Date 12/26/2016, by `@cloudsskyone`

QuadrupleO:

Enjoy OpenShift Origin On OpenStack

… enhancements in process
(new enhancements will be tweeted with #quadrupleo hashtag)

History:

Date	Decription
12/22/16	1st version (3 node install)
12/30/16	2nd version (9 node HA env. with Terraform)
01/01/17	Added Git Repo for Terraform config files
01/09/2017	IoT OCP Example added

Overview

This walkthrough should work on bare metal servers or any other public cloud environments with CentOS 7.2 / 7.3 by leaving out some OpenStack specific steps or with some minor adjustments for public cloud environments.

Please refer to this blog post for more high level background information.

Let’s start with the work!

On our OpenStack RDO Mitaka Lab we’re running 5 CentOS 7.3.1611 VMs.

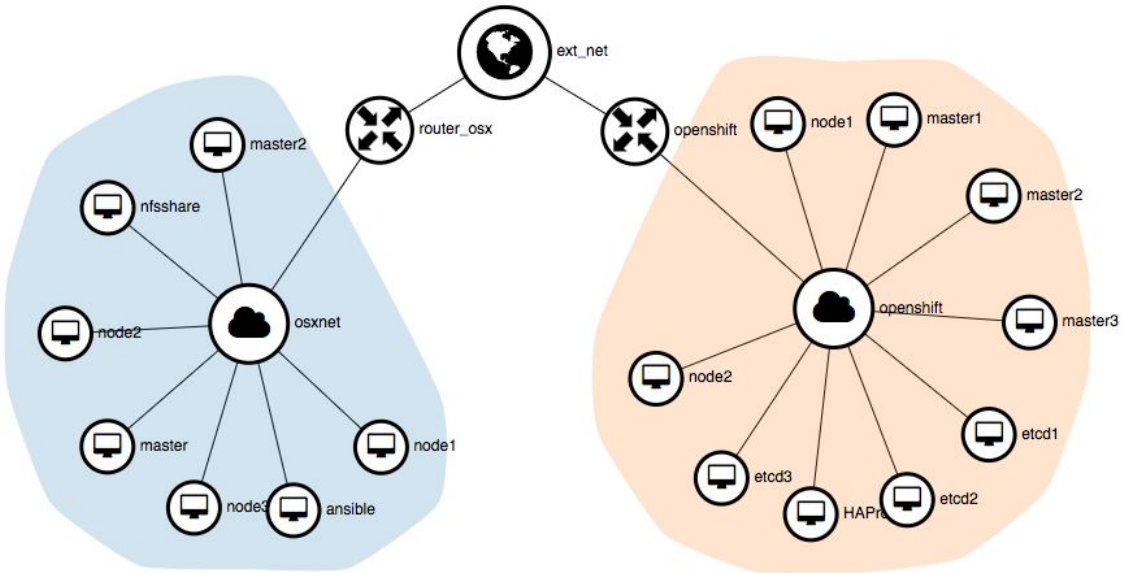
- ansible node
- master node
- worker node1
- worker node2
- NFS Share node

You can run ansible on your laptop, but for this guide we’re running the ansible node on our OpenStack environment.

We’ll extend this environment later with 2 additional nodes, the master2 and the worker node3 nodes to see how adding additional nodes works and how to achieve HA.

Note: to automate / orchestrate the installation on OpenStack for very large environments, these heat templates might work for Origin on OpenStack too.

We recommend to use Terraform in long term to build your OpenShift Cluster Hosts anywhere in less than 2 minutes, a Terraform script for a 9 node install is provided on Github, in Appendix you’ll find a howto about how to use it.



The ansible node (aka Bastian Server) is used to run the installation from it to other nodes, the master node hosts our OpenShift web console and the Kubernetes master, etcd and more, the worker nodes (aka minions) do the real work and run our pods, the docker registry and our routers. The docker registry runs only on one of our worker nodes. The NFS Share node is used to provide persistent storage for our docker storage volumes. If you've OpenStack Newton, you can use Manila to provide shared storage.

After the End of the installation and configuration of the system, we'll deploy one or more legacy apps and see how we can apply (auto-) scaling capabilities to some of our apps and not to others out of the box.

All VMs have a FQDN defined in their "/etc/hosts" file, the master and the 2 worker nodes are registered with their public floating IP in DNS with a wildcard DNS entry. The ansible and NFS Share node don't need to have a floating IP assigned. On all VMs is the root user enabled and you should be able to ssh passwordless from the ansible node into all other nodes by running ssh-keygen (with no password) followed by ssh-copy-id, e.g.:

```
[root@ansible] ssh-copy-id -i ~/.ssh/id_rsa.pub master<your domain>
```

Tip: to do your live easier, we'd recommend to create a base CentOS 7.3 image on OpenStack and create a SnapShot from it and run the master and worker node instances from this golden SnapShot.

The base CentOS image should have the following definitions in /etc/ssh/sshd_config

```
PermitRootLogin yes
PasswordAuthentication yes
```

Use "passwd" to set the root password and delete the following line section in .ssh/authorized_keys

```
no-port-forwarding,no-agent-forwarding,no-X11-forwarding,command="echo 'Please
login as the user \"centos\" rather than the user \"root\".';echo;sleep 10"
```

And run:

```
$ systemctl restart sshd.service
$ ssh-keygen
→ press 2 time enter
$ cat .ssh/id_rsa.pub >> .ssh/authorized_keys
$ exit
```

Now you should be able to ssh into the CentOS instance without your key (by providing the root password).

Install the following packages on your CentOS base image, update and reboot:

```
$ yum -y install wget git net-tools bind-utils iptables-services bridge-utils
bash-completion
$ yum -y update
$ yum -y install NetworkManager
$ systemctl start NetworkManager.service
$ systemctl status NetworkManager.service
$ systemctl enable NetworkManager.service
$ systemctl disable sshd.service#currently needed because sshd dies (at least
on our environment)
$ systemctl enable sshd.socket
$ systemctl reboot
```

We'd recommend to create a SnapShot from this CentOS base image (e.g. Snap-CentOS-7.3-Gold).

Note: After running the master, worker and nfsshare instances, please set the hostname on all nodes:

```
[root@master ~]# hostnamectl set-hostname master.<your domain>
[root@node-1 ~]# hostnamectl set-hostname node1.<your domain>
[root@node-2 ~]# hostnamectl set-hostname node2.<your domain>
```

Preparation is the Key for a Successful Deployment!

First we'll prepare our ansible node and clone the openshift-ansible repo from GitHub to it and run the ansible playbook to install the required packages to the master and 2 other worker nodes and configure the network between these nodes automatically.

Prepare the Ansible node

On your ansible node the /etc/hosts file should look similar to this:
(define this hosts file on the master and worker nodes too)

```
[root@ansible ~]# cat /etc/hosts
10.0.1.41 master.<your domain> master
10.0.1.42 node1.<your domain> node1
10.0.1.43 node2.<your domain> node2
10.0.1.44 nfsshare.<your domain> nfsshare
```

Please replace <your domain> with your own domain name and adjust the IPs.

On the ansible node we need to install epel, ansible and pyOpenSSL packages and clone the openshift-ansible repository from Github by running:

```
[root@ansible ~] yum -y install
https://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-.noarch.rpm
[root@ansible ~] sed -i -e "s/^enabled=1/enabled=0/" /etc/yum.repos.d/epel.repo
[root@ansible ~] yum -y --enablerepo=epel install ansible pyOpenSSL
[root@ansible ~] git clone https://github.com/openshift/openshift-ansible
[root@ansible ~] cd openshift-ansible/
[root@ansible openshift-ansible]# ll
total 232
-rw-r--r--.  1 root root    650 Dec 17 17:53 ansible.cfg.example
drwxr-xr-x.  3 root root     45 Dec 17 17:53 ansible-profile
drwxr-xr-x.  2 root root     20 Dec 17 17:53 bin
-rw-r--r--.  1 root root   1020 Dec 17 17:53 BUILD.md
….
(truncated)
```

The ansible node is now ready to go!

Prepare the Master node

On the master node please install the epel repo and Docker Application Container Engine and verify if docker is running properly.

```
[root@master ~]
yum -y install
https://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-8.noarch.rpm
[root@master ~] sed -i -e "s/^enabled=1/enabled=0/" /etc/yum.repos.d/epel.repo
[root@master ~]  yum install docker -y
[root@master ~] systemctl is-active docker
unknown

[root@master ~] systemctl enable docker.service
[root@master ~] systemctl start docker.service
[root@master ~] systemctl status docker.service

ð docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor
preset: disabled)
  Drop-In: /etc/systemd/system/docker.service.d
           └─custom.conf
         /usr/lib/systemd/system/docker.service.d
           └─docker-sdn-ovs.conf
   Active: active (running) since Wed 2016-12-21 02:48:28 UTC; 15h ago

[root@origin-master ~]# docker version
Client:
 Version:         1.10.3
...
Server:
 Version:         1.10.3
```

Prepare the Worker nodes (node1 and node2)

On the worker nodes you need to install only docker:

On node 1:

```
[root@node1 ~]  yum install docker -y
[root@node1 ~] systemctl is-active docker
[root@node1 ~] systemctl enable docker.service
[root@node1 ~] systemctl start docker.service
[root@node1 ~] systemctl status docker.service
```

On node 2:

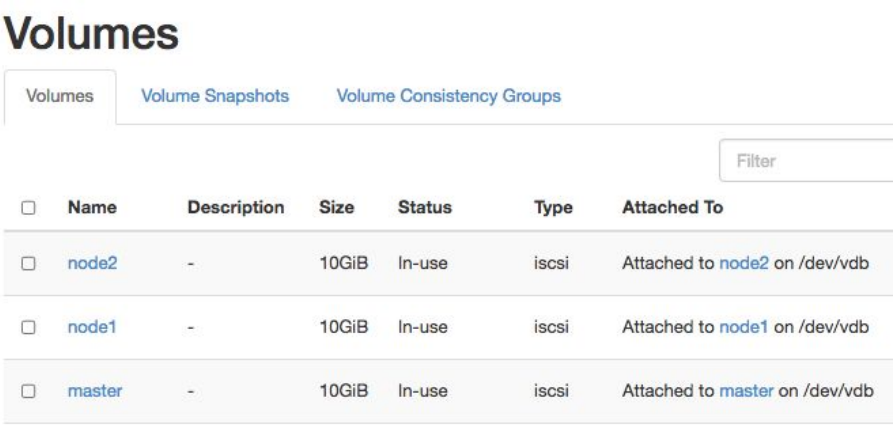
```
[root@node2 ~] yum install docker -y
[root@node2 ~] systemctl is-active docker
[root@node2 ~] systemctl enable docker.service
[root@node2 ~] systemctl start docker.service
[root@node2 ~] systemctl status docker.service
```

Not sure why these steps are not automated through ansible config management yet, but you could install docker on your CentOS Golden OpenShift base image and avoid installing it on every node. If you use Fedora 25 Atomic Host, you don't need to deal with these steps above, but for now we want to go with our lovely CentOS 7.3 golden image!

Configuring Docker Storage on master and worker nodes

On OpenStack create a volume and attach to all instances (master and worker nodes), the volume will be attached to the device /dev/vdb, edit the docker-storage-setup file as follow and verify if the volume groups have been setuped properly (for each node I created a 10GiB master, node1 and node2 volumes):

```
$ vi /etc/sysconfig/docker-storage-setup
DEVS=/dev/vdb
VG=docker-vg

$ docker-storage-setup
$ systemctl is-active docker
$ systemctl stop docker
$ rm -rf /var/lib/docker/*
$ systemctl start docker
$ systemctl status docker.service
$ vgdisplay
$ fdisk -l
$ lsblk
```



Adjust the Security Group Settings on OpenStack

On OpenStack you need to adapt the default security group, or whatever security group you're using for your VMs for your Origin installation, please refer to the following OpenShift doc: https://docs.openshift.org/latest/install_config/install/prerequisites.html

We'd recommend to create a new security group, e.g. OpenShiftOriginSecGroup and free up the following tcp / udp ports:

Table 7: IaaS Deployments:

22	SCO	Qdp thqdc enq RRH ax sgd hmrs`kkdq nq rxrsd l `c l hmhrsq`snq.
53 nq **8053**	SCO/TDO	Qdp thqdc enq DMR qdrnktshnm ne bktrsdq rdquhbdr (RjxDMR). Emrs`kk`shnmr oqhnq sn 1.2 nq dmuhqnm l dmsr tofq`cdc sn 1.2 trd onqs 53. Mdv hmrs`kk`shnmr vhkk trd 8053 ax cde`tks rn sg`s **cmr l `rp** l `x ad bnmehftqdc. Nmkx qdp thqdc sn ad hmsdqm`kkx nodm nm l `rsdq gnrsr.
80 nq **443**	SCO	Fnq HSSO/HSSOR trd enq sgd qntsdq. Qdp thqdc sn ad dwsdqm`kkx nodm nm mncd gnrsr, drodbh`kkx nm mncdr qtmmhmf sgd qntsdq.
1936	SCO	Fnq qntsdq rs`shrshbr trd. Qdp thqdc sn ad nodm vgdm qtmmhmf sgd sd l ok`sd qntsdq sn `bbdrr rs`shrshbr, `mc b`m ad nodm dwsdqm`kkx nq hmsdqm`kkx sn bnmmdbshnmr cdodmchmf nm he xnt v`ms sgd rs`shrshbr sn ad dwoqdrrdc otakhbkx.
4001	SCO	Fnq d l adccdc dsbc (mnm-bktrsdqdc) trd. Nmkx qdp thqdc sn ad hmsdqm`kkx nodm nm sgd l `rsdq gnrs. **4001** hr enq rdqudq-bkhdms bnmmdbshnmr.

2379 `mc **2380**	SCO	Fnq rs`mc`knmd dsbc trd. Nmkx qdp thqdc sn ad hmsdqm`kkx nodm nm sgd l `rsdq gnrs. **2379** hr enq rdqudq-bkhdms bnmmdbshnmr. **2380** hr enq rdqudq-rdqudq bnmmdbshnmr, `mc hr nmkx qdp thqdc he xnt g`ud bktrsdqdc dsbc.
4789	TDO	Fnq UwKAM trd (NodmRghes Nqhfhm RDM). Qdp thqdc nmkx hmsdqm`kkx nm mncd gnrsr.
8443	SCO	Fnq trd ax sgd NodmRghes Nqhfhm vda bnmrnkd, rg`qdc vhsg sgd AOĖ rdqudq.
10250	SCO	Fnq trd ax sgd Jtadkds. Qdp thqdc sn ad dwsdqm`kkx nodm nm mncdr.

NFS Share Storage Setup

On the nfsshare node create the nfsshare folder and make it world readable (your nfs share should only be accessible from your private net).

Install nfs-utils (if not installed already)

```
[root@nfsshare ~]# yum install nfs-utils
[root@nfsshare ~]# mkdir /nfsshare; chmod 777 /nfsshare
[root@nfsshare ~]# chown -R nfsnobody:nfsnobody /nfsshare
[root@nfsshare ~]# systemctl enable nfs-server.service
[root@nfsshare ~]# systemctl start nfs-server.service
```

Edit or create your exports file “/etc/exports”

```
[root@nfsshare ~]# cat /etc/exports
/nfsshare  *(rw,root_squash,no_wdelay)
```

Finally export the /nfsshare folder to be accessible from other client nodes later (node1 and node2) and verify the export:

```
[root@nfsshare ~]# exportfs -a
[root@nfsshare ~]# exportfs
/nfsshare         <world>

[root@nfsshare ~]# ls -lZ /nfsshare
```

After the installation if you run some containers like MySQL, Mariadb or MongoDB which need persistent storage, you should see something like this:

```
[root@nfsshare ~]# ls -lZ /nfsshare
-rw-rw----.   27             27 system_u:object_r:default_t:s0   aria_log.00000001
-rw-rw----.   27             27 system_u:object_r:default_t:s0   aria_log_control
-rw-r--r--.  184 systemd-network system_u:object_r:default_t:s0   collection-0--4010391386787507569.wt
-rw-r--r--.  184 systemd-network system_u:object_r:default_t:s0   collection-2--4010391386787507569.wt
-rw-r--r--.  184 systemd-network system_u:object_r:default_t:s0   collection-4--4010391386787507569.wt
drwxr-xr-x.  184 systemd-network system_u:object_r:default_t:s0   diagnostic.data
-rw-rw----.   27             27 system_u:object_r:default_t:s0   ibdata1
-rw-rw----.   27             27 system_u:object_r:default_t:s0   ib_logfile0
-rw-rw----.   27             27 system_u:object_r:default_t:s0   ib_logfile1
-rw-r--r--.  184 systemd-network system_u:object_r:default_t:s0   index-1--4010391386787507569.wt
-rw-r--r--.  184 systemd-network system_u:object_r:default_t:s0   index-3--4010391386787507569.wt
-rw-r--r--.  184 systemd-network system_u:object_r:default_t:s0   index-5--4010391386787507569.wt
-rw-r--r--.  184 systemd-network system_u:object_r:default_t:s0   index-6--4010391386787507569.wt
drwxr-xr-x.  184 systemd-network system_u:object_r:default_t:s0   journal
-rw-rw----.   27             27 system_u:object_r:default_t:s0   mariadb-1-vrxk1.pid
-rw-r--r--.  184 systemd-network system_u:object_r:default_t:s0   _mdb_catalog.wt
-rw-r--r--.  184 systemd-network system_u:object_r:default_t:s0   mongod.lock
-rw-rw----.   27             27 system_u:object_r:default_t:s0   multi-master.info
drwx------.   27             27 system_u:object_r:default_t:s0   mysql
```

Mount the nfsshare on node1 and node2

On node1 and node2 create the nfsshare folder under /nfsshare and mount it from the nfsshare “server” and allow nfs to mount with selinux enforced:

```
$ mkdir /nfsshare
$ mount nfsshare:/nfsshare /nfsshare/
$ mount -l
$ vi /etc/fstab
→ add this line:
nfsshare:/nfsshare /nfsshare   nfs    defaults 0 0
$ setsebool -P virt_use_nfs 1
$ setsebool -P virt_sandbox_use_nfs 1
$ iptables -I INPUT 1 -p tcp --dport 2049 -j ACCEPT
```

Create the ansible inventory file

On the ansible node we need to create the inventory file, which defines the topology of our OpenShift Kubernetes Cluster (master and nodes), the deployment type (origin), the admin and the user credentials, node labels and more.

By default the ansible hosts inventory file is created under “/etc/ansible/hosts”, if not, please create the “/etc/ansible/hosts” file and replace **<your domain>** with your own domain name:

To understand the details of the hosts inventory file, please refer to this guide for Fedora Atomic.
The inventory file defines the set of servers grouped into different classes for ansible to configure. The members of a certain class, e.g. “masters” will get the same configuration.

```
[root@ansible ~]# vi /etc/ansible/hosts

# Create an OSEv3 group that contains the masters and nodes groups
[OSEv3:children]
masters
nodes
#etcd

# Set variables common for all OSEv3 hosts
[OSEv3:vars]
# SSH user, this user should allow ssh based auth without requiring a password
ansible_ssh_user=root
```

```
# router and registry selector (for HA deployment)

openshift_router_selector='router=true'
openshift_registry_selector='registry=true'

# To deploy origin, change the deployment_type to origin
deployment_type=origin

# enable htpasswd authentication for admin and user users
#The admin user password is OriginAdmin, the user password is OriginUser

openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename':
'/etc/origin/master/htpasswd'}]
openshift_master_htpasswd_users={'admin':
'$apr1$zgSjCrLt$1KSuj66CggeWSv.D.BXOA1', 'user':
'$apr1$.gw8w9i1$ln9bfTRiD6OwuNTG5LvW50'}

openshift_master_default_subdomain=cloudapps.<your domain>

# host group for masters
[masters]
master.<your domain> openshift_node_labels="{'region': 'infra', 'zone':
'default'}" openshift_public_hostname=master.<your domain>
openshift_hostname=master.<your domain> openshift_public_ip=<public floating ip
of the master>

# host group for etcd, should run on a node that is not schedulable (this
doesn’t work currently on CentOS, but works on Fedora Atomic host.

#[etcd]
#<private ip of the master>

# host group for nodes, includes region info
[nodes]
master.<your domain> openshift_node_labels="{'region': 'infra', 'zone':
'default'}" openshift_public_hostname=master.<your domain>
openshift_hostname=master.<your domain>
node1.<your domain> openshift_node_labels="{'region': 'infra', 'zone':
'default'}" openshift_public_hostname=node1.<your domain>
openshift_hostname=node1.<your domain>
node2.<your domain> openshift_node_labels="{'region': 'infra', 'zone':
'default'}" openshift_public_hostname=node2.<your domain>
openshift_hostname=node2.<your domain>
```

DNS Configuration

Add DNS A records in your DNS:

Assign a wildcard DNS entry as *.cloudapps.<your domain> to the public floating IP of your master node as specified in the ansible hosts file (A record).

Assign master.<your domain> to the public floating IP of your master node in DNS (as did for *.cloudapps)
Assign node1.<your domain> to the floating IP of your node1 in DNS
Assign node2.<your domain> to the floating IP of your node2 in DNS

That's it for DNS settings.

Note: you can use the magic xip.io domain service that provides wildcard DNS for any IP address.

Provision your OpenShift Origin Cluster on OpenStack

So, now the exciting moment comes to run our ansible deployment and provision our cluster on master and the 2 worker nodes with the playbook!

Jump on your ansible node and run the ansible playbook, the playbook reads the instructions from your ansible inventory file and executes several tasks such as downloading docker images, configuring the overly network and much more:

```
[root@ansible ~]# cd openshift-ansible/
[root@ansible]# ansible-playbook ~/openshift-ansible/playbooks/byo/config.yml
```

The initial deployment needs approx. 10-15 minutes to finish, if something goes wrong by your fist install, don't panic, we'll get it fixed, please refer to troubleshooting section.

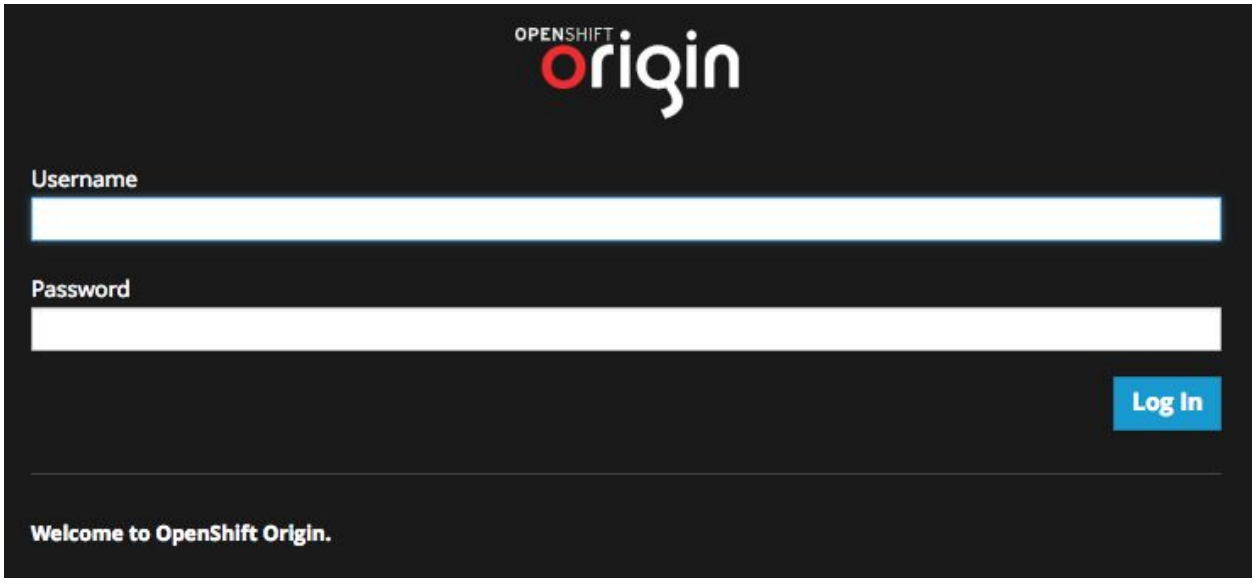
At the end of the installation you should see something like this (the output is truncated):

```
…..
PLAY RECAP
*********************************************************************
localhost                  : ok=12   changed=0    unreachable=0    failed=0
master.<your domain>     : ok=439  changed=114  unreachable=0    failed=0
node1.<your domain>      : ok=153  changed=49   unreachable=0    failed=0
node2.<your domain>      : ok=153  changed=49   unreachable=0    failed=0
```

Verify your installation

You should now be able to login to the OpenShift web console through:

```
https://master.<your domain>:8443
```

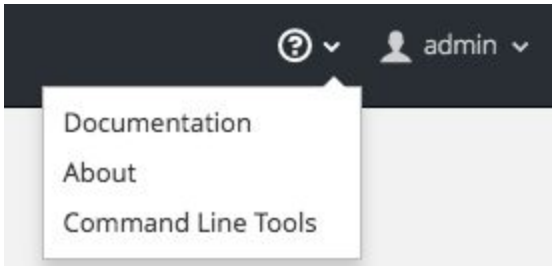


Note: the admin user password is OriginAdmin, the user password is OriginUser
You can change the password of the users with:

```
[root@origin-master ~]# htpasswd /etc/origin/master/htpasswd admin
[root@origin-master ~]# htpasswd /etc/origin/master/htpasswd user
```

Please verify your hopefully successful deployment with the following OpenShift Client “oc” commands and the Kubernetes “kubectl” control command on your master node.

Note: You can download the client tool by visiting the About or the Command Line Tools in the web console:



```
[root@master ~]# oc login -u system:admin
...


[root@master ~]#  kubectl cluster-info dump
…

[root@master ~]# oc status

[root@master ~]# oc version
oc v1.3.1
kubernetes v1.3.0+52492b4
features: Basic-Auth GSSAPI Kerberos SPNEGO
Server https://master.<your domain>:8443
openshift v1.3.1
kubernetes v1.3.0+52492b4


[root@master ~]# oc status
In project openshift on server https://master.<your domain>:8443

You have no services, deployment configs, or build configs.
Run 'oc new-app' to create an application.
[root@master ~]# oc get nodes
NAME                   STATUS                        AGE
master.<your domain>   Ready,SchedulingDisabled   1m
node1.<your domain>    Ready                         1h
node2.<your domain>    Ready                         1h


[root@master ~]# oc get users
[root@master ~]# oc get identity
```

Other useful commands are:

```
[root@master ~]# oc status
[root@master ~]# oc get svc
[root@master ~]# oc get pods
[root@master ~]# oc logs <pod name>
[root@master ~]# oc rsh <pod name> # remote shell to container
```

Last but not least, see how kubernetes rock & rolls:

```
[root@master ~]# kubectl get services --all-namespaces
NAMESPACE      NAME          CLUSTER-IP           EXTERNAL-IP    PORT(S)
default        docker-registry      172.30.243.53          <none>
5000/TCP
```

```
default         kubernetes            172.30.0.1                <none>
443/TCP,53/UDP,53/TCP
default         registry-console      172.30.223.19          <none>
9000/TCP
default         router                        172.30.58.172          <none>
80/TCP,443/TCP,1936/TCP
```

And enjoy Kolla Kubernetes (coming soon)

```
[root@origin-master ~]# kubectl get namespace
NAME                STATUS    AGE
default             Active    4d
kolla               Active    15s
kube-system         Active    4d
logging             Active    4d
```

Configuration

To run your own docker containers which need to run with the root privileges, you need to [manage the security context constraints] and run at least the following commands:

```
[root@master ~]# oadm policy add-scc-to-user anyuid -z default
[root@master ~]# oc policy add-role-to-user admin admin  -n default
```

On the ansible node run:

```
[root@ansible-terraform ~]# ansible masters -a '/usr/local/bin/oadm policy
add-cluster-role-to-user cluster-admin admin'

[root@master ~]# oc edit scc restricted
```

Change:

```
runAsUser:
  type: MustRunAsRange

To:

runAsUser:
  type: RunAsAny
```

Create Persistent Volumes and Persistent Volume Claims

Workloads such as Databases need persistence, to get DB containers running properly we need to create persistent volumes and persistent volume claims. Let's say we want to provide a persistent volume for Mariadb which shall use our nfsshare on our nfsshsre server with the following settings (please create the yaml file as follow):

```
[root@master ~]# vi maria-db.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mariadb
spec:
  capacity:
    storage: 10Gi
  accessModes:
  - ReadWriteOnce
  nfs:
    path: /nfsshare
    server: nfsshare
  persistentVolumeReclaimPolicy: Recycle
```

And run "oc create" to create the persistent volume:

```
[root@origin-master ~]# oc create -f maria-db.yaml
persistentvolume "mariadb" created
```

You can either use the web console (under storage) to create the persistent volume claim, or create the maria-db-claim yaml file as follow and create it:

Note: if you create a new service named "mariadb", the persistent volume claim will be created automatically and bound to the persistent volume "mariadb" created above.

So, this step is optional (please don't use it for now):

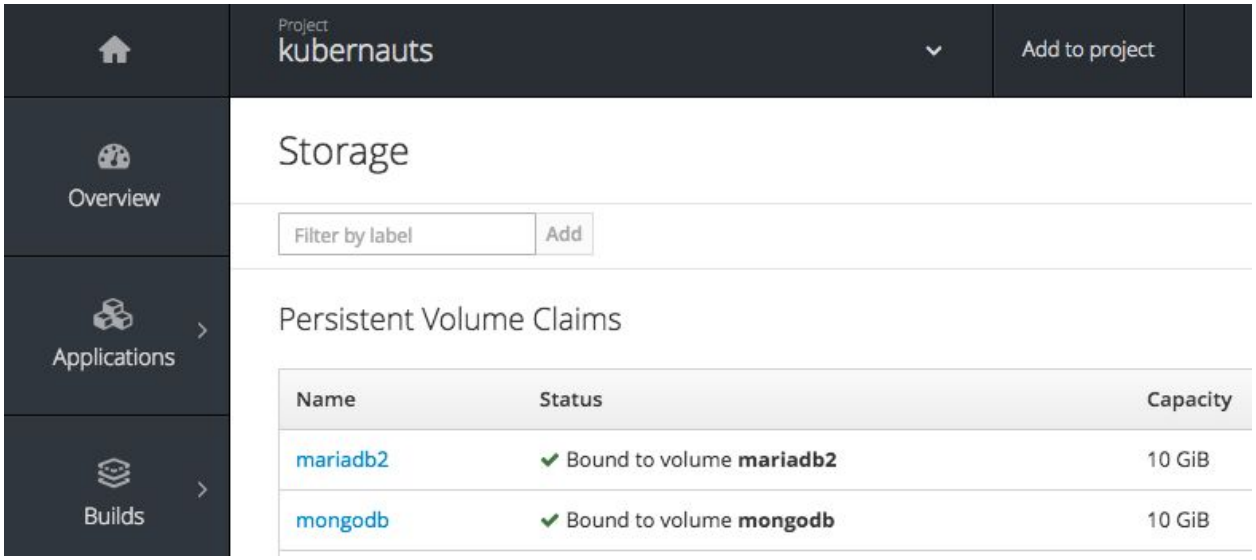
```
[root@origin-master ~]# cat mariadb-claim.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mariadb
spec:
  accessModes:
    - ReadWriteOnce
```

```
  resources:
    requests:
      storage: 5Gi
```

Create the mariadb persistent volume claim by hand:

```
[root@origin-master ~]# oc create -f mariadb-claim.yaml
persistentvolumeclaim "mariadb" created
```

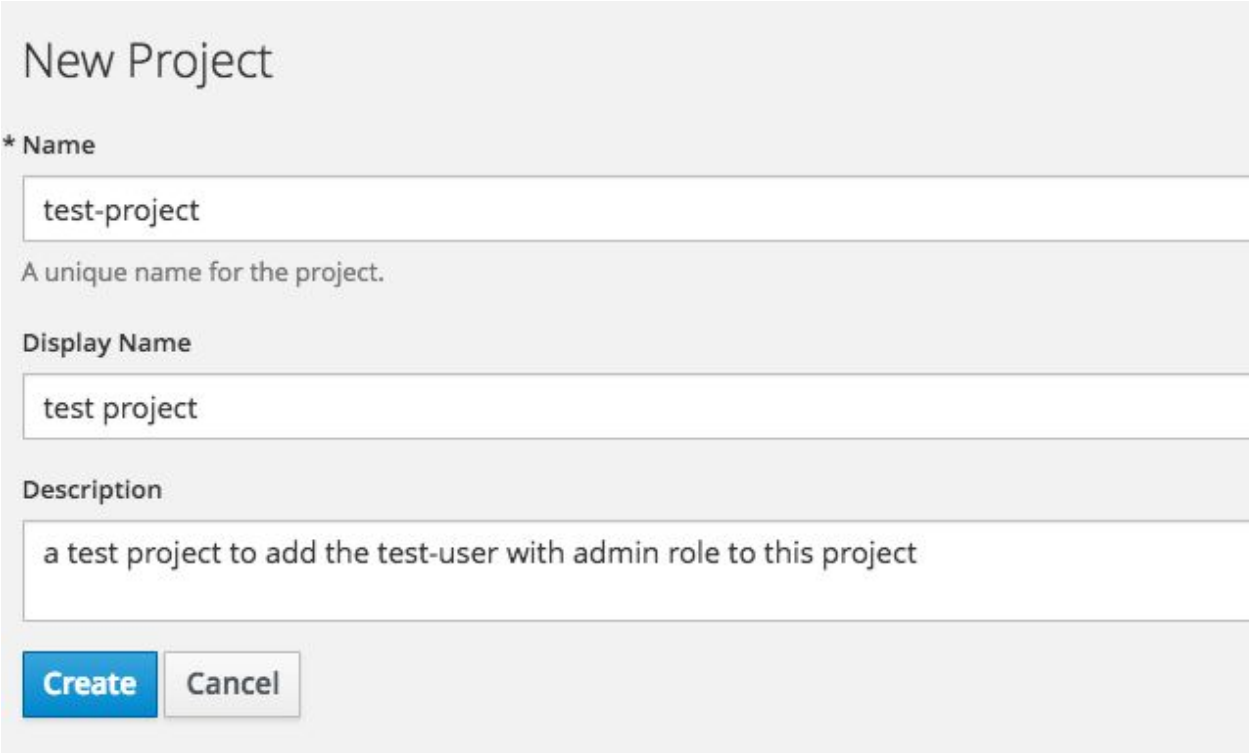
In the web console unter Storage you shall now see your Persistent Volume Claim:



Now click “Add to project” and browse the catalog and select the mariadb-persistent in the catalog, name it mariadb and run it (if you’ve created the claim by hand, you’ll get a warning that the claim already exists, you can ignore it safely).

Adding Users to a project, assign roles, change password

In the Websonsole create a test-project:



Switch to the project test project

```
[root@origin-master ~]# oc project test-project
```

Create a user named test-user

```
[root@origin-master ~]# oc create user test-user  --full-name="Test User"
user "test-user" created
```

Verify if the user has been created properly

```
[root@origin-master ~]# oc get users
NAME         UID                                    FULL NAME   IDENTITIES
admin        1126c0f3-c6d9-11e6-b1cb-fa163e3f8644
htpasswd_auth:admin
test-user    b31c4b74-c954-11e6-9c7f-fa163e3f8644   Test User
user         85b33260-c6e1-11e6-b1cb-fa163e3f8644
htpasswd_auth:user
```

Assign the admin role to the test-user for the project / namespace test-project:

```
[root@origin-master ~]# oc policy add-role-to-user admin test-user -n
test-project
```

Now we want to allow the user to run containers which require root permissions (for instance the mariadb persistent container) .

Note: to be honest, I don't think that's a good idea to allow users to run containers as root and would recommend to change the container to be able to run containers without requiring root permissions and I guess OpenShift online doesn't allow theses settings.

We'd highly recommend to read this great blog post "Getting any Docker image running in your own OpenShift cluster" written by Chris Milsted.

To allow the user (our test-user) which is the current user to run containers in our current namespace (project / context) as root, you might want to run one of the following commands:

```
[root@origin-master ~]# oadm policy add-scc-to-user anyuid -z default
```

Or use:

```
[root@origin-master ~]# oadm policy add-scc-to-user anyuid
system:serviceaccount:test-project:default
```

The outcome of both above commands are the same, if you use:

```
[root@origin-master ~]# oc edit scc anyuid
```

You'll see in the users section, that the following line was added

```
users:
- system:serviceaccount:test-project:default
```

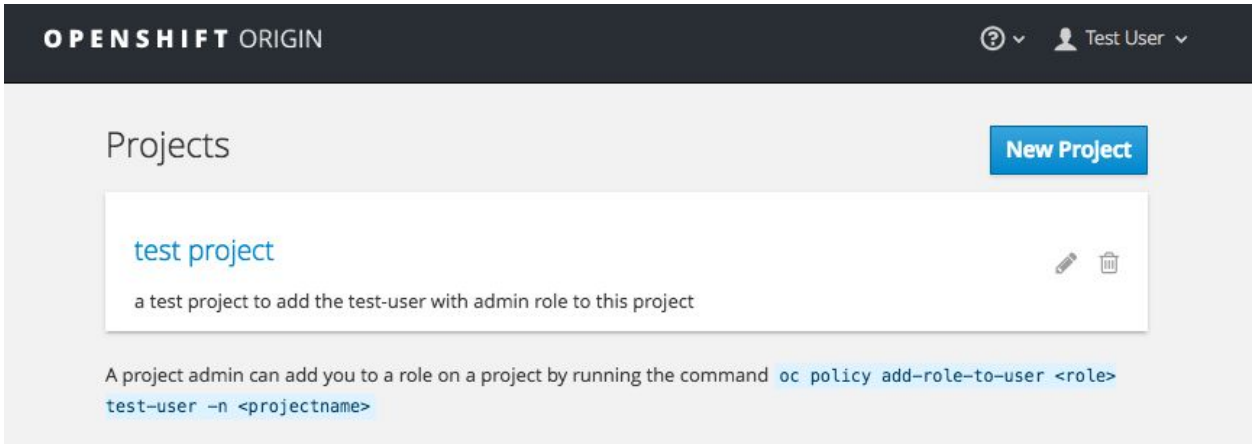
Find where the htpasswd file exists:

```
[root@origin-master ~]# grep htpasswd /etc/origin/master/master-config.yaml
    name: htpasswd_auth
      file: /etc/origin/master/htpasswd
```

Set the password for the test-user

```
[root@origin-master ~]# htpasswd /etc/origin/master/htpasswd test-user
```

Log into the web console with the test-user credentials:



Test if the test-user can create a mariadb named mydb, create a persistent volume first and create the mydb from mariadb catalog in the dashboard:

```
[root@master ~]# cat mydb.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mydb
spec:
  capacity:
    storage: 10Gi
  accessModes:
  - ReadWriteOnce
  nfs:
    path: /nfsshare
    server: nfsshare
  persistentVolumeReclaimPolicy: Recycle
```

```
[root@master ~]# oc create -f mydb.yaml
```

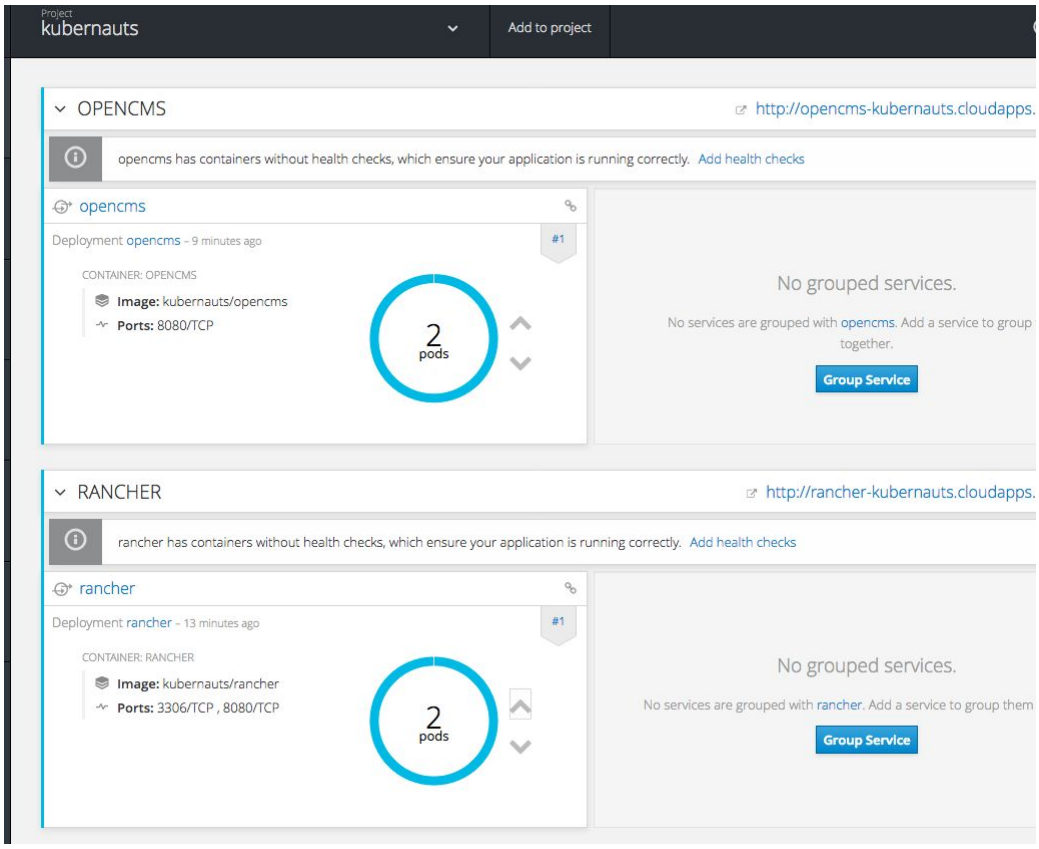
To get the logs, first find the pod and then use “oc logs <pod name>” to get the logs:

```
[root@origin-master ~]# oc get pods
NAME             READY     STATUS    RESTARTS   AGE
mydb-1-deploy    1/1       Running   0          3m
mydb-1-eki2z     0/1       Running   2          3m

[root@origin-master ~]# oc logs mydb-1-eki2z
….

2016-12-23 21:48:58 140119696226496 [ERROR] InnoDB: Unable to lock ./ibdata1,
error: 11
2016-12-23 21:48:58 140119696226496 [Note] InnoDB: Check that you do not
already have another mysqld process using the same InnoDB data or log files.
2016-12-23 21:48:58 140119696226496 [Note] InnoDB: Retrying to lock the first
data file
---> 21:48:59     Waiting for MySQL to start ...
```

Watch the logs:

```
[root@origin-master ~]# watch oc logs mydb-1-eki2z
….
```

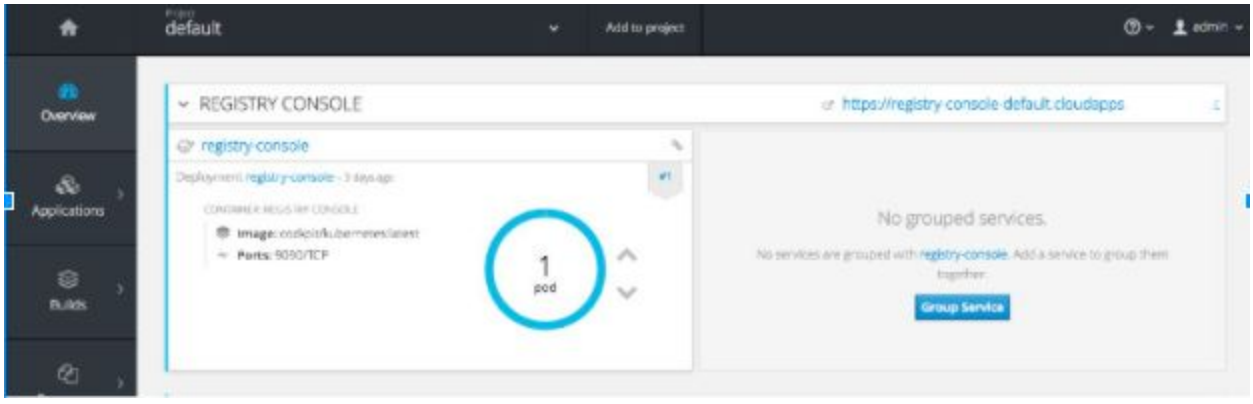
Your Origin Dashboard should look similar to this:



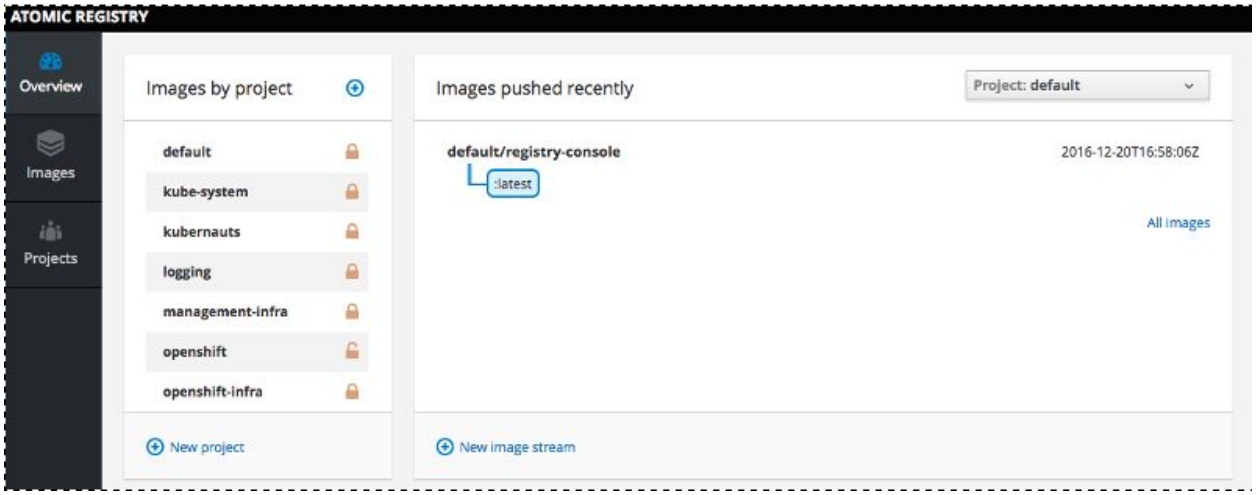
Access the Atomic Registry

After logging into the system and selecting the default project, you should see the registry console with the link to registry console:

https://registry-console-default.cloudapps.**<your domain>**



The Atomic Registry Console provides the Overview, Images and Projects section with some login and image commands which help you get started by pulling and pushing your docker images to the registry:



Run your own Docker Containers

You can run any docker containers from your own images on OpenShift by pushing your own image streams to the registry and running a container in a pod from that image in different projects.

```
$ docker pull docker.io/myimage:latest
$ docker tag myimage docker-registry-default.cloudapps<your domain>/project/name:tag
```

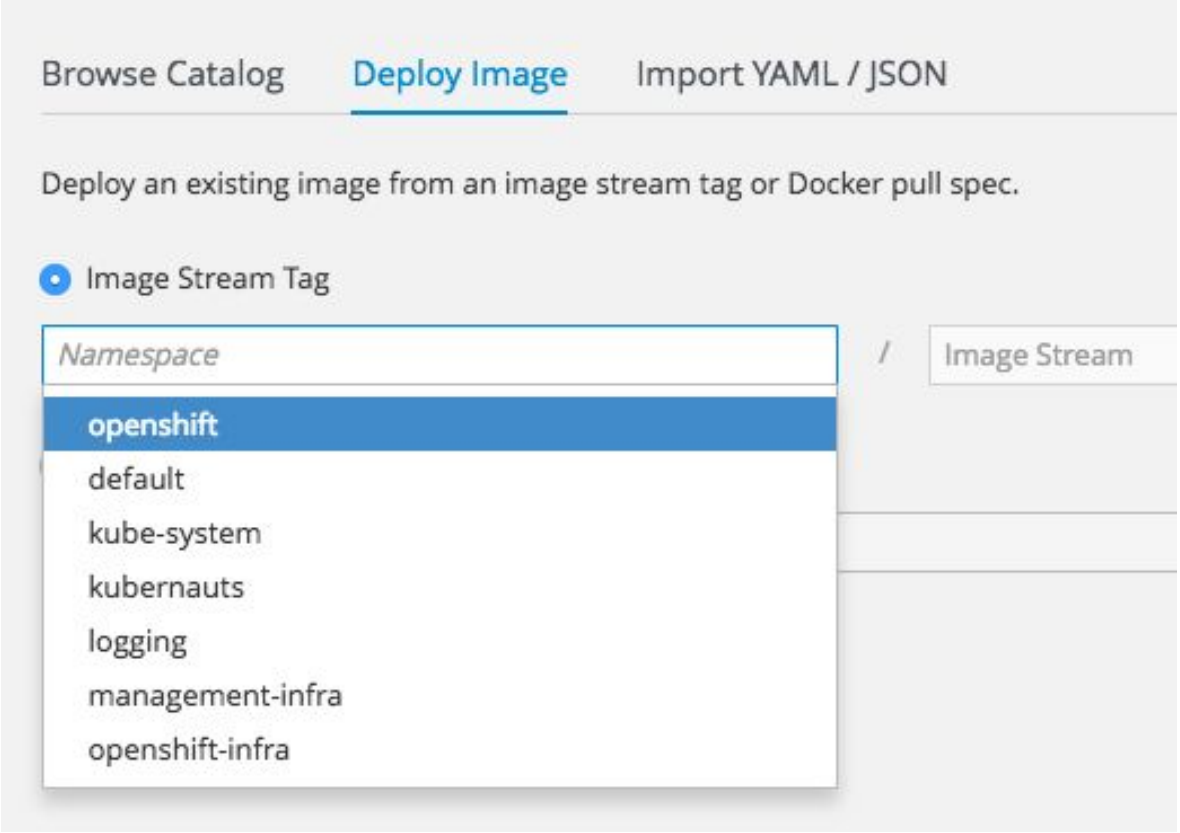
You should log into the registry before pushing images please substitute <your domain> with your own domain name:

```
$ docker login -p XdJHktF7hiMpnvnCcrDVqwqwWQQQnx30P0C7meXHRQnTf_I -e unused -u unused docker-registry-default.cloudapps<your domain>
```

And push your image to the registry:

```
$ docker push docker-registry-default.cloudapps<your domain>/project/name
```

In the web console under your project, please click on “Add to project” and select the “Deploy Image” tab and select your projects namespace, the image stream and the tag:



Example I: Let's Run Rancher on OpenShift

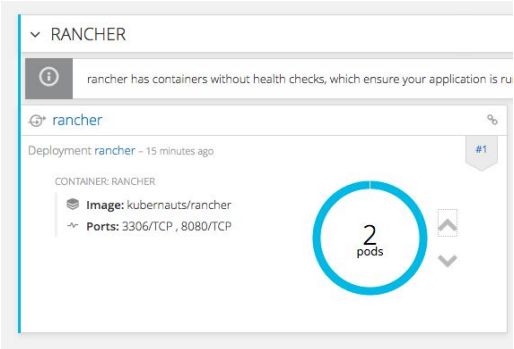
Pull, Push and Run Rancher on OpenShift:

```
[root@origin-master ~]# docker pull rancher/server
```

```
[root@origin-master ~]# docker tag docker.io/rancher/server
docker-registry-default.cloudapps<your domain>/<your project>/rancher:latest
```

```
[root@origin-master ~]# docker push docker-registry-default.cloudapps.<your
domain>/<your domain>/<your project>/rancher
```

Create a router and assign the RANCHER service to it, and you're ready to discover the world of Rancher on OpenShift and deploy Kubernetes, Mesos, Cattle and Docker Swarm on Amazon EC2, DigitalOcean, GCP and more!



Example II: Run the Zeppelin IoT App on OCP

To see if your OpenShift Origin environment is battle proven, let us install the Internet of Things (IoT) OpenShift Demo Project from Github.

Ths project installs the Apache Zeppelin, which provides a web-based notebook that enables interactive data analytics. You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

For our environment with CentOS, you need to create to PVs (Physical Volumes) and pass the --zeppelin-base=centos to the init.sh script:

```
ò _al [dgf] `llhkÆ×× al`mZÇ[ge×kYZj]úùýú×aglÈg[hÇ al
ò [\ aglÈg[h
ò Ç×afalÇk` ÈÈr]hh]dafÈZYk]đ[]flgk
```

The deployment will need near to 30 minutes to complete.

On the NFS Share we created two folders “postgresql” and “rhel-zeppelin” under /nfsshare folders and assigned the PVs to that path:

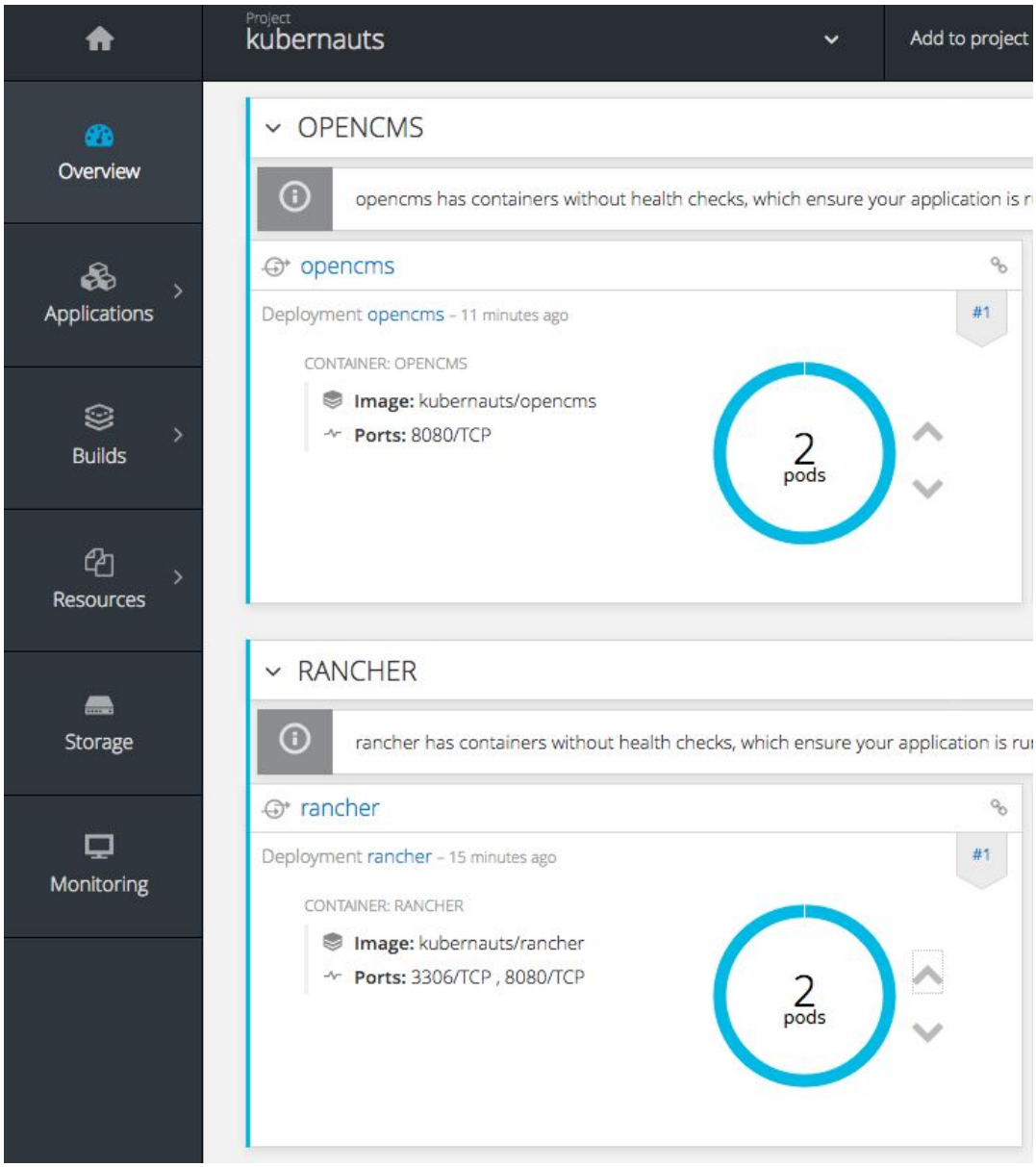
```
[root@origin-master iot-ocp]# cat postresql.yml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: postgresql
spec:
  capacity:
    storage: 10Gi
  accessModes:
  - ReadWriteOnce
  nfs:
    path: /nfsshare/postgresql
    server: nfsshare
  persistentVolumeReclaimPolicy: Recycle
```

```
[root@origin-master iot-ocp]# cat rhel-zeppelin.yml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: rhel-zeppelin
spec:
  capacity:
    storage: 10Gi
  accessModes:
  - ReadWriteOnce
  nfs:
    path: /nfsshare/rhel-zeppelin
    server: nfsshare
  persistentVolumeReclaimPolicy: Recycle
```

Scaling Your Pods

Scaling up or down your Container Pods is easy on OpenShift, but this makes sense only if your apps are cloud aware and can leverage the scaling capabilities on OpenShift, to see if scaling works for your apps, use the up and down arrows near to the pods circle or use the CLI:

[root@origin-master ~]# oc scale dc opencms **--replicas=2**
[root@origin-master ~]# oc scale dc rancher **--replicas=2**



Scaling Up Your Cluster for HA

Adding Master and Worker Nodes

For high availability and resiliency you need to extend your cluster by adding 2 more master nodes and some additional nodes if you run out of capacity.

Please follow the [Adding Hosts Using the Advanced Install page] for adding additional hosts.

The only issue is, that the first command:

```
ò qme mh\Yl] Ylgea[Ègh]fk`a^lÈmladk
```

doesn't work on the ansible node, you need to copy over the "openshift-ansible-centos-paas-sig.repo" file and the GPG key from the master node to your ansible node:

```
[root@origin-master ~]# scp
/etc/yum.repos.d/openshift-ansible-centos-paas-sig.repo
10.0.1.31:/etc/yum.repos.d/

[root@origin-master ~]# scp /etc/pki/rpm-gpg/openshift-ansible-CentOS-SIG-PaaS
10.0.1.31:/etc/pki/rpm-gpg/
```

(Please substitute the IP with the IP of your ansible node)

And install the atomic openshift utils:

```
[root@ansible ~]# yum install -y  atomic-openshift-utils
```

Added 2 more instances for a second master and a worker nodes, extended the ansible inventory file as described in the adding host guide, installed docker and attached 2 volumes to each, extended the /etc/hosts file on all nodes and followed the adding host guide, the deployment couldn't complete, in our case the sshd was dead on the first master node, we'd to enable sshd.socket.

Note: after adding the new hosts make sure that you can ssh from every host to another hosts with the FQDN of the hosts without any warnings such as:

```
[root@ansible ~]# ssh master
The authenticity of host 'master (10.0.1.41)' can't be established.
ECDSA key fingerprint is d9:3f:7d:d9:64:e5:95:ee:1f:82:11:b2:1b:8d:79:a8.
Are you sure you want to continue connecting (yes/no)?
```

Before scaling up your cluster, please set the following lines in your ansible inventory file in the [OSEv3:vars] group:

```
# Cluster method for master (native or pacemaker)
openshift_master_cluster_method=native

# router and registry selector (for HA deployment)
openshift_router_selector='router=true'
openshift_registry_selector='registry=true'
```

We enabled the etcd section in our inventory file as well:

```
[OSEv3:children]
masters
nodes
etcd

[etcd]
master.<your domain>
master2.<your domain>
```

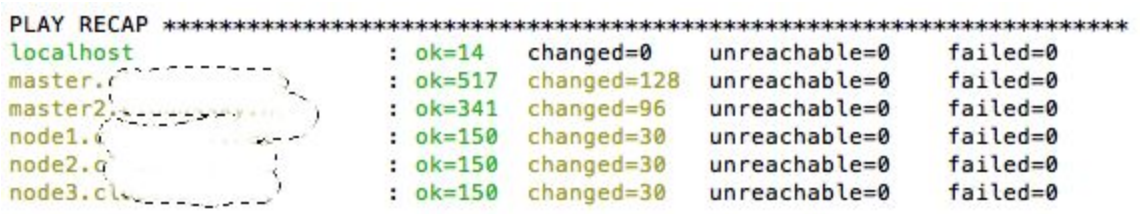
Run the scaleup playbook:

```
[root@ansible ~]# ansible-playbook
/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-node/scaleup.yml
```

After the scaleup action you should see something like this:

```
PLAY RECAP *********************************************************************
localhost                  : ok=10   changed=6    unreachable=0    failed=0
master.                    : ok=18   changed=2    unreachable=0    failed=0
master2.                   : ok=121  changed=5    unreachable=0    failed=0
node1.                     : ok=1    changed=0    unreachable=0    failed=0
node2.                     : ok=1    changed=0    unreachable=0    failed=0
node3.                     : ok=122  changed=5    unreachable=0    failed=0
```

We did an uninstall to see if the new playbook works with 2 masters and 3 worker nodes, and it did (please refer to the final playbook in appendix).



Verify your HA deployment

```
[root@master ~]# oc get nodes
NAME                         STATUS                     AGE
master.<your domain>     Ready,SchedulingDisabled   49m
master2.<your domain>    Ready,SchedulingDisabled   49m
node1.<your domain>      Ready                      49m
node2.<your domain>      Ready                      49m
node3.<your domain>      Ready                      49m
```

Verify your etcd cluster member list and cluster health:

```
[root@master ~]# etcdctl -C \
    https://master.<your domain>:2379,https://master2.<your domain>:2379 \
    --ca-file=/etc/origin/master/master.etcd-ca.crt \
    --cert-file=/etc/origin/master/master.etcd-client.crt \
    --key-file=/etc/origin/master/master.etcd-client.key member list
```

```
[root@master ~]# etcdctl -C \
    https://master.<your domain>:2379,https://master2.<your domain>:2379 \
    --ca-file=/etc/origin/master/master.etcd-ca.crt \
    --cert-file=/etc/origin/master/master.etcd-client.crt \
    --key-file=/etc/origin/master/master.etcd-client.key cluster-health
```

Troubleshooting

Service restart for master and nodes

If something goes wrong or you change some configuration settings, it might be useful to know how to start or restart OpenShift Origin services on all master and node hosts to apply your configuration changes:

```
# systemctl restart origin-master
# systemctl restart origin-node
```

Note: in an HA environment use:

```
#  systemctl status origin-master-api
#  systemctl status origin-master-controllers
```

Uninstall

By some installations, the deployment failed on master after adding a new node or changing some config files, in this case the Uninstall is very helpful.

```
$ ansible-playbook ~/openshift-ansible/playbooks/adhoc/uninstall.yml
```

Examine your Kubernetes Cluster health

On the master node run:

```
# kubectl cluster-info dump
```

The output is very useful for troubleshooting, if something goes wrong, you'll find the errors in the dump and in "/var/log/messages" as well.

By the way did you noticed that the "oc" and "kubectl" command produce the same output?

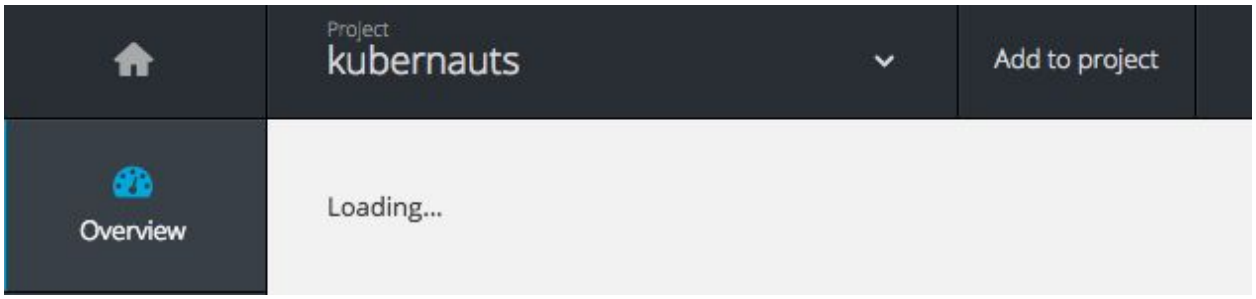
```
[root@origin-master ~]# kubectl get nodes
NAME                    STATUS                     AGE
master.<your domain>    Ready,SchedulingDisabled   2d
node1.<your domain>     Ready                      2d
node2.<your domain>     Ready                      2d
```

Welcome to Kubernetes under OpenShift ;-)

Known Issues

Web Console becomes sometimes unresponsive

The Origin web console sleeps sometimes and becomes unresponsive and needs some few minutes to wake up and keeps loading:



We’re not sure if this has something to do with the OpenStack networking, or firewall or it’s Origin related. It might be a Google Chrome Websocket issue (with FireFox we couldn't observer this issue so often).

Exited containers don’t get purged automatically

On the worker nodes containers which have the status exited don’t get purged automatically, we’d need to do that manually or via a cron job:

Note: this might be a configuration issue, about which we’re not aware yet.

Cleanup exited containers:

```
$ docker rm $(docker ps -q -f status=exited)
```

Cleanup dangling volumes:

```
$ docker volume rm $(docker volume ls -qf dangling=true)
```

Cleanup dangling images:

```
$ docker rmi $(docker images --filter "dangling=true" -q --no-trunc)
```

Origin Node fails to start

From time to time Origin Node service fails to start on the master, most likely docker.service doesn't run properly, this is due to some problems with the "Docker Storage Setup":

```
[root@origin-master ~]# journalctl -xe
-- Unit docker-storage-setup.service has begun starting up.
Jan 27 12:58:40 origin-master.novalocal docker-storage-setup[6674]: INFO: Volume group backing root filesystem could not
Jan 27 12:58:40 origin-master.novalocal docker-storage-setup[6674]: INFO: Device /dev/vdb is already partitioned and is
Jan 27 12:58:40 origin-master.novalocal docker-storage-setup[6674]: INFO: Found an already configured thin pool /dev/mapp
Jan 27 12:58:40 origin-master.novalocal docker-storage-setup[6674]: ERROR: Thin pool /dev/mapper/ does not seem to be ma
Jan 27 12:58:40 origin-master.novalocal systemd[1]: docker-storage-setup.service: main process exited, code=exited, stat
Jan 27 12:58:41 origin-master.novalocal systemd[1]: Failed to start Docker Storage Setup.
-- Subject: Unit docker-storage-setup.service has failed
-- Defined-By: systemd
-- Support: http://lists.freedesktop.org/mailman/listinfo/systemd-devel
--
-- Unit docker-storage-setup.service has failed.
--
-- The result is failed.
Jan 27 12:58:41 origin-master.novalocal systemd[1]: Unit docker-storage-setup.service entered failed state.
Jan 27 12:58:41 origin-master.novalocal systemd[1]: docker-storage-setup.service failed.
Jan 27 12:58:41 origin-master.novalocal systemd[1]: Starting Docker Application Container Engine...
-- Subject: Unit docker.service has begun start-up
-- Defined-By: systemd
-- Support: http://lists.freedesktop.org/mailman/listinfo/systemd-devel
--
-- Unit docker.service has begun starting up.
Jan 27 12:58:41 origin-master.novalocal docker-current[6713]: time="2017-01-27T12:58:41.075906029Z" level=warning msg="d
Jan 27 12:58:41 origin-master.novalocal kernel: device-mapper: table: 252:1: thin: Couldn't open thin internal device
Jan 27 12:58:41 origin-master.novalocal kernel: device-mapper: ioctl: error adding target to table
Jan 27 12:58:41 origin-master.novalocal docker-current[6713]: time="2017-01-27T12:58:41.093338498Z" level=fatal msg="Err
Jan 27 12:58:41 origin-master.novalocal systemd[1]: docker.service: main process exited, code=exited, status=1/FAILURE
Jan 27 12:58:41 origin-master.novalocal systemd[1]: Failed to start Docker Application Container Engine.
```

To fix the issue cleanup the /var/lib/docker/*:

```
[root@origin-master ~]# rm -rf /var/lib/docker/*
```

And verify the Origin Node Service is running properly:

```
[root@origin-master ~]# systemctl status origin-node.service
● origin-node.service - Origin Node
   Loaded: loaded (/usr/lib/systemd/system/origin-node.service; enabled; vendor preset: disabled)
  Drop-In: /usr/lib/systemd/system/origin-node.service.d
           └─openshift-sdn-ovs.conf
   Active: active (running) since Fri 2017-01-27 13:16:27 UTC; 54s ago
     Docs: https://github.com/openshift/origin
```

References and useful links

[1] Advanced OpenShift Origin Installation
[2] Installing an OpenShift Origin Cluster on Fedora 25 Atomic
[3] Getting any Docker image running in your own OpenShift cluster
[4] OpenShift On OpenStack Presentation (by Diane Müller & Daneyon Hansen)
[5] OpenShift 3.3 Piplelines - Deep Dive

Getting help

If you've any questions, please comment on a section on this page or join the openshiftcommons slack community and ask your questions or provide your much appreciated feedback there.

Appendix

The final Ansible inventory file

```
# Create an OSEv3 group that contains the masters and nodes groups
[OSEv3:children]
masters
nodes
etcd
#new_nodes
#new_masters

# Set variables common for all OSEv3 hosts
[OSEv3:vars]
# SSH user, this user should allow ssh based auth without requiring a passwor
ansible_ssh_user=root

# If ansible_ssh_user is not root, ansible_sudo must be set to true
#ansible_sudo=true

# router and registry selector (for HA deployment)
openshift_router_selector='router=true'
openshift_registry_selector='registry=true'

# To deploy origin, change deployment_type to origin
deployment_type=origin

# enable htpasswd authentication

#openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login':
'true', 'challenge': 'true','kind': 'HTPasswdPasswordIdentityProvider',
'filename': '/etc/openshift/openshift-passwd'}]

openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login':
'true', 'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider',
'filename': '/etc/origin/master/htpasswd'}]
openshift_master_htpasswd_users={'admin':
'$apr1$zgSjCrLt$1KSuj66CggeWSv.D.BXOA1', 'user':
'$apr1$.gw8w9i1$ln9bfTRiD6OwuNTG5LvW50'}

openshift_master_default_subdomain=cloudapps.<your domain>
openshift_master_cluster_method=native
```

```
# host group for masters
[masters]
master.<your domain> openshift_node_labels="{'region': 'infra', 'zone':
'default'}" openshift_public_hostname=master.<your domain>
openshift_hostname=master.<your domain> openshift_public_ip=xxx.xxx.xxx.xxx
master2.<your domain> openshift_node_labels="{'region': 'infra', 'zone':
'default'}" openshift_public_hostname=master2.<your domain>
openshift_hostname=master2.<your domain> openshift_public_ip=xxx.xxx.xxx.xxx

# host group for etcd, should run on a node that is not schedulable
[etcd]
#xxx.xxx.xxx.xxx
master.<your domain>
master2.<your domain>

# host group for nodes, includes region info
[nodes]
master.<your domain> openshift_node_labels="{'region': 'infra', 'zone':
'default'}" openshift_public_hostname=master.<your domain>
openshift_hostname=master.<your domain>
master2.<your domain> openshift_node_labels="{'region': 'infra', 'zone':
'default'}" openshift_public_hostname=master2.<your domain>
openshift_hostname=master2.<your domain>
node1.<your domain>
openshift_node_labels="{'router':'true','registry':'true','region': 'infra',
'zone': 'default'}" openshift_public_hostname=node1.<your domain>
openshift_hostname=node1.<your domain>
node2.<your domain>
openshift_node_labels="{'router':'true','registry':'true','region': 'infra',
'zone': 'default'}" openshift_public_hostname=node2.<your domain>
openshift_hostname=node2.<your domain>
node3.<your domain>
openshift_node_labels="{'router':'true','registry':'true','region': 'infra',
'zone': 'default'}" openshift_public_hostname=node3.<your domain>
openshift_hostname=node3.<your domain>

[new_masters]
#master2.<your domain> openshift_node_labels="{'region': 'infra', 'zone':
'default'}"

[new_nodes]
#node3.<your domain> openshift_node_labels="{'region': 'infra', 'zone':
'default'}"
#master2.<your domain> openshift_schedulable=false
```

Terraform Configs for Creating OpenShift Hosts on OpenStack

With Terraform you can spin up your master, worker and etcd nodes on OpenStack within 1-2 minutes. You can install Terraform by following this link.

The following config files from our quadrupleo Github Repo deploys 9 nodes (3 masters, 2 worker nodes, 3 etcds and the LB (HAProxy)) with a new network “openshift”, subnets, router, security group, etc. on your OpenStack cloud .You can adjust this config files as you need, by simply commenting out the masters, nodes and HAProxy or other resources such as floatingIps, etc.

Note: these config files don’t add any volumes to the hosts yet.

You can checkout the config files from Github:

```
$ git clone https://github.com/cloudssky/quadrupleo.git
$ cd quadrupleo/
$ cp terraform.tfvars.sample terraform.tfvars
```

And provide the right values for your OpenStack environment terraform.tfvars file (keep this file in a safe place):

```
$ vi terraform.tfvars
```

```
user_name = "your user name"
tenant_name = "your tenant name"
password= "your password"
auth_url = "http://<your ip/ domain>:5000/v2.0"
external_gateway = "the id of your external gateway"
image = "your image name"
pool = "external floating ip pool. default is public"
```

Check your plan:

show your plan
```
$ terraform plan
```

save your plan
```
$ terraform plan -out quadrupleo-`date +'%s'`.plan
```

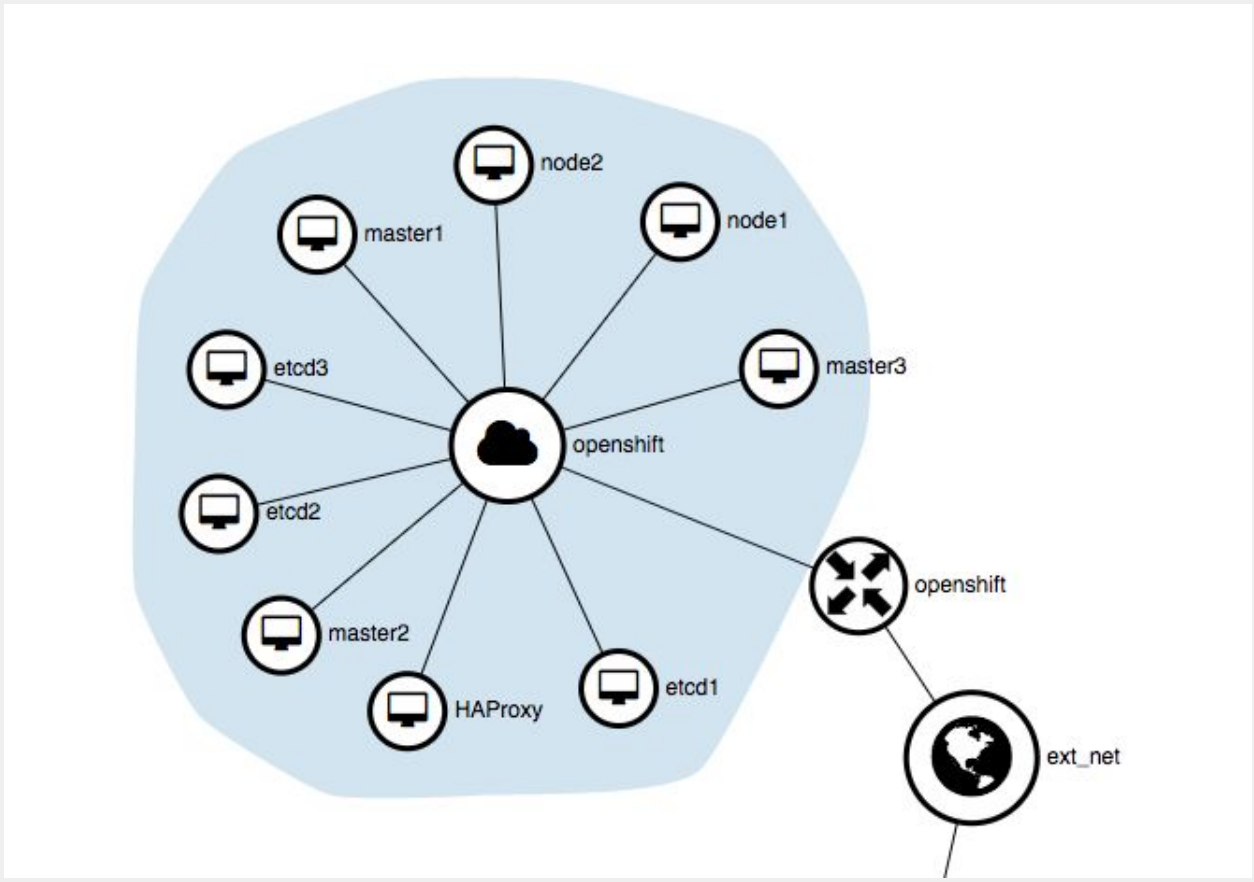
Provision your Origin base cluster

Now you're ready to provision your base cluster in less than one minute:

```
$ terraform apply
```

```
$ terraform show
```

In Horizon, under Network Topology, you should see something similar to this:



destroy your cluster (use it with caution!!!!!)

```
$ terraform destroy
```

To create the Terraform Graph, you might want to use:

```
$ terraform graph > openshift.dot
$ dot openstack.dot -Tsvg -o openshift.svg
```

You'll get something similar to this (only to show how terraform rocks):