Openstack、Docker和Kubernetes是当前IaaS和PaaS领域最为火热的技术。随着Docker容器技术的兴起，容器编排调度引擎成为当前十分抢眼的技术，这其中尤以Google开源的Kubernetes(K8S)以Production-ready和各种完善的功能而最为人熟值。在容器及其编排引擎的推动下，Openstack的主流部署模式也在发生变化，目前的Openstack与Docker容器和Kubernetes已经形成了一种你中有我，我中有你的局面，本文主要探讨如何借助Openstack的Kolla-kubernets项目，通过Kubernetes对Openstack进行容器化部署，并最终部署一套ALL-IN-ONE类型的Openstack容器云。

一、Kubernetes集群部署主机及系统软件需求

■ 主机配置需求

1、操作系统：CentOS7.3，最小安装方式安装

2、网卡：2

3、内存：8GB可用内存

4、硬盘：40GB

■ 软件依赖

1、docker==1.12.5

2、helm >= 2.4.1

3、kubectl >=1.6.1

4、kubeadm >=1.6.1

5、kubelet >=1.6.1

6、kubernets-cni >=0.5.1

7、kolla-ansible==4.0.0

8、kolla-kubernets==0.6.0

9、kolla==4.0.0

本文所有操作都在非root用户权限下进行，如果以root用户权限进行本文介绍的部署操作，则在helm安装部分会出错。

二、Kubernetes集群部署

■ 系统环境配置

1、关闭系统防火墙

sudo systemctl stop firewalld

sudo systemctl disable firewalld

2、关闭SELinux

sudo setenforce 0

sudo sed -i 's/enforcing/disabled/g' /etc/selinux/config

3、网络配置

修改/etc/sysctl.conf，如下：

net.ipv4.ip\_forward=1

net.ipv4.ip\_nonlocal\_bind=1

net.unix.max\_dgram\_qlen=128

net.bridge.bridge-nf-call-iptables=1

net.bridge.bridge-nf-call-ip6tables=1

net.ipv4.conf.all.rp\_filter=0

net.ipv4.conf.default.rp\_filter=0

■ Docker运行时环境安装准备

1、准备Docker安装源并安装Docker

sudo tee /etc/yum.repos.d/docker.repo << 'EOF'

[docker\_repo]

name=Docker Repository

baseurl=https://yum.dockerproject.org/repo/main/centos/$releasever/

enabled=1

gpgcheck=0

EOF

//安装相关的依赖包

sudo yum install python-devel libffi-devel gcc openssl-devel git \

python-pip epel-release

//安装Docker

sudo yum install docker-engine-1.12.5 docker-engine-selinux-1.12.5

2、配置Docker引擎

Docker在CentOS中安装后默认的StorageDrive为lvm-loop模式的DeviceMapper，这里推荐采用lvm-direct模式的DeviceMapper。假设系统中规划由于存储Docker镜像的存储块为/dev/sdb，则Docker的lvm-direct模式DeviceMapper存储引擎配置如下：

sudo pvcreate /dev/sdb

sudo vgcreate docker /dev/sdb

sudo lvcreate --wipesignatures y -n thinpool docker -l 95%VG

sudo lvcreate --wipesignatures y -n thinpoolmeta docker -l 1%VG

sudo lvconvert -y --zero n -c 512K --thinpool docker/thinpool

--poolmetadata docker/thinpoolmeta

sudo echo -e "activation {\nthin\_pool\_autoextend\_threshold=80

\nthin\_pool\_autoextend\_percent=20\n}">/etc/lvm/profile/docker-thinpool.profile

sudo lvchange --metadataprofile docker-thinpool docker/thinpool"

sudo lvs -o+seg\_monitor"

配置Docker引擎的MountFlags标志：

sudo mkdir /etc/systemd/system/docker.service.d

sudo echo -e "[Service]\n MountFlags=shared" >/etc/systemd/system

/docker.service.d/kolla.conf

配置Docker使用lvm-direct存储引擎、使用国内的DockerHub镜像(这里配置的为阿里云的DockerHub镜像)以及使用本地的Docker镜像Registry，如下：

sudo sed -i 's/\/usr\/bin\/dockerd/\/usr\/bin\/dockerd

--registry-mirror=https:\/\/v4wev6hv.mirror.aliyuncs.com

--insecure-registry 192.168.125.30:4000

--storage-driver=devicemapper

--storage-opt=dm.thinpooldev=\/dev\/mapper\/docker-thinpool

--storage-opt=dm.use\_deferred\_removal=true

--storage-opt=dm.use\_deferred\_deletion=true/'

/usr/lib/systemd/system/docker.service

3、启动Docker引擎

sudo /usr/bin/systemctl daemon-reload

sudo /usr/bin/systemctl start docker

sudo /usr/bin/systemctl enable docker

4、创建Docker本地Registry

sudo docker pull registry

sudo mkdir /data/registry

sudo docker run -d -v /data/registry:/var/lib/registry -p 4000:5000

--restart=always --name local\_registry registry:latest

5、通过Kolla项目下载Openstack的Docker镜像源

在正式通过Kubernetes编排部署之前，需要准备好Openstack各个服务项目的容器镜像，本文采用基于Ocata版本的Kolla4.0编译(编译类型为Source)得到的Openstack镜像(下载地址为：http://tarballs.openstack.org/kolla/images/[centos-source-registry-ocata.tar.gz](http://tarballs.openstack.org/kolla/images/centos-source-registry-ocata.tar.gz))，将[centos-source-registry-ocata.tar.gz](http://tarballs.openstack.org/kolla/images/centos-source-registry-ocata.tar.gz)下载并上传到系统后，将其直接解压至Docker本地Registry的目录/data/registry即可：

sudo tar zxvf centos-source-registry-ocata.tar.gz -C /data/registry/

可以通过curl来访问验证本地Registry是否正常：

curl http:// 192.168.125.30:4000/v2/\_catalog

■ 安装Kubernetes集群软件

官方推荐的Kubernetes集群部署软件安装如下：

//准备kubernets安装源，位于google的packages管理仓库中

sudo tee /etc/yum.repos.d/kubernetes.repo<<EOF

[kubernetes]

name=Kubernetes

baseurl=http://yum.kubernetes.io/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=0

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg

https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

EOF

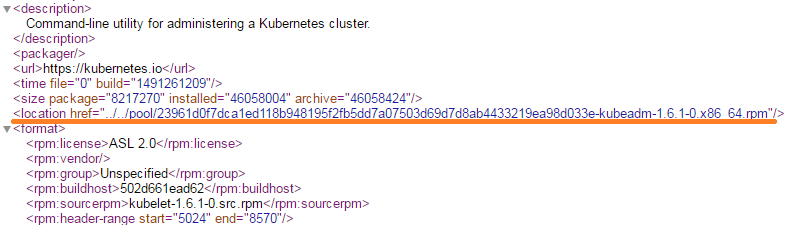
//采用yum直接安装

sudo yum install -y ebtables kubeadm kubectl kubelet kubernetes-cni

不翻墙的情况下，这种方式在国内几乎是不可能安装成功的，要访问google的服务器简直难于上青天！下面介绍一种相对简单可行的Kubernetes安装方式，首先确保你可以访问www.google.com，然后访问如下地址：

https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64/repodata/primary.xml

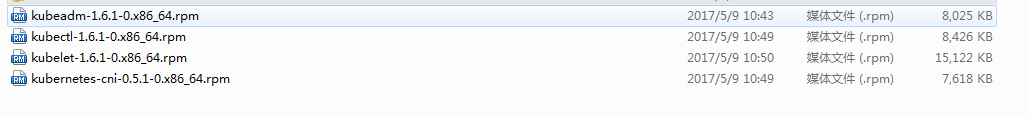
该地址显示的是XML文本，在其中可以找到需要安装的各个Kubernetes软件包源地址，如下：



利用上图中红色实线勾画部分构造RPM包下载地址(添加)，如下：

**https://packages.cloud.google.com/yum**/pool/23961d0f7dca1ed118b948195f2fb5dd7a07503d69d7d8ab4433219ea98d033e-kubeadm-1.6.1-0.x86\_64.rpm

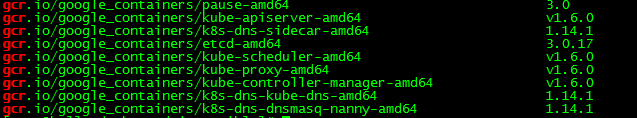
在浏览器中输入上述地址，浏览器将会自动下载kubeadm-1.6.1-0.x86\_64.rpm软件包，重复上述过程，下载全部所需的RPM包，如下：



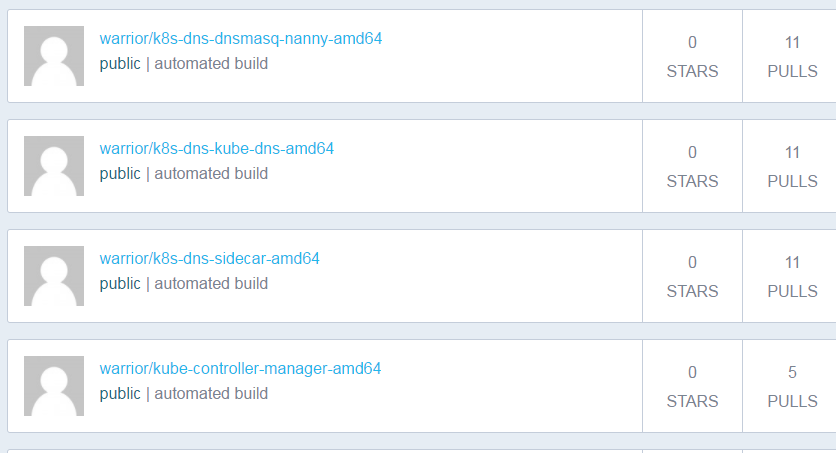
将上述软件包全部上传到CentOS系统中，可以制作一个本地YUM仓库，然后通过YUM进行安装，也可以通过rpm命令行进行安装。记得通过yum形式安装ebtables和socat！

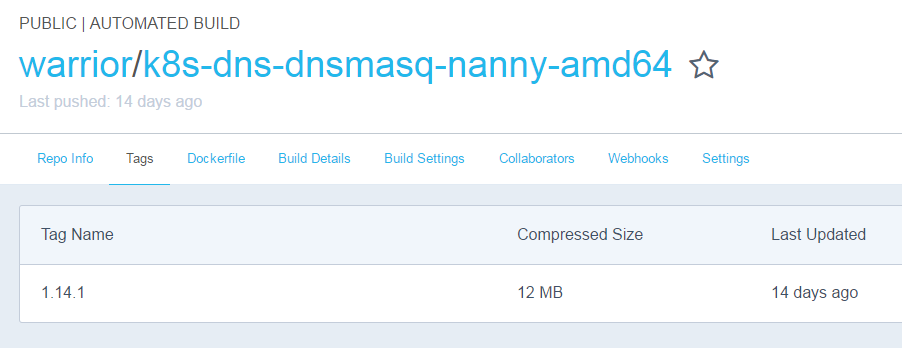
■ Kubernetes镜像获取

在Kubernetes1.6中，Kubernetes集群已经不再以系统进程形式部署在系统中了，而是以Docker容器形式部署。尽管kubeadm init命令会自动到Google容器仓库中Pull所需的镜像，但是这种方式绝对会让你崩溃到想放弃Kubernetes！因此，最好在运行kubeadm init之前准备好全部所需的镜像，在部署Kubernetes1.6.1时，需要的Kubernetes镜像如下(Kubernetes1.6.1仅需要以下镜像)：



获取上述镜像的一个可行方法，就是通过github和dockerhub相互配合，在github上制作编译镜像的Dockerfile文件，然后自己账户下的dockerhub与github关联起来，并通过dockerhub从github上提取Dockerfile文件并编译Docker镜像，由于Dockerhub服务器本身位于海外，因此编译镜像的速度还是可以接受的，待Dockerhub编译完成后，便可通过docker pull命令将其抓取到本地，之后再通过docker tag命令对抓取到本地的镜像进行重命名即可，具体制作过程可参考：https://mritd.me/2016/10/29/set-up-kubernetes-cluster-by-kubeadm/这篇博客。如果安装的是Kubernetes1.6.1，则可直接到笔者的Dockerhub中下载上述Kubernetes镜像（https://hub.docker.com/r/warrior/），如下：





可通过如下命令进行镜像提取：

docker pull warrior/k8s-dns-dnsmasq-nanny-amd64:1.14.1

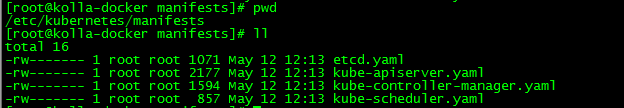
//镜像重命名

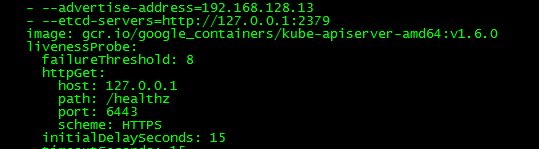
docker tag warrior/k8s-dns-dnsmasq-nanny-amd64:1.14.1 \

gcr.io/ google\_containers/k8s-dns-dnsmasq-nanny-amd64:1.14.1

笔者在https://hub.docker.com/r/warrior/中的镜像编译Dockerfile文件位于Github项目https://github.com/ynwssjx/K8s-images中，有兴趣的读者可自行查看，其实Dockerfile文件内容非常简单，仅有一个FROM语句，如：FROM gcr.io/google\_containers/etcd-amd64:3.0.17

其实在制作Kubernetes镜像时，有个问题就是如何知道kubeadm init命令所需的镜像版本号。在/etc/kubernetes/manifests目录中，有几个yaml文件，其中的image字段定义了所需的版本号，如下：





关于其他镜像的版本号查询，可参考前文提及的博客。

■ Kubernetes集群初始化

在Kubernetes集群中，POD和Service在创建时都会自动分配网络IP地址，这里假设POD网段为10.1.0.0/16，Service网段为10.3.3.0/24。设置Kubernetes集群的DNS 服务器，如下：

sudo sed -i 's/10.96.0.10/10.3.3.10/g' \

/etc/systemd/system/kubelet.service.d/10-kubeadm.conf

Kubernetes使用服务网段的x.x.x.10IP地址来作为DNS服务器地址，1到9被预留作为Kubernetes后续基础服务的扩展需求。启动kubelet服务，如下：

sudo systemctl daemon-reload

sudo systemctl stop kubelet

sudo systemctl enable kubelet

sudo systemctl start kubelet

此时如果通过systemctl status kubelet来查看kubelet的启动状态，将会发现kubelet启动失败，这个时候先不用在意报错，继续后续操作。使用kubeadm命令行进行Kubernetes集群初始化，如下：

sudo kubeadm init --pod-network-cidr=10.1.0.0/16 \

--service-cidr=10.3.3.0/24 \

--apiserver-advertise-address 192.168.125.30

apiserver-advertise-address为API Server的监听地址

等待一段时间，看到successful即说明Kubernetes集群初始化完成。加载kubeadm认证到系统环境变量中，如下：

mkdir -p $HOME/.kube

sudo -H cp /etc/kubernetes/admin.conf $HOME/.kube/config

sudo -H chown $(id -u):$(id -g) $HOME/.kube/config

此时，再开启一个终端，su到部署Kubernetes集群的用户home目录，输入如下命令，即可观察到Kubernetes集群中PODs变化过程：

watch -d kubectl get pods --all-namespaces -o wide

在当前阶段，Kolla-kubernets项目推介使用的Kubernetes集群网络驱动是Canal。部署Canal网络驱动，如下：

sudo curl -L https://raw.githubusercontent.com/projectcalico\

/canal/master/k8s-install/1.6/rbac.yaml -o rbac.yaml

sudo kubectl apply -f rbac.yaml

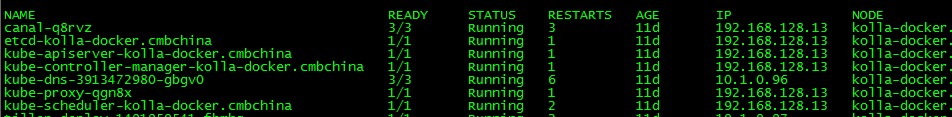
sudo curl -L https://raw.githubusercontent.com/projectcalico/canal\

/master/k8s-install/1.6/canal.yaml -o canal.yaml\

sudo sed -i "s@10.244.0.0/16@10.1.0.0/16@" canal.yaml

sudo kubectl apply -f canal.yaml

Canal网络驱动部署成功后，Kubernetes的kube-system命名空间中的全部pods应该是running状态(在此之前，dns并非running状态)，如下：



因为此处采用的是AIO部署模式，而默认情况下Kubernetes的Master节点是不会被kube-scheduler分配PODs到其上的运行的，为了让PODs在Master上也能够运行，对Master进行untaint操作，如下：

kubectl taint nodes --all=true \

node-role.kubernetes.io/master:NoSchedule-

■ Kubernetes集群验证

待kube-system命名空间中全部pods处于running状态之后，通过kubectl命令启动一个busybox容器，在该容器中进行DNS查询操作，如果操作成功则说明Kubernetes集群部署成功，否则Kubernetes集群是存在问题的，需要检查上述各个步骤。验证过程如下：

kubectl run -i -t $(uuidgen) --image=busybox --restart=Never

//进入容器后，执行nslookup操作

# nslookup kubernetes

Server: 10.3.3.10

Address 1: 10.3.3.10 kube-dns.kube-system.svc.cluster.local

Name: kubernetes

Address 1: 10.3.3.1 kubernetes.default.svc.cluster.local

如果能够看到上述信息，说明Kubernetes集群部署是成功的，否则便是有问题的。

三、部署kolla-kubernets

■ 覆盖默认的RBAC设置

通过kubectl replace命令进行默认RBAC设置的覆盖，如下：

kubectl update -f <(cat <<EOF

apiVersion: rbac.authorization.k8s.io/v1alpha1

kind: ClusterRoleBinding

metadata:

name: cluster-admin

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: Group

name: system:masters

- kind: Group

name: system:authenticated

- kind: Group

name: system:unauthenticated

EOF

)

■ 安装部署Helm

Helm是Kubernetes中的包管理器，类似yum包管理工具，yum用来安装RPM包，而Helm用来安装charts，这里的charts便类似RPM软件包。Helm分为客户端和服务器端，Helm的服务器端称为tiller，服务器端在Kubernetes中以Docker容器形式运行，为了便于Helm安装，可以实先将Tiller的容器镜像下载到本地，可使用如下命令下载：

docker pull warrior/kubernetes-helm:2.4.1

docker tag warrior/kubernetes-helm:2.4.1 \

gcr.io/kubernetes-helm/tiller:v2.4.1

安装Helm最简单的方式如下：

sudo curl -L https://raw.githubusercontent.com/kubernetes/helm\

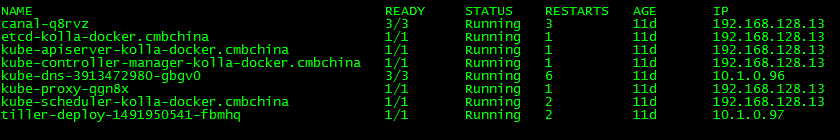
/master/scripts/get > get\_helm.sh

sudo chmod 700 get\_helm.sh

sudo ./get\_helm.sh

sudo helm init

Hel安装完成后，可以看到kube-system命名空间中新增了一个running状态的POD，名称为tiller-deploy-xxx，如下：



Helm安装成功后，通过helm version即可看到客户端和服务器端的信息，如下：



■ 安装kolla-ansible和kolla-kubernets

Clone社区Kolla-ansible源代码，如下：

git clone http://github.com/openstack/kolla-ansible

git clone http://github.com/openstack/kolla-kubernetes

安装kolla-ansible和kolla-kubernets，如下：

sudo pip install -U kolla-ansible/ kolla-kubernetes/

复制默认的kolla配置文件到/etc目录，如下：

sudo cp -aR /usr/share/kolla-ansible/etc\_examples/kolla /etc

复制 kolla-kubernetes 配置文件至/etc目录，如下：

sudo cp -aR kolla-kubernetes/etc/kolla-kubernetes /etc

为Openstack集群各个项目和用户生成密码文件，如下：

sudo kolla-kubernetes-genpwd

在kubernets中创建一个独立的密码空间kolla，如下：

kubectl create namespace kolla

将AIO节点标记为控制节点和计算节点，如下：

kubectl label node $(hostname) kolla\_compute=true

kubectl label node $(hostname) kolla\_controller=true

修改/etc/kolla/globals.yml配置文件，其中：network\_interface和neutron\_external\_interface两个变量需要用户指定，network\_interface是管理接口(如eth0)，默认也是Openstack各个服务项目的API接口，neutron\_external\_interface是Neutron项目用于外网桥接的物理接口(如eth1)，该接口上不要手工配置IP地址。

将需要启动的服务项目添加到/etc/kolla/globals.yml的末尾，如下：

cat <<EOF > add-to-globals.yml

kolla\_install\_type: "source"

tempest\_image\_alt\_id: "{{ tempest\_image\_id }}"

tempest\_flavor\_ref\_alt\_id: "{{ tempest\_flavor\_ref\_id }}"

neutron\_plugin\_agent: "openvswitch"

api\_interface\_address: 0.0.0.0

tunnel\_interface\_address: 0.0.0.0

orchestration\_engine: KUBERNETES

memcached\_servers: "memcached"

keystone\_admin\_url: "http://keystone-admin:35357/v3"

keystone\_internal\_url: "http://keystone-internal:5000/v3"

keystone\_public\_url: "http://keystone-public:5000/v3"

glance\_registry\_host: "glance-registry"

neutron\_host: "neutron"

keystone\_database\_address: "mariadb"

glance\_database\_address: "mariadb"

nova\_database\_address: "mariadb"

nova\_api\_database\_address: "mariadb"

neutron\_database\_address: "mariadb"

cinder\_database\_address: "mariadb"

ironic\_database\_address: "mariadb"

placement\_database\_address: "mariadb"

rabbitmq\_servers: "rabbitmq"

openstack\_logging\_debug: "True"

enable\_haproxy: "no"

enable\_heat: "no"

enable\_cinder: "yes"

enable\_cinder\_backend\_lvm: "yes"

enable\_cinder\_backend\_iscsi: "yes"

enable\_cinder\_backend\_rbd: "no"

enable\_ceph: "no"

enable\_elasticsearch: "no"

enable\_kibana: "no"

glance\_backend\_ceph: "no"

cinder\_backend\_ceph: "no"

nova\_backend\_ceph: "no"

EOF

cat ./add-to-globals.yml | sudo tee -a /etc/kolla/globals.yml

如果是在虚拟机上进行部署，则需要使用qemu虚拟化引擎，如下：

sudo mkdir /etc/kolla/config

sudo tee /etc/kolla/config/nova.conf<<EOF

[libvirt]

virt\_type=qemu

cpu\_mode=none

EOF

生成默认的Openstack各个项目配置文件，如下:

sudo kolla-ansible genconfig

为Openstack各个项目创建Kubernetes秘钥并将其注册到Kubernetes集群中，如下：

kolla-kubernetes/tools/secret-generator.py create

创建并注册kolla的config map，如下:

kollakube res create configmap \

mariadb keystone horizon rabbitmq memcached nova-api nova-conductor\

nova-scheduler glance-api-haproxy glance-registry-haproxy \

glance-api glance-registry neutron-server neutron-dhcp-agent \

neutron-l3-agent neutron-metadata-agent neutron-openvswitch-agent \

openvswitch-db-server openvswitch-vswitchd nova-libvirt nova-compute\

nova-consoleauth nova-novncproxy nova-novncproxy-haproxy\

neutron-server-haproxy nova-api-haproxy cinder-api cinder-api-haproxy\

cinder-backup inder-scheduler cinder-volume iscsid tgtd keepalived\

placement-api placement-api-haproxy

启用resolv.conf解决方法，如下：

kolla-kubernetes/tools/setup-resolv-conf.sh kolla

编译Helm的microcharts、service charts和 metacharts，如下：

kolla-kubernetes/tools/helm\_build\_all.sh ./

编译过程会花费一定时间，编译完成会在当前目录上产生很多.tgz的文件，其数目至少要大于150个。

创建一个本地cloud.yaml文件，用户安装部署Helm的charts，如下：

global:

kolla:

all:

docker\_registry: 192.168.128.13:4000 //本地registry仓库地址

image\_tag: "4.0.0"

kube\_logger: false

external\_vip: "192.168.128.13"

base\_distro: "centos"

install\_type: "source"

tunnel\_interface: "ens34" //管理接口

resolve\_conf\_net\_host\_workaround: true

keystone:

all:

admin\_port\_external: "true"

dns\_name: "192.168.128.13"

public:

all:

port\_external: "true"

rabbitmq:

all:

cookie: 67

glance:

api:

all:

port\_external: "true"

cinder:

api:

all:

port\_external: "true"

volume\_lvm:

all:

element\_name: cinder-volume

daemonset:

lvm\_backends:

- '192.168.128.13': 'cinder-volumes' //cinder后端VG名称

ironic:

conductor:

daemonset:

selector\_key: "kolla\_conductor"

nova:

placement\_api:

all:

port\_external: true

novncproxy:

all:

port: 6080

port\_external: true

openvwswitch:

all:

add\_port: true

ext\_bridge\_name: br-ex

ext\_interface\_name: ens41 //Neutron外网桥接网口

setup\_bridge: true

horizon:

all:

port\_external: true

cloud.yaml文件需要根据用户各自的环境进行修改，上述文件中的192.168.128.13是笔者管理网口ens34上的IP地址，在使用过程中需要进行相应的修改。

■ 使用Helm在Kubernetes上部署Openstack

首先部署MariaDB，并等待其进入running状态，如下：

helm install --debug kolla-kubernetes/helm/service/mariadb \

--namespace kolla --name mariadb --values ./cloud.yaml

待数据库稳定后，部署其他的Openstack服务，如下：

helm install --debug kolla-kubernetes/helm/service/rabbitmq \

--namespace kolla --name rabbitmq --values ./cloud.yaml

helm install --debug kolla-kubernetes/helm/service/memcached \

--namespace kolla --name memcached --values ./cloud.yaml

helm install --debug kolla-kubernetes/helm/service/keystone\

--namespace kolla --name keystone --values ./cloud.yaml

helm install --debug kolla-kubernetes/helm/service/glance \

--namespace kolla --name glance --values ./cloud.yaml

helm install --debug kolla-kubernetes/helm/service/cinder-control\

--namespace kolla --name cinder-control --values ./cloud.yaml

helm install --debug kolla-kubernetes/helm/service/horizon \

--namespace kolla --name horizon --values ./cloud.yaml

helm install --debug kolla-kubernetes/helm/service/openvswitch \

--namespace kolla --name openvswitch --values ./cloud.yaml

helm install --debug kolla-kubernetes/helm/service/neutron \

--namespace kolla --name neutron --values ./cloud.yaml

helm install --debug kolla-kubernetes/helm/service/nova-control\

--namespace kolla --name nova-control --values ./cloud.yaml

helm install --debug kolla-kubernetes/helm/service/nova-compute \

--namespace kolla --name nova-compute --values ./cloud.yaml

当nova-compute进入running状态后，创建cell0数据库，如下：

helm install --debug kolla-kubernetes/helm/microservice\

/nova-cell0-create-db-job --namespace kolla --name \

nova-cell0-create-db-job --values ./cloud.yaml

helm install --debug kolla-kubernetes/helm/microservice\

/nova-api-create-simple-cell-job --namespace kolla \

--name nova-api-create-simple-cell --values ./cloud.yaml

当上述全部POD进入running状态后，部署Cinder LVM。这里假设系统上已经有一个名为cinder-volumes的VG存在，如果还没有cinder-volume这个VG，则需要事先创建该VG，如下：

pvcreate /dev/sdb /dev/sdc

vgcreate cinder-volumes /dev/sdb /dev/sdc

安装部署cinder-volume，如下:

helm install --debug kolla-kubernetes/helm/service/cinder-volume-lvm\

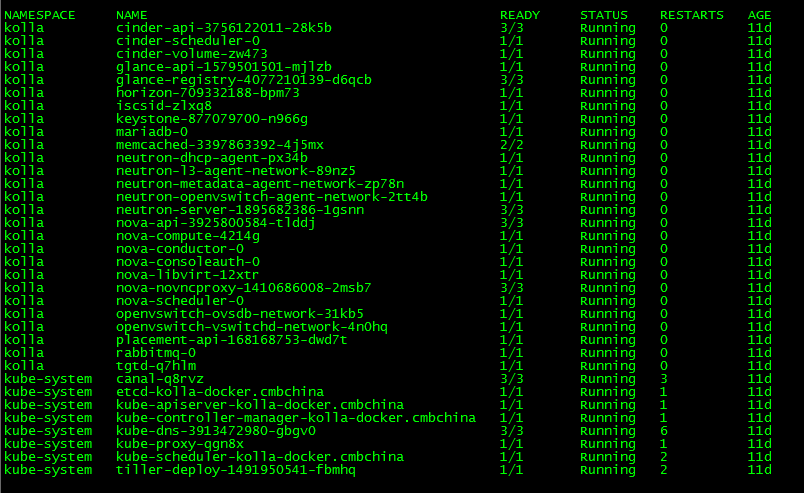
--namespace kolla --name cinder-volume-lvm --values ./cloud.yaml

如果要删除helm部署的charts，如cinder-volume-lvm，则通过命令：

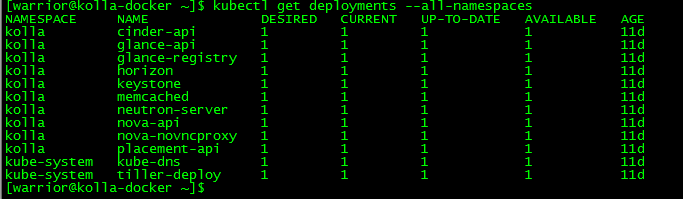
helm delete cinder-volume-lvm --purge

即可从kubernets集群中清除cinder-volume相关的PODs。

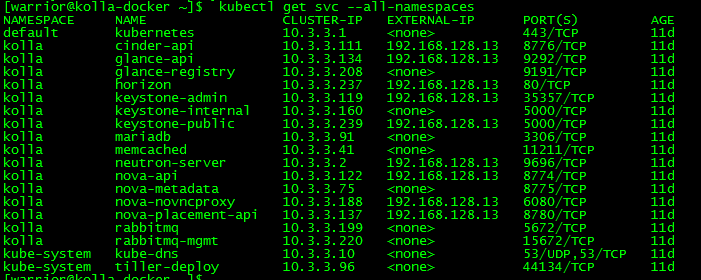
至此，全部Openstack服务已经部署完成，在操作Openstack集群之前，先等待所有Kubernetes集群中的PODs处于running状态，如下：



查看kubernets集群中的全部deployment，如下：



查看kubernets集群中的全部service，如下:



可以看到，每个service都被自动分配了10.3.3.0/24网段的IP地址，并且可以看到各个service对应的端口。在确认Kubernetes的各个API对象正常运行后，便可通过Openstack命令行客户端进行Openstack集群操作。首先，生成并加载openrc文件，如下：

kolla-kubernetes/tools/build\_local\_admin\_keystonerc.sh ext

source ~/keystonerc\_admin

通过kolla-ansible提供的init-runonce脚本初始化Openstack，并launch一个VM，如下：

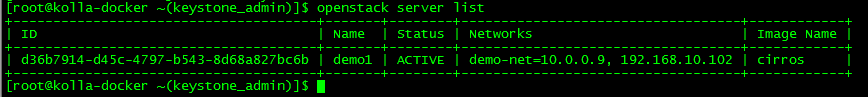
kolla-ansible/tools/init-runonce

创建一个FloatingIP地址，并将其添加到VM上，如下：

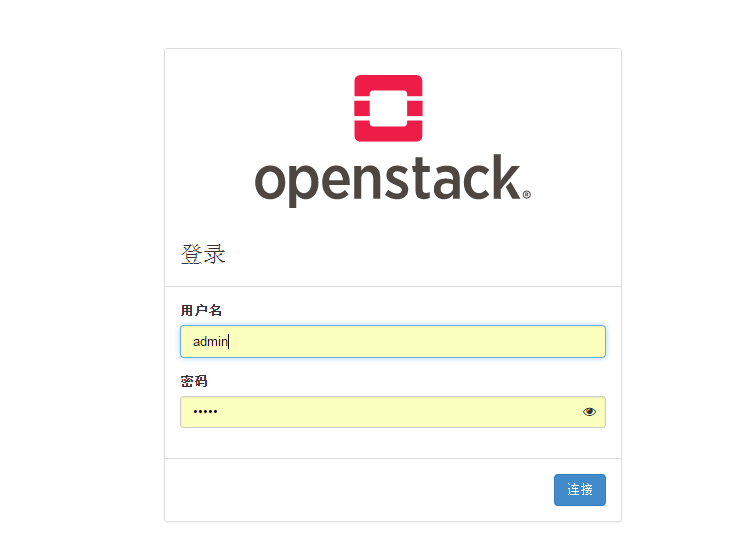
openstack server add floating ip demo1 $(openstack floating ip \

create public1 -f value -c floating\_ip\_address)

查看创建的VM，如下：



登录Dashboard(http://192.168.128.13)，如下：



在dashboard上查看创建的实例，如下：



创建一个块存储，并将其attach到实例demo1上，如下：



到此，Ocata版本的Openstack已经成功部署在Kubernetes集群上。由于诸多原因，目前Kolla-kubernets项目仍然不具备生产环境部署条件，社区目前也仅支持AIO的开发实验性质的部署，相信随着K8S的兴趣，Kolla-kubernets项目的重视程度也会与日俱增，而且可以预言，在不久的将来，通过K8S部署Openstack容器云将会是Openstack的一大主流方向！

参考文档：

1、https://github.com/openstack/kolla-kubernetes/blob/master/doc/deployment-guide.rst

2、https://mritd.me/2016/10/29/set-up-kubernetes-cluster-by-kubeadm/

3、http://yoyolive.com/2017/02/27/Kubernetes-1-5-3-Local-Install/

4、https://github.com/projectcalico/canal/tree/master/k8s-install

5、https://kubernetes.io/docs/admin/kubeadm/

6、https://kubernetes.io/docs/getting-started-guides/kubeadm/