Stackube Documentation

Release

Stackube development team

Oct 29, 2017

Contents

Stackube is a Kubernetes-centric OpenStack distro. It uses Kubernetes, instead of Nova, as the compute fabric controller, to provision containers as the compute instance, along with other OpenStack services (e.g. Cinder, Neutron). It supports multiple container runtime technologies, e.g. Docker, Hyper, and offers built-in soft / hard multi-tenancy (depending on the container runtime used).

CHAPTER 1

Stackube Authors

Stackube is an open source project with an active development community. The project is initiated by HyperHQ, and involves contribution from ARM, China Mobile, etc.

CHAPTER 2

Introduction

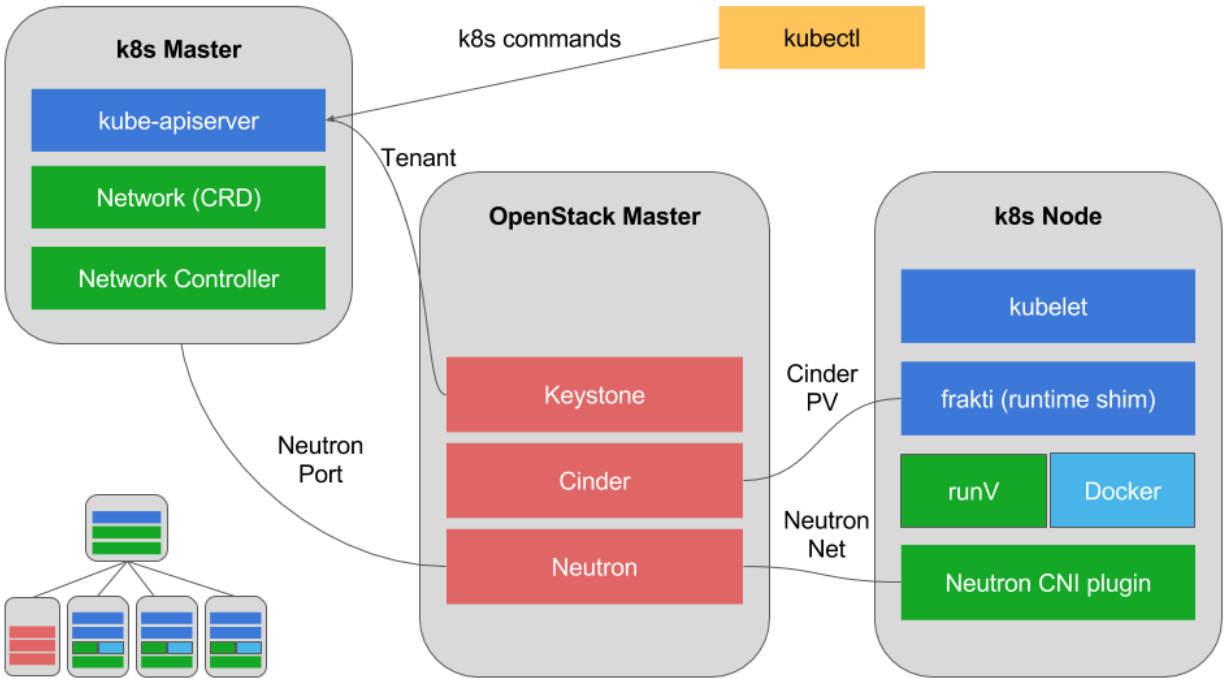
2.1 Stackube Architecture

This page describes the architecture of stackube.

2.1.1 Overview

Stackube is a Kubernetes-centric OpenStack distro. It uses Kubernetes, instead of Nova, as the compute fabric controller, to provision containers as the compute instance, along with other OpenStack services (e.g. Cinder, Neutron). It supports multiple container runtime technologies, e.g. Docker, Hyper, and offers built-in soft/hard multi-tenancy (depending on the container runtime used).

2.1.2 Architecture



2.1.3 Components

1. Standard OpenStack Compoents
   - OpenStack Keytone
   - OpenStack Neutron
   - OpenStack Cinder
2. Standard Kubernetes Components
   - Etcd: the storage of kubernetes.
   - Kube-apiserver: the API, authn/authz and admission control of kubernetes.
   - Kuber-controller-manager: the brain of kubernetes, ensure the state of kubernetes cluster.
   - Kube-scheduler: the scheduler of kubernetes.
   - Kubelet: the container lifecycle and volume manager of kubernetes.
   - Frakti&HyperContainer: a hypervisor-based container runtime.
   - Docker: docker container runtime.
3. Stackube addons
   - Stackube-controller: tenant and network manager.
   - Stackube-proxy: service discovery and load balancing, replacement of kube-proxy.
   - Kubestack: the CNI network plugin, which connects containers to Neutron network.

2.2 Stackube Scope

A multi-tenant and secure Kubernetes deployment enabled by OpenStack core components.

2.2.1 Not another "Kubernetes on OpenStack" project

Stackube is a standard upstream Kubernetes deployment with:

1. Mixed container runtime of Docker (Linux container) and HyperContainer (hypervisor-based container)
2. Keystone for tenant management
3. Neutron for container network
4. Cinder for persistent volume

The main difference between Stackube with existing container service project in OpenStack foundation (e.g. Magnum) is: **Stackube works alongside OpenStack, not on OpenStack**.

This means:

1. Only standalone vanilla OpenStack components are required
2. Traditional VMs are not required because HyperContainer will provide hypervisor level isolation for containerized workloads.
3. All the components mentioned above are managed by Kubernetes plugin API.

2.2.2 What's inside Stackube repo?

1. Keystone RBAC plugin
2. Neutron CNI plugin
    - With a k8s Network object controller
3. Standard k8s upstream Cinder plugin with block device mode
4. Deployment scripts and guide
5. Other documentations

Please note:

1. Plugins above will be deployed as system Pod and DaemonSet.
2. All other Kubernetes volumes are also supported in Stackube, while k8s Cinder plugin with block device mode can provide better performance in mixed runtime which will be preferred by default.

2.2.3 What's the difference between other plugin projects?

1. Kuryr
    - This is a Neutron network plugin for Docker network model, which is not directly supported in Kubernetes. Kuryr can provide CNI interface, but Stackube also requires tenant aware network management which is not included in Kuryr. We will evaluate and propose our multi-tenant model to kuryr-kubernetes as a long term effort, then we can move to use kuryr-kubernetes as the default network plugin.
2. Fuxi

- This is a Cinder volume plugin for Docker volume model, which is not supported in latest CRI based Kubernetes (using k8s volume plugin for now, and soon CSI). Also, Stackube prefers a "block-device to Pod" mode in volume plugin when HyperContainer runtime is enabled, which is not supported in Fuxi.

3. K8s-cloud-provider
   - This is a "Kubernetes on OpenStack" integration which requires full functioning OpenStack deployment.
4. Zun
   - This is a OpenStack API container service, while Stackube exposes well-known Kubernetes API and does not require full OpenStack deployment.

As summary, one distinguishable difference is that plugins in Stackube are designed to enable hard multi-tenancy in Kubernetes as a whole solution, while the other OpenStack plugin projects do not address this and solely focus on just integrating with Kubernetes/Docker as-is. There are many gaps to fill when use them to build a real multi-tenant cloud, for example, how tenants cooperate with networks in k8s.

Another difference is Stackube use mixed container runtimes mode of k8s to enable secure runtime, which is not in scope of existing foundation projects. In fact, all plugins in Stackube should work well for both Docker and HyperContainer.

The architecture of Stackube is fully decoupled and it would be easy for us (and we'd like to) integrate it with any OpenStack-Kubernetes plugin. But right now, we hope to keep everything as simple as possible and focus on the core components.

2.2.4 Deployment workflow

On control nodes

Install standalone Keystone, Neutron, Cinder (ceph rbd). This can be done by any existing tools like devstack, RDO etc.

On other nodes

1. Install neutron L2 agents

   This can be done by any existing tools like devstack, RDO etc.
2. Install Kubernetes
   - Including container runtimes, CRI shims, CNI etc
   - This can be done by any existing tools like kubeadm etc
3. Deploy Stackube

```
kubectl create -f stackube-configmap.yaml
kubectl create -f deployment/stackube-proxy.yaml
kubectl create -f deployment/stackube.yaml
```

This will deploy all Stackube plugins as Pods and DaemonSets to the cluster. You can also deploy all these components in a single node.

After that, users can use Kubernetes API to manage containers with hypervisor isolation, Neutron network, Cinder volume and tenant awareness.

CHAPTER 3

Deployment Guide

3.1 Welcome to Stackube setup documentation!

Stackube is a multi-tenant and secure Kubernetes deployment enabled by OpenStack core components.

3.1.1 Single node devbox

This is a single node devbox

Setting up a single node Stackube

This page describes how to setup a working development environment that can be used in developing stackube on Ubuntu or CentOS. These instructions assume you're already installed git, golang and python on your host.

Getting the code

Grab the code:

```
git clone git://git.openstack.org/openstack/stackube
```

Spawn up Kubernetes and OpenStack

devstack is used to spawn up a kubernetes and openstack environment.

Create stack user:

```
sudo useradd -s /bin/bash -d /opt/stack -m stack
echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
sudo su - stack
```

Grab the devstack:

```
git clone https://git.openstack.org/openstack-dev/devstack -b stable/ocata
cd devstack
```

Create a local.conf:

```
curl -sSL https://raw.githubusercontent.com/openstack/stackube/master/devstack/local.
↪conf.sample -o local.conf
```

Start installation:

```
./stack.sh
```

Setup environment variables for kubectl and openstack client:

```
export KUBECONFIG=/opt/stack/admin.conf
source /opt/stack/devstack/openrc admin admin
```

Setup environment variables for kubectl and openstack client:

```
export KUBECONFIG=/etc/kubernetes/admin.conf
source openrc admin admin
```

3.1.2 Multi nodes deployment

Setting Up A Multi-nodes Stackube (Without HA For Now)

This page describes how to setup a multi-nodes cluster of Stackube.

Prerequisites

Roles

A stackube deployment is comprised by four kinds of nodes: control, network, compute, storage.

- Control
  - The control node is where openstack/kubernetes/ceph's control-plane will run.
  - **At least one and only one node** (for now).
  - Minimum hardware requirements:
    - Two network interfaces
      - One is for public network connection, with a public IP.
      - The other one is for private network connection, with a private IP and MTU >= 1600.
    - 8GB main memory
    - 50GB disk space
- Network
  - The network nodes are where neutron l3/lbaas/dhcp agents will run.
  - At least one node.

  - Minimum hardware requirements:
    - Two network interfaces
      - One is as neutron-external-interface. Public IP is not needed.
      - The other one is for private network connection, with a private IP and MTU >= 1600.
    - 8GB main memory
    - 50GB disk space
- Compute
  - The compute nodes are where your workloads will run.
  - At least one node.
  - Minimum hardware requirements:
    - One network interface
      - For private network connection, with a private IP and MTU >= 1600.
    - 8GB main memory
    - 50GB disk space
- Storage
  - The storage nodes are where ceph-osd(s) will run.
  - At least one node.
  - Minimum hardware requirements:
    - One network interface
      - For private network connection, with a private IP and MTU >= 1600.
    - 8GB main memory
    - 50GB disk space

There is no conflict between any two roles. That means, all of the roles could be deployed on the same node(s).

Host OS

For now only CentOS 7.x is supported.

Public IP Pool

A number of public IPs are needed.

Deploy

All instructions below **must be done on the control node.**

1. SSH To The Control Node, And Become Root

```
sudo su -
```

2. Enable Password-Less SSH

The control node needs to ssh to all nodes when deploying.

- Generate SSH keys on the control node. Leave the passphrase empty:

```
ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
```

- Copy the key to each node (including the control node itself):

```
ssh-copy-id root@NODE_IP
```

3. Clone Stackube Repo

```
git clone https://git.openstack.org/openstack/stackube
```

4. Edit The Config File

```
cd stackube/install
vim config_example
```

5. Do The Deploy

```
bash deploy.sh config_example
```

If failed, please **do remove** (as shown below) before deploy again.

Remove

```
bash remove.sh config_example
```

CHAPTER 4

Developer Guide

4.1 Developer Documentation

This page describes how to setup a working development environment that can be used in developing stackube on Ubuntu or CentOS. These instructions assume you're already installed git, golang and python on your host.

4.1.1 Design Tips

The Stackube project is very simple. The main part of it is a `stackube-controller`, which use Kubernetes `Customized Resource Definition (CRD, previous TPR)` to:

1. Manage tenants based on namespace change in k8s
2. Manage RBAC based on namespace change in k8s
3. Manage networks based on tenants change in k8s

The tenant is a CRD which maps to Keystone tenant, the network is a CRD which maps to Neutron network. We also have a `kubestack` binary which is the CNI plug-in for Neutron.

Also, Stackube has it's own `stackube-proxy` to replace `kube-proxy` because network in Stackube is L2 isolated, so we need a multi-tenant version `kube-proxy` here.

We also replaced `kube-dns` in k8s for the same reason: we need to have a `kube-dns` running in every namespace instead of a global DNS server because namespaces are isolated.

You can see that:

Stackube cluster = upstream Kubernetes + several our own add-ons + standalone OpenStack components.

Please note: Cinder RBD based block device as volume is implemented in https://github.com/kubernetes/frakti, you need to contribute there if you have any idea and build a new stackube/flex-volume Docker image for Stackube to use.

4.1.2 Build

Build binary:

```
make
```

The binary will be placed at:

```
_output/kubestack
_output/stackube-controller
_output/stackube-proxy
```

Build docker images:

```
make docker
```

Three docker images will be built:

```
stackube/stackube-proxy:v1.0beta
stackube/stackube-controller:v1.0beta
stackube/kubestack:v1.0beta
```

4.1.3 (Optional) Configure Stackube

If you deployed Stackube by following official guide, you can skip this part.

But if not, these steps below are needed to make sure your Stackube cluster work.

Please note the following parts suppose you have already deployed an environment of OpenStack and Kubernetes on same baremetal host. And don't forget to setup `--experimental-keystone-url` for `kube-apiserver`, e.g.

```
kube-apiserver --experimental-keystone-url=https://192.168.128.66:5000/v2.0 ...
```

Remove `kube-dns` deployment and `kube-proxy` daemonset if you have already running them.

```
kubectl -n kube-system delete deployment kube-dns
kubectl -n kube-system delete daemonset kube-proxy
```

If you have also configured a CNI network plugin, you should also remove it together with CNI network config.

```
# Remove CNI network components, e.g. deployments or daemonsets first.
# Then remove CNI network config.
rm -f /etc/cni/net.d/*
```

Then create external network in Neutron if there is no one.

```
# Create an external network if there is no one.
# Please replace 10.123.0.x with your own external network
# and remember the id of your created external network
neutron net-create br-ex --router:external=True --shared
neutron subnet-create --ip_version 4 --gateway 10.123.0.1 br-ex 10.123.0.0/16 --
↪allocation-pool start=10.123.0.2,end=10.123.0.200 --name public-subnet
```

And create configure file for Stackube.

```
# Remember to replace them with your own ones.
cat >stackube-configmap.yaml <<EOF
kind: ConfigMap
apiVersion: v1
metadata:
  name: stackube-config
  namespace: kube-system
data:
  auth-url: "https://192.168.128.66/identity_admin/v2.0"
  username: "admin"
  password: "admin"
  tenant-name: "admin"
  region: "RegionOne"
  ext-net-id: "550370a3-4fc2-4494-919d-cae33f5b3de8"
  plugin-name: "ovs"
  integration-bridge: "br-int"
  user-cidr: "10.244.0.0/16"
  user-gateway: "10.244.0.1"
  kubernetes-host: "192.168.0.33"
  kubernetes-port: "6443"
  keyring: "AQBZU5lZ/Z7lEBAAJuC17RYjjqIUANs2QVn7pw=="
EOF
```

Then deploy stackube components:

```
kubectl create -f stackube-configmap.yaml
kubectl create -f deployment/stackube-proxy.yaml
kubectl create -f deployment/stackube.yaml
kubectl create -f deployment/flexvolume/flexvolume-ds.yaml
```

Now, you are ready to try Stackube features.

CHAPTER 5

User Guide

5.1 Stackube User Guide

5.1.1 Tenant and Network Management

In this part, we will introduce tenant management and networking in Stackube. The tenant, which is `1:1` mapped with k8s namespace, is managed by using k8s CRD (previous TPR) to interact with Keystone. And the tenant is also `1:1` mapped with a network automatically, which is also implemented by CRD with standalone Neutron.

1. Create a new tenant

```
$ cat test-tenant.yaml

apiVersion: "stackube.kubernetes.io/v1"
kind: Tenant
metadata:
  name: test
spec:
  username: "test"
  password: "password"

$ kubectl create -f test-tenant.yaml
```

2. Check the auto-created namespace and network. Wait a while, the namespace and network for this tenant should be created automatically:

```
$ kubectl get namespace test
NAME      STATUS    AGE
test      Active    58m

$ kubectl -n test get network test -o yaml
apiVersion: stackube.kubernetes.io/v1
kind: Network
metadata:
```

```
  clusterName: ""
  creationTimestamp: 2017-08-03T11:58:31Z
  generation: 0
  name: test
  namespace: test
  resourceVersion: "3992023"
  selfLink: /apis/stackube.kubernetes.io/v1/namespaces/test/networks/test
  uid: 11d452eb-7843-11e7-8319-68b599b7918c
spec:
  cidr: 10.244.0.0/16
  gateway: 10.244.0.1
  networkID: ""
status:
  state: Active
```

3. Check the Network and Tenant created in Neutron by Stackube controller.

```
$ source ~/keystonerc_admin
$ neutron net-list
+--------------------------------------+----------------------+------------------------
↪-----------+--------------------------------------------------------------+
| id                                   | name                 | tenant_id              ⌴
↪           | subnets                                                      |
+--------------------------------------+----------------------+------------------------
↪-----------+--------------------------------------------------------------+
| 421d913a-a269-408a-9765-2360e202ad5b | kube-test-test       |⌴
↪915b36add7e34018b7241ab63a193530 | bb446a53-de4d-4546-81fc-8736a9a88e3a 10.244.0.0/
↪16       |
```

4. Check the `kube-dns` pods created in the new namespace.

```
# kubectl -n test get pods
NAME                        READY     STATUS    RESTARTS   AGE
kube-dns-1476438210-37jv7   3/3       Running   0          1h
```

5. Create pods and services in the new namespace.

```
# kubectl -n test run nginx --image=nginx
deployment "nginx" created
# kubectl -n test expose deployment nginx --port=80
service "nginx" exposed

# kubectl -n test get pods -o wide
NAME                        READY     STATUS    RESTARTS   AGE       IP           ⌴
↪NODE
kube-dns-1476438210-37jv7   3/3       Running   0          1h        10.244.0.4   ⌴
↪stackube
nginx-4217019353-6gjxq      1/1       Running   0          27s       10.244.0.10  ⌴
↪stackube

# kubectl -n test run -i -t busybox --image=busybox sh
If you don't see a command prompt, try pressing enter.
/ # nslookup nginx
Server:    10.96.0.10
Address 1: 10.96.0.10

Name:      nginx
Address 1: 10.108.57.129 nginx.test.svc.cluster.local
```

```
/ # wget -O- nginx
Connecting to nginx (10.108.57.129:80)
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
-                    100%␣
↪|*******************************************************************************|   612  ␣
↪0:00:00 ETA
/ #
```

6. Finally, remove the tenant.

```
$ kubectl delete tenant test
tenant "test" deleted
```

7. Check Network in Neutron is also deleted by Stackube controller

```
$ neutron net-list
+--------------------------------------+---------+----------------------------------+-
↪-----------------------------------------------------------+
| id                                   | name    | tenant_id                        |␣
↪subnets                                                    |
+--------------------------------------+---------+----------------------------------+-
↪-----------------------------------------------------------+
```

5.1.2 Persistent volume

This part describes the persistent volume design and usage in Stackube.

5.1.3 Standard Kubernetes volume

Stackube is a standard upstream Kubernetes cluster, so any type of Kubernetes volumes. can be used here, for example:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    # FIXME: use the right IP
    server: 10.244.1.4
    path: "/exports"
```

Please note since Stackube is a baremetal k8s cluster, cloud provider based volume are not supported by default.

But unless you are using `emptyDir` or `hostPath`, we will recommend always you the `Cinder RBD based block device as volume` described below in Stackube, this will bring you much higher performance.

5.1.4 Cinder RBD based block device as volume

The reason this volume type is preferred is: by default Stackube will run most of your workloads in a VM-based Pod, in this case directory sharing is used by hypervisor based runtime for volumes mounts, but this actually has more I/O overhead than bind mount.

On the other hand, the hypervisor Pod make it possible to mount block device directly to the VM-based Pod, so we can eliminates directory sharing.

In Stackube, we use a flexvolume to directly use Cinder RBD based block device as Pod volume. The usage is very simple:

1. Create a Cinder volume (skip if you want to use a existing Cinder volume).

```
$ cinder create --name volume 1
```

2. Create a Pod claim to use this Cinder volume.

```
apiVersion: v1
kind: Pod
metadata:
  name: web
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    volumeMounts:
    - name: nginx-persistent-storage
      mountPath: /var/lib/nginx
  volumes:
  - name: nginx-persistent-storage
    flexVolume:
      driver: "cinder/flexvolume_driver"
      fsType: ext4
```

```
    options:
      volumeID: daa7b4e6-1792-462d-ad47-78e900fed429
```

Please note the name of flexvolume should be: `cinder/flexvolume_driver`.

The `daa7b4e6-1792-462d-ad47-78e900fed429` is either volume ID created with Cinder or any existing available Cinder volume ID. After this yaml is applied, the related RBD device will be attached to the VM-based Pod after this is created.

5.1.5 Others

If your cluster is installed by `stackube/devstack` or following other stackube official guide, a `/etc/kubernetes/cinder.conf` file will be generated automatically on every node. Otherwise, you are expected to create a `/etc/kubernetes/cinder.conf` on every node. The contents is like:

```
[Global]
auth-url = _AUTH_URL_
username = _USERNAME_
password = _PASSWORD_
tenant-name = _TENANT_NAME_
region = _REGION_
[RBD]
keyring = _KEYRING_
```

and also, users need to make sure flexvolume_driver binary is in `/usr/libexec/kubernetes/kubelet-plugins/volume/exec/cinder~flexvolume_driver/` of every node.

CHAPTER 6

Release Note

6.1 Stackube Release Note

6.1.1 1.0 Beta

6.1.2 Prelude

This is the 1.0 Beta release of Stackube: a secure, multi-tenant and Kubernetes centric OpenStack distribution.

6.1.3 New Features

1. Implemented a auth controller watches on tenant/networks/namespaces change and setups RBAC roles for tenant. This is how we match Kubernetes authorization control with OpenStack tenant by: tenant namespace 1:1 mapping. Controller is implemented by using CRD of Kubernetes.

2. Implemented a network controller watches on networks/namespaces change and operates OpenStack network based on network namespace 1:1 mapping. This is how to define Kubernetes multi-tenant network by using Neutron. Controller is implemented by using CRD of Kubernetes.

3. Implemented a CNI plug-in which name is kubestack. This is a CNI plug-in for Neutron and work with the network controller model mentioned above. When network of Neutron is ready, kubestack will be responsible to configure Pods to use this network by following standard CNI workflow.

4. Implemented stackube-proxy to replace default kube-proxy in Kubernetes so that proxy will be aware of multi-tenant network.

5. Implemented stackube DNS add-on to replace default kube-dns in Kubernetes so that DNS server will be aware of multi-tenant network.

6. Integrated cinder-flexvolume of kubernetes/frakti project so that hypervisor based container runtime (runV) in Stackube can mount Cinder volume (RBD) directly to VM-based Pod. This will bring better performance to user.

7. Improved stackube-proxy so that Neutron LBaaS based service can be used in Stackube when LoadBalancer type service is created in Kubernetes.

8. Created Docker images for all add-on and plug-in above so they can be deployed in Stackube by one command.

9. Add deployment documentation for Stackube which will install Kubernetes + mixed container runtime + CNI + volume plugin + standalone OpenStack components.

6.1.4 Known Issues

None

6.1.5 Upgrade Notes

None

6.1.6 Deprecation Notes

None

6.1.7 Bug Fixes

1. Fixed CNI network namespace is not cleaned properly

6.1.8 Other Notes

This is the 1.0 Beta release of Stackube, reporting bugs in https:// bugs.launchpad.net/stackube during you exploration is highly appreciated!