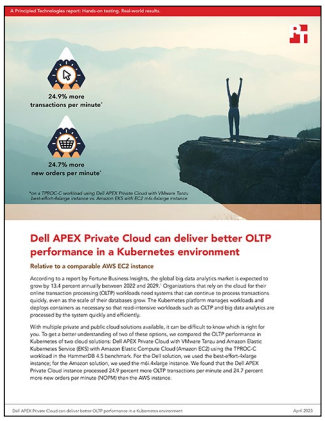

The science behind the report:

Dell APEX Private Cloud can deliver better OLTP performance in a Kubernetes environment

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 Dell APEX Private Cloud can deliver better OLTP performance in a Kubernetes environment.

We concluded our hands-on testing on February 23, 2023. 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 February 14, 2023 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. We ran each test three times and report the median result.

	Dell APEX Private Cloud with VMware Tanzu® solution	Amazon EKS® with EC2® solution
New orders per minute (NOPM)	105,484	84,575
Transactions per minute (TPM)	245,438	196,466

System configuration information

Table 2: Detailed information on the systems we tested.

System configuration information	Dell APEX Private Cloud solution
BIOS name and version	Dell 1.6.5
Non-default BIOS settings	Performance system profile
Operating system, name, and version/build number	VMware ESXi®, 7.0.3, 20328353
Power management policy	Performance
Processor	
Number of processors	2
Vendor and model	Intel® Xeon® Platinum 8358
Core count (per processor)	32
Core frequency (GHz)	2.60
Stepping	1
Memory module(s)	
Total memory in system (GB)	512
Number of memory modules	16
Vendor and model	Hynix® HMAA4GR7CJR8N-XN
Size (GB)	32
Type	DDR4
Speed (MHz)	3,200
Speed running in the server (MHz)	3,200
Storage controller 1	
Vendor and model	Dell HBA355i Front
Cache size (GB)	0
Firmware version	17.15.08.00
Storage controller 2	
Vendor and model	Dell BOSS-S2
Cache size (GB)	0
Firmware version	2.5.13.4008
Local storage (type A)	
Number of drives	2
Drive vendor and model	Micron® MTFDDAV480TDS
Drive size (GB)	447
Drive information (speed,interface,type)	6Gbps, SATA, SSD

System configuration information	Dell APEX Private Cloud solution
Local storage (type B)	
Number of drives	6
Drive vendor and model	Micron MTFDDAK3T8TDT
Drive size (TB)	3.5
Drive information (speed, interface, type)	6Gbps, SATA, SSD
Local storage (type C)	
Number of drives	2
Drive vendor and model	Dell Ent NVMe v2 AGN MU U.2 1.6TB
Drive size (TB)	1.6
Drive information(speed,interface,type)	16GT/s, PCIe, SSD
Network adapter 1	
Vendor and model	Mellanox® MT27800
Number and type of ports	2 x 25GbE
Driver version	16.32.20.04
Network adapter 2	
Vendor and model	Broadcom® BCM57414 NetXtreme-E
Number and type of ports	2 x 25GbE
Driver version	22.00.07.60
Cooling fans	
Number of cooling fans	16
Power supplies	
Vendor and model	Dell PWR SPLY,1100W,RDNT,LTON
Number of power supplies	2
Wattage of each (W)	1,100

Table 3: Detailed information on the cloud instance we tested.

System configuration information	Amazon EKS with EC2 solution
General information	
Date testing ended	2/23/2023
Cloud service provider (CSP)	AWS®
Region	us-east-2
Workload information	
Workload name and version	HammerDB v4.5 TPROC-C
Workload or software specific parameters	500 warehouses, 16 vusers
Iterations and result choice	3 test runs, median reported
Cloud VM or instance details	
Number of VMs	4
VM or instance size	m6i.4xlarge
BIOS name	legacy-bios
vCPU	16
Number of cores/threads	16
Memory (GB)	64
Underlying processor model	Intel Xeon Platinum 8375C
Operating system information	
Image or template name and UUID	amazon-eks-node-1.24-v20230203 ami-07bbfd81503ebcb2e
Operating system name	Amazon Linux 2
Kernel version	5.4.228-131.415.amzn2.x86_64
Date patches last applied	2/20/2023
Changes made from CSP image	No
Instance storage (volume type 1)	
Number of volumes	1
Volume use in this test	Data drive
CSP volume type	EBS gp3
Volume size (GB)	160
IOPS requested	12,000
Throughput requested (MB/s)	1,000
Encryption type	Not encrypted

System configuration information	Amazon EKS with EC2 solution
Instance storage (volume type 2)	
Number of volumes	1
Volume use in this test	OS
CSP volume type	EBS gp3
Volume size (GB)	40
IOPS requested	Default
Throughput requested	Default
Encryption type	Not encrypted

How we tested

Testing overview

In this study, we used the TPROC-C workload from the HammerDB 4.5 benchmark to compare the OLTP performance of containerized SQL Server instances in a VMware Tanzu/Kubernetes cluster in Dell APEX Private Cloud solution and an EKS/Kubernetes cluster in AWS EKS solution. The Dell APEX Private Cloud solution was powered by a four-node Dell VxRail E660F cluster. Each node had dual Intel Xeon Platinum 8358 processors and 512 GB of RAM. The Dell APEX Private Cloud solution also had a vSAN datastore that had two disk groups on each node. Each disk group had an NVMe SSD drive as cache drive and three SATA SSD drives as capacity drives. We created a four-node (best-effort-4xlarge) VMware Tanzu workload cluster on Dell APEX Private Cloud. For the AWS solution, we created an EKS cluster with four nodes and each node was an EC2 m6i.4xlarge instance using EBS gp3 storage volume with IOPS set to 12K. We created a database schema with 500 warehouses and 16 virtual users in both clusters. We measured the number of OLTP transactions per minute each solution handled and the number of new orders per minute each solution processed.

The following sections describe the steps we took to configure the test environment and run the tests.

Setting up a VMware Tanzu Kubernetes cluster in Dell APEX Private Cloud

1. Log into the Kubernetes CLI VM using SSH.
2. Create a file called mssql-tkg-cluster.yaml with the following content:

```
apiVersion: run.tanzu.vmware.com/v1alpha1      #TKGS API endpoint
kind: TanzuKubernetesCluster                   #required parameter
metadata:
  name: mssql-cluster                          #cluster name, user defined
  namespace: demo1                             #vsphere namespace
spec:
  distribution:
    version: v1.21
  settings:
    storage:
       defaultClass: vsan-default-storage-policy
  topology:
    controlPlane:
      count: 1                                           #number of control plane nodes
      class: best-effort-medium                          #vmclass for control plane nodes
      storageClass: vsan-default-storage-policy          #storageclass for control plane
    workers:
      count: 4                                           #number of worker nodes
      class: best-effort-4xlarge                         #vmclass for worker nodes
      storageClass: vsan-default-storage-policy          #storageclass for worker nodes
```

3. Create a workload cluster:

```
kubectl apply -f mssql-tkg-cluster.yaml
```

4. Create a file called pvc.yaml for persistent volume claim:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mssql-data
  annotations:
    volume.beta.kubernetes.io/storage-class: vsan-default-storage-policy
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 160Gi
```

5. Create the PVC:

```
kubectl apply -f pvc.yaml
```

Setting up an EKS Kubernetes cluster in AWS

Creating a harness EC2 VM

1. From the AWS EC2 dashboard, click Instances, and click Launch instances.
2. Type `Ubuntu` in the search bar, and select Ubuntu Server 22.04.
3. For instance type, select t2.micro.
4. For Key pair, select the key pair for ssh login.
5. Click Launch instance.
6. Log into the harness VM using SSH command.
7. Install the latest updates:

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

8. Install prerequisites:

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

9. Update the instance:

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

10. Install and configure 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
```

11. Install ekscli:

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/
eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
sudo mv /tmp/eksctl /usr/local/bin
```

12. 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
```

13. Install helm:

```
curl -sSL https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash
helm repo add stable https://charts.helm.sh/stable
```

Creating an AWS 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
--region us-east-2
```

2. Create a file called cluster-role-trust-policy.json:

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

3. Create an IAM 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 an SSH key-pair:

```
aws ec2 create-key-pair --region us-east-2 --key-name eks-public-key  --query "KeyMaterial" --output
text > eks-public-key.pem
```

6. From AWS CloudFormation console, click the VPC stack you just created, and click the Outputs tab. Make a note of the four SubnetIds. Creating an EKS cluster in the next step will require at least two of the subnets.
7. Create an EKS cluster without nodegroup:

```
eksctl create cluster --name ekscluster --ssh-access=true --ssh-public-key=eks-public-
key --region=us-east-2 --vpc-private-subnets subnet-0f3e3b4ea6db0befb,subnet-0a0453f4707462555
--without-nodegroup
```

8. Create a file called nodegroup.yaml, and edit it to the following:

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: ekscluster
  region: us-east-2
nodeGroups:
  - name: worker-nodes
    labels: { role: worker }
    instanceType: m6i.4xlarge
    desiredCapacity: 4
    ssh:
      publicKeyName: eks-public-key
    volumeSize: 40
    privateNetworking: true
```

9. Create the nodegroup for the EKS cluster:

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

10. Download the EBS csi driver IAM policy from Github:

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

11. Create the EBS csi driver policy:

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

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

```
eksctl utils associate-iam-oidc-provider --region=us-east-2 --cluster=ekscluster --approve
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 --region us-east-2
```

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 EBS csi driver using 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 for EBS storage class:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  csi.storage.k8s.io/fstype: ext4
  type: gp3
  throughput: "1000"
  iops: "12000"
  encrypted: "false"
reclaimPolicy: Delete
```

16. Create the storage class on the cluster:

```
kubectl apply -f storageclass.yaml
```

17. Set the new storage class as 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"}}}'
```

18. Create a file called pvc.yaml for persistent volume claim:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mssql-data
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: ebs-sc
  resources:
    requests:
      storage: 160Gi
```

19. Create the PVC:

```
kubectl apply -f pvc.yaml
```

Creating identical workloads on both Kubernetes clusters

We took the following steps to deploy identical workloads on both Dell APEX Private Cloud and AWS/EKS solutions.

1. Create a secret for SA password:

```
kubectl create secret generic mssql-secret --from-literal=SA_PASSWORD="Password1!"
```

2. Create a file called psp.yaml for Pod Security Policy with the following content:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: administrator-cluster-role-binding
roleRef:
  kind: ClusterRole
  name: psp:vmware-system-privileged
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  name: system:authenticated
  apiGroup: rbac.authorization.k8s.io
```

3. Apply the PSP policy to the cluster:

```
kubectl apply -f psp.yaml
```

4. Create a file with the following content:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mssql-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mssql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mssql
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: mssql
        image: mcr.microsoft.com/mssql/server:2019-latest
        resources:
          requests:
            memory: "32Gi"
            cpu: "15"
          limits:
            memory: "32Gi"
            cpu: "15"
        securityContext:
          runAsUser: 0
        env:
        - name: MSSQL_PID
          value: "Developer"
        - name: ACCEPT_EULA
          value: "Y"
        - name: MSSQL_SA_PASSWORD
          valueFrom:
            secretKeyRef:
```

```
            name: mssql-secret
            key: SA_PASSWORD
        volumeMounts:
        - name: mssqldb
          mountPath: /var/opt/mssql
      volumes:
      - name: mssqldb
        persistentVolumeClaim:
          claimName: mssql-data
```

5. Create the SQL Server deployment:

```
kubectl apply -f mssql-deployment.yaml
```

6. Create a file with the following content:

```
apiVersion: v1
kind: Pod
metadata:
  name: hammerdb
spec:
  containers:
  - name: hammerdb
    image: tpcorg/hammerdb:v4.5
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "while true; do sleep 30; done;" ]
    ports:
      - containerPort: 80
```

7. Deploy the HammerDB POD:

```
kubectl apply -f hammerdb.yaml
```

8. Copy the odbc driver for SQL Server in the HammerDB POD:

```
kubectl exec -it hammerdb -- /bin/bash
root@hammerdb: cp /etc/odbcinst.ini /usr/local/unixODBC/etc/odbcinst.ini
```

9. Modify sample_scripts/tprocc/mssqls_tprocc_build.tcl in the HammerDB POD with the following content:

```
#!/bin/tclsh
# maintainer: Pooja Jain
#Set the path where you want to log results.
puts "SETTING CONFIGURATION"
dbset db mssqls
dbset bm TPC-C

diset connection mssqls_host <POD IP address of the SQL POD>
diset connection mssqls_linux_server <POD IP address of the SQL POD>
diset connection mssqls_authentication windows
diset connection mssqls_odbc_driver "ODBC Driver 17 for SQL Server"
diset connection mssqls_uid sa
diset connection mssqls_pass Password1!

diset tpcc mssqls_count_ware 500
diset tpcc mssqls_num_vu 15
diset tpcc mssqls_dbase tpcc

print dict
vuset logtotemp 1
buildschema
waittocomplete
```

10. Create a TPCC database schema from the HammerDB POD:

```
./hammerdbcli auto sample_scripts/tprocc/mssqls_tprocc_build.tcl
```

11. Create a full backup for the database you just created:

```
sqlcmd -S <IP address for the SQL POD> -U sa -Q "BACKUP DATABASE tpcc TO DISK = N'/var/opt/mssql/
data/tpcc.bak' WITH NOFORMAT, NOINIT, NAME = 'tpcc-full', SKIP, NOREWIND, NOUNLOAD, STATS = 10"
sqlcmd -S <IP address for the SQL POD> -U sa -Q "BACKUP LOG tpcc TO DISK = N'/var/opt/mssql/
data/tpcc_LogBackup.bak' WITH NOFORMAT, NOINIT, NAME = 'tpcc-LogBackup', NOSKIP, NOREWIND,
NOUNLOAD, STATS = 5"
```

Running the tests

In this section, we list the steps to run the HammerDB benchmark on both Kubernetes clusters.

1. Log into the HammerDB POD shell:

```
kubectl exec -it hammerdb -- /bin/bash
```

2. Modify the mssqls_tprocc_run.tcl file with the following content:

```
# maintainer: Pooja Jain
#Set the path for logs directory. By Default logs are logged in /tmp
puts "SETTING CONFIGURATION"
dbset db mssqls
dbset bm TPC-C

diset connection mssqls_host <IP address of the SQL POD>
diset connection mssqls_linux_server <IP address of the SQL POD>
diset connection mssqls_authentication windows
#How to specify the ODBC driver here, it has spaces, in double quotes?
diset connection mssqls_odbc_driver "ODBC Driver 17 for SQL Server"

diset connection mssqls_uid sa
diset connection mssqls_pass Password1!
diset tpcc mssqls_count_ware 500
diset tpcc mssqls_num_vu 15

diset tpcc mssqls_dbase tpcc
diset tpcc mssqls_driver timed
diset tpcc mssqls_total_iterations 10000000
diset tpcc mssqls_checkpoint true
diset tpcc mssqls_rampup 2
diset tpcc mssqls_duration 5
diset tpcc mssqls_timeprofile false

print dict
vuset logtotemp 1
loadscript
puts "TEST STARTED"
vuset vu 15
vucreate
tcstart
tcstatus
vurun
runtimer 500
vudestroy
tcstop
puts "TEST COMPLETE"
```

3. Run the test:

```
./hammerdbcli auto sample_scripts/tprocc/mssqls_tprocc_run.tcl
```

4. Restore the database:

```
sqlcmd -S <IP address of the SQL POD> -U SA -Q "RESTORE DATABASE tpcc FROM DISK = N'/var/opt/mssql/
data/tpcc.bak' WITH FILE = 1, NOUNLOAD, REPLACE, NORECOVERY, STATS = 5"
sqlcmd -S <IP address of the SQL POD> -U SA -Q "RESTORE LOG tpcc FROM DISK = N'/var/opt/mssql/data/
tpcc_LogBackup.bak'"
```

5. Run the same test two more times on each cluster for a total of three tests.

View the original version of this report at https://facts.pt/w2IV9JR

Read the report ▶

This project was commissioned by Dell Technologies.