JOID User Guide

Release brahmaputra.1.0 (3335efa)

OPNFV

March 16, 2016

CONTENTS

CHAPTER

ONE

INTRODUCTION

This document will explain how to install OPNFV Brahmaputra with JOID including installing JOID, configuring JOID for your environment, and deploying OPNFV with different SDN solutions in HA, or non-HA mode. Prerequisites include

- An Ubuntu 14.04 LTS Server Jumphost
- Minimum 2 Networks per Pharos requirement
  - One for the administrative network with gateway to access the Internet
  - One for the OpenStack public network to access OpenStack instances via floating IPs
  - JOID supports multiple isolated networks for data as well as storage based on your network requirement for OpenStack.
- Minimum 6 Physical servers for bare metal environment
  - Jump Host x 1, minimum H/W configuration:
    - CPU cores: 16
    - Memory: 32GB
    - Hard Disk: 1 (250GB)
    - NIC: eth0 (Admin, Management), eth1 (external network)
  - Control Node x 3, minimum H/W configuration:
    - CPU cores: 16
    - Memory: 32GB
    - Hard Disk: 1 (500GB)
    - NIC: eth0 (Admin, Management), eth1 (external network)
  - Compute Node x 2, minimum H/W configuration:
    - CPU cores: 16
    - Memory: 32GB
    - Hard Disk: 1 (1TB), this includes the space for Ceph.
    - NIC: eth0 (Admin, Management), eth1 (external network)

OTE: Above configuration is minimum. For better performance and usage of the OpenStack, please consider higher specs for all nodes.

Make sure all servers are connected to top of rack switch and configured accordingly. No DHCP server should be up and configured. Configure gateways only on eth0 and eth1 networks to access the network outside your lab.

CHAPTER

TWO

ORIENTATION

2.1 JOID in brief

JOID as Juju OPNFV Infrastructure Deployer allows you to deploy different combinations of OpenStack release and SDN solution in HA or non-HA mode. For OpenStack, JOID supports Juno and Liberty. For SDN, it supports Openvswitch, OpenContrail, OpenDayLight, and ONOS. In addition to HA or non-HA mode, it also supports deploying from the latest development tree.

JOID heavily utilizes the technology developed in Juju and MAAS. Juju is a state-of-the-art, open source, universal model for service oriented architecture and service oriented deployments. Juju allows you to deploy, configure, manage, maintain, and scale cloud services quickly and efficiently on public clouds, as well as on physical servers, OpenStack, and containers. You can use Juju from the command line or through its powerful GUI. MAAS (Metal-As-A-Service) brings the dynamism of cloud computing to the world of physical provisioning and Ubuntu. Connect, commission and deploy physical servers in record time, re-allocate nodes between services dynamically, and keep them up to date; and in due course, retire them from use. In conjunction with the Juju service orchestration software, MAAS will enable you to get the most out of your physical hardware and dynamically deploy complex services with ease and confidence.

For more info on Juju and MAAS, please visit https://jujucharms.com/ and http://maas.ubuntu.com.

2.2 Typical JOID Setup

The MAAS server is installed and configured in a VM on the Ubuntu 14.04 LTS Jump Host with access to the Internet. Another VM is created to be managed by MAAS as a bootstrap node for Juju. The rest of the resources, bare metal or virtual, will be registered and provisioned in MAAS. And finally the MAAS environment details are passed to Juju for use.

CHAPTER

THREE

INSTALLATION

We will use MAAS-deployer to automate the deployment of MAAS clusters for use as a Juju provider. MAAS-deployer uses a set of configuration files and simple commands to build a MAAS cluster using virtual machines for the region controller and bootstrap hosts and automatically commission nodes as required so that the only remaining step is to deploy services with Juju. For more information about the maas-deployer, please see https://launchpad.net/maas-deployer.

3.1 Configuring the Jump Host

Let's get started on the Jump Host node.

The MAAS server is going to be installed and configured in a virtual machine. We need to create bridges on the Jump Host prior to setting up the MAAS-deployer.

OTE: For all the commands in this document, please do not use a 'root' user account to run. Please create a non root user account. We recommend using the 'ubuntu' user.

Install the bridge-utils package on the Jump Host and configure a minimum of two bridges, one for the Admin network, the other for the Public network:

```
$ sudo apt-get install bridge-utils

$ cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

iface p1p1 inet manual

auto brAdm
iface brAdm inet static
    address 172.16.50.51
    netmask 255.255.255.0
    bridge_ports p1p1

iface p1p2 inet manual

auto brPublic
iface brPublic inet static
    address 10.10.15.1
    netmask 255.255.240.0
```

```
    gateway 10.10.10.1
    dns-nameservers 8.8.8.8
    bridge_ports p1p2
```

NOTE: If you choose to use separate networks for management, data, and storage, then you need to create a bridge for each interface. In case of VLAN tags, make the appropriate network on jump-host depend upon VLAN ID on the interface.

NOTE: The Ethernet device names can vary from one installation to another. Please change the Ethernet device names according to your environment.

MAAS-deployer has been integrated in the JOID project. To get the JOID code, please run

```
$ sudo apt-get install git
$ git clone https://gerrit.opnfv.org/gerrit/p/joid.git
```

3.2 Setting Up Your Environment for JOID

To set up your own environment, create a directory in joid/ci/maas/<company name>/<pod number>/ and copy an existing JOID environment over. For example:

```
$ cd joid/ci
$ mkdir -p maas/myown/pod
$ cp maas/juniper/pod1/deployment.yaml maas/myown/pod/
```

Now let's configure MAAS-deployer by editing the deployment.yaml file. Let's review each section. We will use the Juniper pod deployment.yaml as an example.

```
# This file defines the deployment for the MAAS environment which is to be
# deployed and automated.
demo-maas:
    maas:
        # Defines the general setup for the MAAS environment, including the
        # username and password for the host as well as the MAAS server.
        user: ubuntu
        password: ubuntu
```

'demo-maas' is the environment name we set, it will be used by Juju. The username and password will be the login credentials for the MAAS server VM and also for the MAAS server web UI.

```
# Contains the virtual machine parameters for creating the MAAS virtual
# server. Here you can configure the name of the virsh domain, the
# parameters for how the network is attached.
name: opnfv-maas-juniper
interfaces: ['bridge=brAdm,model=virtio', 'bridge=brPublic,model=virtio']
memory: 4096
vcpus: 1
arch: amd64
pool: default
disk_size: 160G
```

When it's configured, you will see a KVM VM created and named 'opnfv-maas-juniper' on the Jump Host with 2 network interfaces configured and connected to brAdm and brPublic on the host. You may want to increase the vcpu number and disk size for the VM depending on the resources.

```
# Apt http proxy setting(s)
apt_http_proxy:
```

```
apt_sources:
  - ppa:maas/stable
  - ppa:juju/stable
```

If in your environment uses an http proxy, please enter its information here. In addition, add the MAAS and Juju PPA locations here.

```
# Virsh power settings
# Specifies the uri and keys to use for virsh power control of the
# juju virtual machine. If the uri is omitted, the value for the
# --remote is used. If no power settings are desired, then do not
# supply the virsh block.
virsh:
    rsa_priv_key: /home/ubuntu/.ssh/id_rsa
    rsa_pub_key: /home/ubuntu/.ssh/id_rsa.pub
    uri: qemu+ssh://ubuntu@172.16.50.51/system

# Defines the IP Address that the configuration script will use
# to access the MAAS controller via SSH.
ip_address: 172.16.50.50
```

This section defines MAAS server IP (172.16.50.50) and the virsh power settings. The Juju bootstrap VM is defined later.

```
# This section allows the user to set a series of options on the
# MAAS server itself. The list of config options can be found in
# the upstream MAAS documentation:
#  - http://maas.ubuntu.com/docs/api.html#maas-server
settings:
    main_archive: http://us.archive.ubuntu.com/ubuntu
    upstream_dns: 8.8.8.8
    maas_name: juniperpod1
    # kernel_opts: "console=tty0 console=ttyS1,115200n8"
    # ntp_server: ntp.ubuntu.com
```

Here we specify some settings for the MAAS server itself. Once MAAS is deployed, you will find these settings on http://172.16.50.50/MAAS/settings/.

```
# This section is used to define the networking parameters for when
# the node first comes up. It is fed into the meta-data cloud-init
# configuration and is used to configure the networking piece of the
# service. The contents of this section are written directly to the
# /etc/network/interfaces file.
#
# Please note, this is slightly different than the
# node-group-interfaces section below. This will configure the
# machine's networking params, and the node-group-interfaces will
# configure the maas node-group interfaces which is used for
# controlling the dhcp, dns, etc.
network_config: |
    auto lo
    iface lo inet loopback

    auto eth0
    iface eth0 inet static
    address 172.16.50.50
    netmask 255.255.255.0
    network 172.16.50.0
```

```
    broadcast 172.16.50.255
    dns-nameservers 8.8.8.8 127.0.0.1

    auto eth1
    iface eth1 inet static
    address 10.10.15.50
    netmask 255.255.240.0
    network 10.10.0.0
    broadcast 10.10.15.255
    gateway 10.10.10.1
```

This section defines the MAAS server's network interfaces. Once MAAS is deployed, you will find this setting at /etc/network/interfaces in the MAAS VM.

```
# The node-group-interfaces section is used to configure the MAAS
# network interfaces. Basic configuration is supported, such as which
# device should be bound, the range of IP addresses, etc.
# Note: this may contain the special identifiers:
#   ${maas_net} - the first 3 octets of the ipv4 address
#   ${maas_ip} - the ip address of the MAAS controller
node_group_ifaces:
    - device: eth0
      ip: 172.16.50.50
      subnet_mask: 255.255.255.0
      broadcast_ip: 172.16.50.255
      router_ip: 172.16.50.50
      static_range:
          low: 172.16.50.60
          high: 172.16.50.90
      dynamic_range:
          low: 172.16.50.91
          high: 172.16.50.254
```

This section configures the MAAS cluster controller. Here it configures the MAAS cluster to provide DHCP and DNS services on the eth0 interface with dynamic and static IP ranges defined. You should allocate enough IP addresses for bare metal hosts in the static IP range, and allocate as many as possible in the dynamic IP range.

```
# Defines the physical nodes which are added to the MAAS cluste
# controller upon startup of the node.
nodes:
    - name: 2-R4N4B2-control
      tags: control
      architecture: amd64/generic
      mac_addresses:
          - "0c:c4:7a:16:2a:70"
      power:
          type: ipmi
          address: 10.10.7.92
          user: ADMIN
          pass: ADMIN
          driver: LAN_2_0

    - name: 3-R4N3B1-compute
      tags: compute
      architecture: amd64/generic
      mac_addresses:
          - "0c:c4:7a:53:57:c2"
      power:
          type: ipmi
```

```
          address: 10.10.7.84
          user: ADMIN
          pass: ADMIN
          driver: LAN_2_0
 <snip>
```

This section defines the physical nodes to be added to the MAAS cluster controller. For example, the first node here is named '2-R4N4B2-control', with a tag 'control' and architecture specified as amd64/generic. You will need to know the MAC address of the network interface of the node where it can reach MAAS server; it's the network interface of the node to PXE boot on. You need to tell MAAS how to power control the node by providing the the BMC IP address and BMC admin credentials. MAAS power control not only supports IPMI v2.0, but also supports virsh, Cisco UCS manager, HP moonshot iLO, and Microsoft OCS, among others. Tag is used here with Juju constraints to make sure that a particular service gets deployed only on hardware with the tag you created. Later when we go through the Juju deploy bundle, you will see the constraints setting.

```
# Contains the virtual machine parameters for creating the Juju bootstrap
# node virtual machine
juju-bootstrap:
    name: bootstrap
    interfaces: ['bridge=brAdm,model=virtio', 'bridge=brPublic,model=virtio']
    memory: 4096
    vcpus: 2
    arch: amd64
    pool: default
    disk_size: 120G
```

The last section of the example deployment.yaml file defines the Juju bootstrap VM node. When it's configured, you will see a KVM VM created and named 'juju-boostrap' on the Jump Host with 2 network interfaces configured and connected to brAdm and brPublic on the host. You may want to increase the vcpu number and disk size for the VM depending on the resources.

We are now done providing all the information regarding the MAAS VM and Juju VM, and how nodes and how many of them will be registered in MAAS. This information is very important, if you have questions, please hop on to #opnfv-joid IRC channel on freenode to ask.

Next we will use the 02-maasdeploy.sh in joid/ci to kick off maas-deployer. Before we do that, we will create an entry to tell maas-deployer what deployment.yaml file to use. Use your favorite editor to add an entry under the section case $1. In our example, this is what we add:

```
'juniperpod1' )
    cp maas/juniper/pod1/deployment.yaml ./deployment.yaml
    ;;
```

NOTE: If your username is different from 'ubuntu', please change the ssh key section accordingly:

```
#just make sure the ssh keys are added into maas for the current user
sed --i "s@/home/ubuntu@$HOME@g" ./deployment.yaml
sed --i "s@qemu+ssh://ubuntu@qemu+ssh://$USER@g" ./deployment.yaml
```

3.3 Starting MAAS-deployer

Now run the 02-maasdeploy.sh script with the environment you just created

```
~/joid/ci$ ./02-maasdeploy.sh juniperpod1
```

This will take approximately 40 minutes to couple of hours depending on your environment. This script will do the following: 1. Create 2 VMs (KVM). 2. Install MAAS in one of the VMs. 3. Configure MAAS to enlist and commission a VM for Juju bootstrap node. 4. Configure MAAS to enlist and commission bare metal servers.

When it's done, you should be able to view the MAAS webpage (in our example http://172.16.50.50/MAAS) and see 1 bootstrap node and bare metal servers in the 'Ready' state on the nodes page.

Here is an example output of running 02-maasdeploy.sh: http://pastebin.ubuntu.com/15117137/

3.4 Troubleshooting MAAS deployer

During the installation process, please carefully review the error messages.

Join IRC channel #opnfv-joid on freenode to ask question. After the issues are resolved, re-running 02-maasdeploy.sh will clean up the VMs created previously. There is no need to manually undo what's been done.

3.5 Deploying OPNFV

JOID allows you to deploy different combinations of OpenStack release and SDN solution in HA or non-HA mode. For OpenStack, it supports Juno and Liberty. For SDN, it supports Open vSwitch, OpenContrail, OpenDaylight and ONOS (Open Network Operating System). In addition to HA or non-HA mode, it also supports deploying the latest from the development tree (tip).

The deploy.sh script in the joid/ci directoy will do all the work for you. For example, the following deploys OpenStack Liberty with OpenDaylight in a HA mode in the Intelpod7.

```
~/joid/ci$  ./deploy.sh -o liberty -s odl -t ha -l intelpod7 -f none
```

NOTE: You will need to modify ~/joid/ci/01-deploybundle.sh to deploy to your own environment, explained later.

Take a look at the deploy.sh script. You will find we support the following for each option:

```
[-s]
  nosdn: Open vSwitch.
  odl: OpenDayLight Lithium version.
  opencontrail: OpenContrail.
  onos: ONOS framework as SDN.
[-t]
  nonha: NO HA mode of OpenStack.
  ha: HA mode of OpenStack.
  tip: The tip of the development.
[-o]
  juno: OpenStack Juno version.
  liberty: OpenStack Liberty version.
[-l]
  default: For virtual deployment where installation will be done on KVM created using ./02-maasdeplo
  intelpod5: Install on bare metal OPNFV pod5 of the Intel lab.
  intelpod6: Install on bare metal OPNFV pod6 of the Intel lab.
  orangepod2: Install on bare metal OPNFV pod2 of the Orange lab.
  (other pods)
  Note: if you make changes as per your pod above then please use your pod.
[-f]
  none: no special feature will be enabled.
  ipv6: IPv6 will be enabled for tenant in OpenStack.
```

The script will call 00-bootstrap.sh to bootstrap the Juju VM node, then it will call 01-deploybundle.sh with the corrosponding parameter values.

```
./01-deploybundle.sh $opnfvtype $openstack $opnfvlab $opnfvsdn $opnfvfeature
```

You will notice in the 01-deploybundle.sh, it copies over the charm bundle file based on the ha/nonha/tip setting:

```
case "$1" in
    'nonha' )
        cp $4/juju-deployer/ovs-$4-nonha.yaml ./bundles.yaml
        ;;
    'ha' )
        cp $4/juju-deployer/ovs-$4-ha.yaml ./bundles.yaml
        ;;
    'tip' )
        cp $4/juju-deployer/ovs-$4-tip.yaml ./bundles.yaml
        cp common/source/* ./
        sed -i -- "s|branch: master|branch: stable/$2|g" ./*.yaml
        ;;
    * )
        cp $4/juju-deployer/ovs-$4-nonha.yaml ./bundles.yaml
        ;;
esac
```

After the respective yaml file is copied over and renamed to bundle.yaml, in the next section, it will update the bundle.yaml based on your network configuration and environment. For example, for the Juniper pod 1, we need to change vip suffix from 10.4.1.1 to 172.16.50.1, which is our admin network, and eth1 is on the public network.

```
'juniperpod1' )
    sed -i -- 's/10.4.1.1/172.16.50.1/g' ./bundles.yaml
    sed -i -- 's/#ext-port: "eth1"/ext-port: "eth1"/g' ./bundles.yaml
    ;;
```

NOTE: If you are using a separate data network, then add this line below along with other changes, which signify that network 10.4.9.0/24 will be used as the data network for openstack.

```
sed -i -- 's/#os-data-network: 10.4.8.0\/21/os-data-network: 10.4.9.0\/24/g' ./bundles.yaml
```

By default debug is enabled in the deploy.sh script and error messages will be printed on the SSH terminal where you are running the scripts. It could take an hour to a couple of hours (maximum) to complete. Here is an example output of the deployment: http://pastebin.ubuntu.com/15006924/

You can check the status of the deployment by running this command in another terminal:

```
$ watch juju status --format tabular
```

This will refresh the juju status output in tabular format every 2 seconds. Here is an example output of juju status –format tabular: http://pastebin.ubuntu.com/15134109/

Next we will show you what Juju is deploying and to where, and how you can modify based on your own needs.

3.6 OPNFV Juju Charm Bundles

The magic behind Juju is a collection of software components called charms. They contain all the instructions necessary for deploying and configuring cloud-based services. The charms publicly available in the online Charm Store represent the distilled DevOps knowledge of experts.

A bundle is a set of services with a specific configuration and their corresponding relations that can be deployed together in a single step. Instead of deploying a single service, they can be used to deploy an entire workload, with

working relations and configuration. The use of bundles allows for easy repeatability and for sharing of complex, multi-service deployments.

For OPNFV, we have collected the charm bundles for each SDN deployment. They are stored in each SDN directory in ~/joid/ci. In each SDN folder, there are 3 bundle.yaml files, one for HA, one for non-HA, and the other for tip. For example for OpenDaylight:

```
~/joid/ci/odl/juju-deployer$ ls
ovs-odl-ha.yaml  ovs-odl-nonha.yaml  ovs-odl-tip.yaml  scripts
~/joid/ci/odl/juju-deployer$
```

We use Juju-deployer to deploy a set of charms via a yaml configuration file. You can find the complete format guide for the Juju-deployer configuration file here: http://pythonhosted.org/juju-deployer/config.html

Let's take a quick look at the ovs-odl-nonha.yaml to give you an idea about the charm bundle.

Assuming we are deploying OpenDayling with OpenStack Liberty in non-HA mode, according to the deploy.sh, we know it will run these two commands:

```
juju-deployer -vW -d -t 3600 -c bundles.yaml trusty-liberty-nodes
juju-deployer -vW -d -t 7200 -r 5 -c bundles.yaml trusty-liberty
```

In the ovs-odl-nonha.yaml file, find the section of 'trusty-liberty-nodes' close to the bottom of the file:

```
trusty-liberty-nodes:
  inherits: openstack-phase1
  overrides:
    series: trusty
```

It inherits 'openstack-phase1', which you will find in the beginning of the file:

```
openstack-phase1:
  series: trusty
  services:
    nodes-api:
      charm: "cs:trusty/ubuntu"
      num_units: 1
      constraints: tags=control
    nodes-compute:
      charm: "cs:trusty/ubuntu"
      num_units: 1
      constraints: tags=compute
    ntp:
      charm: "cs:trusty/ntp"
  relations:
    - - "ntp:juju-info"
      - "nodes-api:juju-info"
    - - "ntp:juju-info"
      - "nodes-compute:juju-info"
```

In the 'services' subsection, here we deploy the 'Ubuntu Trusty charm from the charm store,' name the service 'nodes-api,' deploy just one unit, and assign a tag of 'control' to this service. You can deploy the same charm and name it differently such as the second service 'nodes-compute.' The third service we deploy is named 'ntp' and is deployed from the NTP Trusty charm from the Charm Store. The NTP charm is a subordinate charm, which is designed for and deployed to the running space of another service unit.

The tag here is related to what we define in the deployment.yaml file for the MAAS-deployer. When 'constraints' is set, Juju will ask its provider, in this case MAAS, to provide a resource with the tags. In this case, Juju is asking one resource tagged with control and one resource tagged with compute from MAAS. Once the resource information is passed to Juju, Juju will start the installation of the specified version of Ubuntu.

In the next subsection, we define the relations between the services. The beauty of Juju and charms is you can define the relation of two services and all the service units deployed will set up the relations accordingly. This makes scaling out a very easy task. Here we add the relation between NTP and the two bare metal services.

Once the relations are established, Juju-deployer considers the deployment complete and moves to the next.

```
juju-deployer -vW -d -t 7200 -r 5 -c bundles.yaml trusty-liberty
```

It will start at the 'trusty-liberty' section, which inherits the 'contrail' section, which inherits the 'openstack-phase2' section. it follows the same services and relations format as above. We will take a look at another common service configuration next.

```
nova-cloud-controller:
  branch: lp:~openstack-charmers/charms/trusty/nova-cloud-controller/next
  num_units: 1
  options:
    network-manager: Neutron
  to:
    - "lxc:nodes-api=0"
```

We define a service name 'nova-cloud-controller,' which is deployed from the next branch of the nova-cloud-controller Trusty charm hosted on the Launchpad openstack-charmers team. The number of units to be deployed is 1. We set the network-manager option to 'Neutron.' This 1-service unit will be deployed to a LXC container at service 'nodes-api' unit 0.

To find out what other options there are for this particular charm, you can go to the code location at http://bazaar.launchpad.net/~openstack-charmers/charms/trusty/nova-cloud-controller/next/files and the options are defined in the config.yaml file.

Once the service unit is deployed, you can see the current configuration by running juju get:

```
$ juju get nova-cloud-controller
```

You can change the value with juju set, for example:

```
$ juju set nova-cloud-controller network-manager='FlatManager'
```

Charms encapsulate the operation best practices. The number of options you need to configure should be at the minimum. The Juju Charm Store is a great resource to explore what a charm can offer you. Following the nova-cloud-controller charm example, here is the main page of the recommended charm on the Charm Store: https://jujucharms.com/nova-cloud-controller/trusty/66

If you have any questions regarding Juju, please join the IRC channel #opnfv-joid on freenode for JOID related questions or #juju for general questions.

3.7 Testing Your Deployment

Once juju-deployer is complete, use juju status –format tabular to verify that all deployed units are in the ready state.

Find the Openstack-dashboard IP address from the juju status output, and see if you can login via a web browser. The username and password is admin/openstack.

Optionally, see if you can log in to the Juju GUI. The Juju GUI is on the Juju bootstrap node, which is the second VM you define in the 02-maasdeploy.sh file. The username and password is admin/admin.

If you deploy OpenDaylight, OpenContrail or ONOS, find the IP address of the web UI and login. Please refer to each SDN bundle.yaml for the login username/password.

3.8 Troubleshooting

Logs are indispensable when it comes time to troubleshoot. If you want to see all the service unit deployment logs, you can run juju debug-log in another terminal. The debug-log command shows the consolidated logs of all Juju agents (machine and unit logs) running in the environment.

To view a single service unit deployment log, use juju ssh to access to the deployed unit. For example to login into nova-compute unit and look for /var/log/juju/unit-nova-compute-0.log for more info.

```
$ juju ssh nova-compute/0
```

Example:

```
ubuntu@R4N4B1:~$ juju ssh nova-compute/0
Warning: Permanently added '172.16.50.60' (ECDSA) to the list of known hosts.
Warning: Permanently added '3-r4n3b1-compute.maas' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-77-generic x86_64)

* Documentation:  https://help.ubuntu.com/
<skipped>
Last login: Tue Feb  2 21:23:56 2016 from bootstrap.maas
ubuntu@3-R4N3B1-compute:~$ sudo -i
root@3-R4N3B1-compute:~# cd /var/log/juju/
root@3-R4N3B1-compute:/var/log/juju# ls
machine-2.log  unit-ceilometer-agent-0.log  unit-ceph-osd-0.log  unit-neutron-contrail-0.log  unit-no
root@3-R4N3B1-compute:/var/log/juju#
```

NOTE: By default Juju will add the Ubuntu user keys for authentication into the deployed server and only ssh access will be available.

Once you resolve the error, go back to the jump host to rerun the charm hook with:

```
$ juju resolved --retry <unit>
```

If you would like to start over, run juju destroy-environment <environment name> to release the resources, then you can run deploy.sh again.

```
$ juju destroy-environment demo-maas
WARNING! this command will destroy the "demo-maas" environment (type: maas)
This includes all machines, services, data and other resources.

Continue [y/N]? y
$
```

If there is an error destroying the environment, use –force.

```
$ juju destroy-environment demo-maas --force
$
```

If the above command hangs, use Ctrl-C to get out of it, and manually remove the environment file in the ~/.juju/environments/ directory.

```
$ ls ~/.juju/environments/
demo-maas.jenv
$ sudo rm ~/.juju/environments/demo-maas.jenv
$
```

The following are the common issues we have collected from the community:

- The right variables are not passed as part of the deployment procedure.

```
./deploy.sh -o liberty -s odl -t ha -l intelpod5 -f none
```

- If you have setup maas not with 02-maasdeply.sh then the ./clean.sh command could hang, the juju status command may hang because the correct MAAS API keys are not listed in environments.yaml, or environments.yaml does not exist in the current working directory. Solution: Please make sure you have an environments.yaml file under joid/ci directory and the correct MAAS API key has been listed.
- **Deployment times out:** use the command juju status –format=tabular and make sure all service containers receive an IP address and they are executing code. Ensure there is no service in the error state.
- In case the cleanup process hangs,remove the files from the ~/.juju/ directory except environments.yaml and shutdown all nodes manually.

Direct console access via the OpenStack GUI can be quite helpful if you need to login to a VM but cannot get to it over the network. It can be enabled by setting the `console-access-protocol` in the `nova-cloud-controller` to `vnc`. One option is to directly edit the juju-deployer bundle and set it there prior to deploying OpenStack.

```
nova-cloud-controller:
options:
  console-access-protocol: vnc
```

To access the console, just click on the instance in the OpenStack GUI and select the Console tab.

CHAPTER

FOUR

POST INSTALLATION CONFIGURATION

4.1 Configuring OpenStack

At the end of the deployment, the admin-openrc with OpenStack login credentials will be created for you. You can source the file and start configuring OpenStack via CLI.

```
~/joid/ci/cloud$ cat admin-openrc
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://172.16.50.114:5000/v2.0
export OS_REGION_NAME=Canonical
~/joid/ci/cloud$
```

We have prepared some scripts to help your configure the OpenStack cloud that you just deployed. In each SDN directory, for example joid/ci/opencontrail, there is a 'scripts' folder where you can find the scripts. These scripts are created to help you configure a basic OpenStack Cloud to verify the cloud. For more information on OpenStack Cloud configuration, please refer to the OpenStack Cloud Administrator Guide: http://docs.openstack.org/user-guide-admin/. Similarly, for complete SDN configuration, please refer to the respective SDN administrator guide.

Each SDN solution requires slightly different setup. Please refer to the README in each SDN folder. Most likely you will need to modify the openstack.sh and cloud-setup.sh scripts for the floating IP range, private IP network, and SSH keys. Please go through openstack.sh, glance.sh and cloud-setup.sh and make changes as you see fit.

Let's take a look at those for the Open vSwitch and briefly go through each script so you know what you need to change for your own environment.

```
~/joid/ci/nosdn/juju-deployer/scripts$ ls
cloud-setup.sh  glance.sh  openstack.sh
~/joid/ci/nosdn/juju-deployer/scripts$
```

4.1.1 openstack.sh

Let's first look at 'openstack.sh'. First there are 3 functions defined, configOpenrc(), unitAddress(), and unitMachine().

```
configOpenrc()
{
  cat <<-EOF
    export OS_USERNAME=$1
    export OS_PASSWORD=$2
    export OS_TENANT_NAME=$3
    export OS_AUTH_URL=$4
    export OS_REGION_NAME=$5
```

```
  EOF
}

unitAddress()
{
  juju status | python -c "import yaml; import sys; print yaml.load(sys.stdin)[\"services\"][\"$1\"]
}

unitMachine()
{
  juju status | python -c "import yaml; import sys; print yaml.load(sys.stdin)[\"services\"][\"$1\"]
}
```

The function configOpenrc() creates the OpenStack login credentials, the function unitAddress() finds the IP address of the unit, and the function unitMachine() finds the machine info of the unit.

```
mkdir -m 0700 -p cloud
controller_address=$(unitAddress keystone 0)
configOpenrc admin openstack admin http://$controller_address:5000/v2.0 Canonical > cloud/admin-openr
chmod 0600 cloud/admin-openrc
```

This creates a folder named 'cloud', finds the IP address of the keystone unit 0, feeds in the OpenStack admin credentials to a new file name 'admin-openrc' in the 'cloud' folder and change the permission of the file. It's important to change the credentials here if you use a different password in the deployment Juju charm bundle.yaml.

```
machine=$(unitMachine glance 0)
juju scp glance.sh cloud/admin-openrc $machine:
juju run --machine $machine ./glance.sh
```

This section first finds the machine ID of the glance service unit 0, transfers the glance.sh and admin-openrc files over to the glance unit 0, and then run the glance.sh in the glance unit 0. We will take a look at the glance.sh in the next section.

```
machine=$(unitMachine nova-cloud-controller 0)
juju scp cloud-setup.sh cloud/admin-openrc ~/.ssh/id_rsa.pub $machine:
juju run --machine $machine ./cloud-setup.sh
```

This section first finds the the machine ID of the nova-cloud-controller service unit 0, transfers 3 files over to the nova-cloud-controller unit 0, and then runs the cloud-setup.sh in the nova-cloud-controller unit 0. We will take a look at the cloud-setup.sh following glance.sh.

4.1.2 glance.sh

```
. ~/admin-openrc
```

First, this script sources the admin-openrc file.

```
wget -P /tmp/images http://download.cirros-cloud.net/0.3.3/cirros-0.3.3-x86_64-disk.img
wget -P /tmp/images http://cloud-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-disk1.
```

Download two images, Cirros and Ubuntu Trusty cloud image to /tmp/images folder.

```
glance image-create --name "cirros-0.3.3-x86_64" --file /tmp/images/cirros-0.3.3-x86_64-disk.img --di
glance image-create --name "ubuntu-trusty-daily" --file /tmp/images/trusty-server-cloudimg-amd64-disk
rm -rf /tmp/images
```

Use the glance python client to upload those two images, and finally remove those images from the local file system.

If you wish to use different images, please change the image download links and filenames here accordingly.‘

NOTE: The image downloading and uploading might take too long and time out. In this case, use juju ssh glance/0 to log in to the glance unit 0 and run the script again, or manually run the glance commands.

4.1.3 cloud-setup.sh

```
. ~/admin-openrc
```

First, source the the admin-openrc file.

```
# adjust tiny image
nova flavor-delete m1.tiny
nova flavor-create m1.tiny 1 512 8 1
```

Adjust the tiny image profile as the default tiny instance is too small for Ubuntu.

```
# configure security groups
neutron security-group-rule-create --direction ingress --ethertype IPv4 --protocol icmp --remote-ip-p
neutron security-group-rule-create --direction ingress --ethertype IPv4 --protocol tcp --port-range-r
```

Open up the ICMP and SSH access in the default security group.

```
# import key pair
keystone tenant-create --name demo --description "Demo Tenant"
keystone user-create --name demo --tenant demo --pass demo --email demo@demo.demo

nova keypair-add --pub-key id_rsa.pub ubuntu-keypair
```

Create a project called ‘demo’ and create a user called ‘demo’ in this project. Import the key pair.

```
# configure external network
neutron net-create ext-net --router:external --provider:physical_network external --provider:network_
neutron subnet-create ext-net --name ext-subnet --allocation-pool start=10.5.8.5,end=10.5.8.254 --dis
```

This section configures an external network ‘ext-net’ with a subnet called ‘ext-subnet’. In this subnet, the IP pool starts at 10.5.8.5 and ends at 10.5.8.254. DHCP is disabled. The gateway is at 10.5.8.1, and the subnet mask is 10.5.8.0/24. These are the public IPs that will be requested and associated to the instance. Please change the network configuration according to your environment.

```
# create vm network
neutron net-create demo-net
neutron subnet-create --name demo-subnet --gateway 10.20.5.1 demo-net 10.20.5.0/24
```

This section creates a private network for the instances. Please change accordingly.

```
neutron router-create demo-router

neutron router-interface-add demo-router demo-subnet

neutron router-gateway-set demo-router ext-net
```

This section creates a router and connects this router to the two networks we just created.

```
# create pool of floating ips
i=0
while [ $i -ne 10 ]; do
  neutron floatingip-create ext-net
  i=$((i + 1))
done
```

Finally, the script will request 10 floating IPs.

CHAPTER

FIVE

APPENDIX A: SINGLE NODE DEPLOYMENT

By default, running the script ./02-maasdeploy.sh will automatically create the KVM VMs on a single machine and configure everything for you.

```
* )
    virtinstall=1
    ./cleanvm.sh
    cp maas/default/deployment.yaml ./deployment.yaml
    ;;
```

Please change ~/joid/ci/maas/default/deployment.yaml accordingly. The MAAS-deployer will do the following: 1. Create 2 VMs (KVM). 2. Install MAAS in one of the VMs. 3. Configure MAAS to enlist and commission a VM for Juju bootstrap node.

Later, the 02-massdeploy.sh script will create two additional VMs and register them into the MAAS Server:

```
if [ "$virtinstall" -eq 1 ]; then
    # create two more VMs to do the deployment.
    sudo virt-install --connect qemu:///system --name node1-control --ram 8192 --vcpus 4 --disk size=
    sudo virt-install --connect qemu:///system --name node2-compute --ram 8192 --vcpus 4 --disk size=

    node1controlmac=`grep  "mac address" node1-control | head -1 | cut -d "'" -f 2`
    node2computemac=`grep  "mac address" node2-compute | head -1 | cut -d "'" -f 2`

    sudo virsh -c qemu:///system define --file node1-control
    sudo virsh -c qemu:///system define --file node2-compute

    maas maas tags new name='control'
    maas maas tags new name='compute'

    controlnodeid=`maas maas nodes new autodetect_nodegroup='yes' name='node1-control' tags='control

    maas maas tag update-nodes control add=$controlnodeid

    computenodeid=`maas maas nodes new autodetect_nodegroup='yes' name='node2-compute' tags='compute

    maas maas tag update-nodes compute add=$computenodeid

fi
```

CHAPTER

SIX

APPENDIX B: AUTOMATIC DEVICE DISCOVERY

If your bare metal servers support IPMI, they can be discovered and enlisted automatically by the MAAS server. You need to configure bare metal servers to PXE boot on the network interface where they can reach the MAAS server. With nodes set to boot from a PXE image, they will start, look for a DHCP server, receive the PXE boot details, boot the image, contact the MAAS server and shut down.

During this process, the MAAS server will be passed information about the node, including the architecture, MAC address and other details which will be stored in the database of nodes. You can accept and commission the nodes via the web interface. When the nodes have been accepted the selected series of Ubuntu will be installed.

CHAPTER

SEVEN

APPENDIX C: MACHINE CONSTRAINTS

Juju and MAAS together allow you to assign different roles to servers, so that hardware and software can be configured according to their roles. We have briefly mentioned and used this feature in our example. Please visit Juju Machine Constraints https://jujucharms.com/docs/stable/charms-constraints and MAAS tags https://maas.ubuntu.com/docs/tags.html for more information.

CHAPTER

EIGHT

APPENDIX D: OFFLINE DEPLOYMENT

When you have limited access policy in your environment, for example, when only the Jump Host has Internet access, but not the rest of the servers, we provide tools in JOID to support the offline installation.

The following package set is provided to those wishing to experiment with a 'disconnected from the internet' setup when deploying JOID utilizing MAAS. These instructions provide basic guidance as to how to accomplish the task, but it should be noted that due to the current reliance of MAAS and DNS, that behavior and success of deployment may vary depending on infrastructure setup. An official guided setup is in the roadmap for the next release:

1. Get the packages from here: https://launchpad.net/~thomnico/+archive/ubuntu/ubuntu-cloud-mirrors

   NOTE: The mirror is quite large 700GB in size, and does not mirror SDN repo/ppa.

2. Additionally to make juju use a private repository of charms instead of using an external location are provided via the following link and configuring environments.yaml to use cloudimg-base-url: https://github.com/juju/docs/issues/757