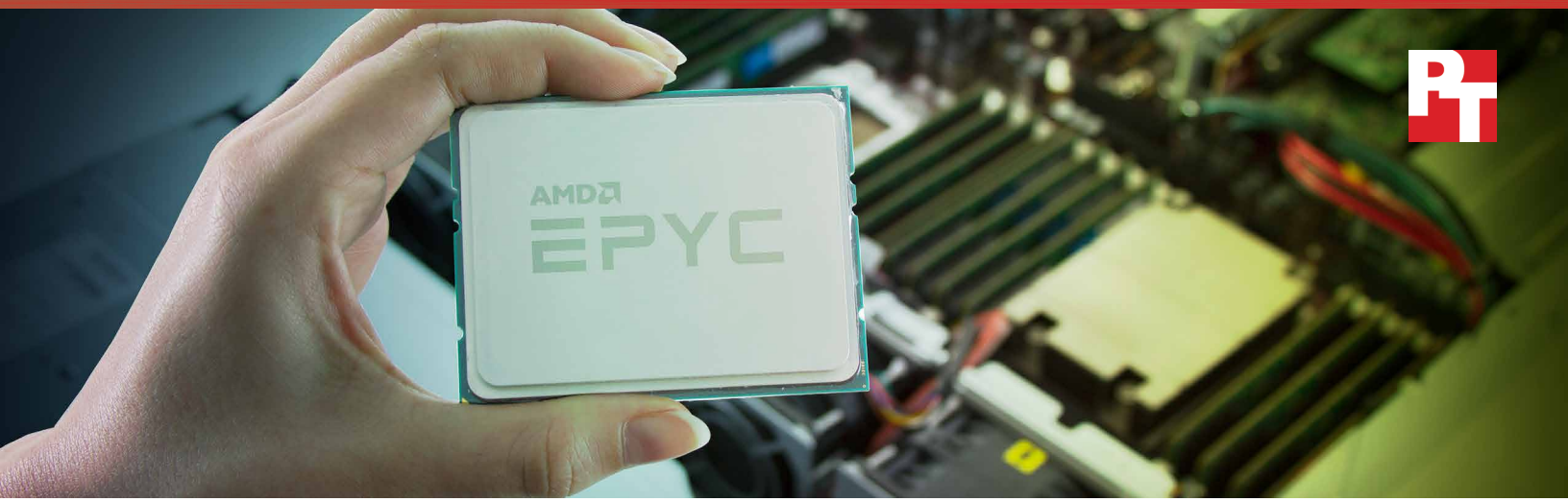


VM density on a Microsoft Storage Spaces Direct solution powered by the AMD EPYC 7601 processor

Businesses seeking solutions for software-defined storage will soon have even more options to choose from in terms of server configuration. Earlier this year, AMD unveiled a new line of processors: the AMD EPYC™ series. Built on an architecture called "Zen," the EPYC series of processors aims to provide datacenters with a high core density.

In the Principled Technologies datacenter, we deployed Microsoft® Storage Spaces Direct to a three-node cluster powered by the AMD EPYC 7601 processor. We then measured the cluster's performance with VMFleet, a tool from Microsoft that generates storage-intensive loads for Microsoft Windows Server® 2016.

The cluster, which used a Mellanox® end-to-end networking solution, hosted 192 virtual machines, each matching the 1,100 IOPS limit of the P15 Premium Managed Disk offering from Microsoft Azure®. The cluster managed this while using (on average) 23.22 percent of the host processor, and while also keeping average read latency at 0.39 milliseconds. Low CPU utilization leaves headroom for other tasks, and the sub-millisecond read latency means applications can quickly receive data from storage. A business that takes advantage of the solution's density could also save on the cost of power, cooling, and management by reducing its datacenter footprint.

This proof-of-concept study provides detailed information on our setup and our testing methods. We report benchmarking data and also provide detailed information on the Zen architecture.

Supported
192 VMs
matching the IOPS requirement for the Microsoft Azure P15 Premium Managed Disk offering

23.22%
average host CPU utilization leaves headroom for other workloads

0.39ms
average read latency ensures fast data delivery

How software-defined storage addresses challenges in traditional, array-based architecture

Traditionally, enterprise and datacenter server environment architectures have kept the functions of compute and storage separate. Persistent storage and its associated requirements (such as resiliency, high performance, deduplication, and tiering) existed separately from the compute layer. However, the traditional architecture has brought challenges. Storage teams and infrastructure teams often exist in isolated silos, and support contracts often vary by storage or compute vendor. Traditional storage often requires multiple administrative consoles, complicating the management process. It can also have barriers of entry in terms of cost, and often require professional services for operational needs such as scaling out.

Software-defined storage (SDS) solutions may address some of these challenges. In an SDS solution, storage devices are locally attached to servers and presented to software[1]. The software then handles features such as resiliency levels, deduplication options, tiering, and write caching. Administrators may present storage pools to users and applications as needed.

In this proof-of-concept study, we present the configuration of an SDS solution that uses Microsoft Storage Spaces Direct with servers powered by the AMD EPYC processor. We also demonstrate the solution's I/O performance using DISKSPD via VMFleet.

Our configuration

Hardware

We installed Microsoft Storage Spaces Direct onto a cluster comprised of three dual-socket, 2U whitebox servers powered by the AMD EPYC 7601 processors. We used a top-of-rack Mellanox® SN2700 Open Ethernet 100GbE networking switch as well as Mellanox ConnectX®-4 Lx 25GbE dual-port adapters. We used 100GbE-to-4x LinkX 25GbE breakout cables to connect the adapters to the switch.

Each server contained eight Samsung PM863 SATA SSDs (configured as capacity drives) and two Micron® 9100 Pro NVMe SSDs (configured as cache drives). Because we used the AMD SOC SATA controller, this configuration did not require an additional storage controller. We used an additional dual-socket, 2U server to host our Active Directory® and Storage Spaces Direct management VMs. We configured each cluster node with 512 GB of DDR4 memory.

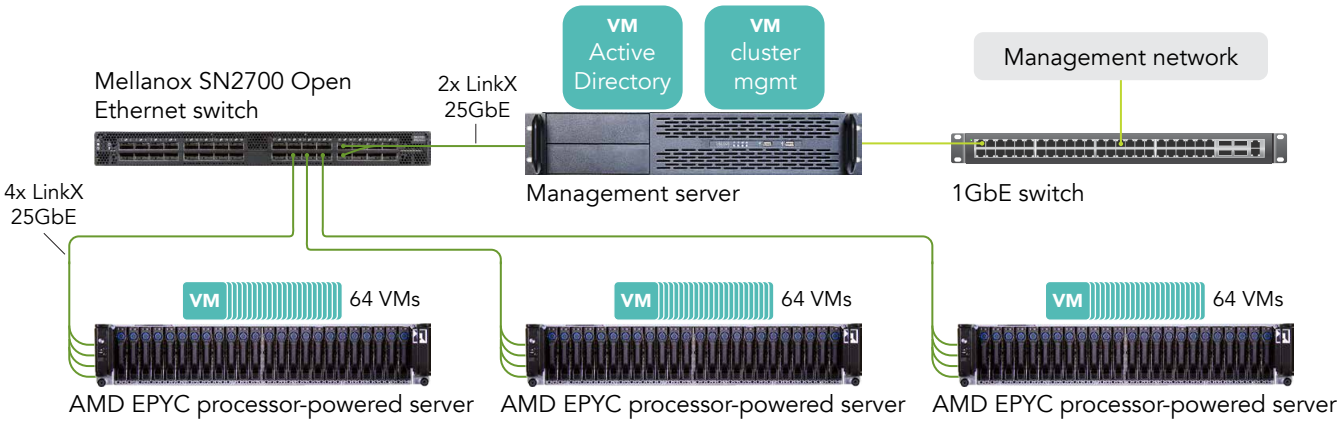
Before testing, we installed the latest BIOS, drivers, and firmware for each component. We also set the BIOS power settings to Performance.

About the Mellanox SN2700 Open Ethernet switch

The Mellanox SN2700 100GbE switch we used is a Spectrum-based, 32-port, ONIE (Open Network Install Environment)-based platform on which you can mount a variety of operating systems. According to Mellanox, the switch lets you use 25, 40, 50 and 100GbE in large scale without changing power infrastructure facilities.

Learn more at http://www.mellanox.com/related-docs/prod_eth_switches/PB_SN2700.pdf

The figure below shows the physical layout of our Storage Spaces Direct cluster.



About Samsung PM863a SSDs

Samsung PM863a 2.5-inch SATA SSDs leverage the company's 3D (V-NAND) technology to provide storage capacities of up to 3.84 TB without increasing the physical footprint of the drive.

Learn more at http://www.samsung.com/semiconductor/minisite/ssd/product/enterprise/pm863a.html

Software

Operating system and prerequisites

Storage Spaces Direct requires Windows Server 2016; we chose to use the Datacenter edition for our testing. The VMFleet workload uses VMs with the Core edition of Windows Server 2016. We applied the latest OS updates via Windows Update, and set the OS power management policy to High Performance.

Assembling the solution

We installed Windows Server 2016 Datacenter onto each cluster node and onto our infrastructure VM host. After configuring Active Directory and adding each cluster node to the domain, we added the following server roles to each cluster node:

- Failover Clustering
- Hyper-V
- Data-Center-Bridging
- RSAT-Clustering-PowerShell
- Hyper-V-PowerShell

We then configured the network for RoCEv2 RDMA functionality by creating a network Quality of Service (QoS) policy for SMB-Direct. We enabled Flow Control for SMB and disabled it for other traffic. We then applied the QoS policy to our network adapters, set a VLAN, and enabled RDMA.

We then ran the PowerShell cmdlet Test-Cluster, which assesses the hardware and software installed on each node and produces a report with failure, warning, or success notifications for each validation test. When Test-Cluster completes without failures, the Storage Spaces Direct cluster is considered ready for deployment. We made sure that the Test-Cluster cmdlet reported only success on all validation tests before deploying the cluster.

To create the cluster, we ran the PowerShell cmdlet New-Cluster with the NoStorage parameter. The NoStorage parameter allowed us to choose the storage devices we wanted after cluster creation. Once the cluster was created, we used the disk cleaning PowerShell script provided by Microsoft (https://docs.microsoft.com/en-us/windows-server/storage/storage-spaces/hyper-converged-solution-using-storage-spaces-direct) to prepare our drives for cluster use. Once the drives were cleaned and showed True for the property CanPool, we added them to our cluster with the Enable-ClusterStorageSpacesDirect cmdlet.

To take full advantage of the high-performance NVMe PCIe drives, we set the Storage Spaces Direct cache to handle reads and writes.

We created volumes with the New-Volume cmdlet, and sized each volume at 2.27 terabytes with three-way mirroring resiliency. We also created a 10GB volume to store VMFleet PowerShell scripts and resources.

We chose the size and count of the VMs to fully provision CPU threads on the host. We used VMFleet to clone 192 VMs from the base Windows Server 2016 Core Edition virtual hard disk. The VMFleet VMs contained two virtual CPUs, 1792MB of DDR4 memory, and a 32GB fixed-size virtual hard disk. Before running the workload, we set a Storage QoS policy using Windows PowerShell and VMFleet tools. We set the IOPS limit of each VM to 1,100 IOPS to match the IOPS limit on the Azure P15 Premium Managed Disk offering.

The workloads we used to test our setup

Basic overview

DISKSPD is a Microsoft tool for measuring storage performance, available via GitHub. Microsoft includes DISKSPD in a set of scripts called "VMFleet," which enables hyperconverged guests in Windows Server 2016 Storage Spaces Direct to run the tool. VMFleet also helps with creating and monitoring VMs, as well as aggregating the resulting data. Using VMFleet allows one to better understand how a Storage Spaces Direct cluster will perform under varying I/O load patterns.[2,3]

Running the workload

We created 64 VMs per node and targeted the Azure P15 Premium Managed Disk service level by setting a Storage QoS policy equal to the Azure P15 limit of 1,100 IOPS.

Before running each test, we ensured each Cluster Shared Volume (CSV) was mounted to the correct server node. For example, the first cluster node owned VMs 1 through 64 as well as the volume (Volume1) that housed them. We also made sure that all CSVs displayed a Healthy status in Failover Cluster Manager, and that no storage jobs were running. This was to ensure that the disks were not undergoing background repair tasks during our test runs.

We chose a workload typical of a web hosting environment that serves large numbers of page views from a static database. We chose a small-block, random IO profile with 90 percent reads and 10 percent writes. We used the following parameters:

- Block size = 4K
- Write percentage = 10
- IO = random
- Threads = 2
- Outstanding IO = 16
- Warm-up = 10 minutes
- Test duration = 20 minutes

When a test run finishes, VMFleet places results into an XML file for each VM. We created a PowerShell script to parse these results and extract the average host CPU utilization, the average CSV latency, and the sum of IOPS across all VMs. The table below shows these results.

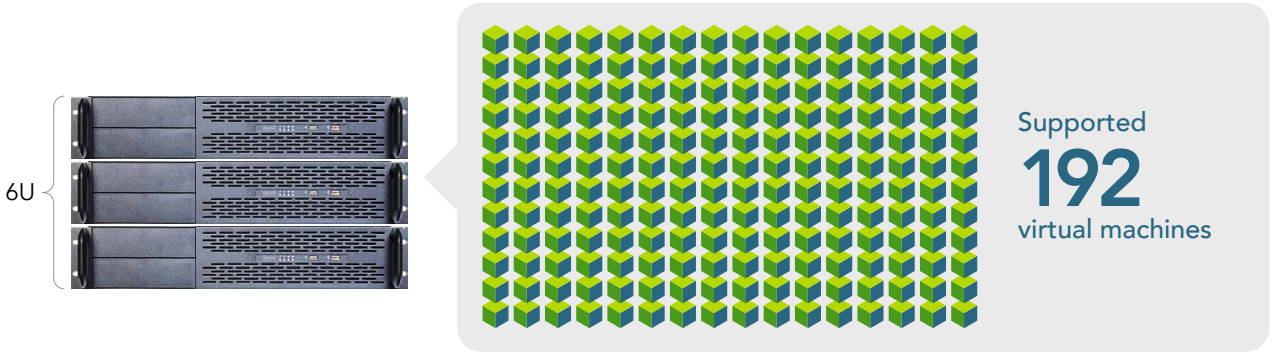
Storage Spaces Direct on an AMD EPYC 7601 processor-powered server cluster	
Total cluster IOPS	213,775
Average IOPS per VM	1,113
Average host CPU utilization (%)	23.22
Average read latency (ms)	0.39
Average write latency (ms)	2.23

The AMD EPYC processor-powered servers maintained the target IOPS with an average latency of 0.39 milliseconds and an average host CPU utilization of 23.22 percent. This means you could run a total of 192 VMs at the Azure P15 service level and still have compute headroom to add VMs or additional workloads to the Storage Spaces Direct cluster.

Benefits of a dense configuration

The servers we used required only 6U of rack space, but hosted 192 VMs at the Microsoft Azure P15 service level. This makes for a density of 32 VMs per 1U of rack space.

This density has several benefits. Assuming a constant application count, more VMs per server means a business could reduce the size of their fleet, making rack space available for future growth. Denser systems also require less power, fewer power outlets, fewer management licenses, and less administrator time commitment.



AMD EPYC architecture overview

AMD EPYC processors have up to up to 32 cores, 64 threads, and can run with 2 TB of DDR4 memory capacity spread across eight channels[4]. This high core density contributed to the dense VM count on our Microsoft Storage Spaces Direct cluster.

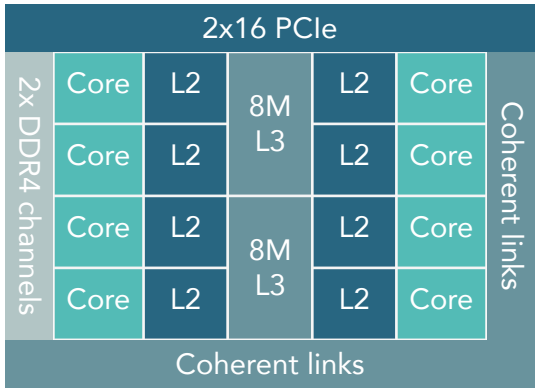
To better understand this high core density, let's look at the components that go into it. Note that the information in this section comes from publicly available materials from the AMD website[5].

Building blocks of the EPYC 7601

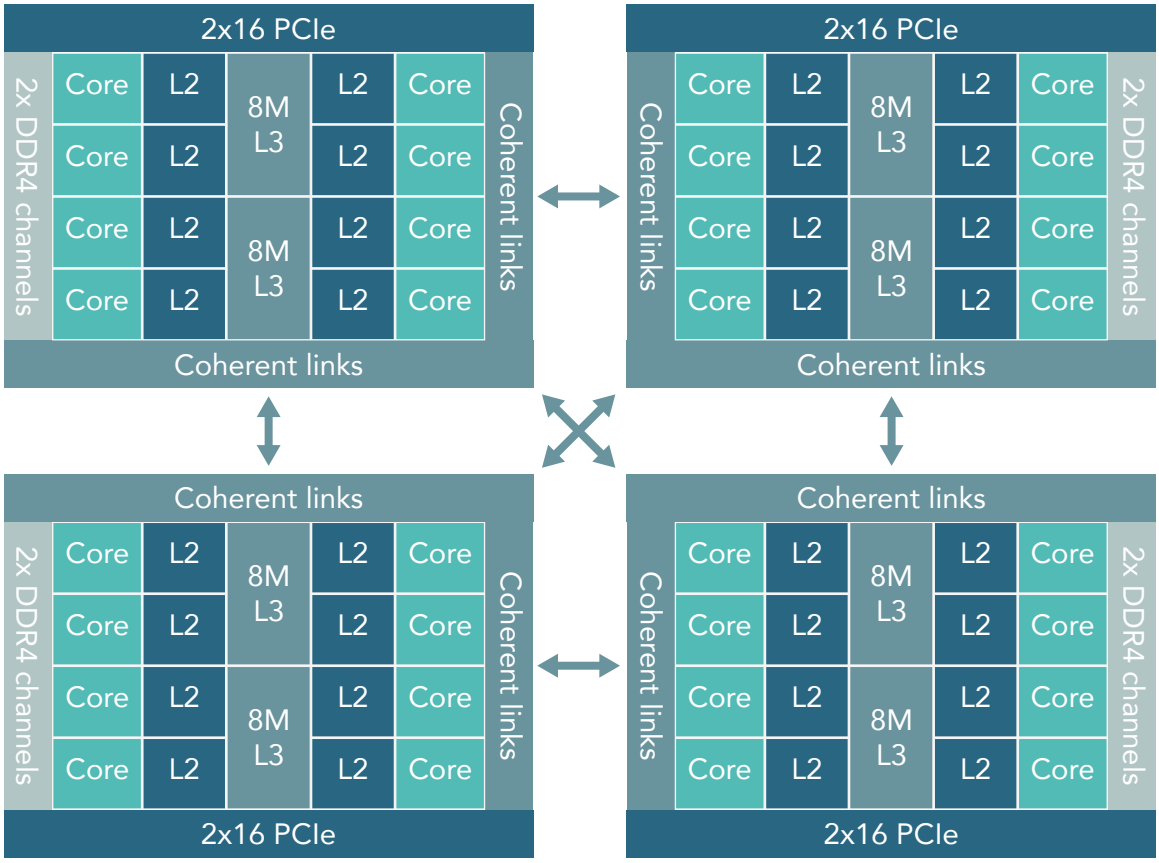
The primary building block is the Zen core[6]. Each Zen core has a 64KB L1 instruction cache and a 32KB L2 data cache, as well as access to a 512KB instruction and data (I+D) L2 cache and a shared L3 cache.

Four Zen cores, along with 8 MB of L3 caching, make up a single CPU Complex (CCX).

A "Zeppelin" die comprises two CCXs, meaning that each Zeppelin die has eight Zen cores.



On an EPYC 7000-series processor, four Zeppelin dies connect to each other with AMD Infinity Fabric (IF) interconnect to form the AMD EPYC processor. As we show below, one EPYC processor comprises eight CCXes, 32 cores, and 64 threads.

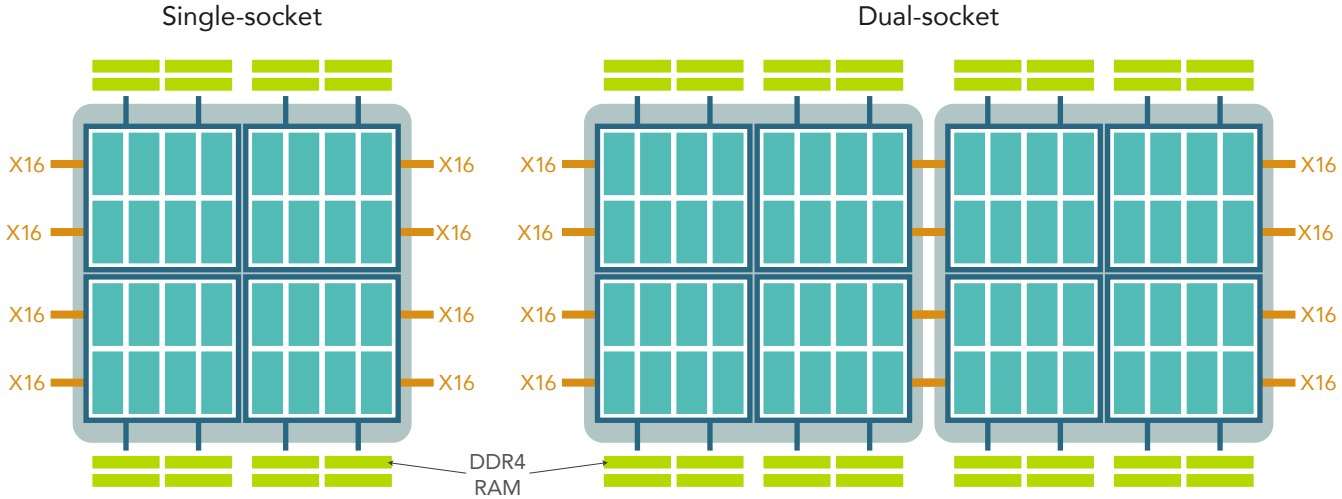


Single-socket

In a single-socket configuration, the EPYC chip has eight DDR4 channels and 128 PCIe lanes.

Dual-socket

In a dual-socket configuration, half of the PCIe lanes are for the Infinity Fabric between both sockets. This configuration has 16 DDR4 channels and 128 PCIe lanes.



Performance and power management

EPYC chips have several features for performance and power management. Performance Determinism sets consistent performance across all CPUs, though the power draw of each processor may vary slightly with manufacturing variance. Power Determinism sets consistent power usage across all processors, though the performance of each processor may also vary. Being able to manage power usage can be useful in datacenters where it is necessary to control the power draw of servers.

Security

Though we did not test them in our datacenter, the EPYC family of processors contains several security features:

- AMD Secure Root-of-Trust technology prevents booting any software that is not cryptographically signed.
- AMD Secure Run Technology encrypts software and data in memory.
- AMD Secure Move technology lets you securely migrate virtual machines and containers between EPYC processor-based servers.

The EPYC family supports encrypted memory features:

- You can use transparent mode to encrypt all memory pages. You can enable transparent mode in the BIOS without having to change the OS or individual applications.
- In Secure Encrypted Virtualization (SEV) mode, you can encrypt specific virtual machines without having to change individual applications.

Memory capacity

Each AMD EPYC processor supports 2 TB of memory, equivalent to 4 TB per two-socket server. The closest competitor supports 1.5 TB of memory per processor, equivalent to 3 TB per two-socket server. This means that supporting 12 TB of memory would require three AMD EPYC processor-powered two-socket servers and four of the competitor's two-socket servers.

Conclusion

A software-defined storage solution brings architectural changes to the traditional array-based datacenter—many of these changes stand to benefit a business. This proof of concept has introduced a new solution for software-defined storage: servers powered by the AMD EPYC series of processors. Based on the new Zen architecture, the EPYC series of processor delivers on-chip resources such as 32 cores (64 threads), up to 4 TB of DDR4 memory in a dual-socket platform, 128 PCIe lanes (with the option to use up to 32 as SATA or NVMe ports), and on-chip SATA controllers.

At Principled Technologies, we deployed Microsoft Storage Spaces Direct to a whitebox cluster of servers powered by the AMD EPYC 7601 processor, and measured its performance with the benchmarking tool VMFleet. In our tests, the server cluster delivered up to 192 VMs with the performance of the Azure P15 Premium Managed Disk, while keeping average host CPU utilization under 24 percent and average read latency under 0.4 milliseconds.

---

1 "What Does the Data Storage Industry Have in Store for Startups?" accessed November 22, 2017, http://marketrealist.com/2015/09/enterprise-storage-moves-software-defined-storage/

2 "DISKSPD now on GitHub, and the mysterious VMFLEET released," by Ned Pyle, accessed November 22, 2017, https://blogs.technet.microsoft.com/filecab/2016/04/11/diskspd-now-on-github-and-the-mysterious-vmfleet-released/

3 "DiskSpd," accessed November 22, 2017, https://github.com/Microsoft/diskspd

4 Core count per CCX decreases for lower-bin processors in the EPYC line.

5 "AMD EPYC,"accessed November 22, 2017, http://www.amd.com/en/products/epyc

6 "The 'Zen' Core Architecture," accessed November 22, 2017, http://www.amd.com/en/technologies/zen-core

On October 9, 2017, we finalized the hardware and software configurations we tested. Updates for current and recently released hardware and software appear often, so unavoidably these configurations may not represent the latest versions available when this report appears. For older systems, we chose configurations representative of typical purchases of those systems. We concluded hands-on testing on October 13, 2017.

Appendix A: System configuration information

Server configuration information	
BIOS name and version	RSW1002E 9/5/2017
Non-default BIOS settings	Performance mode enabled
Operating system name and version/build number	Windows Server 2016 Datacenter Edition Version 1607 (OS Build 14393.1715)
Date of last OS updates/patches applied	9/15/2017
Power management policy	High Performance
Processor	
Number of processors	2
Vendor and model	AMD EPYC 7601
Core count (per processor)	32
Core frequency (GHz)	2.20
Stepping	1
Memory module(s)	
Total memory in system (GB)	512
Number of memory modules	16
Vendor and model	Crucial MTA36ASF4G72PZ-2G6D1
Size (GB)	32
Type	PC4-2666V
Speed (MHz)	21300
Speed running in the server (MHz)	19200
Local storage	
Number of drives	8
Drive vendor and model	Samsung MZ7LM960HMJP-00005
Drive size (GB)	960
Drive information (speed, interface, type)	6Gb SATA SSD

Server configuration information	
PCIe storage	
Number of drives	2
Drive vendor and model	Micron MTFDHAX800MCE-1AN1ZABYY
Drive size (GB)	800
Drive information (interface, type)	PCIe NVMe SSD
Network adapter	
Vendor and model	Mellanox ConnectX-4 Lx
Number and type of ports	2 x 25GbE
Driver version	1.70.18120.0
Cooling fans	
Vendor and model	Delta Electronics GFC0812DS
Number of cooling fans	4
Power Supplies	
Vendor and model	LITEON PS-2112-5Q
Number of power supplies	2
Wattage of each (W)	1200

Appendix B: How we tested

Configuring the top-of-rack switch

We used a Mellanox SN2700 100GbE switch for our cluster networking traffic. We placed all cluster-facing ports into trunk mode, enabled Data-Center-Bridging, and enabled Priority Flow Control on all ports. We also configured ingress and egress buffer sizes. Our switch configuration is shown below (we used ports 1, 3, and 5 for our cluster nodes, and port 23 for the infrastructure host):

```
   interface ethernet 1/1 module-type qsfp-split-4 force
   interface ethernet 1/3 module-type qsfp-split-4 force
   interface ethernet 1/5 module-type qsfp-split-4 force
   interface ethernet 1/23 module-type qsfp-split-4 force

   interface ethernet 1/1/1-1/1/4 ingress-buffer iPort.pg0 map pool iPool0 type lossy reserved 20480
shared alpha 8
   interface ethernet 1/1/1-1/1/4 ingress-buffer iPort.pg3 map pool iPool0 type lossless reserved 90000
xoff 17000 xon 17000 shared alpha 2
   interface ethernet 1/3/1-1/3/4 ingress-buffer iPort.pg0 map pool iPool0 type lossy reserved 20480
shared alpha 8
   interface ethernet 1/3/1-1/3/4 ingress-buffer iPort.pg3 map pool iPool0 type lossless reserved 90000
xoff 17000 xon 17000 shared alpha 2
   interface ethernet 1/5/1-1/5/4 ingress-buffer iPort.pg0 map pool iPool0 type lossy reserved 20480
shared alpha 8
   interface ethernet 1/5/1-1/5/4 ingress-buffer iPort.pg3 map pool iPool0 type lossless reserved 90000
xoff 17000 xon 17000 shared alpha 2
   interface ethernet 1/23/1-1/23/4 ingress-buffer iPort.pg0 map pool iPool0 type lossy reserved 20480
shared alpha 8
   interface ethernet 1/23/1-1/23/4 ingress-buffer iPort.pg3 map pool iPool0 type lossless reserved 90000
xoff 17000 xon 17000 shared alpha 2
   interface ethernet 1/1/1-1/1/4 ingress-buffer iPort.pg3 bind switch-priority 3
   interface ethernet 1/3/1-1/3/4 ingress-buffer iPort.pg3 bind switch-priority 3
   interface ethernet 1/5/1-1/5/4 ingress-buffer iPort.pg3 bind switch-priority 3
   interface ethernet 1/23/1-1/23/4 ingress-buffer iPort.pg3 bind switch-priority 3
   interface ethernet 1/1/1-1/1/4 egress-buffer ePort.tc3 map pool ePool0 reserved 4096 shared alpha inf
   interface ethernet 1/3/1-1/3/4 egress-buffer ePort.tc3 map pool ePool0 reserved 4096 shared alpha inf
   interface ethernet 1/5/1-1/5/4 egress-buffer ePort.tc3 map pool ePool0 reserved 4096 shared alpha inf
   interface ethernet 1/23/1-1/23/4 egress-buffer ePort.tc3 map pool ePool0 reserved 4096 shared alpha inf

   interface ethernet 1/1/1 mtu 9000 force
   interface ethernet 1/1/2 mtu 9000 force
   interface ethernet 1/1/3 mtu 9000 force
   interface ethernet 1/1/4 mtu 9000 force
   interface ethernet 1/3/1 mtu 9000 force
   interface ethernet 1/3/2 mtu 9000 force
   interface ethernet 1/3/3 mtu 9000 force
   interface ethernet 1/3/4 mtu 9000 force
   interface ethernet 1/5/1 mtu 9000 force
   interface ethernet 1/5/2 mtu 9000 force
   interface ethernet 1/5/3 mtu 9000 force
   interface ethernet 1/5/4 mtu 9000 force
   interface ethernet 1/23/1 mtu 9000 force
   interface ethernet 1/23/2 mtu 9000 force
   interface ethernet 1/23/3 mtu 9000 force
   interface ethernet 1/23/4 mtu 9000 force
   interface ethernet 1/1/1 switchport mode trunk
   interface ethernet 1/1/2 switchport mode trunk
   interface ethernet 1/1/3 switchport mode trunk
   interface ethernet 1/1/4 switchport mode trunk
   interface ethernet 1/3/1 switchport mode trunk
   interface ethernet 1/3/2 switchport mode trunk
   interface ethernet 1/3/3 switchport mode trunk
   interface ethernet 1/3/4 switchport mode trunk
   interface ethernet 1/5/1 switchport mode trunk
   interface ethernet 1/5/2 switchport mode trunk
   interface ethernet 1/5/3 switchport mode trunk
   interface ethernet 1/5/4 switchport mode trunk
```

```
interface ethernet 1/23/1 switchport mode trunk
interface ethernet 1/23/2 switchport mode trunk
interface ethernet 1/23/3 switchport mode trunk
interface ethernet 1/23/4 switchport mode trunk

vlan 10-13

spanning-tree port type edge default

dcb priority-flow-control enable force
dcb priority-flow-control priority 3 enable
interface ethernet 1/1/1 dcb priority-flow-control mode on force
interface ethernet 1/1/2 dcb priority-flow-control mode on force
interface ethernet 1/1/3 dcb priority-flow-control mode on force
interface ethernet 1/1/4 dcb priority-flow-control mode on force
interface ethernet 1/3/1 dcb priority-flow-control mode on force
interface ethernet 1/3/2 dcb priority-flow-control mode on force
interface ethernet 1/3/3 dcb priority-flow-control mode on force
interface ethernet 1/3/4 dcb priority-flow-control mode on force
interface ethernet 1/5/1 dcb priority-flow-control mode on force
interface ethernet 1/5/2 dcb priority-flow-control mode on force
interface ethernet 1/5/3 dcb priority-flow-control mode on force
interface ethernet 1/5/4 dcb priority-flow-control mode on force
interface ethernet 1/23/1 dcb priority-flow-control mode on force
interface ethernet 1/23/2 dcb priority-flow-control mode on force
interface ethernet 1/23/3 dcb priority-flow-control mode on force
interface ethernet 1/23/4 dcb priority-flow-control mode on force
```

Configuring the infrastructure server

We used a single 2U, dual-socket server for hosting the Active Directory and Storage Spaces Direct management virtual machines. We installed Windows Server 2016 Datacenter edition and added the Hyper-V role. On the Active Directory VM, we added the Active Directory Domain Service and DNS Server roles, and created a new domain for our test cluster. On the Storage Spaces Direct management VM, we added Remote Server Administration Tools (RSAT) for PowerShell and Failover Clustering.

Configuring the cluster nodes

We used three 2U, dual-socket AMD EPYC-powered servers for our test cluster. We set the BIOS on each server to Performance mode. We first installed Windows Server 2016 Datacenter Edition onto a 128GB SATA SSD, separate from the drives designated for cluster storage. We then set the power policy to High Performance, configured the network adapters with IP addresses and joined each server to the domain. Finally, we ran the network and storage commands to enable RDMA and Storage Spaces Direct. We performed the following steps for each server:

Installing Windows Server 2016

1. In the server BIOS, change the determinism slider to Performance, and disable IOMMU. We re-enabled IOMMU after installing the necessary Windows Updates.
2. Boot the server to the Windows Server 2016 installation media. We used the BMC console virtual media to mount the ISO image and install remotely.
3. At the prompt, press any key to boot from the CD/DVD location.
4. Click Next.
5. Click Install Now.
6. Select Windows Server 2016 Datacenter Edition (Desktop Experience), and click Next.
7. Check I accept the license terms, and click Next.
8. Select the 128GB OS drive, and click Next.
9. After installation, enter a password for Administrator, and click Finish.
10. Open Control Panel and search for Power Options.
11. Click Power Options.
12. Change the power policy to High Performance, and click Apply.
13. Install all available Windows Updates, rebooting the server as necessary.
14. In the server BIOS, re-enable IOMMU.

Configuring VLAN and IP addresses

1. Use the Windows Run dialog to open **ncpa.cpl**.
2. Right-click the first Mellanox adapter port, and click Properties.
3. Click Configure.
4. Click Advanced.
5. For VLAN ID, enter the VLAN for the corresponding switch port. For example, we used VLANs 10, 11, 12, and 13 for the four ports on each server.
6. Click OK.
7. Uncheck IPV6, and double-click the IPV4 protocol to open the IP address dialog box.
8. Enter an IPv4 address, network mask, and DNS server.
9. Click OK.
10. Click OK.

Joining the server to the domain

1. Use the Windows Run dialog to open **sysadm.cpl**.
2. Click Change.
3. Select Domain, and enter the domain name.
4. Click OK.
5. Enter the password for the domain administrator, and click OK.
6. Reboot the server and log in as the domain administrator

Installing server roles and features

1. Open a PowerShell window as the domain administrator and run the following command to install the required roles and features for Storage Spaces Direct:

```
Install-WindowsFeature -Name "Data-Center-Bridging","Failover-Clustering","Hyper-V","RSAT-Clustering-PowerShell","Hyper-V-PowerShell"
```

Configuring the network adapters for RDMA

1. Open a PowerShell window as the domain administrator.
2. To set a network QoS policy, run the following commands:

```
New-NetQosPolicy "SMB" –NetDirectPortMatchCondition 445 –PriorityValue8021Action 3

Enable-NetQosFlowControl –Priority 3

Disable-NetQosFlowControl –Priority 0,1,2,4,5,6,7

Enable-NetAdapterQos –Name "<Ethernet>", "<Ethernet 2>","<Ethernet 3>","<Ethernet 4>"

New-NetQosTrafficClass "SMB" –Priority 3 –BandwidthPercentage 30 –Algorithm ETS
```

3. To enable RDMA on the adapters, run the following commands:

```
Enable-NetAdpaterRDMA *
```

Running the cluster validation tool

1. Open a PowerShell window as the domain administrator.
2. To validate the cluster configuration, run the following command:

```
Test-Cluster –Node <Node1, Node2, Node3 > –Include "Storage Spaces Direct", "Inventory", "Network", "System Configuration"
```

3. Open the cluster validation report, and ensure there are no failures. If there are warnings in the report, verify that they are negligible, or correct the warnings before deployment.

Deploying Storage Spaces Direct

Open a PowerShell window as the domain administrator.
Run the following command to create a new Storage Spaces Direct cluster without storage (we will add storage later):

```
New-Cluster –Name <ClusterName> –Node <Node1,Node2,Node3> –NoStorage
```

Enabling Storage Spaces Direct

Because our drives were new, we did not need to prepare them before enabling the cluster. However, if your drives have data or metadata from a previous deployment, you may need to run the Clean Disks script. See step 3.4 of the following guide for more information: https://docs.microsoft.com/en-us/windows-server/storage/storage-spaces/hyper-converged-solution-using-storage-spaces-direct

1. Open a PowerShell window as the domain administrator.
2. Run the following command to enable Storage Spaces Direct:

```
Enable-ClusterStorageSpacesDirect -CimSession <ClusterName>
```

Enabling Storage Spaces Direct

We used the high-performance PCIe NVMe SSDs to cache reads and writes.

1. Run the following command to enable the read and write cache on Storage Spaces Direct:

```
Set-ClusterS2D -CacheModeSSD ReadWrite
```

Creating the volumes

Open a PowerShell window as the domain administrator.

1. Run the following commands to create a 2.27TB volume on each node, and an additional 20GB volume for storing VMFleet scripts:

```
New-Volume -FriendlyName "Node1" -FileSystem CSVFS_ReFS -StoragePoolFriendlyName "*S2D*" -Size 2.27TB
-ResiliencySettingName Mirror

New-Volume -FriendlyName "Node2" -FileSystem CSVFS_ReFS -StoragePoolFriendlyName "*S2D*" -Size 2.27TB
-ResiliencySettingName Mirror

New-Volume -FriendlyName "Node3" -FileSystem CSVFS_ReFS -StoragePoolFriendlyName "*S2D*" -Size 2.27TB
-ResiliencySettingName Mirror

New-Volume -FriendlyName "collect" -FileSystem CSVFS_ReFS -StoragePoolFriendlyName "*S2D*" -Size 20GB
-ResiliencySettingName Mirror
```

Configuring VMFleet

VMFleet is a set of scripts designed to make it easy to clone out VMs and run Microsoft DISKSPD on each. We used Windows Server 2016 Datacenter Edition Server Core as our master image operating system. We used the VMFleet scripts and documentation at this GitHub repository: https://github.com/Microsoft/diskspd/tree/master/Frameworks/VMFleet

Creating the master image

1. On one of the cluster nodes, open Hyper-V Manager and click New→Virtual Machine.
2. Click Next.
3. Select Generation 2, and click Next.
4. Click Next.
5. Click Next.
6. Select Create a virtual hard disk. Set the size to 32GB, and name it **`VMFleet_gold.vhdx`**.
7. Click Next.
8. Click Finish.
9. Right-click the virtual machine, and click Settings.
10. Click Add New Hardware, select SCSI Controller, and click OK.
11. Select CD/DVD Drive, and select ISO Image File.
12. Enter the location of the Windows Server 2016 installation media, and click OK.
13. At the prompt, press any key to boot from the CD/DVD location.
14. Click Next.
15. Click Install Now.
16. Select Windows Server 2016 Datacenter Edition, and click Next.
17. Check I accept the license terms, and click Next.
18. Select the VHD, and click Next.
19. After installation, enter a password for Administrator, and click Finish.
20. Shut down the virtual machine.
21. Right-click the virtual machine, and click Delete. Click OK. This does not delete the VHDX file created in step 6.
22. Browse to the location of the VHDX file, and make a copy in the Node1 cluster shared volume.

Cloning out the master image

1. Download the VMFleet suite, and extract the contents to a location on the OS drive.
2. From the VMFleet download location, run the following commands to sort the CSVs and install VMFleet:

```
.\install-vmfleet.ps1 -source .
```

3. Copy the 64-bit version of DISKSPD to `C:\ClusterStorage\collect\control\tools` to allow the VMs to download the executable.
4. On one of the cluster nodes, open a PowerShell window as the domain administrator.
5. In PowerShell, navigate to the control folder.
6. Run the following command to clone out 64 VMs per cluster node:

```
.\create-vmfleet.ps1 -BaseVHD C:\ClusterStorage\Node1\VMFleet_gold.vhdx -VMs 64 -Admin Administrator
-AdminPass Password1 -ConnectUser Administrator -ConnectPass Password1 -FixedVHD $true
```

Setting the Storage QoS policy

We used Windows PowerShell and VMFleet tools to set an IOPS limit on each VM.

1. Run the following commands to create and apply a Storage QoS policy to each VM:

```
New-StorageQoSPolicy -Name P15 -MinimumIOPS 1100 -MaximumIOPS 1100 -PolicyType Dedicated
.\set-storageqos -policyname P15
```

2. Restart the VMs to apply the policy.

Running VMFleet

We ran the test three times to ensure repeatability, and reported the median score.

1. Open a PowerShell window as the domain administrator.
2. In PowerShell, navigate to the control folder.
3. Run the following commands to start the VMs:

```
.\start-vmfleet.ps1
.\start-sweep.ps1 -b 4 -t 2 -o 16 -w 10 -p r -d 1200 -warm 600
```

4. Begin a test run with the following parameters:
   a. Block size = 4KB
   b. Write ratio = 10%
   c. Random/sequential = Random
   d. Threads = 2
   e. Outstanding IO = 16
   f. Warmup = 10 minutes
   g. Test duration = 20 minutes

This project was commissioned by AMD.