daisy

Release Latest

May 09, 2019

Contents

CHAPTER 1

OPNFV Daisy4nfv Installation Guide

1.1 Abstract

This document describes how to install the Fraser release of OPNFV when using Daisy4nfv as a deployment tool covering it's limitations, dependencies and required resources.

1.2 Version history

Date	Ver.	Author	Comment
2017-02-07	0.0.1	Zhijiang Hu (ZTE)	Initial version

1.3 Daisy4nfv configuration

This document provides guidelines on how to install and configure the Fraser release of OPNFV when using Daisy as a deployment tool including required software and hardware configurations.

Installation and configuration of host OS, OpenStack etc. can be supported by Daisy on Virtual nodes and Bare Metal nodes.

The audience of this document is assumed to have good knowledge in networking and Unix/Linux administration.

1.3.1 Prerequisites

Before starting the installation of the Fraser release of OPNFV, some plannings must be done.

Retrieve the installation iso image

First of all, the installation iso which includes packages of Daisy, OS, OpenStack, and so on is needed for deploying your OPNFV environment.

The stable release iso image can be retrieved via OPNFV software download page

The daily build iso image can be retrieved via OPNFV artifact repository:

http://artifacts.opnfv.org/daisy.html

NOTE: Search the keyword "daisy/Fraser" to locate the iso image.

E.g. daisy/opnfv-2017-10-06_09-50-23.iso

Download the iso file, then mount it to a specified directory and get the opnfv-*.bin from that directory.

The git url and sha512 checksum of iso image are recorded in properties files. According to these, the corresponding deployment scripts can be retrieved.

Retrieve the deployment scripts

To retrieve the repository of Daisy on Jumphost use the following command:

- git clone https://gerrit.opnfv.org/gerrit/daisy

To get stable Fraser release, you can use the following command:

- git checkout opnfv.6.0

1.3.2 Setup Requirements

If you have only 1 Bare Metal server, Virtual deployment is recommended. if you have more than 3 servers, the Bare Metal deployment is recommended. The minimum number of servers for each role in Bare metal deployment is listed below.

Role	Number of Servers
Jump Host	1
Controller	1
Compute	1

Jumphost Requirements

The Jumphost requirements are outlined below:

1. CentOS 7.2 (Pre-installed).
2. Root access.
3. Libvirt virtualization support(For virtual deployment).
4. Minimum 1 NIC(or 2 NICs for virtual deployment).
    - PXE installation Network (Receiving PXE request from nodes and providing OS provisioning)
    - IPMI Network (Nodes power control and set boot PXE first via IPMI interface)
    - Internet access (For getting latest OS updates)

- External Interface(For virtual deployment, exclusively used by instance traffic to access the rest of the Internet)

5. 16 GB of RAM for a Bare Metal deployment, 64 GB of RAM for a Virtual deployment.
6. CPU cores: 32, Memory: 64 GB, Hard Disk: 500 GB, (Virtual deployment needs 1 TB Hard Disk)

1.3.3 Bare Metal Node Requirements

Bare Metal nodes require:

1. IPMI enabled on OOB interface for power control.
2. BIOS boot priority should be PXE first then local hard disk.
3. Minimum 1 NIC for Compute nodes, 2 NICs for Controller nodes.
    - PXE installation Network (Broadcasting PXE request)
    - IPMI Network (Receiving IPMI command from Jumphost)
    - Internet access (For getting latest OS updates)
    - External Interface(For virtual deployment, exclusively used by instance traffic to access the rest of the Internet)

1.3.4 Network Requirements

Network requirements include:

1. No DHCP or TFTP server running on networks used by OPNFV.
2. 2-7 separate networks with connectivity between Jumphost and nodes.
    - PXE installation Network
    - IPMI Network
    - Internet access Network
    - OpenStack Public API Network
    - OpenStack Private API Network
    - OpenStack External Network
    - OpenStack Tenant Network(currently, VxLAN only)
3. Lights out OOB network access from Jumphost with IPMI node enabled (Bare Metal deployment only).
4. Internet access Network has Internet access, meaning a gateway and DNS availability.
5. OpenStack External Network has Internet access too if you want instances to access the Internet.

Note: **All networks except OpenStack External Network can share one NIC(Default configuration) or use an exclusive NIC(Reconfigurated in network.yml).**

1.3.5 Execution Requirements (Bare Metal Only)

In order to execute a deployment, one must gather the following information:

1. IPMI IP addresses of the nodes.

2. IPMI login information for the nodes (user/password).

1.4 Installation Guide (Bare Metal Deployment)

1.4.1 Nodes Configuration (Bare Metal Deployment)

The below file is the inventory template of deployment nodes:

"./deploy/config/bm_environment/zte-baremetal1/deploy.yml"

You can write your own name/roles reference into it.

- name – Host name for deployment node after installation.
- roles – Components deployed. CONTROLLER_LB is for Controller,

COMPUTER is for Compute role. Currently only these two roles are supported. The first CONTROLLER_LB is also used for ODL controller. 3 hosts in inventory will be chosen to setup the Ceph storage cluster.

Set TYPE and FLAVOR

E.g.

```
TYPE: virtual
FLAVOR: cluster
```

Assignment of different roles to servers

E.g. OpenStack only deployment roles setting

```
hosts:
  - name: host1
    roles:
      - CONTROLLER_LB
  - name: host2
    roles:
      - COMPUTER
  - name: host3
    roles:
      - COMPUTER
```

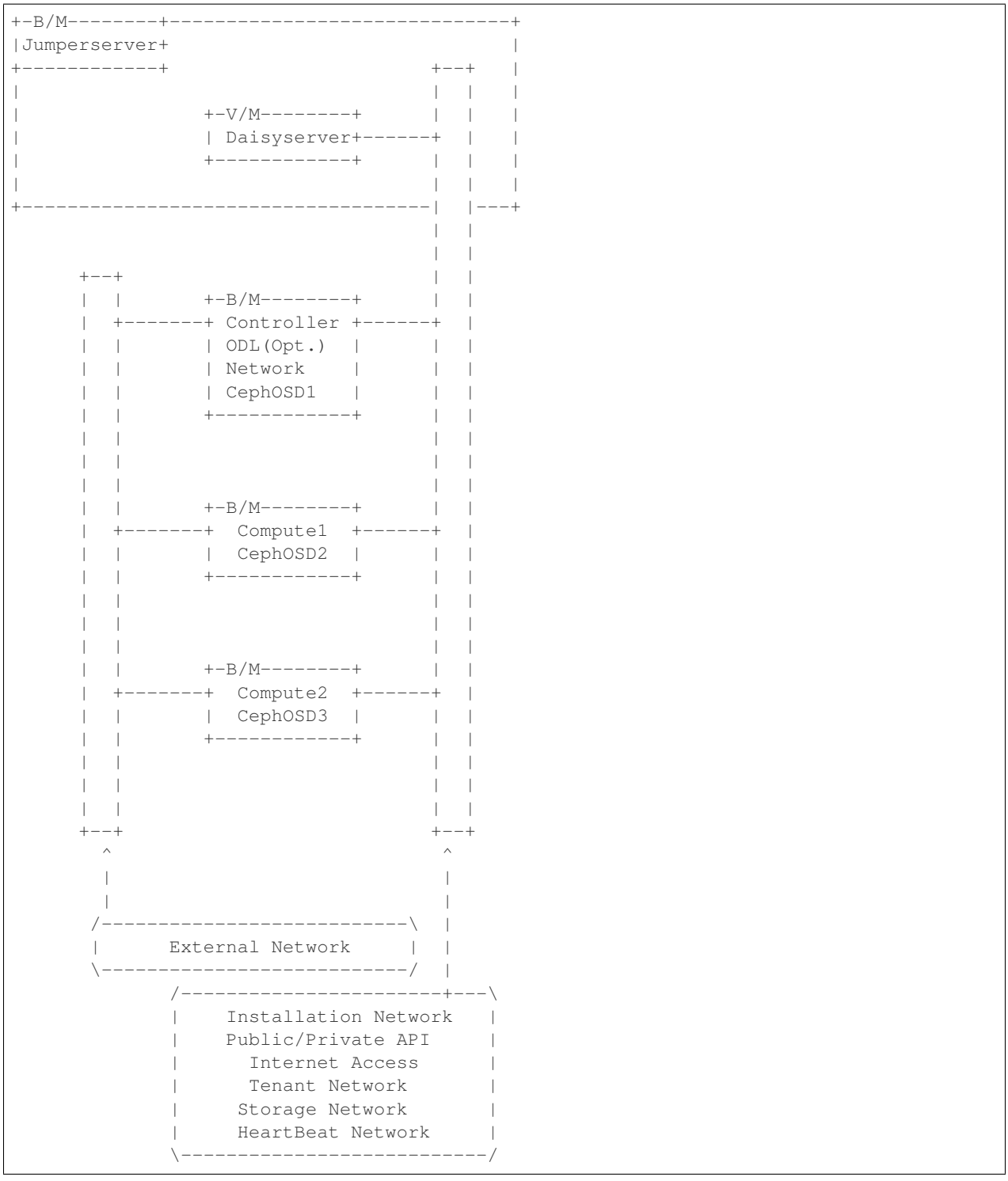
NOTE: For B/M, Daisy uses MAC address defined in deploy.yml to map discovered nodes to node items definition in deploy.yml, then assign role described by node item to the discovered nodes by name pattern. Currently, controller01, controller02, and controller03 will be assigned with Controler role while computer01, 'computer02, computer03, and computer04 will be assigned with Compute role.

NOTE: For V/M, There is no MAC address defined in deploy.yml for each virtual machine. Instead, Daisy will fill that blank by getting MAC from "virsh dump-xml".

1.4.2 Network Configuration (Bare Metal Deployment)

Before deployment, there are some network configurations to be checked based on your network topology. The default network configuration file for Daisy is "./deploy/config/bm_environment/zte-baremetal1/network.yml". You can write your own reference into it.

The following figure shows the default network configuration.



Note: For Flat External networks(which is used by default), a physical interface is needed on each compute node for ODL NetVirt recent versions. HeartBeat network is selected,and if it is configured in network.yml,the keepalived interface will be the heartbeat interface.

1.4.3 Start Deployment (Bare Metal Deployment)

(1) Git clone the latest daisy4nfv code from opnfv: "git clone https://gerrit.opnfv.org/gerrit/daisy"

(2) Download latest bin file(such as opnfv-2017-06-06_23-00-04.bin) of daisy from http://artifacts.opnfv.org/daisy.html and change the bin file name(such as opnfv-2017-06-06_23-00-04.bin) to opnfv.bin. Check the https://build.opnfv.org/ci/job/daisy-os-odl-nofeature-ha-baremetal-daily-master/, and if the 'snaps_health_check' of functest result is 'PASS', you can use this verify-passed bin to deploy the openstack in your own environment

(3) Assumed cloned dir is $workdir, which laid out like below: [root@daisyserver daisy]# ls ci deploy docker INFO LICENSE requirements.txt templates tests tox.ini code deploy.log docs known_hosts setup.py test-requirements.txt tools Make sure the opnfv.bin file is in $workdir

(4) Enter into the $workdir, which laid out like below: [root@daisyserver daisy]# ls ci code deploy docker docs INFO LICENSE requirements.txt setup.py templates test-requirements.txt tests tools tox.ini Create folder of labs/zte/pod2/daisy/config in $workdir

(5) Move the ./deploy/config/bm_environment/zte-baremetal1/deploy.yml and ./deploy/config/bm_environment/zte-baremetal1/network.yml to labs/zte/pod2/daisy/config dir.

Note: If selinux is disabled on the host, please delete all xml files section of below lines in dir templates/physical_environment/vms/

<seclabel type='dynamic' model='selinux' relabel='yes'> <label>system_u:system_r:svirt_t:s0:c182,c195</label>
<imagelabel>system_u:object_r:svirt_image_t:s0:c182,c195</imagelabel>

</seclabel>

(6) Config the bridge in jumperserver,make sure the daisy vm can connect to the targetnode,use the command below: brctl addbr br7 brctl addif br7 enp3s0f3(the interface for jumperserver to connect to daisy vm) ifconfig br7 10.20.7.1 netmask 255.255.255.0 up service network restart

(7) Run the script deploy.sh in daisy/ci/deploy/ with command: sudo ./ci/deploy/deploy.sh -L $(cd ./;pwd) -l zte -p pod2 -s os-nosdn-nofeature-noha

Note: The value after -L should be a absolute path which points to the directory which contents labs/zte/pod2/daisy/config directory. The value after -p parameter(pod2) comes from path "labs/zte/pod2" The value after -l parameter(zte) comes from path "labs/zte" The value after -s "os-nosdn-nofeature-ha" used for deploying multinode openstack The value after -s "os-nosdn-nofeature-noha" used for deploying all-in-one openstack

(8) When deployed successfully,the floating ip of openstack is 10.20.7.11, the login account is "admin" and the password is "keystone"

1.5 Installation Guide (Virtual Deployment)

1.5.1 Nodes Configuration (Virtual Deployment)

The below file is the inventory template of deployment nodes:

"./deploy/conf/vm_environment/zte-virtual1/deploy.yml"

You can write your own name/roles reference into it.

- name – Host name for deployment node after installation.
- roles – Components deployed.

Set TYPE and FLAVOR

E.g.

```
TYPE: virtual
FLAVOR: cluster
```

Assignment of different roles to servers

E.g. OpenStack only deployment roles setting

```
hosts:
  - name: host1
    roles:
      - CONTROLLER_LB

  - name: host2
    roles:
      - COMPUTER
```

NOTE: For B/M, Daisy uses MAC address defined in deploy.yml to map discovered nodes to node items definition in deploy.yml, then assign role described by node item to the discovered nodes by name pattern. Currently, controller01, controller02, and controller03 will be assigned with Controller role while computer01, computer02, computer03, and computer04 will be assigned with Compute role.

NOTE: For V/M, There is no MAC address defined in deploy.yml for each virtual machine. Instead, Daisy will fill that blank by getting MAC from "virsh dump-xml".

E.g. OpenStack and ceph deployment roles setting

```
hosts:
  - name: host1
    roles:
      - controller

  - name: host2
    roles:
      - compute
```

1.5.2 Network Configuration (Virtual Deployment)

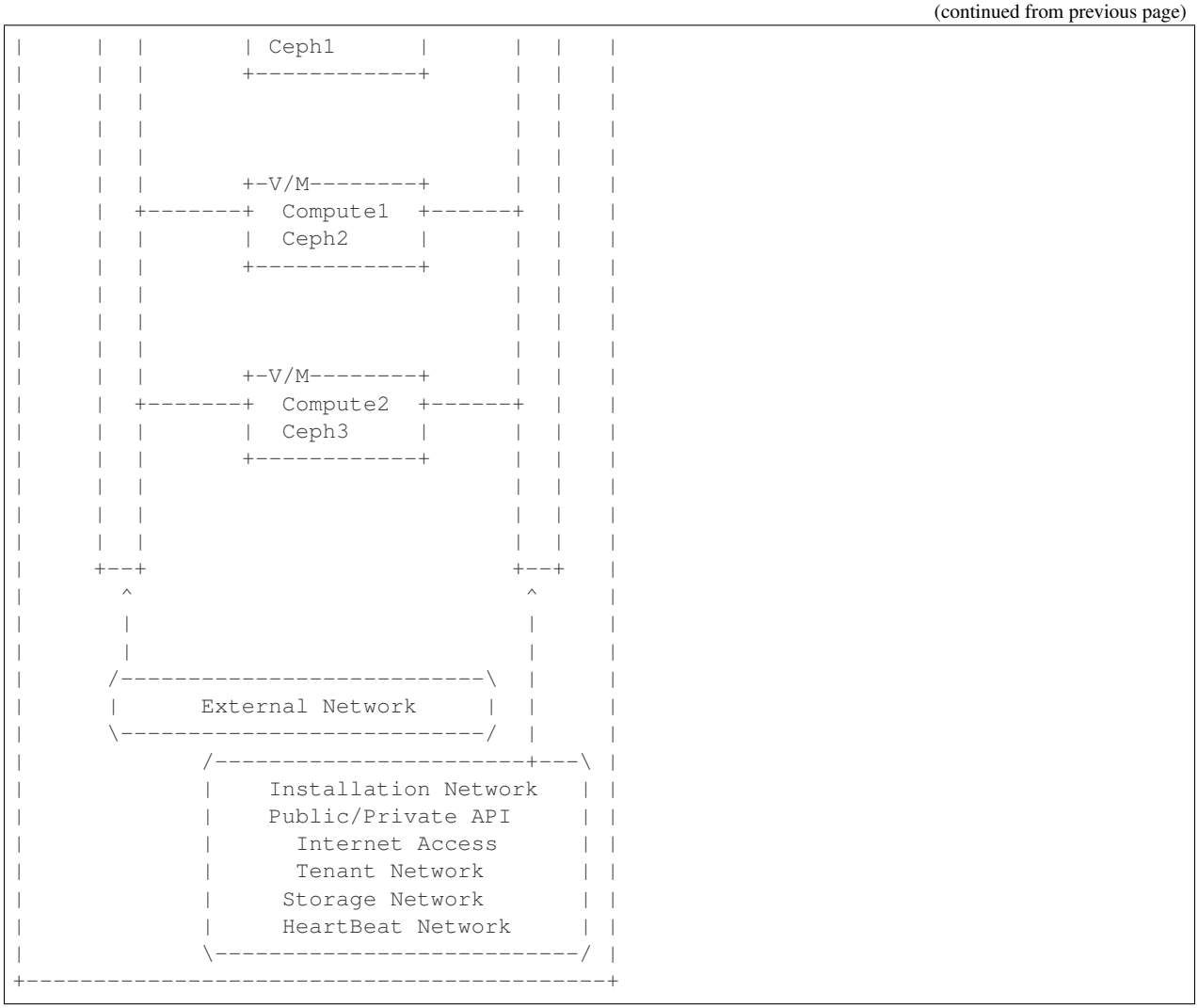
Before deployment, there are some network configurations to be checked based on your network topology. The default network configuration file for Daisy is "daisy/deploy/config/vm_environment/zte-virtual1/network.yml". You can write your own reference into it.

The following figure shows the default network configuration.

```
+-B/M--------+--------------------------------+
|Jumperserver+                                |
+------------+                          +--+  |
|                                       |  |  |
|                 +-V/M--------+        |  |  |
|                 | Daisyserver+------+ |  |  |
|                 +------------+        |  |  |
|                                       |  |  |
|     +--+                              |  |  |
|     |  |        +-V/M--------+        |  |  |
|     |  +--------+ Controller +------+ |  |  |
|     |  |        | ODL(Opt.)  |        |  |  |
|     |  |        | Network    |        |  |  |
```

(continues on next page)

(continued from previous page)



Note: For Flat External networks(which is used by default), a physical interface is needed on each compute node for ODL NetVirt recent versions. HeartBeat network is selected,and if it is configured in network.yml,the keepalived interface will be the heartbeat interface.

1.5.3 Start Deployment (Virtual Deployment)

(1) Git clone the latest daisy4nfv code from opnfv: "git clone https://gerrit.opnfv.org/gerrit/daisy", make sure the current branch is master

(2) Download latest bin file(such as opnfv-2017-06-06_23-00-04.bin) of daisy from http://artifacts.opnfv.org/daisy.html and change the bin file name(such as opnfv-2017-06-06_23-00-04.bin) to opnfv.bin. Check the https://build.opnfv.org/ci/job/daisy-os-odl-nofeature-ha-baremetal-daily-master/, and if the 'snaps_health_check' of functest result is 'PASS', you can use this verify-passed bin to deploy the openstack in your own environment

(3) Assumed cloned dir is $workdir, which laid out like below: [root@daisyserver daisy]# ls ci code deploy docker docs INFO LICENSE requirements.txt setup.py templates test-requirements.txt tests tools tox.ini Make sure the opnfv.bin file is in $workdir

(4) Enter into $workdir, Create folder of labs/zte/virtual1/daisy/config in $workdir

(5) Move the deploy/config/vm_environment/zte-virtual1/deploy.yml and deploy/config/vm_environment/zte-virtual1/network.yml to labs/zte/virtual1/daisy/config dir.

Note: zte-virtual1 config files deploy openstack with five nodes(3 lb nodes and 2 computer nodes), if you want to deploy an all-in-one openstack, change the zte-virtual1 to zte-virtual2

Note: If selinux is disabled on the host, please delete all xml files section of below lines in dir templates/virtual_environment/vms/

<seclabel type='dynamic' model='selinux' relabel='yes'> <label>system_u:system_r:svirt_t:s0:c182,c195</label>
<imagelabel>system_u:object_r:svirt_image_t:s0:c182,c195</imagelabel>

</seclabel>

(6) Run the script deploy.sh in daisy/ci/deploy/ with command: sudo ./ci/deploy/deploy.sh -L $(cd ./;pwd) -l zte -p virtual1 -s os-nosdn-nofeature-ha

Note: The value after -L should be an absolute path which points to the directory which includes labs/zte/virtual1/daisy/config directory. The value after -p parameter(virtual1) is got from labs/zte/virtual1/daisy/config/ The value after -l parameter(zte) is got from labs/ The value after -s "os-nosdn-nofeature-ha" used for deploying multinode openstack The value after -s "os-nosdn-nofeature-noha" used for deploying all-in-one openstack

(7) When deployed successfully,the floating ip of openstack is 10.20.11.11, the login account is "admin" and the password is "keystone"

1.6 Deployment Error Recovery Guide

Deployment may fail due to different kinds of reasons, such as Daisy VM creation error, target nodes failure during OS installation, or Kolla deploy command error. Different errors can be grouped into several error levels. We define Recovery Levels below to fulfill recover requirements in different error levels.

1.6.1 1. Recovery Level 0

This level restart whole deployment again. Mainly to retry to solve errors such as Daisy VM creation failed. For example we use the following command to do virtual deployment(in the jump host):

```
sudo ./ci/deploy/deploy.sh -b ./ -l zte -p virtual1 -s os-nosdn-nofeature-ha
```

If command failed because of Daisy VM creation error, then redoing above command will restart whole deployment which includes rebuilding the daisy VM image and restarting Daisy VM.

1.6.2 2. Recovery Level 1

If Daisy VM was created successfully, but bugs were encountered in Daisy code or software of target OS which prevent deployment from being done, in this case, the user or the developer does not want to recreate the Daisy VM again during next deployment process but just to modify some pieces of code in it. To achieve this, he/she can redo deployment by deleting all clusters and hosts first(in the Daisy VM):

```
source /root/daisyrc_admin
for i in `daisy cluster-list | awk -F "|" '{print $2}' | sed -n '4p' | tr -d " "`;do␣
↪daisy cluster-delete $i;done
for i in `daisy host-list | awk -F "|" '{print $2}'| grep -o "[^ ]\+\( \+[^ ]\+\)*
↪"|tail -n +2`;do daisy host-delete $i;done
```

Then, adjust deployment command as below and run it again(in the jump host):

```
sudo ./ci/deploy/deploy.sh -S -b ./ -l zte -p virtual1 -s os-nosdn-nofeature-ha
```

Pay attention to the "-S" argument above, it lets the deployment process to skip re-creating Daisy VM and use the existing one.

1.6.3 3. Recovery Level 2

If both Daisy VM and target node's OS are OK, but error ocurred when doing OpenStack deployment, then there is even no need to re-install target OS for the deployment retrying. In this level, all we need to do is just retry the Daisy deployment command as follows(in the Daisy VM):

```
source /root/daisyrc_admin
daisy uninstall <cluster-id>
daisy install <cluster-id>
```

This basically does kolla-ansible destruction and kolla-asnible deployment.

1.6.4 4. Recovery Level 3

If previous deployment was failed during kolla-ansible deploy(you can confirm it by checking /var/log/daisy/api.log) or if previous deployment was successful but the default configration is not what you want and it is OK for you to destroy the OPNFV software stack and re-deploy it again, then you can try recovery level 3.

For example, in order to use external iSCSI storage, you are about to deploy iSCSI cinder backend which is not enabled by default. First, cleanup the previous deployment.

ssh into daisy node, then do:

```
[root@daisy daisy]# source /etc/kolla/admin-openrc.sh
[root@daisy daisy]# openstack server delete <all vms you created>
```

Note: /etc/kolla/admin-openrc.sh may not have existed if previous deployment was failed during kolla deploy.

```
[root@daisy daisy]# cd /home/kolla_install/kolla-ansible/
[root@daisy kolla-ansible]# ./tools/kolla-ansible destroy \
-i ./ansible/inventory/multinode --yes-i-really-really-mean-it
```

Then, edit /etc/kolla/globals.yml and append the follwoing line:

```
enable_cinder_backend_iscsi: "yes"
enable_cinder_backend_lvm: "no"
```

Then, re-deploy again:

```
[root@daisy kolla-ansible]# ./tools/kolla-ansible prechecks -i ./ansible/inventory/
↪multinode
[root@daisy kolla-ansible]# ./tools/kolla-ansible deploy -i ./ansible/inventory/
↪multinode
```

After successfully deploying, issue the following command to generate /etc/kolla/admin-openrc.sh file.

```
[root@daisy kolla-ansible]# ./tools/kolla-ansible post-deploy -i ./ansible/inventory/
↪multinode
```

Finally, issue the following command to create necessary resources, and your environment are ready for running OPNFV functest.

```
[root@daisy kolla-ansible]# cd /home/daisy
[root@daisy daisy]# ./deploy/post.sh -n /home/daisy/labs/zte/virtual1/daisy/config/
↪network.yml
```

Note: "zte/virtual1" in above path may vary in your environment.

1.7 OpenStack Minor Version Update Guide

Thanks to Kolla's kolla-ansible upgrade function, Daisy can update OpenStack minor version as the follows:

1. Get new version file only from Daisy team. Since Daisy's Kolla images are built by meeting the OPNFV requirements and have their own file packaging layout, Daisy requires user to always use Kolla image file built by Daisy team. Currently, it can be found at http://artifacts.opnfv.org/daisy/upstream, or please see *this chapter* for how to build your own image.

2. Put new version file into /var/lib/daisy/versionfile/kolla/, for example: /var/lib/daisy/versionfile/kolla/kolla-image-ocata-170811155446.tgz

3. Add version file to Daisy's version management database then get the version ID.

```
[root@daisy ~]# source /root/daisyrc_admin
[root@daisy ~]# daisy version-add kolla-image-ocata-170811155446.tgz kolla
+-------------+--------------------------------------+
| Property    | Value                                |
+-------------+--------------------------------------+
| checksum    | None                                 |
| created_at  | 2017-08-28T06:45:25.000000           |
| description | None                                 |
| id          | 8be92587-34d7-43e8-9862-a5288c651079 |
| name        | kolla-image-ocata-170811155446.tgz   |
| owner       | None                                 |
| size        | 0                                    |
| status      | unused                               |
| target_id   | None                                 |
| type        | kolla                                |
| updated_at  | 2017-08-28T06:45:25.000000           |
| version     | None                                 |
+-------------+--------------------------------------+
```

4. Get cluster ID

```
[root@daisy ~]# daisy cluster-list
+--------------------------------------+-------------+...
| ID                                   | Name        |...
+--------------------------------------+-------------+...
| d4c1e0d3-c4b8-4745-aab0-0510e62f0ebb | clustertest |...
+--------------------------------------+-------------+...
```

5. Issue update command passing cluster ID and version ID

```
[root@daisy ~]# daisy update d4c1e0d3-c4b8-4745-aab0-0510e62f0ebb --update-object␣
↪kolla --version-id 8be92587-34d7-43e8-9862-a5288c651079
+----------+--------------+
```

(continues on next page)

(continued from previous page)

```
| Property | Value        |
+----------+--------------+
| status   | begin update |
+----------+--------------+
```

6. Since step 5's command is non-blocking, the user need to run the following command to get updating progress.

```
[root@daisy ~]# daisy host-list --cluster-id d4c1e0d3-c4b8-4745-aab0-0510e62f0ebb
...+---------------+-------------+-------------------------+
...| Role_progress | Role_status | Role_messages           |
...+---------------+-------------+-------------------------+
...| 0             | updating    | prechecking envirnoment |
...+---------------+-------------+-------------------------+
```

Notes. The above command returns many fields. User only have to take care about the Role_xxx fields in this case.

1.8 Build Your Own Kolla Image For Daisy

The following command will build Ocata Kolla image for Daisy based on Daisy's fork of openstack/kolla project. This is also the method Daisy used for the Euphrates release.

The reason why here use fork of openstack/kolla project is to backport ODL support from pike branch to ocata branch.

```
cd ./ci
./kolla-build.sh
```

After building, the above command will put Kolla image into /tmp/kolla-build-output directory and the image version will be 4.0.2.

If you want to build an image which can update 4.0.2, run the following command:

```
cd ./ci
./kolla-build.sh -e 1
```

This time the image version will be 4.0.2.1 which is higher than 4.0.2 so that it can be used to replace the old version.

1.9 Deployment Test Guide

After successful deployment of openstack, daisy4nfv use Functest to test the api of openstack. You can follow below instruction to test the successfully deployed openstack on jumperserver.

1.docker pull opnfv/functest run 'docker images' command to make sure have the latest functest images.

2.docker run -ti –name functest -e INSTALLER_TYPE="daisy"-e INSTALLER_IP="10.20.11.2" -e NODE_NAME="zte-vtest" -e DEPLOY_SCENARIO="os-nosdn-nofeature-ha" -e BUILD_TAG="jenkins-functest-daisy-virtual-daily-master-1259" -e DEPLOY_TYPE="virt" opnfv/functest:latest /bin/bash Before run above command change below parameters: DEPLOY_SCENARIO: indicate the scenario DEPLOY_TYPE: virt/baremetal NODE_NAME: pod name INSTALLER_IP: daisy vm node ip

3.Log in the daisy vm node to get the /etc/kolla/admin-openrc.sh file, and write them in /home/opnfv/functest/conf/openstack.creds file of functest container.

4.Run command 'functest env prepare' to prepare the functest env.

5.Run command 'functest testcase list' to list all the testcase can be run.

6.Run command ‘functest testcase run testcase_name’ to run the testcase_name testcase of functest.

CHAPTER 2

Release notes for Daisy4nfv

2.1 Abstract

This document compiles the release notes for the Fraser release of OPNFV when using Daisy as a deployment tool.

2.1.1 Configuration Guide

Before installing Daisy4NFV on jump server,you have to configure the daisy.conf file.Then put the right configured daisy.conf file in the /home/daisy_install/ dir.

1. you have to supplement the “daisy_management_ip” field with the ip of management ip of your Daisy server vm.
2. Now the backend field “default_backend_types” just support the “kolla”.
3. “os_install_type” field just support “pxe” for now.
4. Daisy now use pxe server to install the os, the “build_pxe” item must set to “no”.
5. “eth_name” field is the pxe server interface, and this field is required when the “build_pxe” field set to “yes”.This should be set to the interface (in Daisy Server VM) which will be used for communicating with other target nodes on management/PXE net plane. Default is ens3.
6. “ip_address” field is the ip address of pxe server interface.
7. “net_mask” field is the netmask of pxe server,which is required when the “build_pxe” is set to “yes”
8. “client_ip_begin” and “client_ip_end” field are the dhcp range of the pxe server.
9. If you want to use the multicast type to deliver the kolla image to target node, set the “daisy_conf_mcast_enabled” field to “True”

2.2 OpenStack Configuration Guide

2.2.1 Before The First Deployment

When executing deploy.sh, before doing real deployment, Daisy utilizes Kolla's service configuration functionality [1] to specify the following changes to the default OpenStack configuration which comes from Kolla as default.

a) If is it is a VM deployment, set virt_type=qemu amd cpu_mode=none for nova-compute.conf.

b) In nova-api.conf set default_floating_pool to the name of the external network which will be created by Daisy after deployment for nova-api.conf.

c) In heat-api.conf and heat-engine.conf, set deferred_auth_method to trusts and unset trusts_delegated_roles.

Those above changes are requirements of OPNFV or environment's constraints. So it is not recommended to change them. But if the user wants to add more specific configurations to OpenStack services before doing real deployment, we suggest to do it in the same way as deploy.sh do. Currently, this means hacking into deploy/prepare.sh or deploy/prepare/execute.py then add config file as described in [1].

Notes: Suggest to pass the first deployment first, then reconfigure and deploy again.

2.2.2 After The First Deployment

After the first time of deployment of OpenStack, its configurations can also be changed and applied by using Kolla's service configuration functionality [1]. But user has to issue Kolla's command to do it in this release:

[1] https://docs.openstack.org/kolla-ansible/latest/advanced-configuration.html#openstack-service-configuration-in-kolla

CHAPTER 3

Release notes for Daisy4nfv

3.1 Abstract

This document covers features, limitations and required system resources for the OPNFV Fraser release when using Daisy4nfv as a deployment tool.

3.1.1 Introduction

Daisy4nfv is an OPNFV installer project based on open source project Daisycloud-core, which provides containerized deployment and management of OpenStack and other distributed systems such as OpenDaylight.

3.1.2 Release Data

Project	Daisy4nfv
Repo/tag	daisy/opnfv-6.0
Release designation	opnfv-6.0
Release date	
Purpose of the delivery	OPNFV Fraser release

Deliverables

Software deliverables

- Daisy4NFV/opnfv-6.0 ISO, please get it from OPNFV software download page

Documentation deliverables

- OPNFV(Fraser) Daisy4nfv installation instructions
- OPNFV(Fraser) Daisy4nfv Release Notes

Version change

Module version change

This is the Fraser release of Daisy4nfv as a deployment toolchain in OPNFV, the following upstream components supported with this release.

- Centos 7.4
- Openstack (Pike release)
- Opendaylight (Carbon SR3)

Reason for new version

Feature additions

JIRA REFERENCE	SLOGAN
	Support OpenDayLight Carbon SR3
	Support OpenStack Pike
	Support OVS+DPDK

Bug corrections

JIRA TICKETS:

JIRA REFERENCE	SLOGAN

3.1.3 Known Limitations, Issues and Workarounds

System Limitations

Max number of blades: 1 Jumphost, 3 Controllers, 20 Compute blades

Min number of blades: 1 Jumphost, 1 Controller, 1 Compute blade

Storage: Ceph is the only supported storage configuration

Min Jumphost requirements: At least 16GB of RAM, 16 core CPU

Known issues

Scenario	Issue	Workarounds

3.1.4 Test Result

TODO

CHAPTER 4

Design Docs for Daisy4nfv

4.1 CI Job Introduction

4.1.1 CI Base Architech

https://wiki.opnfv.org/display/INF/CI+Evolution

4.1.2 Project Gating And Daily Deployment Test

To save time, currently, Daisy4NFV does not run deployment test in gate job which simply builds and uploads artifacts to low confidence level repo. The project deployment test is triggered on a daily basis. If the artifact passes the test, then it will be promoted to the high confidence level repo.

The low confidence level artifacts are bin files in http://artifacts.opnfv.org/daisy.html named like "daisy/opnfv-Gerrit-39495.bin", while the high confidence level artifacts are named like "daisy/opnfv-2017-08-20_08-00-04.bin".

The daily project deployment status can be found at

https://build.opnfv.org/ci/job/daisy-daily-master/

4.1.3 Production CI

The status of Daisy4NFV's CI/CD which running on OPNFV production CI environments(both B/M and VM) can be found at

https://build.opnfv.org/ci/job/daisy-os-nosdn-nofeature-ha-baremetal-daily-master/ https://build.opnfv.org/ci/job/daisy-os-odl-nofeature-ha-baremetal-daily-master/ https://build.opnfv.org/ci/job/daisy-os-nosdn-nofeature-ha-virtual-daily-master/ https://build.opnfv.org/ci/job/daisy-os-odl-nofeature-ha-virtual-daily-master/

Dashboard for taking a glance on CI health status in a more intuitive way can be found at

http://testresults.opnfv.org/reporting/functest/release/master/index-status-daisy.html

4.2 Deployment Steps

This document takes VM all-in-one environment as example to show what ci/deploy/deploy.sh really do.

1. On jump host, clean up all-in-one vm and networks.
2. On jump host, clean up daisy vm and networks.
3. On jump host, create and start daisy vm and networks.
4. In daisy vm, Install daisy artifact.
5. In daisy vm, config daisy and OpenStack default options.

6. In daisy vm, create cluster, update network and build PXE server for the bootstrap kernel. In short, be ready for discovering target nodes. These tasks are done by running the following command.

python /home/daisy/deploy/tempest.py –dha /home/daisy/labs/zte/virtual1/daisy/config/deploy.yml –network /home/daisy/labs/zte/virtual1/daisy/config/network.yml –cluster 'yes'

7. On jump host, create and start all-in-one vm and networks.
8. On jump host, after all-in-one vm is up, get its mac address and write into /home/daisy/labs/zte/virtual1/daisy/config/deploy.yml.

9. In daisy vm, check if all-in-one vm was discovered, if it was, then update its network assignment and config OpenStack according to OPNFV scenario and setup PXE for OS installaion. These tasks are done by running the following command.

python /home/daisy/deploy/tempest.py –dha /home/daisy/labs/zte/virtual1/daisy/config/deploy.yml –network /home/daisy/labs/zte/virtual1/daisy/config/network.yml –host yes –isbare 0 –scenario os-nosdn-nofeature-noha

Note: Current host status: os_status is "init".

10. On jump host, restart all_in_one vm to install OS.

11. In daisy vm, continue to intall OS by running the following command which for VM environment only.

python /home/daisy/deploy/tempest.py –dha /home/daisy/labs/zte/virtual1/daisy/config/deploy.yml –network /home/daisy/labs/zte/virtual1/daisy/config/network.yml –install 'yes'

12. In daisy vm, run the following command to check OS intallation progress. /home/daisy/deploy/check_os_progress.sh -d 0 -n 1

Note: Current host status: os_status is "installing" during installation, then os_status becomes "active" after OS was succesfully installed.

13. On jump host, reboot all-in-one vm again to get a fresh and first booted OS.

14. In daisy vm, run the following command to check OpenStack/ODL/... intallation progress.

/home/daisy/deploy/check_openstack_progress.sh -n 1

4.3 Kolla Image Multicast Design

4.3.1 Protocol Design

1. All Protocol headers are 1 byte long or align to 4 bytes.

2. Packet size should not exceed above 1500(MTU) bytes including UDP/IP header and should be align to 4 bytes. In future, MTU can be modified larger than 1500(Jumbo Frame) through cmd line option to enlarge the data throughput.

/* Packet header definition (align to 4 bytes) */ struct packet_ctl {

uint32_t seq; // packet seq number start from 0, unique in server life cycle. uint32_t crc; // checksum uint32_t data_size; // payload length uint8_t data[0];

};

/* Buffer info definition (align to 4 bytes) */ struct buffer_ctl {

uint32_t buffer_id; // buffer seq number start from 0, unique in server life cycle. uint32_t buffer_size; // payload total length of a buffer uint32_t packet_id_base; // seq number of the first packet in this buffer. uint32_t pkt_count; // number of packet in this buffer, 0 means EOF.

};

3. 1-byte-long header definition

Signals such as the four below are 1 byte long, to simplify the receive process(since it cannot be spitted).

#define CLIENT_READY 0x1 #define CLIENT_REQ 0x2 #define CLIENT_DONE 0x4 #define SERVER_SENT 0x8

Note: Please see the collaboration diagram for their meanings.

4. Retransmission Request Header

/* Retransmition Request Header (align to 4 bytes) */ struct request_ctl {

uint32_t req_count; // How many seqs below. uint32_t seqs[0]; // packet seqs.

};

5. Buffer operations

void buffer_init(); // Init the buffer_ctl structure and all(say 1024) packet_ctl structures. Allocate buffer memory. long buffer_fill(int fd); // fill a buffer from fd, such as stdin long buffer_flush(int fd); // flush a buffer to fd, say stdout struct packet_ctl *packet_put(struct packet_ctl *new_pkt);// put a packet to a buffer and return a free memory slot for the next packet. struct packet_ctl *packet_get(uint32_t seq);// get a packet data in buffer by indicating the packet seq.

4.3.2 How to sync between server threads

If children's aaa() operation need to wait the parents's init() to be done, then do it literally like this:

UDP Server TCP Server1 = spawn( )—-> TCP Server1

init() TCP Server2 = spawn( )—-> TCP Server2

V(sem)——————-> P(sem) // No child any more V(sem)——————> P(sem)
aaa() // No need to V(sem), for no child

aaa()

If parent's send() operation need to wait the children's ready() done, then do it literally too, but is a reverse way:

UDP Server TCP Server1 TCP Server2

// No child any more

ready() ready() P(sem) <—————— V(sem)

P(sem) <————— V(sem) send()

Note that the aaa() and ready() operations above run in parallel. If this is not the case due to race condition, the sequence above can be modified into this below:

UDP Server TCP Server1 TCP Server2

// No child any more ready()

P(sem) <—————— V(sem) ready()

P(sem) <————— V(sem) send()

In order to implement such chained/zipper sync pattern, a pair of semaphores is needed between the parent and the child. One is used by child to wait parent , the other is used by parent to wait child. semaphore pair can be allocated by parent and pass the pointer to the child over spawn() operation such as pthread_create().

/* semaphore pair definition */ struct semaphores {

sem_t wait_parent; sem_t wait_child;

};

Then the semaphore pair can be recorded by threads by using the semlink struct below: struct semlink {

struct semaphores *this;* / used by parent to point to the struct semaphores which it created during spawn child. */

struct semaphores *parent;* / used by child to point to the struct semaphores which it created by parent */

};

chained/zipper sync API:

void sl_wait_child(struct semlink *sl); void sl_release_child(struct semlink *sl); void sl_wait_parent(struct semlink *sl); void sl_release_parent(struct semlink *sl);

API usage is like this.

Thread1(root parent) Thread2(child) Thread3(grandchild) sl_wait_parent(noop op) sl_release_child

+————->sl_wait_parent

sl_release_child

+————> sl_wait_parent

sl_release_child(noop op) . . . sl_wait_child(noop op)

- sl_release_parent

sl_wait_child <————-

- sl_release_parent

sl_wait_child <————— sl_release_parent(noop op)

API implementation:

void sl_wait_child(struct semlink *sl) {

if (sl->this) { P(sl->this->wait_child);

}

}

void sl_release_child(struct semlink *sl) {

if (sl->this) { V(sl->this->wait_parent);

}

}

void sl_wait_parent(struct semlink *sl) {

if (sl->parent) { P(sl->parent->wait_parent);

}

}

void sl_release_parent(struct semlink *sl) {

if (sl->parent) { V(sl->parent->wait_child);

}

}

4.3.3 Client flow chart

See Collaboration Diagram

4.3.4 UDP thread flow chart

See Collaboration Diagram

4.3.5 TCP thread flow chart

S_INIT — (UDP initialized) —> S_ACCEPT — (accept clients) –+

/————————————————————————-/ V

S_PREP — (UDP prepared abuffer) ^ | | –> S_SYNC — (clients ClIENT_READY) | | | –> S_SEND — (clients CLIENT_DONE) | | | V ——————(bufferctl.pkt_count != 0)————————————+

V

exit() <— (bufferctl.pkt_count == 0)

4.3.6 TCP using poll and message queue

TCP uses poll() to sync with client's events as well as output event from itself, so that we can use non-block socket operations to reduce the latency. POLLIN means there are message from client and POLLOUT means we are ready to send message/retransmission packets to client.

poll main loop pseudo code: void check_clients(struct server_status_data *sdata) {

poll_events = poll(&(sdata->ds[1]), sdata->ccount - 1, timeout);

/* check all connected clients */ for (sdata->cindex = 1; sdata->cindex < sdata->ccount; sdata->cindex++)
{

ds = &(sdata->ds[sdata->cindex]); if (!ds->revents) {

continue;

}

if (ds->revents & (POLLERR|POLLHUP|POLLNVAL)) { handle_error_event(sdata);

} else if (ds->revents & (POLLIN|POLLPRI)) {

handle_pullin_event(sdata); // may set POLLOUT into ds->events // to trigger handle_pullout_event().

} else if (ds->revents & POLLOUT) { handle_pullout_event(sdata);

}

}

}

For TCP, since the message from client may not complete and send data may be also interrupted due to non-block fashion, there should be one send message queue and a receive message queue on the server side for each client (client do not use non-block operations).

TCP message queue definition:

struct tcpq { struct qmsg *head, *tail; long count; /* message count in a queue */ long size; /* Total data size of a queue */

};

TCP message queue item definition:

struct qmsg { struct qmsg *next; void *data; long size;

};

TCP message queue API:

// Allocate and init a queue. struct tcpq * tcpq_queue_init(void);

// Free a queue. void tcpq_queue_free(struct tcpq *q);

// Return queue length. long tcpq_queue_dsize(struct tcpq *q);

// queue new message to tail. void tcpq_queue_tail(struct tcpq *q, void *data, long size);

// queue message that cannot be sent currently back to queue head. void tcpq_queue_head(struct tcpq *q, void *data, long size);

// get one piece from queue head. void * tcpq_dequeue_head(struct tcpq *q, long *size);

// Serialize all pieces of a queue, and move it out of queue, to ease the further //operation on it. void * tcpq_dqueue_flat(struct tcpq *q, long *size);

// Serialize all pieces of a queue, do not move it out of queue, to ease the further //operation on it. void * tcpq_queue_flat_peek(struct tcpq *q, long *size);