bridge cable. We used a terminal emulation program, such as PuTTY, to open an administrative console session on each switch.

Once you've configured the NEC DX1000 integrated switches for communication in your network environment, you can begin installation of the other infrastructure applications necessary to deploy OpenStack. We'll start with the Ubuntu MAAS server.

## Installing and configuring the MAAS server

The Ubuntu MAAS server is built on the Ubuntu Server operating system. Our MAAS server used four virtual CPU cores, 16 GB of RAM, and 50 GB of disk. You can use either physical or virtual hardware for the MAAS server. You'll need Internet access from the management network so that you can update your operating system and download all of the additional packages required to build the MAAS server.

## Installing the operating system on the MAAS server

All of the individual components required for this solution are built on the Ubuntu 14.04 LTS platform. After installation of the base operating system, define your repositories – Internet resources that house additional software packages – and then install the MAAS packages. See [Appendix A](#) for full details on this process.

```
gpg: keyring `/tmp/tmp29hss65i/secring.gpg' created
gpg: keyring `/tmp/tmp29hss65i/pubring.gpg' created
gpg: requesting key 93EE8CC5 from hkp server keyserver.ubuntu.com
gpg: /tmp/tmp29hss65i/trustdb.gpg: trustdb created
gpg: key 93EE8CC5: public key "Launchpad PPA for MaaS Maintainers" imported
gpg: Total number processed: 1
gpg:         imported: 1 (RSA: 1)
OK
admin@MAAS:~$ sudo add-apt-repository ppa:cloud-installer/testing

More info: https://launchpad.net/~cloud-installer/+archive/ubuntu/testing
Press [ENTER] to continue or ctrl-c to cancel adding it

gpg: keyring `/tmp/tmp29hss65i/secring.gpg' created
gpg: keyring `/tmp/tmp29hss65i/pubring.gpg' created
gpg: requesting key 93EE8CC5 from hkp server keyserver.ubuntu.com
gpg: /tmp/tmp29hss65i/trustdb.gpg: trustdb created
gpg: key 93EE8CC5: public key "Launchpad PPA for MaaS Maintainers" imported
gpg: Total number processed: 1
gpg:         imported: 1 (RSA: 1)
OK
admin@MAAS:~$
```

Figure 8: Adding repositories and installing packages for MAAS.

In addition to defining the repositories and supporting packages, use the command-line interface to install MAAS itself, and next create administrative credentials to use in MAAS.

Once installation of the MAAS service is complete, access the MAAS portal using a Web browser pointed to [maas\\_ip/MAAS/](https://maas.ip/MAAS/) as shown in Figure 9 below. Log in with the administrative credentials you created in the command-line interface.

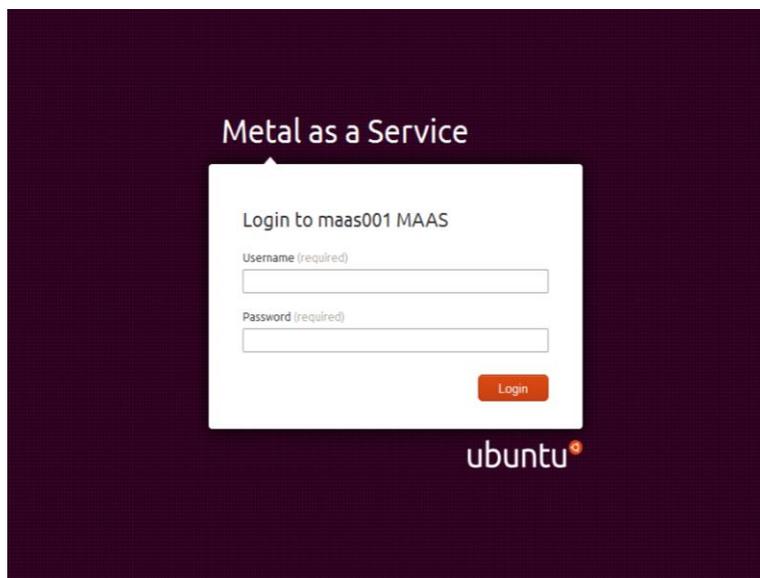


Figure 9: MAAS login screen.

Use the interface to import Linux disk images into MAAS for distribution to MAAS-managed nodes, create SSH keys, and define additional networking parameters. To begin preparing the MAAS environment, first, import the disk images. Shown in Figure 10 below, the disk images we used are cloud-ready Ubuntu operating systems that will be used to run the OpenStack controller software.

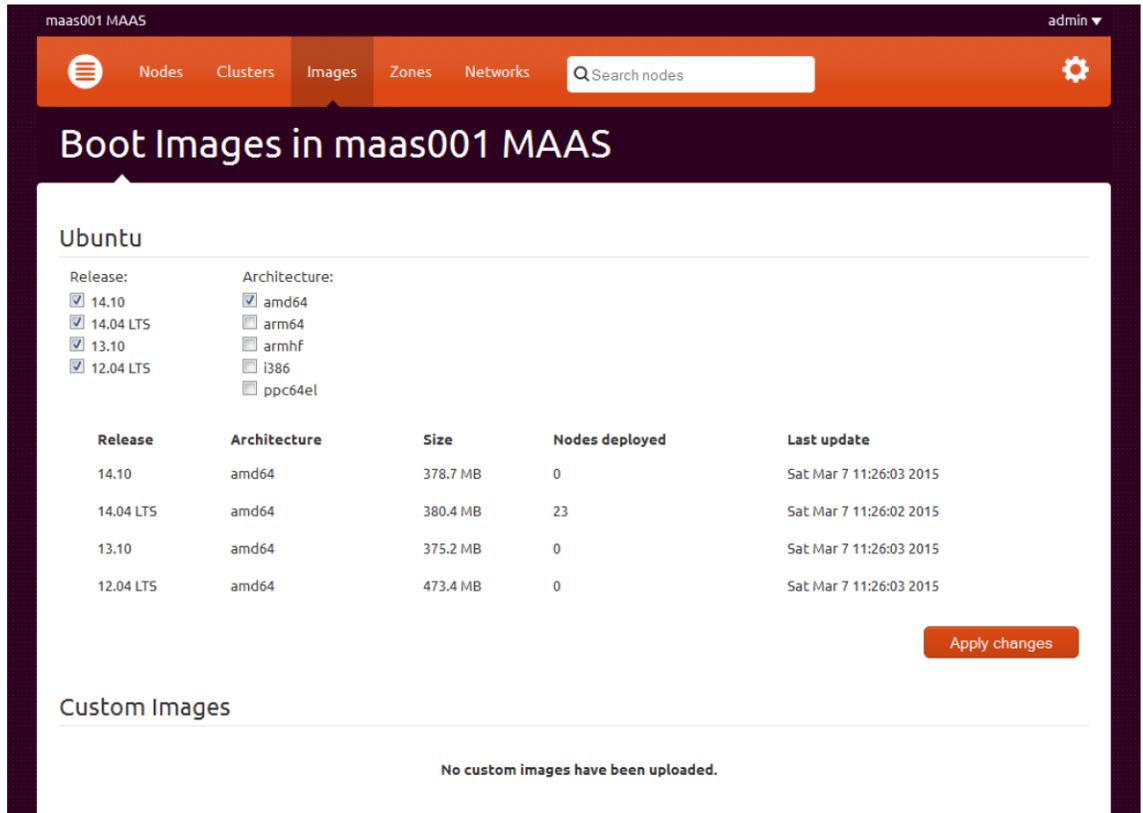


Figure 10: Import the boot images needed for cloud deployment into the MAAS server.

## Configuring networking on the MAAS server

Next, configure the requisite networking. In the cluster section of the Web GUI, shown below in Figure 11, define the NIC that is connected to the deployment network as a DHCP and DNS manager. Next, provide IP addresses ranges for use in a DHCP scope for PXE services, and a pool of static addresses that are assigned after configuration is complete. These ranges should reside within the same network, but not overlap.

The screenshot shows the MAAS web interface for editing a cluster interface. The page title is "Edit Cluster Interface". The interface includes the following fields and options:

- Name:** eth0. Description: Identifying name for this cluster interface. Must be unique within the cluster, and consist only of letters, digits, dashes, and colons.
- Interface:** eth0. Description: Network interface (e.g. 'eth1').
- Management:** DHCP and DNS (selected). Description: If you enable DHCP management, you will need to install the 'maas-dhcp' package on this cluster controller. Similarly, you will need to install the 'maas-dns' package on this region controller to be able to enable DNS management.
- IP (required):** 10.130.1.5. Description: Static IP Address of the interface.
- Subnet mask:** 255.255.0.0. Description: e.g. 255.255.255.0 (defaults to 64-bit netmask for IPv6).
- Broadcast IP:** 10.130.255.255. Description: e.g. 192.168.1.255 (for IPv4 networks only).
- Router IP:** (field is empty).

Figure 11: Configuring the MAAS network settings.

Once configuration of the MAAS server is complete, boot the hardware and wait for your nodes to connect to the MAAS server. You'll see the nodes begin to populate and enter a ready state on the MAAS main screen, represented by a pie graph as shown in Figure 12 below, indicating the total number of nodes versus the number of nodes currently deployed.

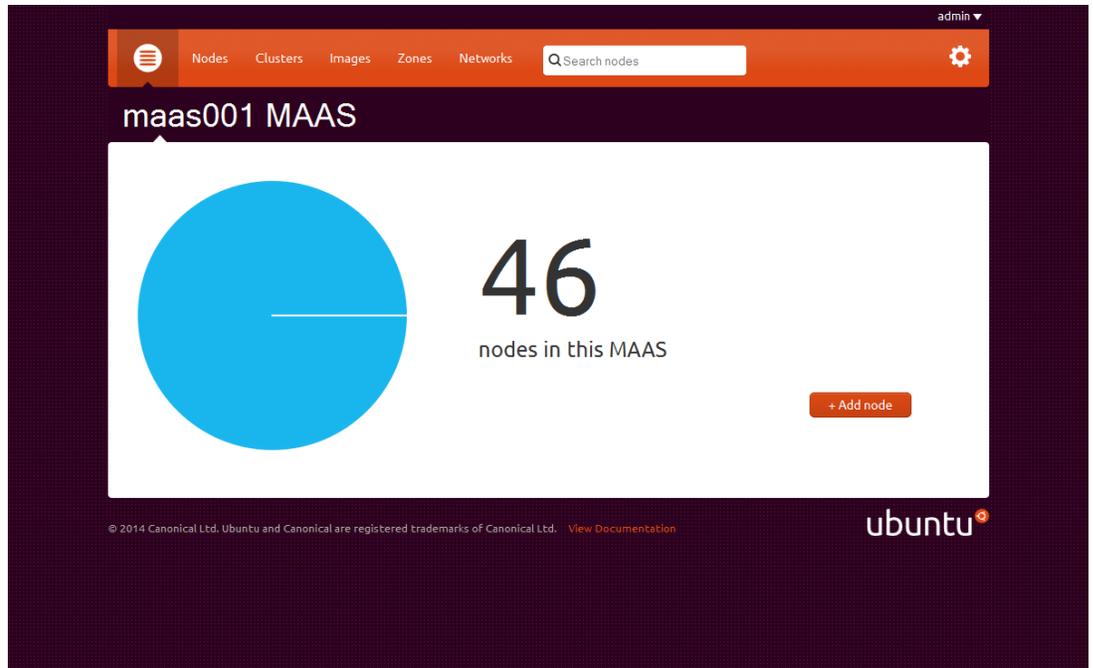


Figure 12: DX1000 nodes populating within our MAAS manager.

The nodes will be assigned unique names within the MAAS system. The unique names are a combination of two random dictionary words, separated by hyphens. For example, at the end of the next steps, our Juju GUI will be deployed to the node named “dramatic-ant.maas.”

## Installing and configuring the Juju server

The Ubuntu Juju server is also built on Ubuntu Server, and uses all of the same repositories and many of the same packages used to configure your MAAS server. Juju is the application manager, and handles orchestration of cloud deployment. The administrator uses Juju to define cloud configurations and to pass that configuration to the MAAS server for deployment via PXE.

## Installing the operating system on the Juju server

As with the Ubuntu MAAS server, Juju is built on the Ubuntu 14.04 LTS platform. Our Juju server used two virtual CPU cores, 8 GB of RAM, and 50 GB of disk. After installation of the base operating system, you’ll need to define the additional repositories and install Juju packages. See [Appendix A](#) for the details on the packages and repositories needed for Juju.

You also need to add the Juju SSH key into the MAAS server. Adding the Juju SSH key creates a trusted relationship within the MAAS server, and allows the MAAS server to accept information from the Juju server. See [Appendix A](#) for the procedure to add the SSH key.

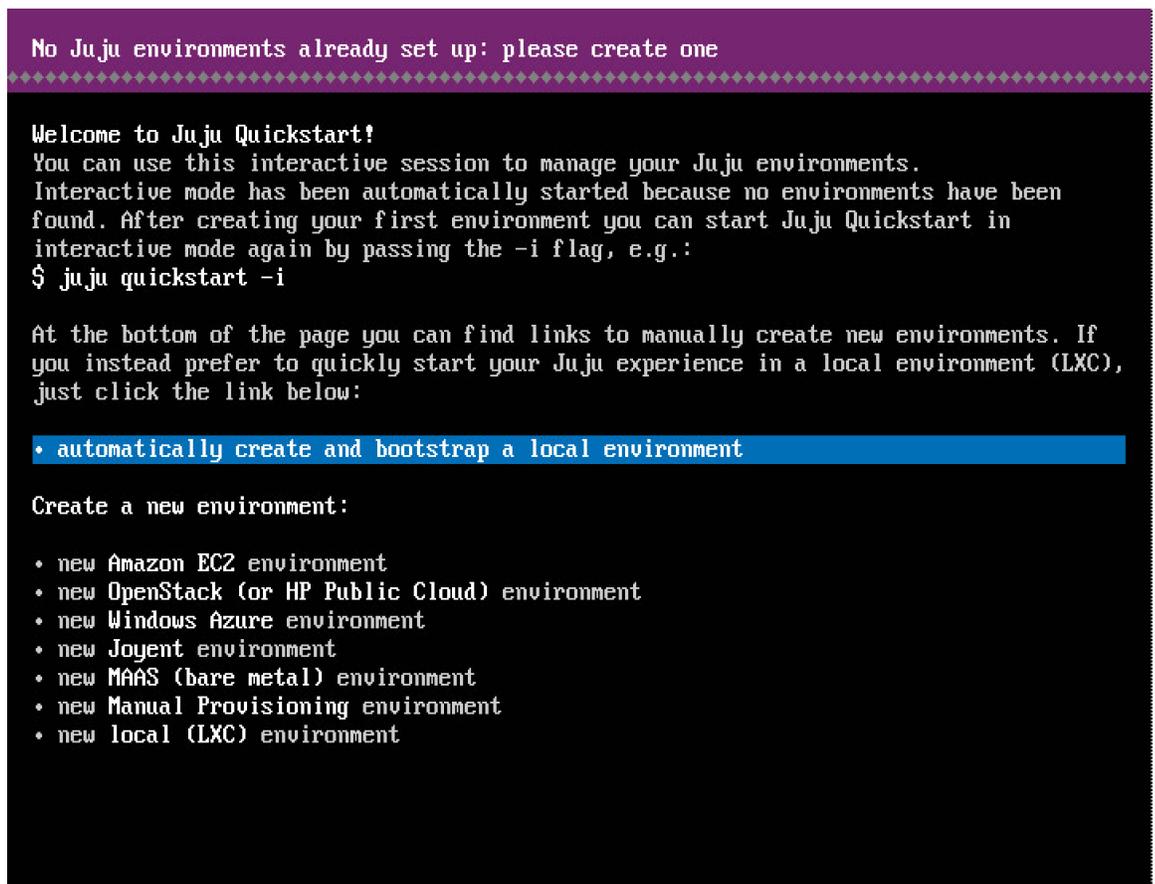
## Configuring and using the Juju bootstrap environment

The last step in this process is to configure a Juju bootstrap environment. This connects the Juju server to the MAAS server, and sets up a GUI with a blank canvas for creating and connecting your cloud environment. At the command prompt, type:

```
juju-quickstart -I --no-browser <enter>
```

A text-based wizard, similar to the one shown below in Figure 13, guides you through the rest of the process. There are a variety of cloud environments you can create using Juju, including custom manual environments.

In this case, we provision the bare metal hardware for the cloud to run on first. After provisioning of the bare metal machines, we will proceed with OpenStack configuration. Under the Create a new environment section, select new MAAS (bare metal) environment.



```
No Juju environments already set up: please create one

Welcome to Juju Quickstart!
You can use this interactive session to manage your Juju environments.
Interactive mode has been automatically started because no environments have been
found. After creating your first environment you can start Juju Quickstart in
interactive mode again by passing the -i flag, e.g.:
$ juju quickstart -i

At the bottom of the page you can find links to manually create new environments. If
you instead prefer to quickly start your Juju experience in a local environment (LXC),
just click the link below:

• automatically create and bootstrap a local environment

Create a new environment:

• new Amazon EC2 environment
• new OpenStack (or HP Public Cloud) environment
• new Windows Azure environment
• new Joyent environment
• new MAAS (bare metal) environment
• new Manual Provisioning environment
• new local (LXC) environment
```

Figure 13: Creating a bootstrap environment in Juju.

The wizard advances to the configuration screen, shown in Figure 14 below, where you'll input information about your MAAS environment, including the MAAS API key and the SSH Public key file. These are used to connect the with the MAAS server and

to allow the MAAS server to automatically distribute the Juju SSH key to the server nodes it manages. Note: The MAAS API key needs to be entered precisely as shown by your MAAS instance.

```
Create a new maas environment

provider type: maas
the provider type enabled for this environment

environment name: Shiba01
the environment name to use with Juju (arbitrary string)

server address: http://10.130.1.5/MAAS
The MAAS server address, for instance http://<my-maas-server>/MAAS.

API key: h8Xns9PF6JCHEQUF7z:JzuHuBU42SHAFWXRPB:gb6qTnAzyhv9FkU4VegumGzgBTaYYTAz
The MAAS API key. This can be found by going to the MAAS user preferences page
(/MAAS/account/prefs/). A link to the account preferences is accessible from the
drop-down menu that appears when clicking your user name at the top-right of the page.

SSH public key file: ~/.ssh/id_rsa.pub
If keys are specified, MAAS will be able to automatically add them to each unit it
manages.

admin secret: admin
the password used to authenticate to the environment
Click here to automatically generate this value

default series: trusty
the default Ubuntu series to use for the bootstrap node

^X exit  ♦ ♦ navigate  ♦ optional field  ♦ field errors
```

Figure 14: Configuring the bootstrap environment in Juju.

Press the down arrow key to the bottom of the text wizard, select Save, and press Enter. Then select Use and press Enter. You'll see a message indicating the environment is being bootstrapped. A quick check on your MAAS server will show one of your nodes has been deployed as a Juju GUI. Your Juju GUI is ready to use as soon as you get a message in your Juju server console session that provides you with a URL and credentials for your environment.

To verify the juju environment has no errors, type

```
juju status <enter>
```

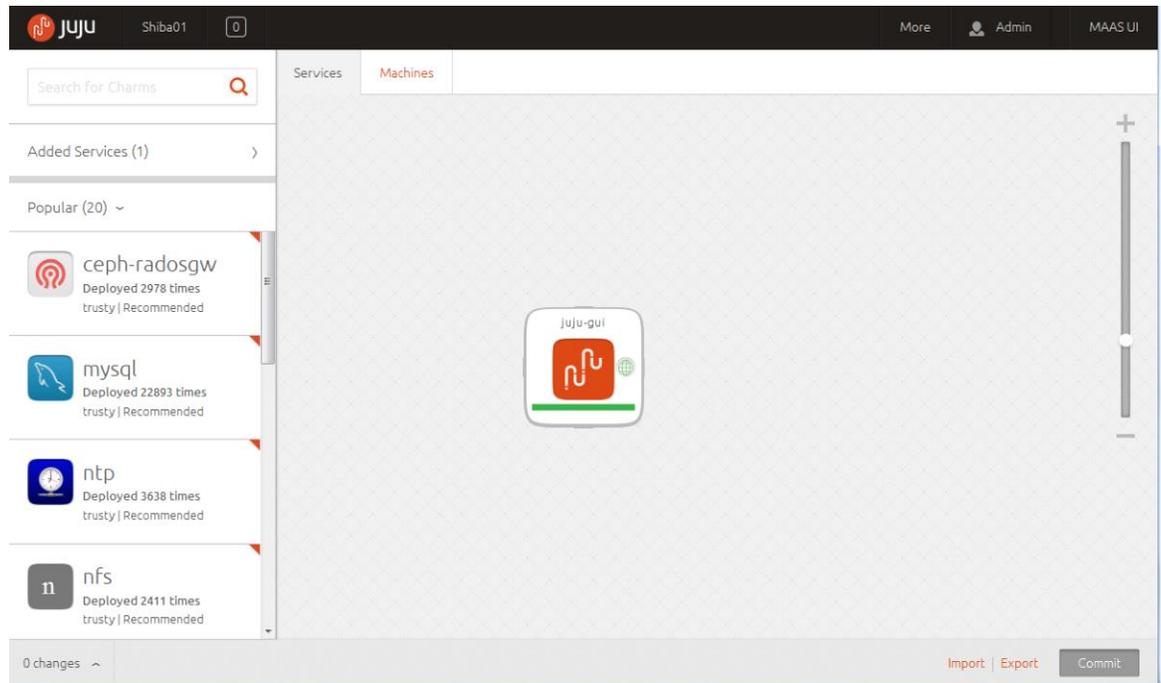
Before exiting the command-line interface type

```
eval `ssh-agent` <enter>.
```

Now we're ready to use the Juju GUI.

## Installing and configuring the Juju charms to use

Open another browser window and connect to the URL provided by the Juju server, accepting any certificates. Log in with the provided credentials, and begin cloud service configuration in the Juju environment, as shown below in Figure 15.

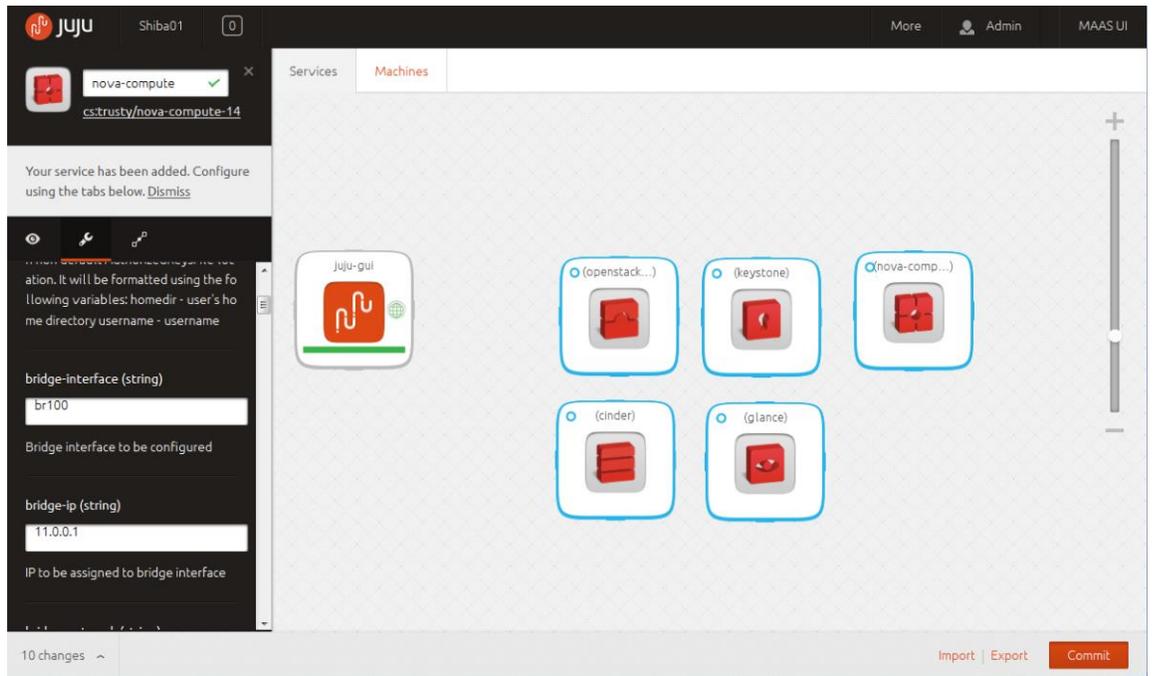


**Figure 15: The Juju GUI configuration environment.**

The icon on the right represents the juju-gui service that was deployed by MAAS. These icons represent Juju charms – predefined components that allow a user to customize a service for their specific environment, and deploy them through MAAS. To add cloud services, use the panel on the left to select and drag charms to the right panel. You can completely customize an OpenStack environment with as many or as few charms as you need for your cloud.

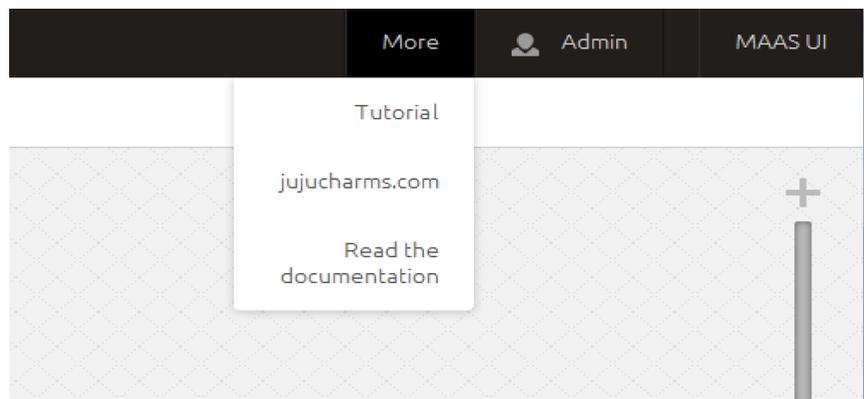
For example, for OpenStack, you need a database server, a message queuing service, a time source to keep all the components synchronized, a controller node, compute nodes, network nodes, storage nodes, and security and authentication components. Additionally, you may choose to deploy monitoring, a user-friendly Web-based dashboard, or advanced storage components.

Drag those component charms listed on the left into your sandbox on the right, begin building relationships between them by connecting them with lines in the GUI, and provide the configuration details for each component, as shown in Figure 16 below. A prerequisite of this guide is basic knowledge of OpenStack components, but for more information on OpenStack in general, see <https://www.openstack.org/>.



**Figure 16: Building the OpenStack configuration.**

For other charms or bundles, click the More menu at the top of the screen, shown below in Figure 17. This will launch a new browser tab and connect to the jujucharms site.



**Figure 17: jujucharms.com in the More options for Juju.**

You can then browse the site for additional charms *or bundles*. Bundles are collections of charms that have already been defined with hardware parameters and relationships built for each component. For example, we found the OpenStack bundle shown below in Figure 18 with 17 different services already preconfigured for deployment.



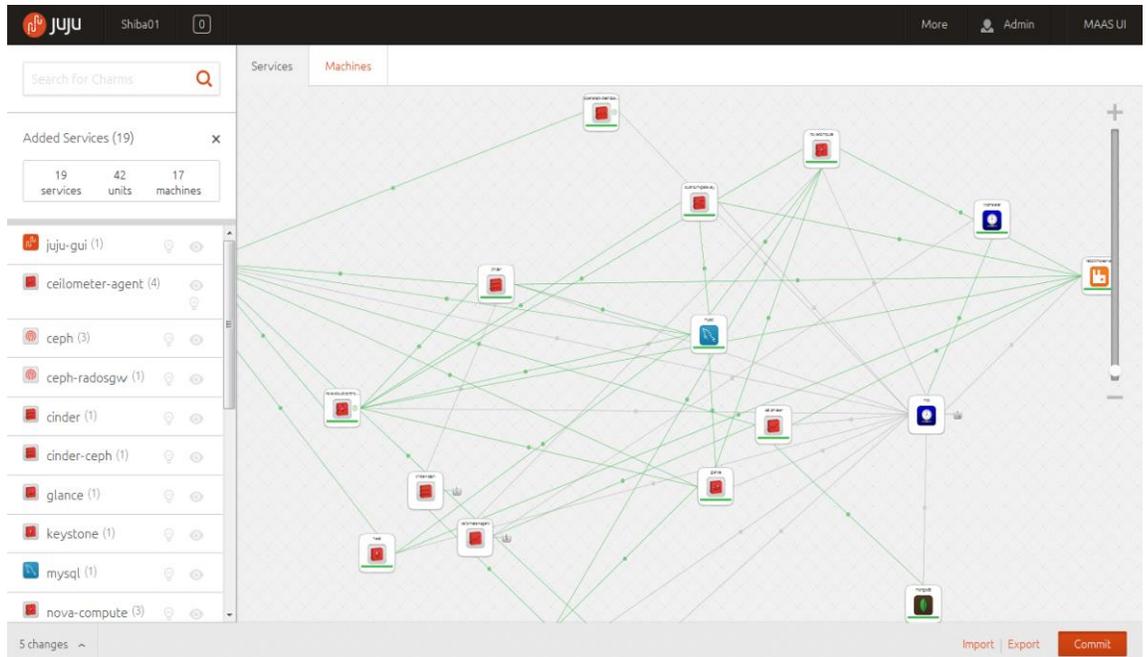
Figure 18: An OpenStack bundle found at [jujucharms.com](http://jujucharms.com).

Click the solution, and review the description. Beneath the download link, you'll see a list of files that are included in the zip file. The "xxxxxxx.yaml" file contains all of the configuration information used by Juju to set up the cloud. Click that file name, and the parameters are displayed in your browser. Copy and paste them into a text file and name the file OpenStack.yaml.

You may need to modify the .yaml file make sure it works with your environment. For example, the .yaml file for the OpenStack bundle on [jujucharms.com](http://jujucharms.com) required two physical disks – one specifically for the OpenStack Ceph component. If your nodes have only a single disk, which is common in microserver environments, you'll have to edit the .yaml file before you can use it.

In [Appendix B](#) of this document, we've provided a .yaml file already customized for the NEC DX1000 Microserver platform. To use it, copy our configuration into your .yaml file, click Import in Juju, browse for the .yaml file, and click OK. The configuration will be imported and will immediately deploy OpenStack to your nodes. You can customize your .yaml file for your environment. For example, we started with three nova-compute nodes defined. To start off with a larger number of compute nodes, modify the "num\_units" definition under "nova-compute" before you upload it.

Return to the Juju GUI and click the Import button in the lower-right corner. Browse for the .yaml file you create, select it, and click OK. The configuration is imported into Juju, with all the relationships already created, and the system is deployed. This process, while automated, can take about 30 minutes to complete. We show a completed deployment in Figure 19.



**Figure 19: OpenStack services and relationships within Juju.**

After the deployment has completed, log into your OpenStack Dashboard. In the Juju GUI, click the menu item for Added Services, scroll down to find your Dashboard service, and click on it. Expand the menu under running units, and click the link for the service. You'll see information about the service, as shown below in Figure 20. Click the IP address to launch the OpenStack Dashboard.

id	endpoint	name	role	scope	status
openstack-dashboard:cluster	(self)	cluster	peer	global	
openstack-dashboard:identity-service	keystone	identity-service	requirer	global	
ntp:juju-info	ntp	juju-info	provider	container	

**Figure 20: Viewing service information for deployed components.**

## Configuring OpenStack components

After we've deployed all the OpenStack components, a small amount of post-installation configuration work must be done to make sure the cloud is ready to deploy virtual machine instances. These tasks include configuration of a tenant, defining the networks for use within that tenant, and obtaining cloud-ready OS distributions for use in your cloud deployment.

We've included a configuration script for this in [Appendix C](#). Executing this script from the Juju server command-line interface will create your tenants, define quotas and networks, modify your instance sizes, and fetch the cloud-ready OS images for you.

After that script has completed, you're ready to start deploying virtual machine instances in your cloud. Log into your OpenStack Dashboard, shown below in Figure 21, using "admin" as both the username and password.

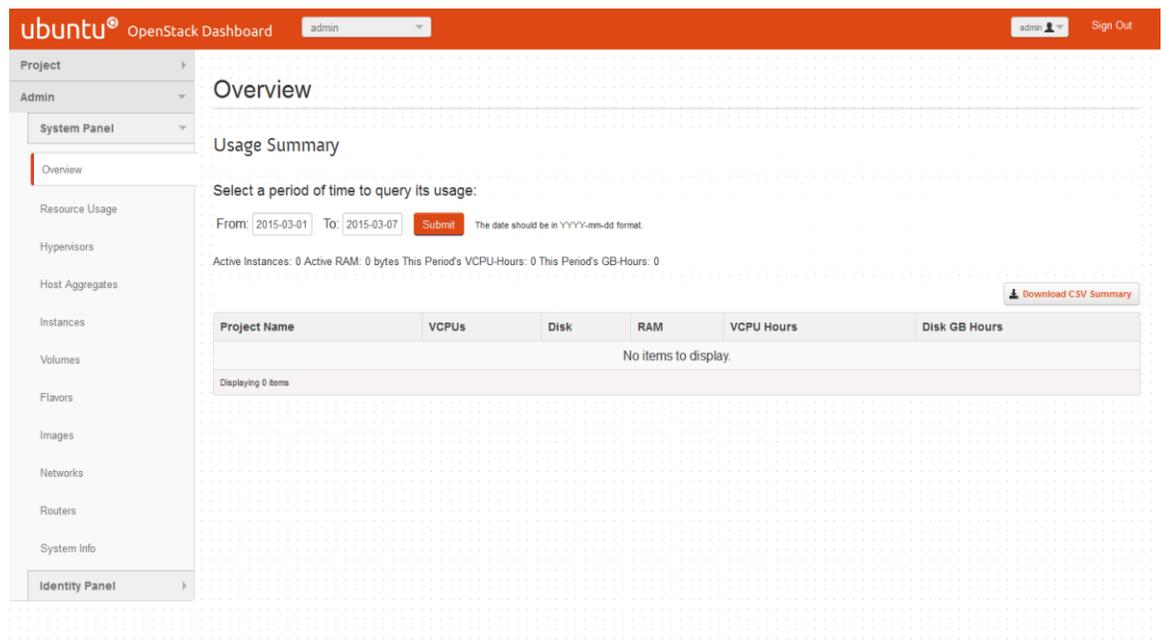


Figure 21: You can use the OpenStack Dashboard for many of your cloud management tasks.

## Expanding your OpenStack Environment

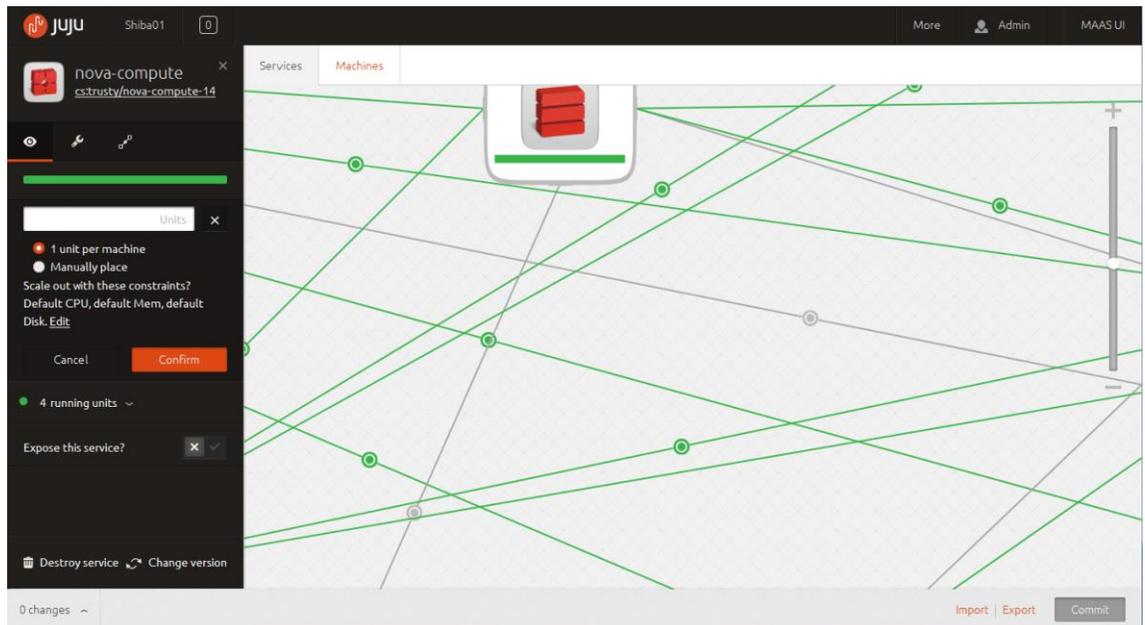
Once your OpenStack cloud has been deployed, you may decide to add additional compute resources, storage nodes, or other components to further enhance the functionality of your cloud. The Juju GUI provides a straightforward method for deploying new hardware to meet your scaling needs.

As shown in Figure 22 below, locate the service component you want to scale, enter the number of new nodes needed for the service, and click confirm. Clicking

Commit in the lower right of the screen gives you the opportunity to review your selections and the choice proceed with deployment or to cancel.

Scaling your environment is as simple as deploying additional NEC DX1000 Microservers, booting the DX10a nodes, and increasing the resources available to your cloud.

See [Appendix A](#) for details on how we scaled our cloud environment.



**Figure 22:** You can scale your cloud to provide as many resources as you need to support the demands placed on your environment.

## Validating the OpenStack installation

In this section, we test our newly deployed environment to demonstrate that it's fully operational. We do this deploying a small virtual machine instance, installing MySQL, and then running a load on it. We'll then resize that instance in order to provide better performance, capturing the amount of time it takes to perform the resize and documenting the amount of downtime during the process.

## Launching OpenStack instances

Launching instances in OpenStack is easy, but before you begin, you should generate a key pair for use with your instances. A key pair is a unique identifier that is split into two parts – a public key and a private key. When you create the key pair, a .PEM file is automatically downloaded to your system. You can use this file to extract the keys, or convert the file to a format useable by your terminal client.

Shown below in Figure 23, in the Project-Compute section of the OpenStack Dashboard, click Access and Security, and select Key Pairs. Click the button for Create

Key Pair. Provide a name for the key pair, and click Create Key Pair. A file will be downloaded to your computer, which will be used to access the instance later.

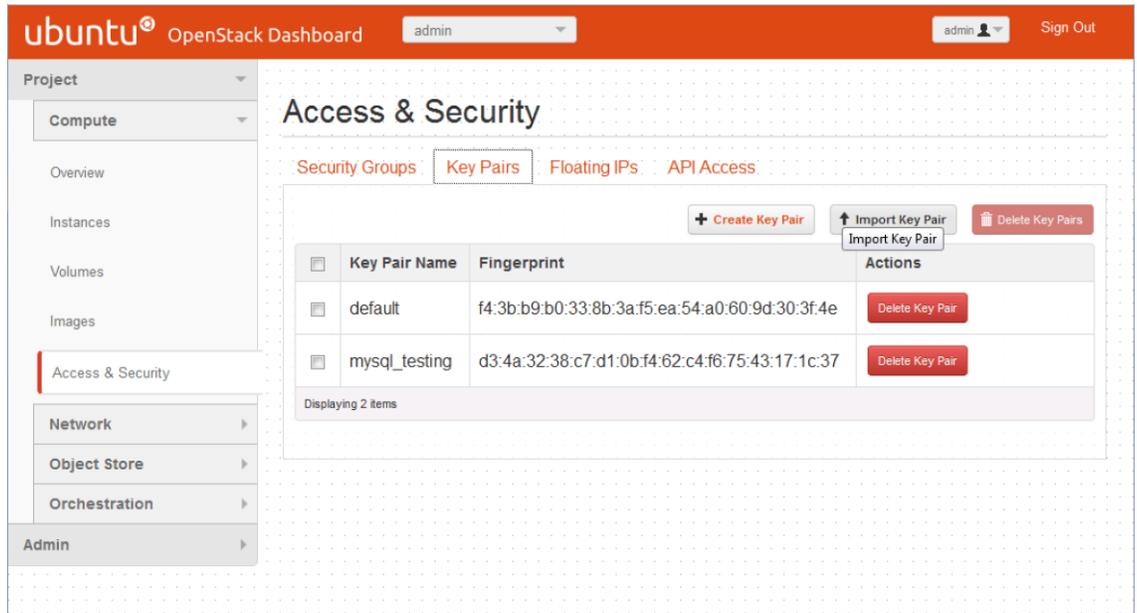


Figure 23: Creating Key Pairs in OpenStack Dashboard for use with your new instances.

Now click the Instances button in the Project-Compute section. Click the Launch Instance button and the window appears, as shown in Figure 24 below. Provide the instance name, select the size of the virtual machine from the pull-down menu, enter the number of VMs you want to deploy, and indicate the boot source. For your first instances, you should select the cloud-ready Linux flavor we downloaded during the execution of the OpenStack configuration script.

Next click Access and Security. Here, you'll select the key pair you want to use for authentication. Click the Networking tab. Add both networks to the instance and click Launch at the bottom right of the window. Your new instance will begin building and will be marked as active when ready for use.

Launch Instance
✕

---

Details \*
Access & Security \*
Networking \*
Post-Creation
Advanced Options

**Availability Zone:**

**Instance Name: \***

**Flavor: \***

**Instance Count: \***

**Instance Boot Source: \***

Specify the details for launching an instance.

The chart below shows the resources used by this project in relation to the project's quotas.

**Flavor Details**

<b>Name</b>	m1.small
<b>VCPUs</b>	1
<b>Root Disk</b>	20 GB
<b>Ephemeral Disk</b>	0 GB
<b>Total Disk</b>	20 GB
<b>RAM</b>	2,048 MB

**Project Limits**

Number of Instances	7 of 100 Used
Number of VCPUs	10 of 200 Used
Total RAM	20,480 of 204,800 MB Used

Cancel
Launch

Figure 24: Launching a new OpenStack Instance.

## Resizing OpenStack instances

OpenStack instances may be initially provisioned with a very small resource footprint, in OpenStack terms called a *flavor*. In cases where higher performance is needed without tearing down and building a new replacement instance, OpenStack offers the ability to resize a provisioned instance.

Resizing an instance adds additional compute resources, such as virtual CPUs, RAM, and disk to provide a performance boost when needed. Resizing an instance is essentially reprovisioning the instance without destroying it. To perform this operation, OpenStack may migrate the instance to another, more available compute node.

### Resizing an active instance

In the Instances section, locate the instance you want to resize, and move your cursor to the far right of the row. Click the More button, and select Resize Instance from

the pull-down menu. You'll select the flavor you want to use for resizing, and then commit the change. The instance resizing status can be viewed in the OpenStack Dashboard, as seen in Figure 25.

See [Appendix A](#) for complete details on resizing active instances.

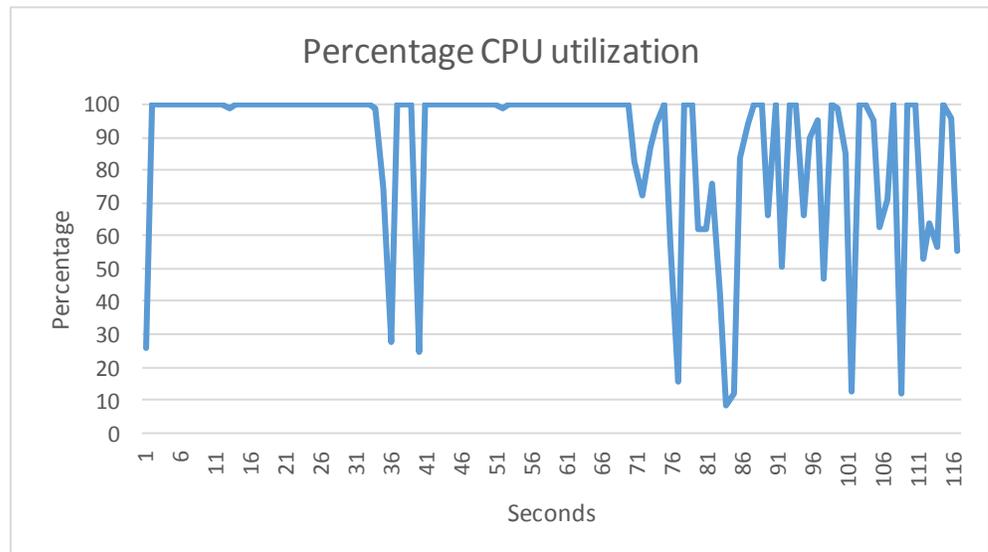
Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Uptime
PGSQL_10GB_TEST	PGSQL_10GB	192.168.14.17 10.130.1.216	m1_small   2GB RAM   1 VCPU   20.0GB Disk	mysql_testing	Resize/Migrate	nova	Resizing or Migrating	Running	39 minutes

**Figure 25: Resizing an active virtual machine instance.**

Next, we'll put a server under a test load and resize it to show the impact that resizing an instance can have on performance.

### Operational validation using a database workload

In this test, we utilized a small database instance, and executed DVD Store to place the server under an intentionally oversized load. Almost immediately, the instance CPU utilization spiked to 100 percent. Operations per second dropped dramatically, and became very erratic after about 71 seconds, and memory utilization rose to just under 800 MB of our available 2 GB of RAM. Figure 26 below shows the CPU utilization of the instance under test, while Figure 27 shows the operations per second. Figure 28 shows the memory utilization during our test run. The small instance was clearly undersized for the load.



**Figure 26: CPU utilization of the small instance before resizing.**

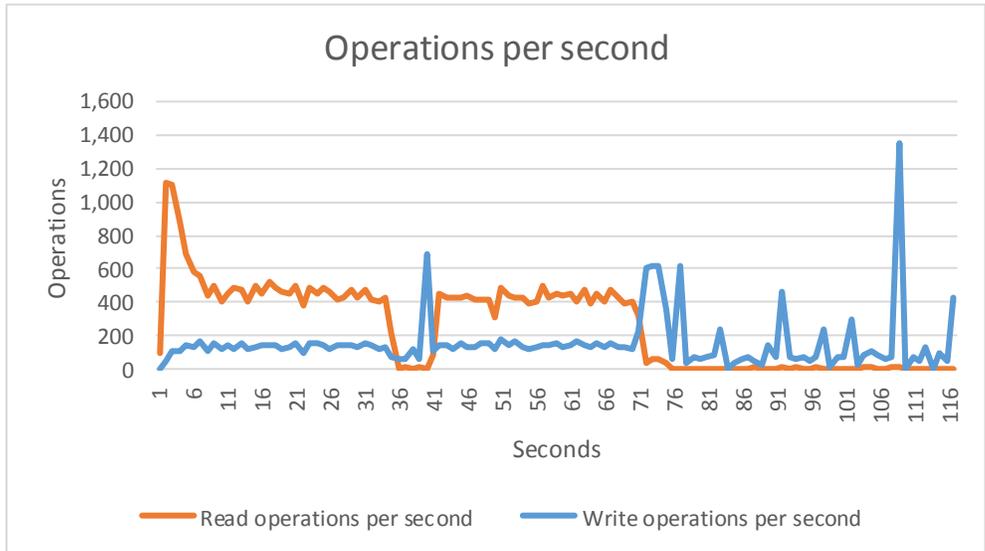


Figure 27: Operations per second on the small instance before resizing.

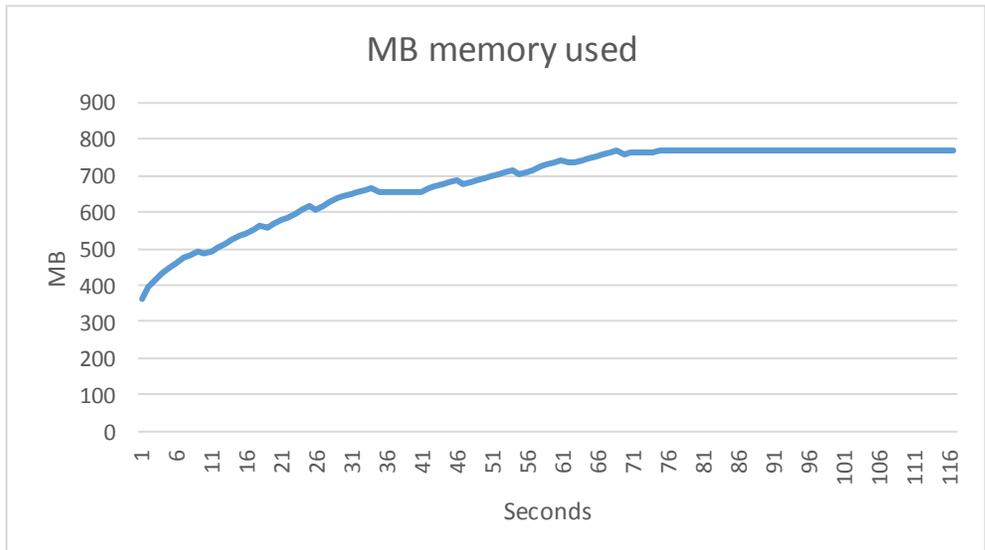
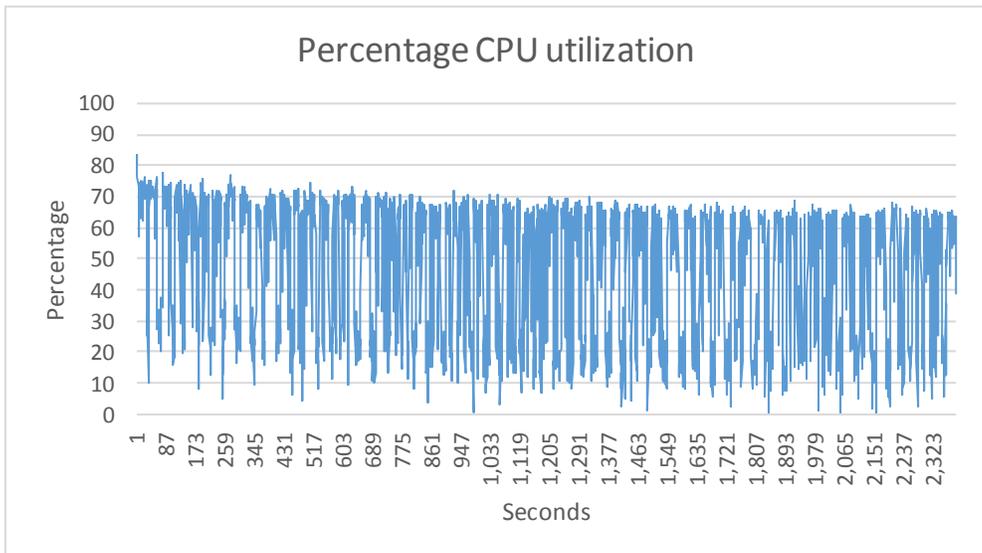


Figure 28: Memory utilization on the small instance before resizing.

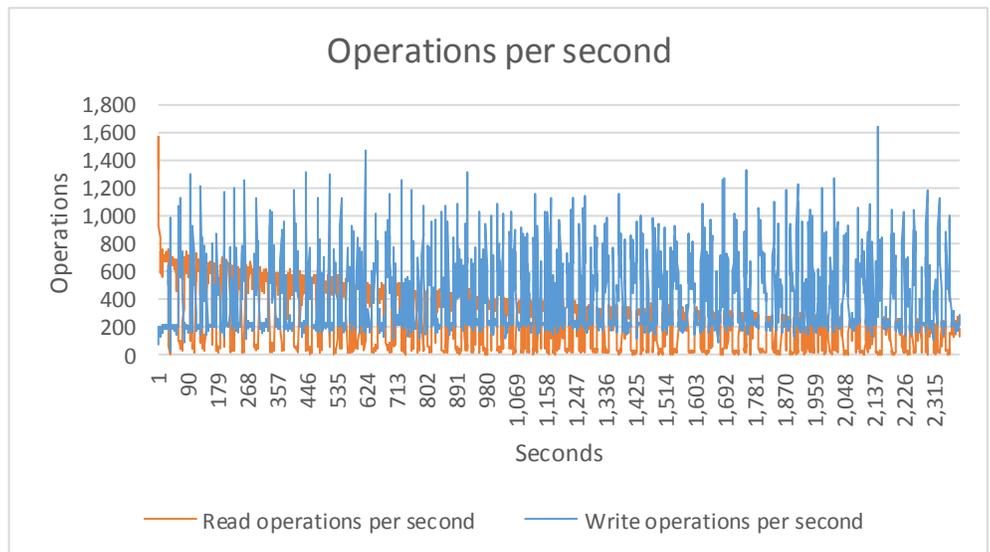
We resized the instance to a large flavor with four vCPUs, 8 GB RAM, and 80 GB of hard drive space, and the results were dramatic. CPU utilization hovered around the 70 percent range, while operations per second remained steady throughout the duration of our test. Memory utilization climbed to around 6 GB of the available 8 GB of space. Most importantly, the test did not terminate due to insufficient resources.

Figure 29 shows the CPU utilization of the newly resized server.



**Figure 29: Performance of the instance after resizing.**

Figure 30 shows the operations per second on the instance after the resize completed.



**Figure 30: Operations per second after resizing.**

Figure 31 below shows the memory utilization for the instance after the resize completed.

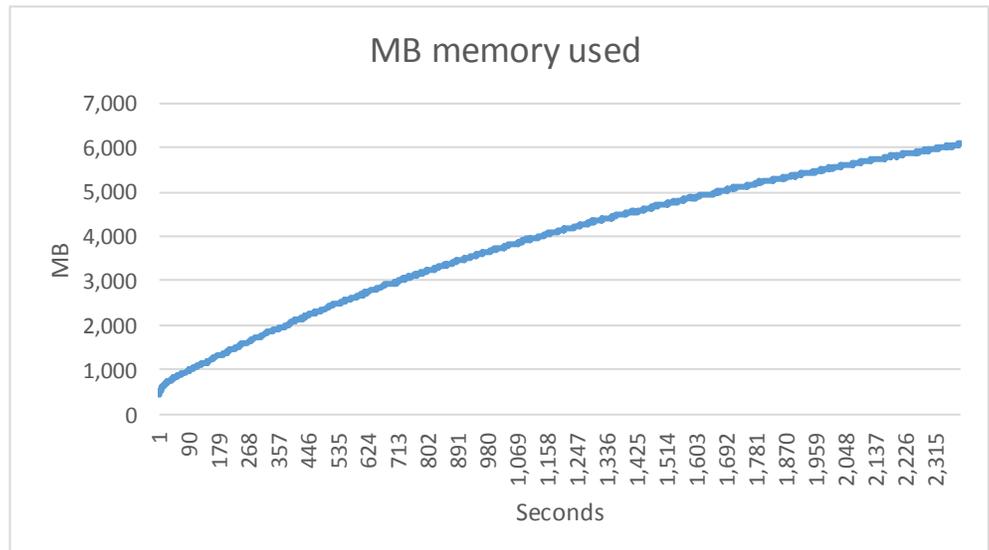


Figure 31: Operations per second after resizing.

The resize action allowed the instance to retain its previously written data, but provided additional resources to the virtual machine without the need to tear-down and provision a new instance. Furthermore, this entire operation occurred within the DX1000 MicroServer chassis itself, which placed no additional load on the core infrastructure.

See [Appendix D](#) for details on how we used DVD Store to place our database instance under load.

## CONCLUSION

The NEC DX1000 MicroServer Chassis is an ultra-dense multi-server platform that offers scale-up and scale-out capabilities, but with a power profile comparable to modern single 2U servers. With enterprise-class features such as redundant switching, high bandwidth, and low-latency connections for up to 46 multi-core server nodes – each with 32 GB of RAM and onboard SSD storage, the NEC DX1000 can be easily leveraged to meet your private cloud provisioning needs.

# APPENDIX A – DETAILED CONFIGURATION STEPS

## Configuring DX1000 Micro Modular server networking

1. Power on unit by connecting AC power cables to rear of unit.
2. Wait until Power switch and Power LED on front panel changes from amber to green.
3. Install NEC provide CP210x USB-UART drivers contained on DX1000 Utility Disk on your management server.
4. The drivers are located on the utility disk in the following folders:
  - Driver for Windows 7/8/8.1 64-bit Edition  
  \drivers\CP210x\_VCP\_Windows\CP210xVCPInstaller\_x64.exe
  - Driver for Windows 7/8/8.1 32-bit Edition  
  \drivers\CP210x\_VCP\_Windows\CP210xVCPInstaller\_x86.exe
5. Install correct driver for OS:
  - a. Double-click driver icon.
  - b. Select Yes.
  - c. Select Next.
  - d. Accept License agreement, then click Next.
  - e. Click Finish.
  - f. Reboot server.
6. Verify Driver CP210 installation:
  - a. Select Start menu.
  - b. Select Control Panel.
  - c. Select device manager.
  - d. Select Ports.
  - e. Verify driver is installed and not in an error state.
7. Download Teraterm-4.85.exe from [en.sourceforge.jp/projects/ttssh2/releases/](http://en.sourceforge.jp/projects/ttssh2/releases/).
8. Install Teraterm-4.85 by double clicking TeraTerm icon from your download location.
9. Select Run, click Yes, then click Next.
10. Accept License Agreement, then click Next.
11. Select Use default location, and click Next three times.
12. Click Install, then click Finish.
13. Connect to front panel Micro USB port 1 using NEC provide micro USB cable.
14. Start TeraTerm terminal emulation program using the following configuration:
  - a. To configure CMM 1, select Serial → Port, then select COM3: Silicon Labs Dual CP210xUSB to UART Bridge: Standard COM Port with the following:
    - 115200 bps
    - 8 data bits
    - 1 stop bit
    - No parity
  - b. To configure CMM 2, select Serial → Port, then select COM4: Silicon Labs Dual CP210xUSB to UART Bridge: Standard COM Port with the following:
    - 115200 bps
    - 8 data bits
    - 1 stop bit

- No parity
15. Log into ONS (utility program of internal switch) using the USB-to-UART bridge as admin. The default password is admin.
  16. At the command prompt, type `enable` and press enter.
  17. Type `configure` and press enter.
  18. Type `interface mgmt-ethernet` and press enter.
  19. To set the management IP address, type `ip address 10.130.1.1 255.255.0.0` and press enter.
  20. To set the default gateway, type `ip default-gateway 10.130.0.1` and press enter.
  21. Type `no shutdown` and press enter.
  22. Type `exit` and press enter.
  23. At the command prompt, type `exit` and press enter.
  24. Type `save config` and press enter to save the configuration.
  25. To reboot the switch, type `reload` and press enter.
  26. Remove the mini USB cable from the Micro USB port 1 and insert it in port 2.
  27. Repeat steps 3 through 17

## Installing MAAS Ubuntu 14.04 Server Edition as a Virtual Machine

We used VMware vSphere for the supporting virtual machines. You can install MAAS on physical or your choice of virtual machine.

1. Select File, then New, then Virtual Machine.
2. Select Typical, and click Next.
3. Enter Machine name, and click Next.
4. Select Datastore, and click Next.
5. Select Linux.
6. Select the proper version (Ubuntu Linux 64-bit) from pull-down menu, and click Next.
7. Use default network configurations, and click Next.
8. Select Virtual disk size of 50GB+ - Thin Provision, and click Next.
9. Check Edit the virtual machine settings before completion, and click Continue.
10. Edit virtual machine:
  - a. Set Memory to 8GB.
  - b. Set CPU to 2 virtual sockets.
  - c. Select New CD/DVD –Select Datastore, then select “from Ubuntu-14.04.1-server-amd64 installation iso.”
  - d. At power on, click Connect.
  - e. Click Finish.
11. Highlight the newly created Virtual Machine, right-click, and click Power on.
12. Highlight the newly created Virtual Machine, right-click, and click Open Console.
13. Select English.
14. Select Install Ubuntu Server.
15. Select English language.
16. Select United States.
17. For Detect keyboard layout, select No.
18. For Configure keyboard, select English.

19. For Keyboard layout, select English.
20. Select Configure the network.
21. Select Configure network manually (if DHCP is not available), then do the following:
  - a. Enter IP address.
  - b. Enter netmask.
  - c. Enter nameserver.
  - d. Enter hostname.
  - e. Enter domainname.
22. Enter Hostname.
23. Enter username.
24. Enter Username for account.
25. Enter and verify password.
26. Select no for encrypt home directory.
27. Select correct Time Zone.
28. Select Guided – use entire disk and set up LVM.
29. Select disk.
30. Select Yes Write changes to disk.
31. Accept defaults for disk size.
32. Select Yes Write changes to disk.
33. Select continue for proxy information.
34. Select No automatic updates.
35. Select OpenSSH server.
36. Select Yes to install boot master record.
37. Select Continue – installation complete.

## Configuring Ubuntu MAAS Server

1. Use PuTTY to log into the newly created virtual machine.
2. To upgrade the package repository, go to the command prompt and type each of the following commands (pressing enter after each command):
 

```
sudo add-apt-repository ppa:juju/stable
sudo add-apt-repository ppa:maas-maintainers/stable
sudo add-apt-repository ppa:cloud-installer/testing
sudo apt update -y
Install VM tools sudo apt install open-vm-tools -y
Install ipmitool sudo apt install ipmitools -y
```
3. To upgrade OS packages, go to the command prompt, type `sudo apt-get upgrade -y` and press enter.
4. To install MAAS, go to the command prompt, type `sudo apt install MAAS -y` and press enter.
5. To create the administrator account, go to the command prompt, type `sudo maas-region-admin createadmin --username=admin --email=root@localhost` and press enter.
6. When you are prompted to create and verify a user password, do so.
7. Log into MAAS GUI <http://<IP>/MAAS> using the credentials you just created.
8. To configure clusters, select Clusters, enter DNS zone name, and select Save cluster controller.
9. Configure interfaces:
  - a. Select edit icon for interface eth0.

- b. From the Management pull-down menu, select DHCP and DNS.
- c. Enter DHCP management information:
  - IP (required) MAAS server IP address
  - Subnet mask
  - Broadcast IP
  - Router IP
  - DHCP dynamic IP range low value
  - DHCP dynamic IP range high value
  - Static IP range low value
  - Static IP range high value
2. Select Save interface.
3. Select Images, check all release versions, and select amd64 architecture.
4. Select Apply changes and wait for image download to complete. Note: This process can take up to 45 minutes.

## Adding nodes to the system

1. If server modules do not automatically populate into MAAS, perform the following steps:
  - a. Add NEC-provided server module management utilities.
  - b. To make a new directory, type `mkdir nec_tools` and press enter.
  - c. Copy management files from NEC utility disk that are located in the NEC/util directory onto the MAAS server in newly created directory.
  - d. Run `mng_niclist` command to obtain address of management LAN:

```
./mng_niclist -I <controller IP> -C all
```

2. Using the MAAS server command line, power on each server module:
 

```
ipmitool -I lanplus -H <Management LAN IP> -U Administrator -P Administrator power on
```
3. Perform this command for each server module to ensure it powers on and PXE boots.
4. To retrieve the MAAS API-key, go to the command prompt and type
 

```
sudo maas-region-admin apikey --username admin
```
5. Log into MAAS server from CLI:
 

```
maas login <maas server> http://<IP>/MAAS/api/1.0 api-key
```
6. To configure MAAS to accept PXE requests, go to the command prompt and type
 

```
maas <maas-server name> nodes accept-all
```

## Configuring Juju Management Server

1. Select File → New → Virtual Machine.
2. Select Typical, and select Next.
3. Enter Machine name, and select Next.
4. Select Datastore, and select Next.
5. Select Linux.
6. Select the proper version (Ubuntu Linux 64-bit) from the pull-down menu, and select Next.
7. Use default network configurations, and select Next.

8. Select Virtual disk size of 50GB+ - Thin Provision, and select Next.
9. Check Edit the virtual machine settings before completion, and click Continue.
10. Edit Virtual machine.
11. Set Memory to 8GB.
12. Set CPU to 2 virtual sockets.
13. Select New CD/DVD –Select Datastore, then select Ubuntu-14.04.1-server-amd64 installation iso.
14. Select Connect at power on.
15. Select Finish.
16. Highlight newly created Virtual Machine → right-click → select Power on.
17. Highlight newly created Virtual Machine → right-click → select Open Console.
18. Select English.
19. Select Install Ubuntu Server.
20. Select English language.
21. Select United States.
22. For Detect keyboard layout, select No.
23. For Configure keyboard, select English.
24. For Keyboard layout, select English.
25. Enter New hostname.
26. Enter domainname.
27. Enter Hostname.
28. Enter username.
29. Enter Username for account.
30. Enter and verify password.
31. For Encrypt home directory, select no.
32. Select correct Time Zone.
33. Select Guided – use entire disk and set up LVM.
34. Select disk.
35. For Write changes to disk, select Yes.
36. Accept defaults for disk size.
37. Select Yes write changes to disk.
38. Select continue for proxy information.
39. Select No automatic updates.
40. Select OpenSSH server.
41. Select Yes to install boot master record.
42. Select Continue – installation complete.
43. Log into newly created server via SSH using PuTTY client.
44. To install Juju base and supporting packages, go to the command prompt and type each of the following commands (pressing enter after each command):

```
sudo add-apt-repository ppa:juju/stable
sudo apt update -y
sudo apt install open-vm-tools -y
sudo apt install juju -y
sudo apt install juju-local -y
sudo apt install juju-quickstart -y
sudo apt install juju-deployer -y
sudo apt install charm-tools -y
```

45. To install OpenStack client tools, go to the command prompt and type each of the following commands (pressing enter after each command):

```
sudo apt install python-keystone -y
sudo apt install python-neutronclient -y
sudo apt install python-novaclient -y
sudo apt install python-glanceclient -y
sudo apt install python-cinderclient -y
sudo apt install python-heatclient -y
sudo apt install python-ceilometerclient -y
```

46. Add SSH key into MAAS:

- a. Go to the command prompt on the MAAS server, and type `ssh-keygen` and press enter. Note: Use default values.
- b. From MAAS GUI, select the admin tab → Preferences → select + add SSH Key.
- c. Go to the command prompt on the command prompt on the Juju management server, and type `cat .ssh/id_rsa.pub` copy output of file
- d. Paste contents into Public Key box → select Add key.

## Creating Juju bootstrap environment

1. At command prompt, type `juju-quickstart -i --no-browser <enter>`
2. Select new MAAS (bare metal) environment and press enter.
3. Enter values for the following form fields:
  - a. Environment name: {Provide a name for your environment}
  - b. Server Address: {`http://MAAS_Server_IP_Address/MAAS`}
  - c. API key: {Enter the API key exactly as shown in the MAAS user preference page}
  - d. SSH public key file: `~/ssh/id_rsa.pub`
  - e. Admin secret: `admin`
4. Press the down key until you arrive at Save and press enter.
5. Press the right arrow to select Use and press enter.
6. Once the bootstrap process is complete, it will provide a Juju GUI URL and credentials. Record the URL and credentials.
7. At the command prompt, type `juju status` to verify that the system has no errors.
8. At command prompt, type `eval `ssh-agent`` then press enter.
9. Open a Web browser and connect to Juju GUI with the URL and credentials provided in step 6.

## Deploying OpenStack using Juju GUI

1. Copy and paste the text in [Appendix B](#) to a text file on your local computer and save it as `openstack.yaml`.
2. In the Juju GUI, click the Import button on the lower right of the screen.
3. Select the `openstack.yaml` file you saved from step 1 then select Open.
4. The Juju GUI will begin to load the OpenStack components. This will automatically install, configure and create relationships for the OpenStack deployment.

## Configuring OpenStack and deploy

1. Connect to the Juju server console.
2. Type `vi OpenStack_config.sh` to create a new file.

3. Press `i` on the keyboard to enter insert mode.
4. Copy and paste the contents of [Appendix C](#) into the new file.
5. Press `Esc` on the keyboard to exit insert mode.
6. Type `:wq` to save and close the file.
7. Type `chmod 755 OpenStack_config.sh` to change the permissions on the script.
8. Type `./OpenStack_config.sh` to execute the script and complete the OpenStack configuration.
9. Once the script completes, log into your new OpenStack with `admin/admin` and begin deploying instances.

## Scaling OpenStack components using Juju GUI

1. Log into Juju GUI with the administrative credentials.
2. In the left panel, select Added Services.
3. Scroll down and select the component you want to scale. For example, select nova-compute to add compute nodes.
4. In the left panel, click the `+` icon to right of Number of Units.
5. Enter the number of units needed for expansion of service. Select 1 unit per machine and click Confirm.
6. Click Commit in the lower right hand of the screen to add the nodes. The status bar on each new node will turn green when the deployment is complete.

## Launching an OpenStack instance

1. On your workstation, download and install PuTTY from the following link: <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>. Download "A Windows installer for everything except PuTTYtel". Install PuTTY.
2. Log into OpenStack dashboard.
3. Select Project → Access & Security → Key Pairs → Create Key Pair:
4. The newly created Key Pair will automatically download to your default location.
5. From the Start menu, click All Programs → PuTTY → PuTTYgen
6. Select Load.
7. Select browse to the location of the private key file (*test.pem*).
8. Select your key file and select Open. Accept all defaults.
9. Select Save private key to save the key in PuTTY's format.key.
10. Log into the OpenStack dashboard.
11. Select Project → Compute → Instances → Launch Instance
12. Select the Availability Zone, enter the Instance Name, select the Flavor (size), enter the Instance Count, and select Boot Source and Image Name.
13. Select the Access & Security tab.
14. Select your Key Pair.
15. Select the Networking tab.
16. Click Private Network to add it to the instance.
17. Click Launch.
18. Select the More pull-down menu for the newly created instance.
19. Select Associate Floating IP address.
20. Click the `+`. Accept the defaults and select Allocate IP.
21. Click Associate.

## Accessing an OpenStack instance

1. From the Start menu on your workstation, click All Programs → PuTTY → PuTTY.
2. From the Category section, select Connection → SSH → Auth.
3. Select browse and select the converted ssh key from step 9 of the previous section.
4. From the Category section, select Session.
5. Enter the Floating IP address of the virtual machine instance then select Open.
6. Select Yes to acknowledge the PuTTY Security Alert.
7. At the login prompt, log in as user Ubuntu.

## Resizing an OpenStack instance

1. Log into the OpenStack dashboard.
2. Select Project → Compute → Instances.
3. Click the More pull-down menu for the Instance you are going to resize and select Resize Instance.
4. Select the Flavor you intend to use to grow the Instance.
5. Click Resize.
6. When prompted, select Confirm Resize/ Migration.

## APPENDIX B – YAML FILE FOR OPENSTACK ON THE NEC DX1000

Copy the text in the following section and paste it into a file on your workstation. Save the file as `openstack.yaml` and import it into Juju. You can customize it for your environment by changing the value of `num_units` under each service.

```
envExport:
  services:
    mysql:
      charm: "cs:trusty/mysql-16"
      num_units: 1
      options:
        "dataset-size": "20%"
      annotations:
        "gui-x": "5637.075439453125"
        "gui-y": "-1042.1895446777344"
    "ceilometer-agent":
      charm: "cs:trusty/ceilometer-agent-3"
      num_units: 0
      annotations:
        "gui-x": "2767.07568359375"
        "gui-y": "1511.8104553222656"
    "nova-cloud-controller":
      charm: "cs:trusty/nova-cloud-controller-51"
      num_units: 1
      options:
        "console-access-protocol": novnc
        "network-manager": Neutron
      expose: true
      annotations:
        "gui-x": "2348.075439453125"
        "gui-y": "445.5604553222656"
    ceph:
      charm: "cs:trusty/ceph-34"
      num_units: 3
      options:
        fsid: "9e7aac42-4bf4-11e3-b4b7-5254006a039c"
        "monitor-secret": AQAavoJSOAv/NRAAgvXP8d7iXN7lWYbvDZzm2Q==
        "osd-devices": /srv
        "osd-reformat": "yes"
      annotations:
        "gui-x": "3414.07568359375"
        "gui-y": "1598.0604553222656"
    "ceph-radosgw":
      charm: "cs:trusty/ceph-radosgw-9"
      num_units: 1
      annotations:
        "gui-x": "3820.075439453125"
        "gui-y": "1593.5604553222656"
    cinder:
      charm: "cs:trusty/cinder-12"
      num_units: 1
```

```
options:
  "block-device": none
annotations:
  "gui-x": "2273.075439453125"
  "gui-y": "1122.5604553222656"
"cinder-ceph":
  charm: "cs:trusty/cinder-ceph-5"
  num_units: 0
  annotations:
    "gui-x": "2359.07568359375"
    "gui-y": "1632.5604553222656"
glance:
  charm: "cs:trusty/glance-11"
  num_units: 1
  annotations:
    "gui-x": "4283.0751953125"
    "gui-y": "1139.8104553222656"
keystone:
  charm: "cs:trusty/keystone-13"
  num_units: 1
  options:
    "admin-password": admin
    "admin-token": admin
  annotations:
    "gui-x": "1547.0753173828125"
    "gui-y": "-247.43954467773438"
"openstack-dashboard":
  charm: "cs:trusty/openstack-dashboard-9"
  num_units: 1
  options:
    webroot: /
  expose: true
  annotations:
    "gui-x": "3663.075439453125"
    "gui-y": "-964.7596740722656"
"nova-compute":
  charm: "cs:trusty/nova-compute-14"
  num_units: 3
  options:
    "enable-live-migration": true
    "enable-resize": true
    "flat-interface": eth0
    "migration-auth-type": ssh
  annotations:
    "gui-x": "2667.1341552734375"
    "gui-y": "-820.4395446777344"
ntp:
  charm: "cs:trusty/ntp-6"
  num_units: 0
  annotations:
    "gui-x": "5312.0751953125"
    "gui-y": "476.8103942871094"
ntpmaster:
  charm: "cs:trusty/ntpmaster-2"
```

```

num_units: 1
annotations:
  "gui-x": "5645.0751953125"
  "gui-y": "-454.1896057128906"
"quantum-gateway":
charm: "cs:trusty/quantum-gateway-10"
num_units: 1
options:
  "ext-port": eth1
  "instance-mtu": 1400
annotations:
  "gui-x": "4161.0755615234375"
  "gui-y": "-537.4395446777344"
"rabbitmq-server":
charm: "cs:trusty/rabbitmq-server-26"
num_units: 1
annotations:
  "gui-x": "6190.075439453125"
  "gui-y": "-185.18954467773438"
ceilometer:
charm: "cs:trusty/ceilometer-3"
num_units: 1
annotations:
  "gui-x": "4140.0751953125"
  "gui-y": "1416.8104553222656"
to:
  - ntpmaster=0
heat:
charm: "cs:trusty/heat-5"
num_units: 1
annotations:
  "gui-x": "1861.075439453125"
  "gui-y": "777.8104553222656"
to:
  - glance=0
mongodb:
charm: "cs:trusty/mongodb-16"
num_units: 1
annotations:
  "gui-x": "5296.0751953125"
  "gui-y": "1380.8104553222656"
to:
  - ntpmaster=0
relations:
  - - "ntp:juju-info"
  - - "heat:juju-info"
  - - "ntp:juju-info"
  - - "keystone:juju-info"
  - - "ntp:juju-info"
  - - "ceph-radosgw:juju-info"
  - - "nova-compute:shared-db"
  - - "mysql:shared-db"
  - - "ntp:juju-info"
  - - "ceph:juju-info"

```

```

- - "ceph-radosgw:identity-service"
- - "keystone:identity-service"
- - "ntp:juju-info"
- - "quantum-gateway:juju-info"
- - "glance:identity-service"
- - "keystone:identity-service"
- - "heat:amqp"
- - "rabbitmq-server:amqp"
- - "ceilometer-agent:ceilometer-service"
- - "ceilometer:ceilometer-service"
- - "ntp:juju-info"
- - "nova-cloud-controller:juju-info"
- - "ceilometer:identity-service"
- - "keystone:identity-service"
- - "ceph:client"
- - "cinder-ceph:ceph"
- - "heat:identity-service"
- - "keystone:identity-service"
- - "ntp:juju-info"
- - "nova-compute:juju-info"
- - "heat:shared-db"
- - "mysql:shared-db"
- - "ceilometer-agent:container"
- - "nova-compute:juju-info"
- - "ceilometer:shared-db"
- - "mongodb:database"
- - "ntp:juju-info"
- - "cinder:juju-info"
- - "ntp:juju-info"
- - "mongodb:juju-info"
- - "ntp:juju-info"
- - "glance:juju-info"
- - "ntp:juju-info"
- - "rabbitmq-server:juju-info"
- - "openstack-dashboard:identity-service"
- - "keystone:identity-service"
- - "quantum-gateway:quantum-network-service"
- - "nova-cloud-controller:quantum-network-service"
- - "ntp:juju-info"
- - "ceilometer:juju-info"
- - "quantum-gateway:amqp"
- - "rabbitmq-server:amqp"
- - "quantum-gateway:shared-db"
- - "mysql:shared-db"
- - "ceph-radosgw:mon"
- - "ceph:radosgw"
- - "cinder:identity-service"
- - "keystone:identity-service"
- - "ceilometer:amqp"
- - "rabbitmq-server:amqp"
- - "cinder:cinder-volume-service"
- - "nova-cloud-controller:cinder-volume-service"
- - "ceph:client"
- - "glance:ceph"

```

```
- - "ceph:client"
- - "nova-compute:ceph"
- - "cinder:storage-backend"
- - "cinder-ceph:storage-backend"
- - "nova-compute:cloud-compute"
- - "nova-cloud-controller:cloud-compute"
- - "nova-compute:image-service"
- - "glance:image-service"
- - "keystone:shared-db"
- - "mysql:shared-db"
- - "ntp:juju-info"
- - "openstack-dashboard:juju-info"
- - "cinder:amqp"
- - "rabbitmq-server:amqp"
- - "ntp:master"
- - "ntpmaster:master"
- - "cinder:shared-db"
- - "mysql:shared-db"
- - "ntpmaster:juju-info"
- - "ceilometer-agent:container"
- - "glance:shared-db"
- - "mysql:shared-db"
- - "nova-cloud-controller:identity-service"
- - "keystone:identity-service"
- - "nova-compute:amqp"
- - "rabbitmq-server:amqp"
- - "nova-cloud-controller:image-service"
- - "glance:image-service"
- - "nova-cloud-controller:shared-db"
- - "mysql:shared-db"
- - "nova-cloud-controller:amqp"
- - "rabbitmq-server:amqp"
series: trusty
```

## APPENDIX C – OPENSTACK\_CONFIG.SH FILE

Copy the text below into the OpenStack\_config.ssh file on the Juju server console session. You can customize this script by replacing the IP addresses in the NEUTRON sections with addresses appropriate for your environment.

```
set -ex

echo "This command is run to configure OpenStack networking, security, quotas and add basic image"

PKGS=" python-keystone python-neutronclient python-novaclient python-glanceclient"
dpkg -l $PKGS > /dev/null || sudo apt-get install -y $PKGS

NEUTRON_EXT_NET_GW="10.130.0.1"
NEUTRON_EXT_NET_CIDR="10.130.0.0/16"
NEUTRON_EXT_NET_NAME="ext_net"
NEUTRON_DNS="10.130.1.5"
NEUTRON_FLOAT_RANGE_START="10.130.1.211"
NEUTRON_FLOAT_RANGE_END="10.130.1.254"

NEUTRON_FIXED_NET_CIDR="192.168.14.0/24"
NEUTRON_FIXED_NET_NAME="admin_net"

keystone=$(juju status keystone | grep public-address | head -1 | awk '{print $2}')

echo "export SERVICE_ENDPOINT=http://$keystone:35357/v2.0/"
export SERVICE_TOKEN=admin
export OS_AUTH_URL=http://$keystone:35357/v2.0/
export OS_USERNAME=admin
export OS_PASSWORD=admin
export OS_TENANT_NAME=admin
export OS_REGION_NAME=RegionOne
" > ~/nova.rc

. ~/nova.rc

# Determine the tenant id for the configured tenant name.
export TENANT_ID="$(keystone tenant-list | grep $OS_TENANT_NAME | awk '{ print $2 }')"
#create ext network with neutron for floating IPs
EXTERNAL_NETWORK_ID=$(neutron net-create ext_net --tenant-id $TENANT_ID -- --
router:external=True | grep " id" | awk '{print $4}')
neutron subnet-create ext_net $NEUTRON_EXT_NET_CIDR --name ext_net_subnet --tenant-id
$TENANT_ID \
--allocation-pool start=$NEUTRON_FLOAT_RANGE_START,end=$NEUTRON_FLOAT_RANGE_END \
--gateway $NEUTRON_EXT_NET_GW --disable-dhcp --dns_nameservers $NEUTRON_DNS list=true
```

```

#Create private network for neutron for tenant VMs
neutron net-create private
SUBNET_ID=$(neutron subnet-create private $NEUTRON_FIXED_NET_CIDR -- --name private_subnet -
-dns_nameservers list=true $NEUTRON_DNS | grep " id" | awk '{print $4}')

#Create router for external network and private network
ROUTER_ID=$(neutron router-create --tenant-id $TENANT_ID provider-router | grep " id" | awk
'{print $4}')
neutron router-interface-add $ROUTER_ID $SUBNET_ID
neutron router-gateway-set $ROUTER_ID $EXTERNAL_NETWORK_ID

#Configure the default security group to allow ICMP and SSH
nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
#for rdp
nova secgroup-add-rule default tcp 3389 3389 0.0.0.0/0

#Upload a default SSH key
nova keypair-add --pub-key ~/.ssh/id_rsa.pub default

#Remove the m1.tiny as it is too small for Ubuntu.
nova flavor-delete m1.tiny

#Modify quotas for the tenant to allow large deployments
nova quota-update --instances 100 $TENANT_ID
nova quota-update --cores 200 $TENANT_ID
nova quota-update --ram 204800 $TENANT_ID
nova quota-update --security-groups 200 $TENANT_ID

mkdir -p ~/images
wget -O ~/images/trusty-server-cloudimg-amd64-disk1.img \
    http://cloud-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-disk1.img
glance image-create --name="Trusty" --is-public=true --progress \
    --container-format=bare --disk-format=qcow2 \
    < ~/images/trusty-server-cloudimg-amd64-disk1.img

exit

```

## APPENDIX D – LOAD TESTING WITH DVD STORE

The information below describes how we used DVD Store to generate load on a virtual machine to demonstrate the effects of resizing an active instance.

### DVD Store

We created and ran an online transaction processing (OLTP) database workload with the benchmark DVD Store 2.1, which delivers results in terms of the number of orders per minute that a solution can complete. DS2 models an online DVD store, where customers log in, search for movies, and make purchases. DS2 reports these actions in orders per minute that the system could handle, to show what kind of performance you could expect for your customers. The DS2 workload also performs other actions, such as adding new customers, to exercise the wide range of database functions you would need to run your ecommerce environment.

Learn more at [www.delltechcenter.com/page/DVD+Store](http://www.delltechcenter.com/page/DVD+Store).

### Configuring the database workload client

For our testing, we used a virtual client. To create this client, we installed Windows Server 2008 R2, assigned a static IP address, and installed .NET 3.5.

### Configuring the database

We generated the data using the Install.pl script included with DVD Store version 2.1 (DS2), providing the parameters for our 10 GB database size and the database platform on which we ran. We ran the Install.pl script on a utility system running Linux. The database schema was also generated by the Install.pl script.

## ABOUT PRINCIPLED TECHNOLOGIES



Principled Technologies, Inc.  
1007 Slater Road, Suite 300  
Durham, NC, 27703  
[www.principledtechnologies.com](http://www.principledtechnologies.com)

We provide industry-leading technology assessment and fact-based marketing services. We bring to every assignment extensive experience with and expertise in all aspects of technology testing and analysis, from researching new technologies, to developing new methodologies, to testing with existing and new tools.

When the assessment is complete, we know how to present the results to a broad range of target audiences. We provide our clients with the materials they need, from market-focused data to use in their own collateral to custom sales aids, such as test reports, performance assessments, and white papers. Every document reflects the results of our trusted independent analysis.

We provide customized services that focus on our clients' individual requirements. Whether the technology involves hardware, software, Web sites, or services, we offer the experience, expertise, and tools to help our clients assess how it will fare against its competition, its performance, its market readiness, and its quality and reliability.

Our founders, Mark L. Van Name and Bill Catchings, have worked together in technology assessment for over 20 years. As journalists, they published over a thousand articles on a wide array of technology subjects. They created and led the Ziff-Davis Benchmark Operation, which developed such industry-standard benchmarks as Ziff Davis Media's Winstone and WebBench. They founded and led eTesting Labs, and after the acquisition of that company by Lionbridge Technologies were the head and CTO of VeriTest.

---

Principled Technologies is a registered trademark of Principled Technologies, Inc.  
All other product names are the trademarks of their respective owners.

---

**Disclaimer of Warranties; Limitation of Liability:**

PRINCIPLED TECHNOLOGIES, INC. HAS MADE REASONABLE EFFORTS TO ENSURE THE ACCURACY AND VALIDITY OF ITS TESTING, HOWEVER, PRINCIPLED TECHNOLOGIES, INC. SPECIFICALLY DISCLAIMS ANY WARRANTY, EXPRESSED OR IMPLIED, RELATING TO THE TEST RESULTS AND ANALYSIS, THEIR ACCURACY, COMPLETENESS OR QUALITY, INCLUDING ANY IMPLIED WARRANTY OF FITNESS FOR ANY PARTICULAR PURPOSE. ALL PERSONS OR ENTITIES RELYING ON THE RESULTS OF ANY TESTING DO SO AT THEIR OWN RISK, AND AGREE THAT PRINCIPLED TECHNOLOGIES, INC., ITS EMPLOYEES AND ITS SUBCONTRACTORS SHALL HAVE NO LIABILITY WHATSOEVER FROM ANY CLAIM OF LOSS OR DAMAGE ON ACCOUNT OF ANY ALLEGED ERROR OR DEFECT IN ANY TESTING PROCEDURE OR RESULT.

IN NO EVENT SHALL PRINCIPLED TECHNOLOGIES, INC. BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH ITS TESTING, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL PRINCIPLED TECHNOLOGIES, INC.'S LIABILITY, INCLUDING FOR DIRECT DAMAGES, EXCEED THE AMOUNTS PAID IN CONNECTION WITH PRINCIPLED TECHNOLOGIES, INC.'S TESTING. CUSTOMER'S SOLE AND EXCLUSIVE REMEDIES ARE AS SET FORTH HEREIN.

---