Crunchy Containers

Origin Installion Instructions and Advice

The setup and installation of Origin is described below.

Install Openshift Origin

Binary releases of Openshift Origin are available here: https://github.com/openshift/origin/releases

Openshift install steps:

- download the binary release
- untar the release to your local host
- run as follows:

```
sudo ./openshift start
```

- set the permissions on the admin kubeconfig file to allow changes

```
sudo chmod 666 ./openshift.local.config/master/admin.kubeconfig
```

- add the following to your bash shell environment to allow you access to the openshift admin account:

```
export KUBECONFIG="$(pwd)"/openshift.local.config/master/admin.kubeconfig
export CURL_CA_BUNDLE="$(pwd)"/openshift.local.config/master/ca.crt
```

- edit the restricted settings, you will need to change the runAsUser type value to RunAsAny to allow the container to run as the postgres user.

```
oc login -u system:admin
oc edit scc restricted --config=./openshift.local.config/master/admin.kubeconfig
```

- login in as test user and create a project called pgproject

```
oc login -u test
oc new-project pgproject
```

Openshift starts an internal DNS server when it starts, and it registers DNS names for each service that you create. To refer to this DNS server, you adjust your /etc/resolv.conf file to include the Openshift DNS server IP as your primary name server, also you adjust the searchpath if you dont want to type '<[projectname].svc.cluster.local when you refer to the various Openshift services you create. Here is an example /etc/resolv.conf that uses an Openshift project name of 'pgproject':

```
search pgproject.svc.cluster.local crunchy.lab
nameserver 192.168.122.71
nameserver 192.168.122.1
```

This example used an Openshift project name of 'pgproject'. The project name is used as part of the DNS names set by Openshift for Services.

Origin Prerequisites

NFS is required for some of the Openshift examples, those dealing with backups and restores will require a working NFS.

NFS is able to run in selinux Enforcing mode if you following the instructions here:

https://github.com/openshift/origin/tree/master/examples/wordpress

Other information on how to install and configure an NFS share is located here:

http://www.itzgeek.com/how-tos/linux/centos-how-tos/how-to-setup-nfs-server-on-centos-7-rhel-7-fedora-22.html

Examples of Openshift NFS can be found here:

https://github.com/openshift/origin/tree/master/examples/wordpress/nfs

The examples specify a test NFS server running at IP address 192.168.0.103

On that server, the /etc/exports file looks like this:

```
/jeffnfs * (rw,sync)
```

if you are running your client on a VM, you will need to add 'insecure' to the exportfs file on the NFS server, this is because of the way port translation is done between the VM host and the VM instance.

see this for more details:

http://serverfault.com/questions/107546/mount-nfs-access-denied-by-server-while-mounting

Origin Tips

Tip 1: Finding the Postgresql Passwords

The passwords used for the PostgreSQL user accounts are generated by the Openshift 'process' command. To inspect what value was supplied, you can inspect the master pod as follows:

```
oc get pod pg-master-rc-1-n5z8r -o json
```

Look for the values of the environment variables:

- PG_USER
- PG_PASSWORD
- PG_DATABASE

Tip 2: Examining a backup job log

Database backups are implemented as a Kubernetes Job. A Job is meant to run one time only and not be restarted by Kubernetes. To view jobs in Openshift you enter:

```
oc get jobs
oc describe job backupjob
```

You can get detailed logs by referring to the pod identifier in the job 'describe' output as follows:

```
oc logs backupjob-pxh2o
```

Tip 3: Backup Lifecycle

Backups require the use of network storage like NFS in Openshift. There is a required order of using NFS volumes in the manner we do database backups.

So, first off, there is a one-to-one relationship between a PV (persistent volume) and a PVC (persistence volume claim). You can NOT have a one-to-many relationship between PV and PVC(s).

So, to do a database backup repeatably, you will need to following this general pattern: * as openshift admin user, create a unique PV (e.g. backup-pv-mydatabase) * as a project user, create a unique PVC (e.g. backup-pvc-mydatabase) * reference the unique PVC within the backup-job template * execute the backup job template * as a project user, delete the job * as a project user, delete the pvc * as openshift admin user, delete the unique PV

This procedure will need to be scripted and executed by the devops team when performing a database backup.

Tip 4: Persistent Volume Matching

Restoring a database from an NFS backup requires the building of a PV which maps to the NFS backup archive path. For example, if you have a backup at /backups/pg-foo/2016-01-29:22:34:20 then we create a PV that maps to that NFS path. We also use a "label" on the PV so that the specific backup PV can be identified.

We use the pod name in the label value to make the PV unique. This way, the related PVC can find the right PV to map to and not some other PV. In the PVC, we specify the same "label" which lets Kubernetes match to the correct PV.

Tip 5: Restore Lifecycle

To perform a database restore, we do the following: * locate the NFS path to the database backup we want to restore with * edit a PV to use that NFS path * edit a PV to specify a unique label * create the PV * edit a PVC to use the previously created PV, specifying the same label used in the PV * edit a database template, specifying the PVC to be used for mounting to the /backup directory in the database pod * create the database pod

If the /pgdata directory is blank AND the /backup directory contains a valid postgres backup, it is assumed the user wants to perform a database restore.

The restore logic will copy /backup files to /pgdata before starting the database. It will take time for the copying of the files to occur since this might be a large amount of data and the volumes might be on slow networks. You can view the logs of the database pod to measure the copy progress.

Tip 6: Password Mgmt

Remember that if you do a database restore, you will get whatever user IDs and passwords that were saved in the backup. So, if you do a restore to a new database and use generated passwords, the new passwords will not be the same as the passwords stored in the backup!

You have various options to deal with managing your passwords.

- externalize your passwords using secrets instead of using generated values
- manually update your passwords to your known values after a restore

Note that you can edit the environment variables when there is a 'dc' using, currently only the slaves have a 'dc' to avoid the possiblity of creating multiple masters, this might need to change in the future, to better support password management:

```
oc env dc/pg-master-rc PG_MASTER_PASSWORD=foo PG_MASTER=user1
```

Tip 7: Log Aggregation

Openshift can be configured to include the EFK stack for log aggregation. Openshift Administrators can configure the EFK stack as documented here:

https://docs.openshift.com/enterprise/3.1/install_config/aggregate_logging.html

Tip 8: nss_wrapper

If an Openshift deployment requires that random generated UIDs be supported by containers, the

Crunchy containers can be modifed similar to those located here to support the use of nss_wrapper to equate the random generated UIDs/GIDs by openshift with the postgres user:

https://github.com/openshift/postgresql/blob/master/9.4/root/usr/share/container-scripts/postgresql/common.sh

Tip 9: build box setup

golang is required to build the pgbadger container, on RH 7.2, golang is found in the 'server optional' repository and needs to be enabled to install.

golang is required to build the pgbadger container, on RH 7.2, golang is found in the 'server optional' repository and needs to be enabled to install.

Tip 10: encoding secrets

You can use kubernetes secrets to set and maintain your database credentials. Secrets requires you base64 encode your user and password values as follows:

```
echo -n 'myuserid' | base64
```

You will paste these values into your JSON secrets files for values.