Deploying a DevStack environment

This document describes an example configuration to create a DevStack environment in any cloud provider. We have tried this in GCE, but the procedure and configuration should be aplicable to any cloud provider that allows to configure a minimal networking for the VMs.

Networking layout

We will need to create a VM with 2 NICs: one will be used for management and to access the VM, and the other one will be where the OpenStack floating networks will be bridged.

In order to avoid network collision and to be able to properly configure the routes in the VM, we will use the following networks:

Networks to be manually created in the cloud provider:

- 192.168.0.0/24 - Management. NIC 1 of the VM
- 192.168.1.0/24 - Not really used, but in order to create NICs in most providers you need to attach them to a network. NIC 2 of the VM.

Networks that will exist virtualized in OpenStack (no need to manually create them):

- 172.16.0.0/16 - Floating IP network range. These will be the "public IPs" for the VMs we deploy in OpenStack.
- 10.0.0.0/8 - When creating private networks in OpenStack we will use networks inside this range. This way we avoid collisions with the floating and provider networks and make routing easier.

Create the VM

Create a virtual machine with the following recommended configuration:

- Ubuntu 16.04
- 4 CPU / 15GB RAM / 50GB Disk
- 2 NICs:
  - First NIC attached to the management network (192.168.0.0/24) and with an elastic IP associated.
  - Second NIC attached to the "floating" network (192.168.1.0/24). No need for an elastic IP here.
  - No firewall. We not only need access to the web console, but to all OpenStack services (Nova, Neutron, Keystone, etc), so for simplicity we'll leave the VM without a firewall.

Install DevStack

1. Create the DevStack user and clone their Git repository as explained here: https://docs.openstack.org/devstack/latest/
2. Once the stack is created and the repo cloned, make sure you are logged in as the *stack* user in the VM. Then copy the attached **local.conf** file to */opt/stack/devstack/* and modify it as follows:
   a. Modify the *ADMIN_PASSWORD* to a secure value.
   b. Modify the value of the *HOST_IP* variable to the "private ip" assigned to the VM's NIC 1.
   c. Modify the value of the *PUBLIC_ENDPOINT_IP* variable to the "public ip" assigned to the instance.
   d. Modify the value of the *PUBLIC_INTERFACE* variable to the name of the interface corresponding to the VM's NIC 2 (that's the interface where the floating ip networks will be bridged).
   e. (Optional) Modify the credentials or any other field to match your deployment and addressing, if you configured a different network layout.
3. Execute the stack script and wait for it to complete: *./stack.sh*
4. Once completed, log in to the web console as 'admin', go to 'API Access', and download the OpenStack RC file.
5. Load the downloaded file: *source admin-openrc.sh*
6. Use the *openstack endpoint list* and *openstack endpoint set* command to fix the endpoints that the DevStack script did not configure properly:

   a. Make sure all endpoints are using the public ip. Fix the wrong ones
   b. Make sure the Keystone endpoints end with: */v3*

Access to the OpenStack Floating IP range

When deploying a DevStack in a public cloud, we don't have control over a "public IP range". This means that we don't have a reachable range we can configure as the floating IP range in OpenStack, so we'll have to use a private one (in our example 172.16.0.0/16).

With the current setup we can access that floating ip range from the OpenStack host, so if we deploy a guest and assign a floating IP to it, we should be able to SSH the OpenStack host, and from there SSH the guest VM via its floating IP.

However, having an unreachable floating IP range, may cause issues with jclouds-compute, as it will try to access the nodes through their floating IPs.

Configuring SSH tunnels to access the floating IP range

To bypass that, we can configure an SSH L2 tunnel from our local machine to the OpenStack host, and route all traffic to the floating IP range through the tunnel, making the range reachable. To do that we'll need to:

1. Enable tunneling in the SSH server:
    a. Add the following to the ***/etc/ssh/sshd_config*** in the OpenStack host: ***PermitTunnel yes***
    b. Restart the SSH server.
2. Add the SSH public key fo your local machine to the OpenStack host 'root' account. To create the tunnels, SSH will create a 'tun' interface in the host, and it needs to be accessed as root. Make sure the SSH configuration allows root login using SSH keys.
3. On your local machine, create the tunnel by running:

    ```
    sudo ssh -N -f -i <path to the private key> -w 1:1 -o Tunnel=ethernet root@<openstack host public ip>
    ```

    This will create a 'tun1' interface in your local machine and in the OpenStack host.
4. Configure the tunnel and routing. The easiest way to configure routing is to assign an IP address to each 'tun1' interface. It is important to pick an unused network range, to avoid messig up the routing. In our example we will use addresses 192.168.10.1 and 192.168.10.2, as they are from an unused range.

    In the OpenStack host:

```
sudo ip link set tun1 up
sudo ip addr add 192.168.10.1/30 dev tun1
```

In the local machine:

```
sudo ip link set tun1 up
sudo ip addr add 192.168.10.2/30 dev tun1
sudo ip route add 172.16.0.0/16 dev tun1
```

With this configuration you now have an SSH tunnel to the OpenStack host and all traffic to the floating ip network routed through the tunnel.

You can now directly access the VMs deployed in OpenStack directly using their floating ip!

Once you are done, you can remove the tunnel by simply killing the SSH process in the local machine.