The science behind the report:

Kubernetes on VMware vSphere vs. bare metal: Which delivered better density and performance?

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 Kubernetes on VMware vSphere vs bare metal: Which delivered better density and performance?.

We concluded our hands-on testing on February 26, 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 February 26, 2021 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

CloudXPRT data analytics workload

Tables 1 and 2 present the results we have selected from the raw data per the criterion used in CloudXPRT testing: the 95th percentile transaction time must be below 90 seconds. All of these results are for one server with 104 hyperthreads, Kubernetes® version 1.18.10, the Higgs1m dataset, and 100 jobs. In the main report, we reported only two of the runs from the bare-metal cluster and three of the runs from the VMware® Tanzu™ Kubernetes Grid (TKG) cluster as a sample demonstrating the similarities in performance. The data we include below further demonstrates the similarities in performance at different Kubernetes cluster configurations.

Table 1: Bare-metal cluster results on CloudXPRT data analytics workload. Higher throughput rates and lower transaction times are better. Source: Principled Technologies.

No. of pods	No. of CPUs/pod	Burstiness (s)	Throughput (jobs/min)	Single-transaction time (s) 95th %tile
3	26	30	1.96	60
5	20	30	1.96	67
2	39	30	1.95	75
1	52	50	1.20	44
2	39	50	1.20	43
3	26	50	1.20	46
5	20	50	1.20	51

Table 2: VMware TKG cluster results on CloudXPRT data analytics workload. Higher throughput rates and lower transaction times are better. Source: Principled Technologies.

No. of nodes	No. of vCPUS/ node	No. of pods	No. of CPUs/pod	Burstiness (s)	Throughput (jobs/min)	Single-transaction time (s) 95th %tile
3	26	3	20	30	1.95	77
3	26	3	20	50	1.19	65
3	26	3	16	50	1.19	74
1	52	1	48	50	1.20	71
1	52	2	20	50	1.19	68
1	52	3	16	50	1.19	76
6	16	6	12	50	1.19	84

We measured the server resources while running the workload and observed that CPU usage was the key resource consumed.

Table 3: Comparison of peak sustained CPU usage for each testbed, running the CloudXPRT data-analytics workload.

Testbed	Test configuration			Throughput (jobs/min)	CPU usage (percentage of all CPU cores)
	# pods	CPUs/pod	Burstiness		
TKG, 3 VMs, 26 vCPUs	3	20	30	1.95	89
Bare metal	3	26	30	1.96	51

The CPU usage for the bare-metal testbed was much lower than the TKG CPU usage (reported in the vCenter performance overview data for the server) because we winnowed results for test configurations that had latencies above the 90-second threshold. Several of these configurations used significantly more CPU. For example, the bare-metal test with data analytics configuration of eight pods and 12 CPUs per pod used 80 percent of the server's CPU resources; however, latency was 528 seconds, well over the threshold.

CloudXPRT web microservices workload

Tables 3 and 4 present the results we have selected from the raw data per the criterion used in CloudXPRT testing: a service-level agreement (SLA) criterion of 3 seconds. All of these results are for one server with 104 hyperthreads and Kubernetes version 1.18.10. The report focuses on four TKG configurations to demonstrate various conditions where performance of the TKG cluster was comparable to or better than that of the bare-metal cluster. In the tables below, we include the rest of the configurations we tested to demonstrate numerous configuration possibilities and their possible strengths and weaknesses versus bare metal.

We used the following settings:

- VM size: 20GB disk and 10GB RAM
- Latency: SLA criterion of under 3 seconds

Table 4: Bare-metal cluster results on CloudXPRT web microservices workload. Higher throughput rates and lower latencies are better. Source: Principled Technologies.

Throughput	Latency (ms)
1,263	820

Table 5: VMware TKG cluster results on CloudXPRT web microservices workload. Higher throughput rates and lower latencies are better. Source: Principled Technologies.

No. of vCPUs	No. of VMs	Total CPU	Throughput	Latency (ms)	Notes
64	1	64	1,256	2,967	
52	1	52	1,367	720	Best combination: second-highest throughput, lowest in latency
48	2	96	1,391	1,710	Best throughput, but higher latency
36	2	72	1,272	2,536	
32	3	96	1,341	1,549	Third-highest throughput, and lower latency than the run with the best throughput
26	3	78	1,258	1,992	
20	4	80	1,266	2,249	
12	8	96	1,286	2,499	
8	12	96	1,283	2,575	

We measured the server resources while running workload and observed that CPU usage was the key resource consumed.

Table 6: Comparison of peak sustained CPU usage for each testbed, running the CloudXPRT web-microservices workload.

Testbed	Throughput	Latency (ms)	CPU usage (percentage of all CPU cores)
TKG, 1 VM, 52 vCPUs	1,367	720	98
Bare metal	1,263	820	65

The CPU usage for the bare-metal testbed was much lower than the TKG CPU usage (reported in the vCenter performance overview data for the server). We attribute this to the way the CloudXPRT harness scales the workload and the greater number of nodes in the TKG testbed. First, we note that this workload has no adjustable test parameters. The test harness increases load on the testbed by increasing the number of concurrent clients. It starts with one client, runs for 60 seconds, and then increases the number of clients by one until it detects a peak in throughput.

The number of clients that produce the peak throughput is not a CloudXPRT metric, but for this description, we noted that both testbeds reached peak throughput with a similar number of clients (about 13). We performed a test on the bare-metal server to determine whether we could drive the application harder. When we artificially increased the number of clients to 27, we measured CPU usage of 98 percent.

System configuration information

Table 7: Detailed information on the systems we tested.

System configuration information	Dell EMC™ PowerEdge™ R740xd (used for VMware TKG cluster)	Dell EMC PowerEdge R740xd (used for bare-metal cluster)
BIOS name and version	Dell 2.8.2	Dell 2.8.2
Non-default BIOS settings	N/A	N/A
Operating system name and version/build number	VMware ESXi, 7.0.1, 6850804	Ubuntu 18.04
Date of last OS updates/patches applied	1/18/2021	1/18/2021
Power management policy	Max Performance	Max Performance
Processor		
Number of processors	2	2
Vendor and model	Intel® Xeon® Platinum 8164	Intel Xeon Platinum 8164
Core count (per processor)	26	26
Core frequency (GHz)	2.00	2.00
Stepping	Model 85 Stepping 4	Model 85 Stepping 4
Memory module(s)		
Total memory in system (GB)	384	384
Number of memory modules	12	12
Vendor and model	Samsung® M393A4K40BB1-CRC	Samsung M393A4K40BB1-CRC
Size (GB)	32	32
Type	DDR-4	DDR-4
Speed (MHz)	2,400	2,400
Speed running in the server (MHz)	2,400	2,400
Storage controller		
Vendor and model	PERC H730P Mini	PERC H730P Mini
Cache size (GB)	2	2
Firmware version	25.5.50005	25.5.6.0009
Local storage (type A)		
Number of drives	2	2
Drive vendor and model	Dell PX05SRB192Y	Dell PX05SRB192Y
Drive size (GB)	1,920	1,920
Drive information (speed, interface, type)	12Gbps, SAS SSD	12Gbps, SAS SSD

System configuration information	Dell EMC™ PowerEdge™ R740xd (used for VMware TKG cluster)	Dell EMC PowerEdge R740xd (used for bare-metal cluster)
Local storage (type B)		
Number of drives	1	N/A
Drive vendor and model	DELL IDSDM	N/A
Drive size (GB)	16	N/A
Drive information (speed, interface, type)	SD	N/A
Network adapter		
Vendor and model	Broadcom 5720 Quad-Port	Broadcom 5720 Quad-Port
Number and type of ports	4 x 1GbE	4 x 1GbE
Driver version	ntg3 4.1.5.0-0vmw.701.0.0.16850804	tg3 3.137
Cooling fans		
Vendor and model	Dell GYNRG-A00	Dell GYNRG-A00
Number of cooling fans	6	6
Power supplies		
Vendor and model	Dell 0PR21CA00	Dell 0TFR9VX03
Number of power supplies	2	2
Wattage of each (W)	1,100	1,100

How we tested

Brief description of the CloudXPRT benchmark and required resources

The CloudXPRT distribution provides a pre-configured Kubespray configuration for installing Kubernetes on a bare-metal server. Version 1.01 uses the Calico CNI for networking. Kubespray installs a Docker environment for Kubernetes, and the backing storage for the etcd daemon is the RAID 1 main volume.

For this single-server cluster, we installed the TKG management cluster ("dev"+"small" options") and worker cluster with Antrea CNI networking.

The CloudXPRT workloads use the "bare-metal" option (with several modifications, which we describe in the methodology below) on the TKG worker cluster because TKG better resembles a cloud cluster than it does a multi-node bare-metal cluster. The main change in the modifications we made is to place all CloudXPRT pods on TKG worker nodes. For pure bare-metal clusters, CloudXPRT deploys worker pods to all nodes, including the management one(s).

For the density-pod test, we made several OS changes to the bare-metal server. We have specified these in the methodology below.

In the remainder of this section, we describe the client environment and the pods and services deployed by the CloudXPRT workloads, all of which are the same for both environments.

CloudXPRT can run the client test harness on or off a worker node in the test environment. We choose to run the harness on one auxiliary VM, which ran on another server. For the data analytics workload, we modified the configuration to accept HTTP traffic from outside the pod network. For the web microservices workload, we used the built-in remote-client mode.

The data analytics workload uses several small persistent-storage volumes (2 GB each) to store data for its Kafka and Zookeeper pods. These volumes use the MinIO abstraction and are backed on local storage – viz., the main RAID 1 volume for the bare-metal server, and the main datastore for the VM nodes. Each "XGBoost" analytics pod, which the benchmark creates during the run, acquires one 2GiB persistent volume.

Similarly, the web microservices workload uses three small persistent volumes (1GB each) for each of its Cassandra pods. These volumes are backed by the same storage as for data analytics.

The data analytics workload creates pods with the following resource limits:

Pod	CPU (m)	Memory (MiB)	Number	Role
kafka	100	512	1	Application environment
zookeeper	10	100	1	Application environment
XGBoost	N/A	N/A	Variable	Application

The web microservices workload creates pods with the following resource limits:

Pod	CPU (m)	Memory (MiB)	Number	Role
cassandra	200	N/A	3	Application environment
crypt service	800	50	1	Application environment
db service	200	10	1	Application environment
redis crypt service	200	500	1	Application environment
redis service	100	100	1	Application environment
redis user service	100	10	1	Application environment
user service	200	10	1	Application environment
web service	500	50	1	Application environment
mc service	4,000	20	Variable	Application

For both workloads, the client harness communicates with the application via API requests via HTTP to the external IP address of the service's NodePort.

Preparing an auxiliary VM for TKG

1. Install Ubuntu 18.04 LTS on an auxiliary VM running on a secondary server in the vSphere environment, and created a local user account on it. We named ours `ubuntu`.
2. Log onto the VM as the local user.
3. Install Docker 19:

```
sudo apt-get remove docker docker-engine docker.io containerd runc
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates curl gnupg lsb-release
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
  sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] \
  https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

4. Add the local user to the Docker group:

```
sudo usermod -a -G docker ubuntu
```

5. Create SSH keys:

```
ssh-keygen -t rsa -b 4096 -C ubuntu@ubu01
```

Installing Tanzu Kubernetes Grid version 1.2

We installed TKG following the installation steps in the VMware documentation: https://docs.vmware.com/en/VMware-Tanzu-Kubernetes-Grid/1.2/vmware-tanzu-kubernetes-grid-12/GUID-install-tkg.html.

We installed ESXi 7 on the server under test and added it to an existing VMware vSphere® environment under the control of a VMware vCenter® appliance. The vSphere cluster uses one distributed switch and physical NICS connected to a TOR switch running DHCP.

1. Download the packages for TKG CLI version 1.2.0 (tkg-linux-amd64-v1.2.0-vmware.1.tar.gz), TKG kubectl version1.19.1 (kubectl-linux-v1.19.1-vmware.2.gz), and an OS image (photon-3-kube-v1.19.1-vmware.2.ova) from the VMware website: https://www.vmware.com/go/get-tkg.
2. Install the TKG CLI and TKG kubectl on the VM:

```
tar -xf tkg-linux-amd64-v1.2.0-vmware.1.tar.g
sudo mv tkg-linux-amd64-v1.2.0-vmware.1.tar/tkg/tkg-linux-amd64-v1.2.0+vmware.1 \
  /usr/local/bin/tkg
gunzip kubectl-linux-v1.19.1-vmware.2.gz
sudo mv kubectl-linux-v1.19.1-vmware.2.gz /usr/local/kubectl
sudo chmod a+rx /usr/local/bin/{tkg,kubectl}
```

3. Initialize the TKG environment for the local user:

```
tkg get mc
```

4. To deploy an OVF template to the vSphere host and system under test, in the vCenter client, use photon-3-kube-v1.19.1-vmware.2.ova as the local file. Use the distributed switch for the network, and the SSD-backed datastore as storage.
5. When the deployment finishes, convert the VM to a template.
6. Insert the following lines to the end of the `~/.tkg/config.yaml` file:

```
VSPHERE_CONTROL_PLANE_DISK_GIB: "20"
VSPHERE_CONTROL_PLANE_MEM_MIB: "2048"
VSPHERE_CONTROL_PLANE_NUM_CPUS: "2"
VSPHERE_WORKER_DISK_GIB: "20"
VSPHERE_WORKER_MEM_MIB: "2048"
VSPHERE_WORKER_NUM_CPUS: "2"
```

7. Insert the following lines to the end of the ~/.tkg/config.yaml file. Note: Replace the parameters to the right of the colons with values appropriate for your vSphere environment. You must create the vSphere folder, datastore, resource pool, and data center separately:

```
VSPHERE_SERVER: 10.214.100.100
VSPHERE_USERNAME: administrator@vsphere.local
VSPHERE_PASSWORD: MyPasswordABC@
VSPHERE_DATACENTER: /Datacenter
VSPHERE_DATASTORE: /Datacenter/datastore/data
VSPHERE_FOLDER: /Datacenter/vm/tkg-vms
VSPHERE_NETWORK: DPortGroup
VSPHERE_RESOURCE_POOL: /Datacenter/host/k8s/Resources/MCpool
```

8. Insert the following line to the end of the ~/.tkg/config.yaml file. Note: Replace the parameter to the right of the colon with the contents to the right of the colon from the public SSH key you created in step 5.

```
VSPHERE_SSH_AUTHORIZED_KEY: ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQC5k3G/IHrsJ9/gyKfVsn+4L6ehOHZna7EV7h9
…
…
…
BVj4+APEpr2ULtpVE87VfeAk/d5MVNdH3DAV5HrxrEqdgcpQQo+3hZLz4KKD4wDQNrIRPdOXQqfyNJHO6nQOBaPC7voL+QuyQ==
ubuntu@ubu01
```

9. Create the TKG management cluster named mgmt. Note: Replace the endpoint IP address with one in the same network as your vSphere network:

```
tkg init -i vsphere --vsphere-controlplane-endpoint-ip 10.214.156.165 -p dev \
  --ceip-participation false --cni antrea -v 6 --name mgmt
```

Creating the Kubernetes clusters with Tanzu Kubernetes Grid

1. Create and upload an Ubuntu 18.04 OVA to use for the worker VMs according to the instructions at the following link: https://docs.vmware.com/en/VMware-Tanzu-Kubernetes-Grid/1.2/vmware-tanzu-kubernetes-grid-12/GUID-build-images-capv-images.html.
2. Create a base manifest for the cluster, and change the worker nodes' memory to 10 GB:

```
tkg config cluster my-cluster --plan dev --vsphere-controlplane-endpoint-ip \
  10.214.46.48 --kubernetes-version v1.18.8+vmware.1 | awk \
  awk '/memoryMiB:/{c++;if(c==2){sub("memoryMiB: 2048","memoryMiB: 10240")}}1' >\
   Base.yaml
```

3. Create a cluster manifest from the base manifest, and set the desired number of worker VMs and number of vCPUs per VM. For example, to create four worker nodes with 20 vCPUs, run the following commands:

```
awk '/replicas:/{c++;if(c==3){sub("replicas: 1","replicas: 4")}}1' Base.yaml |\
  awk '/numCPUs:/{c++;if(c==2){sub("numCPUs: 2","numCPUs: 4")}}1' > cluster_04x02.yaml
```

4. Create the Kubernetes cluster from the manifest (e.g., `cluster_04x02.yaml`):

```
tkg create cluster my-cluster --manifest ~/ cluster_04x02.yaml -v 15
```

5. Add the cluster-access credentials to the Kubernetes configuration, and set the Kubernetes context to that of the cluster:

```
tkg get credentials my-cluster
kubectl config use-context my-cluster-admin@my-cluster
```

Installing open-source Kubernetes v1.18 on the bare-metal server for CloudXPRT v1.01

1. From the CloudXPRT GitHub repository (https://github.com/BenchmarkXPRT/CloudXPRT), download the CloudXPRT Data Analytics packages to the auxiliary client system:

```
wget https://github.com/BenchmarkXPRT/CloudXPRT/raw/master/Archived_versions/CloudXPRT_v1.01_data-analytics.tar.gz
```

2. Extract the benchmark, and change directories to its installation directory:

```
tar -xf CloudXPRT_v1.01_data-analytics.tar.gz
cd CloudXPRT_v1.01_data-analytics/installation
```

3. In the cluster_config.json file navigate to the "nodes" stanza.

4. Replace the empty parameter "" with the bare-metal server's IP address in double quotation marks in the line containing "ip_address": "" For example:

```
"nodes": [
        {
            "ip_address": "10.214.2.135",
            "hostname": ""
        }
    ],
]
```

Note: Do not change the "hostname" line.

5. Replace the version of kubespray that comes with CloudXPRT with version 2.14 from the Kubernetes release site:

```
wget https://github.com/kubernetes-sigs/kubespray/archive/v2.14.2.tar.gz
tar -xf kubespray-2.14.2.tar.gz
mv kubespray kubespray-orig
mv kubespray-2.14.2.tar.gz kubespray
```

6. Run the `prepare-cluster.sh` and `create-cluster.sh` scripts:

```
sudo ./prepare-cluster.sh
sudo ./create-cluster.sh
```

Installing and running the CloudXPRT Data Analytics v1.01 workload

1. From the CloudXPRT GitHub repository (https://github.com/BenchmarkXPRT/CloudXPRT), download the CloudXPRT Data Analytics packages to the auxiliary client system:

```
wget "https://github.com/BenchmarkXPRT/CloudXPRT/raw/master/Archived_versions/CloudXPRT_v1.01_data-analytics.tar.gz
```

2. Extract the benchmark, and change directories to its cnbrun directory:

```
tar -xf CloudXPRT_v1.01_data-analytics.tar.gz
cd CloudXPRT_v1.01_data-analytics/setup
```

3. For the bare-metal system only, run the set-up script, and skip to step 5:

```
sudo ./cnb-analytics_setup.sh
```

4. For the TKG cluster system only, perform the following modifications before running its set-up scripts:
   a. Get the IP address of one of the worker nodes. Replace all occurrences of the string `IPADDRESS` in this section with one of that IP address.

```
kubectl get nodes --selector='!node-role.kubernetes.io/master' -o \
  jsonpath='{.items[*].status.addresses[?(@.type=="ExternalIP")].address}'
```

   b. Create the file cnb-analytics_tkg_setup.sh with contents we specify in the Files we used in our testing section of this document..
   c. Make the script executable.

```
chmod a+rx cnb-analytics_setup.sh
```

   d. Update the kafka set-up script.

```
sed -i 's/\(# Deploy kafka\)/\1\nsleep 60/' setup_kafka/setup.sh
```

   e. Create templates for kafka and zookeeper storage.

```
head -n -1 setup_kafka/storage/kafka-storage.yml > \
  setup_kafka/storage/kafka-storage.txt
head -n -1 setup_kafka/storage/zookeeper-storage.yml > \
  setup_kafka/storage/zookeeper-storage.txt
```

   f. Increase deployment timeouts for the zookeeper pods.

```
sed -i 's/\(imok.*\)/\1\n          initialDelaySeconds: 30\n          periodSeconds: 15\n
timeoutSeconds: 300/' \
  setup_kafka/zookeeper/50_pzoo.yml
```

g. Update the base repository for helm.

```
sed -i 's/\(helm init --service-account=tiller --history-max 300\)/\1 --stable-repo-url https://
charts.helm.sh/stable/' \
  setup_prometheus/setup.sh
```

h. Create the script setup_ssh/setup.sh. Replace IPADDRESS with the one selected in step a.

```
mkdir setup_ssh
cat > setup_ssh/setup.sh <<EOF
#!/bin/bash
NODE=capv@IPADDRESS
scp ../setup_minio/createbucketAddcontent.py ../setup_minio/deletebucket.py \
   ../../cnbrun/monitoring $NODE:
EOF
chmod a+rx setup_ssh/setup.sh
```

i. Update the minio scripts. Replace all occurrences of `IPADDRESS` with the IP address selected in step a.

```
sed -i s/import ResponseError/import InvalidResponseError' setup_minio/*.py
sed -i -e 'sZ^\(./deletebucket.py\)ZNODE=capv@IPADDRESS\nssh $NODE \1Z' \
  -e 'sZ^\(rm -R /mnt/minio\)Z#\1Z' setup_minio/cleanup.sh
sed -i -e 's/^\(apt update\)/#\1\nNODE=IPADDRESS/' \
 -e 's/^\(apt install python3.6\)/#\1\nssh $NODE sudo apt -y install python3-pip\nssh $NODE pip3
install minio/' \
  -e 'sZ^\( mkdir -p /mnt/minio\)Z/#\1Z' \
  -e 'sZ\(chmod -R 755 data/\*\)Z\1\nscp -r data $NODE:\nZ' \
  ./setup_minio/setup.sh
```

j. Run the set-up scripts.

```
sudo ./cnb-analytics_tkg_setup.sh
sudo ./cnb-analytics_setup.sh
```

5. Change directories to the cnbrun directory:

```
cd ../cnbrun
```

6. Modify the vCPU_per_POD and Lambda parameters in the cnb-analytics_run-automated.sh test harness script. For example:

```
vCPU_per_POD=(13 20 26 39 52)
Lambda=(10 30 50)
```

7. Perform the test runs. For example, using the parameters in step 6, 15 configurations will run.

```
sudo ./cnbrun
```

8. Collect the runs' metrics and statistics into a CSV table using the cnb-analytics_parse-all-results.sh script. Descriptions of the columns are in README.md. Per CloudXPRT guidelines, the key metrics are 95th percentile latency and throughput.

```
sudo ./cnb-analytics_parse-all-results.sh
```

Installing and running the CloudXPRT Web Microservices v1.01 workload

1. From the CloudXPRT GitHub repository https://github.com/BenchmarkXPRT/CloudXPRT) download the CloudXPRT Web Microservices packages to the auxiliary client system:

```
wget https://github.com/BenchmarkXPRT/CloudXPRT/raw/master/Archived_versions/ CloudXPRT_v1.00_web-
microservices-Onprem.tar.gz
```

2. Extract the benchmark, and change directories to its cnbrun directory:

```
tar -xf CloudXPRT_v1.00_web-microservices-Onprem.tar.gz
cd CloudXPRT_v1.01_web-microservices/cnbrun
```

3. For the TKG cluster only, perform the following modifications before running the set-up script:

```
mv mc.sh mc.sh-orig
sed 's/--selector='!node-role.kubernetes.io/master'//' mc.remote.sh > mc.sh
sed -i 's/kubectl label nodes/# kubectl label nodes/' mc.sh
```

4. Perform the test run:

```
sudo ./cnbrun
```

5. To view the run's output files, navigate to the output directory. The last line in the autoloader_mc_YYYYMMDD_XXXXXX.log file summarizes the run's metrics. For example:

```
Best throughput found at 1274 requests with 95th percentile latency of 2669 ms
```

Preparing the environments for the pod-density study

Preparing the Tanzu Kubernetes Grid environment

For the Tanzu Kubernetes Grid environment only, we deleted the original cluster and created a new one with one smaller VM of 2 vCPUs and 8 GB memory. From this base cluster of 1 VM, we will scale it to 25, 30, 35, 40 and 45 VMs, and conduct the pod-density study on each until the VMware or Kubernetes environments can no longer function properly due to higher CPU usage on the server or VM, or lack of memory, the application functions with too high a latency (over 1 second), or the Kubernetes environment becomes unresponsive.

1. On the auxiliary VM, we create the base cluster by modifying the worker-cluster configuration template so that it creates one 2-vCPU, 8-GB VM by following steps 3 through 5 in section Creating the Kubernetes clusters with Tanzu Kubernetes Grid
2. To scale the number of VMs in the cluster to 25, 30, 35, or 40 VMs, we run the following command on the auxiliary VM, replacing the .final argument with the desired number of VMs

```
tkg scale cluster my-cluster -w 25
```

3. We can monitor the worker VM creation from the vCenter console.

Preparing bare-metal environment for the pod-density study

On the bare-metal server, we reconfigured the Kubernetes daemon and OS to allow us to run more than 110 pods in total.

1. Reconfigure the kubelet daemon so that up to 1,500 pods may run on the node. Add --max_pods 1500 to the end of the list of kubelet arguments in the file /etc/kubernetes/kubelet.env. For example,

```
KUBELET_ARGS="--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf \
--config=/etc/kubernetes/kubelet-config.yaml \
--kubeconfig=/etc/kubernetes/kubelet.conf \
--pod-infra-container-image=k8s.gcr.io/pause:3.2 \
--runtime-cgroups=/systemd/system.slice \
--max-pods 1500 \
"
```

2. Restart the kublet daemon:

```
sudo systemctl stop kubelet
sudo systemctl start kubelet
```

3. Increase the effect size of the system's ARP table:

```
cat > sysctl-00.conf <<EOF
net.ipv4.neigh.default.gc_thresh1 = 80000
net.ipv4.neigh.default.gc_thresh2 = 90000
net.ipv4.neigh.default.gc_thresh3 = 100000
EOF
sudo sysctl -p ./ sysctl-00.conf
```

Running the pod-density study

1. Deploy the application, which is based on this example: https://github.com/paulbouwer/hello-kubernetes. The specification file, hello-kubernetes.yaml, is in the Files we used in our testing section of this document.

```
kubectl apply -f hello-kubernetes.yaml
```

2. Set the number of pods by scaling the replicas. To limit the stress on cluster resources, we added replicas in batches of 50 until we reached the final number. For example, to scale to 150 pods, we added pods in three steps:

```
kubectl  scale  deployment hello-kubernetes  --replicas=50
# wait until all pods have started
kubectl  scale  deployment hello-kubernetes  --replicas=100
# wait until all pods have started
kubectl  scale  deployment hello-kubernetes  --replicas=150
# wait until all pods have started
```

3. Get the external IP address for the service:

```
kubectl get svc hello-kubernetes    -o jsonpath="{.spec.clusterIP}"
```

4. Perform the test by executing this script that sends an unending set of queries to the load balancer's IP address, which you found in step 3. The results file contains the name of the pod that received and responded to the query and the total time needed to complete the request.
5. To end the test, type Control+C. For example, if the IP address was 10.233.59.17, the script would be:

```
while : ; do
  curl -s --connect-timeout 10 http://10.233.59.17 -w \
  '\n%{time_total} hello-kubernetes- 0000\n'
done | grep hello-kubernetes- | tee pod-density-results.txt
```

Files we used in our testing

cnb-analytics_tkg_setup.sh

```
#!/bin/bash
#================================================================================
# Copyright 2021 BenchmarkXPRT Development Community
#################################################################
# This script runs the setup for CNB-analytics multi-node on TKG
#################################################################

INSTALLATION_DIRECTORY=$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)

if [ ! -s "$HOME/.ssh/id_rsa" ]; then
  echo "the ~/.ssh/id_rsa file does not exist or is empty."
  exit;
fi

# add nodes
kafka_storage_file=$INSTALLATION_DIRECTORY/setup_kafka/storage/kafka-storage.yml
zookeeper_storage_file=$INSTALLATION_DIRECTORY/setup_kafka/storage/zookeeper-storage.yml

cp ${kafka_storage_file%.yml}.txt ${kafka_storage_file}
cp ${zookeeper_storage_file%.yml}.txt ${zookeeper_storage_file}

working_nodes=$(kubectl get nodes --selector='!node-role.kubernetes.io/master' \
  --template '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}')
for node_name in $working_nodes; do
  node_ip=$(kubectl get nodes --field-selector metadata.name=$node_name -o wide | \
  awk /$node_name/' {print $6}')
  echo "$node_name   $node_ip"
  echo "          - $node_name" >> $kafka_storage_file
  echo "          - $node_name" >> $zookeeper_storage_file
  ssh capv@${node_ip} sudo mkdir -p /mnt/cnb-pv/kafka /mnt/cnb-pv/zookeeper
done

cd $INSTALLATION_DIRECTORY/setup_ssh
./setup.sh
```

hello-kubernetes.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: hello-kubernetes
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: hello-kubernetes
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-kubernetes
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-kubernetes
  template:
    metadata:
      labels:
        app: hello-kubernetes
    spec:
      containers:
      - name: hello-kubernetes
        image: paulbouwer/hello-kubernetes:1.9
        ports:
        - containerPort: 8080
```

Read the report at http://facts.pt/Jfa6Txr ▶

This project was commissioned by VMware.