PURE STORAGE®
Uncomplicate Data Storage, Forever

TECHNICAL WHITE PAPER

Design Guide for SQL Server and VMware on Pure Storage

Design considerations for improving VMware virtualized Microsoft SQL Server performance.

Contents

Introduction 3

VMware and Databases 4

SQL Server and Storage 4

VMware and Storage 5

VMware Storage Types 7

Understanding NUMA 8

SQL Server Storage Engine 9

Reading Pages 11

Writing Pages 11

Storing Data in Tables 13

Understanding Indexes 13

Pure Storage and SQL Server: Better Together 17

FlashArray Reduces Data 17

Leveraging Snapshots for SQL Server 18

Consolidating VMware SQL Servers on Pure Storage 19

Enterprise Grade Storage in the Cloud 19

Summary 21

Introduction

Database workloads like Microsoft SQL Server are some of the biggest consumers of storage in the IT ecosystem. In the last decade, nearly all of those workloads have become virtualized, with VMware being the most popular hypervisor for those systems. With data being more important than ever for business success, performance, stability, and data protection are critical to those databases. For both on-premises and cloud workloads, your hypervisor, storage, and data must work together to meet your business needs.

Busy database workloads can put a lot of pressure on storage subsystems, in terms of bandwidth and IOPs. This is true whether it is an online transaction processing (OLTP) workload which has frequent insert and update activity, or data warehouse [also known as online analytical processing (OLAP)] which scans large volumes of data to perform business intelligence functions. Relational database management systems (RDBMS) are deeply instrumented software that allow database administrators to perform rich troubleshooting and analysis, and identify bottlenecks. This adds up to the need for high performance storage that helps turn those bottlenecks into breakthroughs.

Microsoft's SQL Server is the leading commercial RDBMS offering with over 25 years of history and support from a wide variety of independent software vendor (ISV) applications, as well as development frameworks. SQL Server is broadly used across industries in both on-premises and cloud configurations and supports both Windows and Linux operating systems. Database systems also require flexibility in space allocation, as data can grow very quickly. Additionally, support for backup technologies like snapshots can be very helpful to move data between environments.

While SQL Server can be deployed in several different configurations in both public and private clouds and in on-premises data centers, the most common on-premises implementation is running on VMware with remote storage. While hyperconverged infrastructure (HCI) has become more prominent in recent years, including VMware's vSAN offering, it presents two major problems for database workloads—the first being commercial database systems like SQL Server are licensed by the CPU cores assigned to the VM (or in some cases, the total number of CPU cores on a group of servers). When these cores have to be shared with the storage controller capacity of HCI, you are using CPU resources that could otherwise be used by your database.

In order to scale either memory, storage capacity, or performance in a hyperconverged infrastructure, you need to buy additional nodes, which comes with licensing and hardware costs. This means you can't scale your storage and compute needs independently. Database workloads can dramatically grow in terms of storage capacity—both in volume and performance demands. Having the flexibility to scale these two dimensions independently offers better total cost of ownership (TCO).

Pure Storage® can help you meet goals quickly and easily, while delivering world-class performance. Pure Storage FlashArray™ for block data and FlashBlade® for file and object data or backups span the full range of storage use cases. In the latest release of SQL Server 2022, you can take advantage of S3 connectivity for polybase or rapid restore on FlashBlade, further ensuring the best utilization of all hardware purchases for uncomplicated- unified management and flexible consumption for every workload. Ultimately the data services like SafeMode™ snapshots are what keep customer data safe by securing snapshots from being deleted by accident or malicious intent—ensuring your data is protected from a ransomware attack.

VMware and Databases

In the early days of virtualization, many database administrators (DBAs) were scared about virtualizing production database workloads. Beyond the normal fear, uncertainty, and doubt associated with change, there were a couple of legitimate concerns. As mentioned earlier, database servers are heavy users of IO and memory. In the early 2000s, it was still uncommon to see servers with more than 32GB of memory, which meant limited amounts of RAM to be shared with the database server.

Beyond sharing, the storage area networks (SANs) of that time were only beginning to be optimized for virtualization, and they mostly depended on traditional mechanical disks, which had high latency to begin with. And when combined with early virtual storage drivers, they did not offer the performance needed for mission-critical SQL Server workloads.

Over time, these factors were addressed. VMware improved their software, and storage evolved into all-flash arrays that were built with virtualization in mind—for both the overall system performance and the productivity that virtualizing hardware delivers, including maximizing the use of hardware, high availability, and the automation around deployment. This led many organizations to choose an "all virtualization" approach to their infrastructure.

Beyond performance, modern arrays like Pure Storage FlashArray™ also offer data reduction through the use of deduplication and compression, with up to 5:1 data reduction rate. Additionally, snapshots allow DBAs to refresh their development and test environment in a matter of seconds. The snapshots also allow for instant data protection during application upgrades, or as part of a more general data protection strategy.

SQL Server and Storage

Relational databases like SQL Server are somewhat unique among stateful workloads because they comply with ACID (atomicity, consistency, isolation, durability) properties. This makes them different from workloads like web servers, which can be easily redeployed from source code, or remote desktop, which typically do not require point-in-time recovery. The durability component of ACID is what both requires and allows a database to be restored to a specific point in time or to a specific transaction.

SQL Server databases are made up of a couple of file types. Data is stored in data files (which have two different file extensions: MDF, which is the primary data file, and NDF, which represents any additional data files) and transaction log files (with the file extension LDF). The transaction log file is critical to the database, as it allows for point in time recovery and provides SQL Server with the durability property of ACID. For example, when an application executes a transaction within a database, that transaction is not marked complete until it is recorded in the transaction log. The two largest memory consumers in SQL Server are the buffer pool, which stores recent data pages and the execution plan cache, which stores compiled execution plans. While database engines use memory to cache these, several types of data transaction log writes are synchronous IO—the transaction is not marked complete until the storage tells the database engine that the write is complete.

Beyond the transaction log, SQL Server makes heavy use of storage. Queries, even ones that insert and update tables, will typically perform a great deal of read activity. Beyond just queries, the database engine makes heavy use of TempDB, which is effectively a page file for the database engine, for various activities like version stores, and sort and join operations that spill outside of main memory.

These challenges mean that SQL Server workloads need storage that both performs well with extremely low latency, but also helps DBAs and application teams avoid downtime around the size of data activities.

VMware and Storage

It has been said that "greater than 80% of all problems in a virtualized environment are caused by the storage in some way, shape, or form."1 We would argue that the other 20% are caused by overprovisioning memory among VMware ESXi hosts. The VMware ESXi hypervisor allows you to logically abstract storage from the virtual machines themselves. A virtual machine will use one or more virtual disks (typically database VMs will use a number of virtual disks). VMware uses a logical construct called datastores to manage storage allocated to the cluster of hosts. Each virtual disk will appear as a SCSI drive, which connects to the operating system of the VM using a SCSI controller.

Unlike in physical computers, where you haven't needed to think about a storage controller in many years, the virtualization of devices requires emulation of physical devices. This means if you are using virtual disks (you will learn about raw mapped devices later in this section), IOs are flowing through software before they make their way into your storage array, which adds latency to the process of completing the IO. When you build a virtual machine on VMware, at least one virtual storage controller is created. A virtual storage controller is like a traditional physical hard disk controller, in terms of how the operating system interacts with it. However, choosing and configuring your virtual storage controllers is an important aspect of getting the best performance out of your VMs.

For more in depth and end-to-end recommendations for deploying Microsoft SQL Server in a combined VMware vSphere and Pure Storage FlashArray environment see the Microsoft SQL Server on VMware vSphere with FlashArray reference architecture.

VMware includes a number of virtual controllers, including BusLogic Parallel, LSI Logic Parallel, LSI Logic SAS, and VMware paravirtual SCSI (PVSCSI). In almost all cases for database workloads, you will create all of your data disks with the PVSCSI drivers, as they offer the highest levels of IO throughput with the lowest CPU overhead.

There is also an NVME controller type, which VMware's documentation mentions "perform best with an all-flash disk array." However, the NVME controller type provides no significant performance gains over PVSCSI, and more importantly for SQL Server users, it is not currently supported on Windows operating systems.

Beyond just using less CPU, the PVSCSI controller supports a higher queue depth (256 vs. 128 for LSI Logic SAS) which will let SQL Server issue more concurrent IOs through the operating system, down to the storage arrays. You can also increase the performance though your configuration of your PVSCSI adapters. Ideally, you will want to look at 4 PVSCSI adapters and adjust the ESXI hosts queue depth from 32 to between 64 and 128 for use with databases.

It is a best practice to configure multiple virtual disks and controllers for VMware virtual machines. The reason for this is that the Windows operating systems performs IO queuing by disk controller and not by the drive themselves. For Tier 1 workloads that have large IO demands, you should consider the following storage configuration.

1 Corey et al, "Architecting for Performance: Storage" Virtualizing SQL Server with VMware, 2013, pp. 105

PVSCSI Adapter Number	Virtual Disk Workload
0	Operating System System Databases
1	SQL Server Data files
2	SQL Server Transaction Log Files
3	TempDB Data and Log

TABLE 1 Configuration of SQL Server Virtual Disks

Table 1 shows a recommended configuration for an online transaction processing (OLTP) database server. OLTP systems include e-commerce, retail sales, and order entry. These workloads are characterized by heavy amounts of write activity, which means the transaction log file can become a bottleneck, as well as the data files as SQL Server reads and writes data based on user activity. By splitting the transaction logs and data files onto their own storage controllers, you are maximizing the throughput to each of those virtual disks. Likewise, on a busy server, TempDB will have heavy write activity, as SQL Server uses it for various internal functions. TempDB is unique in that it is minimally logged, so its transaction log is not heavily utilized in most systems.

There are a few more best practices around this design you should consider. Each disk should be mapped to a single controller for maximal IO performance. Obviously, in this case you would have two virtual disks on controller 0 because of the operating system. However, the operating system disk should not have significant amounts of IO. If you need to add additional disks to your VM, you should add them in a round-robin fashion around the controllers.

VMware uses datastores, which are analogous to file systems, as logical containers that abstract some of the details of physical storage devices. In newer versions of VMware, there are two main types of datastores: the virtual machine file system (VMFS), and virtual volumes (vVols), which are simply volumes on the storage array. If you are using virtual disk VMDK files, you should use a mapping of 1:1 for each VMDK file to each datastore, which will provide better performance. An additional best practice for databases using VMDK is to use “eager zeroed thick” provisioning. This simply means that when the virtual file is created, the VMDK file is fully provisioned and has zeros written to the end of the file. This means that when you are writing pages into your database, you do not have to wait on the underlying virtual disk file growth activity.

Thin Provisioning

Thin provisioning comes up a lot when working with virtualized resources and is usually bad news for database performance. Thin provisioning can happen at the storage array or in the hypervisor and allows storage teams to gain more capacity out of their storage. Thin provisioning means that when a virtual disk or LUN is provisioned, the only allocation is for the bytes currently in use. For example, if you created a 1TB thin-provisioned volume, but the actual disk on the VM was only using 5GB, the array will only use that 5GB. The downside to this for databases is that if you have a file that is frequently growing, there is a performance penalty as the file grows, and the storage is allocated within the hypervisor and/or the storage array.

Queue Depth

Queue depth is the number of IO requests that a given storage device or controller can handle at a given time. In the Windows operating system, Windows will hold up to 255 IOs by default per storage device (which in a VMware VM using virtual disks is a storage controller) before issuing them to the controller, and in turn the underlying storage. You can adjust the queue depth for a storage device, though in most cases the default value will be sufficient.

VMware Storage Types

You have several different options for how you can create disks in a VMware, and they can affect performance, flexibility, and availability. VMware VMs support the following disk types:

- Raw Device Mappings (RDM)
- Virtual Disks (VMDK)
- Virtual Volumes (vVol)

Raw device mapping (RDM) is straightforward: it is a directly connected storage device to your virtual machine. RDMs come in two flavors: physical and virtual mode. Physical bypasses the virtual stack, and the VM has direct access to the underlying disk. In virtual mode, the disk acts just like a normal VMDK file. In physical mode, the device offers direct control, which can reduce latency, since the path to the disk is slightly more direct in the case of physical RDMs. However, you will lose some of the flexibility of virtual devices.

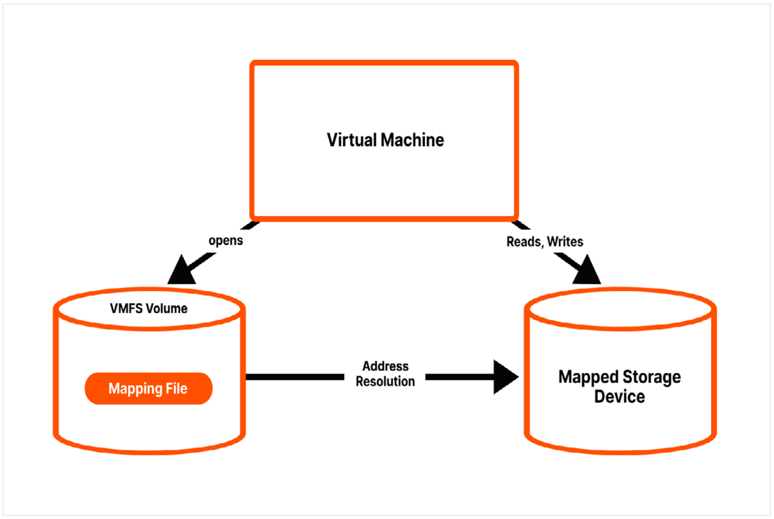


FIGURE 1 Virtualized files

In the past, RDMs were required for use in SQL Server failover cluster instances (FCI). However, newer versions of VMware allow for shared virtual disks. VMware hosts are limited to the number of RDMs (256), and migrations of VMs with RDMs is more challenging.

VMDK files are the most used storage type for VMware virtual machines, as they were created as a proprietary format by VMware (they were open sourced in 2011). VMDK files being fully virtualized, support technologies like snapshots, storage vMotion, and VMware high availability. These files are created using VMware tools like vCenter, and appear to the VM as a standalone file. The data from the operating system is stored within the VMDK file.

Virtual volumes—or as they are more commonly known, vVols—are simply volumes on the storage array, without having a virtual file. Data goes to the volume itself. However, to the operating system, the vVol just appears as a normal virtual disk. vVols have lower management overhead compared to both RDM and VMDKs and have some advantages around portability and copying files. You also have better control over your storage performance levels when using vVols.

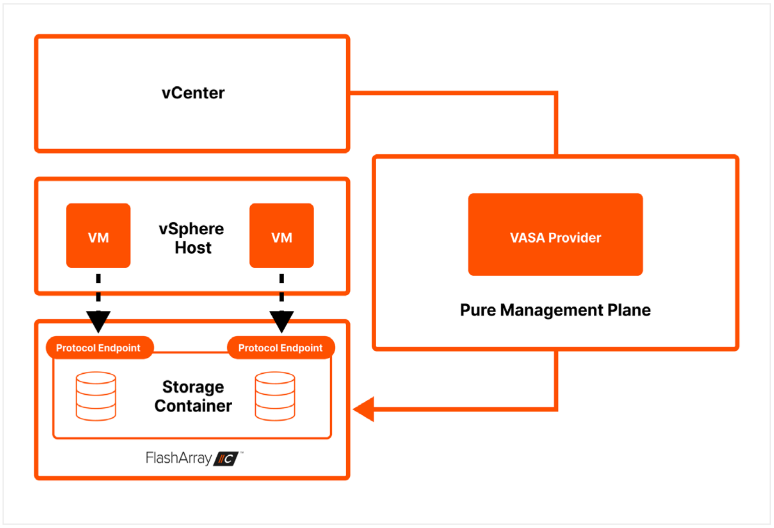


FIGURE 2 vVol Architecture

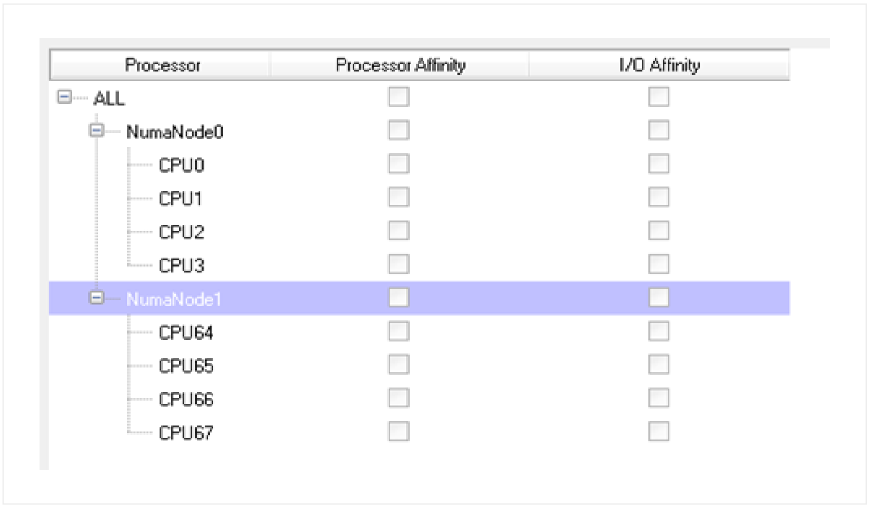
vVols are newer technology that take advantage of software-defined storage, allowing for easier automation of deployment and management tasks.

Beyond just choosing your storage for your VM, there are several best practices around VM configuration for SQL Server database servers. Properly configuring your memory and CPU settings will prevent you from incurring bottlenecks and can even help optimize your VMware host workload.

Understanding NUMA

Non-uniform memory access (NUMA) is a design pattern used in computers with multiple processor sockets. In a NUMA architecture, each processor has its own bank of memory, often referred to as node, which allows for faster access. CPUs operate faster than memory, which means the processors can often be waiting for data. By implementing NUMA, each processor has less memory to scan, meaning overall throughput is improved compared to a single pool of memory. The SQL Server database engine is NUMA-aware, meaning memory grants are allocated to the node associated with the processor thread. Beginning with SQL Server 2016, it will automatically divide larger NUMA nodes into smaller nodes, a feature called Automatic Soft-NUMA.

VMware also implements NUMA, and internally configures the appropriate NUMA configuration in terms of mapping vCPUs to memory nodes. In older versions of VMware, vNUMA had to be manually configured. A good rule of thumb is to match the size of your VMs to your underlying hardware topology. For example, if your physical servers are six sockets per core, it doesn't make sense to create VMs that are four and eight core, since you won't be optimizing your core counts. You can also verify your NUMA configuration in Server Properties from the server hive in SQL Server Management Studio (SSMS) as shown below.



Processor	Processor Affinity	I/O Affinity
ALL	☐	☐
NumaNode0	☐	☐
CPU0	☐	☐
CPU1	☐	☐
CPU2	☐	☐
CPU3	☐	☐
NumaNode1	☐	☐
CPU64	☐	☐
CPU65	☐	☐
CPU66	☐	☐
CPU67	☐	☐

FIGURE 3 3.NUMA configuration in SSMS

Beyond NUMA, there are some other considerations for CPUs and VMware. There is an option for being able to add CPUs to VMs called "CPU Hot Plug," which disables vNUMA and can add up to a 30% CPU cost to the VM. Additionally, the below settings on the host should be configured for optimal performance:

- Enable Turbo Boost
- Enable hyper-threading
- NUMA enabled
- Enable VT-x/VT-d
- Power Management set to OS Controlled
- Disable all processor C-states (C1E halt state)

Ideally, you would have dedicated VMware clusters for your database workloads, for both licensing and performance reasons. However, in most cases you will be running SQL Server VMs alongside other application VMs. These settings should not have any negative impact on those workloads.

SQL Server Storage Engine

SQL Server is a complex and well-instrumented piece of software. At the core of the SQL Server, RDBMS is the storage engine—the internal component responsible for reading and writing data that resides in tables. SQL Server's lowest level data storage mechanism is the 8-kilobyte page (unlike some other databases, SQL Server uses a fixed page size). Pages are numbered continuously within a given data file, starting with 0. Pages are grouped into extents of eight physically contiguous pages which are used to efficiently manage the page.

Each page in SQL Server starts with a 96-byte header to store system metadata about the page. This metadata includes the page number, the page type, the amount of free space in the page, and the allocation unit ID of the object which owns the page. SQL Server contains a number of different types of pages as shown as in the table below.

Page Type	Contents
Data	Rows containing table data, except for off-row data which is stored in text/image pages
Index	Index data
Text/image	Large object data types like nvarchar(max) and variable length columns when the data row exceeds 8KB (varchar, nvarchar)
Global allocation map, shared global allocation map	Metadata about how extents are allocated
Page free space (PFS)	Metadata about how pages are allocated and how much free space there is
Index allocation map (IAM)	Information about extents used by a table or index
Bulk changed map	Information about extents that were modified by bulk operations since the last BACKUP LOG command, which supports log backups in the BULK logged recovery model
Differential changed map	Information about extents that have changed since the last BACKUP DATABASE, which supports differential backups

TABLE 2 SQl page types and contents

Data in a single row in a table is written to the page in order, starting immediately after the header. There is a row offset table, which helps SQL Server locate rows on a page very quickly. By design a row is fully located within a single page, unless the table contains columns of the varchar, nvarchar, varbinary, or sql_variant data types. When a table contains one of these data types, and the size of a row is larger than 8,060 bytes, SQL Server moves one or more of these columns to "off-row storage". This off-row storage places pages in the ROW_OVERFLOW_DATA allocation unit, starting with the widest column. When a record is moved into an off-row page, a 24-byte pointer is stored in the original page, where the rest of the row is located. Having very large rows can affect query performance by making index scan operations less efficient since they have to read the pointer record and subsequently read the linked page.

SQL Server has two types of extents: uniform extents which only contain pages from a single object, and mixed extents which can be shared by up to eight objects. Starting with SQL Server 2016, the default is to use uniform extents for user databases and TempDB, except for the first eight pages of an IAM chain. Starting with SQL Server 2019, you can use the `sys.dm_db_page_info` dynamic management function which returns information about each page in the database.

Reading Pages

SQL Server uses the terms logical and physical IO reads. A logical read happens when the database engine requests a page from the buffer cache. If the page does not exist in the buffer cache, the physical read first copies the page from disk into the buffer. The buffer cache (which is also known as the buffer pool) is a fairly common construct across database engines, with the aim of reducing IO performed by the database engine.

The buffer pool is made up of 8KB pages, just like data or index pages. Buffer managers manage pages using a modified least recently used (LRU) algorithm. A given page will exist in memory until the buffer manager needs the space to read more data into the cache. Data is only written back from the buffer cache after it is modified. However, data can also be modified multiple times before being written back to disk. The size of the buffer cache is a function of the max server memory setting in SQL Server. Max memory will be fully allocated at database engine start if the lock pages in memory setting is configured; otherwise the size of the memory used by the database engine will grow gradually as data flows into the buffer cache and other areas of memory. SQL Server uses many caches, aside from the buffer cache, for operations like query optimization and other asynchronous IO operations.

The buffer manager performs all reads and writes to database pages; other operations like open/close, file shrinking and extension are performed by other components in the engine. All IOs performed by the buffer manager are performed asynchronously, which allows the calling worker thread to continue operating while the IO happens in the background.

When SQL Server has to perform multi-page IO, it uses a scatter-gather pattern, which allows data pages to be transferred out of non-contiguous areas of memory. This then allows SQL Server to quickly flush or fill the buffer pool with a single physical IO request. This unique adaptation is called read-ahead, which anticipates the data and index pages that are needed to service a query and bring the pages into the buffer cache before they are used by the query operator. When SQL Server reads data pages, the storage engine reads IAM pages to generate a sorted list of the physical locations of disk addresses to be read. This operation allows the storage engine to perform large sequential IOs that are efficient for reading large volumes of data. Index reads operate with a slightly different pattern, being read serially in the order of the index key, and then identifies any leaf pages that must be read, until the required reads to answer the query have been completed.

Writing Pages

An important characteristic of SQL Server's storage engine is that all data modifications happen in the buffer cache. When a page is changed in the buffer cache, it is not immediately written to disk—it is marked as a "dirty page". To provide data protection, for each logical write, a record is created in that database's transaction log that records the modification (that transaction log IO is also cached in a separate log called the log cache). Records in the transaction log cache must be written to disk before the database engine can move a dirty page out of the buffer pool and physically write that page into the data file. Beyond that protection, SQL Server uses a technique called write-ahead logging that prevents writing a dirty page into a data file before it is written to the database data file. Write-ahead logging (WAL) means writes to the transaction log can vary in size—the log buffer is 60 kilobytes, but there are several conditions that can trigger a transaction log cache flush:

1. A committed transaction that has an active log record in a log buffer. Upon this commit, all log records will be written to the transaction log file.
2. If the 60k limit for a log buffer was reached, the entire buffer is flushed to disk.
3. If new data pages must be written to data files on disk, but they have log pages related to them, those log records must be written to disk to support WAL.

While transaction log writes are aligned to the sector size of the underlying disk (on Pure Storage arrays this is 512 bytes), the size of the actual IO for log writes will vary based on what is in the log buffer at the time of the log flush.

Writing pages to data files is a different operation, and SQL Server has two main ways of writing data into data files from the buffer pool. The first is a checkpoint, which can be issued either manually via the checkpoint command, or by the database engine which occurs when the number of log records reaches the number that the database engine thinks it can process during the time specified in the recovery interval server configuration option. However, this is not a fixed timeframe, and it can be highly variable based on the workload of the database. When a checkpoint is issued, the engine can write an IO of anywhere from 64KB to 1MB. The other way SQL Server writes data into data files is through a process called a lazy writer, which prunes the buffer pool to make space for data pages by flushing dirty pages to disk, with a similar IO pattern to checkpoint. Finally, SQL Server can write pages directly to disk through the bulk insert process, which uses 256KB IOs.

The table below shows the range of IO sizes by SQL Server for all its storage engine related operations.

Operation	IO Block Size
Transaction log write	512 bytes-60 KB
Checkpoint/Lazy writer	8 KB-1MB
Read-ahead scans	128-512 KB
Bulk Inserts	256 KB
Backup/Restore	1 MB
Columnstore Read-Ahead	8 MB
File initialization	8 MB
In-memory OLTP Checkpoint	1 MB

TABLE 3 IO block sizes for SQL Server storage engine related operations.

As you can see, backups and restores use a 1MB block size for disk media. This means you need to understand how your underlying storage will perform against a wide array of IO sizes, not just 4–8KB.

Storing Data in Tables

Most tables in SQL Server are built with a clustered index, which stores and sorts the data in the table based on their key values. Each table can only ever have one clustered index because it is made up of the data rows—therefore, the clustered index is the table. A table without a clustered index is called a heap. In a heap, data pages are typically stored in the order they are inserted, but the SQL Server engine can move rows around to manage space efficiently. Heaps are typically only used for data warehouse staging tables or other large unordered insert operations, as the data is inserted faster than when using a clustered index.

Query performance tuning is such a broad topic that many books have been written about it. At its core, the basic idea is to reduce the number of logical and physical IOs a given query needs. There are many ways you can approach this as a database administrator (starting with the initial design and normalization of the database, which you do not always have control of), particularly for applications built by third party application vendors. Much like the earlier point about storage being responsible for 90% plus of virtual machine performance issues, improper index configuration is responsible for 90% of database performance issues. Improper indexes can include application and/or T-SQL code and data architecture patterns that preclude the appropriate use of those indexes. So how do you know which indexes to create? The answer is that it's both an art and a science. Below we'll take a deeper dive into this.

Understanding Indexes

You've learned about clustered indexes earlier, but SQL Server also includes several other types of indexes that each have their use cases. For example, there are XML indexes for XML data, hash indexes for memory optimized tables, and full-text indexes for the full-text search feature. However, the most commonly used index types are:

- **Clustered indexes**: Consists of the table data sorted by the order of the clustering key
- **Nonclustered indexes**: A secondary index to a clustered index (these can also be created on heaps) that can be created from one or more columns and include pointers to other columns as included columns.
- **Columnstore indexes**: Primarily used for analytic workloads like data warehouses, split indexes into individual columns which are then compressed to a high degree.

You have learned about clustered indexes, but choosing a clustering key for your table is an important part of architecting databases in SQL Server. Ideally, you would like your clustering key (which is frequently your primary key) to be narrow and ever-increasing. It is very common for a clustered index to be unique and to use an identity column that is an integer data type.

The DBA will typically not have control of how the clustered indexes are built, especially when working with third-party applications. However, they can—with or without software vendor approval—create nonclustered indexes to enhance query performance. The cost of a nonclustered index is two-fold: they consume space within the database by acting as a duplicate copy of one or more columns in a table, and they impact the performance of insert operations, as the index has to be written to, in addition to the write going to the table. However, the benefits in performance can be massive, as shown in the example below from the AdventureWorks database.

```
SELECT CarrierTrackingNumber, ProductID
FROM Sales.SalesOrderDetail
WHERE ProductID = 776
/*
Started executing query at Line 4
SQL Server parse and compile time:
  CPU time = 3 ms, elapsed time = 3 ms.
SQL Server parse and compile time:
  CPU time = 0 ms, elapsed time = 0 ms.
(228 rows affected)
Table '''''''''''‘SalesOrderDetail'''''''''''’. Scan count 1, logical reads 1142, physical reads 3, page server
reads 0, read-ahead reads 120, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0,
lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0.
 SQL Server Execution Times:
  CPU time = 4 ms, elapsed time = 7 ms.
Total execution time: 00:00:00.017
*/
CREATE NONCLUSTERED INDEX [IX_SalesOrderDetail_ProductID2] ON [Sales].[SalesOrderDetail]
(
  [ProductID] ASC,
  [CarrierTrackingNumber] ASC
)

SELECT CarrierTrackingNumber, ProductID
FROM Sales.SalesOrderDetail
WHERE ProductID = 776
/*
Started executing query at Line 32
SQL Server parse and compile time:
  CPU time = 2 ms, elapsed time = 2 ms.
SQL Server parse and compile time:
  CPU time = 0 ms, elapsed time = 0 ms.
(228 rows affected)
Table '''''''''''‘SalesOrderDetail'''''''''''’. Scan count 1, logical reads 5, physical reads 0, page server
reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob
page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0.
 SQL Server Execution Times:
  CPU time = 0 ms, elapsed time = 0 ms.
Total execution time: 00:00:00.008
*/
```

In this example, a new index was created to support the specific query being run—which included the CarrierTrackingNumber column in the index. The number of logical reads the query performed were reduced from 1142 to 5, and the runtime was halved. While both of these queries ran very quickly (they were run on an idle demo system), the IO reduction is very significant and would be even more impactful on a busy system. This doesn't mean you should go ahead and index every column on every table in your database, however.

For queries that are run frequently, this means any columns that are present in the WHERE clause of your frequently run queries should be indexed, as well as any columns that are used in JOIN clauses in those queries. SQL Server does have a missing index feature that will log any missing indexes in your table. You should use that with caution though, as it's not aware of other indexes on the table, and if you follow it blindly, you could end up with overlapping indexes and wasted space.

Columnstore indexes are inherently compressed, but both clustered and nonclustered indexes offer both row and page compression. You might ask yourself "Since my Pure storage array provides data reduction at the storage layer, why would I want to compress data within the database?" By using compression, you increase the amount of data stored on a given page (which is still 8KB). What this means is you need fewer pages to answer a query, both your buffer pool and coming from your storage array to answer a given query. To use the example from above, if we rebuild that index with page compression, you see the following change in statistics:

```
Table ''''''''''SalesOrderDetail''''''''''. Scan count 1, logical reads 4, physical reads 0, page server reads
0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page
server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0.
```

While going from five to four logical reads isn't the most impactful change (this data volume is very small), on larger tables and queries you will see more significant impact; remember pages are still stored as compressed in the buffer pool, so you see multiple benefits from compression. The trade off to compression is a small amount of CPU overhead, but nearly all SQL Server workloads will benefit from the use of compression. Compression is applied at the object level; tables, indexes, and partitions can be compressed independently of each other. Row compression simply converts fixed width data types like CHAR to variable length types like VARCHAR, which does not save a lot of space, but has nearly no overhead. Page compression does perform a level of data deduplication within a page, and offers up to 30% compression in most scenarios, with a little more CPU overhead.

Columnstore indexes also deserve another mention. They offer tremendous performance advantages in data warehouse workloads, specifically tables that are being scanned for all of their queries. In the following example, a query that calculates average sales price by day is being executed first against a columnstore index and then against a similar table with page compression.

```
SELECT orderdatekey
  ,avg(salesAmount)
FROM FactResellerSalesXL_PageCompressed
GROUP BY OrderDateKey
ORDER BY avg(SalesAmount) DESC
/*
 (3651 rows affected)
Table 'FactResellerSalesXL_CCI'. Scan count 4, logical reads 0, physical reads 0, page server reads 0,
read-ahead reads 0, page server read-ahead reads 0, lob logical reads 4455, lob physical reads 0, lob page
server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0.
Table 'FactResellerSalesXL_CCI'. Segment reads 12, segment skipped 0.
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads
0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob
read-ahead reads 0, lob page server read-ahead reads 0.
 SQL Server Execution Times:
  CPU time = 377 ms, elapsed time = 142 ms.
*/
SELECT orderdatekey
  ,avg(salesAmount)
FROM FactResellerSalesXL_PageCompressed
GROUP BY OrderDateKey
ORDER BY avg(SalesAmount) DESC
/*
(3651 rows affected)
Table 'FactResellerSalesXL_PageCompressed'. Scan count 5, logical reads 88408, physical reads 0, page
server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads
0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0.
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads
0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob
read-ahead reads 0, lob page server read-ahead reads 0.
 SQL Server Execution Times:
  CPU time = 3849 ms, elapsed time = 1007 ms.
*/
```

The columnstore query runs in 142 milliseconds, while the page compressed query takes just over a second. There are a couple of reasons why: the columnstore query will **only** scan the columns referenced in the query, which represents a large IO reduction. The scanned columns are highly compressed, much more so than with page compression. Prior to SQL Server 2019, there was a third advantage that columnstore queries could use batch execution mode which processed data in chunks of a thousand rows; however, starting with SQL Server 2019, row store tables (anything that isn't a columnstore) also support batch mode. In fact, the above query execution ran in batch mode. Columnstores are not good for operational tables that often insert and update singleton records of data, as insert and update operations are expensive.

To understand SQL Server performance tuning, you need to begin to understand query execution plans. While these are beyond the scope of this paper, understanding the various operators and their order helps you identify the proper indexes, a nd anti-patterns in your T-SQL code.

Pure Storage and SQL Server: Better Together

One of the other challenges many tier 1 database systems face is the concept of data gravity. On traditional storage arrays once your data grows beyond a couple of terabytes, data has "gravity"—meaning that it can be costly and time-consuming to move it between servers or even within the same server. Pure Storage helps you reduce the issues around data gravity both by offering best in class performance to help you overcome data volume challenges, and in many cases avoid "size of data" (size of data operations, are operations whose duration is directly tied to the volume of data involved) operations by taking advantage of the features of your Pure Storage array.

Database workloads, like SQL Server on VMware, are some of the largest workloads in both volume and IO demands. DBA staff are busy and face a lot of challenges around pleasing development teams, tuning performance, and keeping the lights on. Pure Storage FlashArrays take care of all those important needs for your mission critical SQL Server workloads and provide valued added features that let you get more value and productivity from your storage.

FlashArray Reduces Data

Earlier, you learned about how data compression and columnstore indexes in SQL Server allow you to reclaim storage and improve buffer pool utilization for your SQL Server instance. FlashArray's compression technology takes that a step further by performing in-line deduplication on the storage array. This allows you to reduce your overall physical storage costs and decrease your data center footprint by increasing the overall efficiency of your storage system.

Pure Storage uses Purity Reduce technology which combines five different data-reduction techniques to save space in its all-flash storage arrays. Purity always has the goal of performance over data reduction. For example, the first phase of data reduction takes place as data is initially written to the storage array inline. This means as a write as acknowledged back to the calling operating system. For SQL Server, this means a transaction that is marked committed, will be partially compressed before it reaches disk. The Purity Reduce process uses five different data reduction techniques to save space:

- **Pattern removal:** Purity Reduce finds and eliminates repeating binary patterns, which reduces the data to be processed by the dedupe scanner and compression engine. This helps the performance of the overall process.
- **512B aligned variable dedupe:** This is a high performance inline (or synchronous) process that ensures that only unique blocks of data are written to the storage. At the completion of this step the write is acknowledged back to the calling OS as completed.
- **Inline compression:** Purity Reduce uses an append-only write layout and variable addressing to remove wasted space consumed by fixed block architectures.
- **Deep reduction:** This process takes place after the data has been fully persisted to the array. You can think of it as an asynchronous process, that consumes more resources (without impacting the primary workload on the array) to further increase space savings
- **Copy reduction:** Copies (for example snapshots and clones) on the array only use metadata and provide instant pre-deduplicated copies of data.

Database workloads, whether they be OLAP or OLTP, can expect to see data reduction in the range of 2-4:1 in most cases.

Leveraging Snapshots for SQL Server

As the size of your databases increase, it can be challenging to meet backup and recovery SLAs, or even just complete backup and restore operations within an acceptable window. Storage snapshots can dramatically improve these backup and restore timeframes, but poor implementations of snapshots have made DBAs wary of their use for product work. A snapshot is a point-in-time image of a volume or protection group, which is a collection of volumes. You can combine snapshots and SQL Server to enable specific functionality to protect the data in your environment. Pure Storage FlashArray supports both crash consistent and volume shadow copy snapshots, which be used in conjunction with SQL Server:

- **Crash consistent snapshots:** These snapshots require vVols to be in place, and use redirect-on-write technology rather than copy-on-write. You can snapshot a source volume with zero performance impact, but they do not support point-in-time recovery
- **Volume shadow copy snapshots:** These snapshots integrate into the volume shadow copy (VSS) framework in Windows. The IO on the array is briefly quiesced and still uses the redirect-on-write technology. However, volume copies support point-in-time recovery and integrates with third party backup solutions like Veeam and Commvault.

While these snapshots may not completely replace a traditional backup strategy, they can complement backups and replace scenarios where you might need to do a traditional restore operation. The most common use cases for snapshots with SQL Server on FlashArray are as follows:

- **Refresh dev/test environments instantly:** A frequent task performed by DBAs is restoring backups to lower environments. Even with good automation scripting, this process can be very time consuming, and leave developers waiting to get access to newer data. Volume snapshots can instantly make production data available to test and development environments. Additionally, the data in these snapshots is data-reduced, so you don't need to use the full capacity for each copy on your storage array.
- **Offloading database maintenance activity:** One of the most important tasks a DBA does is to ensure that databases are consistent using the CHECKDB command. CHECKDB is a very resource-intensive process and can impact production activity. In 24×7 environments like hospitals and manufacturing facilities any performance issues can harm user activity. With snapshots, you can clone your production database to another instance and run CHECKDB there. Unless you find corruption upon running CHECKDB off that lower environment, you would never have to run CHECKDB against the production database.
- **Instant database and ransomware protection:** This is particularly useful for deployments of application code. An application deployment might only affect a handful of rows in a couple of tables, but if it has problems, the only way to resolve them is to restore the full database. With snapshots, you can clone a previous snap of the volumes that support your database and immediately get access to the original data. Similarly, these snapshots can protect against ransomware attacks.

For more information on using volume snapshots with SQL Server see the Microsoft SQL Server on VMware vSphere with FlashArray reference architecture.

Consolidating VMware SQL Servers on Pure Storage

One of the challenges faced with SQL Server workloads historically, has been sprawl: Many applications require using SQL Server as a database, and in many cases those SQL Servers get built and deployed without the intervention of the IT organization. This can lead to data protection and performance issues, as well as (likely most importantly) licensing concerns. SQL Server is typically licensed by the individual VM, by the number of CPU cores assigned to each of those VMs. However, there is a licensing option called "unlimited virtualization" that allows firms to license all the physical cores on a given group of VMware hosts and in turn run as many SQL Server VMs as they would like on those hosts.

This consolidation approach requires some careful planning, since you need to ensure that you have the resources to meet the peak resource demands of those consolidated workloads. While consolidating allows you to drive higher levels of infrastructure efficiency through improved resource utilization, the underlying storage needs to be able to meet the performance demands of bursting activity—like a running database query—that could adversely impact the other workloads on the array. Your Pure Storage array can handle these workloads while maintaining the latency and throughput requirements for each of the applications sharing the array.

The Pure Storage concept of Evergreen Storage™ ensures that over time you always have the latest storage controllers, while maintaining the same underlying storage. However, newer storage arrays take advantage of advances in storage protocols like PCIe and NVMe which allow newer disks to be more space efficient. Pure Storage offers a capacity consolidation program to consolidate older shelves of storage into newer, denser storage.

Enterprise Grade Storage in the Cloud

In recent years, many organizations have moved into public cloud providers like Microsoft Azure or Amazon AWS. Some companies have already transitioned some of their workloads to the cloud, but they face many challenges. Moving terabytes or petabytes of data between an on-premises data center and the cloud can take days or weeks, even with a good network connection. Once you have your data in the cloud, do you still have the features to manage your data that you had in your on-premises environment?

Understanding storage performance in the public cloud is also a key part of successful cloud migrations. Whether you are looking to migrate your dev/test workloads into the cloud, leverage the public cloud as a disaster recovery site, or migrate your entire data center into the cloud, you can leverage Pure Cloud Block Store™ to deliver an optimized cloud experience around data management.

Managing storage in the cloud is one of the biggest challenges to moving into the cloud; while features abound, they frequently overlap with other features, and it means your engineering staff needs to learn an entirely new control plane and try to understand those features. Additionally, you can fall into architectural traps leading you to have higher TCO than you need to have.

Understanding Pure Cloud Block Store

Pure Storage brings the primary storage and enterprise data services into Azure. The high-level architecture in Pure Cloud Block Store is made of Azure virtual machines and controllers and uses Azure managed disks for its NVRAM and persistent storage.

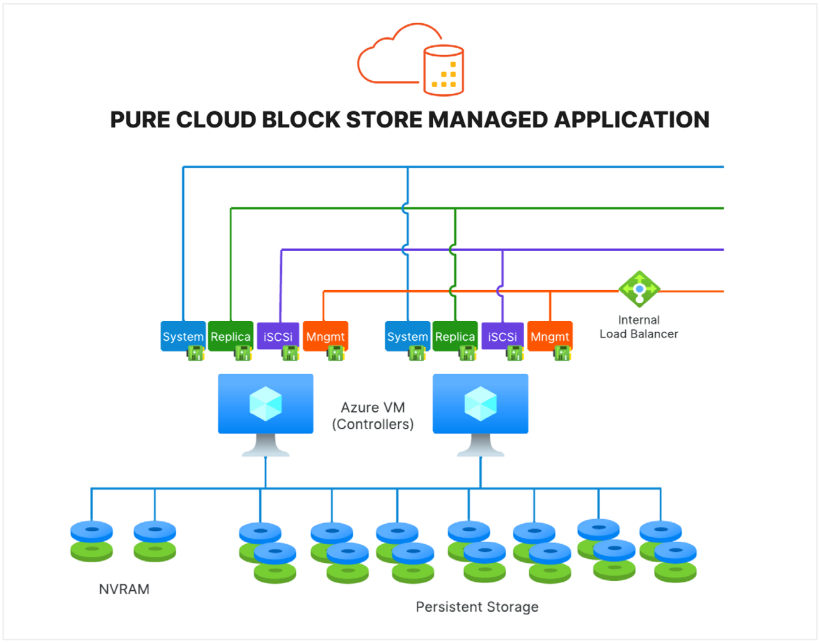
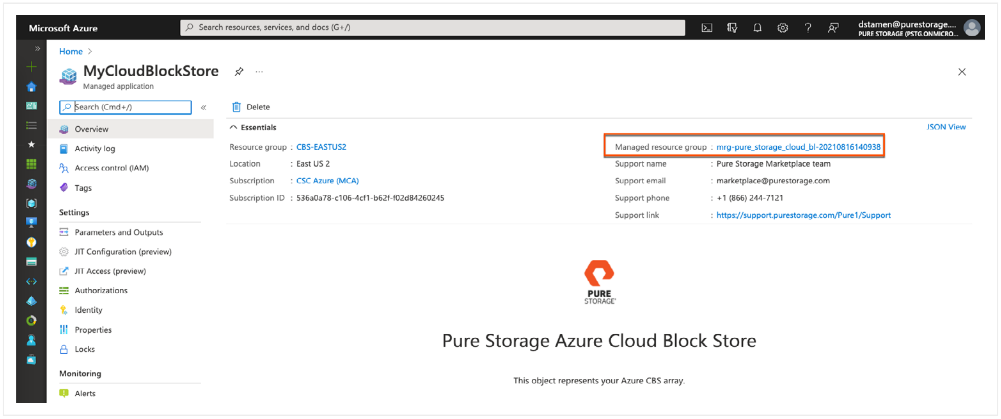


FIGURE 4 Pure Cloud Block Store architecture

The Pure Cloud Block Store will appear as a managed application in the Azure portal as shown below.



Resource group : CBS-EASTUS2
Location : East US 2
Subscription : CSC Azure (MCA)
Subscription ID : 536a0a78-c106-4cf1-b62f-f02d84260245

Managed resource group : mrg-pure_storage_cloud_bl-20210816140938
Support name : Pure Storage Marketplace team
Support email : marketplace@purestorage.com
Support phone : +1 (866) 244-7121
Support link : https://support.purestorage.com/Pure1/Support

Pure Storage Azure Cloud Block Store

This object represents your Azure CBS array.

FIGURE 5 Pure Cloud Block Store shows as a managed application in the Azure portal

Once your Pure Cloud Block Store is created, you can assign storage to your Azure VMs by logging into your dashboard to create volumes and assign them to hosts. This allows you to have the same experience managing cloud storage as you would in your own data center.

There are a number of use cases for Cloud Block Store beyond a common a management plane, such as:

- Easier data migrations with seamless data mobility
- Hybrid cloud business resilience for SQL Server

Data migrations are challenging when you are talking about the size of data operations like moving large amounts of data to the cloud, even with the best networking available, real-time migrations can be challenging, due to the sheer volume of data. If you are using vVols in your on-premises environment, you can leverage FlashArray replication to clone your volumes to your cloud block store as shown in the following architectural diagram.

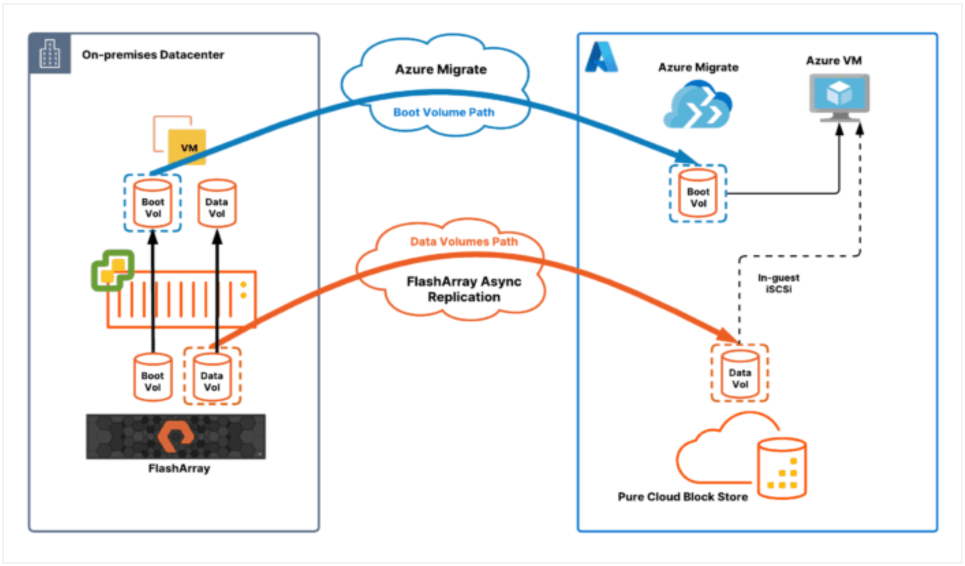


FIGURE 6 FlashArray replication to Pure Cloud Block Store with vVols

This leverages the data reduction strategies of Purity to ease migration of your data disks. You will still need to use Azure Migrate to move the boot volume and convert it into an virtual machine OS disk. Typically the bulk of the data volume is in your data drives, so the data reduction of those volumes will reduce the time and effort for your cloud migration dramatically.

Many organizations still rely on SQL Server's Failover Cluster Instance technology to provide high availability and disaster recovery. Failover cluster instances depend on shared storage volumes for high availability, and array-based replication for disaster recovery. With Pure Cloud Block Store you can use asynchronous replication from your data center into Azure to provide hybrid data resilience.

Summary

Whether you are operating in the cloud or on-premises, your database workloads like Microsoft SQL Server place some of the highest demands on your storage arrays, both in terms of volume and storage performance. Meeting those performance demands is vital to business-critical applications and your customer's experiences when using your apps and websites. Overcoming challenges associated with data gravity migrating storage arrays or executing server migrations requires advanced storage technology to avoid long downtime periods. Pure Storage FlashArray and Pure Cloud Block Store offer both best-in-class performance and advanced features to meet the needs of your largest workloads and offer the most efficient storage for your environment.