Best Practices Running DataStax Enterprise within Docker

July 2016

Table of Contents

Introduction

DataStax has a strategic imperative to accelerate the ability of our customers to power the exploding number of cloud applications that require data distribution across datacenters and clouds. For the past 5+ years, we have being delivering Apache Cassandra™ (with its genesis within large scale Internet companies such as Amazon, Google and Facebook) as part of a database platform purpose built for cloud applications.

Alongside Big Data's adoption within enterprises, there has been a movement to adopt agile development and deployment models utilized at Internet companies. Linux Containers have emerged as a crucial foundation for this DevOps-style application lifecycle management. This meteoric rise is highlighted below in the trend analysis of number of searches on Google[1] for containers vs. virtual machines[2].

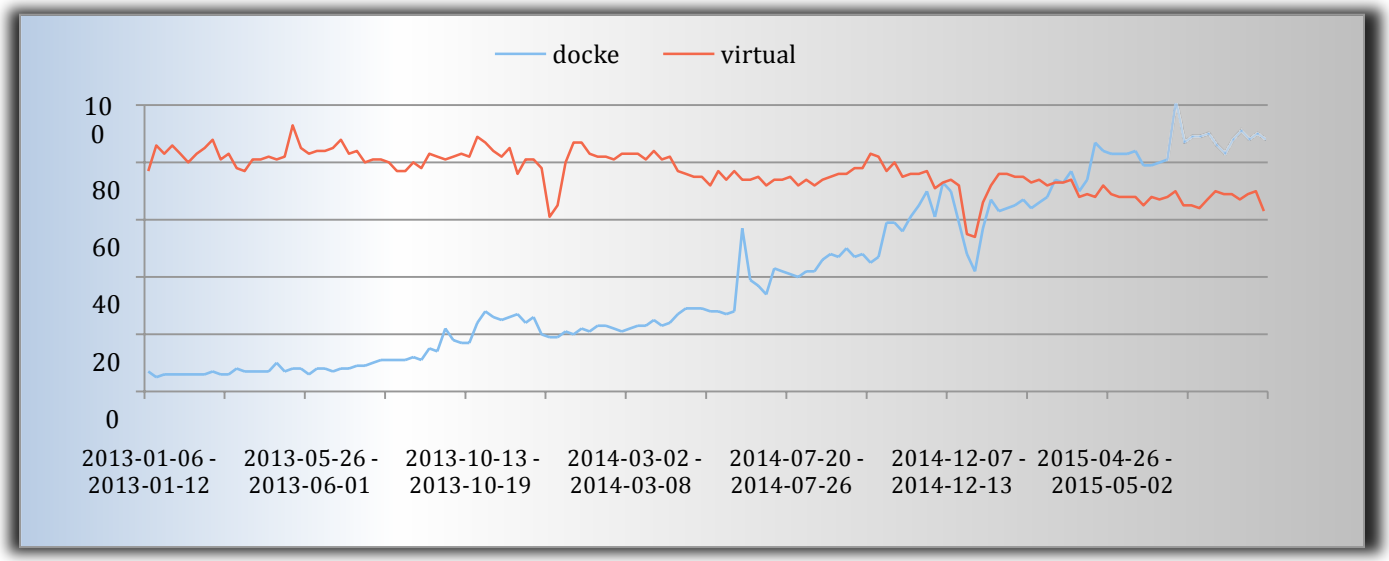

Fig 1 - Rise of container interest between 2013 and 2015; spearheaded by Docker

As is clear from the trend above, the container movement (spearheaded by Docker) has recently overtaken virtual machines in search interest and is expected to grow further. That being the case, it's no surprise that customers have been asking for guidance on how to best run DataStax software with containers. This short paper describes best practices for running DataStax Enterprise (DSE) and OpsCenter within Docker environments, and includes trade-offs to be aware of and pitfalls to avoid[3].

[1] Sourced from Google Trends
[2] VMs were the dominant model of deployment within enterprises in the mid to late 2000's
[3] Please refer to linked resources to understand the basics of Docker, Linux containers etc.

Why Docker?

Before proceeding further, it's prudent to understand the evolution of application development and deployment paradigms in the past few decades as shown in diagram below.

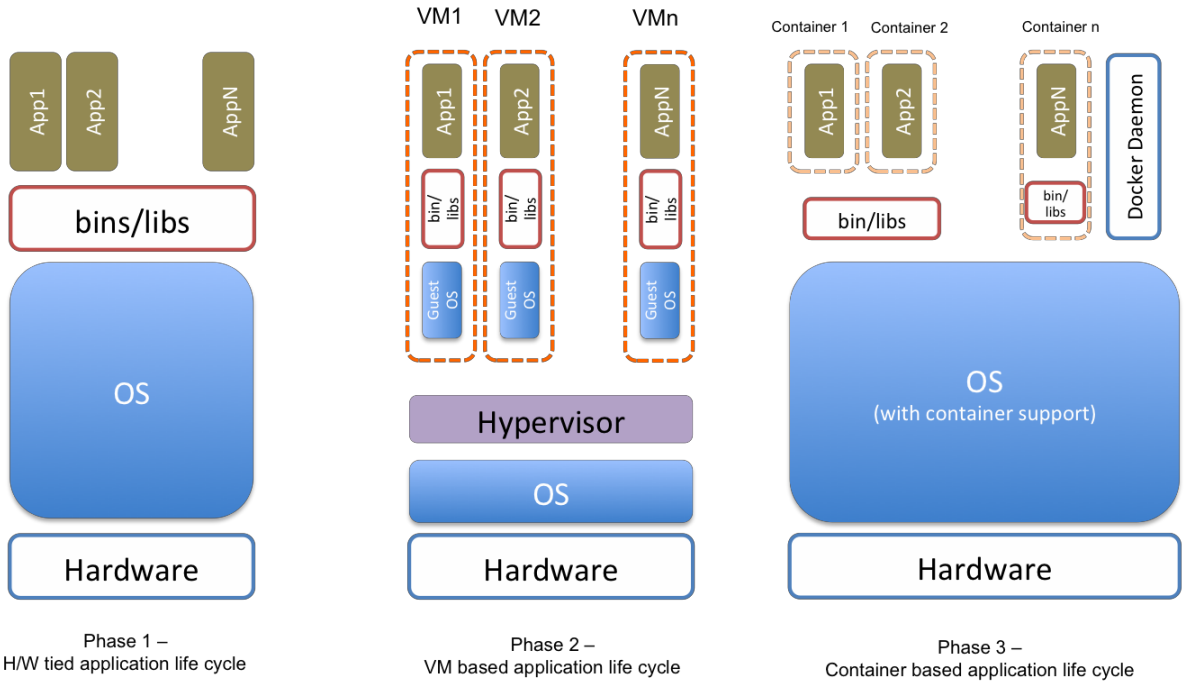

Fig 2 – The evolution of application life cycle deployment

While the evolution is pretty self explanatory, few aspects worth highlighting w.r.to containers:

- Containers provide "middle of the road" characteristics with respect to isolation and overhead.
- Container based deployment requires an operating system with built in "container" support[4].
- It allows for flexibility in sharing binaries and libraries across applications if so desired.
- Containers, unlike VMs, don't allow for heterogeneous operating system deployments to co-exist.
- Docker makes the process of managing, maintaining and deploying containers turnkey.

At a macro level, there are two key reasons why enterprises are adopting Docker (or containers more generally) for application life cycle management:

1. Build once, run anywhere – Any application packaged within the Docker framework is provided with a standards-based mechanism for managing its dependencies. That enables the packaged application to be deployed anywhere and moved around with ease.
2. Better H/W Efficiency – As with the VM movement, underlying hardware has continued to grow at a non- linear pace. That enables a single physical host to masquerade as multiple virtual instances delivering significant total cost of ownership benefits. Containers bring the overhead of this virtualization to as close to bare metal as it can get[5].

[4] Unlike VMs which can be created on any OS

[5] This is possible as containers provide much lower levels of isolation as compared to VMs

Running DataStax Enterprise and OpsCenter with Docker

A typical DSE node[6] runs the following processes on a single instance within the cluster:

- A DSE JVM – including Apache Cassandra, integrated DSE Search, and Spark Master (for HA)
- One or more Spark executor processes
- A single Spark Worker process
- Multiple processes for the integrated Hadoop stack
- Multiple processes which may be started in an adhoc manner (e.g. Spark Job server, SparkSQL CLI, etc.)
- A single OpsCenter agent responsible for monitoring all processes on that DSE instance

The OpsCenter daemon is (logically) separate from the cluster and there is usually one[7] instance for the entire deployment[8].
For the purposes of this document it is recommended that all the DSE processes be configured as 2 containers[9]:

- Container 1 - OpsCenter daemon
- Container 2 - All the JVMs running on a single DSE node (uniformly deployed across the each machine within the cluster)

Prerequisites

1. Install Docker based on instructions found here for the environment under consideration.
2. DataStax provides example Docker files for your use that can be found in a DataStax downloads directory: http://downloads.datastax.com/extra/docker_examples/. Locate the directory that corresponds to the version of DataStax Enterprise that you plan to use and download all files there to a local directory. There may not be a specific version folder for your version in which case use the folder named **latest.** These files are only intended as a starting point, so customization of these files is most likely going to be required.

Steps to Create an OpsCenter Container

1. Download OpsCenter for your environment from here and place it in the same directory where the downloaded example Docker files where placed. In general it is recommended to use the latest version of OpsCenter. When downloaded, please remove the version number part of the filename (or create a symlink), so the resulting file is named opscenter.tar.gz (that way the docker file itself remains version independent).

2. Build the Docker image using the below command

   ```
   docker build . -t <opsc_image_name> -f OpscDockerfile
   ```

3. Check that the image was created

   ```
   docker images
   ```

4. Start the OpsCenter image using the below command and provide it a name that will be used for linking with the DSE container in the following section

   ```
   docker run -d -p 8888:8888 --name <opsc_container_name> <opsc_image_name>
   ```

   - The -p 8888:8888 will expose your containers port 8888 as port 8888 on your local machine for easier access to the OpsCenter web ui.

[6] Node within this context is a single DSE instance
[7] We are ignoring any additional instances that can be used for OpsC HA (e.g. standby)
[8] Note that OpsCenter daemon can manage multiple DSE clusters
[9] In future we plan to provide guidance on utilizing advanced configurations such as splitting up Apache Spark JVMs and DSE Core JVMs within their own separate containers.

5. You will want to retrieve the ip address used by your newly created container by running..This address will be passed in to the DSE containers to allow the DataStax agent to connect to the OpsCenter server

   docker inspect --format '{{ .NetworkSettings.IPAddress }}' <opsc_container_name>

6. Once the DSE cluster is started you can connect to the OpsCenter interface within the browser @ **http://localhost:8888**

7. In the OpsCenter UI you would choose to manage an existing cluster and enter the ip address of the seed node in the next dialog. OpsCenter will contact the cluster and set it up for being managed. Note that it may take a few seconds for the OpsCenter server to come up. You may receive a message asking for you to install agents (automatically or manually). Choose manual and proceed, the agents are already there.

Steps to Create a DSE Cluster

1. Download DSE for your environment from here and place it in the same directory where the downloaded example Docker files where placed. When downloaded, please remove the version number part of the filename (or create a symlink), so the resulting file is named dse-bin.tar.gz (that way the docker file itself remains version independent).

2. Download the DataStax agent for your environment from http://downloads.datastax.com/enterprise/ and place it in the same directory where the downloaded docker example files where placed. When downloaded, please remove the version number part of the filename (or create a symlink), so the resulting file is named datastax-agent_all.deb (that way the docker file itself remains version independent).

Manual Launching

1. Build the Docker image using the below command

   docker build . -t <dse_image_name>

2. Check that the image was created

   docker images

3. Start DSE on the specified Docker container using the below command:

   docker run --link <opsc_container_name> -d CLUSTER_NAME="ClusterName"-
   e STOMP_INTERFACE=<opsc_ip_address> --name <dse_container_name>
   <dse_image_name> <options>

   - --link <opsc_container_name> enables the DSE container to be linked with the OpsCenter container created in an earlier step. For more details w.r.to linking containers please refer to detailed documentation here
   - <options> to be used can be any that can be specified for regular DSE installations. For more details, please refer to the DSE documentation here

4. Repeat steps on all nodes within the cluster. At this point a dockerized DSE cluster and the corresponding OpsCenter setup is up and running.

Launching with Provided Scripts

1. You can use the build_images.sh script to build the opscenter and DSE images. Their names will be opscenter and dse respectively. A WARNING, any preexisting images with those names will be deleted, and you will want to make sure that those images are not currently in use.

2. You can start a number of nodes at once using the downloaded script start_docker_cluster.sh. Please note that you can run the cluster without opscenter by passing in an empty string "" or none as the opsc_image_name. To run DSE Search with a 3-node DSE cluster, specify the –s flag as follows:

   start_docker_cluster.sh <dse_image_name> <opsc_image_name> 3 –s

To run DSE analytics with your 3-node cluster, specify the –k flag:

   start_docker_cluster.sh <dse_image_name> none 3 –k

3. To tear down a cluster deployed with the start_docker_cluster.sh script you can use the teardown_docker_cluster.sh script passing in the number of nodes you specified when creating the cluster.

Try out various DSE commands and utilities:

```
docker exec -it node1 nodetool ring

docker exec -it node1 dsetool status

docker exec -it node1 cqlsh
```

Configuration Notes

While the steps above represent the bare minimum required to create a DSE cluster; certain highly recommended optimizations include the following:

- Given DSE is a distributed database, it is highly recommended that only one DSE node be configured to run per Docker host to achieve optimal networking configuration[10]. To do so, use the --net=host option. Please refer to documentation here for details about networking options. If it is necessary to specify the IP address of the networking interface where services are going to bind, do so using the following command:

  docker run -d -e IP=<ip address> <dse_image_name>

- If using Docker host networking isn't an option the IP addresses to be used can be listed by the following command:

  docker inspect --format '{{ .NetworkSettings.IPAddress }}' <dse_container_name>

- To persist data, configuration and logs across container restarts, pre-create directories and map them via the Docker run command as shown below:

  Assuming the directories for data, conf and logs are

  i. <some root dir>/<dse_image_name>-data for data
  ii. <some root dir>/<dse_image_name>-conf for configuration
  iii. <some root dir>/<dse_image_name>-logs for logs

  These can be mapped to the Docker instance as follows:

  docker run –v <some root dir>/<dse_image_name>-data:/data

[10] This is especially relevant for production deployments

```
–v <some root dir>/<dse_image_name>-conf:/conf
–v <some root dir>/<dse_image_name>-logs:/logs
 –d <dse_image_name>
```

- To provide cluster specific configuration, the following environment variables should be provided via the Docker run command:
    1. CLUSTER_NAME: the name of the cluster to create/connect to
    2. STOMP_INTERFACE: the ip address of the OpsCenter container
    3. SEEDS: the comma-separated list of seed IP addresses, e.g. SEEDS=127.0.0.2,127.0.0.3

- Once a cluster is up and running various tools used to manage DSE such as nodetool, cqlsh, or dsetool can be run with the docker exec command as noted below

```
docker exec -it <dse_container_name> nodetool ring
```

Important Caveats

- Data volumes are required for the commitlog, saved_caches, and data directories (everything in /var/lib/cassandra). The data volume must use a supported file system (usually xfs or ext4).

- Docker's default networking (via Linux bridge) is not recommended for the production use as it slows down networking considerably. Instead, use the host networking (docker run --net=host) or a plugin that can manage IP ranges across clusters of hosts. The host networking limits the number of DSE nodes per a Docker host to one, but this is the recommended configuration to use in production.

- Development and testing benefit from running DSE clusters on a single Docker host and for such scenarios the default networking is just fine.

- Before running DSE within Docker (especially in production), it's prudent that various configuration options are adjusted for the Docker environment (for e.g. fine tuning of dse.yaml, cassandra.yaml, turning swap off on the Docker host, choosing where to manage Cassandra data, calculating optimal JVM heap size, choosing optimal garbage collector, etc).

- The default capability limits of Docker containers break mlockall functionality that Cassandra uses to prevent swapping and page faults. The simplest workaround is to add -XX:+AlwaysPreTouch to the JVM arguments and disable swap on the host OS.

- All containers by default inherit ulimits from the Docker daemon. DSE containers should have them set to unlimited or reasonably high values (for e.g. for mem_locked_memory and max_memory_size).

- There is no support for rack / data center placement for Dockerized nodes. To enable this capability, please refer to this example.

- Token ranges need to be calculated post run (by exposing a node's configuration via mounted volumes).

- Whenever a node starts up it will place its IP address within the /data/ip.address. This can be used as a set of SEEDS to spawn nodes at a later time.

- Certain global state is kept within Cassandra tables (e.g. system.peers, cluster name, etc.). It is not advisable to change any of these whenever a new container is started which re-uses existing data files.

- Upgrading of DSE or OpsC software should follow similar procedures as documented in DataStax Enterprise documentation with the additional requirement of restarting the Docker instance in between upgrades.

- While Docker does provide as close to bare metal performance as possible, the overheads are certainly not zero. Hence, appropriate capacity must be planned for the necessary overhead that may occur.

- **IMPORTANT:** While the DataStax product teams validated best practices listed in this document, adaptations of these instructions may be necessary depending on the deployment under consideration. Hence, it is also highly recommended to execute rigorous tests for the use cases under consideration before deploying a Dockerized DSE installation within production.

DSE/Docker Future

Deploying DSE within Docker isn't trivial, but with adequate guidance and pre-production validation, it's not that difficult. As the container ecosystem evolves, it is expected that future DSE releases will have additional guidelines to make the most of DSE installations under Docker. Some future areas that DataStax is investigating are:

- Further splitting up of DSE processes into separate containers (e.g. running Spark executors and DSE core JVM within a single container, and all other DSE processes within a separate containers)
- Integration of container based deployment with workload management infrastructure components such as Kubernetes, Mesos, etc.
- Enabling the deployment model on a variety of public and private clouds

Conclusion

DataStax acknowledges that containers have rapidly become one of the building blocks for becoming an modern cloud applications. In this paper, we have attempted to provide tips, guidelines and examples to reduce the amount of time required to run DSE in Docker. For more resources and downloads of DataStax Enterprise, visit www.datastax.com today.

About DataStax

DataStax, the leading provider of database software for cloud applications, accelerates the ability of enterprises, government agencies, and systems integrators to power the exploding number of cloud applications that require data distribution across datacenters and clouds, by using our secure, operationally simple platform built on Apache Cassandra™.

With more than 500 customers in over 50 countries, DataStax is the database technology of choice for the world's most innovative companies, such as Netflix, Safeway, ING, Adobe, Intuit, Target and eBay. Based in Santa Clara, Calif., DataStax is backed by industry-leading investors including Comcast Ventures, Crosslink Capital, Lightspeed Venture Partners, Kleiner Perkins Caufield & Byers, Meritech Capital, Premji Invest and Scale Venture Partners. For more information, visit DataStax.com or follow us @DataStax. 07.04.16