

EBOOK

# A Pure Primer: Overcoming SQL Server Storage Challenges

Understanding the variety of new architecture options and how they affect storage.

# Contents

Contributing Author .....	5
<b>CHAPTER 1: SQL Server and Modern Infrastructure.....</b>	<b>6</b>
Microsoft SQL Server.....	6
In-Memory Features .....	8
Project Hekaton and In-Memory OLTP .....	8
Columnstore Indexes.....	8
Platform Improvements .....	8
Up Next .....	9
<b>CHAPTER 2: SQL Server and Modern Infrastructure Components .....</b>	<b>10</b>
Infrastructure Stacks.....	10
Storage .....	11
Physical Server .....	12
Virtualization .....	13
Containers .....	13
Containers vs. Virtual Machines .....	14
Cloud IaaS.....	15
Cloud DBaaS .....	15
Networking and Interconnects .....	17
Automation.....	17
Up Next .....	18
<b>CHAPTER 3: SQL Server on Linux .....</b>	<b>19</b>
Core Platform—How Did They Do it? .....	19
Reasons to Run SQL Server on Linux .....	20
SQL Server-on-Linux and Storage .....	20
Up Next .....	22
<b>CHAPTER 4: Big Data Clusters and Storage .....</b>	<b>23</b>
Scale-out Compute Platform .....	23



BDC and Storage .....	25
Highest Performance Possible .....	25
Up Next .....	25
<b>CHAPTER 5: Block Storage for Cloud-Based Workloads .....</b>	<b>26</b>
Pure Cloud Block Store™ .....	26
Architecture.....	27
Bidirectional Replication .....	29
Data Savings.....	29
Always-On Encryption .....	29
Flexibility and Agility.....	30
Availability.....	30
Use Cases .....	30
Disaster Recovery.....	30
DevTest.....	30
Up Next .....	30
<b>CHAPTER 6: Microsoft Azure Arc and Modern Storage .....</b>	<b>31</b>
Azure Arc Architecture.....	31
Storage Speed.....	32
Up Next .....	33
<b>CHAPTER 7: In-memory OLTP .....</b>	<b>34</b>
Concept.....	34
Memory-Optimized Tables and Indexes .....	34
In-Memory Indexes .....	35
Natively Compiled Stored Procedures .....	35
Memory-Optimized TempDB Metadata.....	35
Current Limitations.....	36
Setting It Up .....	36
CMS Database .....	38
Potential Performance Improvement.....	39
Storage Considerations .....	42



CPU Considerations.....44

Memory Considerations.....44

Up Next .....44

**CHAPTER 8: Columnstore ..... 45**

    Concept..... 46

    Configuration and Usage ..... 46

    Storage Implications..... 51

    Limitations..... 51

    Up Next ..... 52

**CHAPTER 9: Combining In-Memory Technologies..... 53**

    Usage..... 53

    Wrap Up ..... 54





### **Contributing Author**

David Klee is a Microsoft MVP and VMware vExpert with a lifelong passion for the convergence of infrastructure, cloud, and database technologies. David spends his days handling performance and HA/DR architecture of mission-critical SQL Servers as the Founder of Heraflux Technologies. His areas of expertise are virtualization and performance, data center architecture, and risk mitigation through high availability and disaster recovery. When he is not geeking out on technologies, David is an aspiring amateur photographer. You can read his blog at [davidklee.net](http://davidklee.net), and reach him on Twitter at [@kleegeek](https://twitter.com/kleegeek). David speaks at a number of national and regional technology related events, including the PASS Summit, VMware VMworld, IT/Dev Connections, SQL Saturday events, SQL Cruise, SQL PASS virtual chapter webinars, and many SQL Server and VMware User Groups



## CHAPTER 1:

## SQL Server and Modern Infrastructure

Bob Dylan once wrote a song called “The Times They Are a-Changin’.” Nothing could be truer for business in the modern era. Data volumes are exploding in businesses worldwide. Companies are scrambling to manage increasing demands on storage. Cloud and automation are here in big ways. Companies are pivoting to leverage the advancements in technology to improve their businesses. And data platforms are at the core of these initiatives.

Microsoft SQL Server and other large database management systems (DBMS) consume compute and infrastructure resources at rates rarely seen in other applications. As such, infrastructure teams have feared, and sometimes even resented, how these large databases can challenge an enterprise server infrastructure. As the volume of data grows inside an organization, demands have pushed IT organizations to look for the means to improve the performance of the systems that drive the business.

Most of these systems either facilitate the operations of the business (e.g., transactional) or act as decision-making systems (e.g., analytics). In profiling these systems, most of the bottlenecks exist in how data is stored or accessed on persistent storage.

Historically, this non-volatile storage was much cheaper than server memory, and therefore all data access operations were performed from disk. Since then, the price of memory has dropped while capacity has grown. Now, data technologies that can leverage large amounts of system memory and store the full quantity of data in memory to boost performance are here. Database systems with 512GB of memory or more are relatively common and can be ideal for today’s highly concurrent, mission-critical systems. Business intelligence and analytics platforms, scale-up relational workloads, reporting, and more can usually benefit from memory footprints as large as the budget allows.

The trouble is that these new data technologies often require extensive modifications to existing applications, which many software vendors are unable or unwilling to support.

Microsoft Corporation took a different approach.

### Microsoft SQL Server

Microsoft has acknowledged from the beginning of the product line that the consumption of critical resources is an ever-increasing bottleneck to performance. Over the years, they’ve released versions to compensate for the limitations of infrastructure at the time.

The first versions of SQL Server go all the way back to the port of Sybase SQL Server onto OS/2 in 1989. These made valid assumptions about the price and capacity of system memory. At the time, the cost of memory was high and the maximum quantities available in a single server were low—lower than the amount of data being stored. As a result, the original SQL Server designs used disk-based storage, and only the immediate working set of data was kept in memory. As newer data was fetched from disk to fulfill a request, unused, older data was flushed from memory to make room.

More recently, however, system architectures have changed. The memory capacity in modern servers has skyrocketed. The development of 64-bit CPU architecture allowed systems to exceed the 4GB addressable memory limit present in 32-bit architectures. Large quantities of server memory are also much less expensive than before. Some of the largest servers



commercially available today can contain up to 896 physical CPU cores and 48TB of memory at prices far lower than previous generations of servers. Plus, at the speed of systems development, future systems are sure to contain many more times these figures.

The speed of addressable system memory is blazingly fast. System memory performance is gauged in several ways, most importantly in latency, which is measured in nanoseconds. Modern system memory operates between 13.5- and 15-nanosecond latency. Small amounts of cache inside the CPU itself are even faster (down to one nanosecond).

As a result, the amount of memory available and the speed of that memory accessible by SQL Server is greater than ever before. However, SQL Server's core architecture contains a storage engine that is still oriented to disk-based storage. It can only fetch a working set of data into memory for processing and assumes all data is stored on disk and not in memory.

The storage medium on which SQL Server stores data is much slower than server memory. The performance is also measured by latency, but the scale is much higher. Traditional storage arrays in modern data centers and public cloud platforms measure the latency of the array's performance in milliseconds. The fastest storage on the market today, nonvolatile enterprise flash storage, is much faster and is measured in *microseconds* of latency instead of milliseconds, but it's still exponentially slower than server memory.

The differences in scale are tremendous.

Fractions of a Second	Metric Name
0.000 000 001	Nanosecond
0.000 001	Microsecond
0.001	Millisecond

So why not store all the SQL Server data in memory, since it's so much faster?

The primary challenge with leveraging conventional memory for storage of this data is that it's a volatile medium. If the power in the server goes out, the contents of memory are immediately lost. The data must reside on nonvolatile storage that can persist after power cycling.

Additionally, every bit of changed data that the business wishes to keep must be persisted to disk before the operation completes. This includes new data being written to the database, and any changes to existing data. If the working set of data happens to be in memory at the time of the change, the operation of writing the change back to disk slows the entire operation down to the effective speed of that disk.

Today, most modern SQL Server environments are limited in performance due to the speed of underlying data storage systems. Even servers with terabytes of memory are underperforming because of assumptions made by the SQL Server storage engine and the effective speed of the storage underneath.



## In-Memory Features

In recent SQL Server releases, Microsoft's efforts to improve the performance of the server by leveraging in-memory technologies have resulted in two key distinguishing features:

- In-memory online transaction processing (OLTP)
- Columnstore indexes

Microsoft is also committed to making these features available to more users. As of SQL Server 2016 Service Pack 1, In-Memory OLTP and Columnstore indexes are now available for use in SQL Server Standard Edition. Previously, these features were only available to those using Enterprise Edition.

Let's explore these exciting new features.

### Project Hekaton and In-Memory OLTP

Microsoft acknowledged that this basic assumption of storage and memory was becoming an ever-increasing bottleneck to performance. In 2008, it began work on building a database platform designed for in-memory operations. The origins of this project came from the goal to be 100 times faster than the base SQL Server engine, and the codename Hekaton, derived from the Greek word for 100, was chosen.

Microsoft elected to embed the new engine inside the existing SQL Server engine instead of delivering a new product. This became the SQL Server In-Memory OLTP feature that first shipped with the SQL Server 2014 release.

This new engine allows developers to utilize new memory-optimized data structures and improve the performance of key business processes, all while reducing development time because the base platform is extended and not replaced.

### Columnstore Indexes

Microsoft also realized that traditional row-based indexes are inefficient for certain types of queries, including data warehouse queries, analytical processing, and reporting. Based on xVelocity, an advanced technology originally developed for use with PowerPivot and SQL Server Analysis Services, and beginning with the SQL Server 2012 release, developers can now define columnstore indexes on database tables.

Columnstore indexes store columns instead of rows in a memory-optimized columnar format instead of the B-tree structure of traditional indexes. These can be used to improve the efficiency of certain types of queries while reducing the memory footprint of the operation.

## Platform Improvements

Additionally, the core platform itself has been evolving rapidly over the last few years to provide more flexibility and scalability.

The business world has tackled public cloud in numerous ways. Some are taking all-in approaches while others approach it with a best-of approach. The best-of approach allows the organization to use the best platform and tooling for a given task, lending to a "hybrid" multicloud model. Data is moved back and forth from on-premise to the cloud as needed, leveraging features of the various platforms to provide the best functionality for the business.





Managing this movement of data can be quite challenging. To address this challenge, Pure Storage® developed Pure Cloud Block Store™, which allows for data stored on-premises on a Pure Storage device to be seamlessly replicated back and forth to the public cloud. There, data can then be easily leveraged and accessed by cloud servers and services. Chapter 6 of this book has a deep dive into this.

Hybrid cloud models are critical to future system deployments, especially as organizations mature in their platform deployment operations. Day-to-day operations must be efficient and flexible so the platform can respond to ever-changing business requirements. As requirements frequently change, leveraging cloud resources allows IT organizations to scale as needed beyond the bounds of their on-premises infrastructure. The hybrid model helps IT organizations scale with the business, not just expand the capacity of an on-premises data center and stop there. The management of these hybrid systems must be cohesive and simple in nature.

To make it easier to deploy and manage these hybrid platforms, Microsoft unveiled Azure Arc. Arc allows administrators to extend cloud-based Azure services to on-premise infrastructures with a central management system. A much deeper dive on this is also present in Chapter 5.

SQL Server's core underlying architecture presented historical challenges with scaling a SQL Server workload outwards (rather than the traditional upwards) to accommodate larger workload demands. The SQL Server 2019 release introduced a feature called Big Data Clusters (BDC) that allowed a SQL Server platform to scale outwards to better serve modern workloads. To accommodate these diverse workloads that sometimes run in containers, Microsoft expanded to run SQL Server on Linux. A much deeper dive on these topics is also present in Chapters 3 and 4.

## **Up Next**

Each one of these features can lead to significant performance boosts and operational improvements in certain scenarios and use cases. But looking at the big picture, each comes with its own implications and architectural considerations in the enterprise data center. Database administrators (DBAs) might not have visibility into the infrastructure under the data platform, and might not see the net changes in the infrastructure and impacts to the infrastructure as a result of leveraging these technologies.

The following chapters will explore each of these features in-depth along with the underlying infrastructure. Then, we'll present techniques and scenarios for when and how to leverage these features and platforms and discuss the architectural challenges and considerations necessary to successfully implement these features in today's modern data centers.



## CHAPTER 2:

## SQL Server and Modern Infrastructure Components

Microsoft's SQL Server engine can only perform as fast as the slowest bottleneck in the entire system stack allows. In any modern data center, whether in the cloud or on-premises, bottlenecks are sure to exist at, around, or underneath the database layer. SQL Server places an incredible demand on the infrastructure, and in-memory features push certain areas even harder. Rapid orchestration technologies can also be held back if the underlying platform cannot keep up with the demand.

Let's look through these layers to familiarize you with them and how each impacts the performance of the database server. These layers are present in both on-premises and public cloud deployments— even if you might not be able to see that layer directly, as is the case with public cloud platforms.

### Infrastructure Stacks

The layers around the database engine are quite commonplace. When referring to the traditional data center model, the layers are relatively fixed, and virtualization is assumed to be in use.

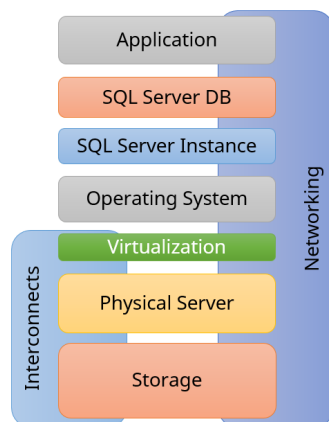


Figure 2-1. Traditional Datacenter Layers

If the SQL Server is placed in a virtual machine (VM) in a public cloud infrastructure-as-a-service (IaaS) platform, such as Microsoft Azure or Amazon Web Services (AWS), these layers are still present, but certain parts of the stack are now abstracted away from and unavailable for configuration by the end user.

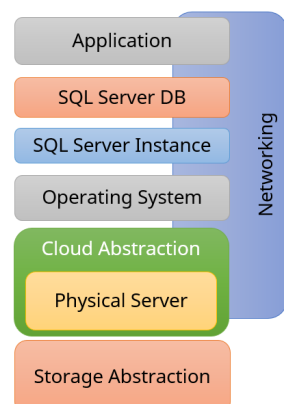


Figure 2-2. Cloud IaaS Layers



In the IaaS model, the virtualization and the physical server layers are replaced with an automated abstraction layer that allows end users to provision VMs. Storage is connected through this abstraction layer and the end user selects the performance class and capacity of the storage. The end user cannot see the specifics of the storage or hardware.

If the model is pushed even further towards the database-as-a-service model (DBaaS), an even greater level of abstraction is exhibited.

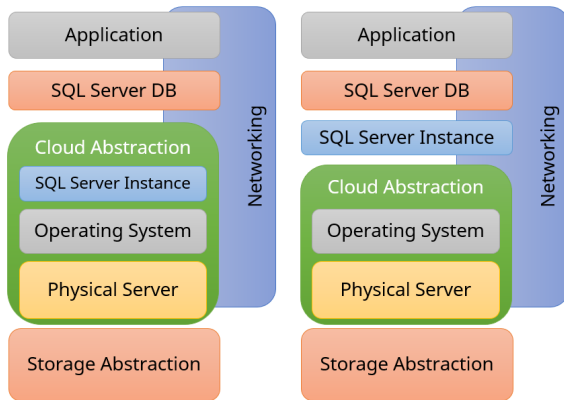


Figure 2-3. DBaaS Layers

In some DBaaS models (Figure 3, left), such as Microsoft Azure SQL Database, the abstraction layer occurs all the way up to the individual SQL Server database, and the SQL Server instance layer is abstracted and not accessible. Individual databases are provisioned independently of the others. In other cases (Figure 3, right), such as Microsoft Azure SQL Managed Instance or Amazon RDS for SQL Server, the abstraction layer stops at the SQL Server instance, and the end user can provision and interact with any number of databases on the instance.

In all the cases, the layers are still there, and each has performance characteristics and associated performance challenges. Let’s briefly explore each layer and discuss how each relates to SQL Server and modern data center platforms.

## Storage

Arguably, the storage layer is the most important component of the infrastructure stack underneath the database itself. The storage is where the most valuable asset of the business, its data, permanently resides. Many options for storage exist, no matter if the target platform is a data center on-premises or in the public cloud. While the vast majority of these options are highly available (and you avoid ones that are not), the performance characteristics and capacity of these platforms vary wildly between versions and vendors.

Additional benefits accompany certain modern storage platforms. For each server deployed, a certain amount of repetitive blocks on disk accompany the copies of the underlying operating system files. For test systems, copies of production databases repeatedly restored onto these test servers for validation and testing purposes are quite common. The underlying storage could reduce the overall storage consumption by deduplicating these blocks at the storage layer by only writing a unique block once and writing pointers the remainder of the time. The actual storage consumption of such data can be reduced, sometimes by a considerable amount.

For some SQL Servers where in-memory features are leveraged, if the purpose of in-memory databases is to have all the data completely in memory and not rely on disk, why is storage so important? Two primary reasons exist.



The first reason is clear. All of this data must be stored on disk if the architect wishes to maintain this data between system reboots. The architect can elect to make certain database tables non-persistent on disk, but this data is lost and gone forever if the database engine or operating system is restarted. For the data that is required to persist, any change to the data must be written to disk before that transaction can complete. The speed of the storage must be quite fast, with the lowest possible latency to disk, to help keep the performance of the transaction at acceptable levels. Essentially, the fastest possible storage is required for persisting in-memory data to disk, because any changes to disk now cause the in-memory operation to perform at the (much slower) speed of disk.

The second reason is less intuitive. The performance of the storage matters not just for the speed of the individual transactions, but also for system startup. For data that is to be in memory, all of it must be read from disk and loaded into memory at system startup before that data is ready for use and is accessible by end users. If the storage subsystem is slow, the startup time for that database can cause the database to be unavailable for an extended period.

## Physical Server

The physical server is the next step up from the storage and contains the primary compute resources of the platform—CPUs and memory. The speed and scale of these servers are, just like any other component in the infrastructure, variable based on design, but are quite fast. Modern servers can scale from one to thirty-two CPU sockets, each with CPUs containing many processing cores and terabytes of available memory.

The speed of these platforms can vary, even within a model line. When storage is no longer the principal bottleneck within the infrastructure, the database workloads are primarily CPU- and memory-bound, and the speed of these two components matter more than before.

CPU performance is determined by the number of cores and clock speed (measured in gigahertz, or GHz) per core. Not all CPUs are created equal, and the speed varies between CPUs. Also, GHz is a measure of speed but not of raw performance. Benchmark metrics exist for all modern CPUs to help you select the fastest possible CPUs for your workload.

For example, based on the Geekbench benchmark at the time of this writing (available at [geekbench.com](https://www.geekbench.com)), a single CPU core on an eight-core Intel Xeon Gold 6244 CPU running at 3.6GHz has an average Geekbench version five single-core score of 1,098. A 16-core Intel Xeon Gold 5218 CPU running at 2.3GHz has a Geekbench single-core score of 786, 28.4% slower per CPU thread. In many mission-critical workloads, a 28% improvement is quite substantial and well worth the small additional price for the faster CPUs on each server. Additionally, if the performance gains per CPU core translate to requiring fewer CPU cores for your SQL Server necessary to maintain performance, your licensing footprint is potentially reduced. This can save your organization licensing capital.

Most operations in an OLTP-based database server are not parallelized. Other workloads on SQL Server, such as analytics, reporting, or decision support workloads, can be widely parallelized. Make your CPU selection wisely according to your workload demands. Spending a small amount of additional capital on a server to purchase CPUs with faster core performance, even at the expense of a slightly reduced core count, can yield a net performance improvement felt by the organization.

In short, ensure that the CPU selection and memory configuration are designed for the most optimal performance that the server budget allows.



## Virtualization

Virtualization is a ubiquitous technology in most modern data centers. It is a software layer, called a hypervisor, that is installed on a physical server. This allows multiple compartmentalized operating systems (VMs) to coexist and run on the same physical server, all while unaware that other VMs are on the same physical server. Common hypervisors are VMware's ESXi and Microsoft's Hyper-V.

VMs are abstracted away from being dependent on a particular type and configuration of hardware. This makes the VMs portable, so they can migrate from one physical server to another with no interruption in service. Because they are no longer dependent on a single physical server, they can be replicated to a disaster recovery site very easily.

Hopefully, by now all your production SQL Servers are virtualized with some modern variant of a virtualization hypervisor. The performance overhead is so insignificant, it can hardly be measured, let alone 'felt' by the applications. The advantages to fully virtualize all servers far outweighs the few challenges with virtualization. At this point, the majority of the challenges are operational rather than technical. These challenges include differences in backup ideologies between the organizational silos, high availability expectations, and other management and monitoring aspects.

## Containers

For organizations that are eager to embrace new technologies, containers are allowing for an even greater amount of abstraction from the underlying platform. Think of a container as a virtual machine with an application installed on it, but without dragging the bloat of the operating system along with the application. The container holds the application, its associated files and objects, and whatever other dependencies that might not be included with the underlying container's operating system. It allows the application to be packaged up for much easier deployment versus traditional VMs. The underlying container engine interfaces with the operating system, providing less overhead and more abstraction for the containers running on them.

Microsoft began supporting containers for the SQL Server engine in the SQL Server 2017 release, originally specifically supporting the Docker container engine and, more recently, the Red Hat OpenShift platform. For organizations embracing containers, SQL Server deployments can now leverage these containers to make deployment of new instances even easier.



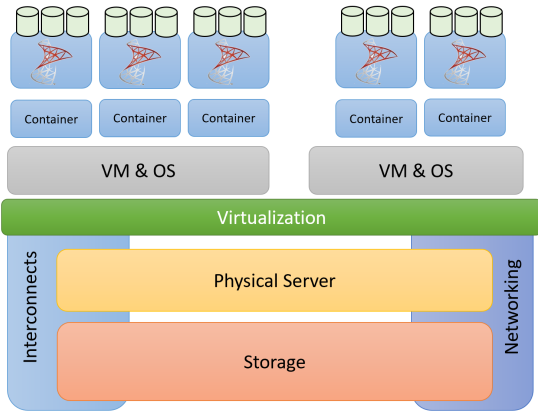


Figure 2-4. SQL Server Container Stack

For environments with rapid software development cycles, the deployment of SQL Server containers can significantly speed up the development process.

### Containers vs. Virtual Machines

Many similarities between containers and VMs exist. They both allow for the portability of a workload as they are not bound by the underlying platform. They both can contain programs and their dependencies. In our case, a SQL Server runtime engine can run successfully in either a container, a traditional VM, or a more traditional physical server deployment.

Containers run inside VMs, but they can also run natively on physical servers without requiring a hypervisor. Running containers in VMs does provide more flexibility in managing the underlying OS needed for the container engine to run, but it is not a requirement.

The primary difference, however, is that the container does not contain the operating system needed to power the applications inside the container. The virtual machine contains this operating system. This underlying operating system is shared between all of the containers running on the underlying container engine, all while keeping the containers isolated from each other. By removing the operating system from the application, the application footprint is much smaller, allowing for reduced compute resources required to power each container.

Containers can run inside VMs while residing on the same VM hosts where traditional SQL Server VMs are deployed.

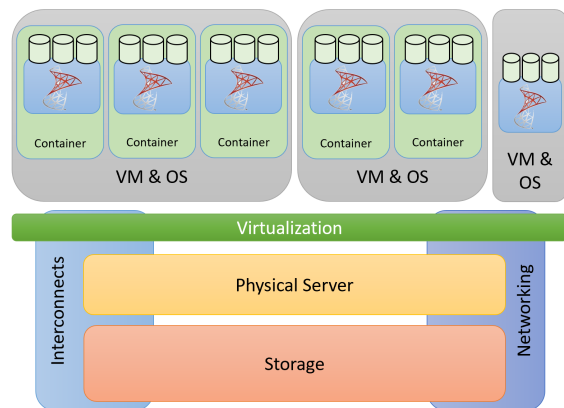


Figure 2-5. Containers and VMs Together



## Cloud IaaS

IaaS technologies are, for all intents and purposes, nearly identical to on-premises virtualization—at least from an architectural level. Public cloud IaaS, or VMs placed in the cloud, is just virtualization in someone else’s data center with sophisticated automation for end-user management and security measures.

The automation portion of IaaS, usually a web-based portal, allows end-users to provision and manage these VMs. Instead of purchasing hardware, storage, interconnects, and associated licensing, users pay for the compute resources, storage, and bandwidth either consumed or allocated on a per-second basis. It is treated more like a utility and can shift the IT infrastructure expenses from capital expenditures (CAPEX) to more of an operational expense (OPEX) model.

Virtual machines are provisioned in the public cloud through a sizing model. Various cloud platforms have their own sizing grids and you should have a thorough understanding of the resource consumption and allocations that your VMs require. Some cloud platforms are harder than others to resize and scale the VM, so be careful when you size the machine. Under-sizing can cause your application to perform poorly. Oversizing the VM can result in elevated costs.

VM Size ↑↓	Offering ↑↓	Family	↑↓	vCPUs ↑↓	RAM (GiB) ↑↓	Data disks ↑↓	Max IOPS
DS11_v2	Standard	Memory optimized	2	14	8	6400	
DS11-1_v2	Standard	Memory optimized	1	14	8	6400	
DS12_v2	Standard	Memory optimized	4	28	16	12800	
DS12-1_v2	Standard	Memory optimized	1	28	16	12800	
DS12-2_v2	Standard	Memory optimized	2	28	16	12800	

**Figure 2-6.** Microsoft Azure VM Sizing Example

Additional cloud-based storage can be provisioned separately, attached to these VMs, and used for additional space and/or speed as the database demand grows over time. Note that the provisioned storage contains an IOPs limitation by data disk, and careful planning should be taken to provision enough disk speed to handle the target workload.

## Cloud DBaaS

Taking the cloud model even further, the database-as-a-service (DBaaS) model allows end users to focus on the database instance and databases and eliminates the need (and even the access) to manage the operating system layer. End users can simply focus on provisioning SQL Server instances and databases (e.g., Microsoft Azure SQL Database Managed Instance or Amazon RDS for SQL Server) or provisioning single databases (e.g., Microsoft Azure SQL Database). The application connects to the SQL Server database using a similar connection string to normal database servers. No operating system maintenance is required by the end user.



Sizing methodologies applied to cloud VMs also apply to cloud databases.

Compute Hardware

Click "Change configuration" to see details for all hardware generations available including memory optimized and compute optimized options

Hardware Configuration

**Gen5**  
up to 80 vCores, up to 408 GB memory  
[Change configuration](#)

Save money

Save up to 55% with a license you already own. Already have a SQL Server license?  Yes  No

vCores [How do vCores compare with DTUs? ↗](#)

2 4 6 8 10 12 14 16 18 20 24 32 40 80

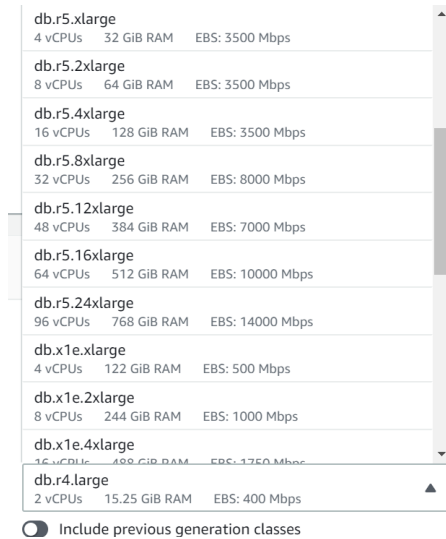
Data max size

1 GB 768 GB 1.5 TB

230.4 GB LOG SPACE ALLOCATED

Figure 2-7. Microsoft Azure SQL DB Compute and Storage Sizing Example





db.r5.xlarge	4 vCPUs	32 GiB RAM	EBS: 3500 Mbps
db.r5.2xlarge	8 vCPUs	64 GiB RAM	EBS: 3500 Mbps
db.r5.4xlarge	16 vCPUs	128 GiB RAM	EBS: 3500 Mbps
db.r5.8xlarge	32 vCPUs	256 GiB RAM	EBS: 8000 Mbps
db.r5.12xlarge	48 vCPUs	384 GiB RAM	EBS: 7000 Mbps
db.r5.16xlarge	64 vCPUs	512 GiB RAM	EBS: 10000 Mbps
db.r5.24xlarge	96 vCPUs	768 GiB RAM	EBS: 14000 Mbps
db.x1e.xlarge	4 vCPUs	122 GiB RAM	EBS: 500 Mbps
db.x1e.2xlarge	8 vCPUs	244 GiB RAM	EBS: 1000 Mbps
db.x1e.4xlarge	16 vCPUs	488 GiB RAM	EBS: 1750 Mbps
db.r4.large	2 vCPUs	15.25 GiB RAM	EBS: 400 Mbps

Include previous generation classes

**Figure 2-7.** Amazon RDS SQL Server Instance Compute Sizing Example

Note that storage still needs to be managed for these databases carefully, and that the in-memory demands on storage should be factored into the storage architecture upfront when the instance or database is provisioned.

## Networking and Interconnects

Every device in the infrastructure stack must communicate with one another. Server-to-server communication is handled through Ethernet networking switches and networking adapters on each server. Storage communication can happen through traditional networking or can leverage Fibre Channel or InfiniBand technologies for dedicated storage communication channels.

As the demands on storage and/or networking interfaces increase, the traffic rates increase. The interconnects between the storage and the servers can become a bottleneck under a heavy load, slowing down operations. Server-to-server communication can also bottleneck between the servers. The sources of traffic are endless, including large file copy operations (backups, video streaming, analytics server requests), “chatty” applications that continuously send requests to other servers, log collection, IoT devices, software updates, and malware outbreaks.

Special care must be taken to ensure that these communication pathways are not overwhelmed during daily operations. The networking and interconnect platform must be designed with multipathing enabled so that the demand can scale to the available interconnects. Monitoring should be present to alert administrators if the available interconnects are running out of available bandwidth.

## Automation

With all of the advancement of on-premises server farms, both physical and virtual, automation capabilities have been limited in nature and scope. Each platform had its own method for automating certain repetitive tasks, such as deploying a virtual machine or installing a database server platform. Few of the methods would interact or plug into other platforms. Therefore, over the years, truly automating IT operations for these on-premises platforms was limited, and most IT operations were (and still are) handled manually and inefficiently.



One of the biggest leaps in management efficiency for these on-premises platforms has been automation improvements. These originally came with public cloud management and eventually extended to on-premises deployments as well. Each cloud provider contained a management layer that allowed administrators to rapidly develop automation routines for common tasks, all while handling the underlying operations, even if these operations were with dissimilar systems.

Orchestration technologies such as Kubernetes made it much easier for administrators to manage complex deployments. For quite a while, however, these cloud object deployment and management tools were disjointed from their on-premises equivalents, and the on-premises tasks still relied on manual human intervention.

Thankfully, Microsoft has improved its management capabilities not only for on-premises deployments but also for hybrid cloud environments. On-premises deployments can now use the same management routines (i.e., reusable scripts) to perform these daily operational tasks in their own data centers as well as in the cloud. The tooling now allows for a single set of management processes, no matter where the underlying server resides. This allows IT to operate more quickly and efficiently and adjust to the ever-changing requirements of the business.

Kubernetes is one of the most commonly used tools for managing container deployments. Kubernetes provides an industry-standard open-source container engine for container deployment, management, and orchestration of routine tasks. Application upgrades, migrations, load balancing, resource utilization, and scale all are handled by this orchestration layer. SQL Server 2019's Big Data Clusters (BDC) leverage Kubernetes as the management and orchestration layer to deploy and scale the BDC components as the workload demand scales. More information on BDC is available in Chapter 4.

Azure Arc was also introduced to allow for even more automation, enabling administrators to manage on-premises environments in the same manner and with the same tooling as cloud deployments. Cloud-native features, such as DBaaS, can now be deployed on-premises in the same easy, self-service manner as cloud deployments. More information on Azure Arc is available in Chapter 5.

Underneath all of the scale and automation, storage speed is incredibly critical. With any deployment and management solution, the time taken to deploy, migrate, or update the platform is mostly dependent on the storage layer underneath. These platforms were designed to improve IT's response time for business requests. Bottom line: the storage layer must be fast enough to complete the request as quickly as possible.

## **Up Next**

Now that the infrastructure underneath SQL Server has been highlighted, let's review the change in the operating system for SQL Server: Linux.



## CHAPTER 3:

# SQL Server on Linux

Microsoft has changed significantly since it first launched SQL Server. Historically, Microsoft maintained a somewhat unfavorable stance toward non-Microsoft operating systems. However, Microsoft has started responding to industry trends and demands. Recently, Microsoft embraced the open-source ecosystem and is now a leading contributor to open-sourced products.

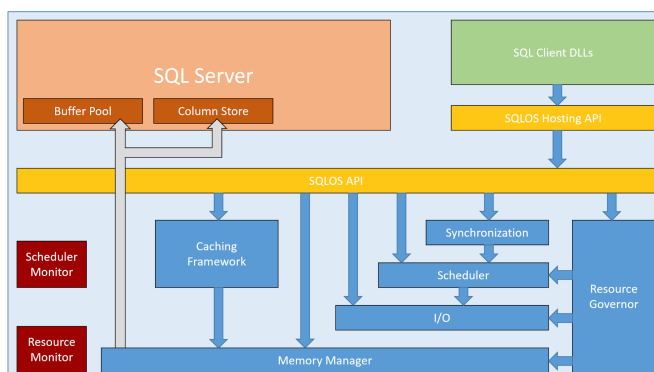
Beginning with its 2017 release, SQL Server can now be installed natively on the Linux operating system. In part, Microsoft knew that they wanted SQL Server to be run in a container. To do this, SQL Server needed to natively run on Linux as it is the base operating system for many of the leading container engines. They also knew they were losing market share to other open-source DBMS platforms, such as MySQL, MariaDB, and PostgreSQL, due to their ability to run on Linux. Finally, they wanted to compete for the developers who run these platforms natively on their development machines (including Apple). Many other Microsoft technologies were expanding onto non-Microsoft platforms, and the SQL Server engineering team decided to take the plunge as well.

## Core Platform—How Did They Do it?

Microsoft SQL Server is so complex that it contains a miniature operating system called the SQLOS. This little operating system handles many of the tasks that Windows would normally perform, mainly so that SQL Server can more precisely manage and control the various processes occurring within the SQL Server engine. It is responsible for CPU scheduling and I/O operations, CPU multithreading management and coordination, memory management, exception handling, and providing a conduit to the external components such as CLR. All of these items are contained within the core SQL Server runtime engine.

When these tasks are broken down into their core dependencies, only about 45 calls are extended through the SQL Server engine into the underlying operating system for tasks such as Active Directory awareness for authentication.

To modify the core SQL Server engine for use on the Linux platform, these underlying OS-dependent tasks were broken out into a new construct called the Platform Abstraction Layer, or “PAL.” The PAL is an abstraction layer that allows anything that is operating system-dependent to be moved into a single location so the rest of the codebase can remain operating system-agnostic. Modifying a Microsoft Research project called “Drawbridge” with components of the SQLOS allowed for the creation of the PAL.



**Figure 3-1.** SQL Server OS

(Adapted from <https://cloudblogs.microsoft.com/sqlserver/2016/12/16/sql-server-on-linux-how-introduction/>)



To maintain the strict compatibility required for such a complex platform, SQL Server on Linux is restricted to the following Linux and container distributions:

- Red Hat Enterprise Linux
- SUSE Linux Enterprise Server
- Ubuntu
- Docker

## Reasons to Run SQL Server on Linux

Why run SQL Server on Linux? There are a few compelling reasons, including cost, compatibility, performance, consistency and security. Let's take a closer look.

- First, if the rest of the servers in an organization are non-Microsoft Windows platforms, it makes sense to maintain the skill and operational practices to support a single base platform.
- Second, the performance of SQL Server-on-Linux is on par with the Windows OS equivalent, especially since it is the same codebase and not a project fork.
- Developers can now develop for SQL Server natively on virtually any development platform in the industry, including Apple (via Docker containers) and Linux.
- And, for cloud platform deployments, Linux consumes fewer base resources than an equivalent Windows server environment. As the scale of deployments grows, requiring substantially less overhead difference can allow organizations to save underlying host compute resources.

Linux also changes the security footprint for the database engine. Many exploits of the SQL Server platform target underlying operating system vulnerabilities. By changing the platform, it changes the target footprint. Another bonus: Linux can also be patched without reboots, unlike Windows, so security patches can often be applied sooner than Windows equivalents.

The Linux OS also costs less than the equivalent Windows Server OS, which never hurts the bottom line.

## SQL Server-on-Linux and Storage

Underneath the Linux server, storage requirements are identical to SQL Server-on-Windows. It boasts extremely low latency to disk, IOPs as high as the SQL Server can demand them, and peak throughput faster than the code or query can accommodate. From inside the Linux operating system, we can take the core principles of Windows and adapt them to the Linux platform.

First, multiple disks are still recommended. For physical Linux servers, multiple underlying LUNs help the OS and the SQL Server layer parallelize and widely distribute I/O requests. For virtual Linux servers, the same concept applies from inside the OS. Having more virtual disks helps spread out the workload. At the hypervisor level, the data stores (formatted LUNs) have their own queues. Monitoring the queues and placing the virtual disks on enough queues to prevent a bottleneck can improve performance. Active multipathing to the underlying storage in either scenario helps to maximize the storage interconnects, improving performance that much more.



The disks are similar to Windows-based object placement. It's best to start with at least one for the user database data files, one for logs, one for TempDB, and one for backups (if you are planning to back up the databases onto the local machine instead of to a network target).

Once the disks are presented to the Linux OS, the SQL Server supports formatting these disks in either the EXT4 or XFS file system formats. We recommend using XFS. XFS can provide a faster crash recovery, can scale to exabytes, and can handle high amounts of concurrent operations more efficiently than EXT4.

```
root@sqllinuxubuntu:/var/opt/mssql/data# sudo lsblk -o NAME,FSTYPE,SIZE,MOUNTPOINT,LABEL
NAME                                FSTYPE      SIZE MOUNTPOINT LABEL
fd0                                  4K
sda                                  200G
sdb                                  500G
sdc                                  250G
sdd                                  40G
├─sdd1                                vfat        512M /boot/efi
├─sdd2                                ext2        488M /boot
├─sdd3                                LVM2_member 39G
│   └─sqllinuxubuntu--vg-root         ext4        23G /
│       └─sqllinuxubuntu--vg-swap_1   swap        16G [SWAP]
sr0                                  iso9660     829M          Ubuntu-Server 16.04.2 LTS amd64
```

Figure 3-2. Connected Disks

Since these disks will most likely need to be expanded, consider using the Logical Volume Manager (LVM) to manage the disks instead of regular partitions. LVM allows for the partition expansion process to occur under active disks in the same way as Windows Disk Manager.

[Find more information on formatting and mounting the virtual disks.](#)

Historically, Windows-based volumes are connected to the operating system either with drive letters (such as H:) or as mount points (such as D:\Mount\SQLData). For Linux, the underlying file system is a logical tree-based structure instead of using arbitrary drive letters. This means mount points are the only option.

Your choices for where to place the mount points are limitless, but convention usually puts these additional virtual disks under folders such as /mnt/ or /var/opt/mssql/data/, the default location for SQL Server objects. The /etc/fstab file can be edited to permanently mount the disks to the location you prefer inside the Linux OS.

```
/dev/mapper/ubuntu--vg-swap_1 none          swap      sw          0          0
/dev/vg_tempdb01/lvol0 /var/opt/mssql/data/tempdb01 xfs       defaults,nobarrier 1 2
/dev/vg_data01/lvol0 /var/opt/mssql/data/data01 xfs       defaults,nobarrier 1 2
/dev/vg_log01/lvol0 /var/opt/mssql/data/log01 xfs       defaults,nobarrier 1 2
```

Figure 3-3: /etc/fstab Mounted Disks

Mounting the disks via the /etc/fstab file ensures that the disks will be properly mounted at each system reboot.



```

root@sqllinuxubuntu:~# df
Filesystem                1K-blocks    Used Available Use% Mounted on
udev                      8195832      0   8195832   0% /dev
tmpfs                     1643208     9076   1634132   1% /run
/dev/mapper/sqllinuxubuntu--vg-root 23627644 2857708   19546668  13% /
tmpfs                     8216028      0   8216028   0% /dev/shm
tmpfs                      5120         0     5120     0% /run/lock
tmpfs                     8216028      0   8216028   0% /sys/fs/cgroup
/dev/sdd2                  483946     66338   392623   15% /boot
/dev/mapper/vg_tempdb01-lvol0 209608708 32960 209575748   1% /var/opt/mssql/data/tempdb01
/dev/mapper/vg_log01-lvol0 262011908 32960 261978948   1% /var/opt/mssql/data/log01
/dev/mapper/vg_data01-lvol0 524027908 32960 523994948   1% /var/opt/mssql/data/data01
/dev/sdd1                   523248      3400   519848   1% /boot/efi
tmpfs                     1643208      0   1643208   0% /run/user/1000
root@sqllinuxubuntu:~# █

```

Figure 3-4: Mounted Disks in Linux

By leveraging the LVM, these disks can be expanded within your hypervisor/SAN, and then expanded inside the Linux OS without disruption to your workloads.

At this point, move your SQL Server-on-Linux default paths for new objects to these locations, and move any databases that you currently have on the system to take advantage of the extra I/O channels at this point.

[Find further information on Linux best practices.](#)

Voila! You're now running SQL Server on the Linux operating system. At this point, it is business as usual for SQL Server DBAs, as the SQL Server engine is the same as on Windows.

## Up Next

Now that the infrastructure of Linux been highlighted, let's review the first truly massive scale-out solution for SQL Server – Big Data Clusters.



## CHAPTER 4:

## Big Data Clusters and Storage

Over the years, Microsoft has added feature after feature to the core platform to boost performance for all sorts of workloads. These features, including In-Memory OLTP and columnstore indexes, have allowed the core relational engine to scale to unprecedented levels and service a greater number of organizational needs.

However, there are limits to how far a single instance of SQL Server can scale. For modern workloads, a scale-out method of analytics is required to help the SQL Server engine accommodate these demands.

With its 2019 SQL Server release, Microsoft released a new feature: Big Data Clusters (BDC). The foundation for this new feature was set with the addition of Linux support and container support. These updates made improved automation possible, and enabled features like BDC.

Essentially, Microsoft has built a SQL Server platform that can scale outwards to multiple compute platforms rather than upwards on a single server. Integrations with Apache Spark and the Hadoop Distributed File System (HDFS) allow you to build an analytics platform that can scale as needed for increasingly diverse business workloads.

Microsoft calls this platform “data virtualization.” Integration, analysis, and scale become seamless without all of the historical complexity and challenges of traditional ETL and analysis processes.

### Scale-out Compute Platform

Kubernetes is an open-source platform that helps administrators automate, scale, and manage containerized applications.

With Kubernetes for orchestration, groups of containers can scale out and self-manage to provide structure for the Big Data Cluster. All of these components make up the foundation for Big Data Clusters.

First, let’s understand the terminology of the platform.

A Kubernetes **“node”** is a virtual or physical machine that contains the compute resources needed to support one or more containers. It provides an interface for the container platform to use underlying compute and interconnect resources, such as networking.

A **“pod”** is a basic execution unit of a Kubernetes work. It represents the smallest unit in the Kubernetes object model, and one or more processes in the Kubernetes cluster. The pod contains the container, storage resources, and a network assignment, in addition to parameters that dictate how the container should function. A pod can contain as few as one container, or it can contain multiple containers that function as one unit of work. One or more pods can be hosted on a node.

A **“pool”** is a group of pods that act together to perform a single function. Each member contains the same configuration as the other members. Big Data Clusters contain four types of pools.



A **“compute pool”** is a group of SQL Server pods that are used for parallel consumption of data from either external sources, such as Oracle, HDFS, or another SQL Server, and for cross-partition management of data to fulfill a query request.

A **“storage pool”** is a group of pods containing SQL Server instance, HDFS data, and Spark containers. This pool provides a scalable storage tier along with accompanying SQL Server and Spark engines in close proximity to the data for performance.

A **“data pool”** is a group of SQL Server pods that are used as a caching layer for incoming data. The data is partitioned and sharded across all SQL Server containers in the pool for performance.

The **“master pool”** is a single pool of SQL Server pods that acts as the control system for the various other pools. It can be a single SQL Server instance or an Always On Availability Group for improved high availability and read scale-out. It acts as if it were a regular SQL Server instance.

Most interaction with the Big Data Cluster is performed through the master instance.

Where does the storage layer come into play? The HDFS is deployed as part of the setup, and HDFS becomes a shared persisted store for both SQL Server and other data to create a data lake without having to export data out of SQL Server into other platforms. Aggregating all the various sources of data your organizations needs to process, analyze, and serve to your consumers allows this platform to become a data hub.

The Apache Spark runtime is co-located with the HDFS data pool. Various Spark features, such as SparkSQL, Dataframes, MLlib, GraphX, and more can be leveraged as a central Big Data hub.

All of these layers and objects come together to provide for a tremendously scalable analytics platform.

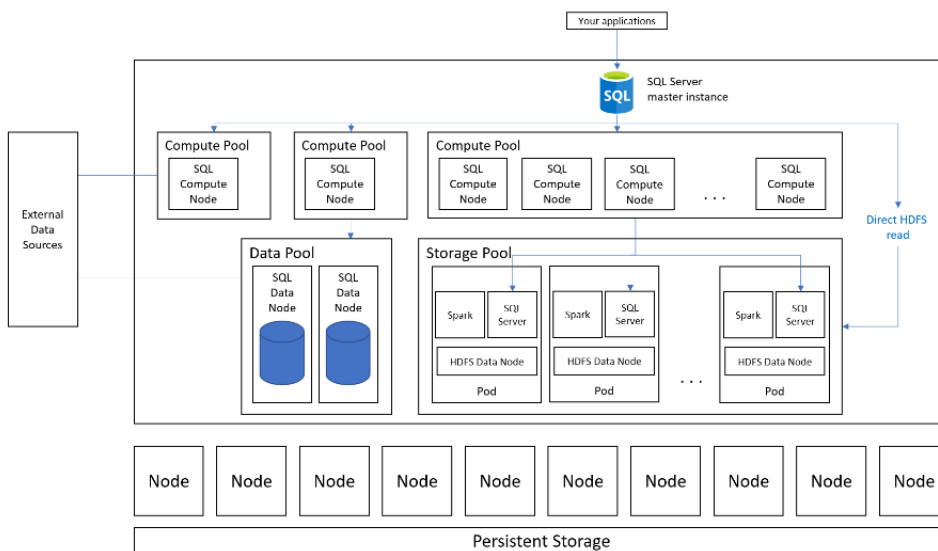


Figure 4-1: Big Data Cluster Architecture





## **BDC and Storage**

Now that SQL Server can scale out the analytics and compute workload to as many compute resources as necessary, optimal storage performance is more critical than ever.

The Hadoop Distributed File System (HDFS) storage architecture is a distributed file system that provides ultra-high-performance access to data spread out across the cluster.

The HDFS storage pool inside Big Data Clusters can be directly queried by the SQL Server components, allowing the storage pool to be used to store any sort of data, including “big” data sets from various external sources. This data source can then be combined with relational data to provide a means to integrate the data for a seamless and cohesive analysis storage platform.

Microsoft and Pure’s vision is to provide a platform where all necessary data resides in one key platform instead of numerous dissimilar and disparate platforms.

## **Highest Performance Possible**

The overarching goal of the BDC platform is to provide the easiest way to create a lightning-fast analysis platform. As with any other technology, the platform is only as fast as the slowest component. Traditional storage is historically the slowest component of any SQL Server deployment. Modern SQL Servers, including BDCs and their intense workloads, will be largely held back and much less effective in their missions if the underlying storage continues to be a significant bottleneck to performance.

The fastest possible storage should be used to provide the most responsive base platform for BDC deployments. Given the investment of the BDC itself, an organization wants to get the most performance out of the platform.

Pure Storage provides some of the world’s fastest and most scalable storage solutions ideal for BDC workloads, including product lines such as FlashBlade® and FlashArray™.

## **Up Next**

Moving data into and out of the cloud for hybrid deployments, effectively making the hybrid component platform differences seamless, has been a significant barrier to hybrid cloud adoption. Modern SQL Server deployments, including both hybrid cloud adoption and simply replicated SQL Servers for disaster recovery, can leverage Pure’s Cloud Block Store to rapidly move data back and forth as needed.



## CHAPTER 5:

## Block Storage for Cloud-Based Workloads

For the majority of organizations, effectively managing and using data resources is essential to creating business value. However, over 40% of enterprises don't feel they have a clear strategy for managing data across multiple clouds, and over 60% feel their organization doesn't yet know what data they'll need to take full advantage of emerging technologies that could potentially fuel the next era of their digital transformation. A recent survey shows that more than 51% of enterprises plan to leverage multicloud<sup>1</sup>, but if you dig into the numbers, you'll find that it's largely block-based workloads running in the cloud for usages like databases, Microsoft Exchange, RAID, and VMs.

The volume of block-based workloads is increasing on AWS, Azure, and GCP, and the use case that's accounting for the largest part of that growth is disaster recovery (DR). Replacement of DR site(s) is a primary driver of public cloud storage usage. Disaster recovery is just one part of a business continuity and lifecycle data management strategy that also includes data protection, real-time replication, and failover.

Pure Storage has recently extended its on-premises storage into the public cloud, incorporating all of the benefits that make Pure a compelling platform for hybrid cloud and multicloud deployments.

### Pure Cloud Block Store™

A hybrid cloud or multicloud strategy gives organizations the flexibility to navigate the limitations of any one data domain/platform, and it gives employees the best environment for innovation. Hyperscalers are introducing a wide variety of innovations to the market in the form of microservices and algorithms, which many next-generation application developers are looking to leverage. The advent of containers has also given organizations a reason to use cloud services to quickly stand up environments to accelerate development efforts.

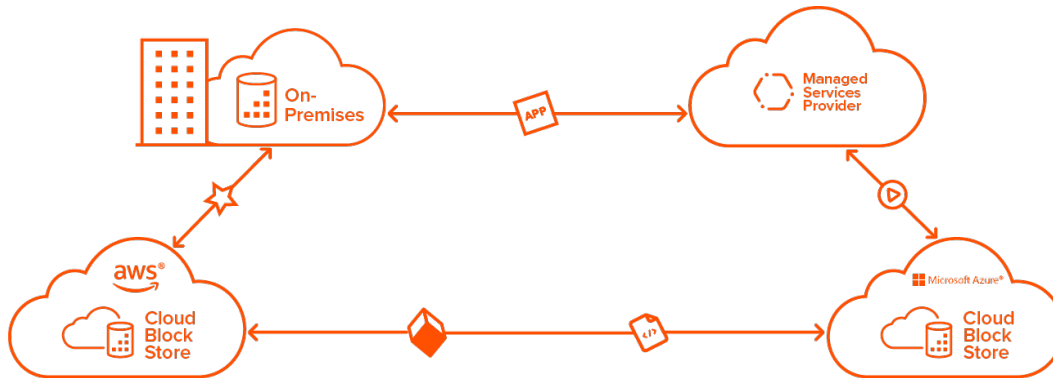
A primary challenge with cloud migration is that cloud and on-premises environments have very different characteristics. Cloud environments have unique services, APIs, cost models, performance characteristics, and architectures, so it's challenging to run the same applications in different environments. To address this challenge, Pure Storage introduced Pure Cloud Block Store. Pure Cloud Block Store bridges the gap between on-premises and cloud environments by providing a common data services layer across on-premise and cloud that abstracts applications and data from their underlying hardware, giving organizations a consistent experience regardless of where their data lives.

Pure Cloud Block Store delivers seamless data mobility across all environments—on-premises, cloud, hybrid cloud, and multicloud. Seamless data mobility enables organizations to strategically manage their IT resources, to promote better collaboration through the elimination of data silos, and to deploy applications faster through DevTest flexibility and efficiency.

---

<sup>1</sup> Critical success factors to achieve a better enterprise data strategy in a multi-cloud environment," Harvard Business Review, 2019 - MC214700919





**Figure 5-1:** Pure Cloud Block Store delivers seamless bidirectional data mobility and consistent experience across all data domains.

A key benefit of Pure Cloud Block Store is the consistency of experience created by using the same Purity framework/software as on-premises Pure Storage arrays. All of the features that organizations have come to rely on in their on-premises data centers are now present in Pure Cloud Block Store with the same management interface that organizations are used to.

## Architecture

Pure Cloud Block Store deploys in the cloud as a software-defined storage solution, offering:

- **Industry-leading cost efficiency:** The Purity Operating Environment enables organizations to reduce the underlying cloud resources required to store their data. Pure Cloud Block Store deduplicates and compresses data before reaching the cloud. Organizations also benefit from instantaneous snapshots, which don't consume additional storage.
- **Data protection:** When organizations deploy mission-critical applications, they must ensure that the applications are resilient to single points of failure. Pure Cloud Block Store offers built-in protection against multiple cloud storage failures using RAID-HA. Pure Cloud Block Store also incorporates spread placement groups into its architecture, reducing physical fault domains. For additional data protection and business continuity, organizations can replicate data between availability zones and regions.
- **Simplicity:** Using a simple CloudFormation template, organizations can deploy Pure Cloud Block Store in minutes and in a cloud-native manner. Pure Cloud Block Store extends Pure FlashArray features to the cloud, giving organizations the same simplicity and consistent user experience.
- **The ability to avoid application refactoring/redesign:** A major obstacle of cloud migration is requiring developers to redesign applications to use different APIs between various cloud environments. Pure Cloud Block Store delivers a common data services layer, enabling developers to use the same APIs on-premises, in the cloud, hybrid cloud to multicloud.
- **Data mobility:** Pure Cloud Block Store's native replication features enable organizations to easily copy or move data for disaster recovery, DevTest, and backup for business continuity.
- **Enterprise compatibility:** Pure Cloud Block Store provides the enterprise features that IT organizations expect in an on-premises environment, including instantaneous snapshot creation/restores.

Deploying Pure Cloud Block Store is straightforward via a CloudFormation template in AWS and as a Managed Application in Azure. You'll need to deploy it via the AWS/Azure Marketplace.



The CloudFormation template consists of two controller VMs in a single Availability Zone. These controllers are then connected to multiple EC2 instances with NVMe instance stores (called the virtual drive) and leverage spread placement groups for redundancy. The data is then mirrored to Amazon S3 to provide the highest level of redundancy available in AWS—11 9s.

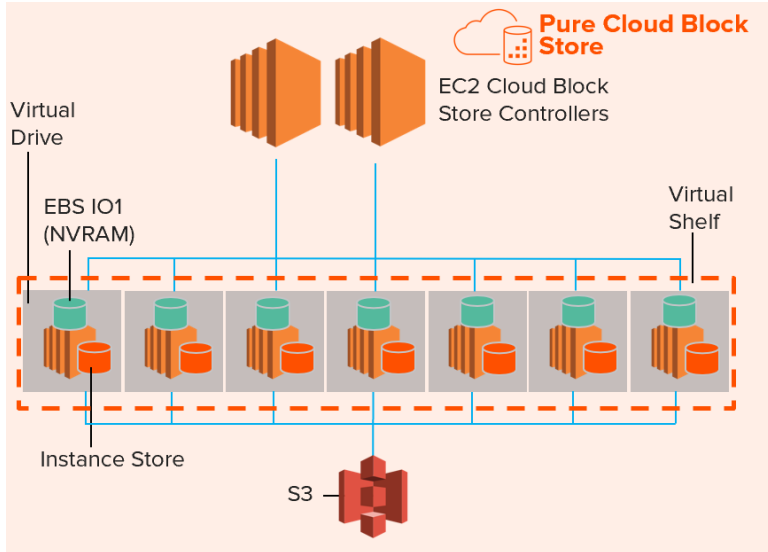


Figure 5-2: Pure Cloud Block Store Architecture for AWS

The architecture can then be mirrored to another Pure Cloud Block Store deployment in a different Availability Zone for even higher availability using Pure Storage Purity ActiveCluster™. The Pure Cloud Block Store architecture on Azure is similar to that on AWS. Pure Cloud Block Store will also run seamlessly on Azure shared disks, a new feature for Azure managed disks that enables you to attach a managed disk to multiple VMs simultaneously. Attaching a managed disk to multiple VMs allows you to get high availability within a single virtual network without having to double your storage footprint.

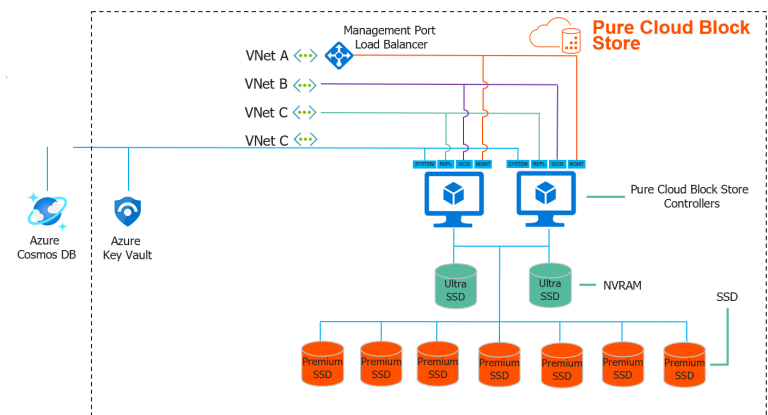


Figure 5-3: Pure Cloud Block Store Architecture for Azure



## Bidirectional Replication

One substantial challenge with cloud solutions has been the act of getting the data to the cloud to begin with. Every application and virtualization technology has its own methods to transfer data and keep it up to date. Managing those layers of replication has always been left to the individual application owner, and a cohesive method for ensuring that all necessary data was replicated within the organizational SLAs was nonexistent. A seamless method was needed to ensure that all of the data not only gets replicated to the cloud but also allows for the same ease to replicate the data back from the cloud.

The Purity Operating Environment, present on all Pure Storage devices, allows for direct and transparent LUN-level block replication to and from the cloud-based Pure Cloud Block Store in the same management interface. This replication allows administrators to seamlessly replicate selected data to the cloud, either in a continuous stream for disaster-recovery purposes, or as a migration strategy to get the data there as a set to be used in the cloud. Replicating back from the cloud also becomes easy—just flip the replication stream backwards to move it from the cloud back to on-premises.

## Data Savings

Ordinary cloud-based storage provides for basic storage functionality only. Pure provides significant advantages in the space consumed on the cloud platform by maintaining the data savings features from the on-premises arrays in the cloud.

Multiple copies of the databases are usually present in enterprise environments. Databases are deployed for production use and then cloned for preproduction development tasks. Traditional cloud storage will incur charges for each copy of the data being stored. The data savings that Pure features, including in-line deduplication and compression, will reduce the footprint of the data and greatly reduce the amount of storage required in the cloud. The storage performance improves due to having to read and write less data to disk, and the overall space consumption in the cloud is reduced and therefore costs less to operate. Snapshots also consume no additional space in the cloud, the same as on-premises.

Additionally, the data replication stream to and from on-premises is also greatly reduced through these data savings features, meaning your cloud egress and ingress costs are significantly lower from the reduced amount of data being transferred.

The savings in the amount of data being transferred can also result in improved replication SLAs and needing less peak bandwidth to your cloud provider.

## Always-On Encryption

With critical data in the cloud, security should be top of mind. Pure Cloud Block Store features Pure's native disk-level encryption at rest, which helps organizations maintain significant levels of security compliance for regulations such as GDPR, HIPAA, and PCI.

SQL Server can leverage its own features for encryption, such as Transparent Database Encryption (TDE) and Always Encrypted, but based on your organizational requirements for database encryption, the cloud-native, encryption-at-rest capabilities of Pure Cloud Block Store mean that you could simplify your architecture and improve security at the same time.



## Flexibility and Agility

Traditional cloud storage is one of the least flexible components in public cloud providers. Expanding a disk became possible only recently on some cloud providers while shrinking a disk requires replacement.

The Pure Cloud Block Store architecture, on the other hand, allows for the various components underneath to be upgraded as cloud providers improve or introduce new offerings, all without disruption to the operations of the workloads on the storage array itself.

## Availability

Single points of failure are common fatal design flaws for enterprise architectures. SQL Server features such as Availability Groups are designed for reducing the risk by improving the high availability of the database platform. Pure factored the availability of the architecture into the Pure Cloud Block Store platform. ActiveCluster, the same feature used to provide high availability across data centers in metro distances, was extended into the Pure Cloud Block Store platform. As a result, you can build the same level of high availability into the cloud across multiple Availability Zones in the same cloud region.

## Use Cases

### Disaster Recovery

Cloud economics, automation, data replication, and recovery orchestration have made DR to the cloud ideal for most organizations, especially for those that lack the resources for a secondary site. Gartner reports that 76% of companies it surveyed experienced an incident in the past two years requiring a DR plan, while over 50% experienced at least two incidents<sup>2</sup>. Unplanned downtime results in excessively long recovery times, a higher potential of unrecoverable data, and lost revenue.

Pure Cloud Block Store enables you to replicate to the cloud in a cost-efficient manner, and its always-on encryption combined with cloud-native cybersecurity provides a compliant solution that safeguards data while preserving its integrity. Businesses also experience continuous uptime, better agility, and decreasing costs.

### DevTest

Developers need to try out many “what-if” scenarios when building software, and leveraging the cloud to build a DevTest environment gives them a powerful capability. Infrastructure replication enables you to instantly test new apps, patches, and updates. You also need to create an environment that’s identical to the production configuration and delivers a consistent user experience to accelerate inner-loop development.

Pure Cloud Block Store enables you to make snapshots and clones in the cloud, meaning you don’t have to purchase and deploy new on-premises hardware. Furthermore, Pure Cloud Block Store snapshots and clones consume negligible cloud storage, benefiting the business through better software time to market and TCO.

## Up Next

In the next chapter, we’ll explore a major advancement in hybrid cloud management from Microsoft called Azure Arc. We’ll look at its architecture and discover why storage performance matters.

---

<sup>2</sup> Survey Analysis: IT Disaster Recovery Trends and Benchmarks," Gartner, April 30, 2020- ID G00719257



CHAPTER 6:

# Microsoft Azure Arc and Modern Storage

One of the largest advantages of the public cloud has always been the plethora of self-service capabilities that allow end-users to rapidly deploy and manage cloud-based objects, such as SQL Server databases. The speed at which items could be created, migrated, and destroyed was a true game-changer that helped IT move at the speed of business.

Conversely, one of the primary disadvantages to true private cloud adoption in the on-premises data center is the lack of these same management tools. Deploying new servers often means submitting deployment tickets and waiting until someone in the IT department receives the request and provisions your object.

Until now. Recently, Microsoft unveiled Azure Arc. With Arc, the Azure management platform can be extended to your on-premises data center. Arc allows for a single management interface so you can manage a truly hybrid environment, including both Azure-based and on-premises data centers.

## Azure Arc Architecture

The Microsoft Azure platform contains a control plane system called the Azure Fabric Controller. This is responsible for managing the lifecycle of services, such as VMs and databases. Every service continuously reports its state to the Fabric Controller. The Azure Resource Manager (ARM) manages the service so that the desired state of the resource can be automated and managed, for example when determining the scale of a virtual machine.

The Azure Arc service is a logical extension of the Azure Control Plane. Arc allows ARM to connect and manage compute resources outside of the Azure infrastructure, including those in your data center.

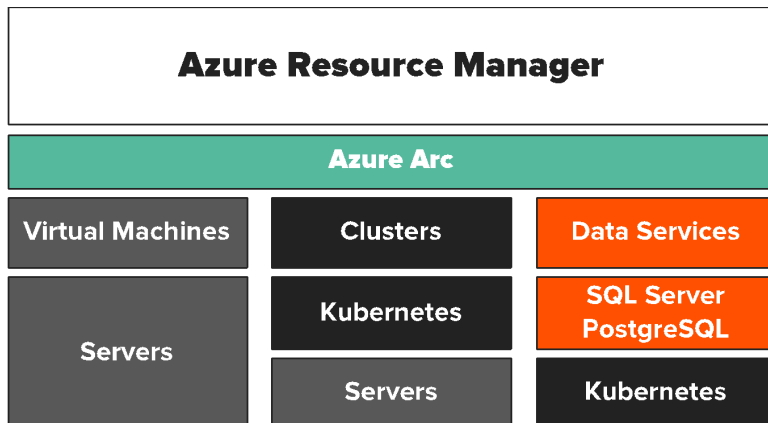


Figure 6-1: Azure Arc Control Plane

One of the first services to support Azure Arc was SQL Server databases. Azure SQL Databases, SQL Managed Instances, and PostgreSQL Hyperscale can now be managed by Arc, but reside in your data center.

The benefits of this are numerous. The management layer can now use a single interface or API to manage all of these services. Automated updates and patching, security scans, and seamless updates can now be handled by the Azure data automation service.



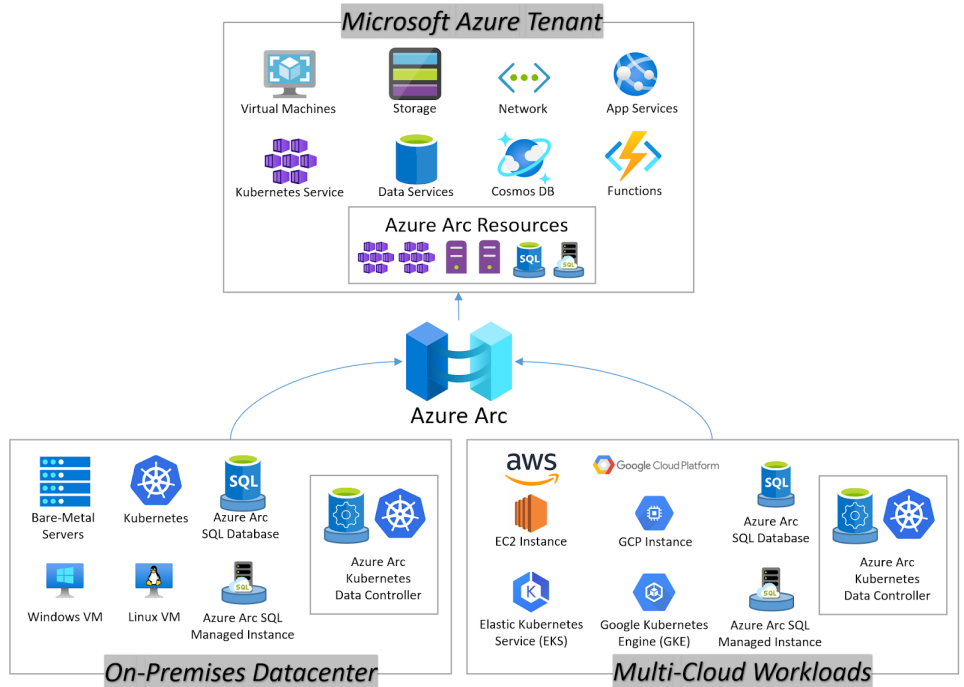


Figure 6-2: Azure Arc Hybrid Infrastructure

## Storage Speed

With any database platform, in the cloud or on-premises, the performance of the storage subsystem is crucial to maintain database-level performance. Azure Arc is no different in that regard. However, where these platforms differ is that the storage speed of the underlying platform. This is critical to ensure the rapid response of the deployment and management framework while also maintaining the speed of the databases that reside upon it.

As an organization adopts the concept of self-service and rapid deployment, demand on the platform increases. And with that increased demand comes increased usage. A storage subsystem must be capable of not only providing peak performance for each workload after deployment once it is in service but also providing the fastest possible platform so that the deployment and management layer for these objects meets business expectations.

Organizations running Azure Arc on Pure Storage® receive all of the benefits of the single control plane of Azure services in their data center with the benefits of Pure Storage. These include aggressive data savings, encryption-at-rest, and SafeMode Snapshots. Combining all of these benefits creates an incredibly powerful platform that can improve business velocity.





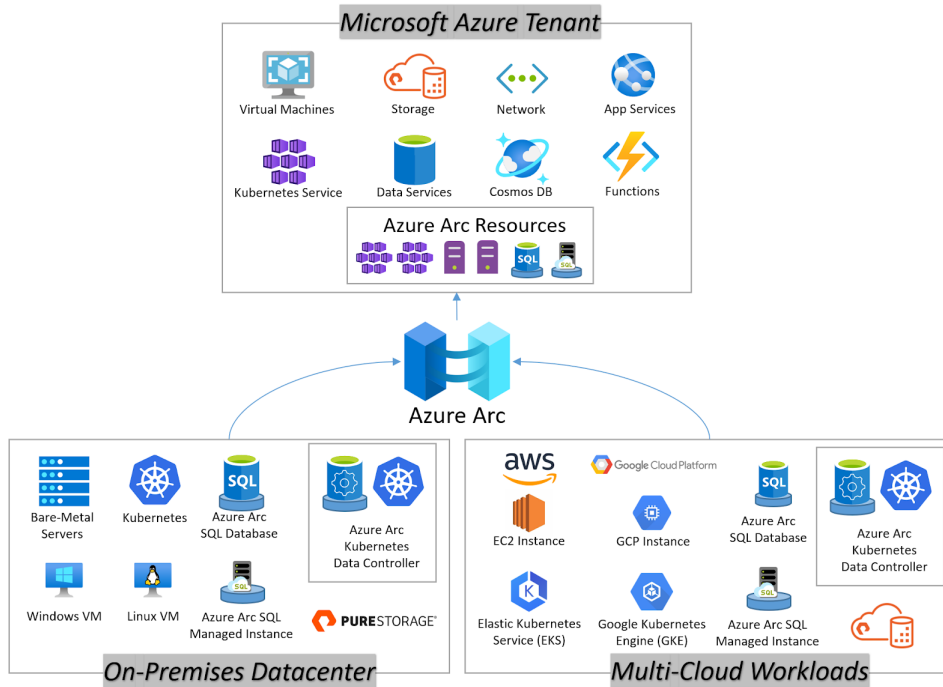


Figure 6-3: Hybrid Infrastructure with Pure Storage

## Up Next

Now that we've reviewed all the options of modern platforms that SQL Server can reside on and use for improvements in management and orchestration, next up is the first SQL Server in-memory feature: In-Memory OLTP.

Next, we'll review the architecture and performance characteristics of this flagship performance feature. We'll also review the demands on infrastructure and the considerations that must be designed into any critical In-Memory OLTP workload.



## CHAPTER 7:

## In-memory OLTP

Microsoft's flagship in-memory technology, In-Memory OLTP, was created out to improve the performance of the SQL Server engine by reducing the historical assumptions that memory quantities are smaller than the amount of data to process. The design of the database engine, or at least the design of a new engine to be embedded inside the SQL Server engine, would actively hold all its data within server memory, and only persist changes to disk as necessary.

### Concept

SQL Server's In-Memory OLTP feature is an engine within the main relational database engine. This allows developers to actively store and manipulate the data in memory, rather than stick with the assumption that all data is on disk as with the primary SQL Server engine.

This engine is not the same approach as simply forcing all the data within a table into memory. Administrators have been trying various tricks and workarounds for years to try and keep all of the working data in RAM, but that only solves half of the challenge. The concurrency engine design of the regular OLTP engine still requires the use of locks and latches to manage activity, which maintains the dependency on disk.

Two key components make up the new In-Memory OLTP engine. The first component includes the data containers and the memory-optimized tables and indexes. The second component includes the optimized means to fetch and manipulate the data and natively compiled stored procedures.

### Memory-Optimized Tables and Indexes

The largest single change between the standard SQL Server storage engine and In-Memory OLTP is that the entire table is stored in memory—always. You cannot store more data than what you possess in server memory.

Because the entire set of data is now always in memory, the concurrency challenges of the traditional on-disk model change, by nature of this new technology. The concurrency model is now truly optimistic. Locking and latching mechanisms are not needed because SQL Server makes changes to data by creating a new row for the change and deleting the original row. This is opposed to updating the row in-place and having to worry about other users accessing the same data at the time of the change.

The tables are also now natively compiled into machine code. Every construct necessary to manage and access that table, including metadata and schema, are now optimally stored and accessed by the engine. This ensures the usage of these tables is the most efficient.

What happens when a user changes data? And what if that change is important for the business to keep? In-memory tables can be created in two ways. The first is to persist all data changes to disk, or `SCHEMA_AND_DATA` durability. If a process is deemed transient or it's unnecessary to persist the changes, such as with temporary operations, the table can be created with `SCHEMA_ONLY` durability. Only the table schema persists between service startups, and any existing data is lost at service shutdown. These tables are beneficial for transient processes that can be restarted or temporary tables that are part of a larger operation.



For those operations that must persist, the data change must be written to disk. SQL Server uses the same transaction log structure that the standard engine uses to write these transactions to disk but writes single log records only when the transaction commits. This reduces the strain on the logging process.

### **In-Memory Indexes**

What good is storing all the data in memory if you must sift through all of it, sometimes repeatedly, to find the data that the user requests? SQL Server uses indexes in the regular engine to help improve the speed of finding the specific data the user requests. New indexes were created for In-Memory OLTP, and these indexes are also entirely contained within server memory.

Two types of indexes are used in In-Memory OLTP: hash indexes and range indexes. A hash index is useful when performing lookups against individual records. A range index is now available for retrieving ranges of values. Traditional index types are not available in in-memory tables.

### **Natively Compiled Stored Procedures**

The stored procedures accessing the data in these memory-optimized tables can also be natively compiled into machine code, which reduces the overhead within the query engine layer. Regular stored procedures are compiled when they are interpreted at their first execution time. Calling the natively compiled stored procedure, existing as a DLL, results in significant performance improvement.

### **Memory-Optimized TempDB Metadata**

One major enhancement in the recent 2019 release is the introduction of memory-optimized TempDB metadata. In-Memory OLTP can now be used for each of the system tables inside the TempDB database. The new latch-free non-durable memory-optimized system tables now provide a much-needed boost for highly concurrent workloads that leverage the TempDB database quite extensively. The results are that much less metadata locking and blocking occurs, yielding more performance gains.

The use of this feature is an opt-in and can be enabled with the following script:

```
ALTER SERVER CONFIGURATION SET MEMORY_OPTIMIZED TEMPDB_METADATA = ON;
```

Please note that columnstore indexes cannot be created on temporary tables when Memory-Optimized TempDB metadata is enabled.



## Current Limitations

The SQL Server In-Memory OLTP engine is effectively an all-new database engine that just happens to be embedded inside the existing engine. As such, it is not feature-complete with the primary engine at the time of this writing. The SQL Server 2014 launch feature set shipped with many omissions or partial implementations of key items that developers use by default. The recent 2019 release greatly improved its features over the initial 2014 release, but several limitations still exist that might make adoption of In-Memory OLTP difficult (or impossible). The following partial list of limitations is present in the 2019 release:

- Database integrity checks
- Partitions
- Replication
- FILESTREAM
- Clustered indexes
- Truncated table
- DROP INDEX
- Common Table Expressions
- Some traditional built-in functions
- Once added, the in-memory filegroup cannot be removed from the database

This list of limitations is improving with each release. See a full list of the current limitations [here](#).

## Setting It Up

Let's set up a sample database for use with In-Memory OLTP. See [the full set of scripts for this operation](#).

We will create it with a separate filegroup for the In-Memory objects, required by the engine.

```
CREATE DATABASE CMSMem
ON
  PRIMARY (NAME = [CMSMem_data],
    FILENAME = 'G:\Data\CMSMem_data.mdf',
    SIZE=500MB),
  FILEGROUP [CMSMem_InMem]
    CONTAINS MEMORY_OPTIMIZED_DATA
    (NAME = [CMSMem_InMem1],
    FILENAME = 'G:\Data\CMSMem_InMem1'),
    (NAME = [CMSMem_InMem2],
    FILENAME = 'H:\Data\CMSMem_InMem2')
LOG ON (name = [CMSMem_log],
  filename='L:\Logs\CMSMem_log.ldf',
  SIZE=500MB);
```



Note the new filegroup that contains the syntax 'CONTAINS MEMORY\_OPTIMIZED\_DATA.' This new filegroup is specified to contain memory-optimized data and is configured for spanning two data files on two different disks to spread out the workload on multiple disk controllers.

Now, set the database to move all isolation levels to snapshot.

```
ALTER DATABASE TestDBInMem SET
MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT = ON;
```

For comparison's sake, we will also create a logically similar database called CMS with no In-Memory OLTP active in the database. It will contain the same logical schema and data, and we will test certain operations between these two databases. The scripts for creating this database are also located [here](#).

To demonstrate the power of this feature, I want a real-world example rather than a contrived example. Let's create a sample scenario to develop against. Say you want to query people living in different states in the USA.

First, create a table to store their information.

```
CREATE TABLE dbo.People (
  [ID] [int] IDENTITY(1,1) NOT NULL ,
  [Name] varchar(32) NOT NULL
  INDEX IX_People_Name HASH
  WITH (BUCKET_COUNT = 75000000),
  [City] varchar(32) NOT NULL
  INDEX IX_People_City HASH
  WITH (BUCKET_COUNT = 50000),
  [State_Province] varchar(32) NOT NULL
  INDEX IX_PEOPLE_State_Province HASH
  WITH (BUCKET_COUNT = 1000),
  PRIMARY KEY NONCLUSTERED HASH (ID) WITH
  (BUCKET_COUNT = 75000000)
) WITH (
  MEMORY_OPTIMIZED = ON,
  DURABILITY = SCHEMA_AND_DATA);
```

Note the syntax changes of the table creation command. We are creating a hash index on some of the columns. A hash index is an array-based index of a certain number of slots, with each slot pointing to the memory address of a particular row in the table. We also want this data to persist between reboots, so the durability parameter is set to 'SCHEMA\_AND\_DATA' instead of just 'SCHEMA.'

One major point of misunderstanding is with the BUCKET\_COUNT value. The bucket count roughly represents the number of index positions in the index. The hashing function used to identify the unique slots is good but not perfect, and hash collisions will happen, which means that two different keys can hash to the same bucket value. In-Memory OLTP solves this by chaining the two with updated index pointers. Thus, most tables should use a bucket count value of roughly 1.5 to two times the number of unique items that you expect to be loaded into this table. I want to load a significant number of fake users into this table, so I set the bucket count to 75 million. I only have a small number of cities, states, and countries in the geographical tables in the database, and as such, the bucket counts for these items will be much smaller.



Keep in mind that this setting directly relates to the size of the hash index. The calculation for this value is:

$$[\text{Hash index GB}] = \frac{(8 * [\text{Actual Bucket Count}])}{1073741824 \text{ (Bytes in 1 GB)}}$$

To identify the number of buckets of the in-memory indexes, execute the following query:

```
SELECT
    OBJECT_NAME(s.object_id) AS TableName,
    i.name as IndexName,
    s.total_bucket_count
FROM
    sys.dm_db_xtp_hash_index_stats s
    INNER JOIN sys.hash_indexes i
        ON s.object_id = i.object_id
        and s.index_id = i.index_id
ORDER BY
    TableName, IndexName
```

## CMS Database

The remainder of the CMS database is straightforward in its nature.

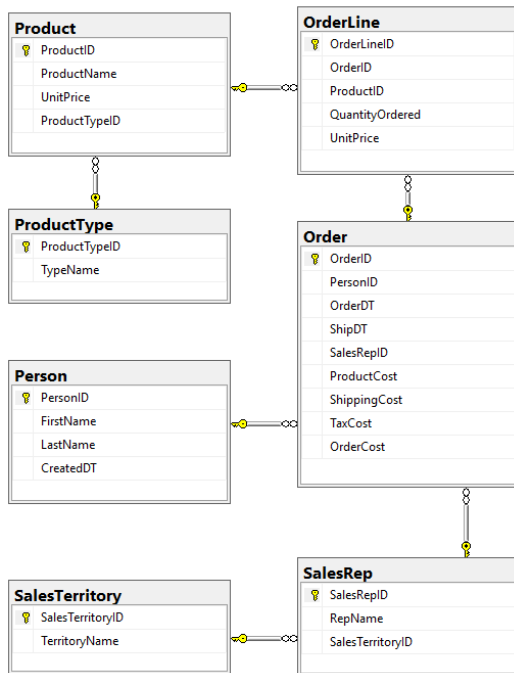


Figure 7-1: CMS schema

Orders are placed using line items consisting of products, each with their own product type. People are the recipients of the orders. Sales representatives assist with and enter the order details, and each sales rep has a sales territory (in this case, states within the United States).

To load the data into the database, a stored procedure called `dbo.Order_Generate` was created to load randomly generated data into the CMS database schema. Once loaded into the on-disk database, staging tables were used to copy the data into the In-Memory OLTP database. This process of loading and cloning the data takes some time, so backup each database once this operation is complete for future database reuse.

The sample testing was performed with 12,000 products, 5.2 million orders, 265 million order line items, and 5,000 sales representatives. All test operations performed for this document were performed on a SQL Server 2019 Enterprise Edition Cumulative Update 4 virtual machine with 24 vCPUs and 128GB RAM.

## Potential Performance Improvement

First and foremost, simply using In-Memory OLTP is not guaranteed to improve the performance of your application. In some cases, it can even substantially hinder your performance. However, when used correctly in the right circumstance, the performance improvements are impressive. For example, querying these tables for product sold count and value for a given month produces a noticeable improvement in performance.

The stored procedure `dbo.Product_By_Date` was created containing a query to produce this sales by date report. The on-disk database contains the following T-SQL:

```
USE [CMS]
GO

CREATE proc [dbo].[Product_by_Date] (
    @StartDT DATETIME,
    @EndDT DATETIME
) AS

SET NOCOUNT ON;

SELECT
    l.ProductID,
    SUM(l.QuantityOrdered) AS SoldCount,
    SUM(l.UnitPrice) AS SoldValue
FROM
    dbo.[Order] o
    INNER JOIN dbo.OrderLine l
        ON o.OrderID = l.OrderID
    INNER JOIN dbo.Person p
        ON p.PersonID = o.PersonID
    INNER JOIN dbo.SalesRep r
        ON r.SalesRepID = o.SalesRepID
    INNER JOIN dbo.SalesTerritory t
        ON t.SalesTerritoryID = r.SalesTerritoryID
WHERE
    o.OrderDT >= @StartDT
    AND o.OrderDT <= @EndDT
GROUP BY
    l.ProductID
ORDER BY
    SoldValue DESC
```



The equivalent stored procedure, constructed for In-Memory OLTP, utilized natively compiled stored procedures to boost the performance of the operation.

```

USE [CMSMem]
GO

CREATE OR ALTER PROC [dbo].[Product_by_Date] (
    @StartDT DATETIME,
    @EndDT DATETIME
)
WITH NATIVE_COMPILATION, SCHEMABINDING, EXECUTE AS OWNER
AS

BEGIN ATOMIC WITH (
    TRANSACTION ISOLATION LEVEL = SNAPSHOT,
    LANGUAGE = N'us_english')

SELECT
    l.ProductID,
    SUM(l.QuantityOrdered) AS SoldCount,
    SUM(l.UnitPrice) AS SoldValue
FROM
    dbo.[Order] o
    INNER JOIN dbo.OrderLine l
        ON o.OrderID = l.OrderID
    INNER JOIN dbo.Person p
        ON p.PersonID = o.PersonID
    INNER JOIN dbo.SalesRep r
        ON r.SalesRepID = o.SalesRepID
    INNER JOIN dbo.SalesTerritory t
        ON t.SalesTerritoryID = r.SalesTerritoryID
WHERE
    o.OrderDT >= @StartDT
    AND o.OrderDT <= @EndDT
GROUP BY
    l.ProductID
ORDER BY
    SoldValue DESC

END;

```

Note that the actual query is identical to the on-disk database. The natively compiled stored procedure requires a few additional commands to successfully compile, such as SCHEMABINDING because of the nature of compiled objects, and BEGIN\_ATOMIC because the command must contain only one operations block.

Executing this stored procedure repeatedly against the on-disk database with the parameters for June 2020 resulted in an average runtime of 24 seconds.

```

USE CMS
GO
EXEC dbo.Product_By_Date '2020-06-01', '2020-06-30'
GO

```





Executing this same stored command against the in-memory database yielded improved results.

```
USE CMSMem
GO
EXEC dbo.Product_By_Date '2020-06-01', '2020-06-30'
GO
```

The performance differences are impressive. The in-memory table, coupled with the natively compiled stored procedure, executed the same query in just three seconds, a nine times improvement.

Additional improvements are seen with many queries against these databases. The same time span was used with these stored procedure calls.

Stored Proc	On Disk Runtime (sec)	In-Memory Runtime (sec)
Product_by_Date	24	3
Product_by_Territory_Date	12	12
Product_Ship_Delays_by_Age_Date	0.353	0.363
Product_Ship_Delays_by_Date	118	0.860
Sales_By_Territory_Rep_Date	0.324	0.282

As you can see, not every scenario resulted in large performance gains, but for those that did improve, the results were substantial.

Additional performance is also exhibited during large updates. The majority of the performance improvements occur when the operations were dependent on the logging to disk. For example, the act of reassigning sales reps can be expensive when on-disk. Thousands of orders need to be updated, and each operation is time-consuming. Performing this same operation on in-memory objects is noticeably less expensive.



```

--Update random sales reps
SET STATISTICS TIME ON
GO
USE CMS
GO
DECLARE @OldRepID INT, @NewRepID INT
SELECT TOP 1 @OldRepID = SalesRepID FROM dbo.SalesRep ORDER BY NEWID()
SELECT TOP 1 @NewRepID = SalesRepID FROM dbo.SalesRep ORDER BY NEWID()
-- So we can transpose the item to the
-- in-memory command for continuity in operations
PRINT @OldRepID
PRINT @NewRepID

EXEC dbo.Sales_Rep_Reassign @OldRepID, @NewRepID
GO
--148ms

USE CMSMem
GO
EXEC dbo.Sales_Rep_Reassign 4652, 1265
GO
--6ms

```

The time saved in this scenario is tremendous. The runtime improves from 148 milliseconds to just 6 milliseconds.

As with any new feature within any application, your experiences with the performance gains (or decreases) will vary depending on your situation. Used wisely, In-Memory OLTP can improve the performance of certain operations by a considerable margin.

## Storage Considerations

The storage considerations when using In-Memory OLTP are substantial. If the table is configured to be non-durable, meaning that the data never persists to disk and is lost if the service or server is restarted, there is little demand on storage. However, it's more often the case that the data must persist to disk, and any change to data must be written to disk before the operation can complete.



Note the durability parameter when creating this in-memory table.

```
CREATE TABLE dbo.People (
  [ID] [int] IDENTITY(1,1) NOT NULL ,
  [Name] varchar(32) NOT NULL
    INDEX IX_People_Name HASH
    WITH (BUCKET_COUNT = 6000000),
  [City] varchar(32) NOT NULL
    INDEX IX_People_City HASH
    WITH (BUCKET_COUNT = 6000000),
  [State_Province] varchar(32) NOT NULL
  PRIMARY KEY NONCLUSTERED HASH (ID) WITH
  (BUCKET_COUNT = 6000000)
) WITH (
  MEMORY_OPTIMIZED = ON,
  DURABILITY = SCHEMA_AND_DATA);
```

The table was created with a durability setting to make both the table definition and the contents of the table persist to disk.

How fast can your storage write these changes to disk? Your in-memory database is now limited to the performance of your disk's ability to write the log buffers to disk.

When considering leveraging In-Memory OLTP in your applications, perform a close examination of your storage's ability to write small blocks of data to disk very quickly. Transactional latency in these operations will slow down any in-memory change operation.

Additionally, the high-performance demands on storage to read all of the data while SQL Server starts up and rebuilds all of the in-memory indexes is enough to cause a significant drain on storage performance resources during this operation. This will cause the SQL Server database to be delayed in starting up.

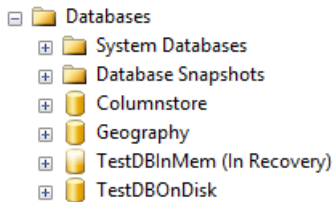


Figure 7-2: In-Memory OLTP Database Startup Delays

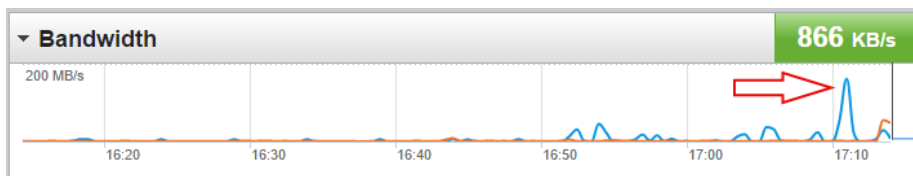


Figure 7-3: Burst Startup I/O Consumption

All-flash storage possesses the high-performance characteristics to be able to accommodate this level of performance with a minimal amount of overhead to the data change operation. Other more traditional types of storage do not contain the equipment needed to sustain these performance levels over time.



## CPU Considerations

No feature to boost the performance of key operations is free. As the bottlenecks shift away from the typical storage bottleneck, they shift upstream in the system stack. If the data is stored in memory and disk-bound operations are no longer the overarching bottleneck to performance, chances are that you will see your CPU consumption elevate as the workload shifts. Sometimes this CPU consumption is substantial, and you do not want to experience CPU pressure during critical time periods.

As you explore In-Memory OLTP, I strongly recommend continuous CPU consumption metric collection. Quite a few third-party applications are on the market to help you collect this information, or you can use the free Windows Server Perfmon utility that comes included with every modern version of Windows. Understand the before and after CPU consumption metrics as you test and adopt this feature. Putting aside the license implications of doing so, it is relatively easy to add CPU resources to a VM or cloud-based database and make note of the effect this change has on system performance. Removing a CPU bottleneck may reveal a new bottleneck elsewhere in the system.

## Memory Considerations

As the bottleneck shifts away from primary storage and all of the memory is leveraged for data storage, the amount and performance of the memory is more important than ever. Above all else, to use In-Memory OLTP for your workloads, you must have more memory than the data you are attempting to store in memory. Beyond the obvious, performance challenges exist with in-memory operations as well.

Many factors contribute to the performance of the memory on your server, including:

- VM memory hot-add enabled (some hypervisors disable in-guest vNUMA presentation when this feature is enabled)
- Misconfigured physical server memory configuration (triple channel memory disabled, or slower than expected memory speed due to suboptimal memory placement on the system mainboard)
- Virtual machine NUMA imbalance
- Virtual machine memory pressure from the hypervisor

Work with your system administrators to ensure that your server is configured for the maximum possible memory performance.

## Up Next

Next up is a deep-dive into how SQL Server columnstore indexes are configured and used, demonstrations on the potential improvements from using them, and the impact on the infrastructure underneath the SQL Server for which the administrators need to monitor and architect.



## CHAPTER 8:

**Columnstore**

Traditional SQL Server indexes (sometimes known as a RowStore) group and store table data for each row and then group the rows together to form an index. For some types of OLTP workloads, where the index necessary to boost performance is not entirely predictable, these indexes work well. However, certain types of workloads, such as larger aggregate reporting and warehouses, can benefit from a different indexing strategy. In these cases, columnstore indexes are better suited for large tables and analytics workloads.

Columnstore indexes are essentially a pivot from a traditional index. Rowstore indexes (the normal indexes you are used to) are stored row by row.

ID	PersonID	OrderDate	OrderAmount
1	10964	01/27/2020 18:29:42	\$5,678.48
2	11674	02/14/2020 04:53:27	\$4,842.68
3	12479	03/21/2020 04:53:14	\$8,226.31
4	13484	04/23/2020 13:28:34	\$7,974.89

**Figure 8-1:** Traditional Rowstore Indexes

In columnstore indexes, data is grouped by column instead of by row. It's stored one column at a time then grouped by rows to form an index.

ID	PersonID	OrderDate	OrderAmount
1	10964	01/27/2020 18:29:42	\$5,678.48
2	11674	02/14/2020	\$4,842.68
3	12479	03/21/2020 04:53:14	\$8,226.31
4	13484	04/23/2020 13:28:34	\$7,974.89



**Figure 8-2:** Columnstore Index

Re-orienting the index per column is more in line with the nature of some workloads, and the improved efficiency can lead to a significant improvement in query runtime. Workloads such as decision support systems, roll-up reporting, and warehouse fact tables can all benefit from columnstore indexes. Microsoft [claims](#) that these workloads can experience up to a ten times performance improvement, all while saving up to seven times the disk space due to high levels of compression versus normal indexes.

## Concept

With the release of SQL Server 2012, Microsoft introduced non-clustered columnstore indexes into the core SQL Server engine. Based on the VertiPaq storage engine's in-memory data compression technology, later renamed xVelocity in-memory analytics engine, columnstore indexes store indexes by column instead of per row. Columnstore indexes use a columnar formatted index structure instead of the traditional B-tree format of row-wise clustered and nonclustered traditional indexes. Each column is stored in separate data pages, so only the data necessary for fulfilling a specific query is accessed.

Each column in the columnstore index is its own segment, and can only store values from that column. A segment is a compressed Large Object (LOB) that is optimized for up to one million rows. Multiple segments can make up a columnstore index. Segments are loaded into memory upon being queried instead of the traditional page or extent.

The xVelocity engine is also optimized for parallel data scanning and can utilize the multi-core processors in modern server hardware. The data stored in-memory is also significantly compressed, so more data can be contained in-memory and less need to be read from disk.

Originally released only for nonclustered indexes, subsequent releases have introduced support for clustered indexes as well. As of SQL Server 2016 SP1, columnstore indexes are now available on Standard Edition. Columnstore indexes are also available in premium Azure databases.

## Configuration and Usage

Columnstore indexes come in two flavors (as of SQL Server 2019): clustered and nonclustered. Creating one is as simple as creating a traditional index. For example, let's create a clustered columnstore index on a simple table:

	Column Name	Data Type	Allow Nulls
▶	PerfDataID	bigint	<input type="checkbox"/>
	ServerID	int	<input type="checkbox"/>
	BatchLoadID	int	<input checked="" type="checkbox"/>
	PerfmonCounterInstanc...	int	<input checked="" type="checkbox"/>
	PerfValue	float	<input checked="" type="checkbox"/>
	CollectionDT	datetimeoffset(2)	<input checked="" type="checkbox"/>

**Figure 8-3:** Clustered Column Store Index

```
CREATE CLUSTERED COLUMNSTORE INDEX [cci_PerfData] ON
[dbo].[PerfData]
GO
```



We do not have a primary key defined on this table at this time, but the column PerfDataID is defined as a bigint IDENTITY column. The syntax to create the clustered columnstore index is as follows:

That's it! Primary keys are not officially needed. If you want to have a unique constraint on a column that is normally enforced by the primary key constraint, create a non-clustered index with a unique constraint on that column(s) and reproduce the same behavior.

Nonclustered indexes are very similar. For the same table, structured with a traditional rowstore index, let's create a nonclustered clustered index on a few of the columns used for reporting. This sample table contains a normal primary key and clustered index.

	Column Name	Data Type	Allow Nulls
🔑	PerfDataID	bigint	<input type="checkbox"/>
	ServerID	int	<input type="checkbox"/>
	BatchLoadID	int	<input checked="" type="checkbox"/>
	PerfmonCounterInstanc...	int	<input checked="" type="checkbox"/>
	PerfValue	float	<input checked="" type="checkbox"/>
	CollectionDT	datetimeoffset(2)	<input checked="" type="checkbox"/>

Figure 8-4: Nonclustered columnstore index

```
CREATE NONCLUSTERED COLUMNSTORE INDEX IX_PerfData_NCCICover01 ON dbo.PerfData
(
    ServerID, PerfmonCounterInstance, PerfValue
)
GO
```

Microsoft gives you plenty of options for use with columnstore indexes. We recommend visiting the [MSDN page](#) before you start creating these in production.

The difference in space consumed by these indexes is also quite impressive. Comparing the three scenarios—rowstore indexes, clustered columnstore indexes, and rowstore primary key indexes—with a nonclustered columnstore index shows the significant disk utilization differences. This query will demonstrate the space consumed by the table and indexes:



```

SELECT
  s.Name AS SchemaName,
  t.NAME AS TableName,
  p.rows AS RowCounter,
  SUM(a.total_pages) * 8 / 1024 AS TotalSpaceMB
FROM
  sys.tables t
INNER JOIN
  sys.indexes i ON t.OBJECT_ID = i.object_id
INNER JOIN
  sys.partitions p ON i.object_id = p.OBJECT_ID AND i.index_id = p.index_id
INNER JOIN
  sys.allocation_units a ON p.partition_id = a.container_id
LEFT OUTER JOIN
  sys.schemas s ON t.schema_id = s.schema_id
WHERE
  t.NAME = 'PerfData'
GROUP BY
  t.Name, s.Name, p.Rows
ORDER BY
  t.Name
GO

```

The results are impressive:

Type	Rowcount	MB
Rowstore	120,436,320	11,104
Clustered Columnstore	120,436,320	714
Rowstore PK and NC Columnstore	120,436,320	7460





The performance differences in the aforementioned reporting and warehousing queries are pronounced as well. A simple aggregate reporting query on this table is as follows:

```

SELECT
    MIN(PerfValue) AS MinVal,
    AVG(PerfValue) AS AvgVal,
    MAX(PerfValue) AS MaxVal
FROM
    dbo.PerfData d
    INNER JOIN dbo.PerfmonCounterInstance i ON
        i.PerfmonCounterInstanceID =
        d.PerfmonCounterInstanceID
    INNER JOIN dbo.PerfmonCounter c ON
        c.PerfmonCounterID = i.PerfmonCounterID
    INNER JOIN dbo.PerfmonCounterSet s ON
        s.PerfmonCounterSetID = c.PerfmonCounterSetID
WHERE
    d.ServerID = 8
    AND s.PerfmonCounterSetID = 23
GO
    
```

Type	Runtime (ms)
Rowstore	4,110
Clustered Columnstore	670
Rowstore PK and NC Columnstore	750

The execution plans between the index types are dramatically different. The rowstore index execution plan is as follows, using SentryOne’s Plan Explorer to visualize the execution plan:

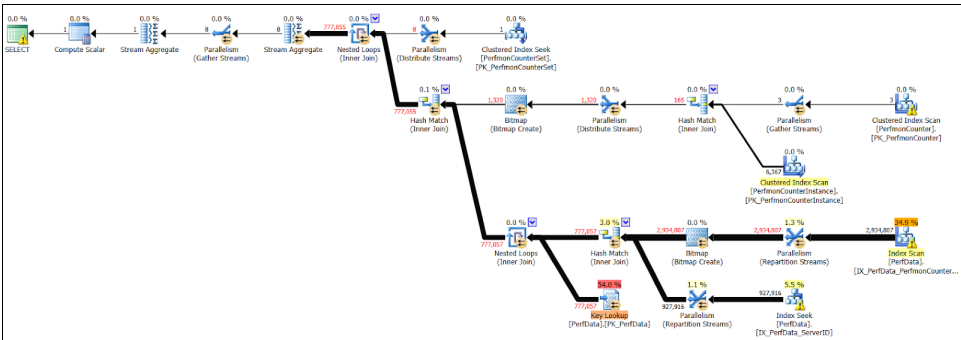


Figure 8-5: Rowstore index execution plan

This execution plan is rather complex, and for this query, SQL Server notes a missing index.



For the clustered columnstore index query, the execution plan is simpler:

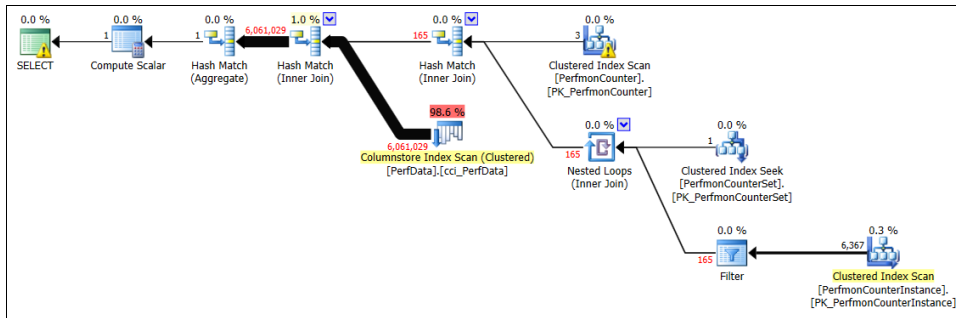


Figure 8-6: Clustered columnstore query execution plan

However, when performing a single value lookup instead of an aggregate query, the performance characteristics change:

```

SELECT
    d.PerfDataID, d.ServerID, d.PerfValue, d.CollectionDT
FROM
    dbo.PerfData d
    INNER JOIN dbo.PerfmonCounterInstance i ON
        i.PerfmonCounterInstanceID =
        d.PerfmonCounterInstanceID
    INNER JOIN dbo.PerfmonCounter c ON
        c.PerfmonCounterID = i.PerfmonCounterID
    INNER JOIN dbo.PerfmonCounterSet s ON
        s.PerfmonCounterSetID = c.PerfmonCounterSetID
WHERE
    d.PerfDataID = 456789
GO
    
```

The runtimes are lower because the working set is in memory, but the differences are present.

Type	Runtime (ms)
Rowstore	5
Clustered Columnstore	23
Rowstore PK and NC Columnstore	4

As you can see, querying the table to use the columnstore index requires nothing new in your query. Microsoft makes the use of this feature transparent to the actual query syntax itself. However, the execution plan varies between index types. For the traditional rowstore index, the execution plan is as follows:



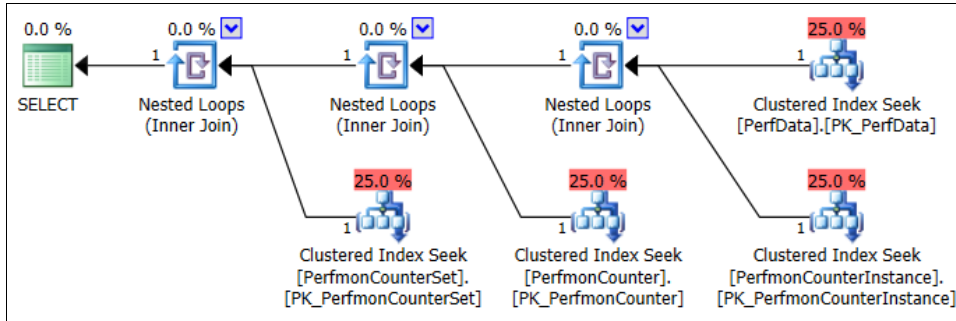


Figure 8-7: Nonclustered columnstore index execution plan

The nonclustered columnstore index query’s execution plan is identical to the above plan. The lookup value was based on the primary key column, which is not in the clustered index.

The clustered columnstore index execution plan is very different.

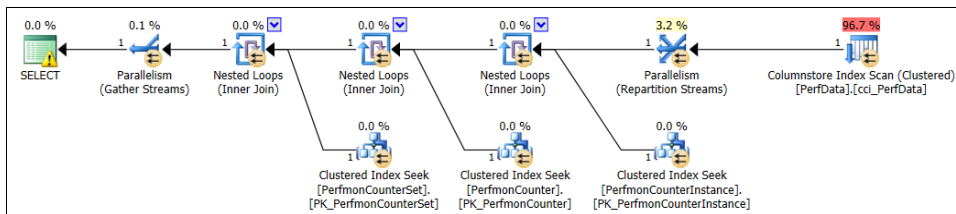


Figure 8-8: Clustered columnstore index execution plan

SQL Server again noted a missing index on the ID column and suggested that the developer place a nonclustered index on that column to improve performance.

**Note:** Test accordingly as you deploy columnstore indexes to ensure that your workload performance responds in a positive manner before you deploy the new indexes to production.

## Storage Implications

While the storage footprint of columnstore indexes is smaller than traditional indexes, the speed of the storage is equally as important as with In-Memory OLTP. Lower latency to disk means that the fetching of the columnstore index segments from disk is improved, and the data is loaded into memory for query processing faster. Unlike In-Memory OLTP, which requires all the data to be resident in memory at all times, columnstore indexes are persisted on disk like normal indexes. They can be pushed out of the buffer pool and will need to be re-fetched as needed. Lower latency and faster throughput speed up how data is reloaded into memory, which improves the performance of the columnstore index.

## Limitations

Currently, columnstore indexes do have some limitations. SQL Server 2016 removed the majority of these with columnstore indexes, especially with Standard Edition. While nonclustered columnstore indexes cannot be created on a table with an existing clustered columnstore index, most of the previous limitations have been lifted.

*[The functionality and use of these indexes have changed considerably since its introduction in SQL Server 2012. See [this detailed list](#) for more differences since earlier versions.]*



Now, columnstore indexes are not always the right solution for all workloads. Columnstore indexes are geared toward warehousing and Decision Support System-type workloads. They're also designed for improving the performance of scan-type operations.

Conversely, seek operations, such as individual row lookups, could actually perform worse than a properly designed traditional rowstore index strategy. Data change operations, including inserts, updates, and deletes, can take substantially longer, as well. I have also seen some independent software vendor ISV packages using columnstore indexes when properly designed traditional rowstore indexes produce a more significant performance gain.

However, when used appropriately, columnstore indexes can be incredibly powerful. The addition of columnstore indexes in SQL Server 2016 SP1 Standard Edition can come in handy frequently for all developers.

### **Up Next**

In-memory functions in SQL Server are not mutually exclusive. In-Memory OLTP and columnstore indexes can be used together to provide dramatically improved performance for your varied workloads.



## CHAPTER 9:

## Combining In-Memory Technologies

Microsoft coined a new phrase with the release of SQL Server 2016. They called the act of performing analytics from an OLTP system “Real-Time Operational Analytics”—and the SQL Server platform holds true to this mantra with the 2019 release.

Historically, businesses have needed to separate their OLTP workloads from their reporting environments. This has required complex means to synchronize the data to maintain the division of OLTP and reporting from negatively impacting both. SQL Server 2019 allows users to perform reporting, warehousing, and ETL workloads off their OLTP environments by allowing the use of both In-Memory OLTP and columnstore indexes at the same time.

By leveraging these new features, the two distinct environments are simplified with removal of the secondary reporting system. Historically, the systems were set up differently and the two configurations could not function well together. Why? OLTP workloads are very random and accommodate highly concurrent workloads of small transactions. Warehousing-type workloads are larger in scale and usually perform aggregate functions on larger sets of data. The database instance configuration, database setup, and indexing strategy are typically different depending on the application and purpose, such as relational, reporting, or decision support. So in the past, setting up an OLTP environment to handle reporting tasks meant compromising performance for both workload types.

The benefits of combining these roles without compromising performance and complexity are significant. Space is saved on the SAN. The additional system no longer needs to be managed and secured. Licensing can be reduced.

But the largest improvement is that the reporting environment no longer lags behind the OLTP system. The traditional reporting platform data is ordinarily refreshed overnight during a database backup and restore operation from the production server. More complex environments could use a SQL Server feature, such as log shipping with snapshots or transactional replication. But this can be overcomplicated and produce its own challenges.

End users can now produce real-time reports without the delays that accompanied the data synchronization with features such as Always On Availability Groups and without incurring the space consumed by multiple copies of the data.

### Usage

Microsoft could not have made using these two features together any simpler. The table structure is now very similar to a traditional table, except that the index creation is performed in-line with table creation.

One difference in usage from traditional tables is that the columnstore index must include all the columns in the in-memory table. As an example, the `dbo.Order` table from Chapter 7 was modified for use with clustered columnstore indexes.



```

CREATE TABLE [dbo].[Order]
(
    [OrderID] [bigint] IDENTITY(1,1) NOT NULL
    PRIMARY KEY NONCLUSTERED,
    [PersonID] [bigint] NOT NULL,
    [OrderDT] [datetime] NOT NULL,
    [ShipDT] [datetime] NULL,
    [SalesRepID] [int] NOT NULL,
    [ProductCost] [numeric](18, 2) NULL,
    [ShippingCost] [numeric](10, 2) NULL,
    [TaxCost] [numeric](10, 2) NULL,
    [OrderCost] [numeric](18, 2) NULL,
    INDEX PK_Order_CCI CLUSTERED COLUMNSTORE
) WITH ( MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA )
GO

```

That's it!

## Wrap Up

In-memory functions in SQL Server are another fantastic tool in the SQL Server professional's toolbox. When used for the right use cases, these new SQL Server features are incredibly powerful. SQL Server In-Memory OLTP is a great means to improve the speed of certain operations by moving the data set entirely to memory with the lock-and-latch-free architecture. Columnstore indexes can help add warehousing and decision-support roles to existing OLTP servers in a way that won't negatively impact the OLTP server itself. And leveraging both at the same time will help you achieve what Microsoft calls "Real-Time Operational Analytics," all with little-to-no changes to your current codebase.

The only challenge to consider is ensuring your infrastructure is up to the task of handling the shift in the resource demand. Big Data Clusters, Azure Arc, and Cloud Block Store can all contribute to the effectiveness of your storage for modern SQL Server deployments, no matter where your databases run.

©2020 Pure Storage, the Pure P Logo, and the marks on the Pure Trademark List at <https://www.purestorage.com/legal/productenduserinfo.html> are trademarks of Pure Storage, Inc. Other names are trademarks of their respective owners. Use of Pure Storage Products and Programs are covered by End User Agreements, IP, and other terms, available at: <https://www.purestorage.com/legal/productenduserinfo.html> and <https://www.purestorage.com/patents>

The Pure Storage products and programs described in this documentation are distributed under a license agreement restricting the use, copying, distribution, and decompilation/reverse engineering of the products. No part of this documentation may be reproduced in any form by any means without prior written authorization from Pure Storage, Inc. and its licensors, if any. Pure Storage may make improvements and/or changes in the Pure Storage products and/or the programs described in this documentation at any time without notice.

THIS DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID. PURE STORAGE SHALL NOT BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS DOCUMENTATION. THE INFORMATION CONTAINED IN THIS DOCUMENTATION IS SUBJECT TO CHANGE WITHOUT NOTICE.

Pure Storage, Inc.  
650 Castro Street, #400  
Mountain View, CA 94041

[purestorage.com](https://www.purestorage.com)

800.379.PURE

