Cloud Native Databases with Kubernetes Persistent Storage on Dell EMC VxFlex family

Abstract
This paper highlights the benefits of Cloud Native Database with Kubernetes Persistent Storage on Dell EMC VxFlex family.

January 2020
Acknowledgements

This paper was produced by the following:

Author: Sudhakar Tekkali

Support: David Adams, Kailas Goliwadekar and Shalini G

The information in this publication is provided "as is." Dell Inc. makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose.

Use, copying, and distribution of any software described in this publication requires an applicable software license.

Copyright © 2020 Dell Inc. or its subsidiaries. All Rights Reserved. Dell, EMC, Dell EMC and other trademarks are trademarks of Dell Inc. or its subsidiaries. Other trademarks may be trademarks of their respective owners. [1/7/2020] [White Paper] [000058]
# Table of contents

Revisions ......................................................................................................................................................... 2

Acknowledgements ........................................................................................................................................... 2

Table of contents ............................................................................................................................................ 3

Executive summary ......................................................................................................................................... 5

1 Introduction ................................................................................................................................................ 6

   1.1 Objective .............................................................................................................................................. 6

   1.2 Audience .............................................................................................................................................. 7

   1.3 Terminology ................................................................................................................................:::::: 7

2 Product overview ......................................................................................................................................... 8

   2.1 VxFlex family ...................................................................................................................................... 8

   2.2 VxFlex OS .......................................................................................................................................... 8

   2.3 Kubernetes ....................................................................................................................................... 9

   2.4 Container Storage Interface .............................................................................................................. 9

3 Solution architecture ................................................................................................................................. 10

   3.1 Logical architecture .......................................................................................................................... 10

   3.2 Two-layer network topology ........................................................................................................... 11

   3.3 Kubernetes ...................................................................................................................................... 12

   3.4 Dell EMC VxFlex OS CSI driver architecture ................................................................................ 13

4 Deployment .............................................................................................................................................. 14

   4.1 Deploying Kubernetes ...................................................................................................................... 14

   4.2 Deploying VxFlex OS CSI driver .................................................................................................... 14

       4.2.1 CSI driver prerequisites ............................................................................................................ 14

       4.2.2 Installing CSI driver for VxFlex OS .................................................................................... 14

5 Cloud Native Databases on Kubernetes ................................................................................................. 16

   5.1 MongoDB deployment ...................................................................................................................... 16

       5.1.1 MongoDB on Kubernetes architecture .................................................................................. 16

       5.1.2 Validating MongoDB on Kubernetes ..................................................................................... 17

   5.2 Cassandra deployment .................................................................................................................... 19

       5.2.1 Cassandra on Kubernetes architecture .................................................................................. 19

       5.2.2 Validating Cassandra on Kubernetes ..................................................................................... 19

   5.3 PostgreSQL deployment ................................................................................................................... 21

       5.3.1 PostgreSQL on Kubernetes architecture .................................................................................. 21

       5.3.2 Validating PostgreSQL on Kubernetes ..................................................................................... 21
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4 CockroachDB deployment</td>
<td>23</td>
</tr>
<tr>
<td>5.4.1 CockroachDB on Kubernetes architecture</td>
<td>23</td>
</tr>
<tr>
<td>5.4.2 Validating CockroachDB on Kubernetes</td>
<td>24</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>26</td>
</tr>
<tr>
<td>A Appendix</td>
<td>27</td>
</tr>
<tr>
<td>A.1 Configuration details</td>
<td>27</td>
</tr>
<tr>
<td>A.2 Deployment details</td>
<td>28</td>
</tr>
<tr>
<td>B Technical support and resources</td>
<td>29</td>
</tr>
<tr>
<td>B.1 Related resources</td>
<td>29</td>
</tr>
</tbody>
</table>
Executive summary

Enterprises across the globe are challenged to improve the business processes, develop new capabilities and business models. Enterprises are digitally transforming to meet the business goals and deliver cutting-edge technology, products, and services. Cloud Native architecture enables rapid development and deployment of agile applications at scale and resiliency. One of the most powerful tools in this new Cloud Native architecture is containerization, which has been evolving new workloads and use cases. Both open-source community and software vendors are constantly adding exciting features to container technology to make it necessary for developers and IT.

Cloud Native architecture enables applications to take advantage of containerization, elasticity, resilience, scale, and orchestration. Cloud Native applications, including databases, require supporting tools that have capabilities to deal with scheduling, load balancing, resource monitoring, scaling, and job isolation of these complex environments. Kubernetes is an open-source container orchestration system with capabilities of automatic, scaling, and managing containerized applications.

This document illustrates how to build Cloud Native containerized database workloads on Dell EMC VxFlex family platform with Kubernetes and Dell EMC VxFlex CSI driver providing persistent storage in a private cloud environment.
Introduction

Modern data center workloads have varying business value and characteristics for the workload and data that govern the performance, throughput, capacity, availability, data protection, and data services requirements. Shrinking IT budgets, push for greater efficiency, and consolidation and workload requirements have made it necessary for the underlying infrastructure to deliver high performance, scalability, resiliency, and most importantly -- flexibility. VxFlex family is an engineered system for Dell EMC designed on five super power principals to meet the key infrastructure requirements. VxFlex family delivers:

- Unmatched **performance**.
- Unprecedented **scale** (1000 nodes and 100s-PB storage capacity).
- Built in redundant hardware components and VxFlex OS mesh mirror architecture delivers unparalleled **resiliency**.
- Infrastructure **flexibility**: VxFlex family is second to none. A VxFlex integrated rack system has 1000s of hardware and software configuration option combinations that can co-exist freeing customers from T-shirt size, dedicated and siloed environments, and accelerating the data center consolidation.
- Engineered system with single call support and life cycle management.

VxFlex family with VxFlex OS offers flexibility for installing virtual machines, containers, and bare metal applications.

1. Objective

This white paper outlines how a Kubernetes operator can deploy Cloud Native databases and Kubernetes as container orchestration layer on Dell EMC VxFlex family platform to meet the performance, scalability, resiliency, and availability requirements. The document also provides detailed information about leveraging Dell EMC VxFlex OS CSI driver to dynamically provision persistent volumes.
1.2 **Audience**

This document is intended for decision makers, business leaders, architects, cloud administrators, DevOps Infrastructure administrators, Cloud Native database administrators, Hyperconverged infrastructure administrators, and technical administrators of IT environments responsible for developing and deploying platform and service to run Cloud Native databases.

The reader of this document must have a working knowledge of Dell EMC VxFlex, VMware vSphere technologies, Linux, Container technologies, Kubernetes, YAML scripting, Cloud Native databases such as Cassandra, PostgreSQL, MongoDB, CockroachDB technologies.

1.3 **Terminology**

The following table defines acronyms and terms that are used throughout this document:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDM</td>
<td>Meta Data Manager</td>
</tr>
<tr>
<td>SDS</td>
<td>Storage Data Server</td>
</tr>
<tr>
<td>SDC</td>
<td>Storage Data Client</td>
</tr>
<tr>
<td>RCM</td>
<td>Release Certification Matrix</td>
</tr>
<tr>
<td>SSD</td>
<td>Solid State Drive</td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure as a Service</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>AU</td>
<td>Allocation Unit</td>
</tr>
<tr>
<td>PD</td>
<td>Protection Domain</td>
</tr>
<tr>
<td>LUN</td>
<td>Logical Unit Number</td>
</tr>
<tr>
<td>PV</td>
<td>PersistentVolume</td>
</tr>
<tr>
<td>PVC</td>
<td>PersistentVolumeClaim</td>
</tr>
<tr>
<td>YCSB</td>
<td>Yahoo cloud servicing benchmark</td>
</tr>
<tr>
<td>TPC-C</td>
<td>Transaction Processing Performance Council Benchmark</td>
</tr>
<tr>
<td>VLT</td>
<td>Virtual Link Trucking</td>
</tr>
<tr>
<td>CNI</td>
<td>Container Network Interface</td>
</tr>
</tbody>
</table>
2 Product overview

2.1 VxFlex family
VxFlex family helps transforming from a traditional three-tier architecture to a modern data center without any trade-offs, meeting business requirements without compromise. The central software layer for VxFlex family is VxFlex OS, scale-out block storage service that enables customers to create a scale-out Server SAN or hyperconverged infrastructure. The VxFlex family currently include VxFlex Ready Nodes, VxFlex appliance, and VxFlex integrated rack.

![VxFlex家族](image)

Figure 2 VxFlex family

2.2 VxFlex OS
VxFlex OS applies the principles of server virtualization to standard x86 servers with local disks, creating high-performance, sharable pools of block storage. VxFlex OS abstracts the local storage out of each contained within each server, including HDDs, SSDs, and all-flash. VxFlex OS uses three lightweight pieces of software to create, consume, and coordinate the storage layer in VxFlex systems.

VxFlex OS can be deployed in the following ways:

- **Two-layer**: Compute resources exist on one set of physical nodes, and storage resources exist on another set of physical nodes which may be scaled independently.
- **Hyperconverged (HCI)**: Compute resources and storage resources exists on the same set of physical nodes.
- **Hybrid**: VxFlex OS also enables deployments that mix the two-layer and HCI deployments.

For more information about VxFlex OS, see [VxFlex OS documentation](#).
2.3 Kubernetes

Kubernetes is an open-source platform which is portable and can be extended to manage containerized workloads and services. Thus Kubernetes, facilitates both declarative configuration and automation. Containers are a good way to bundle and run your applications because it has its own filesystem, CPU, memory, process space, and so on. As they are decoupled from the underlying infrastructure, they are portable across clouds and operating system distributions. Kubernetes provides you with a framework to run distributed systems resiliently.

For more information about Kubernetes, see Kubernetes documentation.

2.4 Container Storage Interface

Kubernetes natively offers some solutions to manage storage, however, native storage options also present challenges with the pod portability. CSI solves the challenges with native Kubernetes solutions. It is a community driven standard for persistent storage on container orchestrators (COs) like Kubernetes. It enables storage providers to develop CSI driver for Kubernetes CO systems. It lets you provision storage for pods through a Kubernetes PersistentVolumeClaim (PVC).

For more information about CSI Overview, see Container Storage Interface (CSI) for Kubernetes GA.
3 Solution architecture

3.1 Logical architecture

The two-layer architecture is where the Storage Data Server (SDS) and Storage Data Client (SDC) components are isolated from each other on separate set of nodes. The following figure demonstrates the two-layer deployment of VxFlex OS where compute nodes run ESXi and storage nodes run Red Hat Enterprise Linux (RHEL):

![VxFlex OS two-layer logical designs](image)

For compute, a VMWare ESXi 6.7 three node cluster is created to host workload VMs, which, in this case, is a Kubernetes cluster. The SDC component is also installed on the Kubernetes worker nodes to allow direct mount of VxFlex OS volumes to Kubernetes pods.

For storage, RHEL 7.6 deployed on four storage defined only ones that acts as SDS. From VxFlex OS, VxFlex OS cluster consists of one product domain.
3.2 Two-layer network topology

The following diagram depicts the two-layer network architecture that is based on VxFlex OS best practices:

- Two Dell S5048F switches are configured with VLT to provide fault condition and enable connectivity with other switches. Storage nodes, Compute nodes, Management, and other Teamed Networks using Link Aggregation Group (LAG).
- Two dual ports 25 Gig Mellanox NICs on each server provide 4 x 25 Gig ports.
- VLAN 51 and 52 (also known as Data 1 and Data 2) network channels are dedicatedly used for SDS-SDS communication. Both VLANs are isolated at switch level to provide high availability and avoid single point of failure.
- VLAN 53 and 54 (also known as Client 1 and Client 2) network channels are dedicatedly used for SDS-SDC communication. Both VLANs are isolated at switch level to provide high availability and avoid single point of failure.
- On compute nodes, 2 x 25 Gig ports are teamed to provide high availability. Teamed Network is used to create Virtual Networks in vSphere. VLAN 56 is configured to provide connectivity with customer network and VLAN 57 is dedicated vMotion.
- On compute nodes, 2 x 10 Gig teamed ports are used to create Management network. VLAN 55 is dedicated to Management network which includes VxFlex OS management and Hypervisor management.
3.3 Kubernetes

Kubernetes cluster follows the client/server architecture. By default, a Kubernetes cluster consists of minimum one master node and multiple worker nodes. For high availability, multiple master nodes need to be considered.

Master server consists of various components to control the cluster and facilitate communication with APIs.

The following figure depicts the high-level architecture of a Kubernetes cluster:

![Kubernetes Architecture Diagram]

**Figure 5** Kubernetes architecture

---

**Note:** Several Container Network Interface (CNI) options are available to implement Kubernetes cluster networking. For more information about Kubernetes Network policy, see [Networking and Network Policy](#).

Kubernetes contains several abstractions that represent the state of the system; containerized workload, their associated network, and disk resources. The objects in Kubernetes API represent these abstractions. The basic Kubernetes objects include:

- **Pod:** This consists of containers, storage resources, and unique network id.
- **Service:** Represents a logical set of pods and acts as a gateway to provide load balancing.
- **Volumes:** Is associated with pods to store data. Volumes can be persistent or non-persistent depending on the application requirements and deployment.
- **Namespaces:** Isolation of names and resources in the cluster.

Kubernetes also contains higher-level abstraction that relies on controllers to build upon basic objects and provide additional functionality. For details, see [Kubernetes Components](#).
3.4 Dell EMC VxFlex OS CSI driver architecture

Dell EMC has developed a CSI driver plug-in for VxFlex OS and Kubernetes. The plug-in provides persistent storage using VxFlex OS storage system. The CSI drive for VxFlex OS and Kubernetes communicates over CSI protocol.

The following figure shows the detailed architecture of the deployment and how the VxFlex OS SDC interacts with CSI components to deliver persistent storage to the Kubernetes cluster:

Figure 6 VxFlex OS CSI Driver for Kubernetes

To access the CSI driver source code and product documentation, see VxFlex OS CSI driver on GitHub.
4 Deployment

4.1 Deploying Kubernetes

Running a Kubernetes cluster requires at least one master and one worker node. For the demonstration of this deployment, one master and three worker nodes for each Kubernetes cluster were used. The hardware and software details of the Kubernetes cluster nodes is available here: Appendix A.2.

There are multiple ways of deploying Kubernetes in production environment. For this solution, kubeadm was used to deploy the Kubernetes clusters. For complete instructions on deploying Kubernetes cluster, see Installing kubeadm.

4.2 Deploying VxFlex OS CSI driver

VxFlex OS CSI driver enables Kubernetes worker nodes to provision persistent volumes from VxFlex OS storage. The following section describes the procedure to deploy CSI driver:

4.2.1 CSI driver prerequisites

Ensure to complete the following requirements before installing CSI Driver:

- Install Kubernetes version 1.13.3
- Enable the required Kubernetes feature gates
- Configure Docker service
- Install Helm and Tiller with a service account
- Install VxFlex OS SDC components on each Kubernetes node

For complete details on prerequisites and full installation, see CSI Driver for VxFlex OS Product Guide.

4.2.2 Installing CSI driver for VxFlex OS

Complete the following steps to install CSI driver version 1.1.3 on VxFlex OS:

1. Download the installation source files from GitHub, using the following command:

   ```
   $ git clone https://github.com/dell/csi-vxflexos
   ```

2. Create a Kubernetes secret file with your VxFlex OS username and password using the template in secret.yaml.

   ```
   $ kubectl create -f secret.yaml -n vxflexos
   ```

3. Copy the csi-vxflexos/values.yaml to a file myvalues.yaml in this directory.

4. Specify the required parameters in myvalues.yaml file.

   ```
   $ cat myvalues.yaml
   systemName: 264facb730a26a0f
   username: admin
   password: *****
   ```
restGateway: https://gateway.flex.com
storagePool: default
mdmIP: mdmIP1, mdmIP2
volumeNamePrefix: k8s
storageClass.name: vxflexos

Note: The mdmIP values that are provided here are just examples. You must use relevant mdmIP values for performing this step.

5. Run the following command to proceed with the installation:

```bash
$ sh install.vxflexos
NAME: vxflexos
LAST DEPLOYED: Fri Aug 30 17:17:32 2019
NAMESPACE: vxflexos
STATUS: DEPLOYED
```

6. Verify that the CSI agent pods are running.

```bash
$ kubectl get pods -n vxflexos
NAME READY STATUS RESTARTS AGE
vxflexos-controller-0 4/4 Running 0 112s
vxflexos-node-2fmz4 2/2 Running 0 112s
vxflexos-node-4c7hh 2/2 Running 0 112s
vxflexos-node-ptvg4 2/2 Running 0 112s

$ kubectl get storageclass
NAME PROVISIONER AGE
vxflexos (default) csi-vxflexos 21s
vxflexos-xfs csi-vxflexos 21s
```
5 Cloud Native Databases on Kubernetes

In this paper, the following Cloud Native Databases are deployed on Kubernetes based on specific requirements followed by validating and providing results for each database using an example:

- MongoDB deployment
- Cassandra deployment
- PostgreSQL deployment
- CockroachDB deployment

Prerequisites

Ensure to complete the following prerequisites before deploying Cloud Native databases:

- Create VxFlex OS cluster and Kubernetes cluster.
- Install and configure network stack for Kubernetes cluster.
- Install and configure Helm and Tiller with service account.
- Install and configure VxFlex OS CSI driver.
- Verify `vxflexos StorageClass` is successfully deployed.

5.1 MongoDB deployment

MongoDB is a cross-platform document-oriented database. MongoDB can be used to store large amount of data volume and has a schema-less data model. The purpose of the replication increases the read capacity, which enables users to perform read operations from different servers. Maintaining copies of data in different data nodes can increase data locality and availability for distributed applications. For complete details, see How Containers Benefit Your Business.

5.1.1 MongoDB on Kubernetes architecture

The `mongos` instances provide the interface between the client applications and the shaded cluster. A replica set in MongoDB is a group of `mongod` instances that maintain the same dataset and it can also handle data requests, manages data access, and performs background management operations.

A replica set contains the following nodes:

- **Primary node** is the only node in cluster to perform write operations, and it can also contain the config files for the cluster.
- **Secondary node** is where the replicas of the data are stored, and it supports read operations.
- **Arbitrary node** plays a role of voter node to allow cluster achieve quorum and in case split brain syndrome, the arbiter node is used to decide the primary node.
The following diagram shows the high-level architecture of MongoDB cluster:

![MongoDB Architecture Logical Design](image)

**Figure 7** MongoDB architecture logical design

For more information about MongoDB components, see [MongoDB Replication](#).

### 5.1.2 Validating MongoDB on Kubernetes

Helm charts provides `stable/mongodb` and `stable/mongodb-replicaset` options to replicate MongoDB pods. For this paper, `stable/mongodb` chart was considered.

MongoDB on Kubernetes cluster running on VxFlex family platform was installed following the procedure available in [MongoDB Helm chart on Github](#).

After adding `stable` repository to the Helm chart, complete the following steps:

1. Specify the following parameters in the `values.yaml` in addition to the default parameters for Kubernetes environment:

   ```yaml
   $ cat values.yaml
   mongodbRootPassword: password
   mongodbUsername: admin
   mongodbPassword: *****
   mongodbDatabase: mydatabase
   replicaSet.enabled: true
   metrics.enabled: true
   metrics.livenessProbe.enabled: true
   metrics.readinessProbe.enabled: true
   storage.persistentVolume.storageClass: vxflexos
   ```

2. Install the stable/mongo Helm chart using `values.yaml`. 
3. Deployed results are displayed as shown here:

```
$ helm status mongo
NAME: mongoreplicatest-myrelease-mongodb
LAST DEPLOYED: <Timestamp>
NAMESPACE: mongo-replica
STATUS: DEPLOYED

RESOURCES:
===> v1/Pod(related)
NAME READY STATUS RESTARTS AGE
mongoreplicatest-myrelease-mongodb-arbiter-0 1/1 Running 3 3h31m
mongoreplicatest-myrelease-mongodb-primary-0 2/2 Running 6 3h29m
mongoreplicatest-myrelease-mongodb-secondary-0 2/2 Running 6 3h27m

===> v1/Service
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
mongoreplicatest-myrelease-mongodb ClusterIP None <none> 7000/TCP,7001/TCP,7199/TCP,9042/TCP,9160/TCP 3h31m

===> v1/StatefulSet
NAME READY AGE
mongoreplicatest-myrelease-mongodb 3/3 3h31m
```

4. Verify that the MongoDB pods are running.

```
$ kubectl get pods -n mongo-replica -o wide
```

5. Verify that PersistentVolumeClaim are created for MongoDB container.

```
$ kubectl get pvc -n mongo-replica
```

6. Verify that the volume is created using VxFlex OS GUI.

Results: On examining VxFlex OS GUI, it is observed that 8 GB volume k8s-242bd64bf2 and k8s-69f2236970 are dynamically created and mounted to the MongoDB pods.

**Benchmarking:**

The benchmarking of MongoDB on Kubernetes was done using YCSB benchmarking tool with the following parameters:

- Number of threads: One
- Block size: 64k
- Number of operations: 1000 for the first instance and 100000 for the second instance
5.2 **Cassandra deployment**
Cassandra is an open source, wide column store, distributed, designed to handle massive amount of data workloads across multiple nodes with no single point of failure. Cassandra architecture ensures data durability and availability by distributed system across homogeneous nodes where data is distributed among all nodes in the cluster. For complete details, see [Cassandra](#).

5.2.1 **Cassandra on Kubernetes architecture**
Each node in the cluster exchange its health information by using Gossip protocol. A sequentially written CommitLog on each node captures write activities. The data is indexed and written to an in-memory structure called memtable. Each time the memory structure is full, the data is flushed to the disk in a SSTable datafile. In Cassandra architecture, each node in the cluster is same where each node can perform both read and write operation.

The following diagram depicts the high-level architecture of Cassandra cluster:

![Cassandra architecture logical design](image)

For more information about Cassandra components, see [Cassandra Architecture](#).

5.2.2 **Validating Cassandra on Kubernetes**
CassandraDB on Kubernetes cluster running on VxFlex family platform was installed following the procedure in [Cassandra Helm Chart on Github](#).

After adding `incubator` repository to the Helm chart, complete the following steps:

1. Specify the following parameters in the `values.yaml` in addition to the default parameters for Kubernetes environment:
   ```yaml
   $ cat values.yaml
   config.cluster_size: 3
   persistence.enabled: true
   persistence.storageClass: vxflexos
   ```

2. Install the `incubator/cassandra` Helm Chart using `values.yaml`.

3. Deployed results are displayed as shown here:
   ```bash
   $ helm status cassandra
   NAME: cassandra
   LAST DEPLOYED: <Timestamp>
   NAMESPACE: cas-deploy
   STATUS: DEPLOYED
   ```
Cloud Native Databases on Kubernetes

RESOURCES:

```markdown
<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>cassandra-0</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>3h31m</td>
</tr>
<tr>
<td>cassandra-1</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>3h29m</td>
</tr>
<tr>
<td>cassandra-2</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>3h27m</td>
</tr>
</tbody>
</table>
```

```markdown
<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>cassandra</td>
<td>ClusterIP</td>
<td>None</td>
<td>&lt;none&gt;</td>
<td>7000/TCP, 7001/TCP, 7199/TCP, 9042/TCP, 9160/TCP</td>
<td>3h31m</td>
</tr>
</tbody>
</table>

```markdown
<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>cassandra</td>
<td>3/3</td>
<td>3h31m</td>
</tr>
</tbody>
</table>
```

4. Verify that the Cassandra pods are running.

```sh
$ kubectl get pods -n cas-deploy -o wide
```

5. Verify that PersistentVolumeClaim are created for Cassandra container.

```sh
$ kubectl get pvc -n cas-deploy
```

6. Verify that the volume is created using VxFlex OS GUI.

![Volume Creation](image.png)

**Results:** On examining VxFlex OS GUI, it is observed that 16 GB volume k8scas-3cd3b89cc0, k8cas-3e33085080, k8scas-d1e6d7b240 and k8scas-d4a629f996 are dynamically created and mounted to the Cassandra pods.

**Benchmarking:** Cassandra offers a built-in java-based stress testing tool within the deployment. The tool can perform various test scenarios like 100% write, 100% read and mixed workload. However, to run read or mixed workload, Cassandra-stress testing tool requires data to be present in the database.
5.3 PostgreSQL deployment
PostgreSQL is a RDBMS with a client/server architecture model. PostgreSQL Server consists of shared memory, background process and datafile. PostgreSQL server manages database files, accepts connections to the databases from clients and perform database operations on behalf of clients. PostgreSQL can support multiple client connections at the same time.

5.3.1 PostgreSQL on Kubernetes architecture
The following figure depicts the overall architecture of PostgreSQL deployment:

![PostgreSQL Architecture Diagram]

For more information PostgreSQL database Architectural Fundamentals, see PostgreSQL Architectural Fundamentals.

5.3.2 Validating PostgreSQL on Kubernetes
PostgreSQL on Kubernetes cluster running on VxFlex family platform was installed following the procedure in PostgreSQL Helm Chart on GitHub.

After adding stable repository to the Helm chart, complete the following steps:

1. Specify the following parameters in the `values.yaml` in addition to the default parameters for Kubernetes environment:

   ```yaml
   $ cat values.yaml
   replication.enabled: true
   replication.slaveReplicas: 3
   replication.synchronousCommit: on
   replication.numSynchronousReplicas: 1
   postgresqlDatabase: flex-db
   postgresqlUsername: postgres
   postgresqlPassword: *******
   persistence.storageClass: vxflexos
   ```

2. Install the stable/postgresql Helm Chart using `values.yaml`.
3. Deployed results are displayed as shown here:

```
NAME: flex-postgres
LAST DEPLOYED: <Timestamp>
NAMESPACE: default
STATUS: DEPLOYED
RESOURCES:
<table>
<thead>
<tr>
<th>RESOURCE</th>
<th>NAME</th>
<th>LAST DEPLOYED</th>
<th>NAMESPACE</th>
<th>STATUS</th>
<th>RESOURCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1/Pod</td>
<td>flex-postgres-postgresql-master-0</td>
<td>0/2 Pending</td>
<td>default</td>
<td>DEPLOYED</td>
<td>v1/Pod(related)</td>
</tr>
<tr>
<td>v1/Pod</td>
<td>flex-postgres-postgresql-slave-0</td>
<td>0/1 Pending</td>
<td>default</td>
<td>DEPLOYED</td>
<td></td>
</tr>
</tbody>
</table>

NAME READY STATUS RESTARTS AGE
flex-postgres-postgresql-master-0 0/2 Pending 0 0s
flex-postgres-postgresql-slave-0 0/1 Pending 0 0s

NAME TYPE DATA AGE
flex-postgres-postgresql Opaque 2 0s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
flex-postgres-postgresql ClusterIP 10.101.250.197 <none> 5432/TCP 0s
flex-postgres-postgresql-headless ClusterIP None <none> 5432/TCP 0s
flex-postgres-postgresql-metrics ClusterIP 10.108.93.75 <none> 9187/TCP 0s
flex-postgres-postgresql-read ClusterIP 10.99.18.36 <none> 5432/TCP 0s

NAME READY AGE
flex-postgres-postgresql-master 0/1 0s
flex-postgres-postgresql-slave 0/3 0s

NOTES:
** Please be patient while the chart is being deployed **
PostgreSQL can be accessed via port 5432 on the following DNS name from within your cluster:
flex-postgres-postgresql.default.svc.cluster.local -Read/Write connection
flex-postgres-postgresql-read.default.svc.cluster.local -Read only connection
To get the password for "postgres" run:
export POSTGRES_PASSWORD=$(kubectl get secret --namespace default flex-postgres-postgresql -o jsonpath="{.data.postgresql-password}" | base64 --decode)
To connect to your database run the following command:

```
$ kubectl run flex-postgres-postgresql-client --rm --tty -i --restart='Never' --namespace default --image docker.io/bitnami/postgresql:11.4.0-debian-9-r34 -- env="PGPASSWORD=POSTGRES_PASSWORD" --command psql --host flex-postgres-postgresql -U postgres -d flex-db -p 5432
```
To connect to your database from outside the cluster execute the following commands:
kubectl port-forward --namespace default svc/flex-postgres-postgresql 5432:5432 &
PGPASSWORD=POSTGRES_PASSWORD psql --host 127.0.0.1 -U postgres -d flex-db -p 5432
```

4. Verify that the PostgreSQL pods are running.

```
$ kubectl get pods -o wide | grep "flex-postgres-postgresql"
```

5. Verify that **PersistentVolumeClaim** are created for Cassandra container.

```bash
$ kubectl get pvc
```

6. Verify that the volume is created using VxFlex OS GUI.

![Volume Creation Screenshot]

**Results:** On examining VxFlex OS GUI, it is observed that 16 GB volume `k8svol-46d8b22491`, `k8svol-23346c1476`, and `k8svol-b539073c65` are dynamically created and mounted to the PostgreSQL pods.

**Benchmarking:** PostgreSQL offers a pre-built performance benchmarking tool using pgbench for Linux bundled within the binaries. The tool includes several load generators for simulating client traffic.

### 5.4 CockroachDB deployment

CockroachDB is a distributed SQL database that is also known as NewSQL. The primary design goals are scalability, strong consistency, and survivability.

#### 5.4.1 CockroachDB on Kubernetes architecture

CockroachDB implements a layered architecture. It depends directly on SQL layer, which provides familiar relational concepts such as Schema, tables, columns, and Indexes. The SQL layer in turn depends on the distributed key value store, which handles the details of range addressing to provide the abstraction of a single, massive key value store. The distributed KV store that can communicate with multiple numbers of physical cockroach nodes. Each node contains one per physical device and one or more stores. Each store contains potentially many ranges, the lowest-level unit of key-value data. Ranges are replicated using the Raft consensus protocol.

The following figure depicts the high-level architecture of CockroachDB cluster:

![CockroachDB Architecture Diagram]

For more detailed about CockroachDB Architecture, see [CockroachDB design](#).
5.4.2 Validating CockroachDB on Kubernetes

CockroachDB on Kubernetes cluster running on VxFlex family platform was installed following the procedure in [CockroachDB Helm Chart on Github](https://github.com/cockroachdb/cloud/tree/master/helm/cockroachdb). After adding `stable` repository to the Helm chart, complete the following steps:

1. Specify the following parameters in the `values.yaml` file in addition to the default parameters for Kubernetes environment:

   ```shell
cat values.yaml
storage.persistentVolume.storageClass: vxflexos
```

2. Install the `stable/cockroachdb` Helm Chart using `values.yaml`.

3. Deployed results are displayed as shown here:

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMESPACE: default</td>
<td>STATUS: DEPLOYED</td>
</tr>
<tr>
<td>RESOURCES:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>```v1/Job</td>
</tr>
<tr>
<td></td>
<td>```v1/Pod (related)</td>
</tr>
<tr>
<td>NAME COMPLETIONS</td>
<td>DURATION AGE</td>
</tr>
<tr>
<td>cock-release-cockroachdb-init</td>
<td>0/1 1s 1s</td>
</tr>
<tr>
<td></td>
<td>```v1/Service</td>
</tr>
<tr>
<td>NAME TYPE</td>
<td>CLUSTER-IP EXTERNAL-IP PORT(S) AGE</td>
</tr>
<tr>
<td>cock-release-cockroachdb-0</td>
<td>ClusterIP None &lt;none&gt;</td>
</tr>
<tr>
<td>cock-release-cockroachdb-1</td>
<td>ClusterIP 10.103.197.144 &lt;none&gt;</td>
</tr>
<tr>
<td></td>
<td>```v1beta1/PodDisruptionBudget</td>
</tr>
<tr>
<td>NAME MIN AVAILABLE</td>
<td>MAX UNAVAILABLE ALLOWED DISRUPTIONS AGE</td>
</tr>
<tr>
<td>cock-release-cockroachdb-budget</td>
<td>N/A 1 0 1s</td>
</tr>
<tr>
<td></td>
<td>```v1beta1/StatefulSet</td>
</tr>
<tr>
<td>NAME READY AGE</td>
<td></td>
</tr>
<tr>
<td>cock-release-cockroachdb</td>
<td>0/3 1s</td>
</tr>
</tbody>
</table>

NOTES:
CockroachDB can be accessed via port 26257 at the following DNS name from within your cluster:
cock-release-public.default.svc.cluster.local

Because CockroachDB supports the PostgreSQL wire protocol, you can connect to the cluster using any available PostgreSQL client. For example, you can open up a SQL shell to the cluster by running:

```shell
$ kubectl run -it --rm cockroach-client --image=cockroachdb/cockroach --restart=Never --command --./cockroach sql --insecure --host cock-release-cockroachdb-public.default
```

From there, you can interact with the SQL shell as you would any other SQL shell, confident that any data you write will be safe and available even if parts of your cluster fail.
Finally, to open up the CockroachDB admin UI, you can port-forward from your local machine into one of the instances in the cluster:

```bash
kubectl port-forward cock-release-cockroachdb-0 8080
```

Then you can access the admin UI at `http://localhost:8080` in your web browser. For more information on using CockroachDB, please see the project's docs at [https://www.cockroachlabs.com/docs/](https://www.cockroachlabs.com/docs/)

4. Verify that the CockroachDB pods are running.

```bash
$ helm get pods
```

5. Verify the **PersistentVolumeClaim** is created for CockroachDB container.

```bash
$ kubectl get pvc
```

6. Verify that the volume is created using VxFlex OS GUI.

![Volume Creation](image)

**Results:** On examining VxFlex OS GUI, it is observed that 104 GB volume `k8scock-138ab1de12`, `k8scock-146caf2fcc`, and `k8scock-72da9b64e1` are dynamically created and mounted to the CockroachDB pods.

**Benchmarking:** CockroachDB offers a built-in performance stress testing tool that is called Workload to test the performance of the deployment. The workload tool includes several load generators for simulating client traffic. For more detailed about CockroachDB benchmark tool, see [TPC-C benchmark tool](https://www.cockroachlabs.com/docs/).
6 Conclusion

This paper concludes that Cloud Native databases perform well on Dell EMC VxFlex family platform with Kubernetes. It allows extreme scalability and flexibility for next generation containerized applications and their workloads. It also detailed the usage of CSI driver in achieving persistent storage for stateful applications and dynamic volume creation, snapshots, deletion in VxFlex OS storage system. For this validation, YSCB and TPC-C like benchmarking tools were used to verify the read and write transactions achieved specified time period each database.
A.1 Configuration details

Note: Standard Dell EMC PowerEdge R740 server. Available storage on storage nodes is the only difference between storage and compute nodes. SSDs on storage nodes is been used to form a storage pool and volumes that are utilized by compute nodes.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Cores</td>
<td>2 x 8 Intel® Xeon® Gold 6134 CPU @ 3.20 GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>12 x 16 DDR4</td>
</tr>
<tr>
<td>NIC</td>
<td>2 x MLNX 25GbE 2P ConnectX4LX Adpt</td>
</tr>
<tr>
<td></td>
<td>1 x Intel® 2P X710/2P I350 rNDC</td>
</tr>
<tr>
<td>Storage</td>
<td>BOSS S1 Controller 2 x 120 GB SATA SSD</td>
</tr>
<tr>
<td></td>
<td>Dell HBA330 controller 6 x 3.84 TB SAS SSD</td>
</tr>
<tr>
<td>operating system</td>
<td>RHEL 7.6</td>
</tr>
<tr>
<td>VxFlex OS</td>
<td>3.0.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Cores</td>
<td>2 x 8 Intel® Xeon® Gold 6134 CPU @ 3.20 GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>12 x 16 DDR4</td>
</tr>
<tr>
<td>NIC</td>
<td>2 x MLNX 25 GbE 2P ConnectX4LX Adpt</td>
</tr>
<tr>
<td></td>
<td>1 x Intel® 2P X710/2P I350 rNDC</td>
</tr>
<tr>
<td>Storage</td>
<td>BOSS S1 Controller 2 x 120 GB SATA SSD</td>
</tr>
<tr>
<td></td>
<td>Dell HBA330 controller 6 x 3.84 TB SAS SSD</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>ESXi 6.7</td>
</tr>
<tr>
<td>VxFlex OS</td>
<td>3.0.1</td>
</tr>
</tbody>
</table>
A.2 Deployment details

Table 4 Kubernetes deployment details

<table>
<thead>
<tr>
<th>Components</th>
<th>Items</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>Virtualized Hardware</td>
<td>vSphere 6.7</td>
</tr>
<tr>
<td></td>
<td>CPU</td>
<td>2 vCPU</td>
</tr>
<tr>
<td></td>
<td>RAM</td>
<td>4 GB</td>
</tr>
<tr>
<td></td>
<td>Hard Disk</td>
<td>40 GB</td>
</tr>
<tr>
<td></td>
<td>NIC</td>
<td>VMware Virtual NIC</td>
</tr>
<tr>
<td>Software</td>
<td>Operating System</td>
<td>Centos 7.6</td>
</tr>
<tr>
<td></td>
<td>Container Runtime</td>
<td>Docker 1.19</td>
</tr>
<tr>
<td></td>
<td>Kubernetes</td>
<td>v1.13.3</td>
</tr>
<tr>
<td></td>
<td>CSI Plugin</td>
<td>Flannel</td>
</tr>
</tbody>
</table>
B Technical support and resources

- Dell.com/support is focused on meeting customer needs with proven services and support.
- Storage technical documents and videos provide expertise that helps to ensure customer success on Dell EMC storage platforms.

B.1 Related resources

Note: Few links might require registration to access.

- Dell EMC VxFlex Product Overview
- VxFlex with Kubernetes
- CSI Driver for VxFlex OS Product Guide
- Kubernetes components
- Flannel CNI plugin
- Kubernetes CSI Developer Documentation
- MongoDB Helm chart
- YCSB benchmark on MongoDB
- Cassandra Architecture
- Cassandra Helm chart
- PostgreSQL components
- PostgreSQL Helm chart
- CockroachDB Helm chart