VxRack FLEX and VxFlex Ready Nodes: Premier HCI Platforms for Kubernetes Stateful Applications

Using VxFlex OS Container Storage Interface (CSI) driver to deliver persistent storage for PostgreSQL and Cassandra.

December 2018
VxRack FLEX and VxFlex Ready Nodes: Premier HCI Platforms for Kubernetes Stateful Applications

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>December 2018</td>
<td>Initial release</td>
</tr>
</tbody>
</table>

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.

© 2018 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.

Dell believes the information in this document is accurate as of its publication date. The information is subject to change without notice.
# Table of contents

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

1 Introduction ......................................................................................................................................................... 4

2 Before you begin .................................................................................................................................................. 7
   2.1 Prerequisites .................................................................................................................................................. 7
   2.2 Introduction to Helm ....................................................................................................................................... 7

3 Container Storage Interface (CSI) driver for VxFlex OS .................................................................................... 8
   3.1 Kubernetes architecture with VxFlex OS CSI driver ...................................................................................... 8
   3.2 Installing the VxFlex OS CSI driver using Helm .......................................................................................... 9

4 PostgreSQL example deployment .......................................................................................................................... 11
   4.1 Installing PostgreSQL using Helm ............................................................................................................... 11
   4.2 Testing PostgreSQL database with pgbench script. .................................................................................... 12
   4.3 Demonstrating benefits of persistent storage with PostgreSQL ...................................................................... 14

5 Cassandra example deployment ........................................................................................................................... 17
   5.1 Installing Cassandra using Helm ................................................................................................................ 17
   5.2 Testing Cassandra with Cassandra Stress Script .......................................................................................... 18
   5.3 Demonstrating benefits of persistent storage with Cassandra ....................................................................... 22

6 Summary ............................................................................................................................................................ 25

7 Technical support and resources ........................................................................................................................ 26
VxRack FLEX and VxFlex Ready Nodes, also known as the Flex family, create a server-based SAN by combining storage virtualization software, known as VxFlex OS, with Dell EMC PowerEdge servers to deliver flexibility, scalability, and capacity on demand. Local storage resources are combined to create a virtual pool of block storage with varying performance tiers. The Flex family enables you to start small (with as little as four nodes) and scale incrementally. The Flex family provides enterprise-grade data protection, multi-tenant capabilities, and add-on enterprise features such as QoS, thin provisioning, and snapshots. VxFlex OS is the key enabler and provides an unmatched combination of performance, resiliency and flexibility to address enterprise data center needs. The unique features of VxFlex OS make it an excellent complement to Kubernetes for stateful applications, such as databases, continuous integration, logging and monitoring platforms.

VxFlex OS components

Storage Data Client (SDC)

- Provides front-end volume access to applications and filesystems
- Installed on servers consuming storage
- Maintains peer-to-peer connections to every SDS managing a pool of storage

Storage Data Server (SDS)

- Abstracts local storage, maintains storage pools, and presents volumes to the SDCs
- Installed on servers contributing local storage to the VxFlex OS cluster

Figure 1 Flex Family with VxFlex OS offers flexibility for installing virtual machines, containers, and bare metal applications.

VxFlex OS is capable of supporting a single, scalable block storage service across hypervisors, container platforms and other data center services. VxFlex OS offers true block storage as a service:

- Provisioned natively through Kubernetes
- Dynamically create and delete volumes on demand
- Support quality of service and security context through container storage interface
- Dynamically scale storage service to match demand
- Support fully non-disruptive updates without future fork-lift migrations

VxRack FLEX and VxFlex Ready Nodes integrate with multiple Kubernetes implementations.
Metadata Manager (MDM)

- Oversees storage cluster configurations, monitoring, rebalances, and rebuilds
- Highly available, independent cluster installed on 3 or 5 different nodes
- May reside alongside SDCs and/or SDSs, or on separate nodes
- Sits outside the data path

Gateway

- Performs installation and configuration checks
- Acts as an endpoint for API calls and passes them to MDM

Direct integration with Kubernetes Dynamic Volume Support

VxFlex OS leverages a Container Storage Interface (CSI) compatible driver with Kubernetes, which supports the broadest set of features for block storage integration. Using storage classes, persistent applications dynamically provision VxFlex OS volumes directly for any persistent volume requirements.

Portability between bare metal and virtualized deployments

VxFlex OS offers a choice of hypervisor and operating system. Kubernetes customers can deploy with a combination of VMware vSphere, Linux, and Windows hosts, and serve block storage as a service to any of those environments. This ability makes it possible to deploy stateful applications with virtualized Kubernetes and transition those applications to bare metal if the applications or infrastructure requires migration.

Supports the most demanding workloads

Stateful applications in container platforms have varied requirements for high availability, performance and protocols. Applications such as Cassandra, Kafka, Elasticsearch, and PostgreSQL often have very high throughput, bandwidth and availability requirements.

VxFlex OS can handle the most demanding workloads in the data center, including databases and analytics platforms.

Deploy VxFlex OS within a Kubernetes cluster, or as external storage

VxFlex OS is extremely lightweight, a small fraction of the CPU and RAM available in the servers; therefore, you can easily deploy VxFlex OS to run on both virtualized and bare metal systems as part of the Kubernetes cluster itself. Conversely, when running large multi-tenant environments, VxFlex OS can be attractive to run a separate storage cluster, apart from Kubernetes. You can deploy VxFlex OS on separate systems as well, with effectively no performance delta between the two deployments.
Introduction

Dynamically upgrade and scale for day-2 operations

The VxFlex OS architecture supports maintenance, lifecycle, and dynamic scalability for container platforms. You can expand the cluster dynamically, adding capacity and performance to an existing environment and increase the performance of existing volumes.

In addition, you can replace VxRack FLEX or VxFlex Ready Node systems with newer hardware, without the need to migrate, take downtime, or reconfigure existing systems.
Before you begin

Understanding the material in this document requires prior knowledge on containers, Kubernetes and VxFlex OS. These concepts are not covered in detail in this document, however, links to helpful resource on these topics are provided throughout the document and in Technical Support and Resources.

2.1 Prerequisites

Ensure you have understood and completed the necessary prerequisites before proceeding to the demonstration sections:

- Installation and configuration of VxFlex OS cluster per best practices
- Kubernetes 1.9.x, 1.10.x or 1.11.x cluster installed with Beta APIs enabled
  - May be installed manually or with a certified installer such as Kubeadm
- VxFlex OS storage data client (SDC) deployed and configured on each Kubernetes worker/slave node
  - Kubernetes worker/slave nodes may be virtual machines or bare metal nodes
- Helm installed in your Kubernetes cluster

2.2 Introduction to Helm

Helm, a Cloud Native Computing Foundation (CNCF) project, is a package manager for Kubernetes. With the help of Helm Charts you can define, install, and upgrade complex Kubernetes applications. Charts are easy to create, version, share, and publish for use by developers and operators.

A repository of community supported Helm Charts is available at https://github.com/helm/charts.

Note: For more information about Helm, Helm Charts and installation instructions, see https://helm.sh/.
3 Container Storage Interface (CSI) driver for VxFlex OS

Container Storage Interface (CSI) is a community driven standard for persistent storage on container orchestrators (COs) like Kubernetes. It enables storage providers to develop a single CSI driver for any CO that has implemented CSI. This lets you dynamically provision storage through a Kubernetes PersistentVolumeClaim (PVC).

Note: For more information about CSI with Kubernetes, see https://kubernetes-csi.github.io/docs/Home.html.

3.1 Kubernetes architecture with VxFlex OS CSI driver

The Kubernetes cluster for this demo was built with one master node and 3 worker nodes deployed on VxFlex OS Ready Nodes. Figure 2 shows the logical view of a CSI driver on a Kubernetes cluster with three slave nodes. The node plugin runs on each node and the controller plugin runs on any one of the nodes. In this demo, the controller plugin is running on Node3. Figure 3 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 2  Logical view of a CSI plugin on a Kubernetes cluster with 3 slave nodes.
3.2 Installing the VxFlex OS CSI driver using Helm

After deploying the Kubernetes cluster with the SDC in each worker node and installing Helm, you can easily deploy the VxFlex OS CSI driver using the available Helm chart for VxFlex OS CSI installation.

**Note:** VxFlex OS CSI driver is open source and therefore support is provided only by the GitHub repository maintainers.

1. Using a command shell, add the VxFlex OS Helm repository to your environment:

   
   ```
   $ helm repo add vxflex https://vxflex-os.github.io/charts
   ```

2. Install the CSI driver by providing the required values in `vxflex.yml` file.

   **Note:** Sample `vxflex.yml` is shown below, modify the values as per your environment.

   ```
   $ helm install --name vxflex-csi --values=vxflex.yml vxflex/vxflex-csi
   ```

   ```yaml
   $ cat vxflex.yml
   systemName: VxFlex_K8s
   username: admin
   password: dellemc@123
   restGateway: https://192.168.11.69
   storagePool: SP01
   volumeNamePrefix: vxvol
   ```
3. Now, verify the pods. It created 3 agent pods on each of the nodes and one controller pod on k8s-slave3 node.

   $ kubectl get pods -o wide --namespace default

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
<th>IP</th>
<th>NODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>vxflex-csi-agent-4nl86</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>7d</td>
<td>192.168.11.82</td>
<td>k8s-slave2</td>
</tr>
<tr>
<td>vxflex-csi-agent-b9fcm</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>7d</td>
<td>192.168.11.83</td>
<td>k8s-slave3</td>
</tr>
<tr>
<td>vxflex-csi-agent-nftnx</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>7d</td>
<td>192.168.11.81</td>
<td>k8s-slave1</td>
</tr>
<tr>
<td>vxflex-csi-controller-0</td>
<td>3/3</td>
<td>Running</td>
<td>0</td>
<td>7d</td>
<td>10.244.3.3</td>
<td>k8s-slave3</td>
</tr>
</tbody>
</table>

4. Verify the storage class created by VxFlex OS.

   Note: VxFlex OS was previously branded as ScaleIO and the CSI driver was developed prior to the rebranding.

   $ kubectl get storageclass

<table>
<thead>
<tr>
<th>NAME</th>
<th>PROVISIONER</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>vxflex (default)</td>
<td>csi-scaleio</td>
<td>7d</td>
</tr>
</tbody>
</table>
4 PostgreSQL example deployment

PostgreSQL is a general purpose and object-relational database management system. It is one of the widely used open-source database systems.

The following section demonstrates the successful deployment of PostgreSQL using the Container Storage Interface (CSI) driver with Helm package manager for Kubernetes.

4.1 Installing PostgreSQL using Helm

1. Run the PostgreSQL using Helm with the following command:

   Note: With Helm, add a specific version of a given chart with the `--version` switch. If you do not provide any version, it downloads the latest chart available in the repo. Additional switches are available to configure parameters of the Cassandra Helm chart to customize the deployment.

   \$ helm install stable/postgresql

   Note: The following command output is truncated. The full output shows all resources (deployments, pods, secrets, persistent volume claims, config maps, services, and so on) deployed by the Helm chart and more notes for interacting with the deployment. For more information, go to: https://kubernetes-csi.github.io/docs/Home.html

   NAME: yucky-mole
   LAST DEPLOYED: Fri Sep 14 03:11:04 2018
   NAMESPACE: default
   STATUS: DEPLOYED
   ...

2. Verify that the PostgreSQL pod is running.

   \$ kubectl get pods -o wide | grep "postgresql"

   yucky-mole-postgresql-647dcf66c6-lqkr 1/1 Running 0 21h
   10.244.3.8 k8s-slave3 <none>

3. Verify the PersistentVolumeClaim was created for PostgreSQL container.

   \$ kubectl get pvc

   NAME STATUS VOLUME CAPACITY ACCESS MODES
   STORAGECLASS AGE
   yucky-mole-postgresql Bound vxvol-01a95d36bb 8Gi RWO vxflex 27m

4. In addition, verify that the volume was created using VxFlex OS GUI.

Note: The 8 GB volume `vxvol-01a95d36bb` was dynamically created for the PostgreSQL pod and mounted to the SDC.
5. The following command helps you know the cluster IP, port number, number of replicas, and the deployment information for the PostgreSQL.

```
$ kubectl get all -l release="yucky-mole"
```

---

### 4.2 Testing PostgreSQL database with pgbench script.

1. The `pgbench_script.sh` script helps in basic validation of Helm deployed Cassandra instances in Kubernetes environment. Download `pgbench_script.sh` from the following link:


2. Add execute permissions to the script.

   ```bash
   $ chmod +x pgbench_script.sh
   ```

3. Initialize the `pgbench_script.sh` script to generate random data on the PostgreSQL database.

   ```bash
   $ ./pgbench_script.sh yucky-mole init
   ```

---

**Note:** The `pgbench_script.sh` script outputs the `kubectl` commands being run to show how it is interacting with the system.

```
$ ./pgbench_script.sh yucky-mole init

/usr/bin/kubectl run --namespace default yucky-mole-postgresql-pgbench-init --restart=Never --rm --tty -i --image postgres --env
```
"PGPASSWORD=aglu1Gs0re" --command -- pgbench -i -s 100 -U postgres -h yucky-mole-postgresql postgres
If you don't see a command prompt, try pressing enter.
100000 of 10000000 tuples (1%) done (elapsed 0.07 s, remaining 6.53 s)
200000 of 10000000 tuples (2%) done (elapsed 0.16 s, remaining 7.76 s)
...
9900000 of 10000000 tuples (99%) done (elapsed 9.34 s, remaining 0.09 s)
10000000 of 10000000 tuples (100%) done (elapsed 9.44 s, remaining 0.00 s)
vacuum...
set primary keys...
done.
pod "yucky-mole-postgresql-pgbench-init" deleted

4. **Benchmark the storage using** `pgbench_script.sh`.

**Note:** This script runs for 80 clients, each client running 5000 per transactions and a single thread.

```
$ ./pgbench_script.sh yucky-mole bench

/usr/bin/kubectl run --namespace default yucky-mole-postgresql-pgbench --
restart=Never --rm --tty -i --image postgres --env "PGPASSWORD=aglu1Gs0re"
--command -- pgbench -c 80 -t 5000 -U postgres -h yucky-mole-postgresql postgres
If you don't see a command prompt, try pressing enter.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 100
query mode: simple
number of clients: 80
number of threads: 1
number of transactions per client: 5000
number of transactions actually processed: 400000/400000
...
tps = 3878.014577 (including connections establishing)
tps = 3878.100832 (excluding connections establishing)
pod "yucky-mole-postgresql-pgbench" deleted
```
5. Open a shell to the database and verify the data by running the following command:

```
$ ./pgbench_script.sh yucky-mole shell
```

If you don't see a command prompt, try pressing enter.

```
postgres=# select * from pgbench_history;
```

4.3 Demonstrating benefits of persistent storage with PostgreSQL

We are going to demonstrate the behavior of persistent volumes if there is a pod failure or crash due to an unforeseen situation. The pod was deployed as a replica set, as defined by the `stable/postgresql` Helm chart, which ensures that specified numbers of replicas should be running at any given point. In the following example, we terminate a pod. It automatically deploys the pod on another node and dynamically maps the existing `PersistentVolumeClaim`.

1. Check that the current state matches the wanted state of the replica set.

```
$ kubectl get replicaset
```

```
NAME                     DESIRED   CURRENT   READY   AGE
yucky-mole-postgresql-647dcf66c6 1         1         1   10d
```

2. Open two shells and watch the state of the containers to see what happens when it is terminated.

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

```
yucky-mole-postgresql-647dcf66c6-lqkq 1/1 Running 0 21h 10.244.3.8
k8s-slave3 <none>
```

3. In one shell, run the command to terminate the pod and ensure that it moves to another host.

```
$ ./pgbench_script.sh yucky-mole kill-and-move
```

```
/usr/bin/kubectl taint node k8s-slave3 key=value:NoSchedule &&
/usr/bin/kubectl delete pod yucky-mole-postgresql-647dcf66c6-lqkqs
node/k8s-slave3 tainted
pod "yucky-mole-postgresql-647dcf66c6-lqkqs" deleted
```
4. In the second shell, observe that pod is terminated and recreated on another host.

   $ kubectl get pods -l release='yucky-mole' -o wide -w

   NAME                                 READY   STATUS   RESTARTS   AGE
   ---                                 ------   --------  -----------   -----
   yucky-mole-postgresql-647dcf66c6-1qkqs 1/1      Running 0 23h
   10.244.3.8   k8s-slave3 <none>
   yucky-mole-postgresql-647dcf66c6-1qkqs 1/1      Terminating 0 1d
   10.244.3.8   k8s-slave3 <none>
   yucky-mole-postgresql-647dcf66c6-rwjxm 0/1  Pending 0 0s
   <none>   k8s-slave1 <none>
   yucky-mole-postgresql-647dcf66c6-rwjxm 0/1  ContainerCreating 0 0s
   <none>   k8s-slave1 <none>

5. The Pod has been moved from k8s-slave3 to k8s-slave1.

   $ kubectl get pods -l release="yucky-mole" -o wide -w

   NAME                                 READY   STATUS   RESTARTS   AGE
   ---                                 ------   --------  -----------   -----
   yucky-mole-postgresql-647dcf66c6-rwjxm 1/1      Running 0 4m
   10.244.1.14   k8s-slave1 <none>

6. Validate that the data is still available.

   $ ./pgbench_script.sh yucky-mole shell

   /usr/bin/kubectl run --namespace default yucky-mole-postgresql-pgbench-shell --restart=Never --rm --tty -i --image postgres --env
   "PGPASSWORD=ag1u1Gs0re" --command -- psql -U postgres -h yucky-mole-postgresql postgres

   If you don't see a command prompt, try pressing enter.
   postgres=# select * from pgbench_history;
   tid | bid | aid | delta | mtime | filler
   +----------------+-----------------+-----------------+-------+-------------------+-------
   251 |  94 | -482 | 2018-09-18 06:37:06.212423 |
   719 | 100 | -689 | 2018-09-18 06:37:06.210452 |
   823 |  67 |  448 | 2018-09-18 06:37:06.211361 |
   253 |  37 |  309 | 2018-09-18 06:37:06.21243 |
   371 |  12 | -471 | 2018-09-18 06:37:06.21119 |
   608 |  28 | 1352 | 2018-09-18 06:37:06.21177 |
   609 |  79 | 2808 | 2018-09-18 06:37:06.21036 |
   106 |  33 |  912 | 2018-09-18 06:37:06.21286 |
   671 |  92 | 2804 | 2018-09-18 06:37:06.21119 |
   425 |  50 | 1284 | 2018-09-18 06:37:06.21235 |
   728 |  47 |  291 | 2018-09-18 06:37:06.20959 |
   807 |  51 | 2234 | 2018-09-18 06:37:06.20976 |
   569 |  61 | 2743 | 2018-09-18 06:37:06.21064 |
   498 |  76 |  797 | 2018-09-18 06:37:06.21000 |
   382 |  90 | 2533 | 2018-09-18 06:37:06.21273 |
   345 |  80 | 8707846 | -425 | 2018-09-18 06:37:06.20915 |
### PostgreSQL example deployment

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>527</th>
<th>72</th>
<th>2528320</th>
<th>1936</th>
<th>2018-09-18 06:37:06.213047</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>550</td>
<td>78</td>
<td>7613351</td>
<td>3556</td>
<td>2018-09-18 06:37:06.212987</td>
</tr>
</tbody>
</table>

...
5 Cassandra example deployment

Cassandra is a linear scaling database with high availability and proven fault-tolerance. It is one of the widely used open-source database systems for mission-critical data. The following section demonstrates the successful deployment of Cassandra using the Container Storage Interface (CSI) driver with Helm package manager for Kubernetes.

5.1 Installing Cassandra using Helm

1. Install Cassandra using Helm with the following command:

   Note: With Helm, you can add a specific version of a given chart with the --version switch. If you do not provide any version, it downloads the latest chart available in the repo. Additional switches are available to configure parameters of the Cassandra Helm chart to customize the deployment.

   $ helm install incubator/cassandra

   Note: The following command output is truncated. The full output shows all resources (deployments, pods, secrets, persistent volume claims, config maps, services, and so on) deployed by the Helm chart and additional notes for interacting with the deployment.

   NAME: muddled-molly
   LAST DEPLOYED: Tue Oct 9 04:23:24 2018
   NAMESPACE: default
   STATUS: DEPLOYED
   ...

2. Verify that the pods were created on each of the slave nodes.

   Note: A StatefulSet pod is deployed on every slave node by the Cassandra Helm Chart.

   $ kubectl get pods -o wide | grep 'muddled-molly'

   muddled-molly-cassandra-0               1/1      Running 0
   11m  10.244.3.22  k8s-slave3  <none>
   muddled-molly-cassandra-1               1/1      Running 0  9m
   10.244.1.32  k8s-slave1  <none>
   muddled-molly-cassandra-2               1/1      Running 0  7m
   10.244.2.40  k8s-slave2  <none>

3. Verify the storage was created for Cassandra container.

   $ kubectl get pvc

   NAME                             STATUS    VOLUME      CAPACITY
   ACCESS MODES STORAGECLASS AGES
   data-muddled-molly-cassandra-0   Bound    vxvol-97082578cb 16Gi
   RWO vxflex 11m
   data-muddled-molly-cassandra-1   Bound    vxvol-ea81dab6cb 16Gi
   RWO vxflex 9m
Cassandra example deployment

4. Verify the storage was created using the VxFlex OS GUI.

**Note:** Three volumes, each with 16 GB, where created by the Cassandra highlighted in blue in Figure 5.

![VxFlex OS GUI showing the dynamically created volumes through a Kubernetes PersistentVolumeClaim](image)

5. The following command helps you know the Cassandra port number, number of replicas, and deployment information.

```sh
$ kubectl get all -l release="muddled-molly"
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>pod/muddled-molly-cassandra-0</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>16m</td>
</tr>
<tr>
<td>pod/muddled-molly-cassandra-1</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>13m</td>
</tr>
<tr>
<td>pod/muddled-molly-cassandra-2</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>11m</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>service/muddled-molly-cassandra</td>
<td>ClusterIP</td>
<td>None</td>
<td>&lt;none&gt;</td>
<td>16m</td>
</tr>
<tr>
<td>7000/TCP, 7001/TCP, 7199/TCP, 9042/TCP, 9160/TCP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>statefulset.apps/muddled-molly-cassandra</td>
<td>3</td>
<td>3</td>
<td>16m</td>
</tr>
</tbody>
</table>

5.2 Testing Cassandra with Cassandra Stress Script

1. The `csstress_script.sh` script helps in basic validation of Helm deployed Cassandra instances in Kubernetes environment. Download `csstress_script.sh` from the given link:

2. Add execute permissions to script.

```
$ chmod +x csstress_script.sh
```

3. Initialize the `csstress_script.sh` script that creates data on Cassandra stress tool.

**Note:** The `csstress_script.sh` script outputs the `kubectl` commands being run to show how it is interacting with the system.

```
$ ./csstress_script.sh muddled-molly init
```

```
/usr/bin/kubectl run --namespace default muddled-molly-cassandra-stress-init --restart=Never --rm --tty -i --image cassandra --command --
cassandra-stress write n=100000 -rate threads=64 -node muddled-molly-cassandra
If you don't see a command prompt, try pressing enter.

*************** Stress Settings ***************
Command:
  Type: write
  Count: 100,000
  No Warmup: false
  Consistency Level: LOCAL_ONE
  Target Uncertainty: not applicable
  Key Size (bytes): 10
  Counter Increment Distribution: add=fixed(1)
Rate:
  Auto: false
  Thread Count: 64
  OpsPer Sec: 0
Population:
  Sequence: 1..100000
  Order: ARBITRARY
  Wrap: true
Insert:
  Revisits: Uniform: min=1,max=1000000
  Visits: Fixed: key=1

```

```
<table>
<thead>
<tr>
<th></th>
<th>total</th>
<th></th>
<th>total</th>
<th></th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>69514, 18363,</td>
<td>18363, 18363,</td>
<td>3.5, 3.4,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.6</td>
<td>8.5, 20.8,</td>
<td>22.4, 4.0,</td>
<td>0.07974,</td>
<td>0, 0, 0,</td>
<td></td>
</tr>
<tr>
<td>0, 0, 0, 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>90344, 20830,</td>
<td>20830, 20830,</td>
<td>3.1, 2.6,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.9</td>
<td>13.9, 23.6,</td>
<td>29.6, 5.0,</td>
<td>0.06942,</td>
<td>0, 0, 0,</td>
<td></td>
</tr>
<tr>
<td>0, 0, 0, 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>100000, 26327,</td>
<td>26327, 26327,</td>
<td>2.4, 2.1,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td>7.3, 19.0,</td>
<td>26.6, 5.4,</td>
<td>0.08947,</td>
<td>0, 0, 0,</td>
<td></td>
</tr>
<tr>
<td>0, 0, 0, 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Results:
```
Op rate : 18,633 op/s [WRITE: 18,633 op/s]
Partition rate : 18,633 pk/s [WRITE: 18,633 pk/s]

END

pod "muddled-molly-cassandra-stress-init" deleted

4. **Benchmark the storage using the csstress_script.sh.**

```
$ ./csstress_script.sh muddled-molly bench
```

```
/usr/bin/kubectl run --namespace default muddled-molly-cassandra-stress-bench --restart=Never --rm --tty -i --image cassandra --command --cassandra-stress read n=100000 -rate threads=64 -node muddled-molly-cassandra
```

If you don't see a command prompt, try pressing enter.

```
************************************************** Stress Settings **************************************************
Command: Type: read
          Count: 100,000
          No Warmup: false
          Consistency Level: LOCAL_ONE
          Target Uncertainty: not applicable
          Key Size (bytes): 10
          Counter Increment Distribution: add=fixed(1)
Rate:  
          Auto: false
          Thread Count: 64
          OpsPer Sec: 0
Population: Distrubution: Gaussian:
          min=1,max=100000,mean=50000.500000,stdev=16666.500000
          Order: ARBITRARY
          Wrap: false

          ...

          total,         72712, 15061, 15061, 15061, 4.2, 2.7,
          12.2, 40.3, 59.2, 64.9, 6.0, 0.17178, 0, 0,
          0, 0, 0, 0
          total, 91941, 19229, 19229, 19229, 3.3, 2.9,
          7.0, 13.4, 36.1, 42.8, 7.0, 0.15204, 0, 0, 0,
          0, 0, 0
          total, 100000, 19661, 19661, 19661, 3.2, 2.5,
          8.0, 18.7, 27.7, 33.3, 7.4, 0.13751, 0, 0, 0,
          0, 0, 0

Results:
Op rate : 13,495 op/s [READ: 13,495 op/s]
Partition rate : 13,495 pk/s [READ: 13,495 pk/s]
Cassandra example deployment

5. Open a shell to the database and verify the data using the following command:

$ ./csstress_script.sh muddled-molly shell

/usr/bin/kubectl run --namespace default muddled-molly-cassandra-stress-shell --restart=Never --rm --tty -i --image cassandra --command -- cqlsh muddled-molly-cassandra

If you don't see a command prompt, try pressing enter.

cqlsh> select * from system_schema.keyspaces;

<table>
<thead>
<tr>
<th>keyspace_name</th>
<th>durable_writes</th>
<th>replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>system_auth</td>
<td>True</td>
<td>{'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '1'}</td>
</tr>
<tr>
<td>system_schema</td>
<td>True</td>
<td>{'class': 'org.apache.cassandra.locator.LocalStrategy'}</td>
</tr>
<tr>
<td>keyspace1</td>
<td>True</td>
<td>{'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '1'}</td>
</tr>
<tr>
<td>system_distributed</td>
<td>True</td>
<td>{'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '3'}</td>
</tr>
<tr>
<td>system</td>
<td>True</td>
<td>{'class': 'org.apache.cassandra.locator.LocalStrategy'}</td>
</tr>
<tr>
<td>system_traces</td>
<td>True</td>
<td>{'class': 'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '2'}</td>
</tr>
</tbody>
</table>

(6 rows)

6. Open a shell to the database and verify the data using the following command:

cqlsh> describe keyspace1;

CREATE KEYSPACE keyspace1 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'} AND durable_writes = true;

CREATE TABLE keyspace1.counter1 (  
  key blob,  
  column1 text,  
  "C0" counter static,  
  "C1" counter static,  
  "C2" counter static,  
  "C3" counter static,  
  "C4" counter static,  
  value counter,  
  PRIMARY KEY (key, column1) ) WITH COMPACT STORAGE  
  AND CLUSTERING ORDER BY (column1 ASC)
AND bloom_filter_fp_chance = 0.01
AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
AND comment = ''
AND compaction = {'class':
'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',
'max_threshold': '32', 'min_threshold': '4'}
AND compression = {'enabled': 'false'}
AND crc_check_chance = 1.0
AND dclocal_read_repair_chance = 0.1
AND default_time_to_live = 0
AND gc_grace_seconds = 864000
AND max_index_interval = 2048
AND memtable_flush_period_in_ms = 0
AND min_index_interval = 128
AND read_repair_chance = 0.0
AND speculative_retry = '99PERCENTILE';

CREATE TABLE keyspace1.standard1 (  
  key blob PRIMARY KEY,
  "C0" blob,
  "C1" blob,
  "C2" blob,
  "C3" blob,
  "C4" blob
) WITH COMPACT STORAGE
AND bloom_filter_fp_chance = 0.01
AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
AND comment = ''
AND compaction = {'class':
'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',
'max_threshold': '32', 'min_threshold': '4'}
AND compression = {'enabled': 'false'}
AND crc_check_chance = 1.0
AND dclocal_read_repair_chance = 0.1
AND default_time_to_live = 0
AND gc_grace_seconds = 864000
AND max_index_interval = 2048
AND memtable_flush_period_in_ms = 0
AND min_index_interval = 128
AND read_repair_chance = 0.0
AND speculative_retry = '99PERCENTILE';
cqlsh>

5.3 Demonstrating benefits of persistent storage with Cassandra
We are going to demonstrate the behavior of persistent volumes if there is a pod failure or crash due to an unforeseen situation. The pod was deployed as a stateful set, as defined by the incubator/cassandra Helm chart, which ensures that specified numbers of replicas should be running at any given point of time. In the example below we terminate a pod, it automatically deploys the pod on another node and dynamically maps the existing PersistentVolumeClaim.
1. Check that the current state matches the wanted state of the stateful set.

   $ kubectl get statefulset 'muddled-molly-cassandra'

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>muddled-molly-cassandra</td>
<td>3</td>
<td>3</td>
<td>29m</td>
</tr>
</tbody>
</table>

2. Open another shell and watch the state of the containers so that you can see what happens when it is terminated.

   $ kubectl get pods -o wide | grep 'muddled-molly'

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>muddled-molly-cassandra-0</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>38m</td>
<td>10.244.3.22</td>
<td>k8s-slave3</td>
</tr>
<tr>
<td>muddled-molly-cassandra-1</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>35m</td>
<td>10.244.1.32</td>
<td>k8s-slave1</td>
</tr>
<tr>
<td>muddled-molly-cassandra-2</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>33m</td>
<td>10.244.2.40</td>
<td>k8s-slave2</td>
</tr>
</tbody>
</table>

3. Run the csstress_script.sh command to terminate the container and ensure that it moves to another host.

   $ ./csstress_script.sh muddled-molly kill-and-move

   /usr/bin/kubectl taint node k8s-slave3 key=value:NoSchedule &&
   /usr/bin/kubectl delete pod muddled-molly-cassandra-0
   node/k8s-slave3 tainted
   pod "muddled-molly-cassandra-0" deleted

4. In another shell, observe that container is terminated, and recreated on another host.

   $ kubectl get pods -o wide | grep 'muddled-molly'

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>muddled-molly-cassandra-0</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>10.244.3.22</td>
<td>k8s-slave3</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>muddled-molly-cassandra-0</td>
<td>0/1</td>
<td>Terminating</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1h</td>
<td>10.244.3.22</td>
<td>k8s-slave3</td>
</tr>
<tr>
<td>muddled-molly-cassandra-1</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1h</td>
<td>10.244.1.32</td>
<td>k8s-slave1</td>
</tr>
<tr>
<td>muddled-molly-cassandra-2</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>58m</td>
<td>10.244.2.40</td>
<td>k8s-slave2</td>
</tr>
<tr>
<td>muddled-molly-cassandra-0</td>
<td>0/1</td>
<td>ContainerCreating</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>7s</td>
<td>&lt;none&gt;</td>
<td>k8s-slave4</td>
</tr>
</tbody>
</table>
5. Next, see that the pod was moved from k8-slave3 to k8-slave1.

```bash
$ kubectl get pods -o wide | grep 'muddled-molly'

muddled-molly-cassandra-0 1/1 Running 0 2m
10.244.5.9 k8s-slave4 <none>
muddled-molly-cassandra-1 1/1 Running 0 1h
10.244.1.32 k8s-slave1 <none>
muddled-molly-cassandra-2 1/1 Running 0 1h
10.244.2.40 k8s-slave2 <none>
```

6. Validate that data is still available.

```bash
$ ./csstress_script.sh muddled-molly shell

/usr/bin/kubectl run --namespace default muddled-molly-cassandra-stress-shell --restart=Never --rm --tty -i --image cassandra --command -- cqlsh muddled-molly-cassandra
If you don't see a command prompt, try pressing enter.
cqlsh> select * from system_schema.keyspaces;

<table>
<thead>
<tr>
<th>keyspace_name</th>
<th>durable_writes</th>
<th>replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------------+----------------</td>
<td>-------------</td>
<td></td>
</tr>
</tbody>
</table>
```

```bash
```

(6 rows)
cqlsh>
```
Summary

VxRack FLEX and VxFlex Ready Nodes with VxFlex OS are HCI offerings that can replace an enterprise grade SAN using Dell EMC PowerEdge servers and intelligent software. They exhibit balanced and predictable behavior, allow for varying performance and capacity ratios, decouple compute and storage resources, and can scale enormously and non-disruptively.

These HCI platforms deliver consistent, predictable IOPS and latency, eliminating hotspots—an excellent match for many Kubernetes environments as demonstrated in this demo.

The demo shows how easy it is to deploy PostgreSQL and Cassandra databases using helm, the usage of CSI driver in achieving persistent storage for stateful applications and dynamic volume creation. The test scripts also help in demonstrating the read and write transactions that are achieved per sec and show the benefits of deploying the pod as replica set that in turn helps in achieving the HA.
Technical support and resources

For more information about this technology, see the following links:

https://github.com/VxFlex-OS

https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/

https://helm.sh/

https://github.com/helm/charts

https://github.com/container-storage-interface/spec/blob/master/spec.md

https://kubernetes-csi.github.io/docs/Home.html

VxRack FLEX Marketing Web page