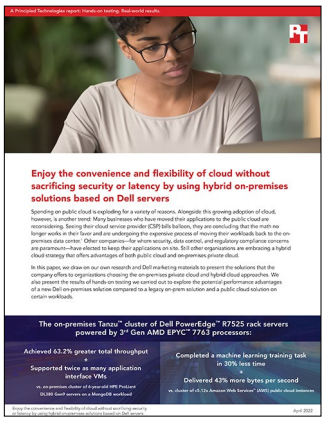

The science behind the report:

Enjoy the convenience and flexibility of cloud without sacrificing security or latency by using hybrid on-premises solutions based on Dell servers

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Enjoy the convenience and flexibility of cloud without sacrificing security or latency by using hybrid on-premises solutions based on Dell servers.

We concluded our hands-on testing on November 19, 2021. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on November 21, 2021 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

To learn more about how we have calculated the wins in this report, go to http://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

Table 1: Results of our testing

	Virtualized MongoDB workload on legacy cluster	Containerized MongoDB workload in Tanzu™ cluster of Dell PowerEdge™ R7525 servers w/ 3rd Gen AMD EPYC™ 7763 processors	Containerized MongoDB workload in Amazon Web Services™ (AWS) Elastic Kubernetes Service (EKS) cluster using c5.12x instances
Total MongoDB throughput in operations per second (higher is better)	241,638	394,466	436,914
Application interface VM density (higher is better)	8	16	-
HiBench machine learning training time in minutes (lower is better)	-	19.25	27.92
HiBench machine learning throughput in bytes per second (higher is better)	-	207,384,363	144,969,448

System configuration information

Table 2: Detailed information on the systems we tested.

System configuration information	4 x HPE ProLiant DL380 Gen9	4 x Dell PowerEdge R7525
BIOS name and version	P89 v2.80	Dell EMC 2.2.5
Non-default BIOS settings	N/A	N/A
Operating system name and version	VMware® ESXi® 7.0 U2 Build 17867351	VMware ESXi 7.0 U2 Build 18426014
Date of last OS updates/patches applied	05/20/2021	05/20/2021
System Profile Settings	Performance	Performance
Processor		
Number of processors	2	2
Vendor and model	Intel® Xeon® E5-2699 v4	AMD EPYC 7763
Core count (per processor)	22	64
Core frequency	2.20 GHz	2.45 GHz
Memory module(s)		
Total memory in system (GB)	128	512
Number of memory modules	8	16
Vendor and model	HPE 809083-091	Hynix® HMA84GR7CJR4N-XN
Size (GB)	32	32
Type	DDR4	DDR4
Speed (MHz)	2,400	3,200
Speed running in the server (MHz)	2,400	3,200
Storage controller		
Vendor and model	HPE Smart Array P440ar	N/A
Cache size (GB)	2	N/A
Firmware version	7.00	N/A
Driver version	70.0051.0.100-2vmw.702.0.0.17867351	N/A
Local storage (Cache)		
Number of drives	2	2
Drive vendor and model	HPE MK000480GWCEV	Dell Express Flash PM1725a
Drive size	480 GB	1.6 TB
Drive information (speed, interface, type)	6 Gbps, SAS, SSD	PCIe® Gen4, SSD

System configuration information	4 x HPE ProLiant DL380 Gen9	4 x Dell PowerEdge R7525
Local storage (Capacity)		
Number of drives	4	4
Drive vendor and model	HPE MK001920GWCFB	Dell Ent NVMe™ AGN RI U.2
Drive size (TB)	1.92	1.92
Drive information (speed, interface, type)	6 Gbps, SATA, SSD	PCIe Gen4, SSD
Network adapter		
Vendor and model	HPE FlexFabric 10Gb	Broadcom Adv Dual 25Gb Ethernet
Number and type of ports	4 x 10Gb	2 x 25Gb
Firmware version	7.18.80	21.65.33.33
Power supplies		
Vendor and model	HPE 720479-B21	Dell PWR SPLY,1400W,RDNT,ARTESYN
Number of power supplies	2	2
Wattage of each (W)	800	1,400

Table 3: Detailed information on the AWS EKS cluster we used.

AWS EKS cluster	
General information	
Date testing ended	11/28/21
Cloud service provider (CSP)	AWS
Region	us-east
Workload information	
Workload name and version	Yahoo Cloud Serving Benchmark (YCSB) / k-means
Iterations and result choice	3 test runs, median reported
Cloud VM or instance details	
Number of VMs	16
VM or instance size	c5.12xlarge
BIOS name and version	Amazon EC2 1.0
vCPU	48
Number of cores/threads	24/48
Memory (GB)	96
Underlying processor model	Intel Xeon Platinum 8275CL
vNIC information/underlying NIC speed	1 x 10Gb Elastic Network Interface
vNIC driver version	1.0

AWS EKS cluster	
Other instance hardware or parameter details	Part of an EKS nodegroup
Operating system information	
Image or template name and UUID	Amazon Linux® 2 AMI
Operating system name	Amazon Linux 2
Kernel version	5.4.149-73.259.amzn2.x86_64
Date patches last applied	N/A
Changes made from CSP image	No
Instance storage (volume type 1)	
Number of volumes	1
Volume use in this test	OS
CSP volume type	Gp3
Volume size (GB)	500

How we tested

In our labs at PT, we set up a legacy four-node VMware vSAN™ cluster using HPE ProLiant DL380 Gen9 servers, a four-node VMware vSphere® with Tanzu vSAN cluster running Kubernetes using Dell PowerEdge R7525 servers, and an Amazon EKS cluster. Additionally, we set up a hybrid cloud using NSX Advanced Load Balancer (ALB) and Global Server Load Balancing (GSLB). The goal of our testing was to show three things: (1) while running the k-means dataset test from the HiBench suite, performance with the vSphere with Tanzu cluster is comparable to or better than performance with the AWS VMs we tested, (2) upgrading from a legacy vSAN cluster to a vSphere with Tanzu vSAN cluster that can give you improved MongoDB performance with Kubernetes, and (3) you can use NSX ALB to create a hybrid cloud between your on-prem environments and AWS so that when you experience an influx of application traffic, GSLB can burst the traffic to the cloud and relieve congestion on your on-prem setup.

To accomplish our goal, we started by configuring two Dell 10Gb switches so that each server in the legacy and Tanzu setup had two network paths each going to the servers. We also set up the client servers running on a vSAN cluster in this same way. We created VLANs for vSAN, VMware vSphere vMotion®, and NSX ALB, and applied them to all of the ports connected to the systems under tests and clients to isolate traffic. We installed ESXi 7.0.2 (OEM custom) in each environment, and added the nodes in the clusters to their own respective VMware vCenter® consoles. After the hypervisor installation was complete, we set up vSAN on the legacy cluster and Tanzu using vSAN on the upgraded cluster. Once installation and configuration was complete, we moved on to our testing phases.

We started with the k-means testing between our Tanzu cluster on-prem and our AWS EKS cluster in the cloud. We created each of our clusters with 16 worker nodes, each node utilizing 24 cores and 96GB of RAM. Because in AWS you only get half the cores above a certain instance size (16 vCPUs), we decided to test AWS with the c5.12xlarge instances for our EKS nodes, and we disabled hyperthreading on each Kubernetes worker node so that we would have 24 cores on both. We then created the containers for testing using the scripts provided. There were three types of servers in our MongoDB sharded cluster: Config Servers, Mongos Servers, and Mongod Servers. We used the same "mongod.conf" configuration file for all three types of servers.

Once we finished with the K-Means testing, we moved on to the MongoDB testing between the three environments. We started on-prem, and used the legacy cluster to deploy MongoDB in VMs and the Tanzu cluster to deploy MongoDB using Kubernetes containers. In our cloud setup on EKS, we used the same scripts from our Tanzu environment, with slight modifications for storage classes, to deploy the same MongoDB containers. We then used YCSB to load and test a 500-million-record database, which is roughly 600GB.

Finally, we set up basic redundant web servers in our legacy on-prem environment and the same web servers in AWS Elastic Compute Cloud (EC2). We then deployed NSX ALB controllers in both locations and created virtual services for the web application using the two web servers in each environment as the backend pool. After that, we also setup a DNS virtual service on prem to which we forwarded our requests from our corporate DNS. We set our corporate DNS to be the authority of the subdomain we used in our GSLB web service. We also added A records for our web servers and a forwarder to our DNS virtual service in NSX. Once that was complete and everything could communicate, we sent traffic to the FQDN of our GSLB web service, and then pushed more traffic until it triggered our alert and the traffic "burst" to AWS.

Below are detailed instructions on how we set up our environments and ran the tests.

Setting up the legacy cluster

Installing VMware ESXi

1. Boot to the VMware ESXi installation media using a USB stick or CD/DVD.
2. Click Enter to continue.
3. Click F11 key to accept and continue.
4. Under Storage Device, select the installation drive, and click Enter to continue.
5. Select US Default for keyboard layout, and click Enter to continue.
6. Enter root password twice, and click Enter to continue.
7. At the Confirm Install window, click F11 to install.
8. At the Installation Complete window, click Enter to reboot.
9. After reboot, click F2 to Configure System.
10. Log in with root user/password, and click Enter.
11. Scroll to Configure Management Network, and click Enter.
12. Scroll to IPv4 Configuration, and click Enter.
13. Scroll to Static IPv4, and use spacebar to select it.
14. Set the IPv4 address, Subnet Mask, and Default gateway.
15. Click Enter for OK to continue.
16. Scroll to IPv6 Configuration, and click Enter.
17. Scroll to Disable IPv6, and use spacebar to select it.
18. Click Enter for OK to continue.
19. Scroll to DNS Configuration, and click Enter.

20. Scroll to manually configure DNS, and use spacebar to select it.
21. Add Primary DNS Server, Alternate DNS Server, and provide the hostname for the system.
22. Scroll to Custom DNS Suffixes, and click Enter.
23. Add the suffix that is required for testing, and click Enter for OK.
24. Click ESC to accept the changes.
25. Click Y to confirm changes. The system will reboot.
26. After reboot, click F2 to Configure System.
27. Log in with root user/password, and click Enter for OK.
28. Scroll to Troubleshooting Options, and click Enter.
29. Select Enable ESXi Shell, and click Enter to enable it.
30. Select Enable SSH, and click Enter to enable it.
31. Scroll to Restart Management Agents, and click Enter.
32. Click F11 for OK to confirm.
33. Click ESC to exit.
34. Click ESC to log out.

Deploying the VMware vCenter Server® 7.0

1. On a Windows server or VM, locate the VMware-VCSA installer image using a USB stick or CD/DVD.
2. Mount the image, navigate to the vcsa-ui-installer folder, and double-click win32.
3. Double-click installer.exe.
4. Click Install.
5. Click Next.
6. Accept the terms of the license agreement, and click Next.
7. Leave the default Center Server with an Embedded Platform Services Controller selected, and click Next.
8. Enter the FQDN or IP address of the host onto which you are deploying the vCenter Server Appliance.
9. Provide the server's username and password, and click Next.
10. To accept the certificate of the host you chose to connect to, click Yes.
11. Provide a name and password for the vCenter Appliance, and click Next.
12. Set an appropriate appliance size, and click Next.
13. Select the appropriate datastore, and click Next.
14. At the Configure Network Settings page, configure the network settings as appropriate for your environment, and click Next.
15. Review your settings, and click Finish.
16. When the deployment completes, click Continue.
17. At the Introduction page, click Next.
18. At the Appliance configuration page, select the time synchronization mode and SSH access settings, and click Next.
19. Select Create a new SSO domain.
20. Provide an SSO Domain name and SSO Site name.
21. Provide a password, confirm it, and click Next.
22. At the CEIP page, click Next.
23. At the Ready to complete page, click Finish.
24. At the Warning pop-up, click OK.
25. Once Stage 2 has completed successfully, click Close.

Creating a cluster and adding the hosts to VMware vCenter

1. Once you have logged into the vCenter, navigate to Hosts and Clusters.
2. Select the primary site management vCenter.
3. Right-click the vCenter object, and select New Datacenter...
4. Enter a name for the new data center, and click OK.
5. Right-click the new data center, and click New Cluster...
6. Name the new cluster Legacy.
7. Leave all of the options untoggled, and click Next.
8. Review the details, and click Finish.
9. Once the cluster has been created, right-click the cluster, and click Add Host.
10. Enter the FQDN or the IP address and root credentials in the spaces provided for the first server.
11. Click Add host and repeat step 10 three more times to add all four servers, and click Next.
12. Check the box beside all four servers to accept the server's certificate, and click OK.
13. On the Host Summary page, click Next.
14. Click Finish.

Configuring the vSAN networking

1. Right-click Datacenter→Distributed Switch→New Distributed Switch.
2. Name the switch Legacy, and click Next.
3. Select 7.0.2, and click Next.
4. Set the number of uplinks to 2.
5. Name the port group vMotion-PG, and click Next.
6. Click Finish.
7. Right-click the Legacy VDS→Distributed Port Group→New Distributed Port Group…
8. Name the port group vSAN-PG, and click Next.
9. In the VLAN type drop-down, select VLAN.
10. Set the VLAN ID to 200, and click Next.
11. Click Finish.
12. Right-click the Legacy VDS→Distributed Port Group→New Distributed Port Group…
13. Name the port group vSAN-PG, and click Next.
14. In the VLAN type drop-down, select VLAN.
15. Set the VLAN ID to 89, and click Next.
16. Click Finish.
17. Right-click the Legacy VDS→Settings→Edit Settings.
18. Click the Advanced tab.
19. Change the MTU size to 9000.
20. Change the discovery protocol to Link Layer Discovery Protocol.
21. Click OK.
22. Right-click vMotion-PG→Edit Settings.
23. Select the VLAN tab.
24. Change the VLAN type to VLAN, and set the VLAN ID to 100.
25. Click OK.
26. Right-click the Legacy VDS, and click Add and Manage Hosts.
27. Select Add Hosts, and click Next.
28. Click New hosts.
29. Select all four hosts, and click OK.
30. Click Next.
31. Select vmnic8, and click Assign uplink.
32. Select Uplink 1, and check the box next to Apply this uplink assignment to the rest of the hosts.
33. Click OK.
34. Select vmnic9, and click Assign uplink.
35. Select Uplink 2, and check the box next to Apply this uplink assignment to the rest of the hosts.
36. Click OK.
37. Click Next three times.
38. Click Finish.
39. In the Hosts and Clusters pane, expand the vSAN cluster.
40. Select the first host, and navigate to the Configure tab.
41. Under Networking, select VMkernel adapters, and click Add Networking.
42. Select VMkernel Network Adapter, and click Next.
43. Under Select an existing network, click Browse.
44. Select vMotion-PG, and click OK.
45. Click Next.
46. Check the box next to the vMotion service, and click Next.
47. Select Use static IPv4 settings, assign an IP address, and click Next.
48. Click Finish.
49. Under Networking, select VMkernel adapters, and click Add Networking.
50. Select VMkernel Network Adapter, and click Next.
51. Under Select an existing network, click Browse.
52. Select vSAN-PG, and click OK.
53. Click Next.
54. Check the box next to the vSAN service, and click Next.
55. Select Use static IPv4 settings, assign an IP address, and click Next.
56. Click Finish.
57. Repeat steps 41 through 56 for the remaining nodes in the vSAN cluster.

Installing and configuring MongoDB on the Config Server VMs

We installed RHEL 8.4 onto each VM, then performed the following steps on each Config Server VM:

1. Log into the VM you are configuring.
2. Stop and disable the firewall service:

```
systemctl stop firewalld
systemctl disable firewalld
```

3. Set SELinux to disabled:

```
sed -i 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config
```

4. Disable transparent hugepages:

```
grub2-editenv - set "$(grub2-editenv - list | grep kernelopts) transparent_hugepage=never"
```

5. Add all host information for all systems into the /etc/hosts file.
6. Reboot the server.
7. Perform system updates:

```
dnf update -y
```

8. Create the mogodb-org.repo file:

```
touch /etc/yum.repos.d/mongodb-org-4.4.repo
```

9. Populate the mongodb-org-4.4.repo file:

```
nano /etc/yum.repos.d/mongodb-org.repo
```

10. Add the following:

```
[mongodb-org-4.4]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.4/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-4.4.asc
```

11. Install MongoDB:

```
yum install -y mongodb-org
```

12. Add the database drive to the application interface VM, and create the database folders for MongoDB:

```
mkfs.xfs /dev/sdb
mkdir -p /var/lib/mongo/data
mount /dev/sdb /var/lib/mongo/data
echo '/dev/sdb  /var/lib/mongo/data xfs  defaults  0  0' >> /etc/fstab
chown mongod:mongod /mnt/var/lib/mongo
```

13. Edit the MongoDB config file at /etc/mongod.conf to match the Config Server mongod.conf file in the Scripts we used in our testing section.

14. Start your mongod server:

```
systemctl start mongod
```

15. Enable the mongod process to start automatically when the server restarts:

```
systemctl enable mongod
```

16. Repeat steps 1 through 15 on your remaining two Config Server VMs.

Creating the config server replica set on the Config Server VMs

1. Log into all three Config Servers and verify that all three are "active (running)."
2. Type `mongo` to enter the MongoDB console on all three Config Servers.
3. In the MongoDB console of only one of the Config Servers, create the replica set:

```
rs.initiate(
  {
    _id : "config-replica-set",
    members: [
      { _id : 0, host : "mongo-config-1:27017" },
      { _id : 1, host : "mongo-config-2:27017" },
      { _id : 2, host : "mongo-config-3:27017" }
    ]
  }
)
```

4. Verify that you have successfully created the config-replica-set by clicking the Enter key several times on all three Config Servers. The command line should change to one of the following:

```
config-replica-set:PRIMARY>
config-replica-set:SECONDARY> (2 of these will be present)
```

Installing and configuring MongoDB on the Mongos Server VMs

We installed RHEL 8.4 onto each VM, then performed the following steps on each Mongos Server VM:

1. Log into the VM you are configuring.
2. Stop and disable the firewall services:

```
systemctl stop firewalld
systemctl disable firewalld
```

3. Set SELinux to disabled:

```
sed -i 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config
```

4. Disable transparent hugepages:

```
grub2-editenv - set "$(grub2-editenv - list | grep kernelopts) transparent_hugepage=never"
```

5. Add all host information for all systems into the /etc/hosts file.
6. Reboot the server.
7. Perform system updates:

```
dnf update -y
```

8. Create the mogodb-org.repo file:

```
touch /etc/yum.repos.d/mongodb-org-4.4.repo
```

9. Populate the mongodb-org-4.4.repo file:

```
nano /etc/yum.repos.d/mongodb-org.repo
```

10. Add the following:

```
[mongodb-org-4.4]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.4/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-4.4.asc
```

11. Install MongoDB:

```
yum install -y mongodb-org
```

12. Edit the mongod config file at /etc/mongod.conf to match the Mongos Server mongod.conf file in the Scripts we used in our testing section.
13. Make mongos start with the operating system:

```
echo "mkdir /var/run/mongodb" >> /etc/rc.d/rc.local
echo "chown -R mongod:mongod /var/run/mongodb" >> /etc/rc.d/rc.local
echo "mongos --config=/etc/mongod.conf" >> /etc/rc.d/rc.local
```

14. Type `mongo` to enter the MongoDB console on the Mongos Server.
15. Verify that the MongoDB console starts in the mongos console. The command line should look like the following:

```
mongos>
```

16. Repeat steps 1-15 on your remaining seven Mongos Server VMs.

Installing and configuring MongoDB on the Mongod Server VMs

We installed RHEL 8.4 onto each VM, then performed the following steps on each Mongod Server VM:

1. Log into the VM you are configuring.
2. Stop and disable the firewall services:

```
systemctl stop firewalld
systemctl disable firewalld
```

3. Set SELinux to disabled:

```
sed -i 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config
```

4. Disable transparent hugepages:

```
grub2-editenv - set "$(grub2-editenv - list | grep kernelopts) transparent_hugepage=never"
```

5. Add all host information for all systems into the /etc/hosts file.
6. Reboot the server.
7. Perform system updates:

```
dnf update -y
```

8. Create the mogodb-org.repo file:

```
touch /etc/yum.repos.d/mongodb-org-4.4.repo
```

9. Populate the mongodb-org-4.4.repo file:

```
nano /etc/yum.repos.d/mongodb-org.repo
```

10. Add the following:

```
[mongodb-org-4.4]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.4/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-4.4.asc
```

11. Install MongoDB:

```
yum install -y mongodb-org
```

12. Add the database drive to the VM, and create the database folders for MongoDBs:

```
mkfs.xfs /dev/sdb
mkdir -p /var/lib/mongo/data
mount /dev/sdb /var/lib/mongo/data
echo '/dev/sdb  /var/lib/mongo/data xfs  defaults  0  0' >> /etc/fstab
chown mongod:mongod /mnt/var/lib/mongo
```

13. Edit the mongod config file at /etc/mongod.conf to match the Mongod Server mongod.conf file in the Scripts we used in our testing section.
14. Start your mongod server:

```
systemctl start mongod
```

15. Enable the mongod process to start automatically when the server restarts:

```
systemctl enable mongod
```

16. Repeat steps 1 through 15 on your remaining 23 Mongod Server VMs.

Creating the config server replica set on the Mongod Server VMs

1. Log into the three Mongod Servers that will make up a three-node replica set, and verify that all three are "active (running)."
2. Type `mongo` to enter the MongoDB console on all three Mongod Servers.
3. In the MongoDB console of only one of the Mongod Servers, create the replica set:

```
rs.initiate(
  {
    _id : "shard-replica-set-##",
    members: [
      { _id : 0, host : "mongod-##-a:27017" },
      { _id : 1, host : "mongod-##-b:27017" },
      { _id : 2, host : "mongod-##-c:27017" }
    ]
  }
)
```

4. Verify that you have successfully created the config-replica-set by clicking the Enter key several times on all three Mongod Servers. The command line should change to one of the following:

```
config-replica-set:PRIMARY>
config-replica-set:SECONDARY> (2 of these will be present)
```

5. Repeat steps 1 through 4 until all 24 servers are configured, using three Mongod Servers per shard-server-set for a total of eight shard-server-sets.

Enabling sharding operations on the Mongos Server VMs

You must enable sharding on the Mongos Server(s) prior to building the database.

1. Log into the first Mongos Server VM you are configuring.
2. Type mongo to enter the MongoDB console on the Mongos Server.
3. Configure two shards on the first Mongos Server:

```
sh.addShard("shard-replica-set-1/<mongod-1-a:27017, mongod-1-b:27017, mongod-1-c:27017")
```

4. Repeat steps 1 through 3 for the remaining seven mongos VMs and mongod replica sets.

Installing and configuring YCSB driver VMs

We installed RHEL 8.4 onto each VM, then performed the following steps on each YCSB driver VM:

1. Log into the VM you are configuring.
2. Stop and disable the firewall service:

```
systemctl stop firewalld
systemctl disable firewalld
```

3. Set SELinux to disabled:

```
sed -i 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config
```

4. Disable transparent hugepages:

```
grub2-editenv - set "$(grub2-editenv - list | grep kernelopts) transparent_hugepage=never"
```

5. Add all host information for all systems into the /etc/hosts file.
6. Reboot the server.

7. Perform system updates:

```
dnf update -y
```

8. Create the mogodb-org.repo file:

```
touch /etc/yum.repos.d/mongodb-org-4.4.repo
```

9. Populate the mongodb-org-4.4.repo file:

```
nano /etc/yum.repos.d/mongodb-org.repo
```

10. Add the following:

```
[mongodb-org-4.4]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.4/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-4.4.asc
```

11. Install MongoDB:

```
yum install -y mongodb-org
```

12. Start your mongod server:

```
systemctl start mongod
```

13. Enable the mongod process to start automatically when the server restarts:

```
systemctl enable mongod
```

14. Install prerequisites for Maven:

```
dnf install nano wget curl nmon java-devel -y
```

15. Install Maven:

```
wget https://mirrors.sonic.net/apache/maven/maven-3/3.8.1/binaries/apache-maven-3.8.1-bin.tar.gz
tar xzf apache-maven-3.8.1-bin.tar.gz -C /usr/local
cd /usr/local/
ln -s apache-maven-* maven
echo "export M2_HOME=/usr/local/maven" >> /etc/profile.d/maven
echo "export PATH=${M2_HOME}/bin:${PATH}" >> /etc/profile.d/maven
```

16. Download and install YCSB:

```
curl -O --location https://github.com/brianfrankcooper/YCSB/releases/download/0.17.0/
ycsb-0.17.0.tar.g
tar xfvz ycsb-0.17.0.tar.gz
```

17. Repeat steps 1 through 16 on your remaining seven YCSB driver VMs.

Creating the YCSB database

1. Log into one of the YCSB driver VMs.
2. Create a database with 30 million records:

```
./bin/ycsb load mongodb -s -P /root/ycsb-0.17.0/workloads/workloadc -threads 32 -p mongodb.
url=mongodb://mongos1:27017/ycsb?w=1 -p recordcount=500000000
```

3. As the system creates the database, log into one of your Mongos Server VMs.
4. Enter MongoDB mongos console:

```
mongo
```

5. Type the following commands to shard and chunk your MongoDB database:

```
use ycsb
sh.enableSharding("ycsb")
sh.shardCollection("ycsb.usertable", { _id: 1 }, true )
```

6. The database will finish building in several hours. After the database is finished building, check on the progress of the shard distribution:

```
sh.status()
```

7. Once all shards have an equivalent number of chunks distributed, you are ready to test.

Running the YCSB test

We used a script to kick off the run on all eight YCSB driver VMs simultaneously.

1. Log into the first YCSB driver VM.
2. To kick off the test, run the following script (see the Scripts we used in our testing section):

```
./run_legacy_mongodb.sh
```

Setting up Tanzu

Installing vSphere 7.0 Update 2 on a Dell PowerEdge R7525

1. Download the Dell Custom Image for ESXi 7.0 Update 2 from the following link: https://my.vmware.com/group/vmware/evalcenter?p=vsphere-eval-7#tab_download.
2. Open a new browser tab, and connect to the IP address of the Dell PowerEdge R7525 server iDRAC.
3. Log in with the iDRAC credentials. We used root/calvin.
4. In the lower left corner of the screen, click Launch Virtual Console.
5. In the console menu bar, click Connect Virtual Media.
6. Under Map CD/DVD, click Browse…, and select the image you downloaded in step 1. Click Open.
7. Click Map Device, and click Close.
8. On the console menu bar, click Boot, and select Virtual CD/DVD/ISO. To confirm, click Yes.
9. On the console menu bar, click Power. Select Power On System. To confirm, click Yes.
10. The system boots to the mounted image and the Loading ESXi installer screen appears. When prompted, click Enter to continue.
11. To Accept the EULA and Continue, click F11.
12. Select the storage device to target for installation. We selected the internal SD card. To continue, click Enter.
13. To confirm the storage targe, click Enter.
14. Select the keyboard layout, and click Enter.
15. Provide a root password, and confirm the password. To continue, click Enter.
16. To install, click F11.
17. Upon completion, to reboot the server, click Enter.

Installing vCenter Server Appliance 7.0 Update 2

1. Download VMware vCenter 7.0 Update 2 from the VMware support portal: https://my.vmware.com.
2. Mount the image on your local system, and browse to the vcsa-ui-installer folder. Expand the folder for your OS, and launch the installer if it doesn't automatically begin.
3. When the vCenter Server Installer wizard opens, click Install.
4. To begin installation of the new vCenter server appliance, click Next.
5. Check the box to accept the license agreement, and click Next.
6. Enter the IP address of one of your newly deployed Dell PowerEdge R7525 servers with ESXi 7.0 Update 2. Provide the root password, and click Next.
7. To accept the SHA1 thumbprint of the server's certificate, click Yes.
8. Accept the VM name, provide and confirm the root password for the VCSA, and click Next.
9. To set the deployment environment size, select Medium, and click Next.
10. Select the datastore to install on, accept the datastore defaults, and click Next.
11. Enter the FQDN, IP address information, and DNS servers you want to use for the vCenter server appliance, and click Next.
12. To begin deployment, click Finish.
13. When Stage 1 has completed, click Close. Click Yes to confirm.
14. Open a browser window, and connect to https://[vcenter.FQDN:5480/.
15. On the Getting Started - vCenter Server page, click Set up.
16. Enter the root password, and click Log in.
17. Click Next.
18. Enable SSH access, and click Next.
19. To confirm the changes, click OK.
20. Enter vsphere.local for the Single Sign-On domain name. Enter a password for the administrator account, confirm it, and click Next.
21. Click Next.
22. Click Finish.

Creating a cluster in vSphere 7.0 Update 2

1. Open a browser, and enter the address of the vCenter server you deployed. For example: https://[vcenter.FQDN]/ui
2. In the left panel, select the vCenter server, right-click, and select New Datacenter.
3. Provide a name for the new data center, and click OK.
4. Select the data center you just created, right-click, and select New Cluster.
5. Name the cluster, enable vSphere DRS, and click OK.
6. In the cluster configuration panel, under Add hosts, click Add.
7. Check the box for Use the same credentials for all hosts. Enter the IP Address and root credentials for the first host, the IP addresses of all remaining hosts, and click Next.
8. Check the box beside Hostname/IP Address to select all hosts, and click Ok.
9. Click Next.
10. Click Finish.

Creating the NSX Controller content library

1. Click the following link to download the NSX Advanced Load Balancer: https://customerconnect.vmware.com/en/downloads/info/slug/networking_security/vmware_nsx_advanced_load_balancer/21_1_x
2. From vSphere client, in the left menu pane, click Content Libraries.
3. In the Content Libraries panel on the right, click Create.
4. Name the content library `NSX Controller`, and click Next
5. Accept the default, and click Next.
6. Choose the storage location for the content library, and click Next.
7. Review, and click Finish.
8. Click the newly created NSX Controller content library.
9. In the upper portion of the right side panel for HAproxy-cl, click the actions drop-down menu, and select Import Item.
10. Change the selection to local file, and click Upload files.
11. Browse to the location of the ovf file you downloaded in step 1, and click Open.
12. Click Import.

Creating the TKG content library

1. From vSphere client, in the left menu pane, click Content Libraries.
2. In the Content Libraries panel on the right, click Create.
3. Name the content library TKG-cl, and click Next.
4. Select Subscribed content library, use https://wp-content.vmware.com/v2/latest/lib.json for the subscription URL., and click Next.
5. To verify, click Yes.
6. Choose the storage location for the content library, and click Next.
7. Review, and click Finish.

Creating the storage tag

1. From the vSphere client, select Menu→Storage.
2. From the left pane, select the vSAN for Tanzu.
3. Under the Summary tab, locate the Tags panel, and click Assign.
4. Click Add Tag.
5. Name the tag Tanzu, and click Create New Category.
6. Give the category the name Tanzu Storage, clear all object types except Datastore, and click Create.
7. Use the Category drop-down menu to select Tanzu Storage, and click Create.
8. Check the box beside the newly created tag, and click Assign.

Creating the VM storage policy

1. From the vSphere client, click Menu→Policies and Profiles.
2. On the left panel, click VM Storage Policies.
3. Click Create.
4. Create a new VM Storage policy named tkg-clusters, and click Next.
5. Check the box for Enable tag based placement rules, and click Next.
6. Use the Tag Category drop-down menu, select the Tanzu Storage policy you created, and click Browse Tags.
7. Click the Tanzu checkbox, and click OK.
8. Click Next.
9. Review the compatible storage, making sure your storage target is marked as compatible, and click Next.
10. Click Finish.

Deploying NSX Advanced Load Balancer for Tanzu load balancing

1. From the vSphere client, click Menu→Content Libraries.
2. Click the NSX Controller library.
3. In the left panel, click OVF & OVA Templates, right-click the NSX template that appears in the panel below, and select New VM from This Template…
4. Provide a simple name (we used `nsx`), select the Datacenter and/or folder you want to deploy to, and click Next.
5. Select the cluster or compute resource where you want to deploy the nsx VM, and click Next.
6. Review details, and click Next.
7. Check the box to accept all license agreements, and click Next.
8. Accept the default configuration, and click Next.
9. Select the target storage for the VM, and click Next.
10. Select VM Network for the Management network, and click Next.
11. Customize the template. We used the following:
    - Management IP: `192.168.10.103`
    - Management subnet: `255.255.255.0`
    - Management gateway: `192.168.10.2`
    - Leave the sysadmin authentication key blank
12. Click Next.
13. Review the summary, and click Finish.
14. Power on the nsx VM.
15. Access the nsx IP (192.168.10.103) in a browser.
16. Enter a password for the admin user account, confirm the password, and click Create Account.
17. In the Welcome Admin page, enter a passphrase and confirm the passphrase. In the DNS resolvers, we entered 10.41.0.10 and clicked Next.

18. For Email/SMTP, select None, and click Next.
19. Leave everything at defaults for Multi-Tenant, and click Save.
20. Ignore and close the Controller Faults error message in the next screen.
21. In the Avi dashboard, click the menu in the upper-left corner, and select Infrastructure.
22. Select the Clouds tab, and click the Edit icon for the Default-Cloud.
23. Click the Select Cloud tab.
24. Select VMware, and click Next.
25. Enter the username, password, and IP address for your vCenter, and click Next
26. In the Data Center tab, select Prefer Static Routes vs Directly Connected Network, and click Save.
27. In the Network tab, select Management Network from the drop-down menu. We selected VM Network.
    - IP Subnet: `192.168.10.0/24`
    - Add Static IP Address Pool: `192.168.10.50-192.168.10.99`
    - Default Gateway: `192.168.10.2`
28. Click Save.
29. In the Avi dashboard, click the menu in the upper-left hand corner, and select Administration.
30. Select Settings→Access Settings, and click the edit icon.
31. Delete the default self-signed certificates in SSL/TLS Certificate.
32. From the SSL/TLS Certificate drop-down menu, select Create Certificate.
33. In the Add Certificate (SSL/TLS) window, enter a name for your certificate. We entered the IP address `192.168.10.103`
34. To add a self-signed certificate, select Type as Self Signed.
35. Enter the following details:
    - Common name: `192.168.10.103`
    - Subject Alternative name (SAN): `192.168.10.103`
    - Algorithm: `EC`
36. Click Save.
37. To download the self-signed certificate that you created, select Templates→Security→SSL/TLS Certificates.
38. Select the certificate you created, and click the download icon.
39. In the Export Certificate page that appears, click Copy to clipboard. You will need this certificate when you enable workload management.
40. In the Avi Controller dashboard, click the menu in the upper-left corner, and select Infrastructure.
41. In the Infrastructure settings page, click Service Engine Group.
42. In the Service Engine Group page, click the edit icon in the Default-Group. The Basic Settings page appears.
43. In the High Availability & Placement Settings section, select the High Availability Mode. We selected N+M(buffer).
44. Click the Advanced tab.
45. In the drop-down menu for Cluster, select the Tanzu cluster you created earlier.
46. Click Save.
47. In the Avi Controller dashboard, click the menu in the upper-left corner, and select Infrastructure.
48. Click Network to display the list of networks on vCenter.
49. Locate the network that provides the virtual IP addresses and click the edit icon to edit the network settings. Select Tanzu Data Network, and click the edit icon.
50. Click + Add Subnet.
51. In the IP subnet field, we entered 192.168.2.0/24, and clicked Add Static IP Address Pool.
52. Select Use Static IP Address for VIPs and SE, and enter `192.168.2.100-192.168.2.149`
53. Click Save twice.
54. In the Avi Controller dashboard, click the menu in the upper-left corner, and select Infrastructure.
55. Click Routing.
56. In the Static Route section, click Create.
57. In Gateway Subnet, enter the subnet for the Workload network. We entered `192.168.1.0/24`
58. In Next Hop, enter the gateway IP address for the Data network. We entered `192.168.2.2`
59. Click Save.
60. In the Avi Controller dashboard, go to Templates→Profile→IPAM/DNS Profiles.
61. Select Create IPAM Profile.

62. Configure the IPAM Profile:
    - Name: ipam-profile
    - Type: AVI Vantage IPAM
    - Allocate IP in VRF: checked
    - Cloud for usable Network: default-cloud
    - Usable Network: Select the Virtual IP network you just created. We selected Tanzu Data Network – 192.168.2.0/24
63. Click Save.
64. Go to Infrastructure→Cloud.
65. Click the Edit icon for the Default-Cloud.
66. In the drop-down menu for IPAM Profile, select the IPAM profile you just created. Leave everything else at defaults.
67. Click Save.
68. In the Avi Controller dashboard, go to Infrastructure→Clouds, and verify the status of the Controller for Default-Cloud is green.
69. The configuration for the NSX Advanced Load Balancer is now complete.

Configuring Workload Management

1. From the vSphere client, click Menu→Workload Management.
2. Click Get Started.
3. Review the messages and warnings regarding supported configurations. Click Next.
4. Select the Cluster on which you want to enable workload management, and click Next.
5. Choose the capacity for the control plane VMs. We chose Small. Click Next.
6. Choose the storage policy you wish to use for the control plane nodes. We chose tkg-clusters. Click Next.
7. Configure the Load Balancer section with the following:
    - Name: nsx
    - Type: avi
    - Data plane API Addresses: 192.168.10.103:443
    - User name: admin
    - Password: <Password>
    - IP Address Ranges for Virtual Servers: 192.168.2.100- 192.168.2.149
    - Server Certificate Authority: Paste the SSL/TLS Certificate you copied from the NSX Controller
8. Click Next.
9. Configure Workload Management with the following:
    - Network: VM Network
    - Starting IP Address: 192.168.10.205
    - Subnet Mask: 255.255.255.0
    - Gateway: 192.168.10.2
    - DNS Server: 10.41.0.10
    - NTP Server: 192.168.10.1
10. Click Next.
11. Configure Workload Network with the following:
    - Leave the default for Services addresses.
    - DNS Servers: 10.41.0.10
    - Under Tanzu Workload Network, click Add.
    - Accept default for network-1.
    - Port Group: Workload Network.
    - Gateway: 192.168.1.2.
    - Subnet: 255.255.255.0
    - IP Address Ranges: 192.168.1.50-192.168.1.99
12. Click Save.
13. For TKG Configuration, do the following:
    - Beside Add Content Library, click Add.
    - Select the TKG-cl library, and click OK.
14. Click Next.
15. Click Finish. The workload management cluster will deploy and configure. You may see apparent errors during configuration, but these will resolve upon successful completion.

Configuring Kubernetes namespace for service deployment

1. In Workload Management, click Namespaces.
2. Click Create Namespace.
3. Select the target cluster, and provide a name. We used `tanzu-ns`. Click Create.
4. Click the Permissions tab, and click Add.
5. Choose vSphere.local for the identity source. Search for Administrator, select the "can edit" role, and click OK.
6. Click the Storage tab.
7. In the Storage Policies section, click Edit.
8. Select the tkg-clusters policy, and click OK. The environment is ready for connection and deploying containers.

Installing and Configuring Ubuntu VM for Tanzu Kubernetes Grid CLI

1. Log into vCenter, and from the drop-down menu, click Storage.
2. Select datastore1, and click Files.
3. Click Upload Files, and upload the Ubuntu 18.04.5 ISO image.
4. Right-click the cluster, and click New Virtual Machine.
5. Click Next.
6. Enter a name for the VM, and click Next.
7. Click Next.
8. Select datastore1, and click Next.
9. Click Next.
10. From the Guest OS Family drop-down menu, select Linux.
11. From the guest OS version drop-down menu, select Ubuntu Linux (64 bit), and click Next.
12. Assign the VM two vCPUs, 8 GB of memory, and a 40GB hard disk.
13. From the New CD/DVD Drive drop-down menu, select Datastore ISO File, and select the Ubuntu ISO you uploaded to the datastore previously. Ensure Connect At Power On is checked, and click Next.
14. Click Finish.
15. Power on the VM, and click Launch Remote Console.
16. Click Install Ubuntu.
17. Click Continue.
18. Select Minimal installation, and click Continue.
19. Click Install now.
20. Click Continue twice.
21. Enter your desired full name, computer name, username, and password, and click Continue.
22. Click Restart Now.
23. Click Enter.
24. Enter your password, and click Sign In.
25. When prompted by the Software Updater, install OS and software updates.
26. Click Restart Later, and power off the VM.
27. Right-click the VM in vCenter, and click Edit Settings.
28. Click Add New Device, and select Network Adapter.
29. From the New Network drop-down menu, select Browse.
30. Click Tanzu Data Network, and click OK.
31. Click OK.
32. Power on the VM, and click Launch Remote Console.
33. Enter your password, and click Sign In.
34. Click the network icon in the top-right corner, and click Ethernet (ens192).
35. Click Wired Settings.
36. Next to Ethernet (ens192) click the gear icon.
37. Click IPv4.
38. Change IPv4 Method to Manual, and enter the following settings:
    - Address: `192.168.2.5`
    - Netmask: `255.255.255.0`
39. Click Apply, and close the settings window.

Installing the Kubectl Plugin and Helm

1. Open a browser on the Ubuntu VM, and navigate to the IP of one of the three Supervisor VMs. In our environment, the first control plane VM IP was 192.168.10.205.
2. Click Advanced, and click Accept the Risk and Continue to bypass the certificate warning.
3. Click Download CLI Plugin Linux.
4. Select Save File, and click OK.
5. Open the Files app, and navigate to the Downloads folder.
6. Right-click vsphere-plugin.zip, and select Extract Here.
7. Open a terminal and navigate to the vsphere-plugin binary within the extracted folder:

```
cd Downloads/vsphere-plugin/bin
```

8. Make the vsphere-plugin binary executable, and add it to PATH:

```
sudo mv kubectl-vsphere /usr/local/bin/
sudo mv kubectl /usr/local/bin/
```

9. Download Helm from https://github.com/helm/helm/releases, and run the following command:

```
cd ~/Downloads/
tar -zxvf helm-v3.6.3-linux-amd64.tar.gz
sudo cp ./linux-amd64/helm /usr/local/bin
```

Creating a Tanzu Kubernetes Cluster

1. Log into the Tanzu CLI VM.
2. Modify tanzu-cluster.yaml to match what's in the Scripts we used in our testing section. The current Kubernetes cluster has three control nodes and 16 worker nodes. Each worker node was deployed as a custom guaranteed-12xlarge size (24 vCPUs and 96 GB of memory). See https://docs.vmware.com/en/VMware-vSphere/7.0/vmware-vsphere-with-tanzu/GUID-977D8CF6-957D-4330-90BD-6D3E184ACF70.html for steps on creating custom Tanzu Virtual Machine Classes.
3. To create a new Kubernetes cluster:

```
kubectl apply -f tanzu-cluster.yaml
```

4. To delete an existing cluster:

```
kubectl delete tkc <cluster name>
```

Deploying MongoDB Pods in Tanzu

1. Log into the Tanzu CLI VM.
2. Log into the Kubernetes cluster you just created:

```
kubectl vsphere login --insecure-skip-tls-verify --vsphere-username administrator@vsphere.local
--server=https://192.168.2.100 --tanzu-kubernetes-cluster-name mongodb-k8s-cluster --tanzu-
kubernetes-cluster-namespace tanzu-ns
```

3. Modify the Helm chart tanzu-mongodb-values.yaml to match what's in the Scripts we used in our testing section.
4. Set up Rolebinding before deploying Mongodb Pods by creating a file called mongo-rbac-tanzu.yaml with the following:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rolebinding-default-privileged-sa-ns_default
roleRef:
  kind: ClusterRole
  name: psp:vmware-system-privileged
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:serviceaccounts
```

5. Apply rolebinding for MongoDB:

```
kubectl apply -f mongo-rbac-tanzu.yaml
```

6. Deploy a new Bitnami/Mongodb-sharded cluster:

```
  helm install mongodb -f tanzu-mongodb-values.yaml bitnami/mongodb-sharded
```

7. After the deployment, wait until all the pods are in running state and ready:

```
  kubectl get pods
```

Loading and running the YCSB workload

1. Log into the Tanzu CLI VM.
2. Get the Pod IP of one of the three Mongos Pods:

```
Kubectl describe pod mongodb-mongodb-sharded-mongos-6f4bbbb89b-cwkr2
```

3. Log into the mongos Pod:

```
kubectl exec -it mongodb-mongodb-sharded-mongos-6f4bbbb89b-cwkr2 -- bash
```

4. Log into mongo shell:

```
mongo --username root --password <Password>
```

5. Create the YCSB database:

```
use ycsb
db.dropDatabase()
db.createCollection("ycsb")
```

6. Create a new user for YCSB database:

```
use ycsb
db.createUser({user: "ptuser", pwd: "<Password>", roles: [{role: "readWrite", db: "ycsb"}]})
```

7. Verify you have successfully created the user:

```
db.users()
```

8. Enable sharding on YCSB database:

```
use ycsb
sh.enableSharding("ycsb")
sh.shardCollection("ycsb.usertable", { _id: 1 }, true )
```

9. Log into one of the YCSB Pods, and load the YCSB database:

```
kubectl exec -it ycsb1 -- bash
cd /root/ycsb-0.17.0
./bin/ycsb load mongodb -s -P workloads/workloadc -threads 64 -p mongodb.url="mongodb://
ptuser:<password>@172.16.9.9/ycsb" -p recordcount=500000000
     (172.16.9.9 is the POD IP of one of the mongos servers)
```

10. Create the YCSB containers and start the YCSB test run from all 16 Pods using the scripts in the Scripts we used in our testing section. Each YCSB pod runs against one of the 16 mongos servers:

```
./setup_ycsb.sh
./run_tanzu_mongo.sh
```

11. We ran the test three times and reported the median run.

Deploying Hadoop Pods on Tanzu

1. From the Ubuntu jumper VM, add the Hadoop repository to Helm:

```
helm repo add stable https://charts.helm.sh/stable
helm repo update
```

2. Create a rolebinding configuration file named hadoop-rbac.yaml for Hadoop and fill it with the following:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: spark
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: spark-role
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["*"]
- apiGroups: [""]
  resources: ["services"]
  verbs: ["*"]
- apiGroups: [""]
  resources: ["configMap"]
  verbs: ["*"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: spark-role-binding
  namespace: default
subjects:
- kind: ServiceAccount
  name: spark
  namespace: default
roleRef:
  kind: Role
  name: spark-role
  apiGroup: rbac.authorization.k8s.io
```

3. Apply hadoop-rbac.yaml:

```
kubectl apply -f hadoop-rbac.yaml
```

4. Edit hadoop-values-tanzu.yaml to match the configuration in the Scripts we used in our testing section.
5. Deploy Hadoop containers with Helm:

```
helm install hadoop -f hadoop-values-tanzu.yaml stable/hadoop
```

Configuring the test Pod

1. Copy one of the ycsb.yaml files to test.yaml and fill it with the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
  - name: test
    image: 0track/ycsb:latest
    ports:
      - containerPort: 80
```

2. Label the Pod as a testing Pod:

```
kubectl label test is-test=true
```

3. Launch the test container:

```
kubectl apply -f test.yaml
```

4. Log into the test pod:

```
kubectl exec -it test -- bash
```

5. Create a data directory, and switch to it:

```
mkdir /data && cd /data
```

6. Download the HiBench test suite:

```
git clone https://github.com/intel-hadoop/HiBench.git
```

7. In a separate terminal, download Hadoop 2.7.3 and Spark 2.4.8 for Hadoop 2.7:

```
wget https://archive.apache.org/dist/hadoop/common/hadoop-2.7.3/hadoop-2.7.3.tar.gz
wget https://downloads.apache.org/spark/spark-2.4.8/spark-2.4.8-bin-hadoop2.7.tgz
```

8. Copy the files you downloaded in step 6 to the test pod:

```
kubectl cp hadoop-2.7.3.tar.gz test:/data
kubectl cp spark-2.4.8-bin-hadoop2.7.tgz test:/data
```

9. In the test pod terminal, untar the files you copied in step 8:

```
tar -xzf hadoop-2.7.3.tar.gz
tar -xzf spark-2.4.8-bin-hadoop2.7.tgz
```

10. Install nano:

```
apt-get install nano
```

11. Edit /data/hadoop-2.7.3/etc/hadoop/hadoop-env.sh to match what's in the Scripts we used in our testing section.
12. Create a soft link for Spark:

```
ln-s /data/spark-2.4.8-bin-hadoop2.7 /opt/spark
```

13. Change to the HiBench directory:

```
cd /data/HiBench
```

14. Edit the HiBench configuration file at conf/hibench.conf to match what's in the Scripts we used in our testing section.
15. Edit the Hadoop configuration file at conf/hadoop.conf to match what's in the Scripts we used in our testing section.
16. Edit the Spark configuration file at conf/spark.conf to match what's in the Scripts we used in our testing section.
17. Edit the k-means configuration file at conf/workloads/ml/kmeans.conf and set the following:

```
hibench.kmeans.bigdata.k 1000
```

18. Build HiBench with Maven:

```
mvn -Dspark=2.4 -Dscala=2.11 clean package
```

Loading the k-means dataset

1. Navigate to the HiBench directory:

```
cd /data/HiBench
```

2. Prepare the k-means dataset:

```
bin/workloads/ml/kmeans/prepare/prepare.sh
```

Running the k-means test

We used a script in the Scripts we used in our testing section to run the test and collect the data.

1. Run the k-means test:

```
./run_spark_tanzu.sh <run#> <date>
```

Setting up AWS EKS

Creating the run-harness VM

1. Click Instances, and click Launch instances.
2. Type Ubuntu into the search bar. On the Ubuntu 18.04 AMI, click Select.
3. For the instance type, select t2.medium, and click Next: Configure Instance Details.
4. Click Next: Add Storage.
5. Click Next: Add Tags.
6. Click Next: Configure Security Group.
7. Configure rules to open the following ports:
   - 80
   - 22
8. Click Review and Launch.
9. Click Launch.

Configuring the run-harness VM

1. Log into the run-harness VM using the SSH command in the Connect tab for the VM.
2. Install the latest updates on the VM:

```
sudo apt update
sudo apt-get upgrade -y sudo reboot
```

3. Install prerequisites:

```
sudo apt-get unzip nmon wget git
```

4. Update the instance:

```
sudo apt-get upgrade -y
sudo apt-get update -y
sudo reboot
```

5. Install the Amazon AWS CLI v2:

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip sudo /aws/install aws configure
```

6. Enter the username, password, and AWS region to configure the AWS CLI credentials.
7. Install kubectl:

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.18.9/2020-11-02/bin/
linux/amd64/kubectl
chmod +x /kubectl
```

Creating an EKS cluster

1. Create an AWS VPC:

```
aws cloudformation create-stack --stack-name eks-stack --template-url https://s3.us-west-2.
amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml
```

2. Create a file called cluster-role-trust-policy.json, and edit it to the following:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow", "Principal": {
      "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. Create the role:

```
aws iam create-role --role-name EKSClusterRole --assume-role-policy-document file://"cluster-role-
trust-policy.json"
```

4. Attach the policy to the role:

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy --role-
name EKSClusterRole
```

5. Create a key pair:

```
aws ec2 create-key-pair --region us-east-1 --key-name eks-public-key
```

6. Create the EKS cluster:

```
eksctl create cluster --name ekscluster --ssh-access=true --ssh-public-key=eks-public-
key --region=us-east-1 --zones=us-east-1a,us-east-1b,us-east-1c,us-east-1d,us-east-1f
--without-nodegroup
```

7. Create a file called nodegroups.yaml, and edit it to the following, changing the SUT instance type as needed:

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: ekscluster
  region: us-east-1

nodeGroups:
  - name: worker-nodes
    labels: { role: worker }
    instanceType: c5.12xlarge
    desiredCapacity: 16
    ssh:
      publicKeyName: eks-public-key
    volumeSize: 500
```

8. Create the nodegroups:

```
eksctl create nodegroup --config-file=nodegroup.yaml
```

9. Download the ebs csi driver IAM policy document from GitHub:

```
curl -o example-iam-policy.json https://raw.githubusercontent.com/kubernetes-sigs/aws-ebs-csi-driver/master/docs/example-iam-policy.json
```

10. Create the EBS CSI driver policy:

```
aws iam create-policy \
    --policy-name AmazonEKS_EBS_CSI_Driver_Policy \
    --policy-document file://example-iam-policy.json
```

11. Create an IAM role, and attach the IAM policy to it:

```
eksctl create iamserviceaccount --name ebs-csi-controller-sa --namespace kube-system --cluster
ekscluster --attach-policy-arn arn:aws:iam::<ACCOUNT ID>:policy/AmazonEKS_EBS_CSI_Driver_Policy
--approve --override-existing-serviceaccounts
```

12. Download Helm from https://github.com/helm/helm/releases, and run the following command:

```
   cd ~/Downloads/
   tar -zxvf helm-v3.6.3-linux-amd64.tar.gz
   sudo cp ./linux-amd64/helm /usr/local/bin
```

13. Add the aws-ebs-csi-driver Helm repository:

```
helm repo add aws-ebs-csi-driver https://kubernetes-sigs.github.io/aws-ebs-csi-driver
helm repo update
```

14. Install the driver using the Helm chart:

```
helm upgrade -install aws-ebs-csi-driver aws-ebs-csi-driver/aws-ebs-csi-driver --namespace kube-system --set image.repository=602401143452.dkr.ecr.us-east-1.amazonaws.com/eks/aws-ebs-csi-driver --set controller.serviceAccount.create=false --set controller.serviceAccount.name=ebs-csi-controller-sa
```

15. Create a file called gp3.yaml, and edit it to the following:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gp3
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  csi.storage.k8s.io/fstype: ext4
  type: gp3
  throughput: "1000"
  iops: "16000"
  encrypted: "false"
reclaimPolicy: Delete
```

16. Create the gp3 storage class:

```
kubectl apply -f gp3.yaml
```

17. Set the gp3 storageclass as the default:

```
kubectl patch storageclass gp3 -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-
default-class":"true"}}}'
kubectl patch storageclass gp2 -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-
default-class":"false"}}}'
```

Deploying Hadoop Pods in AWS

1. From the Ubuntu jumper VM, add the Hadoop repository to Helm:

```
helm repo add stable https://charts.helm.sh/stable
helm repo update
```

2. Create a rolebinding configuration file named hadoop-rbac.yaml for Hadoop, and fill it with the following:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: spark
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: spark-role
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["*"]
- apiGroups: [""]
  resources: ["services"]
  verbs: ["*"]
- apiGroups: [""]
  resources: ["configMap"]
  verbs: ["*"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: spark-role-binding
  namespace: default
subjects:
- kind: ServiceAccount
  name: spark
  namespace: default
roleRef:
  kind: Role
  name: spark-role
  apiGroup: rbac.authorization.k8s.io
```

3. Apply hadoop-rbac.yaml:

```
kubectl apply -f hadoop-rbac.yaml
```

4. Edit hadoop-values-aws.yaml to match what's in the Scripts we used in our testing section.
5. Deploy Hadoop containers with Helm:

```
helm install hadoop -f hadoop-values-aws.yaml stable/hadoop
```

Configuring the test Pod

1. Copy one of the ycsb.yaml files to test.yaml and fill it with the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
  - name: test
    image: 0track/ycsb:latest
    ports:
      - containerPort: 80
```

2. Label the pod as a testing pod:

```
kubectl label test is-test=true
```

3. Launch the test container:

```
kubectl apply -f test.yaml
```

4. Log into the test pod:

```
kubectl exec -it test -- bash
```

5. Create a data directory, and switch to it.

```
mkdir /data && cd /data
```

6. Download the HiBench test suite:

```
git clone https://github.com/intel-hadoop/HiBench.git
```

7. In a separate terminal, download Hadoop 2.7.3 and Spark 2.4.8 for Hadoop 2.7.

```
wget https://archive.apache.org/dist/hadoop/common/hadoop-2.7.3/hadoop-2.7.3.tar.gz
wget https://downloads.apache.org/spark/spark-2.4.8/spark-2.4.8-bin-hadoop2.7.tgz
```

8. Copy the files downloaded in step 7 to the test pod:

```
kubectl cp hadoop-2.7.3.tar.gz test:/data
kubectl cp spark-2.4.8-bin-hadoop2.7.tgz test:/data
```

9. In the test pod terminal, untar the files copied in step 8:

```
tar -xzf hadoop-2.7.3.tar.gz
tar -xzf spark-2.4.8-bin-hadoop2.7.tgz
```

10. Install nano:

```
apt-get install nano
```

11. Edit /data/hadoop-2.7.3/etc/hadoop/hadoop-env.sh to match what's in the Scripts we used in our testing section.
12. Create a soft link for Spark:

```
ln-s /data/spark-2.4.8-bin-hadoop2.7 /opt/spark
```

13. Change to the HiBench directory:

```
cd /data/HiBench
```

14. Edit the HiBench configuration file at conf/hibench.conf to match what's in the Scripts we used in our testing section.
15. Edit the Hadoop configuration file at conf/hadoop.conf to match what's in the Scripts we used in our testing section.
16. Edit the Spark configuration file at conf/spark.conf to match what's in the Scripts we used in our testing section.
17. Edit the k-means configuration file at conf/workloads/ml/kmeans.conf and set the following:

```
hibench.kmeans.bigdata.k 1000
```

18. Build HiBench with Maven:

```
mvn -Dspark=2.4 -Dscala=2.11 clean package
```

Loading the k-means dataset

1. Navigate to the HiBench directory:

```
cd /data/HiBench
```

2. Prepare the k-means dataset:

```
bin/workloads/ml/kmeans/prepare/prepare.sh
```

Running the k-means test

We used a script in the Scripts we used in our testing section to run the test and collect the data.

1. Run the k-means test:

```
./run_spark_aws.sh <run#> <date>
```

2. We ran this script three times and reported the median run.

Deploying MongoDB Pods in AWS

1. Log into the test harness VM.
2. Modify the Helm chart aws-mongodb-values.yaml to match what's in the Scripts we used in our testing section.
3. Set up Rolebinding before deploying Mongodb pods by creating a file called mongo-rbac-aws.yaml with the following:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rolebinding-default-privileged-sa-ns_default
roleRef:
  kind: ClusterRole
  name: psp:vmware-system-privileged
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:serviceaccounts
```

4. Apply rolebinding for MongoDB:

```
kubectl apply -f mongo-rbac-aws.yaml
```

5. Deploy a new Bitnami/Mongodb-sharded cluster:

```
helm install mongodb -f aws-mongodb-values.yaml bitnami/mongodb-sharded
```

6. After the deployment, wait until all the pods are in running state and ready:

```
kubectl get pods
```

Loading and running the YCSB workload

1. Log into the Tanzu CLI VM.
2. Get the Pod IP of one of the three Mongos Pods:

```
Kubectl describe pod mongodb-mongodb-sharded-mongos-6f4bbbb89b-cwkr2
```

3. Log into the mongos Pod:

```
kubectl exec -it mongodb-mongodb-sharded-mongos-6f4bbbb89b-cwkr2 -- bash
```

4. Log into mongo shell:

```
mongo --username root --password <Password>
```

5. Create the YCSB database:

```
use ycsb
db.dropDatabase()
db.createCollection("ycsb")
```

6. Create a new user for YCSB database:

```
use ycsb
db.createUser({user: "ptuser", pwd: "<Password>", roles: [{role: "readWrite", db: "ycsb"}]})
```

7. Verify you have successfully created the user:

```
db.users()
```

8. Enable sharding on YCSB database:

```
use ycsb
sh.enableSharding("ycsb")
sh.shardCollection("ycsb.usertable", { _id: 1 }, true )
```

9. Log into one of the YCSB Pods, and load the YCSB database:

```
kubectl exec -it ycsb1 -- bash
cd /root/ycsb-0.17.0
./bin/ycsb load mongodb -s -P workloads/workloadc -threads 64 -p mongodb.url="mongodb://
ptuser:<password>@172.16.9.9/ycsb" -p recordcount=500000000
     (172.16.9.9 is the POD IP of one of the mongos servers)
```

10. Create the YCSB containers and start the YCSB test run from all 16 Pods using the scripts in the Scripts we used in our testing section. Each YCSB pod runs against one of the 16 mongos servers:

```
./setup_ycsb.sh
./run_aws_mongo.sh
```

11. We ran the test three times and recorded the median run.

Deploying VMware NSX Advanced Load Balancer for application "cloud-bursting"

Deploying a NSX ALB controller on-premises

1. Download NSX ALB version 21.1.3-9051 from the VMware product download page.
2. Log into vCenter.
3. Right-click the legacy cluster→Deploy OVF Template…
4. Select Local file, navigate to the OVA file you downloaded in step 1, and click Next.
5. Name the controller, select a folder for the location, and click Next.
6. Select the cluster for the compute resource, and click Next.
7. Review the details, and click Next.
8. Select the vSAN datastore you created earlier, and click Next.
9. Select the PG you created during the vSAN networking setup for NSX ALB, and click Next.
10. Enter an IP Address, Subnet, and Default Gateway for the controller, and click Next.
11. Click Finish.
12. Once the template is deployed, power on the VM.
13. Wait for the controller to initialize, and log into the IP address you gave it during deployment.
14. Create a password, and confirm that password in the following field. Optionally, enter an email address for password recovery, and click Next.
15. Create a system passphrase for the controller backup, confirm that passphrase in the following field, and click Next.
16. Enter an IP address for the DNS Resolver you are using for your environment, a DNS Search Domain, and click Next.
17. Optionally, configure an Email/SMTP server, and click Next.
18. Leave the multi-tenant settings as defaults, and click Save.
19. Navigate to Infrastructure→Clouds.
20. Click on the cog next to the pencil for the default-cloud, and select VMware vCenter for the cloud type.
21. Click Yes, Continue.
22. Fill in the Username, vCenter Address, and Password fields with the appropriate values.
23. In the IPAM Profile drop-down, select Create IPAM/DNS Profile.
24. Give the IPAM profile a name, and click Save.
25. In the DNS Profile drop-down, select Create IPAM/DNS Profile.
26. Give the DNS profile a name, and click Save.

27. Click Next.
28. Select the Datacenter, and click Next.
29. Select the management network, and give it a static range of IP addresses.
30. Click Save.

Deploying a NSX ALB controller in AWS

1. Log into aws.amazon.com.
2. Navigate to the AWS marketplace, and search for AVI Vantage Platform.
3. Subscribe to the software.
4. Navigate to your subscriptions, and, under the Avi software, click Launch new instance.
5. To go to the full AWS Marketplace website, click the link to select a different software version.
6. Select version 21.1.3-9051, and click Continue to Launch.
7. Select an EC2 Instance type, VPC Settings, Subnet Settings, Security Group Settings, and Key Pair Settings, and click Launch.
8. Navigate to the EC2 Instances page, and confirm the deployment is running.
9. Wait a couple of minutes for the controller to initialize, and navigate to the public IP address assigned to the controller VM.
10. Run the following command to complete the admin user account setup:

```
ssh -i <KEY PAIR>.pem admin@<IP ADDRESS> "sudo /opt/avi/scripts/initialize_admin_user.py
--password <ADMIN PASSWORD>"
```

11. Create a system passphrase for the controller backup, confirm that passphrase in the following field, and click Next.
12. Enter an IP address for the DNS Resolver you are using for your environment, a DNS Search Domain, and click Next.
13. Optionally, configure an Email/SMTP server, and click Next.
14. Leave the multi-tenant settings as defaults, and click Save.
15. Navigate to Infrastructure→Clouds.
16. Click on the cog next to Default-Cloud to convert the cloud type to AWS.
17. Select Amazon Web Services, and click Yes, Continue.
18. Select the appropriate AWS Region.
19. Fill in the Access Key ID and Secret Access Key fields with the appropriate credentials, and click Next.
20. Select the appropriate VPC, Availability Zone, and Service Engine Management Network, and click Save.

Creating the backend web servers

We used the following steps to configure the four backend target web servers, two each in VMware and AWS, and used Nginx to host a simple website.

Creating the VMs on-premises

1. Log into vCenter.
2. Right-click on the cluster→New Virtual Machine…
3. Select Create a new virtual machine, and click Next.
4. Name the virtual machine, select the folder location, and click Next.
5. Select the cluster for the VMs, and click Next.
6. Select the vSAN datastore, and click Next.
7. Set the compatibility level to ESXi 7.0 U2 and later, and click Next.
8. Select Linux and Red Hat Enterprise Linux 8 (64-bit) from the drop down menus, and click Next.
9. Customize the hardware to your specifications, and click Next. We used the following configuration:
   - 4 CPUs
   - 16 GB RAM
   - 100 GB Hard disk
10. Review the configuration, and click Finish.

Installing RHEL 8.5 on-premises

1. Boot the VM to the RHEL 8.5 installation media.
2. Choose your language, and click Continue.
3. Click Time & Date, and click the gear in the top-right corner.
4. Uncheck the default ntp file, add your own, and click OK.
5. Click Done.
6. Click Software Selection, and click Minimal Install.
7. Click Done.
8. Click Installation Destination, and click 100GB VMware Virtual disk.
9. Check I will configure partitioning, and click Done.
10. Choose Click here to create them automatically.
11. Double check the partitions are accurate, and the swap file is sized appropriately for your testing. We used the defaults for this test. If necessary, remove the /home partition, and resize the root partition to use the entire space available.
12. Click Done.
13. Click Accept Changes on the pop-up Summary of Changes window.
14. Choose KDUMP, and uncheck Enable KDUMP.
15. Click Done.
16. Choose Network & Host Name
17. Click the switch to turn the Ethernet port to ON. If you do not have DHCP, click Configure to set an IP.
18. Set a host name, and click Apply.
19. Click Done.
20. Click Begin Installation.
21. When the package download begins, click Root Password, and set the password for the Root user.
22. When the installation is finished, click Reboot.

Deploying a RHEL 8.5 instance in AWS

1. Log into the aws.amazon.com console.
2. Navigate to the Instances tab.
3. Click Launch Instance.
4. Search for RHEL 8.5.
5. In the Community AMIs section, select the RHEL-8.5_HVM-20220127-arm64-3-Hourly2-GP2 ami provided by Red Hat, Inc.
6. Choose your instance size. We selected t2.micro.
7. Click Next: Configure Instance Details.
8. Configure the appropriate Network, and leave the rest as default.
9. Click Next: Add Storage.
10. Click Next: Add Tags.
11. Add the appropriate tags, and click Next: Configure Security Group.
12. Leave the default to create a new security group, or select an existing security group.
13. Click Review and Launch.
14. Click Launch.
15. In the pop-up window, select an existing key pair, check the acknowledgement box, and click Launch Instances.

Updating and configuring the operating system

1. Log into the RHEL VM as the root user.
2. Disable the system firewall by running the following commands:

```
systemctl stop firewalld
systemctl disable firewalld
```

3. Run the following command to set SELinux to disabled:

```
sed -i 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config
```

4. Update the server:

```
yum update -y
```

5. Install Nginx:

```
yum install nginx -y
```

6. Reboot the server.
7. Log into the RHEL VM.
8. Edit /etc/nginx/nginx.conf to match the nginx.conf file in the Scripts we used in our testing section.
9. Back up index.html:

```
cp /usr/share/nginx/html/index.html /usr/share/nginx/html/index.html.orig
```

10. Use a text editor to edit /usr/share/nginx/html/index.html with the following:

```
<head>test</head>
```

11. Restart Nginx:

```
systemctl restart nginx
```

Creating the backend web server pool

1. Log into the NSX controllers.
2. Click the Pools tab on the left-hand side.
3. Click Create Pool.
4. Give the pool a name, and click Next.
5. Click Select Servers by Network.
6. Select the network that the web servers reside in, check the web servers, and click Add Servers.
7. Click Next twice.
8. Click Save.

Creating the Virtual Service VIP for the backend web servers on-premises

1. Log into the NSX controller.
2. Click the VS VIPs tab on the left-hand side.
3. Click Create.
4. Under VIPs (0), click Add.
5. Under Private IP, click Auto-Allocate.
6. Select the VIP Address Allocation Network and the IPv4 Subnet, and click Save.
7. Click Save.

Creating the Virtual Service VIP for the backend web servers in AWS

1. Log into the aws.amazon.com console.
2. Under Network & Security, select Elastic Ips.
3. Click Allocate Elastic IP address.
4. Click Allocate.
5. Log into the NSX controller.
6. Click the VS VIPs tab on the left-hand side.
7. Click Create.
8. Under VIPs (0), click Add.
9. Select the Availability Zone.
10. Under Private IP, select Auto-Allocate.

11. Select the VIP Address Allocation Network with the backend web servers.
12. Select the IPv4 Subnet.
13. Under Public IP, select Static.
14. Enter the elastic IP address you created in steps 1-4 in the IPv4 Address field, and click Save.
15. Click Save.

Creating the web Virtual Services

1. Log into the NSX controllers.
2. Click the Virtual Services tab on the left-hand side.
3. Click Create Virtual Service→Advanced Setup.
4. Name the Virtual Service.
5. Select the VS VIP created in previous steps.
6. Select the Pool created in the previous steps.
7. Click Next three times.
8. Click Save.

Creating the Service Engine Group for the on-premises DNS virtual service

1. Log into the on-premises NSX controller.
2. Navigate to Infrastructure→Service Engine Group.
3. Click Create.
4. Give the SE group a name and leave the rest default.
5. Click Save.

Creating the virtual service VIP for the on-premises DNS virtual service

1. Log into the NSX controller.
2. Click the VS VIPs tab on the left-hand side.
3. Click Create.
4. Under VIPs (0), click Add.
5. Under Private IP, click Auto-Allocate.
6. Select the VIP Address Allocation Network and the IPv4 Subnet, and click Save.
7. Click Save.

Creating the DNS virtual service on-premises

1. Log into the on-premises NSX controller.
2. Navigate to Infrastructure→Virtual Services.
3. Click Create Virtual Service→Advanced Setup.
4. Assign a name, and select the virtual service VIP that we created in the previous section.
5. In the Application Profile drop-down, select System-DNS and make sure the field under TCP/UDP Profile is filled with System-UDP-Per-Pkt.
6. In the Service Port section, click Switch to Advanced.
7. Click Add Port.
8. Set Port Min and Port Max to 53, and check the box next to Override TCP/UDP. In the drop-down field that appears, select System-TCP-Proxy.
9. Click Next four times.
10. Click Save.

Enabling GSLB and create the on-premises active/leader site

1. Log into the on-premises NSX controller.
2. Navigate to Infrastructure→GSLB→Site Configuration.
3. Click the pencil to turn GSLB on, and configure the first site.
4. Give the on-premises site a name.
5. Enter the username and password for the on-premises controller.
6. Enter the IP address and port.
7. Enter a subdomain, click Save, and Set DNS Virtual Services.
8. In the DNS Virtual Service drop-down, select the on-premises DNS virtual service, and the subdomain associated with it.
9. Click Save.

Creating a passive AWS site in GSLB configuration

1. On the GSLB site configuration page, click Add New Site.
2. Give the site a name.
3. Enter the username and password for the AWS controller.
4. Enter the IP address and port.
5. Click Save.

Creating the GSLB service for the web application

1. In the on-premises controller, navigate to Applications→GSLB Services.
2. Click Create→Advanced Setup.
3. Give the service a name.
4. Fill in Application Name field with the application name of the FQDN for the service, and verify the subdomain.
5. In the GSLB pools section, click Add Pool.
6. Give the on-premises pool a name.
7. In the Pool Member section, check Radio (next to Virtual Service).
8. Under the Site Cluster Controller drop-down, select the on-premises controller.
9. Under the Virtual Service drop-down, select the web virtual service you created earlier.
10. Click Done.
11. Repeat steps 5 through 10 to add the AWS pool, selecting the AWS controller in step 8.
12. Click Save.

Preparing the NSX ALB for the cloud-bursting demo

Uploading the GSLB service pool enable script

We used the enable_gslbpoolmember.py script in the Scripts we used in our testing section to enable the AWS pool in the GSLB service.

1. In the on-premises controller, navigate to Templates→ControlScripts.
2. Click Create.
3. Give the control script a name.
4. Import the python script from your local machine or paste it into the control script field.
5. Click Save.

Creating an alert action

1. In the on-premises controller, navigate to Operations→Alert Actions.
2. Click Create.
3. Enter a name for the alert action.
4. In the control script drop-down, select the control script uploaded in the previous steps.
5. Click Save.

Creating an alert config

1. In the on-premises controller, navigate to Operations→Alert Actions.
2. Click Create.
3. Name the alert config.
4. Under conditions, set the Throttle Alert to 0 seconds.
5. Under Source, check Radio (next to Metrics).
6. Under Object, select Virtual Service.
7. Under the Instance drop-down, select the on-premises web virtual service.
8. Under the Metric Occurs drop-down, select l4_client.avg_bandwidth.
9. Under the Comparator drop-down, select greater than or equal to.
10. In the Duration field, enter `10`.
11. In the Value field, enter `20971520`.
12. Under Actions, select the alert action you created in the previous steps from the Alert Action drop-down.
13. Click Save.

Running the failover demo

We used a clone of one of the web servers to act as our client for the demo.

1. Log into the RHEL 8.5 VM.
2. Download the Apache benchmarking tool:

```
yum install http-tools
```

3. Run the following command to start sending traffic to the GSLB FQDN:

```
while true; do ab -n 10000 -c 1000 http://webapp.web.mathis.com/; done
```

4. Wait 5 minutes, open a second terminal, and run the command in step 3 to send more traffic.
5. Check in the GUI of both controllers. You should see an alert in the on-prem controller and traffic beginning to flow to the second one in AWS.

Scripts we used in our testing

mongod.conf

There were three types of servers in our MongoDB sharded cluster: Config Servers, Mongos Servers, and Mongod Servers. We used the same "mongod.conf" configuration file script for all three types of servers.

```
# mongod.conf
# for documentation of all options, see:
#   http://docs.mongodb.org/manual/reference/configuration-options/
# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log
# Where and how to store data.
storage:
  dbPath: /var/lib/mongo/data
  journal:
    enabled: true
  engine: "wiredTiger"
#  wiredTiger:
# how the process runs
processManagement:
  fork: true  # fork and run in background
  pidFilePath: /var/run/mongodb/mongod.pid  # location of pidfile
  timeZoneInfo: /usr/share/zoneinfo
# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0  # Enter 0.0.0.0,:: to bind to all IPv4 and IPv6 addresses or, alternatively, use
the net.bindIpAll setting.
#security:
#operationProfiling:
replication:
  replSetName: config-replica-set
sharding:
  clusterRole: configsvr
## Enterprise-Only Options
#auditLog:
#snmp:
```

mongo-ycsb{1-8}.sh

This script is called by run_legacy_mongodb.sh to kickoff all VMs simultaneously.

```
~/ycsb-0.17.0/bin/ycsb run mongodb -s -P ~/ycsb-0.17.0/workloads/workloadc -threads 64 -p mongodb.
url="mongodb://mongos1:27017/ycsb" -p operationcount=30000000 > ~/results/legacy_ycsb1_run1.txt
```

run_legacy_mongodb.sh

```
screen -d -m ~/mongo-ycsb1.sh
ssh ycsb2 screen -d -m ~/mongo-ycsb2.sh
ssh ycsb3 screen -d -m ~/mongo-ycsb3.sh
ssh ycsb4 screen -d -m ~/mongo-ycsb4.sh
ssh ycsb5 screen -d -m ~/mongo-ycsb5.sh
ssh ycsb6 screen -d -m ~/mongo-ycsb6.sh
ssh ycsb7 screen -d -m ~/mongo-ycsb7.sh
ssh ycsb8 screen -d -m ~/mongo-ycsb8.sh
```

tanzu-cluster.yaml

```
apiVersion: run.tanzu.vmware.com/v1alpha1
kind: TanzuKubernetesCluster
metadata:
  name: tanzu-cluster
  namespace: tanzu-ns
spec:
  topology:
    controlPlane:
      count: 3
      class: best-effort-large
      storageClass: tkg-clusters
    workers:
      count: 16
      class: best-effort-12x
      storageClass: tkg-clusters
  distribution:
    version: v1.20
  settings:
    network:
      cni:
        name: antrea
      services:
        cidrBlocks: ["10.96.1.0/24"]        #Cannot overlap with Supervisor Cluster
      pods:
        cidrBlocks: ["172.16.0.0/16"]           #Cannot overlap with Supervisor Cluster
```

tanzu-mongodb-values.yaml

```
## Global Docker image parameters
## Please, note that this will override the image parameters, including dependencies, configured to
use the global value
## Current available global Docker image parameters: imageRegistry, imagePullSecrets and storageClass
##
# global:

## Bitnami MongoDB(R) Sharded image version
## ref: https://hub.docker.com/r/bitnami/mongodb-sharded/tags/
##
image:
  registry: docker.io
  repository: bitnami/mongodb-sharded
  tag: 4.4.6-debian-10-r40
  ## Specify a imagePullPolicy
  ## Defaults to 'Always' if image tag is 'latest', else set to 'IfNotPresent'
  ## ref: http://kubernetes.io/docs/user-guide/images/#pre-pulling-images
  ##
  pullPolicy: IfNotPresent
  ## Set to true if you would like to see extra information on logs
  ##
  debug: false

## MongoDB(R) root password
## If set to null it will be randomly generated
## ref: https://github.com/bitnami/bitnami-docker-mongodb/blob/master/README.md#setting-the-root-
password-on-first-run
##
## mongodbRootPassword: password
##
mongodbRootPassword: <Password>

## Name of a secret containing all the credentials above
## ref:
##
```

```
## existingSecret: name-of-existing-secret
##
existingSecret:

## Mount credentials as files instead of using environment variables
##
usePasswordFile: false

## Number of MongoDB(R) Shards
## ref: https://docs.mongodb.com/manual/core/sharded-cluster-shards/
##
shards: 12

## Shard replica set properties
## ref: https://docs.mongodb.com/manual/replication/index.html
##
shardsvr:
  ## Properties for data nodes (primary and secondary)
  ##
  dataNode:
    ## Number of replicas. A value of replicas=1 is simply a primary node
    ##
    replicas: 3
    ## Configure resource requests and limits
    ## ref: http://kubernetes.io/docs/user-guide/compute-resources/
    ##
    resources: {}
    ## resources:
      # We usually recommend not to specify default resources and to leave this as a conscious
      # choice for the user. This also increases chances charts run on environments with little
      # resources, such as Minikube. If you do want to specify resources, uncomment the following
      # lines, adjust them as necessary, and remove the curly braces after 'resources:'.
      ## limits:
      ##   cpu: 10
      ##   memory: 32Gi
      ## requests:
      ##   cpu: 10
      ##   memory: 32Gi

      ## Pod anti-affinity preset
    ## ref: https://kubernetes.io/docs/concepts/scheduling-eviction/assign-f-node/#inter-pod-affinity-
and-anti-affinity
    ## Allowed values: soft, hard
    ##
    podAntiAffinityPreset: soft

    ## Affinity for pod assignment
    ## ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-
and-anti-affinity
    ## You can set dataNodeLoopId (or any other parameter) by setting the below code block under this
'affinity' section:
    ##
    podManagementPolicy: OrderedReady
    ## updateStrategy for MongoDB(R) Primary, Secondary and Arbiter statefulsets
    ## ref: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/#update-strategies
    ##
    updateStrategy:
      type: RollingUpdate
    ## Deployment pod host aliases
    ## https://kubernetes.io/docs/concepts/services-networking/add-entries-to-pod-etc-hosts-
with-host-aliases/
    ##
    ## Array to add extra mounts (normally used with extraVolumes)
    ##
    extraVolumeMounts: []
    ## Use an alternate scheduler, e.g. "stork".
    ## ref: https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/
```

```
    ##
    schedulerName:
    ## Pod disruption budget
    ##
    pdb:
      enabled: false
      ## Use 0 to disable
      ##
      minAvailable: 0
      ## Use 0 to disable
      ##
      maxUnavailable: 1
    ## K8s Service Account.
    ## ref: https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/
    ##
    serviceAccount:
      ## Specifies whether a ServiceAccount should be created for shardsvr.
      ##
      create: false
      ## The name of the ServiceAccount to use.
      ## If not set and create is true, a name is generated using the XXX.fullname template
      ##
      # name:
    ## MongoDB(R) additional command line flags
    ##
    ## Can be used to specify command line flags, for example:
    ##
    podAntiAffinityPreset: soft

    ## Affinity for pod assignment
    ## ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-
and-anti-affinity
    ## You can set arbiterLoopId (or any other parameter) by setting the below code block under this
'affinity' section:
    ## podManagementPolicy for the statefulset, allows parallel startup of pods
    ## ref: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/#pod-
management-policies
    ##
    podManagementPolicy: OrderedReady
    ## updateStrategy for MongoDB(R) Primary, Secondary and Arbiter statefulsets
    ## ref: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/#update-strategies
    ##
    updateStrategy:
      type: RollingUpdate
    ## Entries for the MongoDB(R) config file. For documentation of all options, see:
    ## ref: http://docs.mongodb.org/manual/reference/configuration-options/
    ##
    config:
    ## Name of a ConfigMap containing a MongoDB(R) config file (cannot be used at the same
time as config)
    ## ref: http://docs.mongodb.org/manual/reference/configuration-options/
    ##
    configCM:
    ## An array to add extra env vars
    ## For example:
    ## extraEnvVars:
    ##  - name: KIBANA_ELASTICSEARCH_URL
    ##    value: test
    ##
    extraEnvVars:
    ## Name of a ConfigMap containing extra env vars
    ##
    extraEnvVarsCM:
    ## Name of a Secret containing extra env vars
    ##
    extraEnvVarsSecret:
    ## Add sidecars to the pod
```

```
  ## For example:
  ## sidecars:
  ##   - name: your-image-name
  ##     image: your-image
  ##     imagePullPolicy: Always
  ##     ports:
  ##       - name: portname
  ##         containerPort: 1234
  ##
  sidecars: []
  ## Add init containers to the pod
  ## For example:
  ## initcontainers:
  ##   - name: your-image-name
  ##     image: your-image
  ##     imagePullPolicy: Always
  ##
  initContainers: []
  ## Additional pod annotations
  ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
  ##
  podAnnotations: {}
  ## Additional pod labels
  ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/
  ##
  podLabels: {}
  ## Array to add extra volumes
  ##
  extraVolumes: []
  ## Array to add extra mounts (normally used with extraVolumes)
  ##
  extraVolumeMounts: []
  ## Use an alternate scheduler, e.g. "stork".
  ## ref: https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/
  ##
  schedulerName:
  ## K8s Service Account.
  ## ref: https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/
  ##
  serviceAccount:
    ## Specifies whether a ServiceAccount should be created for shardsvr arbiter nodes.
    ##
    create: false
    ## The name of the ServiceAccount to use.
    ## If not set and create is true, a name is generated using the XXX.fullname template
    ##
    # name:
## Enable persistence using Persistent Volume Claims
## ref: http://kubernetes.io/docs/user-guide/persistent-volumes/
##
persistence:
  enabled: true
  ## The path the volume will be mounted at, useful when using different
  ## MongoDB(R) images.
  ##
  mountPath: /bitnami/mongodb

  ## The subdirectory of the volume to mount to, useful in dev environments
  ## and one PV for multiple services.
  ##
  subPath: ""

  ## mongodb data Persistent Volume Storage Class
  ## If defined, storageClassName: <storageClass>
  ## If set to "-", storageClassName: "", which disables dynamic provisioning
  ## If undefined (the default) or set to null, no storageClassName spec is
  ##   set, choosing the default provisioner.  (gp2 on AWS, standard on
```

```
    ##   GKE, AWS & OpenStack)
    ##
    # storageClass: "-"
    storageClass : "tkg-clusters"
    accessModes:
      - ReadWriteOnce
    ## PersistentVolumeClaim size
    ##
    size: 600Gi
    ## Additional volume annotations
    ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
    ##
    annotations: {}

## Config Server replica set properties
## ref: https://docs.mongodb.com/manual/core/sharded-cluster-config-servers/
##
configsvr:
  ## Number of replicas. A value of replicas=1 is simply a primary node
  ##
  replicas: 3
  ## Configure resource requests and limits
  ## ref: http://kubernetes.io/docs/user-guide/compute-resources/
  ##
  resources: {}
  ## resources:
      # We usually recommend not to specify default resources and to leave this as a conscious
      # choice for the user. This also increases chances charts run on environments with little
      # resources, such as Minikube. If you do want to specify resources, uncomment the following
      # lines, adjust them as necessary, and remove the curly braces after 'resources:'.
  ##  limits:
  ##     cpu: 8
  ##     memory: 8Gi
  ##  requests:
  ##     cpu: 8
  ##     memory: 8Gi



  ## Deployment pod host aliases
  ## https://kubernetes.io/docs/concepts/services-networking/add-entries-to-pod-etc-hosts-
with-host-aliases/
  ##
  ## Allowed values: soft, hard
  ##
  nodeAffinityPreset:
    ## Node affinity type
    ## Allowed values: soft, hard
    ##
    type: ""
    ## Node label key to match
    ## E.g.
    ## key: "kubernetes.io/e2e-az-name"
    ##
    key: ""
    ## Node label values to match
    ## E.g.
    ## values:
    ##   - e2e-az1
    ##   - e2e-az2
    ##
    values: []
  ## Affinity for pod assignment
  ## ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity
  ## Note: configsvr.podAffinityPreset, configsvr.podAntiAffinityPreset, and configsvr.nodeAffinityPreset
will be ignored when it's set
  ##
```

```
  affinity: {}
  ## Node labels for pod assignment
  ## ref: https://kubernetes.io/docs/user-guide/node-selection/
  ##
  nodeSelector: {}
  ## Tolerations for pod assignment
  ## ref: https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/
  ##
  tolerations: []
  ## podManagementPolicy for the statefulset, allows parallel startup of pods
  ## ref: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/#pod-
management-policies
  ##
  podManagementPolicy: OrderedReady
  ## updateStrategy for MongoDB(R) Primary, Secondary and Arbiter statefulsets
  ## ref: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/#update-strategies
  ##
  updateStrategy:
    type: RollingUpdate
  ## Entries for the MongoDB(R) config file. For documentation of all options, see:
  ## ref: http://docs.mongodb.org/manual/reference/configuration-options/
  ##
  config:
  ## Name of a ConfigMap containing a MongoDB(R) config file (cannot be used at the same time as config)
  ## ref: http://docs.mongodb.org/manual/reference/configuration-options/
  ##
  configCM:
  ## An array to add extra env vars
  ## For example:
  ## extraEnvVars:
  ##  - name: KIBANA_ELASTICSEARCH_URL
  ##    value: test
  ##
  extraEnvVars:
  ## Name of a ConfigMap containing extra env vars
  ##
  extraEnvVarsCM:
  ## Name of a Secret containing extra env vars
  ##
  extraEnvVarsSecret:
  ## Add sidecars to the pod
  ## For example:
  ## sidecars:
  ##   - name: your-image-name
  ##     image: your-image
  ##     imagePullPolicy: Always
  ##     ports:
  ##       - name: portname
  ##         containerPort: 1234
  ##
  sidecars: []
  ## Add init containers to the pod
  ## For example:
  ## initcontainers:
  ##   - name: your-image-name
  ##     image: your-image
  ##     imagePullPolicy: Always
  ##
  initContainers: []
  ## Additional pod annotations
  ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
  ##
  podAnnotations: {}
  ## Additional pod labels
  ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/
  ##
  podLabels: {}
```

```
  ## Array to add extra volumes
  ##
  extraVolumes: []
  ## Array to add extra mounts (normally used with extraVolumes)
  ##
  extraVolumeMounts: []
  ## Use an alternate scheduler, e.g. "stork".
  ## ref: https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/
  ##
  schedulerName:
  ## Pod disruption budget
  ##
  pdb:
    enabled: false
    ## Use 0 to disable
    ##
    minAvailable: 0
    ## Use 0 to disable
    ##
    maxUnavailable: 1
  ## Enable persistence using Persistent Volume Claims
  ## ref: http://kubernetes.io/docs/user-guide/persistent-volumes/
  ##
  persistence:
    enabled: true
    ## The path the volume will be mounted at, useful when using different
    ## MongoDB(R) images.
    ##
    mountPath: /bitnami/mongodb

    ## The subdirectory of the volume to mount to, useful in dev environments
    ## and one PV for multiple services.
    ##
    subPath: ""

    ## mongodb data Persistent Volume Storage Class
    ## If defined, storageClassName: <storageClass>
    ## If set to "-", storageClassName: "", which disables dynamic provisioning
    ## If undefined (the default) or set to null, no storageClassName spec is
    ##   set, choosing the default provisioner.  (gp2 on AWS, standard on
    ##   GKE, AWS & OpenStack)
    ##
    # storageClass: "-"

    storageClass: "tkg-clusters"

    accessModes:
      - ReadWriteOnce
    ## PersistentVolumeClaim size
    ##
    size: 200Gi
    ## Additional volume annotations
    ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
    ##
    annotations: {}

## Mongos properties
## ref: https://docs.mongodb.com/manual/reference/program/mongos/#bin.mongos
##
mongos:
  ## Number of replicas
  ##
  replicas: 16
  ## Configure resource requests and limits
  ## ref: http://kubernetes.io/docs/user-guide/compute-resources/
  ##
  resources: {}
```

```
  ## resources:
      # We usually recommend not to specify default resources and to leave this as a conscious
      # choice for the user. This also increases chances charts run on environments with little
      # resources, such as Minikube. If you do want to specify resources, uncomment the following
      # lines, adjust them as necessary, and remove the curly braces after 'resources:'.
  ##  limits:
  ##     cpu: 8
  ##     memory: 32Gi
  ##  requests:
  ##     cpu: 8
  ##     memory: 32Gi



  ## Deployment pod host aliases
  ## https://kubernetes.io/docs/concepts/services-networking/add-entries-to-pod-etc-hosts-
with-host-aliases/
  ##
  ## Allowed values: soft, hard
  ##
  podAntiAffinityPreset: soft
  ## Node affinity preset
  ## ref: https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#node-affinity
  ## Allowed values: soft, hard
  ##
  nodeAffinityPreset:
    ## Node affinity type
    ## Allowed values: soft, hard
    ##
    type: ""
    ## Node label key to match
    ## E.g.
    ## key: "kubernetes.io/e2e-az-name"
    ##
    key: ""
    ## Node label values to match
    ## E.g.
    ## values:
    ##   - e2e-az1
    ##   - e2e-az2
    ##
    values: []
  ## Affinity for pod assignment
  ## ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity
  ## Note: mongos.podAffinityPreset, mongos.podAntiAffinityPreset, and mongos.nodeAffinityPreset will be
ignored when it's set
  ##
  updateStrategy:
    type: RollingUpdate

  useStatefulSet: false
  ## When using a statefulset, you can enable one service per replica
  ## This is useful when exposing the mongos through load balancers to make sure clients
  ## connect to the same mongos and therefore can follow their cursors
  ##
  servicePerReplica:
    enabled: false
    ## Additional service annotations (evaluate as a template)
    ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
    ##
    annotations: {}
    ## Service type
    ## ref: https://kubernetes.io/docs/concepts/services-networking/service/#publishing-
services-service-types
    ##
    type: ClusterIP
    ## External traffic policy
```

```
    ## ref: https://kubernetes.io/docs/concepts/services-networking/service/#publishing-
services-service-types
    ##
    externalTrafficPolicy: Cluster
    ## MongoDB(R) Service port and Container Port
    ##
    port: 27017

  ## Pod disruption budget
  ##
  pdb:
    enabled: false
    ## Use 0 to disable
    ##
    minAvailable: 0
    ## Use 0 to disable
    ##
    maxUnavailable: 1
  ## K8s Service Account.
  ## ref: https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/
  ##
  serviceAccount:
    ## Specifies whether a ServiceAccount should be created for mongos.
    ##
    create: false


## Properties for all of the pods in the cluster (shards, config servers and mongos)
##
common:
  ## Use hostnames instead of IP addresses
  ##
  useHostnames: true
  ## Whether enable/disable IPv6 on MongoDB(R)
  ## ref: https://github.com/bitnami/bitnami-docker-mongodb/blob/master/README.
md#enabling/disabling-ipv6
  ##
  mongodbEnableIPv6: false
  ## Whether enable/disable DirectoryPerDB on MongoDB(R)
  ## ref: https://github.com/bitnami/bitnami-docker-mongodb/blob/master/README.md#enabling/
disabling-directoryperdb
  ##
  mongodbDirectoryPerDB: false
  ## System log verbosity level
  ## ref: https://docs.mongodb.com/manual/reference/program/mongo/#cmdoption-mongo-ipv6
  ##
  mongodbSystemLogVerbosity: 0
  ## MongoDB(R) System Log configuration
  ## ref: https://github.com/bitnami/bitnami-docker-mongodb#configuring-system-log-verbosity-level
  ##
  mongodbDisableSystemLog: false
  ## Maximum peer node waiting timeout (in seconds)
  ##
  mongodbMaxWaitTimeout: 120

## Init containers parameters:
## volumePermissions: Change the owner and group of the persistent volume mountpoint to
runAsUser:fsGroup values from the securityContext section.
##
volumePermissions:
  enabled: false
  image:
    registry: docker.io
    repository: bitnami/bitnami-shell
    tag: 10-debian-10-r126
    pullPolicy: Always
    ## Optionally specify an array of imagePullSecrets.
```

```
    ## Secrets must be manually created in the namespace.
    ## ref: https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/
    ##
    # pullSecrets:
    #   - myRegistryKeySecretName
  resources: {}

## Pod Security Context
## ref: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/
##
securityContext:
  enabled: true
  fsGroup: 1001
  runAsUser: 1001
  runAsNonRoot: true

## Kubernetes Cluster Domain
## ref: https://kubernetes.io/docs/tasks/administer-cluster/dns-custom-nameservers/#introduction
##
clusterDomain: cluster.local

## Kubernetes service type
## ref: https://kubernetes.io/docs/concepts/services-networking/service/
##
service:
  ## Specify an explicit service name
  ##
  name:
  ## Additional service annotations (evaluate as a template)
  ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
  ##
  annotations: {}
  ## Service type
  ## ref: https://kubernetes.io/docs/concepts/services-networking/service/#publishing-
services-service-types
  ##
  type: ClusterIP
  ## External traffic policy
  ## ref: https://kubernetes.io/docs/concepts/services-networking/service/#publishing-
services-service-types
  ##
  externalTrafficPolicy: Cluster
  ## MongoDB(R) Service port and Container Port
  ##
  port: 27017

  ## Specify the sessionAffinity setting for the service. Can be "None" or "ClientIP".
  ## If "ClientIP", consecutive client requests will be directed to the same mongos Pod
  ## ref: https://kubernetes.io/docs/concepts/services-networking/service/#virtual-ips-
and-service-proxies
  ##
  sessionAffinity: None

## Configure extra options for liveness and readiness probes
## This applies to all the MongoDB(R) in the sharded cluster
## ref: https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-
probes/#configure-probes)
##
livenessProbe:
  enabled: true
  initialDelaySeconds: 360
  periodSeconds: 10
  timeoutSeconds: 5
  failureThreshold: 6
  successThreshold: 1
readinessProbe:
  enabled: true
```

```
  initialDelaySeconds: 360
  periodSeconds: 10
  timeoutSeconds: 5
  failureThreshold: 6
  successThreshold: 1

## Prometheus Exporter / Metrics
##
metrics:
  enabled: false

  image:
    registry: docker.io
    repository: bitnami/mongodb-exporter
    tag: 0.11.2-debian-10-r212
    pullPolicy: Always
    ## Optionally specify an array of imagePullSecrets.
  ## Metrics exporter liveness and readiness probes
  ## ref: https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-
probes/#configure-probes)
  ##
  livenessProbe:
    enabled: false
    initialDelaySeconds: 15
    periodSeconds: 5
    timeoutSeconds: 5
    failureThreshold: 3
    successThreshold: 1
  readinessProbe:
    enabled: false
    initialDelaySeconds: 5
    periodSeconds: 5
    timeoutSeconds: 1
    failureThreshold: 3
    successThreshold: 1

  ## Metrics container port
  ##
  containerPort: 9216

  ## Metrics exporter pod Annotation
  ##
  podAnnotations:
    prometheus.io/scrape: "true"
    prometheus.io/port: "{{ .Values.metrics.containerPort }}"
```

aws-mongodb-values.yaml

```
## Global Docker image parameters
## Please, note that this will override the image parameters, including dependencies, configured to
use the global value
## Current available global Docker image parameters: imageRegistry, imagePullSecrets and storageClass
##
# global:

## Bitnami MongoDB(R) Sharded image version
## ref: https://hub.docker.com/r/bitnami/mongodb-sharded/tags/
##
image:
  registry: docker.io
  repository: bitnami/mongodb-sharded
  tag: 4.4.6-debian-10-r40
  ## Specify a imagePullPolicy
  ## Defaults to 'Always' if image tag is 'latest', else set to 'IfNotPresent'
  ## ref: http://kubernetes.io/docs/user-guide/images/#pre-pulling-images
  ##
```

```
  pullPolicy: IfNotPresent
  ## Set to true if you would like to see extra information on logs
  ##
  debug: false

## MongoDB(R) root password
## If set to null it will be randomly generated
## ref: https://github.com/bitnami/bitnami-docker-mongodb/blob/master/README.md#setting-the-root-
password-on-first-run
##
## mongodbRootPassword: password
##
mongodbRootPassword: <Password>

## Name of a secret containing all the credentials above
## ref:
##
## existingSecret: name-of-existing-secret
##
existingSecret:

## Mount credentials as files instead of using environment variables
##
usePasswordFile: false

## Number of MongoDB(R) Shards
## ref: https://docs.mongodb.com/manual/core/sharded-cluster-shards/
##
shards: 12

## Shard replica set properties
## ref: https://docs.mongodb.com/manual/replication/index.html
##
shardsvr:
  ## Properties for data nodes (primary and secondary)
  ##
  dataNode:
    ## Number of replicas. A value of replicas=1 is simply a primary node
    ##
    replicas: 3
    ## Configure resource requests and limits
    ## ref: http://kubernetes.io/docs/user-guide/compute-resources/
    ##
    resources: {}
    ## resources:
      # We usually recommend not to specify default resources and to leave this as a conscious
      # choice for the user. This also increases chances charts run on environments with little
      # resources, such as Minikube. If you do want to specify resources, uncomment the following
      # lines, adjust them as necessary, and remove the curly braces after 'resources:'.
      ## limits:
      ##   cpu: 10
      ##   memory: 32Gi
      ## requests:
      ##   cpu: 10
      ##   memory: 32Gi

      ## Pod anti-affinity preset
    ## ref: https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#inter-pod-
affinity-and-anti-affinity
    ## Allowed values: soft, hard
    ##
    podAntiAffinityPreset: soft

    ## Affinity for pod assignment
    ## ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-
and-anti-affinity
    ## You can set dataNodeLoopId (or any other parameter) by setting the below code block under this
```

```
'affinity' section:
    ##
    podManagementPolicy: OrderedReady
    ## updateStrategy for MongoDB(R) Primary, Secondary and Arbiter statefulsets
    ## ref: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/#update-strategies
    ##
    updateStrategy:
      type: RollingUpdate
    ## Deployment pod host aliases
    ## https://kubernetes.io/docs/concepts/services-networking/add-entries-to-pod-etc-hosts-
with-host-aliases/
    ##
    ## Array to add extra mounts (normally used with extraVolumes)
    ##
    extraVolumeMounts: []
    ## Use an alternate scheduler, e.g. "stork".
    ## ref: https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/
    ##
    schedulerName:
    ## Pod disruption budget
    ##
    pdb:
      enabled: false
      ## Use 0 to disable
      ##
      minAvailable: 0
      ## Use 0 to disable
      ##
      maxUnavailable: 1
    ## K8s Service Account.
    ## ref: https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/
    ##
    serviceAccount:
      ## Specifies whether a ServiceAccount should be created for shardsvr.
      ##
      create: false
      ## The name of the ServiceAccount to use.
      ## If not set and create is true, a name is generated using the XXX.fullname template
      ##
      # name:
    ## MongoDB(R) additional command line flags
    ##
    ## Can be used to specify command line flags, for example:
    ##
    podAntiAffinityPreset: soft

    ## Affinity for pod assignment
    ## ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-
and-anti-affinity
    ## You can set arbiterLoopId (or any other parameter) by setting the below code block under this
'affinity' section:
    ## podManagementPolicy for the statefulset, allows parallel startup of pods
    ## ref: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/#pod-
management-policies
    ##
    podManagementPolicy: OrderedReady
    ## updateStrategy for MongoDB(R) Primary, Secondary and Arbiter statefulsets
    ## ref: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/#update-strategies
    ##
    updateStrategy:
      type: RollingUpdate
    ## Entries for the MongoDB(R) config file. For documentation of all options, see:
    ## ref: http://docs.mongodb.org/manual/reference/configuration-options/
    ##
    config:
    ## Name of a ConfigMap containing a MongoDB(R) config file (cannot be used at the same
time as config)
```

```
  ## ref: http://docs.mongodb.org/manual/reference/configuration-options/
  ##
  configCM:
  ## An array to add extra env vars
  ## For example:
  ## extraEnvVars:
  ##  - name: KIBANA_ELASTICSEARCH_URL
  ##    value: test
  ##
  extraEnvVars:
  ## Name of a ConfigMap containing extra env vars
  ##
  extraEnvVarsCM:
  ## Name of a Secret containing extra env vars
  ##
  extraEnvVarsSecret:
  ## Add sidecars to the pod
  ## For example:
  ## sidecars:
  ##   - name: your-image-name
  ##     image: your-image
  ##     imagePullPolicy: Always
  ##     ports:
  ##       - name: portname
  ##         containerPort: 1234
  ##
  sidecars: []
  ## Add init containers to the pod
  ## For example:
  ## initcontainers:
  ##   - name: your-image-name
  ##     image: your-image
  ##     imagePullPolicy: Always
  ##
  initContainers: []
  ## Additional pod annotations
  ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
  ##
  podAnnotations: {}
  ## Additional pod labels
  ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/
  ##
  podLabels: {}
  ## Array to add extra volumes
  ##
  extraVolumes: []
  ## Array to add extra mounts (normally used with extraVolumes)
  ##
  extraVolumeMounts: []
  ## Use an alternate scheduler, e.g. "stork".
  ## ref: https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/
  ##
  schedulerName:
  ## K8s Service Account.
  ## ref: https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/
  ##
  serviceAccount:
    ## Specifies whether a ServiceAccount should be created for shardsvr arbiter nodes.
    ##
    create: false
    ## The name of the ServiceAccount to use.
    ## If not set and create is true, a name is generated using the XXX.fullname template
    ##
    # name:
## Enable persistence using Persistent Volume Claims
## ref: http://kubernetes.io/docs/user-guide/persistent-volumes/
##
```

```
  persistence:
    enabled: true
    ## The path the volume will be mounted at, useful when using different
    ## MongoDB(R) images.
    ##
    mountPath: /bitnami/mongodb

    ## The subdirectory of the volume to mount to, useful in dev environments
    ## and one PV for multiple services.
    ##
    subPath: ""

    ## mongodb data Persistent Volume Storage Class
    ## If defined, storageClassName: <storageClass>
    ## If set to "-", storageClassName: "", which disables dynamic provisioning
    ## If undefined (the default) or set to null, no storageClassName spec is
    ##   set, choosing the default provisioner.  (gp2 on AWS, standard on
    ##   GKE, AWS & OpenStack)
    ##
    # storageClass: "-"
    storageClass : "gp3"
    accessModes:
      - ReadWriteOnce
    ## PersistentVolumeClaim size
    ##
    size: 600Gi
    ## Additional volume annotations
    ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
    ##
    annotations: {}

## Config Server replica set properties
## ref: https://docs.mongodb.com/manual/core/sharded-cluster-config-servers/
##
configsvr:
  ## Number of replicas. A value of replicas=1 is simply a primary node
  ##
  replicas: 3
  ## Configure resource requests and limits
  ## ref: http://kubernetes.io/docs/user-guide/compute-resources/
  ##
  resources: {}
  ## resources:
      # We usually recommend not to specify default resources and to leave this as a conscious
      # choice for the user. This also increases chances charts run on environments with little
      # resources, such as Minikube. If you do want to specify resources, uncomment the following
      # lines, adjust them as necessary, and remove the curly braces after 'resources:'.
  ##  limits:
  ##     cpu: 8
  ##     memory: 8Gi
  ##  requests:
  ##     cpu: 8
  ##     memory: 8Gi



  ## Deployment pod host aliases
  ## https://kubernetes.io/docs/concepts/services-networking/add-entries-to-pod-etc-hosts-
with-host-aliases/
  ##
  ## Allowed values: soft, hard
  ##
  nodeAffinityPreset:
    ## Node affinity type
    ## Allowed values: soft, hard
    ##
    type: ""
```

```
    ## Node label key to match
    ## E.g.
    ## key: "kubernetes.io/e2e-az-name"
    ##
    key: ""
    ## Node label values to match
    ## E.g.
    ## values:
    ##   - e2e-az1
    ##   - e2e-az2
    ##
    values: []
  ## Affinity for pod assignment
  ## ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity
  ## Note: configsvr.podAffinityPreset, configsvr.podAntiAffinityPreset, and configsvr.nodeAffinityPreset
will be ignored when it's set
  ##
  affinity: {}
  ## Node labels for pod assignment
  ## ref: https://kubernetes.io/docs/user-guide/node-selection/
  ##
  nodeSelector: {}
  ## Tolerations for pod assignment
  ## ref: https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/
  ##
  tolerations: []
  ## podManagementPolicy for the statefulset, allows parallel startup of pods
  ## ref: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/#pod-
management-policies
  ##
  podManagementPolicy: OrderedReady
  ## updateStrategy for MongoDB(R) Primary, Secondary and Arbiter statefulsets
  ## ref: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/#update-strategies
  ##
  updateStrategy:
    type: RollingUpdate
  ## Entries for the MongoDB(R) config file. For documentation of all options, see:
  ## ref: http://docs.mongodb.org/manual/reference/configuration-options/
  ##
  config:
  ## Name of a ConfigMap containing a MongoDB(R) config file (cannot be used at the same time as config)
  ## ref: http://docs.mongodb.org/manual/reference/configuration-options/
  ##
  configCM:
  ## An array to add extra env vars
  ## For example:
  ## extraEnvVars:
  ##  - name: KIBANA_ELASTICSEARCH_URL
  ##    value: test
  ##
  extraEnvVars:
  ## Name of a ConfigMap containing extra env vars
  ##
  extraEnvVarsCM:
  ## Name of a Secret containing extra env vars
  ##
  extraEnvVarsSecret:
  ## Add sidecars to the pod
  ## For example:
  ## sidecars:
  ##   - name: your-image-name
  ##     image: your-image
  ##     imagePullPolicy: Always
  ##     ports:
  ##       - name: portname
  ##         containerPort: 1234
  ##
```

```
sidecars: []
## Add init containers to the pod
## For example:
## initcontainers:
##   - name: your-image-name
##     image: your-image
##     imagePullPolicy: Always
##
initContainers: []
## Additional pod annotations
## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
##
podAnnotations: {}
## Additional pod labels
## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/
##
podLabels: {}
## Array to add extra volumes
##
extraVolumes: []
## Array to add extra mounts (normally used with extraVolumes)
##
extraVolumeMounts: []
## Use an alternate scheduler, e.g. "stork".
## ref: https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/
##
schedulerName:
## Pod disruption budget
##
pdb:
  enabled: false
  ## Use 0 to disable
  ##
  minAvailable: 0
  ## Use 0 to disable
  ##
  maxUnavailable: 1
## Enable persistence using Persistent Volume Claims
## ref: http://kubernetes.io/docs/user-guide/persistent-volumes/
##
persistence:
  enabled: true
  ## The path the volume will be mounted at, useful when using different
  ## MongoDB(R) images.
  ##
  mountPath: /bitnami/mongodb

  ## The subdirectory of the volume to mount to, useful in dev environments
  ## and one PV for multiple services.
  ##
  subPath: ""

  ## mongodb data Persistent Volume Storage Class
  ## If defined, storageClassName: <storageClass>
  ## If set to "-", storageClassName: "", which disables dynamic provisioning
  ## If undefined (the default) or set to null, no storageClassName spec is
  ##   set, choosing the default provisioner.  (gp2 on AWS, standard on
  ##   GKE, AWS & OpenStack)
  ##
  # storageClass: "-"

  storageClass: "gp3"

  accessModes:
    - ReadWriteOnce
  ## PersistentVolumeClaim size
  ##
```

```
    size: 200Gi
    ## Additional volume annotations
    ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
    ##
    annotations: {}

## Mongos properties
## ref: https://docs.mongodb.com/manual/reference/program/mongos/#bin.mongos
##
mongos:
  ## Number of replicas
  ##
  replicas: 16
  ## Configure resource requests and limits
  ## ref: http://kubernetes.io/docs/user-guide/compute-resources/
  ##
  resources: {}
  ## resources:
      # We usually recommend not to specify default resources and to leave this as a conscious
      # choice for the user. This also increases chances charts run on environments with little
      # resources, such as Minikube. If you do want to specify resources, uncomment the following
      # lines, adjust them as necessary, and remove the curly braces after 'resources:'.
  ##  limits:
  ##     cpu: 8
  ##     memory: 32Gi
  ##  requests:
  ##     cpu: 8
  ##     memory: 32Gi



  ## Deployment pod host aliases
  ## https://kubernetes.io/docs/concepts/services-networking/add-entries-to-pod-etc-hosts-
with-host-aliases/
  ##
  ## Allowed values: soft, hard
  ##
  podAntiAffinityPreset: soft
  ## Node affinity preset
  ## ref: https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#node-affinity
  ## Allowed values: soft, hard
  ##
  nodeAffinityPreset:
    ## Node affinity type
    ## Allowed values: soft, hard
    ##
    type: ""
    ## Node label key to match
    ## E.g.
    ## key: "kubernetes.io/e2e-az-name"
    ##
    key: ""
    ## Node label values to match
    ## E.g.
    ## values:
    ##   - e2e-az1
    ##   - e2e-az2
    ##
    values: []
  ## Affinity for pod assignment
  ## ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity
  ## Note: mongos.podAffinityPreset, mongos.podAntiAffinityPreset, and mongos.nodeAffinityPreset will be
ignored when it's set
  ##
  updateStrategy:
    type: RollingUpdate
```

```
  useStatefulSet: false
  ## When using a statefulset, you can enable one service per replica
  ## This is useful when exposing the mongos through load balancers to make sure clients
  ## connect to the same mongos and therefore can follow their cursors
  ##
  servicePerReplica:
    enabled: false
    ## Additional service annotations (evaluate as a template)
    ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
    ##
    annotations: {}
    ## Service type
    ## ref: https://kubernetes.io/docs/concepts/services-networking/service/#publishing-
services-service-types
    ##
    type: ClusterIP
    ## External traffic policy
    ## ref: https://kubernetes.io/docs/concepts/services-networking/service/#publishing-
services-service-types
    ##
    externalTrafficPolicy: Cluster
    ## MongoDB(R) Service port and Container Port
    ##
    port: 27017

  ## Pod disruption budget
  ##
  pdb:
    enabled: false
    ## Use 0 to disable
    ##
    minAvailable: 0
    ## Use 0 to disable
    ##
    maxUnavailable: 1
  ## K8s Service Account.
  ## ref: https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/
  ##
  serviceAccount:
    ## Specifies whether a ServiceAccount should be created for mongos.
    ##
    create: false


## Properties for all of the pods in the cluster (shards, config servers and mongos)
##
common:
  ## Use hostnames instead of IP addresses
  ##
  useHostnames: true
  ## Whether enable/disable IPv6 on MongoDB(R)
  ## ref: https://github.com/bitnami/bitnami-docker-mongodb/blob/master/README.
md#enabling/disabling-ipv6
  ##
  mongodbEnableIPv6: false
  ## Whether enable/disable DirectoryPerDB on MongoDB(R)
  ## ref: https://github.com/bitnami/bitnami-docker-mongodb/blob/master/README.md#enabling/
disabling-directoryperdb
  ##
  mongodbDirectoryPerDB: false
  ## System log verbosity level
  ## ref: https://docs.mongodb.com/manual/reference/program/mongo/#cmdoption-mongo-ipv6
  ##
  mongodbSystemLogVerbosity: 0
  ## MongoDB(R) System Log configuration
  ## ref: https://github.com/bitnami/bitnami-docker-mongodb#configuring-system-log-verbosity-level
  ##
```

```
  mongodbDisableSystemLog: false
  ## Maximum peer node waiting timeout (in seconds)
  ##
  mongodbMaxWaitTimeout: 120

## Init containers parameters:
## volumePermissions: Change the owner and group of the persistent volume mountpoint to
runAsUser:fsGroup values from the securityContext section.
##
volumePermissions:
  enabled: false
  image:
    registry: docker.io
    repository: bitnami/bitnami-shell
    tag: 10-debian-10-r126
    pullPolicy: Always
    ## Optionally specify an array of imagePullSecrets.
    ## Secrets must be manually created in the namespace.
    ## ref: https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/
    ##
    # pullSecrets:
    #   - myRegistryKeySecretName
  resources: {}

## Pod Security Context
## ref: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/
##
securityContext:
  enabled: true
  fsGroup: 1001
  runAsUser: 1001
  runAsNonRoot: true

## Kubernetes Cluster Domain
## ref: https://kubernetes.io/docs/tasks/administer-cluster/dns-custom-nameservers/#introduction
##
clusterDomain: cluster.local

## Kubernetes service type
## ref: https://kubernetes.io/docs/concepts/services-networking/service/
##
service:
  ## Specify an explicit service name
  ##
  name:
  ## Additional service annotations (evaluate as a template)
  ## ref: https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/
  ##
  annotations: {}
  ## Service type
  ## ref: https://kubernetes.io/docs/concepts/services-networking/service/#publishing-
services-service-types
  ##
  type: ClusterIP
  ## External traffic policy
  ## ref: https://kubernetes.io/docs/concepts/services-networking/service/#publishing-
services-service-types
  ##
  externalTrafficPolicy: Cluster
  ## MongoDB(R) Service port and Container Port
  ##
  port: 27017

  ## Specify the sessionAffinity setting for the service. Can be "None" or "ClientIP".
  ## If "ClientIP", consecutive client requests will be directed to the same mongos Pod
  ## ref: https://kubernetes.io/docs/concepts/services-networking/service/#virtual-ips-
and-service-proxies
```

```
  ##
  sessionAffinity: None

## Configure extra options for liveness and readiness probes
## This applies to all the MongoDB(R) in the sharded cluster
## ref: https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-
probes/#configure-probes)
##
livenessProbe:
  enabled: true
  initialDelaySeconds: 360
  periodSeconds: 10
  timeoutSeconds: 5
  failureThreshold: 6
  successThreshold: 1
readinessProbe:
  enabled: true
  initialDelaySeconds: 360
  periodSeconds: 10
  timeoutSeconds: 5
  failureThreshold: 6
  successThreshold: 1

## Prometheus Exporter / Metrics
##
metrics:
  enabled: false

  image:
    registry: docker.io
    repository: bitnami/mongodb-exporter
    tag: 0.11.2-debian-10-r212
    pullPolicy: Always
    ## Optionally specify an array of imagePullSecrets.
  ## Metrics exporter liveness and readiness probes
  ## ref: https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-
probes/#configure-probes)
  ##
  livenessProbe:
    enabled: false
    initialDelaySeconds: 15
    periodSeconds: 5
    timeoutSeconds: 5
    failureThreshold: 3
    successThreshold: 1
  readinessProbe:
    enabled: false
    initialDelaySeconds: 5
    periodSeconds: 5
    timeoutSeconds: 1
    failureThreshold: 3
    successThreshold: 1

  ## Metrics container port
  ##
  containerPort: 9216

  ## Metrics exporter pod Annotation
  ##
  podAnnotations:
    prometheus.io/scrape: "true"
    prometheus.io/port: "{{ .Values.metrics.containerPort }}"
```

update_ycsb_yaml.sh

This script gets the node info to spread the VMs across them evenly (called by setup_ycsb.sh).

```
#!/bin/bash

j=1
IPs=$(kubectl get nodes | grep workers | grep " Ready" | awk '{ print $1 }')

for n in {1..16};
do
    sed -i '$d' ~/ycsb${n}.yaml
done

for i in ${IPs[@]};
do
    echo "  nodeName: ${i}" | tee -a  ~/ycsb${j}.yaml
    ((j = j + 1))
done
```

setup_ycsb.sh

This script copies the ycsb install to the ycsb pods and then unpacks it.

```
#!/bin/bash

~/update_ycsb_yaml.sh
sleep 10

for i in {1..16};do kubectl apply -f ~/aws_yaml/ycsb${i}.yaml;done
sleep 300

for i in {1..16};do kubectl cp ycsb-0.17.0.tar.gz ycsb${i}:/root/;done
for i in {1..16};do kubectl exec ycsb${i} -- tar -xzf /root/ycsb-0.17.0.tar.gz -C /root/;done
```

run_tanzu_mongo.sh

```
#!/bin/bash
IPs=$(kubectl get pods -o wide | grep mongos | awk '{ print $6 }')
nodes=$(kubectl get nodes | grep workers | grep " Ready" | awk '{ print $1 }')
run=${1}

mkdir -p ~/mongodb/results/${run}
run_folder=~/mongodb/results/${run}
cp ~/tanzu-mongodb-values.yaml ${run_folder}/tanzu-mongodb-values.yaml
kubectl get nodes -o wide | tee -a ${run_folder}/nodes.txt
kubectl get pods -o wide | tee -a  ${run_folder}/pods.txt

for node in ${nodes[@]};
do
    kubectl describe node ${node} | tee -a ${run_folder}/${node}_description.txt
done

j=1
for i in ${IPs[@]}; do
   kubectl exec -it ycsb$j -- nohup bash -c "/root/ycsb-0.17.0/bin/ycsb run mongodb -s -P /root/
ycsb-0.17.0/workloads/workloadc -threads 64 -p mongodb.url="mongodb://ptuser:password1@${i}/ycsb" -p
recordcount=500000000 -p operationcount=30000000 > /root/ycsb$j.out &"
   ((j = j + 1))
done
exit 0
```

run_aws_mongo.sh

```
#!/bin/bash
IPs=$(kubectl get pods -o wide | grep mongos | awk '{ print $6 }')
nodes=$(kubectl get nodes | grep workers | grep " Ready" | awk '{ print $7 }')
run=${1}

mkdir -p ~/mongodb/results/${run}
run_folder=~/mongodb/results/${run}
cp ~/aws-mongodb-values.yaml ${run_folder}/aws-mongodb-values.yaml
kubectl get nodes -o wide | tee -a ${run_folder}/nodes.txt
kubectl get pods -o wide | tee -a  ${run_folder}/pods.txt

for node in ${nodes[@]};
do
    kubectl describe node ${node} | tee -a ${run_folder}/${node}_description.txt
done

j=1
for i in ${IPs[@]}; do
   kubectl exec -it ycsb$j -- nohup bash -c "/root/ycsb-0.17.0/bin/ycsb run mongodb -s -P /root/
ycsb-0.17.0/workloads/workloadc -threads 64 -p mongodb.url="mongodb://ptuser:password1@${i}/ycsb" -p
recordcount=500000000 -p operationcount=30000000 > /root/ycsb$j.out &"
   ((j = j + 1))
done
exit 0
```

hadoop-values-tanzu.yaml

```
# The base hadoop image to use for all components.
# See this repo for image build details: https://github.com/Comcast/kube-yarn/tree/master/image
image:
  repository: danisla/hadoop
  tag: 2.9.0
  pullPolicy: IfNotPresent

# The version of the hadoop libraries being used in the image.
hadoopVersion: 2.9.0

# Select antiAffinity as either hard or soft, default is soft
antiAffinity: "soft"

hdfs:
  nameNode:
    pdbMinAvailable: 1

    resources:
      requests:
        memory: "256Mi"
        cpu: "10m"
      limits:
        memory: "2048Mi"
        cpu: "1000m"

  dataNode:
    replicas: 1

    pdbMinAvailable: 1

    resources:
      requests:
        memory: "256Mi"
        cpu: "10m"
      limits:
        memory: "2048Mi"
```

```
        cpu: "1000m"

  webhdfs:
    enabled: false

yarn:
  resourceManager:
    pdbMinAvailable: 1

    resources:
      requests:
        memory: "256Mi"
        cpu: "10m"
      limits:
        memory: "2048Mi"
        cpu: "2000m"

  nodeManager:
    pdbMinAvailable: 1

    # The number of YARN NodeManager instances.
    replicas: 1

    # Create statefulsets in parallel (K8S 1.7+)
    parallelCreate: false

    # CPU and memory resources allocated to each node manager pod.
    # This should be tuned to fit your workload.
    resources:
      requests:
        memory: "2048Mi"
        cpu: "1000m"
      limits:
        memory: "2048Mi"
        cpu: "1000m"

persistence:
  nameNode:
    enabled: true
    storageClass: "tkg-clusters"
    accessMode: ReadWriteOnce
    size: 200Gi

  dataNode:
    enabled: true
    storageClass: "tkg-clusters"
    accessMode: ReadWriteOnce
    size: 500Gi
```

hadoop-env.sh

```
# export JAVA_HOME=${JAVA_HOME}
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
```

hadoop.conf

```
# Hadoop home
hibench.hadoop.home     /data/hadoop-2.7.3


# The path of hadoop executable
hibench.hadoop.executable     ${hibench.hadoop.home}/bin/hadoop

# Hadoop configraution directory
hibench.hadoop.configure.dir  ${hibench.hadoop.home}/etc/hadoop

# The root HDFS path to store HiBench data
hibench.hdfs.master       hdfs://<NAMENODE_IP>:9000


# Hadoop release provider. Supported value: apache
hibench.hadoop.release    apache
```

spark.conf

```
# Spark home
hibench.spark.home      /opt/spark


# Spark master
#   standalone mode: spark://xxx:7077
#   YARN mode: yarn-client
hibench.spark.master    k8s://<CONTROL PLANE ADDRESS>:6443

# executor number and cores when running on Yarn
hibench.yarn.executor.num     8
hibench.yarn.executor.cores   4
spark.executor.instances 16
spark.executor.memory 80g
spark.executor.cores 22
spark.kubernetes.container.image docker.io/wangyijian/spark:v2.4.8

# executor and driver memory in standalone & YARN mode
spark.driver.memory    4g


# set spark parallelism property according to hibench's parallelism value
spark.default.parallelism     ${hibench.default.map.parallelism}

# set spark sql's default shuffle partitions according to hibench's parallelism value
spark.sql.shuffle.partitions  ${hibench.default.shuffle.parallelism}

#======================================================
# Spark Streaming
#======================================================
# Spark streaming Batchnterval in millisecond (default 100)
hibench.streambench.spark.batchInterval          100

# Number of nodes that will receive kafka input (default: 4)
hibench.streambench.spark.receiverNumber        4

# Indicate RDD storage level. (default: 2)
# 0 = StorageLevel.MEMORY_ONLY
# 1 = StorageLevel.MEMORY_AND_DISK_SER
# other = StorageLevel.MEMORY_AND_DISK_SER_2
hibench.streambench.spark.storageLevel 2

# indicate whether to test the write ahead log new feature (default: false)
hibench.streambench.spark.enableWAL false
```

```
# if testWAL is true, this path to store stream context in hdfs shall be specified. If false, it can
be empty (default: /var/tmp)
hibench.streambench.spark.checkpointPath /var/tmp

# whether to use direct approach or not (dafault: true)
hibench.streambench.spark.useDirectMode true
```

hibench.conf

```
# Data scale profile. Available value is tiny, small, large, huge, gigantic and bigdata.
# The definition of these profiles can be found in the workload's conf file i.e. conf/workloads/
micro/wordcount.conf
hibench.scale.profile                bigdata
# Mapper number in hadoop, partition number in Spark
hibench.default.map.parallelism         316

# Reducer nubmer in hadoop, shuffle partition number in Spark
hibench.default.shuffle.parallelism     316


#======================================================
# Report files
#======================================================
# default report formats
hibench.report.formats              "%-12s %-10s %-8s %-20s %-20s %-20s %-20s\n"

# default report dir path
hibench.report.dir          ${hibench.home}/report

# default report file name
hibench.report.name         hibench.report

# input/output format settings. Available formats: Text, Sequence, Null.
sparkbench.inputformat         Sequence
sparkbench.outputformat        Sequence

# hibench config folder
hibench.configure.dir        ${hibench.home}/conf

# default hibench HDFS root
hibench.hdfs.data.dir               ${hibench.hdfs.master}/HiBench

# path of hibench jars
hibench.hibench.datatool.dir          ${hibench.home}/autogen/target/autogen-8.0-SNAPSHOT-jar-with-
dependencies.jar
hibench.common.jar                      ${hibench.home}/common/target/hibench-common-8.0-SNAPSHOT-
jar-with-dependencies.jar
hibench.sparkbench.jar                  ${hibench.home}/sparkbench/assembly/target/sparkbench-
assembly-8.0-SNAPSHOT-dist.jar
hibench.streambench.stormbench.jar      ${hibench.home}/stormbench/streaming/target/stormbench-
streaming-8.0-SNAPSHOT.jar
hibench.streambench.gearpump.jar        ${hibench.home}/gearpumpbench/streaming/target/gearpumpbench-
streaming-8.0-SNAPSHOT-jar-with-dependencies.jar
hibench.streambench.flinkbench.jar      ${hibench.home}/flinkbench/streaming/target/flinkbench-
streaming-8.0-SNAPSHOT-jar-with-dependencies.jar

#======================================================
# workload home/input/ouput path
#======================================================
hibench.hive.home           ${hibench.home}/hadoopbench/sql/target/${hibench.hive.release}
hibench.hive.release        apache-hive-3.0.0-bin
hibench.hivebench.template.dir      ${hibench.home}/hadoopbench/sql/hive_template
hibench.bayes.dir.name.input${hibench.workload.dir.name.input}
hibench.bayes.dir.name.output       ${hibench.workload.dir.name.output}
```

```
hibench.mahout.release.apache   apache-mahout-distribution-0.11.0
hibench.mahout.release                    ${hibench.mahout.release.${hibench.hadoop.release}}
hibench.mahout.home                    ${hibench.home}/hadoopbench/mahout/target/${hibench.
mahout.release}

hibench.masters.hostnames
hibench.slaves.hostnames

hibench.workload.input
hibench.workload.output
hibench.workload.dir.name.input         Input
hibench.workload.dir.name.output        Output

hibench.nutch.dir.name.input${hibench.workload.dir.name.input}
hibench.nutch.dir.name.output       ${hibench.workload.dir.name.output}
hibench.nutch.nutchindexing.dir     ${hibench.home}/hadoopbench/nutchindexing/
hibench.nutch.release               nutch-1.2
hibench.nutch.home          ${hibench.home}/hadoopbench/nutchindexing/target/${hibench.nutch.release}

hibench.dfsioe.dir.name.input       ${hibench.workload.dir.name.input}
hibench.dfsioe.dir.name.output      ${hibench.workload.dir.name.output}


#=====================================================
# Streaming General
#=====================================================
# Indicate whether in debug mode for correctness verfication (default: false)
hibench.streambench.debugMode false
hibench.streambench.sampleProbability 0.1
hibench.streambench.fixWindowDuration            10000
hibench.streambench.fixWindowSlideStep           10000


#=====================================================
# Kafka for streaming benchmarks
#=====================================================
hibench.streambench.kafka.home                   /PATH/TO/YOUR/KAFKA/HOME
# zookeeper host:port of kafka cluster, host1:port1,host2:port2...
hibench.streambench.zkHost
# Kafka broker lists, written in mode host:port,host:port,..
hibench.streambench.kafka.brokerList
hibench.streambench.kafka.consumerGroup          HiBench
# number of partitions of generated topic (default 20)
hibench.streambench.kafka.topicPartitions       20
# consumer group of the consumer for kafka (default: HiBench)
hibench.streambench.kafka.consumerGroup HiBench
# Set the starting offset of kafkaConsumer (default: largest)
hibench.streambench.kafka.offsetReset largest


#=====================================================
# Data generator for streaming benchmarks
#=====================================================
# Interval span in millisecond (default: 50)
hibench.streambench.datagen.intervalSpan         50
# Number of records to generate per interval span (default: 5)
hibench.streambench.datagen.recordsPerInterval   5
# fixed length of record (default: 200)
hibench.streambench.datagen.recordLength         200
# Number of KafkaProducer running on different thread (default: 1)
hibench.streambench.datagen.producerNumber       1
# Total round count of data send (default: -1 means infinity)
hibench.streambench.datagen.totalRounds          -1
# Number of total records that will be generated (default: -1 means infinity)
hibench.streambench.datagen.totalRecords        -1
# default path to store seed files (default: ${hibench.hdfs.data.dir}/Streaming)
hibench.streambench.datagen.dir                          ${hibench.hdfs.data.dir}/Streaming
```

```
# default path setting for genearate data1 & data2
hibench.streambench.datagen.data1.name                  Seed
hibench.streambench.datagen.data1.dir                   ${hibench.streambench.datagen.dir}/${hibench.
streambench.datagen.data1.name}
hibench.streambench.datagen.data2_cluster.dir           ${hibench.streambench.datagen.
dir}/Kmeans/Cluster
hibench.streambench.datagen.data2_samples.dir           ${hibench.streambench.datagen.
dir}/Kmeans/Samples

#========================================================
# MetricsReader for streaming benchmarks
#========================================================
# Number of sample records for `MetricsReader` (default: 5000000)
hibench.streambench.metricsReader.sampleNum      5000000
# Number of thread for `MetricsReader` (default: 20)
hibench.streambench.metricsReader.threadNum      20
# The dir where stored the report of benchmarks (default: ${hibench.home}/report)
hibench.streambench.metricsReader.outputDir      ${hibench.home}/report
```

run-spark-tanzu.sh

```
#!/bin/bash
nodes=$(kubectl get node -o wide | grep ip- | awk '{ print $7 }')
run=${1}
date=${2}

mkdir -p ~/results/spark/${date}/${run}
touch ~/results/spark/${date}/${run}/spark.out

kubectl exec test -- bash /data/HiBench/bin/workloads/ml/kmeans/spark/run.sh 2>&1 | tee ~/results/
spark/${date}/${run}/spark.out
sleep 30

kubectl cp test:/data/HiBench/conf/spark.conf ~/results/spark/${date}/${run}/spark.conf
kubectl cp test:/data/HiBench/conf/hadoop.conf ~/results/spark/${date}/${run}/hadoop.conf
kubectl cp test:/data/HiBench/conf/hibench.conf ~/results/spark/${date}/${run}/hibench.conf
kubectl cp test:/data/HiBench/conf/workloads/ml/kmeans.conf ~/results/
spark/${date}/${run}/kmeans.conf

exit 0
```

hadoop-values-aws.yaml

```
# The base hadoop image to use for all components.
# See this repo for image build details: https://github.com/Comcast/kube-yarn/tree/master/image
image:
  repository: danisla/hadoop
  tag: 2.9.0
  pullPolicy: IfNotPresent

# The version of the hadoop libraries being used in the image.
hadoopVersion: 2.9.0

# Select antiAffinity as either hard or soft, default is soft
antiAffinity: "soft"

hdfs:
  nameNode:
    pdbMinAvailable: 1

    resources:
      requests:
        memory: "256Mi"
        cpu: "10m"
```

```
      limits:
        memory: "2048Mi"
        cpu: "1000m"

  dataNode:
    replicas: 1

    pdbMinAvailable: 1

    resources:
      requests:
        memory: "256Mi"
        cpu: "10m"
      limits:
        memory: "2048Mi"
        cpu: "1000m"

  serviceAccountName: spark

  webhdfs:
    enabled: false

yarn:
  resourceManager:
    pdbMinAvailable: 1

    resources:
      requests:
        memory: "256Mi"
        cpu: "10m"
      limits:
        memory: "2048Mi"
        cpu: "2000m"

  nodeManager:
    pdbMinAvailable: 1

    # The number of YARN NodeManager instances.
    replicas: 1

    # Create statefulsets in parallel (K8S 1.7+)
    parallelCreate: false

    # CPU and memory resources allocated to each node manager pod.
    # This should be tuned to fit your workload.
    resources:
      requests:
        memory: "2048Mi"
        cpu: "1000m"
      limits:
        memory: "2048Mi"
        cpu: "1000m"

  serviceAccountName: spark

persistence:
  nameNode:
    enabled: true
    storageClass: "gp3"
    accessMode: ReadWriteOnce
    size: 200Gi

  dataNode:
    enabled: true
    storageClass: "gp3"
    accessMode: ReadWriteOnce
    size: 500Gi
```

run_spark_aws.sh

```
#!/bin/bash
nodes=$(kubectl get node -o wide | grep ip- | awk '{ print $7 }')
run=${1}
date=${2}

mkdir -p ~/results/spark/${date}/${run}
touch ~/results/spark/${date}/${run}/spark.out

for ip in ${nodes[@]};
do
    ssh -i eks-public-key.pem ec2-user@$ip nmon -F /tmp/${ip}_${run}.nmon -s5 -c545 -J -t
done

kubectl exec test -- bash /data/HiBench/bin/workloads/ml/kmeans/spark/run.sh 2>&1 | tee ~/results/
spark/${date}/${run}/spark.out

for ip in ${nodes[@]};
do
    ssh -i eks-public-key.pem ec2-user@$ip pkill nmon
done

sleep 30

~/cp_spark.sh ${run}

exit 0
```

nginx.conf

```
user root;
worker_priority 0;
worker_processes 20;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 20800;
        multi_accept on;
        use epoll;
        accept_mutex on;
}

http {
        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 65;
        types_hash_max_size 2048;

        error_log off;
        access_log off;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        server
        {
                listen 0.0.0.0:80 ;
                location /
                {
                        root  /usr/share/nginx/html ;
                        index index.html ;
                }
        }
}
```

enable_gslbpoolmember.py

```
#!/usr/bin/python

import os
import sys
import json
from avi.sdk.avi_api import ApiSession
from requests.packages import urllib3

urllib3.disable_warnings()

token=os.environ.get('API_TOKEN')
user=os.environ.get('USER')
tenant=os.environ.get('TENANT')

with ApiSession("localhost", user, token=token, tenant=tenant) as session:
  gslb_service = session.get_object_by_name('gslbservice', 'webapp')
  gslb_service['groups'][1]['members'][0]['enabled'] = True
  r = session.put('gslbservice/%s' % gslb_service['uuid'], data=json.dumps(gslb_service))
```

Read the report at https://facts.pt/Axmmz8G ▶

This project was commissioned by Dell Technologies.

Principled Technologies®

Facts matter.®

Principled Technologies is a registered trademark of Principled Technologies, Inc.
All other product names are the trademarks of their respective owners.

DISCLAIMER OF WARRANTIES; LIMITATION OF LIABILITY:
Principled Technologies, Inc. has made reasonable efforts to ensure the accuracy and validity of its testing, however, Principled Technologies, Inc. specifically disclaims any warranty, expressed or implied, relating to the test results and analysis, their accuracy, completeness or quality, including any implied warranty of fitness for any particular purpose. All persons or entities relying on the results of any testing do so at their own risk, and agree that Principled Technologies, Inc., its employees and its subcontractors shall have no liability whatsoever from any claim of loss or damage on account of any alleged error or defect in any testing procedure or result.

In no event shall Principled Technologies, Inc. be liable for indirect, special, incidental, or consequential damages in connection with its testing, even if advised of the possibility of such damages. In no event shall Principled Technologies, Inc.'s liability, including for direct damages, exceed the amounts paid in connection with Principled Technologies, Inc.'s testing. Customer's sole and exclusive remedies are as set forth herein.