InfoScale Storage & Docker

Introduction to Enterprise Persistent Storage for Containers

Carlos Carrero
Rajagopal Vaideeswaran

PRODUCT MANAGEMENT
Table of Contents

01
Introduction

02
Using InfoScale Storage & Docker

03
InfoScale Storage Plug-in for Docker

04
Engage
Introduction

+ Containers and State-full Data
+ InfoScale Storage
+ Software Defined Storage for Containers
CONTAINERS AND STATE-FULL DATA

Unlike VMs, containers typically do not have state-full data stored inside.

Containers avoid the overhead of native or hosted hypervisors making Virtual Machines unnecessary and provides a much more light weight approach. This represent a mind-set change in resource utilization and how applications are developed and operated.

While initially stateless applications were developed using container technologies, enterprise users can see the benefits of using containers for more critical applications that have a need to keep the data available, using container volumes.

The requirement to have state-full services running in an agile, dynamic and changing environment introduces new challenges to the way that data is stored and managed.

Data needs to be stored in data volumes to make it persistent

Want to run a database or other state-full data service in a container?

( *) Scalebase Research
INFOSCALE STORAGE

Veritas enables Docker to run containers with persistent data volumes in any host within a cluster, reducing storage costs by commoditizing the use of hardware and providing enterprise storage management software.

InfoScale Storage provides a Software Defined Storage solution that enables persistent storage for Dockers Containers. InfoScale capabilities allow the creation of a robust storage infrastructure where commodity servers and storage can be leveraged.

The lightweight Docker processes can run close to the storage, having all the functionalities of an Enterprise storage solution.
SOFTWARE DEFINED STORAGE FOR CONTAINERS

The advantages of using an Enterprise Storage solution defined by Software to enable data agility for Containers.

Persistent Storage Management
State-full containers need persistent storage. Veritas InfoScale Storage makes the Docker data volume persistent and accessible across any server within the cluster.

Scale-Out
Add more compute or storage as needed. By using commodity servers with server-side storage, the acquisition costs can be reduced. Get more agility and reduce operational expenses.

Snapshots & I/O Acceleration
Take a copy of the container persistent data anytime, without using additional storage. Use those copies to bring up new containers anywhere.

Disaster Recovery
Persistent data can be replicated to DR sites using traditional networks. Recovery can simply be orchestrated end-to-end.

Ecosystem Support
Use our plug-in for Docker in Technology Preview Mode to experiment how data management chores can be simplified and automated using Docker and InfoScale Storage Plug-in.

Accelerate I/O by automatically caching data on Flash devices using SmartIO.
Using InfoScale Storage & Docker

+ Define Storage Cluster Architecture
+ Docker Install
+ Storage Pool Configuration
+ Creating MySQL Container
+ Filling the Database with Persistent Data
+ Running the Container in any other Host
+ Reduce Storage with Snapshots
+ Create a Snapshot for an Existing Database
DEFINE STORAGE CLUSTER ARCHITECTURE

Using the installer script for InfoScale Storage, you can create a cluster with all the hosts available. This local storage will be added to a pool where all the hosts will have access to, no matter where the local devices reside. This will create a virtualized storage layer where state-full containers can store, share, make copies and retrieve data.

The same nodes will be running the containers. Given that the virtualized storage layer provides access to the storage from any node, the containers can be scheduled to run in any host.

If one host fails, the container can run in any other host, keeping access to the same data.

InfoScale Storage provides data resilience so any failure will not compromise data availability. InfoScale Enterprise also can manage container’s availability.

The configuration presented in this document uses three servers with local storage that are connected using InfiniBand. Containers will run in the same hosts the storage is attached to.
DOCKER INSTALL

Docker web page provides clear instructions to successfully install Docker using Red Hat 7.1

Here we provide a summary of the steps taken in our environment to have the Docker daemon up and running, so our servers will be ready to deploy any image and run any container

First we download the Docker software, then disable the firewall, start the service and make sure it will restart on boot

Docker bits need to be downloaded and enabled in every host that will be running a container

[root@target-3 ~]# curl -sSL https://get.docker.com/ | sh

[root@target-3]# systemctl mask firewalld
ln -s '/dev/null' '/etc/systemd/system/firewalld.service'
[root@target-3]# systemctl stop firewalld

[root@target-3]# sudo service docker start

[root@target-3]# sudo systemctl enable docker
**STORAGE POOL CONFIGURATION**

InfoScale Storage allows the creation of a persistent volume with the required resiliency and performance by choosing the appropriate layout.

The 12 disks that are available across our cluster will be added into a pool called pool_ssd.

From this pool, as many volumes as necessary will be carved out to provision storage to the containers. When not using the trialware plug-in, a file system has to be created.

This file system will be mounted on all the hosts, so any container can have access to the information stored there.

```
# vxdg -o fss -s init pool_ssd intel_nvme0_0 intel_nvme1_0 intel_nvme2_0 intel_nvme3_0 target-4_intel_nvme0_0 target-4_intel_nvme1_0 target-4_intel_nvme2_0 target-4_intel_nvme3_0 target-5_intel_nvme0_0 target-5_intel_nvme1_0 target-5_intel_nvme2_0 target-5_intel_nvme3_0

# vxassist -g pool_ssd -o ordered make mysql_1 300G layout=stripe-mirror alloc=target-3_intel_nvme0_0,target-3_intel_nvme1_0,target-3_intel_nvme2_0,target-3_intel_nvme3_0,target-4_intel_nvme0_0,target-4_intel_nvme1_0,target-4_intel_nvme2_0,target-4_intel_nvme3_0,target-5_intel_nvme0_0,target-5_intel_nvme1_0,target-5_intel_nvme2_0,target-5_intel_nvme3_0

# mkfs -t vxfs /dev/vx/rdsk/pool_ssd/mysql_1

# cfsmntadm add pool_ssd mysql_1 /mysql_1 all=crw

# cfsmount /mysql_1
```
CREATING A MYSQL CONTAINER

A database is a typical workload where persistent storage may be needed. A multi-tiered application consisting of web servers, application servers and a database will have to make sure the data is available. Also, probably different teams will need access to that data, so being able to reproduce that environment for testing in a very agile form may be necessary.

Our first step is to download the mysql image.

We run the mysql image creating a container called mysql_1 where we map the default /var/lib/mysql directory inside the container (where databases are stored) into previously created /mysql_1 directory within the host.

```bash
# docker pull mysql

# docker run --name mysql_1 -v /mysql_1:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=root -d mysql:5.6

463c380a208df90257ec4ba04a1d303e51c9f2c27420e1ddb2a7d5af93ef0bae

[root@target-3 ~]# docker exec -it mysql_1 bash
root@463c380a208d:/# ls -l /var/lib/mysql
total 110599
-rw-rw---- 1 mysql mysql 56 Sep 10 14:45 auto.cnf
-rw-rw---- 1 mysql mysql 50331648 Sep 10 14:45 ib_logfile0
-rw-rw---- 1 mysql mysql 50331648 Sep 10 14:45 ib_logfile1
-rw-rw---- 1 mysql mysql 12582912 Sep 10 14:45 ibdata1
drwxr-xr-x 2 mysql mysql 96 Sep 10 13:58 lost+found
drwx-------- 2 mysql mysql 3072 Sep 10 14:45 mysql
drwx-------- 2 mysql mysql 3072 Sep 10 14:45 performance_schema
```
Now we can create a new database and a table with some content. First we login to the container and execute `mysql`.

Using SQL commands we create the database, create a table and add two rows with data.

Finally we can verify how that data exists at the host level in the shared file system that is mounted across all the hosts within the cluster.

This capability will allow us to run a container in any other host and still have access to the content we have already created.

```
# docker exec -it mysql_1 bash

root@f856ef8b83b6:/# mysql -proot

mysql> create database ccarrero;
mysql> use ccarrero;
mysql> create table users (name text, age integer);
mysql> insert into users values ('Marina',4);
mysql> insert into users values ('Diego',2);

mysql> select * from users;
+--------+-------+
| name   | age   |
+--------+-------+
| Marina |  4    |
| Diego  |  2    |
+--------+-------+

# ls /mysql_1
auto.cnf  ccarrero  ibdata1  ib_logfile0  ib_logfile1
lost+found  mysql  performance_schema
```
RUNNING THE CONTAINER IN ANY OTHER HOST

The fact that the persistent file system is shared across all the hosts, allows that any container can map that persistent storage. That enables containers to run in any host.

In our example, we stop the container running on host target-3, pull a mysql image on host target-4 (as it had not been downloaded before) and we run the container mapping the persistent data to the shared storage.

We can verify that the table previously created can be accessed from this other host.

*InfoScale Storage allows containers to create and consume data from any host at any time*
Your persistent data, or gold copy of your database may have to be used across different teams. Using a Software Defined Storage solution allows taking point in time copies of that data and share it instantaneously with other teams. New containers can be run using those copies.

InfoScale Storage allows taking those copies without incurring extra storage capacity. New storage capacity will be automatically allocated only for new writes.

**Point in Time Copies allows containers to run with a private copy of the database in any host**
CREATE A SNAPSHOT FROM AN EXISTING DATABASE

1. **# fsckptadm create**
   mysql_1_snap1 /mysql_1

2. **# mkdir /mysql_1_snap1**

3. **# mount -t vxfs -o ckpt=mysql_1_snap1,cluster /dev/vx/dsk/pool_ssd/mysql_1:mysql_1_snap1 /mysql_1_snap1 /mysql_1_snap1**

   **# docker run --name mysql_1_snap1-v /mysql_1_snap1:/var/lib/mysql -d mysql**

We can create a snapshot of the current data used by container mysql_1 which resides under /mysql_1 directory.

That snapshot will be mounted under /mysql_1_snap1 and a new container will be run pointing to that new data.

Within the new container mysql_1_snap1, we can add a new row to the table users, and verify that we have the new data together with the previous table content.

If we connect to the mysql_1 container, we see that the content is not being modified and we still conserve the original information.

```sql
mysql> insert into users values ('Carlos',23);

mysql> select * from users;
```

```
+--------+
| name   |
| age    |
+--------+
| Carlos |
| 23     |
```

```sql
mysql> select * from users;
```

```
+--------+
| name   |
| age    |
+--------+
| Marina |
| 4      |
| Diego  |
| 2      |
```

© Veritas Technologies. 2015
InfoScale Storage Plug-in for Docker

+ Enhance Storage Consumption Agility for Docker
+ Steps to Configure InfoScale Storage Plug-in for Docker
+ Storage Configuration for Docker
+ Plug-in Configuration

+ Create a Container using the Veritas Plug-in
+ Fill the Database
+ Run in any other Host
+ Create Point in Time Copies
+ Run from Snapshots
ENHANCE STORAGE CONSUMPTION AGILITY FOR DOCKER

This plug-in is in Technology Preview Mode, and therefore it is not meant for production environments. It is offered to customers so they can evaluate and influence its design and functionalities using the internal Veritas Community for Containers.

Docker users should concentrate on running applications and the storage being an abstract virtualized layer for them.

Veritas is evolving a plug-in that will simplify those storage related activities, by embedding them into the Docker commands.

In this first version, when creating a container, we only have to specify the volume name.

The Storage administrator has already prepared the required infrastructure, so the volumes will be created dynamically when needed.

The beta plug-in automates some of the storage related activities, improving agility when using containers.
STEPS TO CONFIGURE INFOSCALE STORAGE PLUG-IN FOR DOCKER

The Storage or Server Administrators will create the infrastructure needed to provision storage for Docker. That means install and configure InfoScale Storage/Enterprise and create a pool of storage called dockerdg.

Docker version that integrates with the plug-in needs to be 1.8.

Once the plug-in is installed and configured, Docker users can start deploying containers.
The 12 disks that are available across our cluster will be added into a pool called dockerdg. Note that the flag “-o fss” is being used as this is a shared nothing configuration.

The Storage or Server Administrators will have to create a special first volume where metadata will be stored. A file system needs to be created and mounted across all the hosts within the cluster.

All disks that will be used for Docker will be added to a dockerdg disk group

```
# vxdg -o fss -s init dockerdg intel_nvme0_0 intel_nvme1_0
    intel_nvme2_0 intel_nvme3_0 target-4_intel_nvme0_0 target-4_intel_nvme1_0
    target-4_intel_nvme2_0 target-4_intel_nvme3_0
    target-5_intel_nvme0_0 target-5_intel_nvme1_0 target-5_intel_nvme2_0
    target-5_intel_nvme3_0

# vxassist -g dockerdg make metadata 200M

# mkfs -t vxfs /dev/vx/rdsk/dockerdg/metadata

# cfsmntadm add dockerdg metadata /dockerfs/.metadata
    all=cluster

# cfsmount /dockerfs/.metadata
```
Once you have downloaded the vx_trialware.tar.gz file, uncompress it in a directory and run the install script.

Once it is installed, modify the file /etc/vx/docker/default.conf to specify the preferred size and layout.

```
# sh install.sh
Following daemons will be started as part of installation
  vxgin
  vxvfss
  ..... installation completed successfully..
```

```
# cat /etc/vx/docker/default.conf
size=300m
layout=mirror
```

*Each volume created will be 300 MB size*

*A mirror configuration will create two copies with each copy residing in different hosts*
Once the storage has been properly configured, now when creating a container a new volume can be created on the fly. Users only need to specify the volume-driver as veritas and give a name to the volume.

In this example, we are creating a MySQL database where the default directory for database is being mounted on a new volume that will be automatically created by the plug-in.

```bash
# docker run --name mysql_c -v vol_mysql_data1:/var/lib/mysql --volume-driver veritas -e MYSQL_ROOT_PASSWORD=root -d mysql:5.6
```

Name of the volume to be created

Because the veritas driver is used, the volume will be automatically carved from the dockerdg storage pool
Now we can create a new database and a table with some content. First we login to the container and execute mysql.

Using SQL commands we create the database, create a table and add two rows with data.

This capability will allow us to run a container in any other host and still have access to the content we have already created.

```sh
# docker exec -it mysql_c bash

root@c94f0db1f5a8:/# mysql -proot

mysql> create database ccarrero;
mysql> use ccarrero;
mysql> create table users (name text, age integer);
mysql> insert into users values ('Marina',4);
mysql> insert into users values ('Diego',2);

mysql> select * from users;
+----------+
| name     | age  |
+----------+
| Marina   | 4    |
| Diego    | 2    |
+----------+
```
As we did in the previous section, we can run the container in any other host. Now, instead of a file system we are using a volume directly. But that volume is visible across all hosts, so any host will be able to run a MySQL container with that data.

```
Target-3 # docker stop mysql_1
```

```
Target-4 # docker run --name mysql_c -v vol_mysql_data1:/var/lib/mysql --volume-driver veritas -d mysql:5.6
```

```
Target-4 # docker exec -it mysql_c bash
```

```
root@5b24495cc4b6:/# mysql -proot

mysql> use ccarrero;
mysql> select * from users;
+----------+
| name | age |
+----------+
| Marina | 4 |
| Diego | 2 |
+----------+
```

Volumes created with the plug-in are accessible from any host within the cluster.
Because the plug-in initially maps volumes to the containers, we are going to use InfoScale Storage capabilities to take snapshots for any volume. We can use that snapshot for any purpose, and here we are going to demonstrate how to bring up any test database using one snapshot, without affecting the original database data.

InfoScale Storage allows full-size instant, space-optimized instant and break-off snapshots. The administrator guide contains complete details for each type of snapshot. With our configuration, we are going to use space-optimized instant snapshots to save storage capacity.

Before creating volume snapshots, the Storage or Server Admin needs to create a cache object that will be used to annotate any data modified during writes.
RUN CONTAINERS FROM SNAPSHOTS

We need a cache object that will be associated to the volume whose snapshot will be taken.

Once this preparation is done snapshots can be triggered as needed

```
# vxassist -g dockerdg make cachevol 100M
# vxmake -g dockerdg cache cacheobj cachevolname=cachevol
# vxcache -g dockerdg start cacheobj
```

```
# vxsnap -g dockerdg make source=vol_mysql_data1/newvol=vol_mysql_data1_snap1/cache=cacheobj

# docker run --name mysql_snap1 -v vol_mysql_data1_snap1:/var/lib/mysql --volume-driver veritas -d mysql:5.6
```

Original volume

Name of the snapshot volume

Specify the name of the snapshot volume
Engage

+ Evolve Persistent Storage for Containers
EVOLVE PERSISTENT STORAGE FOR CONTAINERS

Join our Veritas Community for Containers group and engage in conversations with Engineering and Product Management. Fit the products to your needs.