OpenStack Virtual Baremetal

Release 0.0.1

Mar 11, 2021

Contents

OpenStack Virtual Baremetal is a tool for using OpenStack instances to test baremetal-style deployments.

CHAPTER 1

Table of Contents

1.1 Introduction

OpenStack Virtual Baremetal is a way to use OpenStack instances to do simulated baremetal deployments. This project is a collection of tools and documentation that make it much easier to do so. It primarily consists of the following pieces:

- Patches and documentation for setting up a host cloud.
- A deployment CLI that leverages the OpenStack Heat project to deploy the VMs, networks, and other resources needed.
- An OpenStack BMC that can be used to control OpenStack instances via IPMI commands.
- A tool to collect details from the "baremetal" VMs so they can be added as nodes in the OpenStack Ironic baremetal deployment project.

A basic OVB environment is just a BMC VM configured to control a number of "baremetal" VMs. This allows them to be treated largely the same way a real baremetal system with a BMC would. A number of additional features can also be enabled to add more to the environment.

OVB was initially conceived as an improved method to deploy environments for OpenStack TripleO development and testing. As such, much of the terminology is specific to TripleO. However, it should be possible to use it for any non-TripleO scenarios where a baremetal-style deployment is desired.

1.1.1 Benefits and Drawbacks

As noted above, OVB started as part of the OpenStack TripleO project. Previous methods for deploying virtual environments for TripleO focused on setting up all the vms for a given environment on a single box. This had a number of drawbacks:

- Each developer needed to have their own system. Sharing was possible, but more complex and generally not done. Multi-tenancy is a basic design tenet of OpenStack so this is not a problem when using it to provision the VMs. A large number of developers can make use of a much smaller number of physical systems.

- If a deployment called for more VMs than could fit on a single system, it was a complex manual process to scale out to multiple systems. An OVB environment is only limited by the number of instances the host cloud can support.
- Pre-OVB test environments were generally static because there was not an API for dynamic provisioning. By using the OpenStack API to create all of the resources, test environments can be easily tailored to their intended use case.

One drawback to OVB at this time is that it does have hard requirements on a few OpenStack features (Heat, Neutron port-security, private image uploads, for example) that are not all widely available in public clouds. Fortunately, as they move to newer and newer versions of OpenStack that situation should improve.

It should also be noted that without the Nova PXE boot patch, OVB is not compatible with any workflows that write to the root disk before deployment. This includes Ironic node cleaning.

1.2 Host Cloud Setup

Instructions for setting up the host cloud[1].

1: The host cloud is any OpenStack cloud providing the necessary functionality to run OVB. The host cloud must be running on real baremetal.

1.2.1 Patching the Host Cloud

Note: Patching the host cloud is now optional. On clouds where the Neutron port-security extension is enabled, it is now possible to run without patching. However, the PXE boot patch may provide a better user experience with OVB, so patching may still be desirable.

The changes described in this section apply to compute nodes in the host cloud.

Apply the Nova pxe boot patch file in the `patches` directory to the host cloud Nova. `nova-pxe-boot.patch` can be used with all releases prior to Pike, `nova-pxe-boot-pike.patch` must be used with Pike and later.

Examples:

TripleO/RDO:

```
sudo patch -p1 -d /usr/lib/python2.7/site-packages < patches/nova/nova-pxe-boot.patch
```

or

```
sudo patch -p1 -d /usr/lib/python2.7/site-packages < patches/nova/nova-pxe-boot-pike.
↪patch
```

Devstack:

Note: You probably don't want to try to run this with devstack anymore. Devstack no longer supports rejoining an existing stack, so if you have to reboot your host cloud you will have to rebuild from scratch.

Note: The patch may not apply cleanly against master Nova code. If/when that happens, the patch will need to be applied manually.

```
cp patches/nova/nova-pxe-boot.patch /opt/stack/nova
cd /opt/stack/nova
patch -p1 < nova-pxe-boot.patch
```

or

```
cp patches/nova/nova-pxe-boot-pike.patch /opt/stack/nova
cd /opt/stack/nova
patch -p1 < nova-pxe-boot-pike.patch
```

1.2.2 Configuring the Host Cloud

Some of the configuration recommended below is optional, but applying all of it will provide the optimal experience.

The changes described in this document apply to compute nodes in the host cloud.

1. The Nova option `force_config_drive` must _not_ be set. If you have to change this option, restart `nova-compute` to apply it.
2. Ideally, jumbo frames should be enabled on the host cloud. This avoids MTU problems when deploying to instances over tunneled Neutron networks with VXLAN or GRE.

   For TripleO-based host clouds, this can be done by setting `mtu` on all interfaces and vlans in the network isolation nic-configs. A value of at least 1550 should be sufficient to avoid problems.

   If this cannot be done (perhaps because you don't have access to make such a change on the host cloud), it will likely be necessary to configure a smaller MTU on the deployed virtual instances. Details on doing so can be found on the *Using a Deployed OVB Environment* page.

1.2.3 Preparing the Host Cloud Environment

1. Build or download an ipxe-boot image for the baremetal instances.
   1. To download a pre-built image:

      ```
      wget https://repos.fedorapeople.org/repos/openstack-m/ovb/ipxe-boot.qcow2
      ```

   2. To build the image, run the following from the root of the OVB repo:

      ```
      make -C ipxe
      ```

      To install the required build dependencies on a Fedora system:

      ```
      sudo dnf install -y make gcc perl xz-devel genisoimage qemu-img
      ```

2. Source an rc file that will provide admin credentials for the host cloud.
3. Upload an ipxe-boot image for the baremetal instances:

   ```
   glance image-create --name ipxe-boot --disk-format qcow2 --property os_shutdown_
   ↪timeout=5 --container-format bare < ipxe/ipxe-boot.qcow2
   ```

Note: The path provided to ipxe-boot.qcow2 is relative to the root of the OVB repo. If the command is run from a different working directory, the path will need to be adjusted accordingly.

Note: os_shutdown_timeout=5 is to avoid server shutdown delays since since these servers won't respond to graceful shutdown requests.

Note: On a UEFI enabled openstack cloud, to boot the baremetal instances with uefi (instead of the default bios firmware) the image should be created with the parameters –property="hw_firmware_type=uefi".

4. Upload a CentOS 7 image for use as the base image:

```
wget http://cloud.centos.org/centos/7/images/CentOS-7-x86_64-GenericCloud.qcow2

glance image-create --name CentOS-7-x86_64-GenericCloud --disk-format qcow2 --
↪container-format bare < CentOS-7-x86_64-GenericCloud.qcow2
```

5. (Optional) Create a pre-populated base BMC image. This is a CentOS 7 image with the required packages for the BMC pre-installed. This eliminates one potential point of failure during the deployment of an OVB environment because the BMC will not require any external network resources:

```
wget https://repos.fedorapeople.org/repos/openstack-m/ovb/bmc-base.qcow2

glance image-create --name bmc-base --disk-format qcow2 --container-format bare <␣
↪bmc-base.qcow2
```

   To use this image, configure `bmc_image` in env.yaml to be `bmc-base` instead of the generic CentOS 7 image.

6. Create recommended flavors:

```
nova flavor-create baremetal auto 8192 50 2
nova flavor-create bmc auto 512 20 1
```

   These flavors can be customized if desired. For large environments with many baremetal instances it may be wise to give the bmc flavor more memory. A 512 MB BMC will run out of memory around 20 baremetal instances.

7. Source an rc file that will provide user credentials for the host cloud.

8. Add a Nova keypair to be injected into instances:

```
nova keypair-add --pub-key ~/.ssh/id_rsa.pub default
```

9. (Optional) Configure quotas. When running in a dedicated OVB cloud, it may be helpful to set some quotas to very large/unlimited values to avoid running out of quota when deploying multiple or large environments:

```
neutron quota-update --security_group 1000
neutron quota-update --port -1
neutron quota-update --network -1
neutron quota-update --subnet -1
nova quota-update --instances -1 --cores -1 --ram -1 [tenant uuid]
```

1.3 Deploying the Heat stack

There are two options for deploying the Heat stack.

1.3.1 Deploying with QuintupleO

QuintupleO is short for OpenStack on OpenStack on OpenStack. It was the original name for OVB, and has been repurposed to indicate that this deployment method is able to deploy a full TripleO development environment in one command. It should be useful for non-TripleO users of OVB as well, however.

1. Copy the example env file and edit it to reflect the host environment:

```
cp environments/base.yaml env.yaml
vi env.yaml
```

2. Deploy a QuintupleO stack. The example command includes a number of environment files intended to simplify the deployment process or make it compatible with a broader set of host clouds. However, these environments are not necessary in every situation and may not even work with some older clouds. See below for details on customizing an OVB deployment for your particular situation:

```
bin/deploy.py --quintupleo -e env.yaml -e environments/all-networks.yaml -e␣
↪environments/create-private-network.yaml
```

Note: There is a quintupleo-specific option `--id` in deploy.py. It appends the value passed in to the name of all resources in the stack. For example, if `undercloud_name` is set to 'undercloud' and `--id foo` is passed to deploy.py, the resulting undercloud VM will be named 'undercloud-foo'. It is recommended that this be used any time multiple environments are being deployed in the same cloud/tenant to avoid name collisions.

Be aware that when `--id` is used, a new environment file will be generated that reflects the new names. The name of the new file will be `env-${id}.yaml`. This new file should be passed to build-nodes-json instead of the original.

Note: See *Advanced Options* for other ways to customize an OVB deployment.

---

3. Wait for Heat stack to complete. To make this easier, the `--poll` option can be passed to `deploy.py`.

Note: The BMC instance does post-deployment configuration that can take a while to complete, so the Heat stack completing does not necessarily mean the environment is entirely ready for use. To determine whether the BMC is finished starting up, run `nova console-log bmc`. The BMC service outputs a message like "Managing instance [uuid]" when it is fully configured. There should be one of these messages for each baremetal instance.

```
heat stack-show quintupleo
```

4. Build a nodes.json file that can be imported into Ironic:

```
bin/build-nodes-json
scp nodes.json centos@[undercloud floating ip]:~/instackenv.json
```

Note: Only the base environment file needs to be passed to this command. Additional option environments that may have been passed to the deploy command should *not* be included here.

Note: If `--id` was used to deploy the stack, make sure to pass the generated `env-${id}.yaml` file to

build-nodes-json using the `--env` parameter. Example:

```
bin/build-nodes-json --env env-foo.yaml
```

Note: If roles were used for the deployment, separate node files named `nodes-<profile>.json` will also be output that list only the nodes for that particular profile. Nodes with no profile specified will go in `nodes-no-profile.json`. The base `nodes.json` will still contain all of the nodes in the deployment, regardless of profile.

Note: `build-nodes-json` also outputs a file named `bmc_bm_pairs` that lists which BMC address corresponds to a given baremetal instance.

Deleting a QuintupleO Environment

All of the OpenStack resources created by OVB are part of the Heat stack, so to delete the environment just delete the Heat stack. There are a few local files that may also have been created as part of the deployment, such as ID environment files, nodes.json files, and bmc_bm_pairs. Once the stack is deleted these can be removed safely as well.

Advanced Options

There are also a number of advanced options that can be enabled for a QuintupleO deployment. For each such option there is a sample environment to be passed to the deploy command.

For example, to deploy all networks needed for TripleO network isolation, the following command could be used:

```
bin/deploy.py --quintupleo -e env.yaml -e environments/all-networks.yaml
```

Important: When deploying with multiple environment files, `env.yaml` *must* be explicitly passed to the deploy command. `deploy.py` will only default to using `env.yaml` if no environments are specified.

Some options may have additional configuration parameters. These parameters will be listed in the environment file.

A full list of the environments available can be found at *Sample Environment Index*.

Network Isolation

There are a number of environments related to enabling the network isolation functionality in OVB. These environments are named `all-networks*.yaml` and cause OVB to deploy additional network interfaces on the baremetal instances that allow the use of TripleO's network isolation.

Note: There are templates suitable for doing a TripleO overcloud deployment with network isolation in the `overcloud-templates` directory. See the readme files in those directories for details on how to use them.

The v2 versions of the templates are suitable for use with the TripleO Ocata release and later. The others can be used in Newton and earlier.

Three primary networking layouts are included:

- Basic. This is the default and will only deploy a provisioning interface to the baremetal nodes. It is not suitable for use with network isolation.
- All Networks. This will deploy an interface per isolated network to the baremetal instances. It is suitable for use with any of the overcloud network isolation templates not starting with 'bond'.
- All Networks, Public Bond. This will also deploy an interface per isolated network to the baremetal instances, but it will additionally deploy a second interface for the 'public' network that can be used to test bonding in an OVB environment. The `bond-*` overcloud templates must be used with this type of environment.

QuintupleO and routed networks

TripleO supports deploying OpenStack with nodes on multiple network segments which is connected via L3 routing. OVB can set up a full development environment with routers and DHCP-relay service. This environment is targeted for TripleO development, however it should be useful for non-TripleO users of OVB as well.

1. When deploying QuintupleO with routed networks environment files to enable routed networks must be included, as well as one or more role environment files. See *Enable Routed Networks*, *Configuration for Routed Networks*, and *Base Role Configuration for Routed Networks* in the *Sample Environment Index* for details.
2. Copy the example env file and edit it to reflect the host environment:

```
cp environments/base.yaml env.yaml
vi env.yaml
```

3. Copy the `routed-networks-configuration.yaml` sample environment file and edit it to reflect the host environment:

```
cp environments/routed-networks-configuration.yaml env-routed-networks.yaml
vi env-routed-networks.yaml
```

4. For each desired role, copy the `routed-networks-role.yaml` sample environment file and edit it to reflect the host environment:

```
cp environments/routed-networks-role.yaml env-leaf1.yaml
vi env-leaf1.yaml
```

5. Deploy the QuintupleO routed networks environment by running the deploy.py command. For example:

```
./bin/deploy.py --env env.yaml \
                --quintupleo \
                --env environments/all-networks.yaml \
                --env environments/routed-networks.yaml \
                --env env-routed-networks.yaml \
                --role env-leaf1.yaml
```

6. When generating the `nodes.json` file for TripleO undercloud node import, the environment `env-routed.yaml` should be specified. Also, to include physical network attributes of the node ports in `nodes.json` specify the `--physical_network` option when running `build-nodes-json`. For example:

```
bin/build-nodes-json --physical_network
```

   The following is an example node definition produced when using the `--physical_network` options. Notice that ports are defined with both `address` and `physical_network` attributes.

```
 {
   "pm_password": "password",
   "name": "baremetal-leaf1-0",
   "memory": 8192,
   "pm_addr": "10.0.1.13",
   "ports": [
     {
       "physical_network": "provision2",
       "address": "fa:16:3e:2f:a1:cf"
     }
   ],
   "capabilities": "boot_option:local,profile:leaf1",
   "pm_type": "pxe_ipmitool",
   "disk": 80,
   "arch": "x86_64",
   "cpu": 4,
   "pm_user": "admin"
 }

.. NOTE:: Due to technical debet (backward compatibility) the TripleO
          Undercloud uses ``ctlplane`` as the physical network name for the
          subnet that is local to the Undercloud itself. Either override
          the name of the provision network in the ovb environment by
          setting: ``provision_net: ctlplane`` in the
          ``parameters_defaults`` section or edit the generated nodes.json
          file, replacing:
          ``"physical_network": "<name-used-for-provision_net>"`` with
          ``"physical_network": "ctlplane"``.
```

7. For convenience router addresses are made available via the `network_environment_data` key in the stack output of the quintupleo heat stack. To retrieve this data run the `openstack stack show` command. For example:

```
$ openstack stack show quintupleo -c outputs -f yaml

outputs:
- description: floating ip of the undercloud instance
  output_key: undercloud_host_floating_ip
  output_value: 38.145.35.98
- description: Network environment data, router addresses etc.
  output_key: network_environment_data
  output_value:
    internal2_router: 172.17.1.204
    internal_router_address: 172.17.0.201
    provision2_router: 192.168.25.254
    provision3_router: 192.168.26.254
    provision_router: 192.168.24.254
    storage2_router_address: 172.18.1.254
    storage_mgmt2_router_address: 172.19.1.254
    storage_mgmt_router_address: 172.19.0.254
    storage_router_address: 172.18.0.254
    tenant2_router_address: 172.16.1.254
    tenant_router_address: 172.16.0.254
- description: ip of the undercloud instance on the private network
  output_key: undercloud_host_private_ip
  output_value: 10.0.1.14
```

8. Below is an example TripleO Undercloud configuration (`undercloud.conf`) with routed networks support

enabled and the three provisioning networks defined.

```
[DEFAULT]
enable_routed_networks = true
enable_ui = false
overcloud_domain_name = localdomain
scheduler_max_attempts = 2
undercloud_ntp_servers = pool.ntp.org
undercloud_hostname = undercloud.rdocloud
local_interface = eth1
local_mtu = 1450
local_ip = 192.168.24.1/24
undercloud_public_host = 192.168.24.2
undercloud_admin_host = 192.168.24.3
undercloud_nameservers = 8.8.8.8,8.8.4.4
local_subnet = provision
subnets = provision,provision2,provision3

[provision]
cidr = 192.168.24.0/24
dhcp_start = 192.168.24.10
dhcp_end = 192.168.24.30
gateway = 192.168.24.254
inspection_iprange = 192.168.24.100,192.168.24.120
masquerade = true

[provision2]
cidr = 192.168.25.0/24
dhcp_start = 192.168.25.10
dhcp_end = 192.168.25.30
gateway = 192.168.25.254
inspection_iprange = 192.168.25.100,192.168.25.120
masquerade = true

[provision3]
cidr = 192.168.26.0/24
dhcp_start = 192.168.26.10
dhcp_end = 192.168.26.30
gateway = 192.168.26.254
inspection_iprange = 192.168.26.100,192.168.26.120
masquerade = true
```

1.3.2 Deploying a Standalone Baremetal Stack

The process described here will create a very minimal OVB environment, and the user will be responsible for creating most of the resources manually. In most cases it will be easier to use the *QuintupleO* deployment method, which creates most of the resources needed automatically.

1. Create private network.

   If your cloud provider has already created a private network for your use then you can skip this step and reference the existing network in your OVB environment file.

   ```
   neutron net-create private
   neutron subnet-create --name private private 10.0.1.0/24 --dns-nameserver 8.8.8.8
   ```

   You will also need to create a router so traffic from your private network can get to the external network. The external network should have been created by the cloud provider:

```
neutron router-create router
neutron router-gateway-set router [external network name or id]
neutron router-interface-add router private
```

2. Create provisioning network.

Note: The CIDR used for the subnet does not matter. Standard tenant and external networks are also needed to provide floating ip access to the undercloud and bmc instances

Warning: Do not enable DHCP on this network. Addresses will be assigned by the undercloud Neutron.

```
neutron net-create provision
neutron subnet-create --name provision --no-gateway --disable-dhcp provision 192.
↪168.24.0/24
```

3. Create "public" network.

Note: The CIDR used for the subnet does not matter. This can be used as the network for the public API endpoints on the overcloud, but it does not have to be accessible externally. Only the undercloud VM will need to have access to this network.

Warning: Do not enable DHCP on this network. Doing so may cause conflicts between the host cloud metadata service and the undercloud metadata service. Overcloud nodes will be assigned addresses on this network by the undercloud Neutron.

```
neutron net-create public
neutron subnet-create --name public --no-gateway --disable-dhcp public 10.0.0.0/24
```

4. Copy the example env file and edit it to reflect the host environment:

Note: Some of the parameters in the base environment file are only used for QuintupleO deployments. Their values will be ignored in a plain virtual-baremetal deployment.

```
cp environments/base.yaml env.yaml
vi env.yaml
```

5. Deploy the stack:

```
bin/deploy.py
```

6. Wait for Heat stack to complete:

Note: The BMC instance does post-deployment configuration that can take a while to complete, so the Heat stack completing does not necessarily mean the environment is entirely ready for use. To determine whether the BMC is finished starting up, run `nova console-log bmc`. The BMC service outputs a message like "Man-

aging instance [uuid]" when it is fully configured. There should be one of these messages for each baremetal instance.

```
heat stack-show baremetal
```

7. Boot a VM to serve as the undercloud:

```
nova boot undercloud --flavor m1.xlarge --image centos7 --nic net-id=[tenant net␣
↪uuid] --nic net-id=[provisioning net uuid]
neutron floatingip-create [external net uuid]
neutron port-list
neutron floatingip-associate [floatingip uuid] [undercloud instance port id]
```

8. Turn off port-security on the undercloud provisioning port:

```
neutron port-update [UUID of undercloud port on the provision network] --no-
↪security-groups --port-security-enabled=False
```

9. Build a nodes.json file that can be imported into Ironic:

```
bin/build-nodes-json
scp nodes.json centos@[undercloud floating ip]:~/instackenv.json
```

Note: `build-nodes-json` also outputs a file named `bmc_bm_pairs` that lists which BMC address corresponds to a given baremetal instance.

10. The undercloud vm can now be used with something like TripleO to do a baremetal-style deployment to the virtual baremetal instances deployed previously.

Deleting an OVB Environment

All of the OpenStack resources created by OVB are part of the Heat stack, so to delete the environment just delete the Heat stack. There are a few local files that may also have been created as part of the deployment, such as nodes.json files and bmc_bm_pairs. Once the stack is deleted these can be removed safely as well.

1.3.3 Deploying Heterogeneous Environments

It is possible to deploy an OVB environment with multiple "baremetal" node types. The *QuintupleO* deployment method must be used, so it would be best to start with a working configuration for that before moving on to heterogeneous deployments.

Each node type will be identified as a `role`. A simple QuintupleO deployment can be thought of as a single-role deployment. To deploy multiple roles, additional environment files describing the extra roles are required. These environments are simplified versions of the standard environment file. See `environments/base-role.yaml` for a starting point when writing these role files.

Note: Each extra role consists of exactly one environment file. This means that the standalone option environments cannot be used with roles. To override the options specified for the primary role in a secondary role, the parameter_defaults and resource_registry entries from the option environment must be copied into the role environment.

However, note that most resource_registry entries are filtered out of role environments anyway since they are not relevant for a secondary stack.

Steps for deploying the environment:

1. Customize the environment files. Make sure all environments have a `role` key in the `parameter_defaults` section. When building nodes.json, this role will be automatically assigned to the node, so it is simplest to use one of the default TripleO roles (control, compute, cephstorage, etc.).
2. Deploy with both roles:

```
bin/deploy.py --quintupleo --env env-control.yaml --role env-compute.yaml
```

3. One Heat stack will be created for each role being deployed. Wait for them all to complete before proceeding.

   Note: Be aware that the extra role stacks will be connected to networks in the primary role stack, so the extra stacks must be deleted before the primary one or the neutron subnets will not delete cleanly.

4. Build a nodes.json file that can be imported into Ironic:

```
bin/build-nodes-json --env env-control.yaml
```

   Note: Only the primary environment file needs to be passed here. The resources deployed as part of the secondary roles will be named such that they appear to be part of the primary environment.

   Note: If `--id` was used when deploying, remember to pass the generated environment file to this command instead of the original.

1.3.4 Sample Environment Index

Deploy with All Networks Enabled and Two Public Interfaces

File: environments/all-networks-public-bond.yaml

Description: Deploy an OVB stack that adds interfaces for all the standard TripleO network isolation networks. This version will deploy duplicate public network interfaces on the baremetal instances so that the public network can be configured as a bond.

Deploy with All Networks Enabled

File: environments/all-networks.yaml

Description: Deploy an OVB stack that adds interfaces for all the standard TripleO network isolation networks.

Base Configuration Options for Extra Nodes with All Ports Open

File: environments/base-extra-node-all.yaml

Description: Configuration options that need to be set when deploying an OVB environment with extra undercloud-like nodes. This environment should be used like a role file, but will deploy an undercloud-like node instead of more baremetal nodes.

Base Configuration Options for Extra Nodes

File: environments/base-extra-node.yaml

Description: Configuration options that need to be set when deploying an OVB environment with extra undercloud-like nodes. This environment should be used like a role file, but will deploy an undercloud-like node instead of more baremetal nodes.

Base Configuration Options for Secondary Roles

File: environments/base-role.yaml

Description: Configuration options that need to be set when deploying an OVB environment that has multiple roles.

Base Configuration Options

File: environments/base.yaml

Description: Basic configuration options needed for all OVB environments

Enable Instance Status Caching in BMC

File: environments/bmc-use-cache.yaml

Description: Enable caching of instance status in the BMC. This should reduce load on the host cloud, but at the cost of potential inconsistency if the state of a baremetal instance is changed without using the BMC.

Boot Baremetal Instances from Volume

File: environments/boot-baremetal-from-volume.yaml

Description: Boot the baremetal instances from Cinder volumes instead of ephemeral storage.

Boot Undercloud and Baremetal Instances from Volume

File: environments/boot-from-volume.yaml

Description: Boot the undercloud and baremetal instances from Cinder volumes instead of ephemeral storage.

Boot Undercloud Instance from Volume

File: environments/boot-undercloud-from-volume.yaml

Description: Boot the undercloud instance from a Cinder volume instead of ephemeral storage.

Create a Private Network

File: environments/create-private-network.yaml

Description: Create the private network as part of the OVB stack instead of using an existing one.

Disable BMC

File: environments/disable-bmc.yaml

Description: Deploy a stack without a BMC. This will obviously make it impossible to control the instances via IPMI. It will also prevent use of ovb-build-nodes-json because there will be no BMC addresses.

Configuration for router advertisement daemon (radvd)

File: environments/ipv6-radvd-configuration.yaml

Description: Contains the available parameters that need to be configured when using a IPv6 network. Requires the ipv6-radvd.yaml environment.

Enable router advertisement daemon (radvd)

File: environments/ipv6-radvd.yaml

Description: Deploy the stack with a router advertisement daemon running for the provisioning network.

Public Network External Router

File: environments/public-router.yaml

Description: Deploy a router that connects the public and external networks. This allows the public network to be used as a gateway instead of routing all traffic through the undercloud.

Disable the Undercloud in a QuintupleO Stack

File: environments/quintupleo-no-undercloud.yaml

Description: Deploy a QuintupleO environment, but do not create the undercloud instance.

Configuration for Routed Networks

File: environments/routed-networks-configuration.yaml

Description: Contains the available parameters that need to be configured when using a routed networks environment. Requires the routed-networks.yaml or routed-networks-ipv6.yaml environment.

Enable Routed Networks IPv6

File: environments/routed-networks-ipv6.yaml

Description: Enable use of routed IPv6 networks, where there may be multiple separate networks connected with a router, router advertisement daemon (radvd), and DHCP relay. Do not pass any other network configuration environments after this one or they may override the changes made by this environment. When this environment is in use, the routed-networks-configuration environment should usually be included as well.

Base Role Configuration for Routed Networks

File: environments/routed-networks-role.yaml

Description: A base role environment that contains the necessary parameters for deploying with routed networks.

Enable Routed Networks

File: environments/routed-networks.yaml

Description: Enable use of routed networks, where there may be multiple separate networks connected with a router and DHCP relay. Do not pass any other network configuration environments after this one or they may override the changes made by this environment. When this environment is in use, the routed-networks-configuration environment should usually be included as well.

Assign the Undercloud an Existing Floating IP

File: environments/undercloud-floating-existing.yaml

Description: When deploying the undercloud, assign it an existing floating IP instead of creating a new one.

Do Not Assign a Floating IP to the Undercloud

File: environments/undercloud-floating-none.yaml

Description: When deploying the undercloud, do not assign a floating ip to it.

1.4 Using a Deployed OVB Environment

After an OVB environment has been deployed, there are a few things to know.

1. The undercloud vm can be used with something like TripleO to do a baremetal-style deployment to the virtual baremetal instances deployed previously.
2. To reset the environment, usually it is sufficient to do a `nova rebuild` on the undercloud to return it to the original image. To ensure that no traces of the old environment remain, the baremetal vms can be rebuilt to the ipxe-boot image as well.

Note: If you are relying on the ipxe-boot image to provide PXE boot support in your cloud because Nova does not know how to PXE boot natively, the baremetal instances must always be rebuilt before subsequent deployments.

Note: **Do not** rebuild the bmc. It is unnecessary and not guaranteed to work.

3. If the host cloud's tenant network MTU is 1500 or less, it will be necessary to configure the deployed interfaces with a smaller MTU. The tenant network MTU minus 50 is usually a safe value. For the undercloud this can be done by setting `local_mtu` in `undercloud.conf`.

   Note: In Mitaka and older versions of TripleO it will be necessary to do the MTU configuration manually. That can be done with the following commands (as root):

   ```
   # Replace 'eth1' with the actual device to be used for the
   # provisioning network
   ip link set eth1 mtu 1350
   echo -e "\ndhcp-option-force=26,1350" >> /etc/dnsmasq-ironic.conf
   systemctl restart 'neutron-*'
   ```

4. If using the full network isolation provided by one of the `all-networks*.yaml` environments then a TripleO overcloud can be deployed in the OVB environment by using the network templates in the `overcloud-templates` directory. The names are fairly descriptive, but this is a brief explanation of each:

   - **network-templates:** IPv4 multi-nic. Usable with the network layout deployed by the `all-networks.yaml` environment.
   - **ipv6-network-templates:** IPv6 multi-nic. Usable with the network layout deployed by the `all-networks.yaml` environment.
   - **bond-network-templates:** IPv4 multi-nic, with duplicate *public* interfaces for testing bonded nics. Usable with the network layout deployed by the `all-networks-public-bond.yaml` environment.

   The undercloud's `public` interface should be configured with the address of the default route from the templates in use. Firewall rules for forwarding the traffic from that interface should also be added. The following commands will make the necessary configuration:

   ```
   cat >> /tmp/eth2.cfg <<EOF_CAT
   network_config:
     - type: interface
       name: eth2
       use_dhcp: false
       addresses:
         - ip_netmask: 10.0.0.1/24
         - ip_netmask: 2001:db8:fd00:1000::1/64
   EOF_CAT
   sudo os-net-config -c /tmp/eth2.cfg -v
   sudo iptables -A POSTROUTING -s 10.0.0.0/24 ! -d 10.0.0.0/24 -j MASQUERADE -t nat
   ```

1.5 Troubleshooting

A list of common problems and their solutions.

1.5.1 Nodes hang while downloading the deploy ramdisk or kernel

Cause: Improper MTU settings on deployment interfaces.

Solution: Set the MTU on the deployment interfaces to allow PXE booting to work correctly. For TripleO-based deployments, see the readme for details on how to do this. For others, make sure that the deployment nic on the undercloud vm has the MTU set appropriately and that the DHCP server responding to PXE requests advertises the same MTU. Note that this MTU should be 50 bytes smaller than the physical MTU of the host cloud.

1.5.2 Nodes are deployed, but cannot talk to each other

In OpenStack deployments, this often presents as rabbitmq connectivity issues from compute nodes.

Cause: Improper MTU settings on deployed instances.

Solution: Essentially the same as the previous problem. Ensure that the MTU being used on the deployed instances is 50 bytes smaller than the physical MTU of the host cloud. Again, for TripleO-based deployments the readme has details on how to do this.

1.5.3 Nodes fail to PXE boot

Cause: The nodes are not configured to PXE boot properly.

Solution: This depends on the method being used to PXE boot. If the Nova patch is being used to provide this functionality, then ensure that it has been applied on all compute nodes and those nodes' nova-compute service has been restarted. If the ipxe-boot image is being used without the Nova patch, the baremetal instances must be rebuilt to the ipxe-boot image before subsequent deployments.

1.5.4 Nodes fail to PXE boot 2

DHCP requests are seen on the undercloud VM, but responses never get to the baremetal instances.

Cause: Neutron port security blocking DHCP from the undercloud.

Solution: Ensure that the Neutron port-security extension is present in the host cloud. It is required for OVB to function properly.

1.5.5 The BMC does not respond to IPMI requests

Cause: Several. Neutron may not be configured to allow the BMC to listen on arbitrary addresses. The BMC deployment may have failed for some reason.

Solution: Neutron must be configured to allow the BMC to listen on arbitrary addresses. This requires the port-security extension as in the previous solution. If this is already configured correctly, then the BMC may have failed to deploy properly. This can usually be determined by looking at the nova console-log of the BMC instance. A correctly working BMC will display 'Managing instance [uuid]' for each baremetal node in the environment. If those messages are not found, then the BMC has failed to start properly. The relevant error messages should be found in the console-log of the BMC. If that is not sufficient to troubleshoot the problem, the BMC can be accessed using the ssh key configured in the OVB environment yaml as the 'centos' user.

1.6 Python API

1.6.1 build_nodes_json

1.6.2 deploy

1.6.3 openstackbmc

class openstackbmc.**OpenStackBmc**(*authdata*, *port*, *address*, *instance*, *cache_status*, *os_cloud*)

get_boot_device()
Return the currently configured boot device

get_power_state()
Returns the current power state of the managed instance

log(**msg*)
Helper function that prints msg and flushes stdout

power_off()
Stop the managed instance

power_on()
Start the managed instance

power_reset()
Not implemented

power_shutdown()
Stop the managed instance

set_boot_device(*bootdevice*)
Set the boot device for the managed instance

Parameters **bootdevice** – One of ['network', 'hd] to set the boot device to network or hard disk respectively.

1.6.4 auth

auth.**_cloud_json**()
Return the current cloud's data in JSON

Retrieves the cloud from os-client-config and serializes it to JSON.

auth.**_create_auth_parameters**()
Read keystone auth parameters from appropriate source

If the environment variable OS_CLOUD is set, read the auth information from os_client_config. Otherwise, read it from environment variables. When reading from the environment, also validate that all of the required values are set.

Returns A dict containing the following keys: os_user, os_password, os_tenant, os_auth_url, os_project, os_user_domain, os_project_domain.

CHAPTER 2

Index

- genindex
- modindex

Python Module Index

a

b

Index

Symbols

A

B

G

L

O

P

S