Installation Guide - Crunchy Containers for PostgreSQL

Crunchy Data Solutions, Inc.

Version 1.4, 2017-04-25

Installation Instructions

The crunchy-containers can run on different environments including:

- standalone docker
- openshift enterprise
- kubernetes 1.2.X

In this document we list the basic installation steps required for these environments.

Project Setup

This instruction assumes you are installing on Centos 7 or RHEL 7.

Assumptions

The install assumes the following about your host configuration:

- centos7 (or RHEL7) VM
- user called someuser has been created
- someuser has sudo privileges with no password

Step 1 - Project Directory Structure

First add the following lines to your .bashrc file to set the project paths:

```
export GOPATH=$HOME/cdev
export GOBIN=$GOPATH/bin
export PATH=$PATH:$GOBIN
export CCP_BASEOS=centos7
export CCP_PGVERSION=9.5
export CCP_VERSION=1.2.8
export CCP_IMAGE_TAG=$CCP_BASEOS-$CCP_PGVERSION-$CCP_VERSION
export CCPROOT=$GOPATH/src/github.com/crunchydata/crunchy-containers
```

You will then need to restart your computer for the changes to your .bashrc file to take effect.

Next, set up a project directory structure and pull down the project:

```
mkdir $HOME/cdev $HOME/cdev/src $HOME/cdev/pkg $HOME/cdev/bin
cd $GOPATH
sudo yum -y install golang git docker
go get github.com/tools/godep
cd src/github.com
mkdir crunchydata
cd crunchydata
git clone https://github.com/crunchydata/crunchy-containers
cd crunchy-containers
git checkout 1.4.0
godep restore
```

If you are a Crunchy enterprise customer, you will place the CRUNCHY repo key and yum repo file into the $CCPROOT/conf directory at this point.

Step 2 - Install the Host Dependencies

Next, install system dependencies:

```
sudo yum -y update
sudo groupadd docker
sudo usermod -a -G docker someuser
```

Remember to log out of the **someuser** account for the docker group to be added to your current session. Adding **docker** group to your user account allows you to run docker commands from your user account.

```
su - someuser
```

You can ensure your **someuser** account is added to the **docker** group by running the following command and ensuring **docker** appears as one of the results:

```
groups
```

Before you start docker, you might consider configuring docker storage: This is described if you run:

```
man docker-storage-setup
```

Next, we enable and start up docker:

```
sudo systemctl enable docker.service
sudo systemctl start docker.service
```

Step 3 - Build the Containers

At this point, you have a decision to make - either download prebuilt containers from dockerhub, **or** build the containers on your local host.

To download the prebuilt containers, make sure you can login to dockerhub.com, and then run the following:

```
docker login
cd $CCPROOT
./bin/pull-from-dockerhub.sh
```

Or if you'd rather build the containers from source, perform a container build as follows:

```
cd $CCPROOT
make setup
make all
```

After this, you will have all the Crunchy containers built and are ready for use in a **standalone docker** environment.

To install an Openshift environment, continue on to Step 4.

Step 4 - Openshift Enterprise Installation

See the OSE installation guide for details on how to install OSE on your host. The main instructions are here:

https://docs.openshift.com/enterprise/3.2/install_config/install/index.html

Step 4a - Openshift Router Setup

To install the OSE router, here are the instructions I use:

https://github.com/openshift/training/blob/master/05-Services-Routing-Complete-Example.md#routing

https://docs.openshift.com/enterprise/3.2/install_config/install/deploy_router.html

I have the commands to create the router here, **run as root user**:

```
bin/create-router.sh
```

An example of using the router:

```
oc expose service v3simple-spatial -l name=v3simple-spatial
oc get route
```

Step 4b - Openshift DNS Setup

To set up dnsmasq, here are the instructions I basically follow this set of instructions:

http://developers.redhat.com/blog/2015/11/19/dns-your-openshift-v3-cluster/

However! On my dev setup, I use KVM (libvirt) to run OSE within a VM. The libvirt already runs a dnsmasq server on the host to handle the default libvirt network which causes a conflict when trying to run another dnsmasq on the same server.

Details on libvirt and dnsmasq are here:

http://wiki.libvirt.org/page/Libvirtd_and_dnsmasq

So, here is how I modified by dnsmasq:

Uncommented the following, setting the listen-address to my eth0 IP address:

```
listen-address=192.168.0.114
bind-interfaces
host-record=osejeff.crunchy.lab,192.168.0.114
address=/apps.crunchy.lab/192.168.0.114
server=/local/127.0.0.1#53
server=/17.30.172.in-addr.arpa/127.0.0.1#53
```

Within the /etc/origin/master/master-config.yaml I use the following:

```
  bindAddress: 127.0.0.1:53
```

Also, as of OSE 3.2, be sure to place the host-record, address, server config lines BEFORE the conf-file and conf-dir lines at the end of /etc/dnsmasq.conf. If you put them before the end, they will be overwritten with the OSE dnsmasq configuration.

Your /etc/resolv.conf should look roughly like this:

```
search openshift.svc.cluster.local crunchy.lab
nameserver 192.168.0.114
nameserver 192.168.0.1
```

After this, you will have OSE listening on 127.0.0.1:53 and our special dnsmasq listening on 192.168.0.114:53. You can run the following commands to verify your DNS setup:

```
dig @127.0.0.1 single-master.openshift.svc.cluster.local
dig @192.168.0.114 single-master.openshift.svc.cluster.local
dig @192.168.0.114 osejeff.crunchy.lab
```

You can now log in as an OSE user and start creating containers:

```
oc login -u system:admin
oc project openshift
```

The most recent DNS related configuration files that I test with are included in the $CCPROOT/docs/openshift-install directory as a reference.

Step 5 - Configure Container UID Control

Openshift will run containers with a random UID by default. This can cause files written by the Postgres containers to have random UIDs if you persist data for example to an NFS fileshare. The containers will work with a random UID, but you can also override this Openshift security setting to have the containers run as the postgres UID (26).

You can make Openshift run as the Postgres UID if you set the **runAsUser** security setting to the **RunAsAny** value as follows:

```
oc login -u system:admin
oc edit scc restricted
```

Step 6 - Configure NFS for Persistence Examples

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

First, if you are running your NFS system with SELinux in enforcing mode, you will need to run the following command to allow NFS write permissions:

```
sudo setsebool -P virt_use_nfs 1
```

Next, you will need to set the permissions of your NFS path so that your pods can have write access. For the Crunchy examples, the **nfsnobody** GUI was chosen as an example. Pods will reference the **nfsnobody** GID (65534) as a security context **supplementalGroup** attribute. This setting will allow the pod to have group permissions of 65534 and therefore be able to write to the NFS persistent volumes.

The permissions on the NFS path are set as follows:

```
drwxrwx---.   3 nfsnobody nfsnobody   23 Dec 16 11:28 nfsfileshare
```

Most of the Crunchy containers run as the postgres UID (26), but you will notice that when **supplementalGroups** are specified, the pod will include the nfsnobody group in the list of groups for the pod user.

The case of Amazon file systems is different, for that you use the **fsGroup** security context setting but the idea for allowing write permissions is the same.

Here are the instructions I use when setting up NFS:

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:

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

Test your NFS configuration out by mounting a local directory:

```
mount 192.168.0.114:/nfsfileshare /mnt/nfsfileshare
```

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

Openshift 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: 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. For example:

```
subscription-manager repos --enable=rhel-7-server-optional-rpms
sudo yum -y install golang
```

Tip 9: 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.

Tip 10: DNS host entry and DeploymentConfig

If your openshift environment can not resolve your hostname via a DNS server (external to openshift!), you will get errors when trying to create a DeploymentConfig. So, you can either install dnsmasq and reconfigure openshift for that, or, you can run a DNS server on another host and add the openshift host entry to that DNS server. I use the skybridge2 Docker container for this purpose. You have to remember to adjust your /etc/resolv.conf to specify this new DNS server.

Tip 11: Setting up Docker storage

I typically set up Docker storage this way:

- add an extra IDE drive to my VM
- fdisk /dev/sd? to format the drive
- vgcreate /dev/sd?1 to create a volume group on the new drive partition
- add VG=docker-vg to /etc/sysconfig/docker-storage-setup
- run docker-storage-setup to use that new volume group

Legal Notices

CRUNCHY DATA SOLUTIONS, INC. PROVIDES THIS GUIDE "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.